WebSphere® 멀티플랫폼용

IBM WebSphere Process Server

버전 7.0.0

모듈 개발 및 전개

IBM®

WebSphere® 멀티플랫폼용

IBM WebSphere Process Server

버전 7.0.0

모듈 개발 및 전개

IBM®

2010년 4월

이 개정판은 새 개정판에 별도로 명시하지 않는 한, 멀티플랫폼용 WebSphere Process Server의 버전 7, 릴리스 0, 수정 0(제품 번호 5724-L01) 및 모든 후속 릴리스와 수정에 적용됩니다.

이 문서에 대한 사용자 의견을 보내시려면 ibmkspoe@kr.ibm.com으로 전자 우편 메시지를 보내십시오. 사용자의 의견을 기다리고 있습니다.

IBM에 정보를 보내는 경우, IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

목차

표

제 1 부 응용프로그램 개발

제 1 장 비즈니스 프로세스 관리 솔루션 개발

이 절에서는 BPM(Business Process Management) 프로그래밍 모델의 기본사항에 대해 설명합니다. SCA(Service Component Architecture)에 대해 소개하며 비즈니스 통합과 관련된 패턴을 설명합니다.

BPM은 회사가 비즈니스 프로세스를 식별, 통합 및 최적화시킬 수 있는 부문입니다. BPM의 목적은 생산성을 개선시키고 조직의 효율성을 개선시키는 것입니다. BPM에 대한 관심은 회사의 병합 및 합병과 다른 정보 자산 라이브러리가 확장됨에 따라 더욱 증가하고 있습니다. 이러한 자산은 일관성 및 통합 기능이 적으므로 "정보의 섬(islands of information: 지리자료가 수치적으로 변환, 저장, 분석, 디스플레이 되는 독립시스템)" 문제가 발생합니다.

BPM은 SOA(Service Oriented Architecture)와 관련성이 매우 높습니다. 회사의 특성과 통합 필요 범위에 따라, BPM은 IT 부서의 다양한 요구사항까지 포함합니다. 일부 프로젝트는 단지 몇 개의 측면만을 처리하는 반면, 기타 규모가 큰 프로젝트는 이들 요구사항 중 많은 것을 포함합니다. 다음은 BPM 프로젝트의 가장 일반적인 측면에 대한 설명입니다.

- **응용프로그램 통합**은 일반적인 요구사항입니다. 응용프로그램 통합 프로젝트의 복잡도는 적은 수의 응용프로그램이 정보를 공유할 수 있음을 확인할 필요가 있는 간단한 경우에서부터 트랜잭션 및 데이터 교환이 다중 백엔드 응용프로그램에서 동시에 반영될 필요가 있는 더 복잡한 상황까지 다양합니다. 복합 응용프로그램 통합은 변환 및 맵핑은 물론 복합 작업 단위 관리를 필요로 합니다.
- **프로세스 자동화**는 개인 또는 조직에서 조직적으로 수행하는 활동으로 인해 필연적으로 수행해야 할 다른 활동이 발생한다는 점을 확인하는 또 다른 중요한 측면입니다. 이것은 전반적인 비즈니스 프로세스를 성공적으로 완료함을 보장합니다. 예를 들어, 회사에서 직원을 고용할 때, 급여 정보가 갱신되어야 하며, 보안 부서에서의 적절한 조치가 필요하며, 직원에게 필요한 도구를 제공할 필요가 있다는 것 등입니다. 프로세스에서 일부 조치를 수행하여 직원이 직접 입력하고 상호작용하는 것을 캡처할 수 있지만, 기타 조치를 수행하여 백엔드 시스템에서 스크립트를 호출하고 해당 환경에서 다른 서비스를 호출할 수 있습니다.
- **연결성**은 회사 및 비즈니스 파트너 측면 양쪽에서 추상적이나 중요합니다. 연결성은 조직이나 회사 간의 정보 플로우와 분산된 IT 서비스에 액세스하는 기능 둘 다를 참조합니다.

비즈니스 통합 구현의 기술적인 인증 확인 몇 가지는 다음과 같이 요약할 수 있습니다.

- 다른 데이터 형식을 처리하므로 효율적인 데이터 변환을 수행할 수 없음

- 다른 기술을 사용하여 개발해 온 IT 서비스에 액세스하기 위해 다양한 프로토콜 및 메커니즘 처리
- 지리적으로 분산되거나 다른 조직에서 제공할 수 있는 여러 IT 서비스의 조정
- 사용 가능한(거버넌스(Governance)) 서비스를 분류하고 관리하기 위한 규칙 및 메커니즘 제공

이상과 같이, BPM은 SOA에 공통인 많은 요소와 테마를 포괄합니다. IBM®의 BPM에 대한 비전은 SOA에서 찾을 수 있는 동일한 여러 기본적인 개념에 기초하고 있습니다. 이 비전에 대한 즉각적인 결과 중 하나는 BPM 솔루션이 해당 실현을 위해 다양한 제품을 필요로 한다는 것입니다. IBM에서는 모든 다양한 단계와 조작적인 측면을 지원하는 도구 및 런타임 플랫폼의 포트폴리오를 제공합니다.

BPM에 대한 IBM의 비전을 달리 표현하면, SOA IT 인프라에서 실행하는 응용프로그램을 사용하여 회사가 비즈니스 프로세스를 정의, 작성, 병합, 결합 및 능률화시킬 수 있습니다. BPM 작업은 사실 역할 기반입니다. 매크로 레벨에서, 프로세스 응용프로그램을 모델링하고, 개발하고, 거버넌스(Governance)하고, 관리하며 모니터링하는 것과 관련됩니다. 적절한 도구 및 프로시저를 사용하여 엔터프라이즈 내부 및 외부 모두 사용자 및 이기종 시스템과 관련하여 비즈니스 프로세스를 자동화할 수 있게 합니다. BPM의 핵심 측면 중 하나는 변경을 처리할 만큼 충분히 효율적이고 확장 가능하며 신뢰성 있고 유연하도록 비즈니스 조작을 최적화하는 기능입니다.

BPM에는 개발 도구, 런타임 서버, 모니터링 도구, 서비스 저장소, 툴킷 및 프로세스 템플리트가 필요합니다. BPM에 대한 너무 많은 측면이 있기 때문에 솔루션을 개발하기 위해서는 하나 이상의 개발 도구를 사용해야 합니다. 이들 도구는 통합 개발자가 복합 비즈니스 솔루션을 어셈블할 수 있게 합니다. 서버는 복합 응용프로그램을 실행하는 고성능 비즈니스 엔진 또는 서비스 컨테이너입니다. 관리는 조직(모니터링 도구가 역할을 수행하는 위치)에서 누가 무엇을 수행하고 있는가를 파악하는 것입니다. 엔터프라이즈가 이들 비즈니스 프로세스나 서비스를 작성함에 따라, 이들 서비스의 거버넌스(Governance), 분류 및 저장영역이 중요해집니다. 해당 기능은 서비스 저장소에서 제공합니다. 기존 시스템에 대한 커넥터 또는 어댑터와 같이 특수화된 솔루션 부분을 작성하는 특정 툴킷이 필요한 경우가 있습니다.

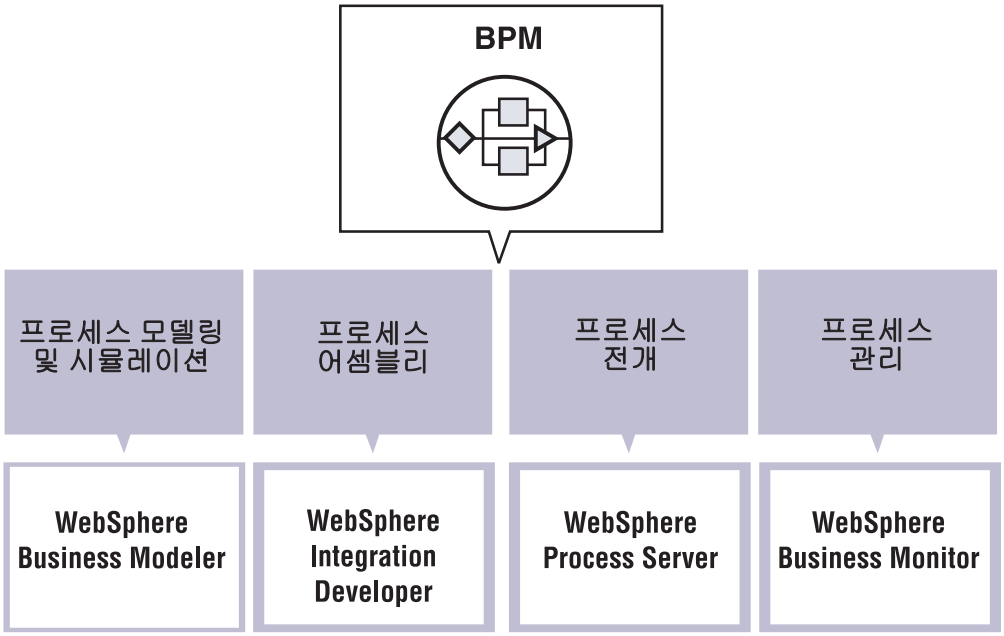


그림 1. IBM 도구는 프로세스를 모델, 어셈블, 전개 및 관리할 수 있도록 하는 전체 BPM 라이프사이클을 포함합니다.

BPM은 단일 제품을 기반으로 하지 않습니다. 조직 내에서 그리고 조직 전체에서 거의 모든 사용자와 비즈니스 측면 전체와 관련됩니다. BPM은 SOA 참조 아키텍처에서 여러 서비스 및 요소를 포함합니다.

프로그래밍 예제와 함께 이들 개념에 대한 자세한 사항은 다음을 참조하십시오.

- *WebSphere® Business Integration Primer: Process Server, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1: Development*, IBM Redbooks®, SG24-7608-00, June 2008.
- *IBM Business Process Management Reviewer's Guide*, IBM Redpapers, REDP-4433-01, 2009년 4월.

비즈니스 통합 아키텍처 및 패턴

일반 비즈니스 관리 프로젝트는 여러 가지 다른 IT 자산 조정과 관련되며, 잠재적으로 다른 플랫폼에서 실행하고, 다른 기술을 사용하여 다른 시간에 개발되는 것과 관련됩니다. 다양한 컴포넌트 세트를 이용하여 정보를 손쉽게 조작하고 교환할 수 있는 것이 주요 기술적인 관건입니다. 이러한 문제는 비즈니스 통합 솔루션을 개발하기 위해 사용되는 프로그래밍 모델을 통해 가장 잘 해결할 수 있습니다.

이 절에서는 SCA(Service Component Architecture)에 대해 소개하며 비즈니스 통합과 관련된 패턴을 설명합니다. 패턴은 우리 주변에 매우 다양하게 분포되어 있습니다.

바느질 패턴, 아이들이 생각하고 배우는 패턴, 주택 건설 패턴, 목각 패턴, 비행 패턴, 바람 패턴, 의학에서의 실습 패턴, 고객 구매 패턴, 워크플로우 패턴, 전산학에서의 설계 패턴 및 다양한 여러 가지 패턴이 있습니다.

패턴을 활용하면 솔루션 디자이너 및 개발자에게 유용하다는 사실이 증명되었습니다. 그러므로, 이제 비즈니스 통합 패턴 및 엔터프라이즈 통합 패턴을 갖는 것이 당연한 일입니다. 요청과 응답 라우팅, 채널 패턴(예: 공개/등록) 등에 대한 패턴을 포함하여 비즈니스 통합에 적용 가능한 광범위한 배열 패턴이 있습니다. 추상 패턴은 문제점의 특정 카테고리를 해석하는 템플리트를 제공하는 반면, 구체적 패턴은 특정 솔루션을 구현하는 방법에 대한 특정 표시를 더 많이 제공합니다. 이 절에서는 WebSphere 비즈니스 통합을 위한 IBM 소프트웨어 전략의 프로그래밍 모델 토대가 되며, 데이터와 서비스를 처리하는 패턴에 중점을 두어 설명하겠습니다.

비즈니스 통합 시나리오

엔터프라이즈에는 관련 비즈니스를 실행하기 위해 사용하는 여러 소프트웨어 시스템이 있습니다. 그리고, 이들 비즈니스 컴포넌트를 통합하는 고유한 방식을 가지고 있습니다.

두 개의 가장 일반적인 비즈니스 통합 시나리오는 다음과 같습니다.

- **통합 브로커**: 이 시나리오에서 비즈니스 통합 솔루션은 다양한 백엔드 응용프로그램 사이에 위치한 매개체로서 역할을 합니다. 예를 들어, 고객이 온라인 주문 관리 응용프로그램을 사용하여 주문할 때 트랜잭션이 CRM(Customer Relationship Management) 백엔드에서 관련 정보를 갱신하는지 확인할 필요가 있습니다. 이 시나리오에서, 통합 솔루션은 주문 관리 응용프로그램에서 필요한 정보를 캡처 및 변환하고 CRM 응용프로그램에서 적절한 서비스를 호출할 수 있어야 합니다.
- **프로세스 자동화**: 이 시나리오에서, 통합 솔루션은 여러 IT 서비스를 연결하는 역할을 하며 기타 역할은 수행하지 않습니다. 예를 들어, 회사에서 사원을 채용할 때, 다음과 같은 일련의 조치를 수행해야 합니다.
  - 사원 정보가 급여 시스템에 추가됩니다.
  - 회사 시설에 대한 물리적 액세스 권한을 사원에게 부여해야 하며 명찰을 제공해야 합니다.
  - 회사에서는 물리적 자원 세트를 사원(오피스 영역, 컴퓨터 등)에게 할당해야 합니다.
  - IT 부서에서는 사원의 사용자 프로파일을 작성하고 일련의 응용프로그램에 대한 액세스 권한을 부여해야 합니다.

    이러한 프로세스를 자동화하는 작업도 비즈니스 통합 시나리오에서의 일반적인 유스 케이스입니다. 이 시나리오에서 솔루션은 급여 시스템에 사원을 추가함으로써 트리거되는 자동화된 플로우를 구현합니다. 차후에 조치를 취하는 데 책임이 있는 직원의 작업 항목을 작성하거나 적절한 서비스를 호출하여 플로우에서 다른 단계를 트리거합니다.

두 가지 시나리오 모두에서 통합 솔루션은 다음 역할을 수행해야 합니다.

1. 정보 및 여러 데이터 형식의 다른 소스로 작업하고 다른 형식 간에 정보를 변환할 수 있게 합니다.
2. 잠재적으로 여러 호출 메커니즘 및 프로토콜을 사용하여 다양한 서비스를 호출할 수 있게 합니다.

역할, 제품 및 기술 인증 확인

성공적인 비즈니스 통합 프로젝트는 특수화된 개발 역할, 프로그래밍 기술 및 도구 세트를 적절히 결합하여 사용하는지 여부에 영향을 받습니다.

비즈니스 통합 프로젝트에는 다음과 같은 몇 가지 기본적인 구성요소가 필요합니다.

- 일반적으로 개발된 각 컴포넌트의 품질을 개선시키는 전문화를 증진시키기 위해 개발 조직에서의 명확한 역할 분리
- 비즈니스 정보를 공통 논리 모델에 표시할 수 있게 하는 공통 비즈니스 오브젝트(BO)
- 인터페이스를 구현과는 완전히 분리시키며, 완전히 구현과 독립적인 일반 서비스 호출 메커니즘을 지원하며 인터페이스 처리에만 관련된 프로그래밍 모델
- 개발 역할을 지원하며 분리를 보존하는 통합된 도구 및 제품 세트

다음 절에서는 이러한 각 구성요소를 세분화하여 설명합니다.

명확한 역할 분리

비즈니스 통합 프로젝트에는 다음의 네 가지 역할이 필요하며 이 역할들은 상호 협조적이면서도 명확히 구분된 역할입니다.

- **비즈니스 분석자**: 비즈니스 분석자는 프로세스의 비즈니스 측면을 캡처하고 프로세스 자체를 적절하게 나타내는 프로세스 모델을 작성하는 일과 관련이 있습니다. 이들은 프로세스의 재정적인 성과를 최적화하는 일에 중점을 둡니다. 비즈니스 분석자는 프로세스를 구현하는 기술적인 측면과는 관련이 없습니다.
- **컴포넌트 개발자**: 컴포넌트 개발자는 각 서비스와 컴포넌트를 구현하는 일에 관여합니다. 이들은 구현에 사용되는 특정 기술에 중점을 둡니다. 이 역할을 수행하려면 프로그래밍 관련 배경 지식이 매우 요구됩니다.
- **통합 전문가**: 상대적으로 새로운 이 역할은 기본 컴포넌트 세트를 더 큰 비즈니스 통합 솔루션으로 어셈블하는 데 관여하는 역할입니다. 통합 개발자는 재사용하고 함께 연결하는 각 컴포넌트와 서비스의 기술적인 세부사항을 알 필요는 없습니다. 이론상으로, 통합 개발자는 어셈블하는 서비스의 인터페이스를 이해하는 일에만 관련이 있습니다. 통합 개발자는 어셈블리 프로세스의 통합 도구를 사용해야 합니다.
- **솔루션 전개자**: 솔루션 개발자와 관리자는 일반 사용자에게 사용 가능한 비즈니스 통합 솔루션과 관련이 있습니다. 이론상으로, 솔루션 전개자는 솔루션이 제대로 역할을 수행하는 데 필요한 물리적 자원(데이터베이스, 큐 관리자 등)과 솔루션을 결합하는

일에 관여하며 솔루션의 핵심 내용까지 완전히 이해하는 일과는 관련이 없습니다. 솔루션 전개자는 서비스 품질(QoS)에 중점을 둡니다.

공통 비즈니스 오브젝트 모델

설명한 바와 같이, 비즈니스 통합 프로젝트의 핵심적인 측면은 여러 컴포넌트의 호출을 조정하고 컴포넌트 간의 데이터 교환을 처리하는 기능을 포함합니다. 특히, 여러 컴포넌트는 다양한 기술을 사용하여 주문, 고객 정보 등에서의 데이터와 같이 비즈니스 항목을 나타낼 수 있습니다. 예를 들어, COBOL 사본에서 정보를 구성하는 레거시 응용프로그램과 비즈니스 항목을 나타내기 위해 엔티티 EJB(Enterprise Java™ Bean)를 사용하는 Java 응용프로그램을 통합해야 합니다. 그러므로, 통합 솔루션을 단순하게 작성하기 위한 플랫폼은 데이터 처리를 위해 백엔드 시스템에서 사용하는 기술과 무관하게 비즈니스 항목을 나타내는 일반적인 방법을 제공해야 합니다. 이러한 목적은 *비즈니스 오브젝트 프레임워크*를 통해 WebSphere Process Server 및 WebSphere Enterprise Service Bus에서 달성할 수 있습니다.

비즈니스 오브젝트 프레임워크는 개발자가 XML 스키마를 사용하여 Java 코드를 통해 비즈니스 데이터의 구조를 정의하고 이들 데이터 구조(비즈니스 오브젝트)의 인스턴스에 액세스하고 이를 조작하게 합니다. 비즈니스 오브젝트 프레임워크는 서비스 데이터 오브젝트(SDO) 표준을 기초로 합니다.

SCA(Service Component Architecture) 프로그래밍 모델

SCA 프로그래밍 모델은 모든 솔루션의 토대가 WebSphere Process Server 및 WebSphere Enterprise Service Bus에서 개발됨을 나타냅니다. SCA는 재사용 가능한 컴포넌트에서 개발자가 서비스 구현을 캡슐화하는 방법을 제공합니다. 이는 사용하는 기술에 대해 독립적인 방식으로 인터페이스, 구현 및 참조를 정의할 수 있도록 합니다. 이 접근 방식은 사용자가 선택한 어떤 기술이든 요소를 바인드하는 기회를 제공합니다. 이러한 컴포넌트의 호출을 사용 가능하게 하는 SCA 클라이언트 프로그래밍 모델도 있습니다. 특히, Java에 기초한 런타임 인프라가 비Java 런타임과 상호작용할 수 있도록 합니다. SCA는 서비스 호출을 위한 데이터 항목으로 비즈니스 오브젝트를 사용합니다.

도구 및 제품

IBM WebSphere Integration Developer는 통합된 개발 환경으로, 방금 언급한 기술에 근거하여 비즈니스 통합 솔루션을 작성하고 구성하기 위해 필요한 모든 도구를 포함하고 있습니다. 일반적으로 이들 솔루션은 WebSphere Process Server로 전개되며, 어떤 경우에는 WebSphere Enterprise Service Bus로 전개됩니다.

제 2 장 바인딩

SOA(Service Oriented Architecture)의 핵심에는 컴퓨팅 장치 간 상호작용에 의해 수행되는 기능 단위인 *서비스*라는 개념이 있습니다. *내보내기*에서는 모듈 내의 SCA(Service Component Architecture) 컴포넌트가 해당 서비스를 외부 클라이언트에 제공할 수 있도록 모듈의 외부 인터페이스(또는 액세스 위치)를 정의합니다. *가져오기*에서는 모듈 내에서 서비스를 호출할 수 있도록 모듈 외부에서 서비스에 대한 인터페이스를 정의합니다. 가져오기 및 내보내기와 함께 프로토콜 특정 *바인딩*을 사용하여 데이터와 모듈 간 전송을 수행하는 수단을 지정합니다.

내보내기

외부 클라이언트는 다양한 형식의 데이터(예: XML, CSV, COBOL 및 JavaBean)로 다양한 프로토콜(예: HTTP, JMS, MQ 및 RMI/IIOP)을 통해 통합 모듈의 SCA 컴포넌트를 호출할 수 있습니다. 내보내기는 외부 소스에서 이러한 요청을 수신한 후 SCA 프로그래밍 모델을 사용하여 WebSphere Process Server 컴포넌트를 호출하는 컴포넌트입니다.

예를 들어, 다음 그림에서 내보내기는 HTTP 프로토콜을 통해 클라이언트 응용프로그램에서 요청을 수신합니다. SCA 컴포넌트에서 사용하는 형식인 비즈니스 오브젝트로 데이터가 변환됩니다. 그런 다음, 해당 데이터 오브젝트로 컴포넌트가 호출됩니다.

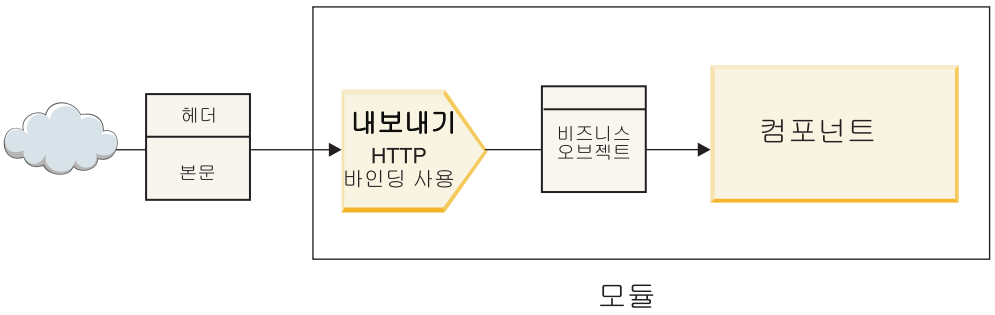


그림 2. HTTP 바인딩이 있는 내보내기

가져오기

SCA 컴포넌트는 다른 형식의 데이터를 예상하는 비SCA 외부 서비스를 호출할 수 있습니다. SCA 프로그래밍 모델을 사용하여 외부 서비스를 호출하기 위해 SCA 컴포넌트에서 가져오기가 사용됩니다. 그런 다음, 가져오기는 서비스에서 예상하는 방식으로 대상 서비스를 호출합니다.

예를 들어, 다음 그림에서는 가져오기에 의해 SCA 컴포넌트로부터의 요청이 외부 서비스에 전송됩니다. SCA 컴포넌트에서 사용하는 형식인 비즈니스 오브젝트가 서비스에서 예상하는 형식으로 변환되고 서비스가 호출됩니다.

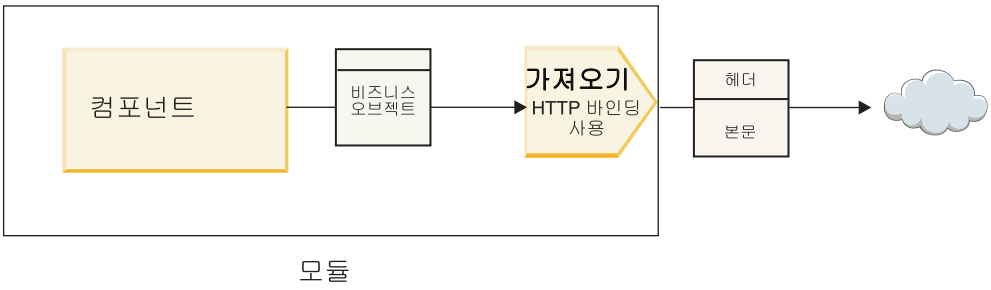


그림 3. HTTP 바인딩이 있는 가져오기

바인딩 목록

WebSphere Integration Developer를 사용하여 가져오기 또는 내보내기를 위한 바인딩을 생성하고 바인딩을 구성합니다. 사용 가능한 바인딩 유형이 다음 목록에 설명되어 있습니다.

- SCA

  SCA 바인딩(기본값)을 사용하면 서비스가 다른 SCA 모듈에 있는 서비스와 통신할 수 있습니다. SCA 바인딩이 있는 가져오기를 사용하여 다른 SCA 모듈에 있는 서비스에 액세스합니다. SCA 바인딩이 있는 내보내기를 사용하여 다른 SCA 모듈에 서비스를 제공합니다.

- 웹 서비스

  웹 서비스 바인딩을 통해 상호 운영이 가능한 SOAP 메시지 및 서비스 품질을 사용하여 외부 서비스에 액세스할 수 있습니다. 웹 서비스 바인딩을 사용하여 SOAP 메시지의 파트로 첨부를 포함할 수 있습니다.

  웹 서비스 바인딩은 SOAP/HTTP(HTTP를 통한 SOAP) 또는 SOAP/JMS(JMS를 통한 SOAP) 전송 프로토콜을 사용할 수 있습니다. SOAP 메시지를 전달하는 데 사용된 전송(HTTP 또는 JMS)과 상관 없이 웹 서비스 바인딩은 항상 요청/응답 상호작용을 동기적으로 처리합니다.

- HTTP

  HTTP 바인딩을 통해 HTTP 프로토콜을 사용하여 외부 서비스에 액세스할 수 있으며, 여기서 SOAP이 아닌 메시지가 사용되거나, 직접 HTTP 액세스가 필요합니다. 이 바인딩은 HTTP 모델을 기반으로 하는 웹 서비스(즉, GET, PUT, DELETE 등과 같은 잘 알려진 HTTP 인터페이스 조작을 사용하는 서비스)에 대해 작업하는 경우에 사용됩니다.

- EJB(Enterprise JavaBeans™)

  EJB 바인딩을 사용하면 Java EE 서버에서 실행 중인 Java EE 비즈니스 로직이 제공하는 서비스가 SCA 컴포넌트와 상호작용할 수 있습니다.

- EIS

  엔터프라이즈 정보 시스템(EIS) 바인딩을 사용하면 엔터프라이즈 정보 시스템의 서비스에 액세스하거나 EIS에서 서비스를 사용할 수 있습니다(JCA 자원 어댑터와 함께 사용된 경우).

- JMS 바인딩

  JMS(Java Message Service), 일반 JMS 및 MQ JMS(WebSphere MQ JMS) 바인딩은 메시지 큐를 통한 비동기 통신이 신뢰도를 위해 중요한 메시징 시스템과의 상호작용에 사용됩니다.

  JMS 바인딩 중 하나가 있는 내보내기에서는 큐에서 메시지의 도달을 감시하며 응답이 있는 경우 비동기적으로 응답 큐로 전송합니다. JMS 바인딩 중 하나가 있는 가져오기에서는 메시지를 빌드하여 JMS 큐에 전송하며 응답이 있는 경우 큐에서 응답의 도달을 감시합니다.

  – JMS

    JMS 바인딩을 사용하면 WebSphere 임베디드 JMS 프로바이더에 액세스할 수 있습니다.

  – 일반 JMS

    일반 JMS 바인딩을 사용하면 IBM이 아닌 벤더 메시징 시스템에 액세스할 수 있습니다.

  – MQ JMS

    MQ JMS 바인딩을 사용하면 WebSphere MQ 메시징 시스템의 JMS 서브세트에 액세스할 수 있습니다. 해당 기능의 JMS 서브세트가 응용프로그램에 대해 충분한 경우 이 바인딩을 사용합니다.

- MQ

  WebSphere MQ 바인딩을 사용하면 MQ 기본 응용프로그램을 SOA(Service Oriented Architecture) 프레임워크로 가져온 후 MQ 특정 헤더 정보에 대한 액세스를 제공하여 MQ 기본 응용프로그램과 통신할 수 있습니다. MQ 기본 기능을 사용해야 하는 경우 이 바인딩을 사용합니다.

내보내기 및 가져오기 바인딩 개요

내보내기를 사용하면 외부 클라이언트에 사용 가능한 통합 모듈에 서비스를 작성할 수 있으며 가져오기를 사용하면 통합 모듈의 SCA 컴포넌트가 외부 서비스를 호출할 수 있습니다. 내보내기 또는 가져오기와 연관된 바인딩은 프로토콜 메시지와 비즈니스 오브젝트 간의 관계를 지정합니다. 조작과 결함을 선택하는 방식도 지정합니다.

내보내기를 통한 정보 플로우

연관된 바인딩을 통해 판별한 특정한 전송을 수행하여 내보내기가 연결된 컴포넌트에 대한 요청을 내보내기에서 수신합니다(예: HTTP).

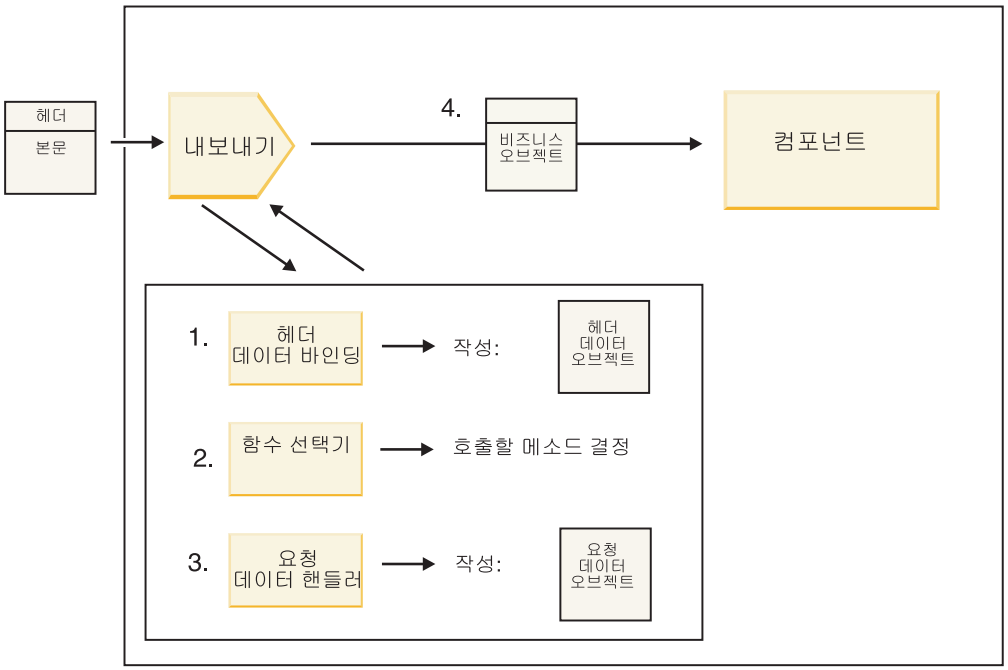


그림 4. 컴포넌트에 내보내기를 통한 요청 플로우

내보내기가 요청을 수신하면 다음과 같은 일련의 이벤트가 발생합니다.

1. WebSphere MQ 바인딩의 경우에만, 헤더 데이터 바인딩이 프로토콜 헤더를 헤더 데이터 오브젝트로 변환합니다.
2. 함수 선택기가 프로토콜 메시지의 기본 메소드 이름을 판별합니다. 기본 메소드 이름은 내보내기 구성에 의해 내보내기 인터페이스에서의 조작 이름으로 맵핑됩니다.
3. 메소드의 요청 데이터 핸들러 또는 데이터 바인딩이 요청을 요청 비즈니스 오브젝트로 변환합니다.
4. 내보내기가 요청 비즈니스 오브젝트를 사용하여 컴포넌트 메소드를 호출합니다.
   - HTTP 내보내기 바인딩, 웹 서비스 내보내기 바인딩 및 EJB 내보내기 바인딩은 SCA 컴포넌트를 동기적으로 호출합니다.

- JMS, 일반 JMS, MQ JMS 및 WebSphere MQ 내보내기 바인딩은 SCA 컴포넌트를 비동기적으로 호출합니다.

컨텍스트 전파가 사용 가능한 경우, 내보내기는 프로토콜을 통해 수신하는 헤더 및 사용자 특성을 전파할 수 있습니다. 그런 다음, 내보내기에 연결된 컴포넌트가 이러한 헤더 및 사용자 특성에 액세스할 수 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center에서 『전파』를 참조하십시오.

양방향 조작의 경우 컴포넌트는 응답을 리턴합니다.

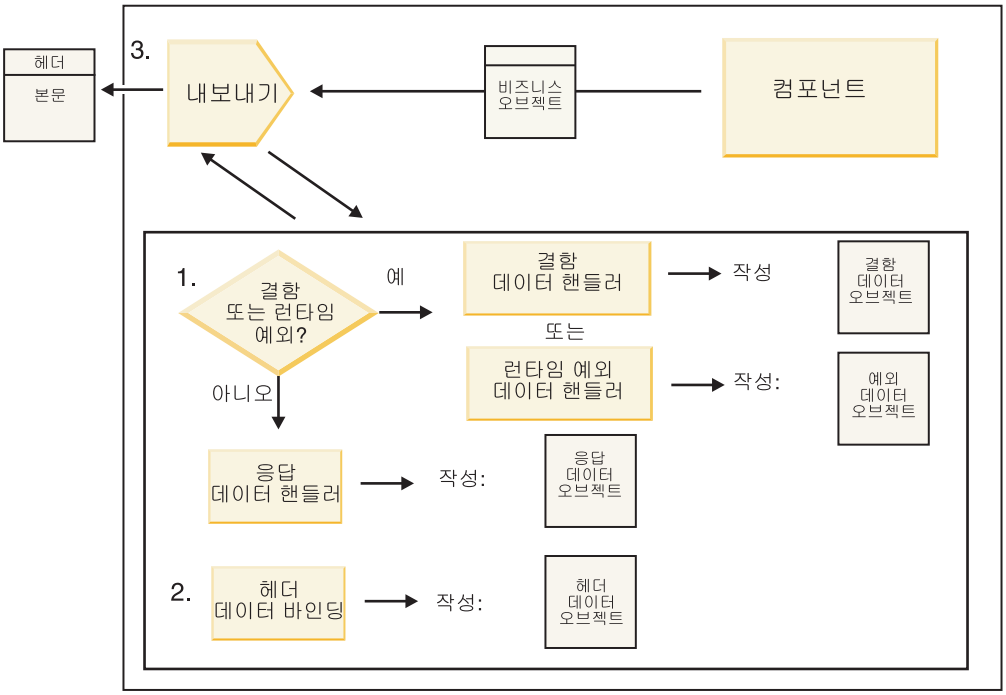


그림 5. 내보내기를 통한 응답 플로우

다음과 같은 일련의 단계가 발생합니다.

1. 내보내기 바인딩에 의해 표준 응답 메시지가 수신되면 메소드의 응답 데이터 핸들러 또는 데이터 바인딩이 비즈니스 오브젝트를 응답으로 변환합니다.

   응답이 결함인 경우 메소드의 결함 데이터 핸들러 또는 데이터 바인딩이 결함을 결함 응답으로 변환합니다.

   HTTP 내보내기 바인딩의 경우에만, 응답이 런타임 예외인 경우 구성되었다면 런타임 예외 데이터 핸들러가 호출됩니다.
2. WebSphere MQ 바인딩의 경우에만 헤더 데이터 바인딩이 헤더 데이터 오브젝트를 프로토콜 헤더로 변환합니다.
3. 내보내기가 전송을 통해 서비스 응답을 전송합니다.

가져오기를 통한 정보 플로우

컴포넌트는 가져오기를 사용하여 모듈 외부에 있는 서비스로 요청을 전송합니다. 연관된 바인딩으로 판별된 전송을 통해 요청이 전송됩니다.

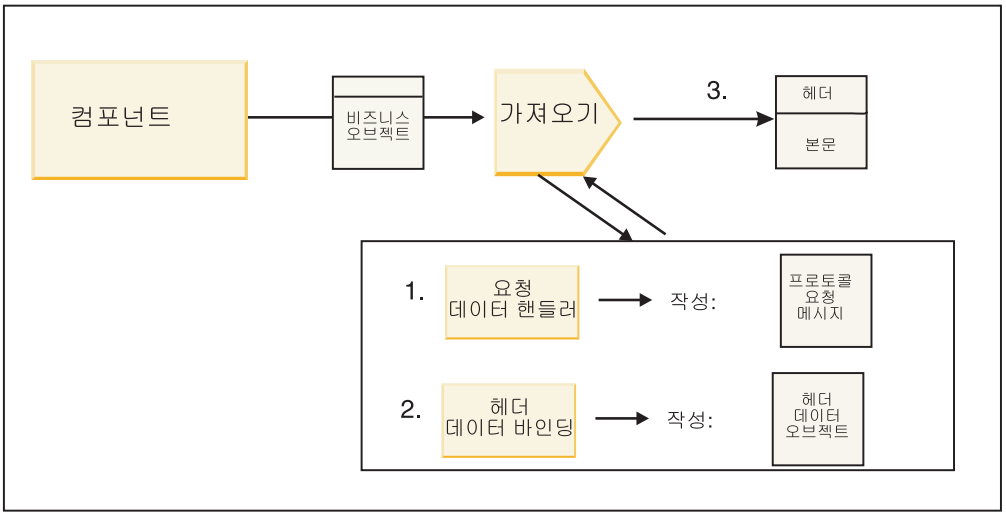


그림 6. 가져오기를 통한 컴포넌트에서 서비스로의 플로우

컴포넌트는 요청 비즈니스 오브젝트를 사용하여 가져오기를 호출합니다.

주:

- HTTP 가져오기 바인딩, 웹 서비스 가져오기 바인딩 및 EJB 가져오기 바인딩은 호출 컴포넌트에 의해 동기적으로 호출되어야 합니다.
- JMS, 일반 JMS, MQ JMS 및 WebSphere MQ 가져오기 바인딩은 비동기적으로 호출되어야 합니다.

컴포넌트가 가져오기를 호출한 후에, 다음과 같은 일련의 이벤트가 발생합니다.

1. 메소드의 요청 데이터 핸들러 또는 데이터 바인딩이 요청 비즈니스 오브젝트를 프로토콜 요청 메시지로 변환합니다.
2. WebSphere MQ 바인딩의 경우에만 메소드의 헤더 데이터 바인딩이 프로토콜 헤더에서 헤더 비즈니스 오브젝트를 설정합니다.
3. 가져오기는 전송을 통해 서비스 요청을 사용하여 서비스를 호출합니다.

양방향 조작인 경우 서비스는 응답을 리턴하고 다음과 같은 일련의 단계가 발생합니다.

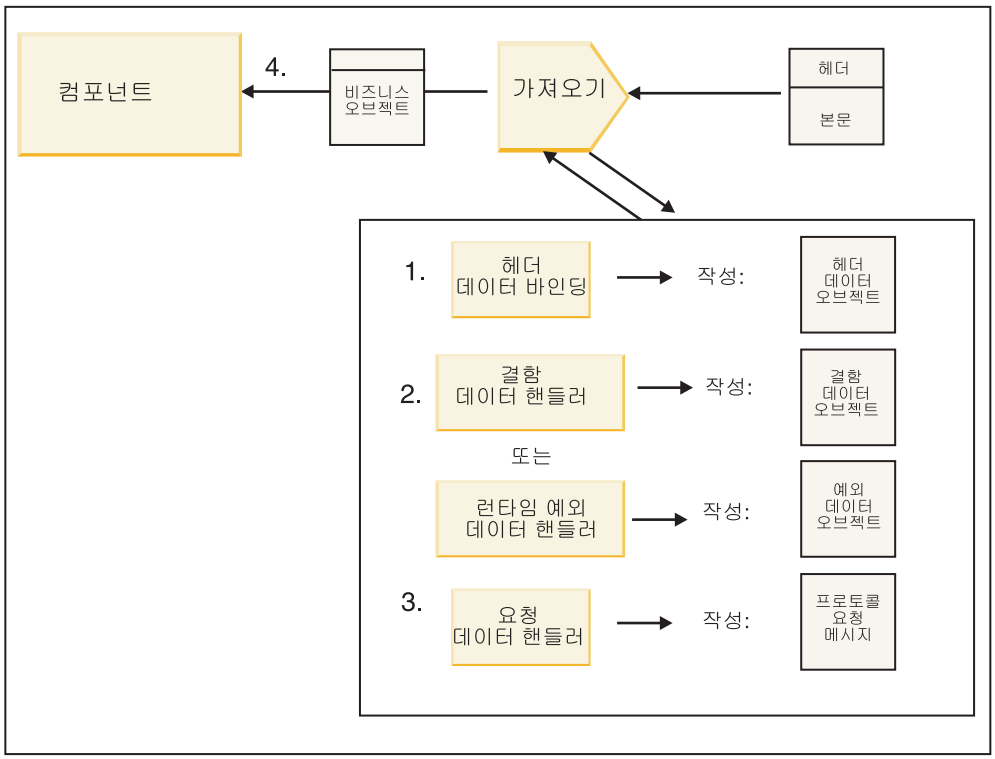


그림 7. 가져오기를 통한 응답 플로우

1. WebSphere MQ 바인딩의 경우에만, 헤더 데이터 바인딩이 프로토콜 헤더를 헤더 데이터 오브젝트로 변환합니다.
2. 응답이 결함인지 여부를 결정합니다.
   - 응답이 결함인 경우 결함 선택기가 결함을 검사하여 맵핑하는 WSDL 결함을 판별합니다. 그런 다음, 메소드의 결함 데이터 핸들러가 결함을 결함 응답으로 변환합니다.
   - 응답이 런타임 예외인 경우 런타임 예외 데이터 핸들러가 호출됩니다(구성된 경우).
3. 메소드의 응답 데이터 핸들러 또는 바인딩이 응답을 응답 비즈니스 오브젝트로 변환합니다.
4. 가져오기가 응답 비즈니스 오브젝트를 컴포넌트에 리턴합니다.

바인딩 구성 내보내기 및 가져오기

내보내기 및 가져오기 바인딩의 핵심 중 하나는 데이터 형식 변환으로, 데이터가 기본 연결 형식에서 비즈니스 오브젝트로 맵핑되는(직렬화 해제) 방식을 표시합니다. 내보내기와 연관된 바인딩의 경우에는 데이터에서 수행해야 하는 조작을 표시하도록 함수 선택기를 지정할 수도 있습니다. 내보내기 또는 가져오기와 연관된 바인딩의 경우 처리 중에 발생하는 결함이 처리되는 방법을 표시할 수 있습니다.

또한 바인딩에 대한 전송 정보를 지정하십시오. 예를 들어, HTTP 바인딩의 경우 엔드 포인트 URL을 지정합니다. WebSphere Integration Developer Information Center에서 자세한 정보를 찾을 수 있습니다. 예를 들어, HTTP 바인딩의 경우 전송 고유의 정보가 『HTTP 가져오기 바인딩 생성』 및 『HTTP 내보내기 바인딩 생성』 주제에 설명되어 있습니다.

가져오기 및 내보내기에서 데이터 형식 변환

내보내기 또는 가져오기 바인딩이 WebSphere Integration Developer에서 구성될 때 지정하는 구성 특성 중 하나는 바인딩에서 사용하는 데이터 형식입니다.

- 클라이언트 응용프로그램이 SCA 컴포넌트에 요청을 전송하고 SCA 컴포넌트에서 응답을 수신하는 내보내기 바인딩의 경우 기본 데이터의 형식을 표시합니다. 형식에 따라 시스템에서는 기본 데이터를 비즈니스 오브젝트로 변환(SCA 컴포넌트에 의해 사용됨)하고 반대로 비즈니스 오브젝트를 기본 데이터로 변환(클라이언트 응용프로그램에 대한 응답임)하기 위해 적합한 데이터 핸들러 또는 데이터 바인딩을 선택합니다.
- SCA 컴포넌트가 모듈 외부의 서비스에 요청을 전송하고 해당 서비스에서 응답을 수신하는 가져오기 바인딩의 경우 기본 데이터의 데이터 형식을 표시합니다. 형식에 따라 시스템에서는 비즈니스 오브젝트를 기본 데이터로 또는 그 반대로 변환하기 위해 적합한 데이터 핸들러 또는 데이터 바인딩을 선택합니다.

WebSphere Process Server에서는 사전 정의된 데이터 형식 세트 및 해당 형식을 지원하는 해당 데이터 핸들러 또는 데이터 바인딩을 제공합니다. 자체 사용자 정의 데이터 핸들러를 작성하여 해당 데이터 핸들러에 대한 데이터 형식을 등록할 수도 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center에서 『데이터 핸들러 개발』 주제를 참조하십시오.

- *데이터 핸들러*는 프로토콜 중립적이며 데이터를 한 형식에서 다른 형식으로 변환합니다. WebSphere Process Server에서 데이터 핸들러는 일반적으로 기본 데이터(예: XML, CSV 및 COBOL)를 비즈니스 오브젝트로 변환하고 비즈니스 오브젝트를 기본 데이터로 변환합니다. 데이터 핸들러는 프로토콜 중립적이므로 동일한 데이터 핸들러를 다양한 내보내기 및 가져오기 바인딩과 함께 재사용할 수 있습니다. 예를 들어, 동일한 XML 데이터 핸들러를 HTTP 내보내기 또는 가져오기 바인딩 또는 JMS 내보내기 또는 가져오기 바인딩과 함께 사용할 수 있습니다.
- *데이터 바인딩*은 기본 데이터에서 비즈니스 오브젝트로(또는 그 반대로)도 변환하지만 프로토콜마다 다릅니다. 예를 들어, HTTP 데이터 바인딩은 HTTP 내보내기 또는 가져오기 바인딩과 함께만 사용할 수 있습니다. 데이터 핸들러와는 달리 HTTP 데이터 바인딩은 MQ 내보내기 또는 가져오기 바인딩과 함께 재사용할 수 없습니다.

  주: 세 개의 HTTP 데이터 바인딩(HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML 및 HTTPServiceGatewayDataBinding)은 WebSphere Process Server 버전 7.0에서 권장되지 않습니다. 가능하면 데이터 핸들러를 사용하십시오.

이전에 언급한 대로 필요한 경우 사용자 정의 데이터 핸들러를 작성할 수 있습니다. 또한 사용자 정의 데이터 바인딩을 작성할 수 있지만, 다중 바인딩에서 사용될 수 있으므로 사용자 정의 데이터 핸들러를 작성하는 것이 좋습니다.

데이터 핸들러

데이터 핸들러는 내보내기 및 가져오기 바인딩에 대해 구성되어 데이터를 프로토콜 중립 방식으로 한 형식에서 다른 형식으로 변환합니다. 여러 데이터 핸들러가 제품의 일부로 제공되지만 필요한 경우 자체 데이터 핸들러를 작성할 수도 있습니다. 두 가지 레벨 중 하나에서 데이터 핸들러를 내보내기 또는 가져오기 바인딩과 연관시킬 수 있으며, 내보내기 또는 가져오기의 인터페이스에서 모든 조작과 연관시키거나 요청 또는 응답의 특정 조작과 연관시킬 수 있습니다.

사전 정의된 데이터 핸들러

WebSphere Integration Developer를 사용하여 사용하려는 데이터 핸들러를 지정합니다.

사전 정의된 데이터 핸들러는 다음 표에 표시되어 있으며 이 표에는 각 데이터 핸들러가 인바운드 및 아웃바운드 데이터를 변환하는 방법에 대해서도 설명합니다.

주: 표시한 위치를 제외하고, 이 데이터 핸들러는 JMS, 일반 JMS, MQ JMS, WebSphere MQ 및 HTTP 바인딩과 함께 사용될 수 있습니다.
자세한 정보는 WebSphere Integration Developer Information Center에서 『데이터 핸들러』 주제를 참조하십시오.

표 1. 사전 정의된 데이터 핸들러

데이터 핸들러	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
ATOM	ATOM 피드를 ATOM 피드 비즈니스 오브젝트로 구문 분석합니다.	ATOM 피드 비즈니스 오브젝트를 ATOM 피드로 직렬화합니다.
구분됨	구분된 데이터를 비즈니스 오브젝트로 구문 분석합니다.	비즈니스 오브젝트를 구분된 데이터(CSV 포함)로 직렬화합니다.
고정 너비	고정 너비 데이터를 비즈니스 오브젝트로 구문 분석합니다.	비즈니스 오브젝트를 고정 너비 데이터로 직렬화합니다.
WTX에서 처리	데이터 형식 변환을 WTX(WebSphere Transformation Extender)로 위임합니다. WTX 맵 이름은 데이터 핸들러에 의해 파생됩니다.	데이터 형식 변환을 WTX(WebSphere Transformation Extender)로 위임합니다. WTX 맵 이름은 데이터 핸들러에 의해 파생됩니다.
WTX 호출자가 처리	데이터 형식 변환을 WTX(WebSphere Transformation Extender)로 위임합니다. WTX 맵 이름은 사용자가 제공합니다.	데이터 형식 변환을 WTX(WebSphere Transformation Extender)로 위임합니다. WTX 맵 이름은 사용자가 제공합니다.

표 1. 사전 정의된 데이터 핸들러 (계속)

데이터 핸들러	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
JAXB	JAXB(Java Architecture for XML Binding) 스펙이 정의한 맵핑 규칙을 사용하여 Java Bean을 비즈니스 오브젝트로 직렬화합니다.	JAXB 스펙이 정의한 맵핑 규칙을 사용하여 비즈니스 오브젝트를 Java Bean으로 직렬화 해제합니다.
JAXWS **주:** 데이터 핸들러는 EJB 바인딩과 함께인 경우에만 사용될 수 있습니다.	JAX-WS(Java API for XML Web Services) 스펙이 정의한 맵핑 규칙을 사용하여 응답 Java 오브젝트 또는 예외 Java 오브젝트를 응답 비즈니스 오브젝트로 변환하기 위해 EJB 바인딩이 사용합니다.	JAX-WS 스펙이 정의한 맵핑 규칙을 사용하여 비즈니스 오브젝트를 전송 Java 메소드 매개변수로 변환하기 위해 EJB 바인딩이 사용합니다.
JSON	JSON 데이터를 비즈니스 오브젝트로 구문 분석합니다.	비즈니스 오브젝트를 JSON 데이터로 직렬화합니다.
기본 본문	기본 바이트, 텍스트, 맵, 스트림 또는 오브젝트를 다섯 개의 기본 비즈니스 오브젝트(텍스트, 바이트, 맵, 스트림 또는 오브젝트) 중 하나로 구문 분석합니다.	다섯 개의 기본 비즈니스 오브젝트를 바이트, 텍스트, 맵, 스트림 또는 오브젝트로 변환합니다.
SOAP	SOAP 메시지(및 헤더)를 비즈니스 오브젝트로 구문 분석합니다.	비즈니스 오브젝트를 SOAP 메시지로 직렬화합니다.
XML	XML 데이터를 비즈니스 오브젝트로 구문 분석합니다.	비즈니스 오브젝트를 XML 데이터로 직렬화합니다.
UTF8XMLDataHandler	UTF-8 인코드 XML 데이터를 비즈니스 오브젝트로 구문 분석합니다.	메시지 전송 시 비즈니스 오브젝트를 UTF-8 인코드 XML 데이터로 직렬화합니다.

데이터 핸들러 작성

데이터 핸들러에 대한 자세한 정보는 WebSphere Integration Developer Information Center의 『데이터 핸들러 개발』 주제에서 찾을 수 있습니다.

데이터 바인딩

데이터 바인딩은 데이터를 한 형식에서 다른 형식으로 변환할 수 있도록 내보내기 및 가져오기 바인딩을 대상으로 구성되어 있습니다. 데이터 바인딩은 프로토콜마다 고유합니다. 여러 데이터 바인딩이 제품의 일부로 제공되지만 필요한 경우 자체 데이터 바인딩을 작성할 수도 있습니다. 두 가지 레벨 중 하나에서 데이터 바인딩을 내보내기 또는 가져오기 바인딩과 연관시킬 수 있으며, 내보내기 또는 가져오기의 인터페이스에서 모든 조작과 연관시키거나 요청 또는 응답의 특정 조작과 연관시킬 수 있습니다.

WebSphere Integration Developer를 사용하여 사용할 데이터 바인딩을 지정하거나 자체 데이터 바인딩을 작성합니다. 데이터 바인딩 작성에 대한 설명은 WebSphere Integration Developer Information Center의 『JMS, MQ JMS 및 일반 JMS 바인딩 개요』 절을 참조하십시오.

JMS 바인딩

다음 표에는 다음과 함께 사용할 수 있는 데이터 바인딩이 표시됩니다.

- JMS 바인딩
- 일반 JMS 바인딩
- WebSphere MQ JMS 바인딩

이 표에는 데이터 바인딩이 수행하는 타스크에 대한 설명도 포함되어 있습니다.

표 2. JMS 바인딩의 사전 정의된 데이터 바인딩

데이터 바인딩	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
직렬화된 Java 오브젝트	Java 직렬화된 오브젝트를 비즈니스 오브젝트로 변환합니다 (WSDL에서 입력 또는 출력 유형으로 맵핑됨).	비즈니스 오브젝트를 Java 오브젝트 메시지의 Java 직렬화 오브젝트로 직렬화합니다.
랩핑된 바이트	수신 JMS 바이트 메시지에서 바이트를 추출하여 JMSBytesBody 비즈니스 오브젝트에 랩핑합니다.	JMSBytesBody 비즈니스 오브젝트에서 바이트를 추출하여 전송 JMS 바이트 메시지에 랩핑합니다.
랩핑된 맵 항목	수신 JMS 맵 메시지에 있는 모든 항목의 이름, 값 및 유형 정보를 추출하고 MapEntry 비즈니스 오브젝트 목록을 작성합니다. 그런 다음, 목록을 JMSMapBody 비즈니스 오브젝트에 랩핑합니다.	JMSMapBody 비즈니스 오브젝트의 MapEntry 목록에서 이름, 값 및 유형 정보를 추출하고 전송 JMS 맵 메시지에 해당 항목을 작성합니다.
랩핑된 오브젝트	수신 JMS 오브젝트 메시지에서 오브젝트를 추출하여 JMSObjectBody 비즈니스 오브젝트에 랩핑합니다.	JMSObjectBody 비즈니스 오브젝트에서 오브젝트를 추출하여 전송 JMS 오브젝트 메시지에 랩핑합니다.
랩핑된 텍스트	수신 JMS 텍스트 메시지에서 텍스트를 추출하여 JMSTextBody 비즈니스 오브젝트에 랩핑합니다.	JMSTextBody 비즈니스 오브젝트에서 텍스트를 추출하여 전송 JMS 텍스트 메시지에 랩핑합니다.

WebSphere MQ 바인딩

다음 표에는 WebSphere MQ와 함께 사용할 수 있는 데이터 바인딩이 표시되고 데이터 바인딩이 수행하는 타스크가 설명되어 있습니다.

표 3. WebSphere MQ 바인딩의 사전 정의된 데이터 바인딩

데이터 바인딩	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
직렬화된 Java 오브젝트	Java 직렬화된 오브젝트를 수신 메시지에서 비즈니스 오브젝트로 변환합니다(WSDL에서 입력 또는 출력 유형으로 맵핑됨).	비즈니스 오브젝트를 전송 메시지의 Java 직렬화 오브젝트로 변환합니다.

표 3. WebSphere MQ 바인딩의 사전 정의된 데이터 바인딩 (계속)

데이터 바인딩	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
랩핑된 바이트	구조화되지 않은 MQ 바이트 메시지에서 바이트를 추출하여 JMSBytesBody 비즈니스 오브젝트에 랩핑합니다.	JMSBytesBody 비즈니스 오브젝트에서 바이트를 추출하여 구조화되지 않은 전송 MQ 바이트 메시지에 바이트를 랩핑합니다.
랩핑된 텍스트	구조화되지 않은 MQ 텍스트 메시지에서 텍스트를 추출하여 JMSTextBody 비즈니스 오브젝트에 랩핑합니다.	JMSTextBody 비즈니스 오브젝트에서 텍스트를 추출하여 구조화되지 않은 MQ 텍스트 메시지에 랩핑합니다.
랩핑된 스트림 항목	수신 JMS 스트림 메시지에 있는 모든 항목의 이름 및 유형 정보를 추출하고 StreamEntry 비즈니스 오브젝트 목록을 작성합니다. 그런 다음, 목록을 JMSStreamBody 비즈니스 오브젝트에 랩핑합니다.	JMSStreamBody 비즈니스 오브젝트의 StreamEntry 목록에서 이름 및 유형 정보를 추출하고 전송 JMSStreamMessage에 해당 항목을 작성합니다.

19 페이지의 표 3에서 나열된 데이터 바인딩 이외에 WebSphere MQ는 헤더 데이터 바인딩을 사용합니다. 세부사항은 WebSphere Integration Developer Information Center를 참조하십시오.

HTTP 바인딩

다음 표에는 HTTP와 함께 사용할 수 있는 데이터 바인딩이 표시되고 데이터 바인딩이 수행하는 타스크가 설명되어 있습니다.

표 4. HTTP 바인딩의 사전 정의된 데이터 바인딩

데이터 바인딩	기본 데이터를 비즈니스 오브젝트로 변환	비즈니스 오브젝트를 기본 데이터로 변환
랩핑된 바이트	수신 HTTP 메시지 본문에서 바이트를 추출하여 HTTPBytes 비즈니스 오브젝트에 랩핑합니다.	HTTPBytes 비즈니스 오브젝트에서 바이트를 추출하여 전송 HTTP 메시지 본문에 추가합니다.
랩핑된 텍스트	수신 HTTP 메시지 본문에서 텍스트를 추출하여 HTTPText 비즈니스 오브젝트에 랩핑합니다.	HTTPText 비즈니스 오브젝트에서 텍스트를 추출하여 전송 HTTP 메시지 본문에 추가합니다.

내보내기 바인딩에서 함수 선택기

함수 선택기는 요청 메시지의 데이터에서 수행해야 하는 조작을 표시하기 위해 사용됩니다. 함수 선택기는 내보내기 바인딩의 파트로서 구성됩니다.

인터페이스를 표시하는 SCA 내보내기가 있다고 간주하십시오. 인터페이스에는 두 가지 조작(작성 및 갱신)이 포함되어 있습니다. 내보내기에는 큐에서 읽는 JMS 바인딩이 있습니다.

메시지가 큐에 도달하면 내보내기에 연관된 데이터가 전달되지만, 내보내기의 인터페이스로부터 어떤 조작이 연결된 컴포넌트에서 호출되어야 합니까? 이 조작은 함수 선택기 및 내보내기 바인딩 구성에 의해 판별됩니다.

함수 선택기는 기본 함수 이름(메시지를 전송한 클라이언트 시스템의 함수 이름)을 리턴합니다. 그런 다음, 기본 함수 이름은 내보내기와 연관된 인터페이스의 조작 또는 함수 이름에 맵핑됩니다. 예를 들어, 다음 그림에서 함수 선택기는 수신 메시지의 기본 함수 이름 (CRT)을 리턴하고 기본 함수 이름이 작성 조작에 맵핑되며 비즈니스 오브젝트가 작성 조작을 통해 SCA 컴포넌트에 전송됩니다.

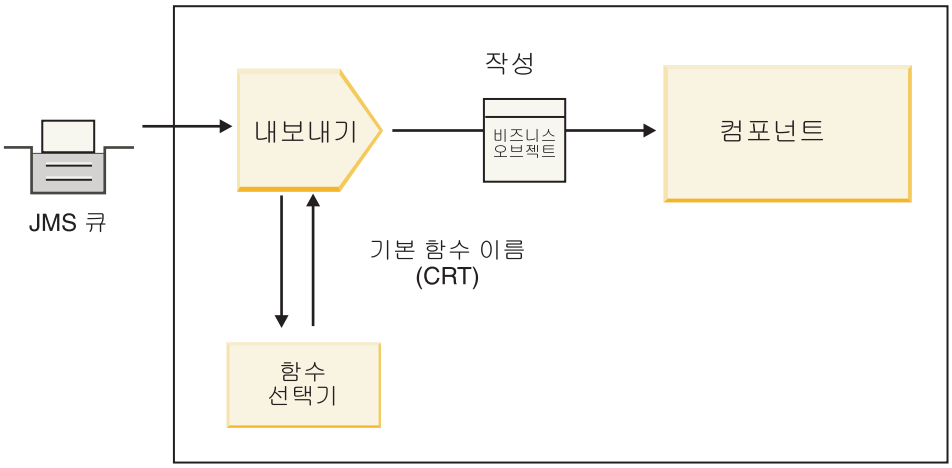


그림 8. 함수 선택기

인터페이스에 하나의 조작만 있는 경우, 함수 선택기를 지정할 필요가 없습니다.

여러 함수 선택기가 사용 가능하며 다음 절에 표로 설명됩니다.

JMS 바인딩

다음 표에는 다음과 함께 사용할 수 있는 함수 선택기가 설명되어 있습니다.

- JMS 바인딩
- 일반 JMS 바인딩
- WebSphere MQ JMS 바인딩

표 5. JMS 바인딩의 사전 정의된 함수 선택기

함수 선택기	설명
단순 JMS 데이터 바인딩의 JMS 함수 선택기	조작 이름을 선택하기 위해 메시지의 JMSType 특성을 사용합니다.
JMS 헤더 특성 함수 선택기	헤더에서 JMS 문자열 특성, TargetFunctionName을 리턴합니다.
JMS 서비스 게이트웨이 함수 선택기	클라이언트에 의해 설정된 JMSReplyTo 특성을 보고 요청이 한 가지 방식인지 아니면 두 가지 방식 조작인지 판별하십시오.

WebSphere MQ 바인딩

다음 표에는 WebSphere MQ와 함께 사용할 수 있는 함수 선택기가 설명되어 있습니다.

표 6. WebSphere MQ 바인딩의 사전 정의된 함수 선택기

함수 선택기	설명
MQ handleMessage 함수 선택기	인터페이스에서 조작의 이름으로의 내보내기 메소드 바인딩을 사용하여 맵핑되는 값으로 handleMessage를 리턴합니다.
MQ에서 JMS 기본 함수 선택기 사용	MQRFH2 헤더 폴더의 TargetFunctionName 특성에서 기본 조작을 읽습니다.
MQ에서 메시지 본문 형식을 기본 함수로 사용	마지막 헤더의 형식 필드를 찾아 해당 필드를 문자열로 리턴합니다.
MQ 유형 함수 선택기	MQRFH2 헤더에 있는 Msd, 설정, 유형 및 형식 특성을 포함하는 URL을 검색하여 내보내기 바인딩에서 메소드를 작성합니다.
MQ 서비스 게이트웨이 함수 선택기	조작 이름을 판별하기 위해 MQMD 헤더의 MsgType 특성을 사용합니다.

HTTP 바인딩

다음 표에는 HTTP 바인딩과 함께 사용할 수 있는 함수 선택기가 설명되어 있습니다.

표 7. HTTP 바인딩의 사전 정의된 함수 선택기

함수 선택기	설명
TargetFunctionName 헤더에 근거한 HTTP 함수 선택기	내보내기에서 런타임 시 호출하는 조작을 판별하기 위해 클라이언트에서 TargetFunctionName HTTP 헤더 특성을 사용합니다.
URL 및 HTTP 메소드에 근거한 HTTP 함수 선택기	내보내기에 정의된 기본 조작을 판별하기 위해 클라이언트로부터 HTTP 메소드를 이용해 추가된 URL에서 상대 경로를 사용합니다.
조작 이름이 있는 URL에 근거한 HTTP 서비스 게이트웨이 함수 선택기	"operationMode = oneWay"가 요청 URL에 추가된 경우 URL에 근거한 메소드를 호출할지 여부를 판별합니다.

주: WebSphere Integration Developer를 사용하여 자체 함수 선택기를 작성할 수도 있습니다. 함수 선택기 작성에 대한 정보는 WebSphere Integration Developer Information Center에서 제공됩니다. 예를 들어, WebSphere MQ 바인딩의 함수 선택기 작성에 대한 설명은 『MQ 함수 선택기 개요』에 있습니다.

결함 처리

결함 데이터 핸들러를 지정하여 처리 중에 발생하는 결함(예: 비즈니스 예외)을 처리하도록 가져오기 및 내보내기 바인딩을 구성할 수 있습니다. 세 가지 레벨에서 결함 데이

터 핸들러를 설정할 수 있습니다. 결함 데이터 핸들러를 결함, 조작 또는 바인딩을 이용한 모든 조작에 대해 연관시킬 수 있습니다.

결함 데이터 핸들러는 결함 데이터를 처리하여 내보내기 또는 가져오기 바인딩에 의해 전송되는 올바른 형식으로 변환합니다.

- 내보내기 바인딩의 경우 결함 데이터 핸들러는 컴포넌트에서 전송된 예외 비즈니스 오브젝트를 클라이언트 응용프로그램에서 사용할 수 있는 응답 메시지로 변환합니다.
- 가져오기 바인딩의 경우 결함 데이터 핸들러는 서비스에서 전송된 결함 데이터 또는 응답 메시지를 SCA 컴포넌트에서 사용할 수 있는 예외 비즈니스 오브젝트로 변환합니다.

가져오기 바인딩의 경우 바인딩은 응답 메시지가 표준 응답, 비즈니스 결함, 런타임 예외 중 어느 것인지 판별하는 결함 선택기를 호출합니다.

특정 결함, 조작 및 바인딩을 이용한 모든 조작에 대해 결함 데이터 핸들러를 지정할 수 있습니다.

- 결함 데이터 핸들러가 세 레벨 모두에서 설정된 경우에는 특정 결함과 연관된 데이터 핸들러가 호출됩니다.
- 결함 데이터 핸들러가 조작 및 바인딩 레벨에서 설정된 경우에는 조작과 연관된 데이터 핸들러가 호출됩니다.

결함 처리를 지정하기 위해 두 개의 편집기가 WebSphere Integration Developer에서 사용됩니다. 인터페이스 편집기는 조작에 결함이 있는지를 표시하는 데 사용됩니다. 이 인터페이스로 바인딩이 생성된 후 특정 보기의 편집기를 사용하여 결함이 처리되는 방법을 구성할 수 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center에서 『결함 선택기』 주제를 참조하십시오.

내보내기 바인딩에서 결함이 처리되는 방법

클라이언트 응용프로그램으로부터의 요청을 처리하는 중에 결함이 발생하면 내보내기 바인딩이 결함 정보를 클라이언트에 리턴할 수 있습니다. 내보내기 바인딩을 구성하여 결함을 처리하고 클라이언트에 리턴하는 방법을 지정합니다.

WebSphere Integration Developer를 사용하여 내보내기 바인딩을 구성합니다.

요청을 처리하는 동안 클라이언트는 요청을 사용하여 내보내기를 호출하고 내보내기에서는 SCA 컴포넌트를 호출합니다. 요청을 처리하는 동안 SCA 컴포넌트는 비즈니스 응답을 리턴하거나 비즈니스 예외 또는 서비스 런타임 예외를 처리할 수 있습니다. 이러한 예외가 발생하면 다음 그림과 다음 절의 설명대로 내보내기 바인딩은 예외를 결함 메시지로 변환한 후 클라이언트에 전송합니다.

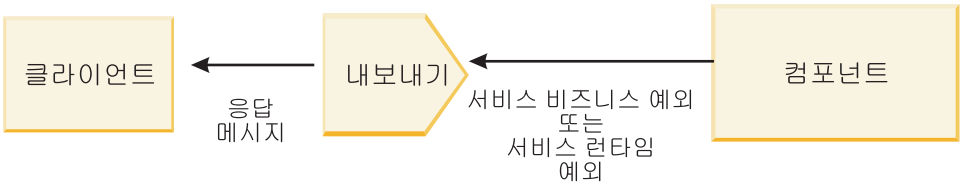


그림 9. 내보내기 바인딩을 통해 컴포넌트에서 클라이언트로 결함 정보가 전송되는 방법

사용자 정의 데이터 핸들러 또는 데이터 바인딩을 작성하여 결함을 처리할 수 있습니다.

비즈니스 결함

비즈니스 결함은 처리 중에 발생하는 비즈니스 오류 또는 예외입니다.

createCustomer 조작이 있는 다음과 같은 인터페이스를 고려하십시오. 이 조작에는 두 가지 비즈니스 결함(CustomerAlreadyExists 및 MissingCustomerId)이 정의되어 있습니다.

그림 10. 두 가지 결함이 있는 인터페이스

이 예제에서는 클라이언트가 고객을 작성하기 위한 요청을 이 SCA 컴포넌트에 전송하고 해당 고객이 이미 있는 경우 컴포넌트는 CustomerAlreadyExists 결함을 내보내기에 전달합니다. 내보내기에서는 이 비즈니스 결함을 다시 호출하는 클라이언트로 전파해야 합니다. 이를 위해서 내보내기에서는 내보내기 바인딩에 설정된 결함 데이터 핸들러를 사용합니다.

내보내기 바인딩에서 비즈니스 결함을 수신하면 다음과 같은 프로세스가 발생합니다.

1. 바인딩은 결함을 처리하기 위해 호출하는 결함 데이터 핸들러를 판별합니다. 서비스 비즈니스 예외에 결함 이름이 포함되어 있는 경우 결함에 설정된 데이터 핸들러가 호출됩니다. 서비스 비즈니스 예외에 결함의 이름이 포함되지 않은 경우 결함 유형을 일치시켜 결함 이름이 파생됩니다.
2. 바인딩은 서비스 비즈니스 예외의 데이터 오브젝트를 사용하여 결함 데이터 핸들러를 호출합니다.

3. 결함 데이터 핸들러는 결함 데이터 오브젝트를 응답 메시지로 변환하여 내보내기 바인딩에 리턴합니다.
4. 내보내기에서는 응답 메시지를 클라이언트에 리턴합니다.

서비스 비즈니스 예외에 결함 이름이 포함되어 있는 경우 결함에 설정된 데이터 핸들러가 호출됩니다. 서비스 비즈니스 예외에 결함의 이름이 포함되지 않은 경우 결함 유형을 일치시켜 결함 이름이 파생됩니다.

런타임 예외

런타임 예외는 비즈니스 결함과 일치하지 않는 요청을 처리하는 동안 SCA 응용프로그램에 발생하는 예외입니다. 비즈니스 결함과는 달리 런타임 예외는 인터페이스에 정의되어 있지 않습니다.

특정 시나리오에서는 클라이언트 응용프로그램이 적절한 조치를 취할 수 있도록 이러한 런타임 예외를 클라이언트 응용프로그램에 전파할 수 있습니다.

예를 들어, 클라이언트가 SCA 컴포넌트에 고객 작성 요청을 전송하고 요청 처리 중에 권한 오류가 발생하는 경우 컴포넌트에는 런타임 예외가 발생합니다. 권한에 대해 적절한 조치를 취할 수 있도록 이 런타임 예외를 호출하는 클라이언트에 다시 전파해야 합니다. 이러한 전파는 내보내기 바인딩에 구성된 런타임 예외 데이터 핸들러를 통해 수행됩니다.

주: HTTP 바인딩에서만 런타임 예외 데이터 핸들러를 구성할 수 있습니다.

런타임 예외 처리는 비즈니스 결함 처리와 비슷합니다. 런타임 예외 데이터 핸들러가 설정된 경우 다음과 같은 처리가 발생합니다.

1. 내보내기 바인딩이 서비스 런타임 예외를 사용하여 적합한 데이터 핸들러를 호출합니다.
2. 데이터 핸들러가 결함 데이터 오브젝트를 응답 메시지로 변환하여 내보내기 바인딩에 리턴합니다.
3. 내보내기에서는 응답 메시지를 클라이언트에 리턴합니다.

결함 처리 및 런타임 예외 처리는 선택적입니다. 호출하는 클라이언트에 결함 또는 런타임 예외를 전파하지 않으려면 결함 데이터 핸들러 또는 런타임 예외 데이터 핸들러를 구성하지 마십시오.

가져오기 바인딩에서 결함이 처리되는 방법

컴포넌트는 모듈 외부의 서비스로 요청을 전송하기 위해 가져오기를 사용합니다. 요청 처리 중에 결함이 발생하면 서비스는 가져오기 바인딩에 결함을 리턴합니다. 가져오기 바인딩을 구성하여 결함을 처리하고 컴포넌트에 리턴하는 방법을 지정합니다.

WebSphere Integration Developer를 사용하여 가져오기 바인딩을 구성합니다. 결함 데이터 핸들러(또는 데이터 바인딩)를 지정할 수 있으며 결함 선택기도 지정합니다.

결함 데이터 핸들러

요청을 처리하는 서비스가 예외 또는 결함 데이터를 포함하는 응답 메시지 형식의 결함 정보를 가져오기 바인딩에 전송합니다.

가져오기 바인딩은 다음 그림 및 다음 절의 설명대로 서비스 예외 또는 응답 메시지를 서비스 비즈니스 예외 또는 서비스 런타임 예외로 변환합니다.

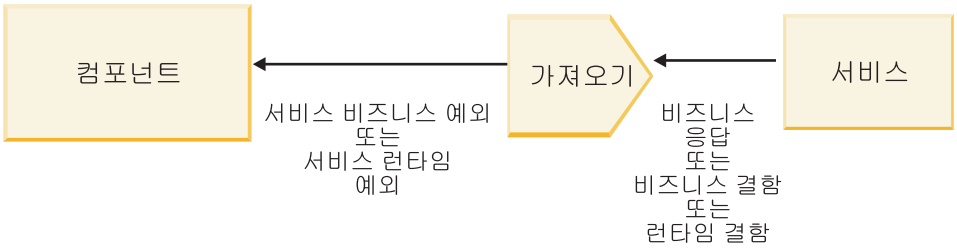


그림 11. 가져오기를 통해 서비스에서 컴포넌트로 결함 정보가 전송되는 방법

사용자 정의 데이터 핸들러 또는 데이터 바인딩을 작성하여 결함을 처리할 수 있습니다.

결함 선택기

가져오기 바인딩을 구성하면 결함 선택기를 지정할 수 있습니다. 결함 선택기는 가져오기 응답이 실제 응답인지, 비즈니스 예외인지 아니면 런타임 결함인지를 판별합니다. 또한 결함 선택기는 응답 본문 또는 헤더에서 기본 결함 이름을 판별하는데, 이 이름은 연관된 인터페이스에서 결함의 이름으로 바인딩 구성에 의해 맵핑됩니다.

사전 패키지화된 두 가지 유형의 결함 선택기가 JMS, MQ JMS, 일반 JMS, WebSphere MQ 및 HTTP 가져오기에 사용 가능합니다.

표 8. 사전 패키지화된 결함 선택기

결함 선택기 유형	설명
헤더 기반	응답 메시지가 비즈니스 결함, 런타임 예외인지, 아니면 수신 응답 메시지의 헤더에 근거한 일반 메시지인지 판별합니다.
SOAP	응답 SOAP 메시지가 일반 응답, 비즈니스 결함 또는 런타임 예외인지 판별합니다.

다음에서는 헤더 기반 결함 선택기와 SOAP 결함 선택기의 예제를 표시합니다.

- 헤더 기반 결함 선택기

수신 메시지가 비즈니스 결함인지 응용프로그램에 표시하려면, 다음과 같이 표시되는 비즈니스 결함의 수신 메시지에 두 개의 헤더가 있어야 합니다.

```
Header name = FaultType, Header value = Business
Header name = FaultName, Header value = <user defined native fault name>
```

수신 응답 메시지가 런타임 예외임을 응용프로그램에 표시하려는 경우, 다음과 같이 표시되는 수신 메시지에 하나의 헤더가 있어야 합니다.

```
Header name = FaultType, Header value = Runtime
```

- SOAP 결함 선택기

  비즈니스 결함은 다음과 같은 SOAP 헤더가 있는 SOAP 메시지의 일부로 전송될 수 있습니다. "CustomerAlreadyExists"는 이 경우 결함 이름입니다.

```
<ibmSoap:BusinessFaultName
xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
<ibmSoap:BusinessFaultName>
```

결함 선택기는 선택적입니다. 결함 선택기를 지정하지 않으면 가져오기 바인딩은 응답 유형을 판별할 수 없습니다. 그러므로 바인딩은 응답 유형을 비즈니스 응답으로 처리하며 응답 데이터 핸들러 또는 데이터 바인딩을 호출합니다.

사용자 정의 결함 선택기를 작성할 수 있습니다. 사용자 정의 결함 선택기 작성을 위한 단계는 WebSphere Integration Developer Information Center의 『사용자 정의 결함 선택기 개발』 주제에서 제공됩니다.

비즈니스 결함

비즈니스 결함은 요청 처리 시 오류가 있는 경우 발생할 수 있습니다. 예를 들어, 고객 작성 요청을 전송했을 때 해당 고객이 이미 있는 경우 서비스는 가져오기 바인딩에 비즈니스 예외를 전송합니다.

바인딩에 의해 비즈니스 예외가 수신되는 경우 해당 바인딩에 대해 결함 선택기가 설정되어 있는지 여부에 따라 처리 단계가 달라집니다.

- 결함 선택기가 설정되지 않은 경우 바인딩은 응답 데이터 핸들러나 데이터 바인딩을 호출합니다.
- 결함 선택기가 설정된 경우 다음 처리가 발생합니다.
  1. 가져오기 바인딩은 결함 선택기를 호출하여 응답이 비즈니스 결함, 비즈니스 응답 또는 런타임 결함인지 여부를 판별합니다.
  2. 응답이 비즈니스 결함이면, 가져오기 바인딩은 결함 선택기를 호출하여 기본 결함 이름을 제공합니다.
  3. 가져오기 바인딩은 결함 선택기가 리턴하는 기본 결함 이름에 해당하는 WSDL 결함을 판별합니다.

4. 가져오기 바인딩은 이 WSDL 결함을 위해 구성되는 결함 데이터 핸들러를 판별합니다.
5. 가져오기 바인딩은 결함 데이터를 사용하여 이 결함 데이터를 호출합니다.
6. 결함 데이터 핸들러가 결함 데이터를 데이터 오브젝트로 변환하여 가져오기 바인딩에 리턴합니다.
7. 가져오기 바인딩이 데이터 오브젝트 및 결함 이름을 사용하여 서비스 비즈니스 예외 오브젝트를 구성합니다.
8. 가져오기가 서비스 비즈니스 예외 오브젝트를 컴포넌트에 리턴합니다.

런타임 예외

런타임 예외는 서비스와의 통신에 문제가 있는 경우 발생할 수 있습니다. 런타임 예외의 처리는 비즈니스 예외의 처리와 유사합니다. 결함 선택기가 설정된 경우 다음 처리가 발생합니다.

1. 가져오기 바인딩은 예외 데이터를 이용하여 적절한 런타임 예외 데이터 핸들러를 호출합니다.
2. 런타임 예외 데이터 핸들러는 예외 데이터를 서비스 런타임 예외 오브젝트로 변환하여 가져오기 바인딩으로 리턴합니다.
3. 가져오기가 서비스 런타임 예외 오브젝트를 컴포넌트에 리턴합니다.

SCA 모듈 및 공개 SCA 서비스 사이의 상호운영성

IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture(SCA)는 공개 SCA 스펙을 기초로 응용프로그램을 구성하기 위한 단순하면서 강력한 프로그래밍 모델을 제공합니다. WebSphere Process Server의 SCA 모듈은 가져오기 및 내보내기 바인딩을 사용하여 Rational® Application Developer 환경에서 개발되고 Service Component Architecture용 WebSphere Application Server 기능 팩에서 호스트되는 공개 SCA 서비스와 상호 운영합니다.

SCA 응용프로그램은 가져오기 바인딩 방식으로 공개 SCA 응용프로그램을 호출합니다. SCA 응용프로그램은 내보내기 바인딩 방식으로 공개 SCA 응용프로그램에서 호출을 수신합니다. 지원되는 바인딩 목록은 30 페이지의 『상호 운영 가능한 바인딩을 통해 서비스 호출』에 있습니다.

SCA 모듈에서 공개 SCA 서비스 호출

WebSphere Integration Developer로 개발된 SCA 응용프로그램은 Rational Application Developer 환경에서 개발된 공개 SCA 응용프로그램을 호출할 수 있습니다. 이 절에서는 SCA 가져오기 바인딩을 사용하여 SCA 모듈에서 공개 SCA 서비스를 호출하는 예제를 제공합니다.

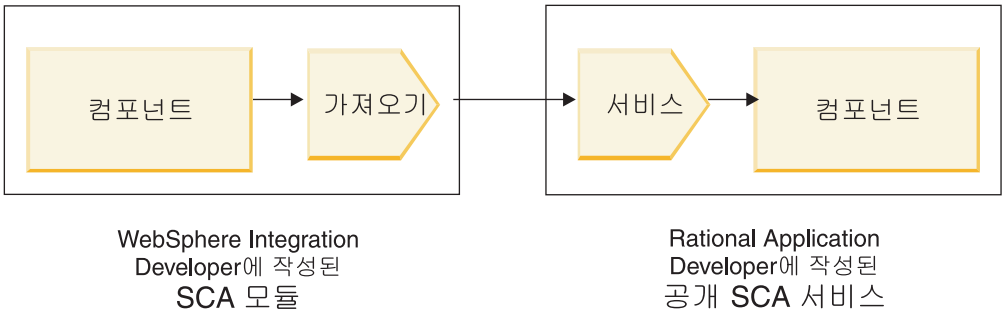


그림 12. 공개 SCA 서비스를 호출하는 SCA 모듈의 컴포넌트

공개 SCA 서비스를 호출하는 데 필요한 특별한 구성은 없습니다.

SCA 가져오기 바인딩 방식으로 공개 SCA 서비스에 연결하기 위해 가져오기 바인딩에서 공개 SCA 서비스의 서비스 이름과 컴포넌트 이름을 제공합니다.

1. 공개 SCA 컴포지트에서 대상 컴포넌트 및 서비스의 이름을 얻으려면 다음 단계를 수행하십시오.
   a. **창 → 보기 표시 → 특성**을 클릭하여 **특성** 탭이 열려 있는지 확인하십시오.
   b. 컴포넌트 및 서비스를 포함하는 복합 다이어그램을 두 번 클릭하여 복합 편집기를 여십시오. 예를 들어, **customer** 컴포넌트의 경우 복합 다이어그램은 **customer.composite_diagram**입니다.
   c. 대상 컴포넌트를 클릭하십시오.
   d. **특성** 탭의 **이름** 필드에서 대상 컴포넌트의 이름을 기록하십시오.
   e. 컴포넌트와 연관되는 서비스 아이콘을 클릭하십시오.
   f. **특성** 탭의 **이름** 필드에서 서비스의 이름을 기록하십시오.
2. 공개 SCA 서비스에 연결하도록 WebSphere Process Server 가져오기를 구성하려면 다음 단계를 수행하십시오.
   a. WebSphere Integration Developer에서, 공개 SCA 서비스에 연결하려는 SCA 가져오기의 **특성** 탭을 탐색하십시오.
   b. **모듈 이름** 필드에서 1d단계의 컴포넌트 이름을 입력하십시오.
   c. **내보내기 이름** 필드에서 1f단계의 서비스 이름을 입력하십시오.
   d. Ctrl+S를 눌러서 작업을 저장하십시오.

공개 SCA 서비스에서 SCA 모듈 호출

Rational Application Developer 환경에서 개발된 공개 SCA 응용프로그램은 WebSphere Integration Developer로 개발된 SCA 응용프로그램을 호출할 수 있습니다. 이 절에서는 공개 SCA 서비스에서 SCA 모듈을 호출하는(SCA 내보내기 바인딩 방식으로) 예제를 제공합니다.

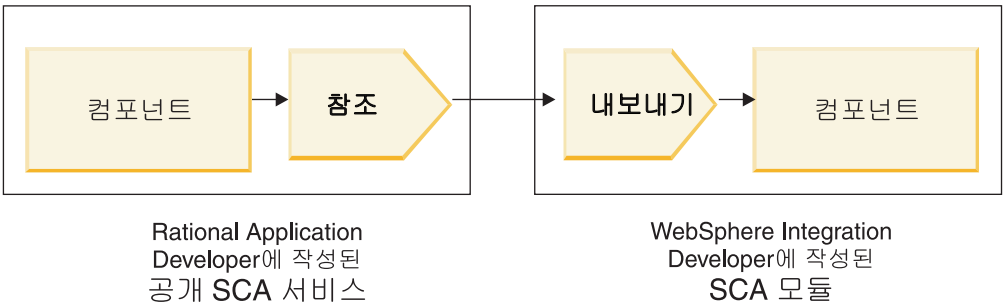


그림 13. SCA 모듈에서 컴포넌트를 호출하는 공개 SCA 서비스

공개 SCA 참조 바인딩 방식으로 SCA 컴포넌트에 연결하기 위해 모듈 이름 및 내보내기 이름을 제공합니다.

1. 대상 모듈 및 내보내기의 이름을 얻으려면 다음 단계를 수행하십시오.
   a. WebSphere Integration Developer의 어셈블리 편집기에서 모듈을 두 번 클릭하여 모듈을 여십시오.
   b. 내보내기를 클릭하십시오.
   c. **특성** 탭의 **이름** 필드에서 내보내기의 이름을 기록하십시오.
2. WebSphere Process Server 모듈 및 내보내기에 연결하려는 공개 SCA 참조를 구성하십시오.
   a. Rational Application Developer에서, 컴포넌트 및 서비스를 포함하는 복합 다이어그램을 두 번 클릭하여 복합 편집기를 여십시오.
   b. 컴포넌트 참조의 참조 아이콘을 클릭하여 **특성** 탭에서 참조 특성을 표시하십시오.
   c. 페이지의 왼쪽에 있는 **바인딩** 탭을 클릭하십시오.
   d. **바인딩**을 클릭한 후 **추가**를 클릭하십시오.
   e. **SCA** 바인딩을 선택하십시오.
   f. **URI** 필드에서 WebSphere Process Server 모듈 이름과 슬래시(『/』), 내보내기 이름(1c 단계에서 판별한)을 차례로 입력하십시오.
   g. **확인**을 클릭하십시오.
   h. Ctrl+S를 눌러서 작업을 저장하십시오.

상호 운영 가능한 바인딩을 통해 서비스 호출

다음 바인딩은 공개 SCA 서비스와의 상호운영성에 대해 지원됩니다.

- SCA 바인딩

  WebSphere Process Server SCA 모듈이 SCA 가져오기 바인딩 방식으로 공개 SCA 서비스를 호출하는 경우 다음 호출 스타일이 지원됩니다.

  – 비동기(단방향)

– 동기(요청/응답)

SCA 가져오기 인터페이스 및 공개 SCA 서비스 인터페이스는 WS-I(Web services interoperability) 준수 WSDL 인터페이스를 사용해야 합니다.

SCA 바인딩은 트랜잭션 및 보안 컨텍스트 전파를 지원합니다.

- SOAP1.1/HTTP 또는 SOAP1.2/HTTP 프로토콜이 있는 웹 서비스(JAX-WS) 바인딩

  SCA 가져오기 인터페이스 및 공개 SCA 서비스 인터페이스는 WS-I(Web services interoperability) 준수 WSDL 인터페이스를 사용해야 합니다.

  또한 다음 서비스 품질이 지원됩니다.

  – WSAT(Web Services Atomic Transaction)

  – 웹 서비스 보안

- EJB 바인딩

  Java 인터페이스는 EJB 바인딩이 사용될 때 SCA 모듈 및 공개 SCA 서비스 사이의 상호작용을 정의하기 위해 사용됩니다.

  EJB 바인딩은 트랜잭션 및 보안 컨텍스트 전파를 지원합니다.

- JMS 바인딩

  SCA 가져오기 인터페이스 및 공개 SCA 서비스 인터페이스는 WS-I(Web services interoperability) 준수 WSDL 인터페이스를 사용해야 합니다.

  다음 JMS 프로바이더가 지원됩니다.

  – WebSphere Platform Messaging(JMS 바인딩)

  – WebSphere MQ(MQ JMS 바인딩)

주: 비즈니스 그래프는 SCA 바인딩 전체에서 상호 운영이 가능하지 않으므로, Service Component Architecture용 WebSphere Application Server 기능 팩과 상호 운영하는 데 사용되는 인터페이스에서는 지원되지 않습니다.

바인딩 유형

가져오기 및 내보내기와 함께 프로토콜 특정 *바인딩*을 사용하여 데이터와 모듈 간 전송을 수행하는 수단을 지정합니다.

적절한 바인딩 선택

사용자 응용프로그램의 필요사항에 맞추기 위해 사용할 수 있는 다양한 바인딩이 있습니다.

WebSphere Integration Developer에서 사용 가능한 바인딩은 선택 범위를 제공합니다. 다음 목록은 한 유형의 바인딩이 응용프로그램의 필요사항에 보다 적합한 경우를 식별하는 데 도움이 됩니다.

다음 요소를 적용 가능한 경우 *SCA* 바인딩을 고려하십시오.

- 모든 서비스가 WebSphere Integration Developer 모듈에 포함되어 있습니다. 즉, 외부 서비스가 없습니다.
- 서로 직접 상호작용하는 여러 SCA 모듈로 기능을 구분하려고 합니다.
- 모듈이 단단하게 결합되어 있습니다.

다음 요소를 적용 가능한 경우 *웹 서비스* 바인딩을 고려하십시오.

- 인터넷을 통해 외부 서버에 액세스하거나 인터넷을 통해 서비스를 제공해야 합니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 동기 통신이 선호됩니다. 즉, 하나의 서비스의 요청이 다른 서비스의 응답을 기다릴 수 있습니다.
- 액세스 중인 외부 서비스의 프로토콜이나 제공하려는 서비스가 SOAP/HTTP 또는 SOAP/JMS입니다.

다음 요소를 적용 가능한 경우 *HTTP* 바인딩을 고려하십시오.

- 인터넷을 통해 외부 서비스에 액세스하거나 인터넷을 통해 서비스를 제공해야 하며 HTTP 모델을 기반으로 하는 다른 웹 서비스(즉, GET, PUT, DELETE 등과 같은 잘 알려진 HTTP 인터페이스 조작을 사용하는 서비스)에 대해 작업 중입니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 동기 통신이 선호됩니다. 즉, 하나의 서비스의 요청이 다른 서비스의 응답을 기다릴 수 있습니다.

다음 요소를 적용 가능한 경우 *EJB* 바인딩을 고려하십시오.

- 바인딩이 EJB 자체이거나 EJB 클라이언트가 액세스해야 하는 가져온 서비스에 대한 바인딩입니다.
- 가져온 서비스가 느슨하게 결합되어 있습니다.
- Stateful EJB 상호작용이 필요하지 않습니다.
- 동기 통신이 선호됩니다. 즉, 하나의 서비스의 요청이 다른 서비스의 응답을 기다릴 수 있습니다.

다음 요소를 적용 가능한 경우 *EIS* 바인딩을 고려하십시오.

- 자원 어댑터를 사용하여 EIS 시스템의 서비스에 액세스해야 합니다.
- 동기 데이터 전송을 비동기보다 선호합니다.

다음 요소를 적용 가능한 경우 *JMS* 바인딩을 고려하십시오.

주: 몇 가지 유형의 JMS 바인딩이 있습니다. JMS를 사용하여 SOAP 메시지를 교환할 것을 예상하면 SOAP/JMS 프로파일이 사용되는 웹 서비스 바인딩을 고려하십시오. 34 페이지의 『웹 서비스 바인딩』의 내용을 참조하십시오.

- 메시징 시스템에 액세스해야 합니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 비동기 데이터 전송을 동기보다 선호합니다.

다음 요소를 적용 가능한 경우 *일반 JMS* 바인딩을 고려하십시오.

- 비IBM 벤더 메시징 시스템에 액세스해야 합니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 성능보다 신뢰도가 더 중요합니다. 즉, 비동기 데이터 전송을 동기보다 선호합니다.

다음 요소를 적용 가능한 경우 *MQ* 바인딩을 고려하십시오.

- WebSphere MQ 메시징 시스템에 액세스해야 하고 MQ 기본 함수를 사용해야 합니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 성능보다 신뢰도가 더 중요합니다. 즉, 비동기 데이터 전송을 동기보다 선호합니다.

다음 요소를 적용 가능한 경우 *MQ JMS* 바인딩을 고려하십시오.

- WebSphere MQ 메시징 시스템에 액세스해야 하지만 JMS 컨텍스트 내에서 액세스할 수 있습니다. 즉, 함수의 JMS 서브세트가 응용프로그램에 충분합니다.
- 서비스가 느슨하게 결합되어 있습니다.
- 성능보다 신뢰도가 더 중요합니다. 즉, 비동기 데이터 전송을 동기보다 선호합니다.

SCA 바인딩

SCA(Service Component Architecture) 바인딩을 사용하면 서비스가 다른 모듈에 있는 다른 서비스와 통신할 수 있습니다. SCA 바인딩이 있는 가져오기를 사용하면 다른 SCA 모듈에 있는 서비스에 액세스할 수 있습니다. SCA 바인딩이 있는 내보내기를 사용하면 다른 모듈에 서비스를 제공할 수 있습니다.

WebSphere Integration Developer를 사용하여 SCA 모듈에서 가져오기 및 내보내기에 대한 SCA 바인딩을 생성하고 구성합니다.

모듈이 동일한 서버에서 실행 중이거나 동일한 클러스터에서 전개되는 경우 SCA 바인딩은 사용하기에 가장 쉽고 빠른 바인딩입니다.

SCA 바인딩이 포함된 모듈이 서버에 전개되면 관리 콘솔을 사용하여 바인딩에 대한 정보를 보거나, 가져오기 바인딩의 경우 바인딩의 선택된 특성을 변경할 수 있습니다.

웹 서비스 바인딩

웹 서비스 바인딩은 SCA(Service Component Architecture) 컴포넌트에서 웹 서비스로(및 그 반대로) 메시지를 전송하는 수단입니다.

웹 서비스 바인딩 개요

웹 서비스 가져오기 바인딩을 사용하면 SCA(Service Component Architecture) 컴포넌트에서 외부 웹 서비스를 호출할 수 있습니다. 웹 서비스 내보내기 바인딩을 사용하면 SCA 컴포넌트를 클라이언트에 웹 서비스로 표시할 수 있습니다.

웹 서비스 바인딩을 사용하는 경우 상호 운영이 가능한 SOAP 메시지 및 서비스 품질(QoS)을 사용하여 외부 서비스에 액세스합니다.

WebSphere Integration Developer를 사용하여 SCA 모듈에서 가져오기 및 내보내기에 대한 웹 서비스 바인딩을 생성하고 구성합니다. 다음 유형의 웹 서비스 바인딩을 사용할 수 있습니다.

- SOAP1.2/HTTP 및 SOAP1.1/HTTP

  이러한 바인딩은 웹 서비스 작성을 위한 Java 프로그래밍 API인 JAX-WS(Java API for XML Web Services)에 기반을 둡니다.

  – 웹 서비스가 SOAP 1.2 스펙을 준수하는 경우 SOAP1.2/HTTP를 사용하십시오.
  – 웹 서비스가 SOAP 1.1 스펙을 준수하는 경우 SOAP1.1/HTTP를 사용하십시오.

  이러한 바인딩 중 하나를 선택하는 경우 SOAP 메시지로 첨부를 전송할 수 있습니다.

  웹 서비스 바인딩은 표준 SOAP 메시지에 대해 작업합니다. 그러나 웹 서비스 JAX-WS 바인딩 중 하나를 사용하면 SOAP 메시지가 구문 분석 또는 작성되는 방법을 사용자 정의할 수 있습니다. 예를 들어, SOAP 메시지의 비표준 요소를 처리하거나 추가 처리를 SOAP 메시지에 적용할 수 있습니다. 바인딩을 구성하는 경우 SOAP 메시지에서 이 처리를 수행하는 사용자 정의 데이터 핸들러를 지정합니다.

- SOAP1.1/HTTP

  JAX-RPC(Java API for XML-based RPC)를 기반으로 하는 SOAP 인코드 메시지를 사용하는 웹 서비스를 작성하려는 경우 이 바인딩을 사용하십시오.

- SOAP1.1/JMS

  JMS(Java Message Service) 대상을 사용하여 SOAP 메시지를 전송하거나 수신하려면 이 바인딩을 사용하십시오.

SOAP 메시지를 전달하는 데 사용된 전송(HTTP 또는 JMS)과 상관 없이 웹 서비스 바인딩은 항상 요청/응답 상호 작용을 동기적으로 처리합니다. 서비스 프로바이더에 호출을 작성하는 스레드는 프로바이더에서 응답을 수신할 때까지 블록화됩니다. 이 호출 스타일에 대한 자세한 정보는 『동기 호출』을 참조하십시오.

중요사항: 웹 서비스 바인딩의 다음 조합은 동일한 모듈의 내보내기에서 사용할 수 없습니다. 하나 이상의 이러한 내보내기 바인딩을 사용해서 컴포넌트를 표시해야 할 경우 독립 모듈에 각 컴포넌트가 있어야 하고 그런 다음 SCA 바인딩을 사용해서 해당 모듈을 컴포넌트에 연결해야 합니다.

- JAX-RPC를 사용하는 SOAP 1.1/JMS 및 SOAP 1.1/HTTP
- JAX-RPC를 사용하는 SOAP 1.1/HTTP 및 JAX-WS를 사용하는 SOAP 1.1/HTTP
- JAX-RPC를 사용하는 SOAP 1.1/HTTP 및 JAX-WS를 사용하는 SOAP 1.2/HTTP

웹 서비스 바인딩이 포함된 SCA 모듈이 서버에 전개된 후 관리 콘솔을 사용하여 바인딩에 대한 정보를 보고 바인딩의 선택된 특성을 변경할 수 있습니다.

주: 웹 서비스를 사용하면 응용프로그램은 변경한 메시지의 표준 형식 및 서비스의 표준 설명을 사용해서 상호 운영할 수 있습니다. 예를 들어, 웹 서비스 가져오기 및 내보내기 바인딩은 Microsoft® .NET용 WSE(Web Services Enhancements) 버전 3.5 및 WCF(Windows® Communication Foundation) 버전 3.5를 사용하여 구현되는 서비스와 상호 운영할 수 있습니다. 이러한 서비스와 상호 운영하는 경우 다음 사항을 확인해야 합니다.

- 웹 서비스 내보내기에 액세스하는 데 사용되는 WSDL(Web Services Description Language) 파일에는 인터페이스에서 각 조작에 사용할 비어 있지 않은 SOAP 조치 값이 포함됩니다.
- 웹 서비스 클라이언트는 웹 서비스 내보내기로 메시지 전송 시 SOAPAction 헤더 또는 wsa:Action 헤더를 설정합니다.

SOAP 헤더 전파

SOAP 메시지를 처리하는 경우, 수신한 메시지에서 특정 SOAP 헤더의 정보에 액세스해야 하며 SOAP 헤더가 있는 메시지가 특정 값과 함께 전송되었는지 확인해야 합니다. 또는 SOAP 헤더가 모듈을 통해 전달되도록 허용해야 합니다.

WebSphere Integration Developer에서 웹 서비스 바인딩을 구성하는 경우 SOAP 헤더가 전파되기를 원한다고 표시할 수 있습니다.

- 내보내기에서 요청을 수신하거나 가져오기에서 응답을 수신한 경우 SOAP 헤더 정보에 액세스할 수 있으므로 모듈의 논리는 헤더 값을 기반으로 할 수 있으며 이러한 헤더를 수정할 수 있습니다.
- 내보내기에서 요청을 전송하거나 가져오기에서 응답을 전송한 경우 SOAP 헤더 정보에는 해당 메시지가 포함될 수 있습니다.

전파된 SOAP 헤더의 양식 및 사용 여부는 37 페이지의 표 9의 설명과 같이 가져오기 또는 내보내기에 구성된 정책 세트의 영향을 받을 수 있습니다.

가져오기 또는 내보내기의 SOAP 헤더 전파를 구성하려면 WebSphere Integration Developer의 특성 보기에서 **프로토콜 헤더 전파** 탭을 선택하고 필요한 옵션을 선택하십시오.

WS-Addressing 헤더

WS-Addressing 헤더는 웹 서비스(JAX-WS) 바인딩에 의해 전파될 수 있습니다.

WS-Addressing 헤더를 전파하는 경우, 다음 정보를 파악하고 있어야 합니다.

- WS-Addressing 헤더를 전파하는 경우 헤더는 다음 상황의 모듈로 전파됩니다.
  - 내보내기에서 요청을 수신하는 경우
  - 가져오기에서 응답을 수신하는 경우
- WS-Addressing 헤더가 WebSphere Process Server에서 아웃바운드 메시지로 전파되지 않는 경우(즉 가져오기에서 요청을 전송하거나 내보내기에서 응답을 전송할 때 헤더가 전파되지 않는 경우)

WS-Security 헤더

WS-Security 헤더는 웹 서비스(JAX-WS) 바인딩과 웹 서비스(JAX-RPC) 바인딩 둘 다를 사용하여 전파될 수 있습니다.

웹 서비스 WS-Security 스펙은 메시지 무결성, 메시지 기밀성 및 단일 메시지 인증을 통해 보호되는 품질을 제공하기 위해 SOAP 메시징의 개선사항에 대해 설명합니다. 이 메커니즘을 사용하여 다양한 보안 모델 및 암호화 기술을 수용할 수 있습니다.

WS-Security 헤더를 전파하는 경우, 다음 정보를 알아야 합니다.

- WS-Security 헤더에 대해 전파를 사용하는 경우 헤더는 다음 상황의 모듈을 통해 전파됩니다.
  - 내보내기에서 요청을 수신하는 경우
  - 가져오기에서 요청을 전송하는 경우
  - 가져오기에서 응답을 수신하는 경우
- 내보내기에서 응답을 전송할 때 헤더는 기본적으로 전파되지 *않습니다*. 그러나 JVM 특성 `WSSECURITY.ECHO.ENABLED`를 `true`로 설정하면 내보내기에서 응답을 전송할 때 헤더가 전파됩니다. 이 경우 요청 경로에 있는 WS-Security 헤더가 수정되지 않으면 WS-Security 헤더는 요청에서 응답으로 자동으로 에코됩니다.
- 요청 가져오기에서 또는 응답 내보내기에서 전송된 SOAP 메시지의 정확한 형식은 원래 수신한 SOAP 메시지와 정확히 일치하지 않을 수 있습니다. 따라서 디지털 서명이 유효하지 않은 것으로 간주될 수 있습니다. 전송하는 메시지에 디지털 서명이

필요한 경우 적절한 보안 정책 세트를 사용하여 설정되어야 하며 수신 메시지에서 디지털 서명과 관련된 WS-Security 헤더는 모듈에서 제거되어야 합니다.

WS-Security 헤더를 전파하려면 응용프로그램 모듈과 함께 WS-Security 스키마를 포함시켜야 합니다. 스키마를 포함시키는 프로시저에 대해서는 38 페이지의 『응용프로그램 모듈에 WS-Security 스키마 포함』을 참조하십시오.

헤더 전파 방법

헤더가 전파되는 방법은 표 9에서와 같이 가져오기 또는 내보내기 바인딩의 보안 정책 설정에 따라 다릅니다.

표 9. 보안 헤더 전달 방법

	보안 정책이 없는 내보내기 바인딩	**보안 정책이 있는 내보내기 바인딩**
보안 정책이 없는 가져오기 바인딩	보안 헤더는 모듈을 통해 그대로 전달됩니다. 암호가 해독되지 않습니다. 헤더는 수신한 것과 동일한 형식으로 아웃바운드 전송됩니다. 디지털 서명은 유효하지 않을 수 있습니다.	모듈에 서명 확인 및 인증이 있는 경우에도 보안 헤더는 암호 해독되어 전달됩니다. 암호 해독된 헤더가 아웃바운드 전송됩니다. 디지털 서명은 유효하지 않을 수 있습니다.
보안 정책이 있는 가져오기 바인딩	보안 헤더는 모듈을 통해 그대로 전달됩니다. 암호가 해독되지 않습니다. 헤더는 가져오기로 전파되지 않아야 합니다. 그렇지 않으면 중복 때문에 오류가 발생합니다.	모듈에 서명 확인 및 인증이 있는 경우에도 보안 헤더는 암호 해독되어 전달됩니다. 헤더는 가져오기로 전파되지 않아야 합니다. 그렇지 않으면 중복 때문에 오류가 발생합니다.

이는 서비스 요청자를 서비스 프로바이더의 구성 또는 QoS 요구사항에 대한 변경사항과 분리하므로 내보내기 및 가져오기 바인딩에 적절한 정책 세트를 구성하십시오. 표준 SOAP 헤더를 모듈에 표시되게 하면 이를 사용하여 모듈에서의 처리(예: 로깅 및 추적)에 영향을 미칠 수 있습니다. 한 모듈의 수신 메시지에서 전송 메시지로 SOAP 헤더를 전파하면 모듈을 분리하는 이점이 줄어듭니다.

가져오기 또는 내보내기에 연관된 정책 세트가 있는 경우, 일반적으로 해당 헤더가 생성되므로 요청에서 가져오기로 또는 응답에서 내보내기로 표준 헤더(예: WS-Security 헤더)가 전파되지 않아야 합니다. 그렇지 않으면 헤더가 중복되어 오류가 발생합니다. 대신 헤더를 명시적으로 제거하거나 프로토콜 헤더의 전파를 방지하도록 가져오기 또는 내보내기 바인딩을 구성해야 합니다.

SOAP 헤더 액세스

웹 서비스 가져오기 또는 내보내기에서 SOAP 헤더가 있는 메시지를 수신하는 경우, 헤더는 SMO(Service Message Object)의 헤더 섹션에 저장됩니다. 『SMO의 SOAP 헤

더 정보에 액세스』의 설명대로 헤더 정보에 액세스할 수 있습니다.

응용프로그램 모듈에 WS-Security 스키마 포함

다음 프로시저는 응용프로그램 모듈에 스키마를 포함하는 단계의 개요를 설명합니다.

- WebSphere Integration Developer를 실행 중인 컴퓨터에서 인터넷에 액세스할 수 있는 경우 다음 단계를 수행하십시오.
  1. 비즈니스 통합 Perspective에서 프로젝트에 대해 **종속성**을 선택하십시오.
  2. **사전 정의된 자원**을 펼치고 **WS-Security 1.0 스키마 파일** 또는 **WS-Security 1.1 스키마 파일**을 선택하여 해당 스키마를 사용자 모듈로 가져오십시오.
  3. 프로젝트를 정리하고 다시 빌드하십시오.
- WebSphere Integration Developer가 실행 중인 컴퓨터에 인터넷 액세스가 없는 경우에는 인터넷 액세스가 있는 다른 컴퓨터로 스키마를 다운로드할 수 있습니다. 그런 다음 WebSphere Integration Developer를 실행 중인 컴퓨터로 복사할 수 있습니다.
  1. 인터넷 액세스가 있는 컴퓨터에서 원격 스키마를 다운로드하십시오.
     a. **파일 → 가져오기 → 비즈니스 통합 → WSDL 및 XSD**를 클릭하십시오.
     b. **원격 WSDL** 또는 **XSD 파일**을 선택하십시오.
     c. 다음 스키마를 가져오십시오.

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. 스키마를 인터넷 액세스가 없는 컴퓨터로 복사하십시오.
  3. 인터넷 액세스가 없는 컴퓨터에서 스키마를 가져오십시오.
     a. **파일 → 가져오기 → 비즈니스 통합 → WSDL 및 XSD**를 클릭하십시오.
     b. **로컬 WSDL** 또는 **XSD 파일**을 선택하십시오.
  4. oasis-wss-wssecurity_secext-1.1.xsd의 스키마 위치를 변경하십시오.
     a. *workplace_location*/*module_name*/StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd에 있는 스키마를 여십시오.
     b. 다음 코드를 찾으십시오.

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
        ```

        다음과 같이 변경하십시오.

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
        ```

     c. 다음 코드를 찾으십시오.

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
```

다음과 같이 변경하십시오.

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
```

5. oasis-200401-wss-wssecurity-secext-1.0.xsd의 스키마 위치를 변경하십시오.
   a. *workplace_location*/*module_name*/StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd에 있는 스키마를 여십시오.
   b. 다음 코드를 찾으십시오.

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
```

다음과 같이 변경하십시오.

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
```

6. 프로젝트를 정리하고 다시 빌드하십시오.

SOAP 메시지의 첨부

2진 데이터(예: PDF 파일 또는 JPEG 이미지)를 첨부로 포함하는 SOAP 메시지를 전송 및 수신할 수 있습니다. 첨부는 *참조*(즉, 명시적으로 서비스 인터페이스에 있는 메시지 파트로 표시됨)되거나 *참조 해제*(개수 및 유형에 관계없이 첨부가 포함될 수 있음)될 수 있습니다.

참조된 첨부는 다음 방식 중 하나로 표시될 수 있습니다.

- 메시지 스키마에서 wsi:swaRef-typed 요소로 표시

  wsi:swaRef 유형을 사용하여 정의된 첨부는 메시지 요소가 MIME 파트에 관련되는 방법을 정의하는 WS-I(Web Services Interoperability Organization) *Attachments Profile Version 1.0*(http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html)을 준수합니다.

- 2진 스키마 유형을 사용하여, 최상위 레벨 메시지 파트로 표시

  최상위 레벨 메시지 파트로 표시되는 첨부는 *SOAP Messages with Attachments* (http://www.w3.org/TR/SOAP-attachments) 스펙을 준수합니다.

참조되지 않는 첨부는 메시지 스키마에 표시하지 않고 SOAP 메시지로 전달됩니다.

모든 경우에서, WSDL SOAP 바인딩에는 반드시 사용될 첨부에 대한 MIME 바인딩이 포함되어야 합니다. 그리고 첨부의 최대 크기는 20MB를 초과할 수 없습니다.

주: 첨부가 있는 SOAP 메시지를 전송 또는 수신하려면 JAX-WS(Java API for XML Web Services)를 기초로 웹 서비스 바인딩 중 하나를 사용해야 합니다.

참조된 첨부: swaRef 유형 요소:

swaRef 유형 요소로 서비스 인터페이스에 표시된 첨부를 포함하는 SOAP 메시지를 전송하고 수신할 수 있습니다.

swaRef 유형 요소는 WS-I(Web Services Interoperability Organization) *Attachments Profile* Version 1.0(http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html)에 정의됩니다. 해당 요소는 메시지 요소가 MIME 파트와 연관되는 방법을 정의합니다.

주: 생성되거나 이용된 SOAP 메시지는 WS-I 첨부 프로파일 준수를 보장하지 않습니다. 특히 WS-I 첨부 프로파일 1.0의 절 3.8에 설명된 것처럼 『컨텐츠 ID 파트 인코딩』은 지원되지 않습니다.

SOAP 메시지에서 SOAP 본문에는 첨부의 컨텐츠 ID를 식별하는 swaRef 유형 요소가 포함됩니다.

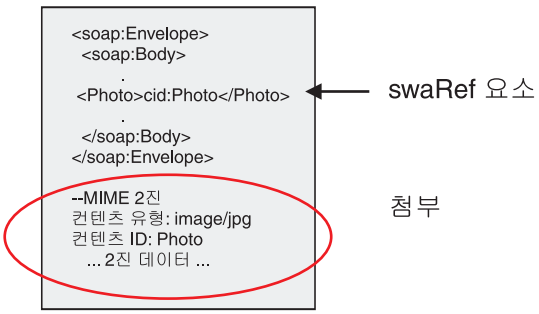


그림 14. swaRef 요소가 있는 SOAP 메시지

SOAP 메시지에 대한 WSDL에는 첨부를 식별하는 메시지 파트 내에 swaRef 유형 요소가 포함됩니다.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

WSDL에는 MIME 여러 부분 메시지가 사용됨을 표시하는 MIME 바인딩도 포함되어야 합니다.

주: MIME 바인딩이 최상위 레벨 메시지 파트에만 적용되기 때문에 WSDL에는 특정 swaRef 유형 메시지 요소에 대한 MIME 바인딩이 포함되지 *않습니다.*

swaRef 유형 요소로 표시된 첨부는 중개 플로우 컴포넌트를 통해서만 전파될 수 있습니다. 다른 컴포넌트 유형이 첨부에 액세스하거나 첨부를 전파해야 하는 경우 중개 플로우 컴포넌트를 사용하여 해당 컴포넌트가 액세스할 수 있는 위치로 첨부를 이동하십시오.

첨부의 인바운드 처리

WebSphere Integration Developer를 사용하여 첨부를 수신할 내보내기 바인딩을 구성합니다. 유형 swaRef의 요소를 포함해서 모듈과 연관된 인터페이스 및 조작을 작성합니다. 그런 다음 웹 서비스(JAX-WS) 바인딩을 작성합니다.

주: 자세한 정보는 WebSphere Integration Developer Information Center의 『첨부 작업』 주제를 참조하십시오.

클라이언트가 swaRef 첨부가 있는 SOAP 메시지를 SCA(Service Component Architecture) 컴포넌트로 전달할 때 웹 서비스(JAX-WS) 내보내기 바인딩은 먼저 해당 첨부를 제거합니다. 그런 다음 메시지의 SOAP 파트를 구문 분석하고 비즈니스 오브젝트를 작성합니다. 마지막으로 바인딩이 비즈니스 오브젝트에 첨부의 컨텐츠 ID를 설정합니다.

그림 15. 웹 서비스(JAX-WS) 내보내기 바인딩의 swaRef 첨부가 있는 SOAP 메시지 처리 방법

중개 플로우 컴포넌트의 첨부 메타데이터 액세스

42 페이지의 그림 16에 표시된 것과 같이 swaRef 첨부가 컴포넌트에 의해 액세스될 때 첨부 컨텐츠 ID가 유형 swaRef의 요소로 나타납니다.

SOAP 메시지의 각 첨부에는 SMO의 해당 첨부 요소가 있습니다. WS-I swaRef 유형을 사용할 때 첨부 요소에는 첨부의 실제 2진 데이터 뿐만 아니라 첨부 컨텐츠 유형 및 컨텐츠 ID가 포함됩니다.

swaRef 첨부 값을 얻으려면 swaRef 유형 요소의 값을 얻은 다음 해당 `contentID` 값으로 첨부 요소를 찾아야 합니다. `contentID` 값에는 일반적으로 swaRef 값에서 제거된 `cid:` 접두부가 있다는 것을 유의하십시오.

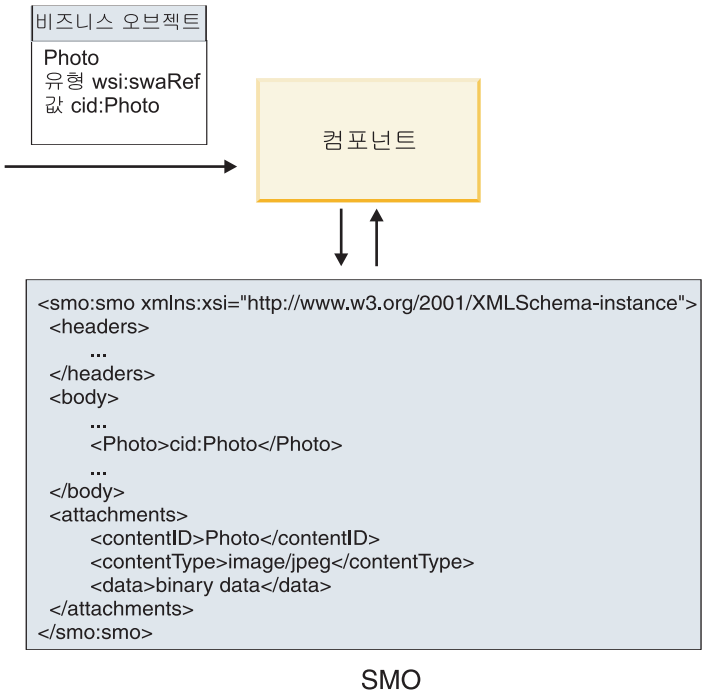


그림 16. swaRef 첨부가 SMO에 나타나는 방법

아웃바운드 처리

WebSphere Integration Developer를 사용하여 외부 웹 서비스를 호출하기 위해 웹 서비스(JAX-WS) 가져오기 바인딩을 구성합니다. 가져오기 바인딩은 호출할 웹 서비스를 설명하고 웹 서비스로 전달되어야 하는 첨부를 정의하는 WSDL 문서로 구성됩니다.

SCA 메시지가 웹 서비스(JAX-WS) 가져오기 바인딩에 의해 수신될 때 가져오기가 중개 플로우 컴포넌트에 연결되고 swaRef 유형 요소에 해당 첨부 요소가 있는 경우 swaRef 유형 요소가 첨부로 전송됩니다.

아웃바운드 처리의 경우 swaRef 유형 요소가 항상 해당 컨텐츠 ID 값으로 전송됩니다. 그러나 중개 모듈은 일치하는 `contentID` 값에 해당하는 첨부 요소가 있는지 확인해야 합니다.

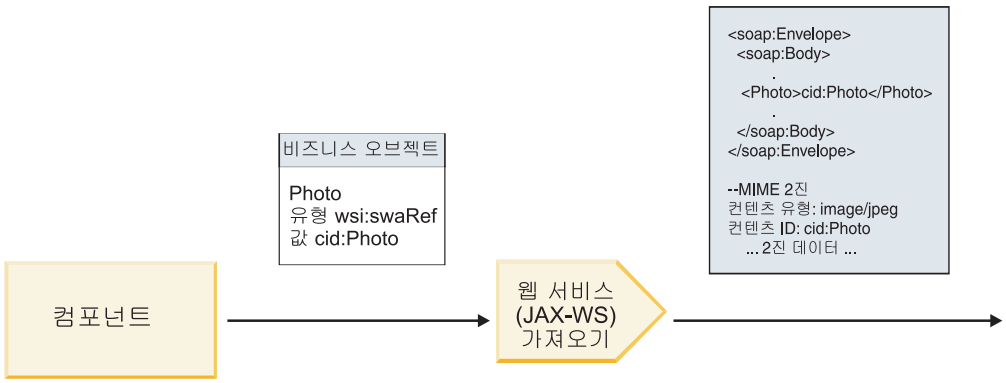


그림 17. 웹 서비스(JAX-WS) 가져오기 바인딩의 swaRef 첨부가 있는 SOAP 메시지 생성 방법

중개 플로우 컴포넌트에 첨부 메타데이터 설정

SMO에서 swaRef 유형 요소 값과 첨부 요소가 있는 경우 바인딩은 SOAP 메시지(첨부가 있는)를 준비하고 이를 받는 사람에게 전송합니다.

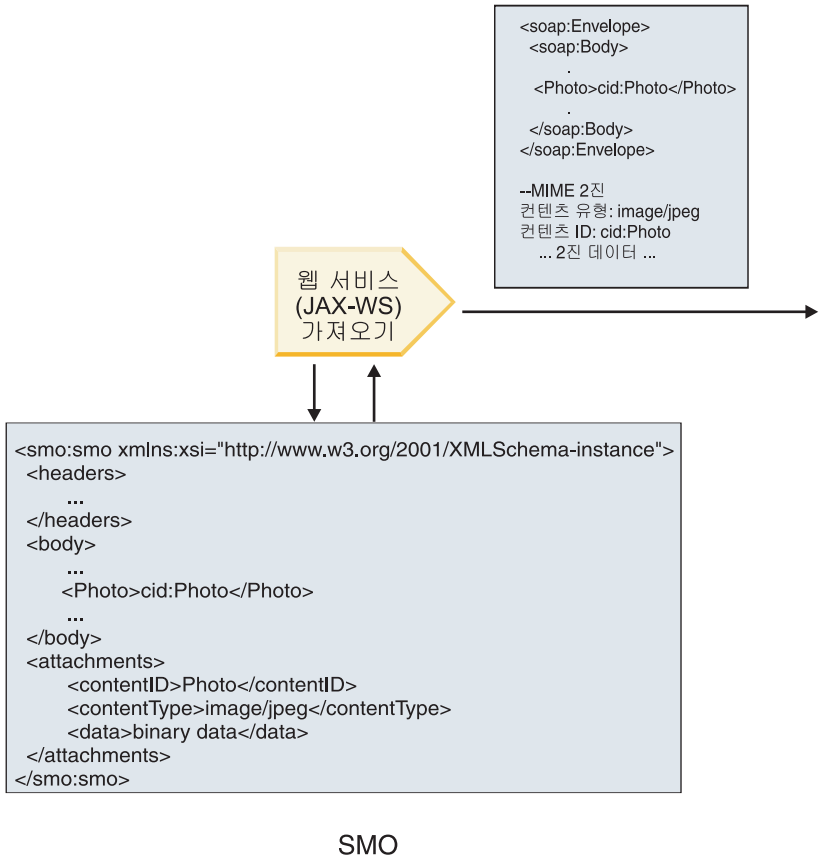


그림 18. SMO의 swaRef 첨부에 SOAP 메시지를 작성하기 위해 액세스하는 방법

중개 플로우 컴포넌트가 가져오기 또는 내보내기에 직접 연결된 경우에만 첨부 요소가 SMO에 존재합니다. 이는 기타 컴포넌트 유형을 통해 전달되지 않습니다. 기타 컴포넌

트 유형을 포함하는 모듈에 값이 필요한 경우 중개 플로우 컴포넌트를 사용하여 모듈에서 값에 액세스할 수 있는 위치로 값을 복사하고 기타 중개 플로우 컴포넌트를 사용하여 웹 서비스 가져오기를 통해 아웃바운드 호출 전에 올바른 값으로 설정합니다.

중요사항: 『SMO의 XML 표시』에 설명된 대로 XSL 변환 중개 기본요소는 XSLT 1.0 변환을 사용하여 메시지를 변환합니다. 변환은 SMO의 XML 직렬화에 대해 작동합니다. XSL 변환 중개 기본요소를 사용하면 직렬화 루트를 지정하고 XML 문서의 루트 요소를 이 루트에 반영할 수 있습니다.

SOAP 메시지를 첨부 파일과 함께 송신할 때 사용자가 선택한 루트 요소는 첨부 파일이 사용되는 방법을 결정합니다.

- 『/body』를 XML 맵의 루트로 사용하는 경우 모든 첨부 파일은 기본적으로 맵에 걸쳐 사용됩니다.
- 『/』를 맵의 루트로 사용하는 경우 첨부 파일의 사용을 제어할 수 있습니다.

참조된 첨부: 최상위 레벨 메시지 파트:

서비스 인터페이스에서 파트로 선언된 2진 첨부를 포함하는 SOAP 메시지를 전송하고 수신할 수 있습니다.

MIME 여러 부분 SOAP 메시지에서 SOAP 본문은 메시지의 첫 번째 파트이고 첨부는 후속 파트에 있습니다. 첨부에 대한 참조는 SOAP 본문에 포함됩니다.

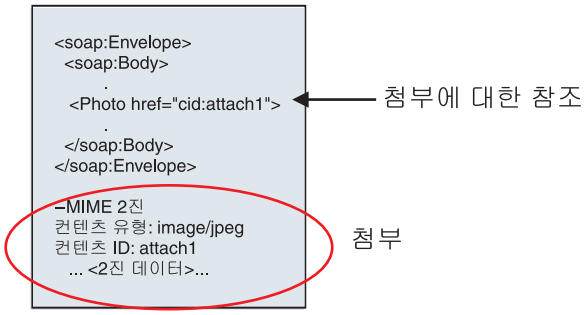


그림 19. 참조된 첨부가 있는 SOAP 메시지

SOAP 메시지에서 참조된 첨부를 전송하고 수신하는 이점은 무엇입니까? 첨부를 구성하는 2진 데이터(종종 상당히 큼)가 XML로 구문 분석될 필요가 없도록 SOAP 메시지 본문에서 독립적으로 보유됩니다. 이는 2진 데이터가 XML 요소 내에 보유되는 경우보다 더 효과적인 처리의 결과를 가져옵니다.

참조된 첨부의 인바운드 처리

WebSphere Integration Developer를 사용하여 내보내기 바인딩을 구성합니다. 모듈과 연관된 인터페이스 및 조작을 작성합니다. 그런 다음 웹 서비스(JAX-WS) 바인딩을 작성합니다. 참조된 첨부 페이지에는 작성된 조작의 모든 2진 파트가 표시되고 첨부될 파트를 선택합니다.

주: 2진 유형(base64Binary 또는 hexBinary)이 있는 최상위 레벨 메시지 파트(즉, 입력 또는 출력 메시지 내의 파트로 WSDL portType에 정의된 요소)만 참조된 첨부로 전송 또는 수신될 수 있습니다.
자세한 정보는 WebSphere Integration Developer Information Center의 『첨부 작업』 주제를 참조하십시오.

클라이언트가 첨부가 있는 SOAP 메시지를 SCA(Service Component Architecture) 컴포넌트로 전달할 때 웹 서비스(JAX-WS) 내보내기 바인딩은 먼저 첨부를 제거합니다. 그런 다음 메시지의 SOAP 파트를 구문 분석하고 비즈니스 오브젝트를 작성합니다. 마지막으로 바인딩이 비즈니스 오브젝트에 첨부 2진을 설정합니다.

그림 20. 웹 서비스(JAX-WS) 내보내기 바인딩의 참조된 첨부가 있는 SOAP 메시지 처리 방법

중개 플로우 컴포넌트의 첨부 메타데이터 액세스

그림 20에 표시된 것과 같이 컴포넌트가 참조된 첨부에 액세스할 때 첨부 데이터가 바이트 배열로 나타납니다.

SOAP 메시지의 각 참조된 첨부에는 SMO의 해당 첨부 요소가 있습니다. 첨부 요소에는 첨부가 보류된 메시지 본문 요소에 대한 경로 및 첨부 컨텐츠 유형이 포함됩니다.

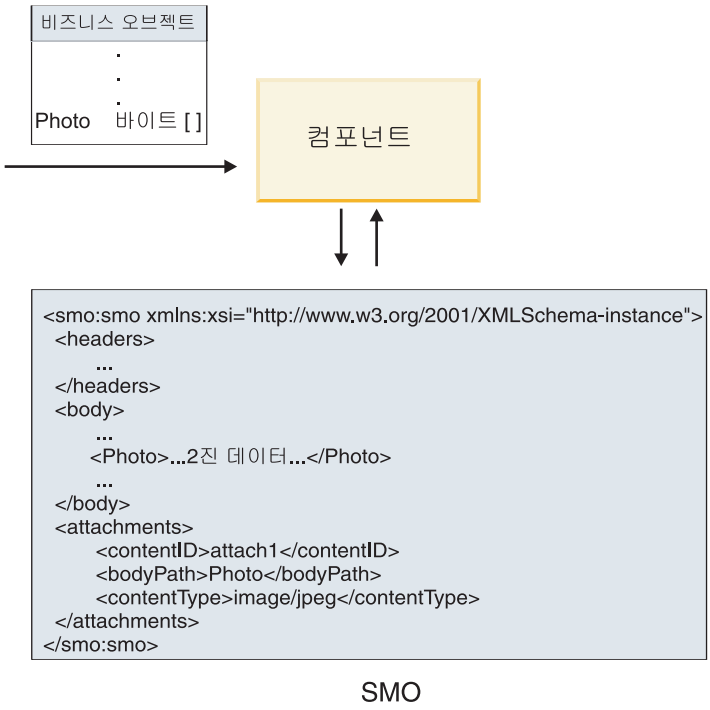


그림 21. SMO에 참조된 첨부가 나타나는 방법

중요사항: 메시지가 변환되고 첨부가 이동한 경우 메시지 본문 요소에 대한 경로는 자동으로 갱신되지 않습니다. 중개 플로우를 사용하여 새 경로가 있는 첨부 요소를 갱신할 수 있습니다(예: 독립 메시지 요소 Setter를 사용하거나 변환의 파트로).

참조된 첨부의 아웃바운드 처리

WebSphere Integration Developer를 사용하여 외부 웹 서비스를 호출하기 위해 웹 서비스(JAX-WS) 가져오기 바인딩을 구성합니다. 가져오기 바인딩은 호출할 웹 서비스를 설명하고 첨부로 전달되어야 하는 메시지 파트를 정의하는 WSDL 문서로 구성됩니다.

주: WSDL에 정의된 것과 같이 첨부를 표시하는 파트는 단순한 유형이어야 합니다(base64Binary 또는 hexBinary). 파트가 complexType에 의해 정의되면 해당 파트가 첨부로 취급되지 않습니다.

가져오기 바인딩은 SMO의 정보를 사용하여 2진 최상위 레벨 메시지 파트를 첨부로 전송할 방법을 판별합니다.

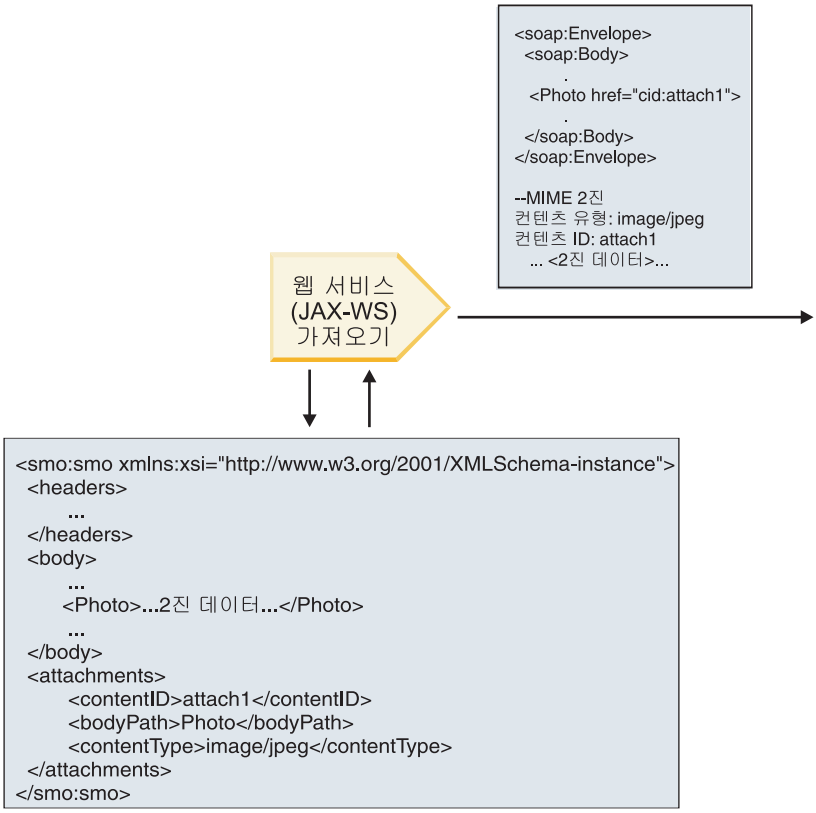


그림 22. SMO의 참조된 첨부에 SOAP 메시지를 작성하기 위해 액세스하는 방법

중개 플로우 컴포넌트가 가져오기 또는 내보내기에 직접 연결된 경우에만 첨부 요소가 SMO에 존재합니다. 이는 기타 컴포넌트 유형을 통해 전달되지 않습니다. 기타 컴포넌트 유형을 포함하는 모듈에 값이 필요한 경우 중개 플로우 컴포넌트를 사용하여 모듈에서 값에 액세스할 수 있는 위치로 값을 복사하고 기타 중개 플로우 컴포넌트를 사용하여 웹 서비스 가져오기를 통해 아웃바운드 호출 전에 올바른 값으로 설정합니다.

바인딩은 다음 조건의 조합을 사용하여 메시지를 전송할 방법(또는 여부)을 판별합니다.

- 최상위 레벨 2진 메시지 파트의 WSDL MIME 바인딩이 있는지 여부 및 있는 경우 컨텐츠 유형이 정의된 방법
- `bodyPath` 값이 최상위 레벨 2진 파트를 참조하는 SMO에 `attachments` 요소가 있는지 여부

SMO에 `attachment` 요소가 있는 경우 첨부 작성 방법

다음 표는 SMO에 메시지 이름 파트와 일치하는 `bodyPath`의 `attachment` 요소가 있는 경우 첨부를 작성하고 전송하는 방법을 보여줍니다.

표 10. 첨부 생성 방법

최상위 레벨 2진 메시지 파트의 WSDL MIME 바인딩 상태	메시지 작성 및 전송 방법
다음 중 하나와 함께 있음 • 메시지 파트에 대해 정의된 컨텐츠 유형이 없음 • 다중 컨텐츠 유형이 정의됨 • 와일드카드 컨텐츠 유형이 정의됨	메시지 파트가 첨부로 전송됩니다. Content-Id는 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 생성됩니다. Content-Type은 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 application/octet-stream으로 설정됩니다.
메시지 파트에 와일드카드가 없는 단일 컨텐츠로 제공	메시지 파트가 첨부로 전송됩니다. Content-Id는 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 생성됩니다. Content-Type은 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 WSDL MIME 컨텐츠 요소에 정의된 유형으로 설정됩니다.
없음	메시지 파트가 첨부로 전송됩니다. Content-Id는 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 생성됩니다. Content-Type은 있는 경우 첨부 요소의 값으로 설정됩니다. 없는 경우에는 application/octet-stream으로 설정됩니다. **주:** WSDL에 첨부로 정의되지 않은 메시지 파트를 첨부로 전송하면 WS-I 첨부 프로파일 1.0 준수가 손상될 수 있으므로 가능하면 피하십시오.

SMO에 attachment 요소가 없는 경우 첨부 작성 방법

다음 표는 SMO에 메시지 이름 파트와 일치하는 bodyPath의 attachment 요소가 없는 경우 첨부를 작성하고 전송하는 방법을 보여줍니다.

표 11. 첨부 생성 방법

최상위 레벨 2진 메시지 파트의 WSDL MIME 바인딩 상태	메시지 작성 및 전송 방법
다음 중 하나와 함께 있음 • 메시지 파트에 대해 정의된 컨텐츠 유형이 없음 • 다중 컨텐츠 유형이 정의됨 • 와일드카드 컨텐츠 유형이 정의됨	메시지 파트가 첨부로 전송됩니다. Content-Id가 생성됩니다. Content-Type이 application/octet-stream으로 설정됩니다.
메시지 파트에 와일드카드가 없는 단일 컨텐츠로 제공	메시지 파트가 첨부로 전송됩니다. Content-Id가 생성됩니다. Content-Type은 WSDL MIME 컨텐츠 요소에 정의된 유형으로 설정됩니다.
없음	메시지 파트가 첨부로 전송되지 않습니다.

중요사항: 『SMO의 XML 표시』에 설명된 대로 XSL 변환 중개 기본요소는 XSLT 1.0 변환을 사용하여 메시지를 변환합니다. 변환은 SMO의 XML 직렬화에 대해 작동합니다. XSL 변환 중개 기본요소를 사용하면 직렬화 루트를 지정하고 XML 문서의 루트 요소를 이 루트에 반영할 수 있습니다.

SOAP 메시지를 첨부 파일과 함께 송신할 때 사용자가 선택한 루트 요소는 첨부 파일이 사용되는 방법을 결정합니다.

- 『/body』를 XML 맵의 루트로 사용하는 경우 모든 첨부 파일은 기본적으로 맵에 걸쳐 사용됩니다.
- 『/』를 맵의 루트로 사용하는 경우 첨부 파일의 사용을 제어할 수 있습니다.

참조되지 않은 첨부:

서비스 인터페이스에 선언되지 않은 *참조되지* 않은 첨부를 전송 및 수신할 수 있습니다.

MIME 여러 부분 SOAP 메시지에서 SOAP 본문은 메시지의 첫 번째 파트이고 첨부는 후속 파트에 있습니다. 첨부에 대한 참조는 SOAP 본문에 포함되어 있지 않습니다.

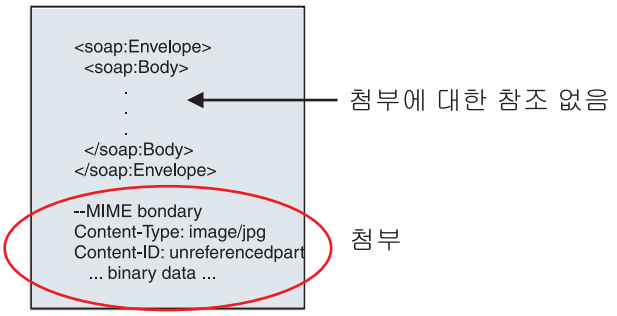


그림 23. 참조되지 않은 첨부가 있는 SOAP 메시지

웹 서비스 내보내기를 통해 참조되지 않은 첨부가 있는 SOAP 메시지를 웹 서비스 가져오기에 전송할 수 있습니다. 대상 웹 서비스에 전송되는 출력 메시지에는 첨부가 포함되어 있습니다.

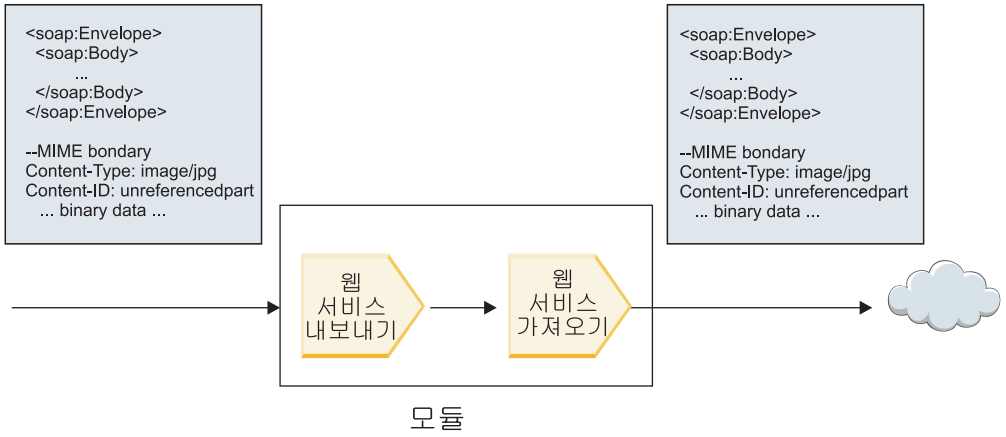


그림 24. SCA 모듈을 통해 전달되는 첨부

그림 24에서는 첨부가 있는 SOAP 메시지가 수정 없이 전달됩니다.

또한 중개 플로우 컴포넌트를 사용하여 SOAP 메시지를 수정할 수 있습니다. 예를 들어, 중개 플로우 컴포넌트를 사용하여 SOAP 메시지에서 데이터(이 경우에는 메시지의 본문에 있는 2진 데이터)를 추출하고 첨부 메시지가 있는 SOAP를 작성할 수 있습니다. 데이터는 SMO(Service Message Object) 첨부 요소의 파트로 처리됩니다.

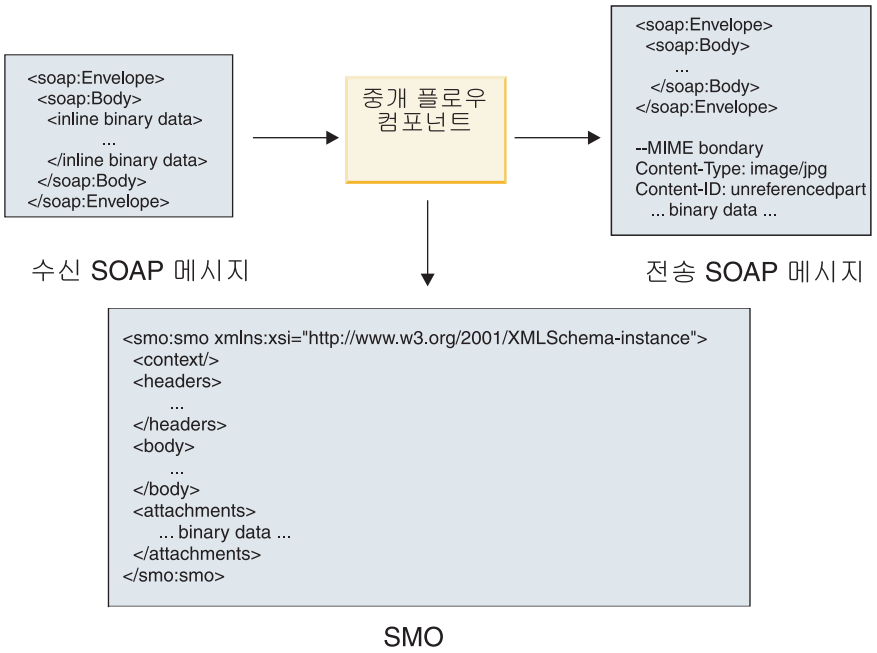


그림 25. 중개 플로우 컴포넌트에서 처리하는 메시지

반대로 중개 플로우 컴포넌트는 첨부를 추출하고 인코딩한 후 첨부가 없는 메시지를 전송하여 수신 메시지를 변환할 수 있습니다.

수신 SOAP 메시지에서 데이터를 추출하여 첨부 메시지가 있는 SOAP를 작성하는 대신 데이터베이스와 같은 외부 소스에서 첨부 데이터를 얻을 수 있습니다.

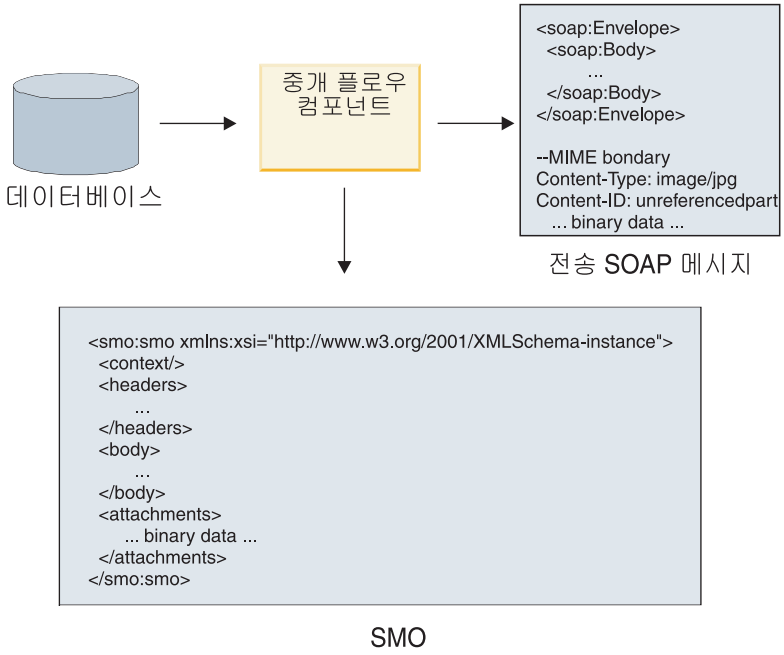


그림 26. 데이터베이스에서 얻어 SOAP 메시지에 추가된 첨부

반대로 중개 플로우 컴포넌트는 수신 SOAP 메시지에서 첨부를 추출하고 메시지를 처리할 수 있습니다(예: 데이터베이스에 첨부 저장).

참조되지 않은 첨부는 중개 플로우 컴포넌트를 통해서만 전파할 수 있습니다. 다른 컴포넌트 유형이 첨부에 액세스하거나 첨부를 전파해야 하는 경우 중개 플로우 컴포넌트를 사용하여 해당 컴포넌트가 액세스할 수 있는 위치로 첨부를 이동하십시오.

중요사항: 『SMO의 XML 표시』에 설명된 대로 XSL 변환 중개 기본요소는 XSLT 1.0 변환을 사용하여 메시지를 변환합니다. 변환은 SMO의 XML 직렬화에 대해 작동합니다. XSL 변환 중개 기본요소를 사용하면 직렬화 루트를 지정하고 XML 문서의 루트 요소를 이 루트에 반영할 수 있습니다.

SOAP 메시지를 첨부 파일과 함께 송신할 때 사용자가 선택한 루트 요소는 첨부 파일이 사용되는 방법을 결정합니다.

- 『/body』를 XML 맵의 루트로 사용하는 경우 모든 첨부 파일은 기본적으로 맵에 걸쳐 사용됩니다.
- 『/』를 맵의 루트로 사용하는 경우 첨부 파일의 사용을 제어할 수 있습니다.

멀티파트 메시지와 함께 WSDL 문서 스타일 바인딩 사용

WS-I(Web Services Interoperability Organization) 조직이 상호운영성을 위해 WSDL 방식으로 웹 서비스를 설명해야 하는 방법과 해당되는 SOAP 메시지를 형성해야 하는 방법에 대한 규칙 세트를 정의했습니다.

이 규칙은 WS-I *Basic Profile Version 1.1*(http://www.ws-i.org/Profiles/BasicProfile-1.1.html)에 지정되어 있습니다.

특히, 문서 스타일 SOAP 바인딩의 경우 WS-I 프로파일을 준수하려면 WSDL 문서에서, 문서 스타일을 사용하는 조작에 대해 단일 메시지 파트만 SOAP 본문에 바인드되고 이에 해당되는 SOAP 메시지에 바인드된 파트와 일치하는 단일 하위 요소가 포함되어야 합니다.

이는 메시지(입력, 출력 또는 결함)가 여러 파트에 대해 정의된 조작에 대해 문서 스타일 SOAP 바인딩을 사용하는 경우 WS-I 기본 프로파일 1.1을 준수하려면 해당 파트 중 하나만 SOAP 본문에 바인드되어야 합니다.

다음 접근방식은 이러한 경우에 웹 서비스(JAX-WS 및 JAX-RPC) 바인딩이 있는 내보내기에 대해 WSDL 설명이 생성될 때 사용됩니다.

- 첫 번째 메시지 파트가 SOAP 본문에 바인드됩니다.
- JAX-WS 바인딩의 경우 유형 "hexBinary" 또는 "base64Binary"의 다른 모든 메시지 파트가 참조된 첨부로 바인드됩니다. 44 페이지의 『참조된 첨부: 최상위 레벨 메시지 파트』의 내용을 참조하십시오.
- 다른 모든 메시지 파트가 SOAP 헤더로 바인드됩니다.

JAX-RPC 및 JAX-WS 가져오기 바인딩은 여러 파트를 SOAP 본문에 바인드할 경우에도 멀티파트 문서 스타일 메시지가 있는 기존 WSDL 문서에서 SOAP 바인딩을 받아들입니다. 그러나 사용자는 Rational Application Developer에서 그러한 WSDL 문서에 대해 웹 서비스 클라이언트를 생성할 수 없습니다.

주: JAX-RPC 바인딩은 첨부를 지원하지 않습니다.

따라서 문서 스타일 SOAP 바인딩이 있는 조작에서 멀티파트 메시지를 사용하는 경우 권장되는 패턴은 다음과 같습니다.

1. 문서/리터럴 랩핑된 스타일을 사용합니다. 이 경우, 메시지에는 항상 단일 파트가 있습니다. 그러나 이 경우에는 첨부가 참조 해제되거나(49 페이지의 『참조되지 않은 첨부』에 설명된 대로) swaRef 유형 지정될 수 있습니다(39 페이지의 『참조된 첨부 : swaRef 유형 요소』에 설명된 대로).

2. RPC/리터럴 스타일을 사용합니다. 이 경우에는 SOAP 본문(결과에 항상 호출되는 조작을 나타내는 단일 하위 요소와, 해당 요소의 하위 요소가 되는 메시지 파트가 있는 SOAP 메시지)에 바인드되는 파트 수 측면에서 WSDL 바인딩에 대한 제한사항이 없습니다.
3. JAX-WS 바인딩의 경우 첫 번째 메시지 파트는 "hexBinary" 또는 "base64Binary" 유형이 아닌 파트가 되고 다른 모든 파트는 두 유형 중 하나의 파트가 되도록 합니다(모두 첨부로 바인드됨).
4. 다른 경우에는 위에 설명된 대로 동작합니다.

주: 참조된 첨부가 있는 멀티파트 문서 유형 SOAP 메시지를 수신할 경우, JAX-WS 바인딩은 값이 해당 컨텐츠 ID로 첨부를 식별하는 href 속성이 있는 SOAP 본문 하위 요소에 의해 각각의 참조된 첨부가 표시될 것을 예상합니다. JAX-WS 바인딩은 그러한 메시지에 대해 참조된 첨부를 동일한 방식으로 전송합니다. 이 동작은 WS-I 기본 프로파일을 준수하지 않습니다. WS-I 첨부 프로파일은 "컨텐츠 ID 파트 인코딩"을 정의합니다. 이는 href 속성이 있는 하위 요소가 생략되도록 허용하므로 그러한 메시지가 기본 프로파일을 준수하도록 합니다. JAX-WS 바인딩은 컨텐츠 ID 파트 인코딩을 사용하는 메시지의 전송 및 수신을 지원하지 않습니다. 사용자의 메시지가 준수하도록 하려면 위의 목록에 있는 52 페이지의 1 또는 2 접근방식을 따르거나 그러한 메시지에 참조된 첨부를 사용하지 말고 참조 해제되거나 swaRef 유형 지정된 참조를 대신 사용합니다.

HTTP 바인딩

HTTP 바인딩은 HTTP에 SCA(Service Component Architecture) 연결을 제공하도록 설계되었습니다. 결과적으로 기존 또는 새로 개발된 HTTP 응용프로그램이 SOA(Service Oriented Architecture) 환경에 포함될 수 있습니다.

하이퍼텍스트 전송 프로토콜(HTTP)은 웹에서의 정보 전송을 위해 널리 사용되는 프로토콜입니다. HTTP 프로토콜을 사용하는 외부 응용프로그램으로 작업 시, HTTP 바인딩이 필요합니다. HTTP 바인딩은 기본 형식의 메시지로서 SCA 응용프로그램의 비즈니스 오브젝트로 전달되는 데이터의 변환을 처리합니다. 또한 HTTP 바인딩은 비즈니스 오브젝트로서 전달되는 데이터를 수신 메시징을 위해 외부 응용프로그램에서 기대하는 기본 형식으로 변환할 수 있습니다.

주: 웹 서비스 SOAP/HTTP 프로토콜을 사용하는 서비스 및 클라이언트와 상호작용하려면 웹 서비스 표준 서비스 품질 처리에 대한 추가 기능을 제공하는 웹 서비스 바인딩 중 하나를 사용하는 것을 고려하십시오.

HTTP 바인딩을 사용하기 위한 일부 공통 시나리오는 다음 목록에서 설명됩니다.

- SCA 호스트 서비스는 HTTP 가져오기를 사용하여 HTTP 응용프로그램을 호출할 수 있습니다.

- SCA 호스트 서비스는 HTTP 내보내기를 사용하여 HTTP 클라이언트에서 이 서비스를 사용하도록 HTTP 사용 응용프로그램으로 표시할 수 있습니다.
- WebSphere Process Server 및 WebSphere Enterprise Service Bus는 HTTP 인프라에서 서로 통신할 수 있고 그 결과 사용자가 회사 표준에 따라 통신을 관리할 수 있습니다.
- WebSphere Process Server 및 WebSphere Enterprise Service Bus는 HTTP 통신의 중개자로서 역할하며, 메시지를 변환하고 라우팅하여 HTTP 네트워크를 사용한 응용프로그램의 통합을 개선합니다.
- WebSphere Process Server 및 WebSphere Enterprise Service Bus는 SOAP/HTTP 웹 서비스, JCA(Java Connector Architecture) 기반 자원 어댑터, JMS 등과 같이, HTTP와 다른 프로토콜 간의 브릿지에 사용될 수 있습니다.

HTTP 가져오기 및 내보내기 바인딩 작성에 대한 자세한 정보는 WebSphere Integration Developer Information Center에서 찾을 수 있습니다. **통합 응용프로그램 개발 → HTTP를 사용한 외부 서비스 액세스>** 주제를 참조하십시오.

HTTP 바인딩 개요

HTTP 바인딩은 HTTP 호스트 응용프로그램에 연결성을 제공합니다. HTTP 응용프로그램 간에 통신을 중개하며 기존의 HTTP 기반 응용프로그램이 모듈에서 호출되게 합니다.

HTTP 가져오기 바인딩

HTTP 가져오기 바인딩은 SCA(Service Component Architecture) 응용프로그램에서 HTTP 서버나 응용프로그램으로 아웃바운드 연결성을 제공합니다.

가져오기는 HTTP 엔드포인트 URL을 호출합니다. URL은 다음 세 가지 방법 중 하나로 지정될 수 있습니다.

- URL은 동적 대체 URL을 통해 HTTP 헤더에서 동적으로 설정될 수 있습니다.
- URL은 SMO 대상 주소 요소에 동적으로 설정될 수 있습니다.
- URL은 가져오기에서 구성 특성으로 지정될 수 있습니다.

이 호출은 사실상 항상 비동기입니다.

HTTP 호출은 항상 요청-응답이지만, HTTP 가져오기는 단방향 및 양방향 조작 둘 다를 지원하며 단방향 조작의 경우 응답을 무시합니다.

HTTP 내보내기 바인딩

HTTP 내보내기 바인딩은 HTTP 응용프로그램에서 SCA 응용프로그램으로 인바운드 연결성을 제공합니다.

URL은 HTTP 내보내기에 정의됩니다. 요청 메시지를 내보내기로 전송하려는 HTTP 응용프로그램은 이 URL을 사용하여 내보내기를 호출합니다.

HTTP 내보내기는 ping도 지원합니다.

런타임 시 HTTP 바인딩

런타임 시 HTTP 바인딩을 이용하는 가져오기는 메시지 본문의 데이터를 가지고 또는 데이터 없이 요청을 SCA 응용프로그램에서 외부 웹 서비스로 전송합니다. 그림 27에 표시된 것과 같이 SCA 응용프로그램에서 외부 웹 서비스로 요청됩니다.

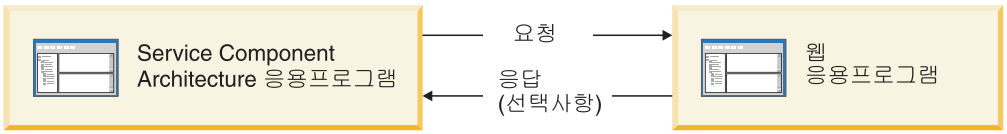


그림 27. SCA 응용프로그램에서 웹 응용프로그램으로의 요청 플로우

선택적으로, HTTP 바인딩을 이용한 가져오기는 요청에 응답하여 웹 응용프로그램에서 다시 데이터를 수신할 수 있습니다.

내보내기의 경우, 그림 28에 표시된 대로 클라이언트 응용프로그램에 의해 웹 서비스로 요청됩니다.

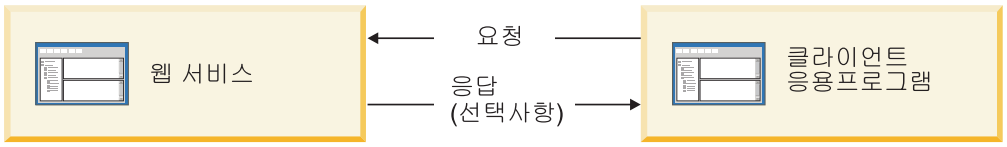


그림 28. 웹 서비스에서 클라이언트 응용프로그램으로의 요청 플로우.

웹 서비스는 서버에서 실행 중인 웹 응용프로그램입니다. 클라이언트가 해당 요청을 URL 주소로 전송하도록 서버로서 해당 웹 응용프로그램에서 내보내기가 구현됩니다. 서블릿은 런타임에 SCA 응용프로그램에 요청을 전달합니다.

선택적으로, 내보내기는 요청에 응답하여 데이터를 클라이언트 응용프로그램으로 전송할 수 있습니다.

HTTP 헤더

HTTP 가져오기 및 내보내기 바인딩을 사용하면 HTTP 헤더를 구성할 수 있으며 그 값을 아웃바운드 메시지에 사용할 수 있습니다. HTTP 가져오기에서는 요청에 이들 헤더를 사용하고 HTTP 내보내기에서는 응답에 이를 사용합니다.

정적으로 구성된 헤더 및 제어 정보가 런타임 시 동적으로 설정된 값보다 우선시됩니다. 그러나 동적 대체 URL, 버전 및 메소드 제어 값이 정적 값을 대체하며 그렇지 않은 경우 정적 값이 기본값으로 간주됩니다.

바인딩에서는 런타임 시 HTTP 대상 URL, 버전 및 메소드 값을 판별하여 HTTP 가져오기 URL의 동적 네이처를 지원합니다. 엔드포인트 참조, 동적 대체 URL, 버전 및 메소드의 값을 추출하여 이들 값을 판별합니다.

- 엔드포인트 참조의 경우, com.ibm.websphere.sca.addressing.EndpointReference API를 사용하거나 SMO 헤더에서 /headers/SMOHeader/Target/address 필드를 설정합니다.
- 동적 대체 URL, 버전 및 메소드의 경우 SCA(Service Component Architecture) 메시지의 HTTP 제어 매개변수 절을 사용하십시오. 동적 대체 URL이 대상 엔드포인트 참조보다 우위에 있지만 엔드포인트 참조가 바인딩을 통해 적용되기 때문에 엔드포인트 참조는 선호되는 접근 방식이고 가능한 곳에 사용되어야 한다는 것에 유의하십시오.

주: URL 형식, 구문 및 사용법에 대한 특정 정보와 동적 호출 정보는 관련 개념을 참조하십시오.

HTTP 내보내기 및 가져오기 바인딩 아래 있는 아웃바운드 메시지의 제어 및 헤더 정보는 다음 순서 대로 처리됩니다.

1. HTTP 동적 대체 URL, 버전 및 SCA 메시지의 메소드를 제외한 헤더 및 제어 정보(낮은 우선순위)
2. 내보내기/가져오기 레벨에 있는 관리 콘솔의 변경사항
3. 내보내기 또는 가져오기의 메소드 레벨에 있는 관리 콘솔의 변경사항
4. 엔드포인트 참조 또는 SMO 헤더를 통해 지정된 대상 주소
5. SCA 메시지의 메소드, 버전 및 동적 대체 URL
6. 데이터 핸들러 또는 데이터 바인딩의 헤더 및 제어 정보(가장 높은 우선순위)

HTTP 내보내기 및 가져오기는 프로토콜 헤더 전파가 `True`로 설정된 경우에만 수신 메시지(HTTPExportRequest 및 HTTPImportResponse)의 데이터로 인바운드 방향 헤더 및 제어 매개변수를 채웁니다. 이와 반대로, HTTP 내보내기 및 가져오기는 프로토콜 헤더 전파가 `True`로 설정된 경우에만 아웃바운드 헤더 및 제어 매개변수(HTTPExportResponse 및 HTTPImportRequest)를 읽고 처리합니다.

주: 가져오기 응답이나 내보내기 요청에서 헤더나 제어 매개변수에 대한 데이터 핸들러 또는 데이터 바인딩 변경사항은 가져오기 또는 내보내기 바인딩 내에서 메시지의 처리 명령어를 변경하지 않으며 수정된 값을 다운스트림 SCA 컴포넌트로 전파하기 위해서만 사용되어야 합니다.

컨텍스트 서비스는 SCA 호출 경로를 따라 컨텍스트(HTTP 헤더와 같은 프로토콜 헤더 및 계정 ID와 같은 사용자 컨텍스트 포함)를 전파합니다. WebSphere Integration Developer의 개발 동안 가져오기 및 내보내기 특성을 통해 컨텍스트의 전파를 제어할

수 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center 에서 가져오기 및 내보내기 바인딩 정보를 참조하십시오.

제공되는 HTTP 헤더 구조 및 지원

표 12에는 HTTP 가져오기 및 HTTP 내보내기 요청 및 응답의 요청/응답 매개변수가 목록으로 표시되어 있습니다.

표 12. 제공되는 HTTP 헤더 정보

제어 이름	**HTTP** 가져오기 요청	**HTTP** 가져오기 응답	**HTTP** 내보내기 요청	**HTTP** 내보내기 응답
URL	무시됨	설정되지 않음	요청 메시지에서 읽음. **주:** 조회 문자열도 URL 제어 매개변수의 일부입니다.	무시됨
버전(가능한 값: 1.0, 1.1, 기본값은 1.1)	무시됨	설정되지 않음	요청 메시지에서 읽음	무시됨
메소드	무시됨	설정되지 않음	요청 메시지에서 읽음	무시됨
동적 대체 URL	데이터 핸들러 및 데이터 바인딩에서 설정한 경우, HTTP 가져오기 URL을 대체함. 요청 행의 메시지에 기록됨. **주:** 조회 문자열도 URL 제어 매개변수의 일부입니다.	설정되지 않음	설정되지 않음	무시됨
동적 대체 버전	설정된 경우 HTTP 가져오기 버전을 대체함. 요청 행의 메시지에 기록됨.	설정되지 않음	설정되지 않음	무시됨
동적 대체 메소드	설정된 경우 HTTP 가져오기 메소드를 대체함. 요청 행의 메시지에 기록됨.	설정되지 않음	설정되지 않음	무시됨
매체 유형(이 제어 매개변수에는 컨텐츠 유형 HTTP 헤더 값의 일부가 포함되어 있습니다.)	요청이 있는 경우 컨텐츠 유형 헤더의 일부로서 메시지에 기록됨 **주:** 데이터 핸들러 또는 데이터 바인딩에서 이 제어 요소 값을 제공해야 합니다.	응답 메시지, 컨텐츠 유형 헤더에서 읽음	요청 메시지, 컨텐츠 유형 헤더에서 읽음	요청이 있는 경우 컨텐츠 유형 헤더의 일부로서 메시지에 기록됨 **주:** 데이터 핸들러 또는 데이터 바인딩에서 이 제어 요소 값을 제공해야 합니다.

표 12. 제공되는 HTTP 헤더 정보 (계속)

제어 이름	**HTTP** 가져오기 요청	**HTTP** 가져오기 응답	**HTTP** 내보내기 요청	**HTTP** 내보내기 응답
문자 세트(기본값: UTF-8)	요청이 있는 경우 컨텐츠 유형 헤더의 일부로서 메시지에 기록됨 **주:** 데이터 바인딩에서 이 제어 요소 값을 제공해야 합니다.	응답 메시지, 컨텐츠 유형 헤더에서 읽음	요청 메시지, 컨텐츠 유형 헤더에서 읽음	지원됨. 컨텐츠 유형 헤더의 일부로서 메시지에 기록됨. **주:** 데이터 바인딩에서 이 제어 요소 값을 제공해야 합니다.
전송 인코딩(가능한 값: 청크, ID, 기본값은 ID)	요청이 있는 경우 헤더로서 메시지에 기록되고 메시지 변환을 인코드하는 방법을 제어함	응답 메시지에서 읽음	요청 메시지에서 읽음	요청이 있는 경우 헤더로서 메시지에 기록되고 메시지 변환을 인코드하는 방법을 제어함
컨텐츠 인코딩(가능한 값: gzip, x-gzip, deflate, identity, 기본값은 identity)	요청이 있는 경우 헤더로서 메시지에 기록되고 페이로드를 인코드하는 방법을 제어함	응답 메시지에서 읽음	요청 메시지에서 읽음	요청이 있는 경우 헤더로서 메시지에 기록되고 페이로드를 인코드하는 방법을 제어함
데이터 길이	무시됨	응답 메시지에서 읽음	요청 메시지에서 읽음	무시됨
StatusCode(기본값: 200)	지원되지 않음	응답 메시지에서 읽음	지원되지 않음	요청이 있는 경우 응답 행의 메시지에 기록됨
ReasonPhrase(기본값: OK)	지원되지 않음	응답 메시지에서 읽음	지원되지 않음	제어 값이 무시됨. 메시지 응답 행 값은 StatusCode에서 생성됩니다.
인증(여러 개의 특성 포함)	요청이 있는 경우 기본 인증 헤더를 생성하는 데 사용됨 **주:** 이 헤더의 값은 HTTP 프로토콜에서만 인코드됩니다. SCA에서, 이 값은 디코드되어 일반 텍스트로 전달됩니다.	적용 불가능	요청 메시지 기본 인증 헤더에서 읽음. 이 헤더가 존재한다고 해서 사용자가 인증을 받은 아닙니다. 서블릿 구성에서 인증을 제어해야 합니다. **주:** 이 헤더의 값은 HTTP 프로토콜에서만 인코드됩니다. SCA에서, 이 값은 디코드되어 일반 텍스트로 전달됩니다.	적용 불가능
프록시(여러 개의 특성 포함: 호스트, 포트, 인증)	요청이 있는 경우 프록시를 통해 연결하는 데 사용됨	적용 불가능	적용 불가능	적용 불가능

표 12. 제공되는 HTTP 헤더 정보 (계속)

제어 이름	**HTTP** 가져오기 요청	**HTTP** 가져오기 응답	**HTTP** 내보내기 요청	**HTTP** 내보내기 응답
SSL(여러 개의 특성 포함: 키 저장소, 키 저장소 암호, 신뢰 저장소, 신뢰 저장소 암호, ClientAuth)	정보가 입력되고 대상 URL이 HTTPS인 경우 SSL을 통해 연결하는 데 사용됨	적용 불가능	적용 불가능	적용 불가능

HTTP 데이터 바인딩

SCA(Service Component Architecture) 메시지와 HTTP 프로토콜 메시지 간 데이터 맵핑이 각각 다른 경우는 반드시 데이터 핸들러 또는 HTTP 데이터 바인딩이 구성되어야 합니다. 데이터 핸들러는 전송 바인딩 전체에서 재사용하고 권장되는 접근방법을 나타낼 수 있는 바인딩 중립 인터페이스를 제공합니다. 데이터 바인딩은 특정 전송 바인딩에 대해 고유합니다. HTTP 고유 데이터 바인딩 클래스가 제공되며 사용자 정의 데이터 핸들러 또는 데이터 바인딩을 작성할 수도 있습니다.

주: 이 주제에서 설명한 세 개의 HTTP 데이터 바인딩 클래스 (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML 및 HTTPServiceGatewayDataBinding)는 WebSphere Process Server 버전 7.0에서 권장되지 않습니다. 이 주제에서 설명한 데이터 바인딩을 사용하는 대신 다음 데이터 핸들러를 고려하십시오.

- HTTPStreamDataBindingSOAP 대신 SOAPDataHandler 사용
- HTTPStreamDataBindingXML 대신 UTF8XMLDataHandler 사용
- HTTPServiceGatewayDataBinding 대신 GatewayTextDataHandler 사용

데이터 바인딩(2진 데이터 바인딩, XML 데이터 바인딩 및 SOAP 데이터 바인딩)은 HTTP 가져오기 및 HTTP 내보내기에서 사용하도록 제공됩니다. 응답 데이터 바인딩은 단방향 조작에는 필요하지 않습니다. 데이터 바인딩은 Java 클래스의 이름으로 표시되며 이 클래스 인스턴스는 HTTP에서 ServiceDataObject로, 또는 그 반대로 변환될 수 있습니다. 함수 선택기는 메소드 바인딩과 함께 결정할 수 있고 데이터 바인딩이 사용되고 조작이 호출되는 내보내기에서 사용되어야 합니다, 제공되는 데이터 바인딩은 다음과 같습니다.

- 본문이 비구조화 2진 데이터로 간주되는 2진 데이터 바인딩입니다. 2진 데이터 바인딩 XSD 스키마의 구현은 다음과 같습니다.

```
<
xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
```

```
</xsd:complexType>

<xsd:complexType name="HTTPBytesBody">
  <xsd:complexContent>
    <xsd:extension base="tns:HTTPBaseBody">
      <xsd:sequence>
        <xsd:element name="value" type="xsd:hexBinary"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- XML 데이터 바인딩은 본문을 XML 데이터로 지원합니다. XML 데이터 바인딩의 구현은 JMS XML 데이터 바인딩과 유사하며 인터페이스 스키마에 대한 어떠한 제한사항도 없습니다.
- SOAP 데이터 바인딩은 본문을 SOAP 데이터로 지원합니다. SOAP 데이터 바인딩의 구현에는 인터페이스 스키마에 대한 제한사항이 없습니다.

사용자 정의 HTTP 데이터 바인딩 구현

이 절은 사용자 정의 HTTP 데이터 바인딩의 구현 방법을 설명합니다.

주: 권장되는 접근방법은 전송 바인딩 전체에서 재사용할 수 있으므로 사용자 정의 데이터 핸들러를 구현하는 것입니다.

HTTPStreamDataBinding은 사용자 정의 HTTP 메시지를 처리하기 위한 프린시펄 인터페이스입니다. 인터페이스는 대용량 페이로드를 처리할 수 있도록 설계되어 있습니다. 그러나, 그러한 구현을 위해 이 데이터 바인딩은 메시지를 스트림으로 쓰기 전에 제어 정보와 헤더를 리턴해야 합니다.

아래에 나열된 실행 메소드와 순서는 사용자 정의 데이터 바인딩에 의해 구현되어야 합니다.

데이터 바인딩을 사용자 정의하려면 HTTPStreamDataBinding을 구현하는 클래스를 써야 합니다. 데이터 바인딩에는 다음과 같은 4개의 개인용 특성이 있어야 합니다.

- 개인용 DataObject pDataObject
- 개인용 HTTPControl pCtrl
- 개인용 HTTPHeaders pHeaders
- 개인용 yourNativeDataType nativeData

HTTP 바인딩은 다음과 같은 순서로 사용자 정의 데이터 바인딩을 호출합니다.

- 아웃바운드 처리(DataObject ~ 원시 형식):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)

4. setBusinessException(...)
5. convertToNativeData()
6. getControlParameters()
7. getHeaders()
8. write(...)

- 인바운드 처리(원시 형식 ~ DataObject):

1. setControlParameters(...)
2. setHeaders(...)
3. convertFromNativeData(...)
4. isBusinessException()
5. getDataObject()
6. getControlParameters()
7. getHeaders()

setDataObject(...)를 convertFromNativeData(...)에서 호출하여 원시 데이터에서 개인용 특성 "pDataObject"로 변환되는 dataObject 값을 설정해야 합니다.

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
  this.pHeaders = arg0;
}
/*
* Add http header "IsBusinessException" in pHeaders.
* Two steps:
* 1.Remove all the header with name IsBusinessException
(case-insensitive) first.
*   This is to make sure only one header is present.
* 2.Add the new header "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //remove all the header with name

  IsBusinessException (case-insensitive) first.
  //This is to make sure only one header is present.
  //add the new header "IsBusinessException", code example:
  HTTPHeader header=HeadersFactory.eINSTANCE

  .createHTTPHeader();
 entry1.setCName("BPEWebClient_localhost_server1");
  service.invoke("methodMultiParamater",paramObject);
```

```
  service.invoke("methodMultiParamater",paramObject);
}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Get header "IsBusinessException" from pHeaders,
return its boolean value
*/
public boolean isBusinessException() {
  String headerValue =

  getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();

  .get(0).getValueTemplates();
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Customer-developed method to
  //Read data from HTTPInputStream
  //Convert it to DataObject
DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
    .get(0).getValueTemplates();
}
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  .get(0).getValueTemplates();
}
```

EJB 바인딩

EJB(Enterprise JavaBeans) 가져오기 바인딩은 SCA(Service Component Architecture) 컴포넌트가 Java EE 서버에서 실행 중인 Java EE 비즈니스 로직이 제공한 서비스를 호출할 수 있도록 합니다. EJB 내보내기 바인딩을 사용하면 Java EE 비즈니스 로직이 SCA 컴포넌트를 호출할 수 있도록 SCA 컴포넌트가 Enterprise JavaBeans로 공개될 수 있습니다. 그렇지 않으면 해당 컴포넌트에 사용할 수 없습니다.

EJB 가져오기 바인딩

EJB 가져오기 바인딩을 사용하면 이용하는 모듈이 외부 EJB로 바인드되는 방법을 지정하여 SCA 모듈이 EJB 구현을 호출할 수 있습니다. 외부 EJB 구현에서 서비스를 가져오면 사용자는 비즈니스 로직을 WebSphere Process Server 환경에 연결하고 비즈니스 프로세스에 참여할 수 있습니다.

WebSphere Integration Developer를 사용하여 EJB 가져오기 바인딩을 작성합니다. 다음 프로시저 중 하나를 사용하여 바인딩을 생성할 수 있습니다.

- 외부 서비스 마법사를 사용하여 EJB 가져오기 작성

  WebSphere Integration Developer의 외부 서비스 마법사를 사용하여 기존 구현을 기반으로 하는 EJB 가져오기를 빌드할 수 있습니다. 외부 서비스 마법사는 사용자가 제공한 기준을 기반으로 서비스를 작성합니다. 그런 다음 발견된 서비스를 기반으로 하는 가져오기 파일, 인터페이스 및 비즈니스 오브젝트를 생성합니다.

- 어셈블리 편집기를 사용하여 EJB 가져오기 작성

  WebSphere Integration Developer 어셈블리 편집기를 사용하여 어셈블리 다이어그램 내에 EJB 가져오기를 작성할 수 있습니다. 팔레트에서 가져오기를 사용하거나 Java 클래스를 사용하여 EJB 바인딩을 작성할 수 있습니다.

생성된 가져오기에는 Java 브릿지 컴포넌트를 요구하는 대신 Java-WSDL 연결을 작성하는 데이터 바인딩이 있습니다. WSDL(Web Services Description Language) 참조가 있는 컴포넌트를 Java 인터페이스를 사용하는 EJB 기반 서비스와 통신하는 EJB 가져오기로 직접 연결할 수 있습니다.

EJB 가져오기는 EJB 2.1 프로그래밍 모델 또는 EJB 3.0 프로그래밍 모델을 사용해서 Java EE 비즈니스 로직과 상호작용할 수 있습니다.

Java EE 비즈니스 로직에 대한 호출은 원격 또는 로컬(EJB 3.0 전용)일 수 있습니다.

- 로컬 호출은 가져오기와 같은 서버에 상주하는 Java EE 비즈니스 로직을 호출하려는 경우 사용됩니다.

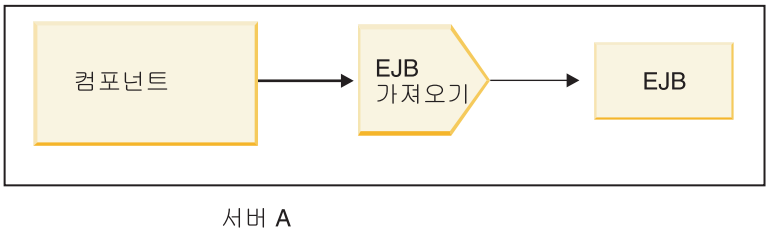


그림 29. EJB의 로컬 호출(EJB 3.0 전용)

- 원격 호출은 가져오기와 같은 서버에 상주하지 않는 Java EE 비즈니스 로직을 호출하려는 경우 사용됩니다.

  예를 들어, 다음 그림에서 EJB 가져오기는 RMI/IIOP(Remote Method Invocation over Internet InterORB Protocol)를 사용하여 다른 서버에서 EJB 메소드를 호출할 수 있습니다.

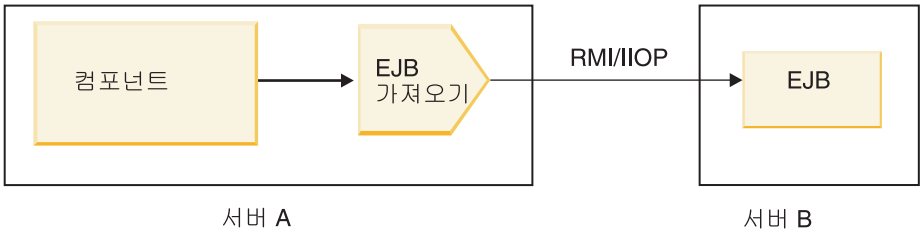


그림 30. EJB의 원격 호출

EJB 바인딩을 구성할 때 WebSphere Integration Developer는 JNDI 이름을 사용하여 EJB 프로그래밍 모델 레벨 및 호출 유형(로컬 또는 원격)을 결정할 수 있습니다.

EJB 가져오기 바인딩에는 다음 주 컴포넌트가 포함됩니다.

- JAX-WS 데이터 핸들러
- EJB 결함 선택기
- EJB 가져오기 함수 선택기

사용자 시나리오가 JAX-WS 맵핑을 기반으로 하지 않는 경우 타스크를 수행하기 위해 사용자 정의 데이터 핸들러, 함수 선택기 및 결함 선택기가 필요할 수 있습니다. 그렇지 않은 경우에는 EJB 가져오기 바인딩의 파트인 컴포넌트를 통해 완료됩니다. 여기에는 일반적으로 사용자 정의 맵핑 알고리즘에 의해 완료되는 맵핑이 포함됩니다.

EJB 내보내기 바인딩

외부 Java EE 응용프로그램은 EJB 내보내기 바인딩 방식으로 SCA 컴포넌트를 호출할 수 있습니다. EJB 내보내기를 사용하면 외부 Java EE 응용프로그램이 EJB 프로그래밍 모델을 사용하여 해당 컴포넌트를 호출할 수 있습니다.

주: EJB 내보내기는 Stateless Bean입니다.

WebSphere Integration Developer를 사용하여 EJB 바인딩을 작성합니다. 다음 프로시저 중 하나를 사용하여 바인딩을 생성할 수 있습니다.

- 외부 서비스 마법사를 사용하여 EJB 내보내기 바인딩 작성

  WebSphere Integration Developer에서 외부 서비스 마법사를 사용하여 기존 구현을 기초로 EJB 내보내기 서비스를 빌드할 수 있습니다. 외부 서비스 마법사는 사용자가 제공한 기준을 기반으로 서비스를 작성합니다. 그런 다음 발견된 서비스를 기초로 비즈니스 오브젝트, 인터페이스 및 내보내기 파일을 생성합니다.
- 어셈블리 편집기를 사용하여 EJB 내보내기 바인딩 작성

  WebSphere Integration Developer 어셈블리 편집기를 사용하여 EJB 내보내기를 작성할 수 있습니다.

기존 SCA 컴포넌트에서 바인딩을 생성하거나, Java 인터페이스에 대한 EJB 바인딩으로 내보내기를 생성할 수 있습니다.

- 기존의 WSDL 인터페이스가 있는 기존 SCA 컴포넌트에 대해 내보내기를 생성할 경우 내보내기에는 Java 인터페이스가 지정됩니다.
- Java 인터페이스에 대한 내보내기를 생성할 때 내보내기에 대한 WSDL 또는 Java 인터페이스를 선택할 수 있습니다.

  주: EJB 내보내기를 작성하기 위해 사용되는 Java 인터페이스에는 원격 호출의 매개변수로 전달되는 오브젝트(입력 및 출력 매개변수와 예외)에 대해 다음과 같은 제한사항이 있습니다.

  – 구체적 유형(인터페이스 또는 추상 유형 대신)이어야 합니다.
  – Enterprise JavaBean 스펙을 준수해야 합니다. 직렬화 가능하고 기본값인 인수 없는 생성자를 보유해야 하며 모든 특성이 getter 및 setter 메소드를 통해 액세스 가능해야 합니다.

    Enterprise JavaBean 스펙에 대한 정보는 Sun Microsystems, Inc. 웹 사이트 (http://java.sun.com)를 참조하십시오.

  또한 예외는 java.lang.Exception에서 상속된 확인된 예외여야 하며 단일형(즉, 여러 개의 확인된 예외 유형 발생을 지원하지 않음)이어야 합니다.

  또한 Java EnterpriseBean의 비즈니스 인터페이스는 일반 Java 인터페이스이며 javax.ejb.EJBObject 또는 javax.ejb.EJBLocalObject를 확장하면 안됩니다. 비즈니스 인터페이스의 메소드는 java.rmi.Remote.Exception을 처리하면 안됩니다.

EJB 내보내기 바인딩은 EJB 2.1 프로그래밍 모델이나 EJB 3.0 프로그래밍 모델을 사용하여 Java EE 비즈니스 로직과 상호작용할 수 있습니다.

호출은 로컬(EJB 3.0의 경우만) 또는 원격이 될 수 있습니다.

- 로컬 호출은 Java EE 비즈니스 로직이 내보내기와 동일한 서버에 상주하는 SCA 컴포넌트를 호출할 때 사용됩니다.
- 원격 호출은 Java EE 비즈니스 로직이 내보내기와 동일한 서버에 상주하지 않는 경우에 사용됩니다.

  예를 들어, 다음 그림에서 EJB는 RMI/IIOP를 사용하여 다른 서버에서 SCA 컴포넌트를 호출합니다.

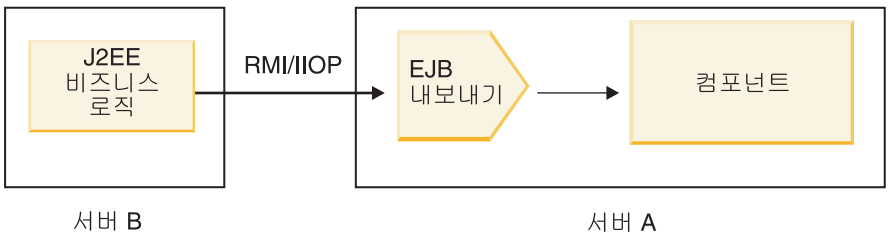


그림 31. EJB 내보내기 방식으로 클라이언트에서 SCA 컴포넌트로의 원격 호출

EJB 바인딩을 구성할 때 WebSphere Integration Developer는 JNDI 이름을 사용하여 EJB 프로그래밍 모델 레벨 및 호출 유형(로컬 또는 원격)을 결정할 수 있습니다.

EJB 내보내기 바인딩에는 다음의 주요 컴포넌트가 포함됩니다.

- JAX-WS 데이터 핸들러
- EJB 내보내기 함수 선택기

사용자 시나리오가 JAX-WS 맵핑을 기초로 하지 않는 경우 타스크를 수행하기 위해 사용자 정의 데이터 핸들러 및 함수 선택기가 필요할 수 있습니다. 그렇지 않은 경우에는 EJB 내보내기 바인딩의 일부인 컴포넌트에 의해 완료됩니다. 여기에는 일반적으로 사용자 정의 맵핑 알고리즘에 의해 완료되는 맵핑이 포함됩니다.

EJB 바인딩 특성

EJB 가져오기 바인딩은 해당되는 구성된 JNDI 이름을 사용하여 EJB 프로그래밍 모델 레벨 및 호출 유형(로컬 또는 원격)을 결정할 수 있습니다. EJB 가져오기 및 내보내기 바인딩은 데이터 변환에 대해 JAX-WS 데이터 핸들러를 사용합니다. EJB 가져오기 바인딩은 EJB 가져오기 함수 선택기 및 EJB 결함 선택기를 사용하고 EJB 내보내기 바인딩은 EJB 내보내기 함수 선택기를 사용합니다.

JNDI 이름 및 EJB 가져오기 바인딩:

가져오기에 대해 EJB 바인딩을 구성할 때 WebSphere Integration Developer는 JNDI 이름을 사용하여 EJB 프로그래밍 모델 레벨 및 호출 유형(로컬 또는 원격)을 결정할 수 있습니다.

JNDI 이름이 지정되지 않은 경우 기본 EJB 인터페이스 바인딩이 사용됩니다. 작성되는 기본 이름은 EJB 2.1 JavaBean 또는 EJB 3.0 JavaBean의 호출 여부에 따라 달라집니다.

주: 이름 지정 규칙에 대한 자세한 정보는 WebSphere Application Server Information Center에서 EJB 3.0 응용프로그램 바인딩 개요를 참조하십시오.

- EJB 2.1 JavaBean

  WebSphere Integration Developer에 의해 사전 선택된 기본 JNDI 이름은 기본 EJB 2.1 바인딩으로, `ejb/`와 홈 인터페이스를 슬래시로 구분하여 형성됩니다.

예를 들어, com.mycompany.myremotebusinesshome에 대한 EJB 2.1 JavaBean의 홈 인터페이스 경우 기본 바인딩은 다음과 같습니다.

```
ejb/com/mycompany/myremotebusinesshome
```

EJB 2.1의 경우에는 원격 EJB 호출만 지원됩니다.

- EJB 3.0 JavaBean

  로컬 JNDI에 대해 WebSphere Integration Developer에서 사전 선택된 기본 JNDI 이름은 앞에 `ejblocal:`이 붙은 로컬 인터페이스의 완전한 클래스 이름입니다. 예를 들어, 로컬 interface com.mycompany.mylocalbusiness의 완전한 인터페이스에 대해 사전 선택된 EJB 3.0 JNDI는 다음과 같습니다.

  ```
  ejblocal:com.mycompany.mylocalbusiness
  ```

  원격 interface com.mycompany.myremotebusiness의 경우 사전 선택된 EJB 3.0 JNDI는 완전한 인터페이스입니다.

  ```
  com.mycompany.myremotebusiness
  ```

  EJB 3.0 기본 응용프로그램 바인딩은 EJB 3.0 응용프로그램 바인딩 개요에서 설명됩니다.

  WebSphere Integration Developer는 버전 3.0 프로그래밍 모델을 사용하는 EJB에 대한 기본 JNDI 위치로 "축약형" 이름을 사용합니다.

  주: 사용자 정의 맵핑이 사용되었거나 구성되어, 대상 EJB의 전개된 JNDI 참조가 기본 JNDI 바인딩 위치와 다른 경우 대상 JNDI 이름을 적절하게 지정해야 합니다. 전개하기 전에 WebSphere Integration Developer에서 이름을 지정하거나, 가져오기 바인딩의 경우 관리 콘솔에서 이름을 변경하여(전개 후) 대상 EJB의 JNDI 이름을 일치시킬 수 있습니다.

EJB 바인딩 작성에 대한 자세한 정보는 WebSphere Integration Developer Information Center에서 EJB 바인딩에 대한 작업 관련 절을 참조하십시오.

JAX-WS 데이터 핸들러:

EJB(Enterprise JavaBeans) 가져오기 바인딩은 JAX-WS 데이터 핸들러를 사용하여 요청 비즈니스 오브젝트를 Java 오브젝트 매개변수로 변경하고 Java 오브젝트 리턴값을 응답 비즈니스 오브젝트로 변경합니다. EJB 내보내기 바인딩은 JAX-WS 데이터 핸들러를 사용하여 요청 EJB를 요청 비즈니스 오브젝트로 변경하고 응답 비즈니스 오브젝트를 리턴값으로 변경합니다.

이 데이터 핸들러는 JAX-WS(Java API for XML Web Services) 스펙 및 JAXB(Java Architecture for XML Binding) 스펙을 사용하여 SCA 지정 WSDL 인터페이스에서 대상 EJB Java 인터페이스 (및 반대로)로 데이터를 맵핑합니다.

주: 현재 지원은 JAX-WS 2.1.1 및 JAXB 2.1.3 스펙으로 제한됩니다.

EJB 바인딩 레벨에서 지정된 데이터 핸들러는 요청, 응답, 결함 및 런타임 예외 처리를 수행하는 데 사용됩니다.

주: 결함의 경우, 특정 데이터 핸들러는 faultBindingType 구성 특성을 지정하여 각 결함용으로 지정될 수 있습니다. 이것은 EJB 바인딩 레벨에서 지정한 값을 대체합니다.

JAX-WS 데이터 핸들러는 EJB 바인딩이 WSDL 인터페이스를 가질 때 기본값으로 사용됩니다. 이 데이터 핸들러는 JAX-WS 호출을 표시하는 SOAP 메시지를 데이터 오브젝트로 변환하는 데 사용할 수 없습니다.

EJB 가져오기 바인딩은 데이터 핸들러를 사용하여 데이터 오브젝트를 Java 오브젝트 배열(Object[]) 로 변환합니다. 아웃바운드 통신 중에 다음 처리가 발생합니다.

1. EJB 바인딩이 WSDL에 지정된 예상 유형, 예상 요소 및 대상 메소드 이름과 일치시키기 위해 BindingContext에 설정합니다.
2. EJB 바인딩이 데이터 변환이 필요한 데이터 오브젝트의 변환 메소드를 호출합니다.
3. 데이터 핸들러가 메소드의 매개변수를(메소드 안에서 해당 정의의 순서로) 표시하는 Object[]를 리턴합니다.
4. EJB 바인딩이 Object[]를 사용하여 대상 EJB 인터페이스에서 메소드를 호출합니다.

바인딩은 또한 EJB 호출로부터의 응답을 처리하기 위해 Object[]를 준비합니다.

- Object[]의 첫 번째 요소는 Java 메소드 호출로부터의 리턴값입니다.
- 후속 값은 메소드의 입력 매개변수를 표시합니다.

이는 In/Out 및 Out 유형의 매개변수를 지원하기 위해 필요합니다.

Out 유형의 매개변수의 경우 응답 데이터 오브젝트에 값이 리턴되어야 합니다.

데이터 핸들러는 Object[]에서 찾은 값을 처리하여 변환한 후 데이터 오브젝트에 대한 응답을 리턴합니다.

데이터 핸들러는 기타 XSD 데이터 유형과 함께 xs:AnyType, xs:AnySimpleType 및 xs:Any를 지원합니다. xs:Any에 대한 지원을 사용 가능하게 하려면 다음 예제와 같이 Java 코드의 JavaBean 특성에 대해 `@XmlAnyElement (lax=true)`를 사용하십시오.

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )
```

```
 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

TestType의 특성 오브젝트가 xs:any 필드로 됩니다. xs:any 필드에 사용되는 Java 클래스 값에는 `@XmlAnyElement` 어노테이션이 있어야 합니다. 예를 들어, 오브젝트 배열을 채우는 데 사용 중인 Java 클래스가 주소인 경우 주소 클래스에는 `@XmlRootElement` 어노테이션이 있어야 합니다.

주: XSD 유형에서 JAX-WS 스펙에 의해 정의된 Java 유형으로의 맵핑을 사용자 정의하려면 비즈니스 요구에 맞게 JAXB 어노테이션을 변경하십시오. JAX-WS 데이터 핸들러는 xs:any, xs:anyType 및 xs:anySimpleType을 지원합니다.

JAX-WS 데이터 핸들러에 대해 다음 제한사항을 적용할 수 있습니다.

- 데이터 핸들러에는 헤더 속성 `@WebParam` 어노테이션에 대한 지원이 포함되지 않습니다.
- 비즈니스 오브젝트 스키마 파일(XSD 파일)의 네임스페이스가 Java 패키지 이름의 기본 맵핑을 포함하지 않습니다. package-info.java의 어노테이션 `@XMLSchema`도 작동하지 않습니다. 네임스페이스가 있는 XSD를 작성하는 유일한 방법은 `@XmlType` 및 `@XmlRootElement` 어노테이션을 사용하는 것입니다. `@XmlRootElement`는 JavaBean 유형의 글로벌 요소에 대한 대상 네임스페이스를 정의합니다.
- EJB 가져오기 마법사는 관련되지 않은 클래스에 대해 XSD 파일을 작성하지 않습니다. 버전 2.0에서는 `@XmlSeeAlso` 어노테이션을 지원하지 않으므로 하위 클래스가 상위 클래스로부터 직접 참조되지 않으면 XSD가 작성되지 않습니다. 이 문제점은 해당 하위 클래스에 대해 SchemaGen을 실행하여 해결합니다.

  SchemaGen은 지정된 JavaBean에 대한 XSD 파일을 작성하기 위해 제공되는 명령행 유틸리티(*WPS_Install_Home*/bin 디렉토리에 위치)입니다. 이러한 XSD는 작동할 솔루션에 대한 모듈에 수동으로 복사되어야 합니다.

EJB 결함 선택기:

EJB 결함 선택기는 EJB 호출로 인해 결함, 런타임 예외 또는 성공적인 응답이 발생했는지 여부를 판별합니다.

결함이 발견되면 EJB 결함 선택기가 기본 결함 이름을 런타임 바인딩으로 리턴해서 JAX-WS 데이터 핸들러가 예외 오브젝트를 결함 비즈니스 오브젝트로 변환할 수 있습니다.

성공적인(비결함) 응답의 경우 EJB 가져오기 바인딩은 값을 리턴하도록 Java 오브젝트 배열(Object[])을 어셈블합니다.

- Object[]의 첫 번째 요소는 Java 메소드 호출로부터의 리턴값입니다.

- 후속 값은 메소드의 입력 매개변수를 표시합니다.

이는 In/Out 및 Out 유형의 매개변수를 지원하기 위해 필요합니다.

예외 시나리오의 경우 바인딩은 Object[]를 어셈블하며 첫 번째 요소는 메소드에 의해 발생한 예외를 표시합니다.

결함 선택기는 다음과 같은 값을 리턴할 수 있습니다.

표 13. 리턴값

유형	리턴값	설명
결함	`ResponseType.FAULT`	전달된 Object[]에 예외 오브젝트가 포함될 때 리턴됩니다.
런타임 예외	`ResponseType.RUNTIME`	예외 오브젝트가 메소드의 선언된 예외 유형과 일치하지 않으면 리턴됩니다.
정상 응답	`ResponseType.RESPONSE`	기타 모든 경우에 리턴됩니다.

결함 선택기가 `ResponseType.FAULT` 값을 리턴하면 기본 결함 이름이 리턴됩니다. 이 기본 결함 이름은 모델에서 해당 WSDL 결함 이름을 결정하고 올바른 결함 데이터 핸들러를 호출하기 위해 바인딩에 의해 사용됩니다.

EJB 함수 선택기:

EJB 바인딩은 가져오기 함수 선택기(아웃바운드 처리를 위한)나 내보내기 함수 선택기(인바운드 처리를 위한)를 사용하여 호출할 EJB 메소드를 판별합니다.

가져오기 함수 선택기

아웃바운드 처리를 위해, 가져오기 함수 선택기는 EJB 가져오기에 연결된 SCA 컴포넌트에 의해 호출된 조작의 이름을 기초로 EJB 메소드 유형을 파생시킵니다. 함수 선택기는 WebSphere Integration Developer에서 생성된 JAX-WS 어노테이션이 있는 Java 클래스에서 @WebMethod 어노테이션을 찾아서 연관된 대상 조작 이름을 판별합니다.

- @WebMethod 어노테이션이 존재하면, 함수 선택기는 @WebMethod 어노테이션을 사용하여 WSDL 메소드에 대한 올바른 Java 메소드 맵핑을 판별합니다.
- @WebMethod 어노테이션이 누락된 경우 함수 선택기는 Java 메소드 이름이 호출된 조작 이름과 같은 것으로 간주합니다.

주: 이 함수 선택기는 EJB 가져오기의 Java 유형 지정 인터페이스가 아니라 EJB 가져오기의 WSDL 유형 지정 인터페이스에 대해서만 유효합니다.

함수 선택기는 EJB 인터페이스의 메소드를 표시하는 java.lang.reflect.Method 오브젝트를 리턴합니다.

함수 선택기는 Java 오브젝트 배열(Object[])을 사용하여 대상 메소드의 응답을 포함합니다. Object[]의 첫 번째 요소는 WSDL의 이름이 있는 Java 메소드이고 Object[]의

두 번째 요소는 입력 비즈니스 오브젝트입니다.

내보내기 함수 선택기

인바운드 처리를 위해, 내보내기 함수 선택기는 Java 메소드에서 호출될 대상 메소드를 파생시킵니다.

내보내기 함수 선택기는 EJB 클라이언트가 호출한 Java 조작 이름을 대상 컴포넌트의 인터페이스에 있는 조작의 이름으로 맵핑합니다. 메소드 이름이 문자열로 리턴되고 대상 컴포넌트의 인터페이스 유형에 따라 SCA 런타임에 의해 분석됩니다.

EIS 바인딩

EIS(Enterprise Information System) 바인딩은 SCA 컴포넌트 및 외부 EIS 간의 연결성을 제공합니다. 이 통신은 JCA 1.5 자원 어댑터와 Websphere Adapter를 지원하는 EIS 내보내기 및 EIS 가져오기를 사용하여 수행됩니다.

SCA 컴포넌트에서 데이터를 외부 EIS로 전송하거나 가져와야 할 수도 있습니다. 이러한 연결이 필요한 SCA 모듈을 작성하는 경우 특정 외부 EIS와의 통신을 위해 SCA 컴포넌트뿐만 아니라 EIS 바인딩에서 가져오기 또는 내보내기를 포함합니다.

WebSphere Integration Developer의 자원 어댑터는 가져오기 또는 내보내기의 컨텍스트 내에서 사용됩니다. 외부 서비스 마법사에서 가져오기 또는 내보내기를 개발하고, 개발 시 자원 어댑터를 포함합니다. 응용프로그램이 EIS 시스템에서 서비스를 호출하게 하는 EIS 가져오기, 또는 EIS 시스템에서 응용프로그램이 WebSphere Integration Developer에서 개발되는 서비스를 호출하게 하는 EIS 내보내기가 특정 자원 어댑터로 작성됩니다. 예를 들어, JD Edwards 시스템에서 서비스를 호출하기 위해 JD Edwards 어댑터로 가져오기를 작성합니다.

외부 서비스 마법사 사용 시, EIS 바인딩 정보가 작성됩니다. 또한 다른 도구(어셈블리 편집기)를 사용하여 바인딩 정보를 추가 또는 수정할 수 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center를 참조하십시오.

EIS 바인딩을 포함하는 SCA 모듈이 서버로 전개된 후에, 관리 콘솔을 사용하여 바인딩에 대한 정보를 보고 바인딩을 구성할 수 있습니다.

EIS 바인딩 개요

엔터프라이즈 정보 시스템(EIS) 바인딩을 사용하면 엔터프라이즈 정보 시스템의 서비스에 액세스하거나 EIS에서 서비스를 사용할 수 있습니다(JCA 자원 어댑터와 함께 사용된 경우).

다음 예제는 SCA 모듈 ContactSyncModule은 Siebel 시스템과 SAP 시스템 간의 문의 정보를 동기화합니다.

1. SCA 컴포넌트인 ContactSync는 Siebel Contact라는 EIS 응용프로그램 내보내기를 통해 Siebel 문의에 대한 변경사항을 청취합니다.
2. ContactSync SCA 컴포넌트는 SAP 문의 정보를 적절히 갱신하기 위해 EIS 응용프로그램 가져오기를 통해 SAP 응용프로그램을 사용합니다.

Siebel 및 SAP 시스템에서 문의 정보를 저장하는 데 사용되는 데이터 구조가 서로 다르므로 ContactSync SCA 컴포넌트가 맵핑을 제공해야 합니다.

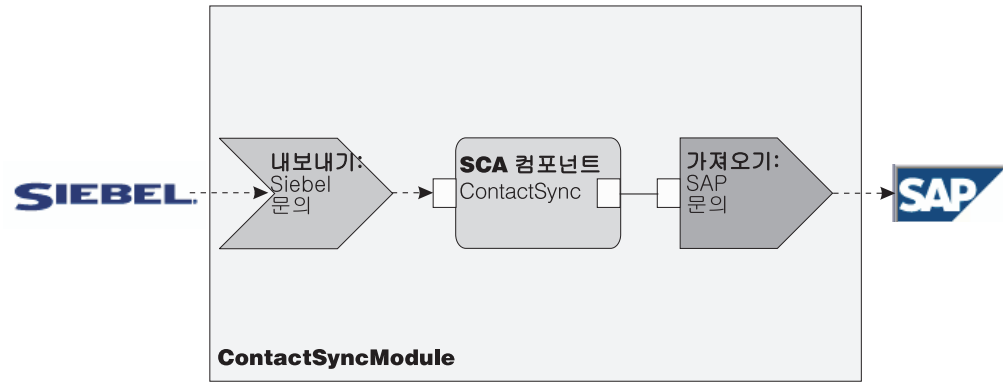


그림 32. Siebel 시스템에서 SAP 시스템으로의 플로우

Siebel 문의 내보내기 및 SAP 문의 가져오기를 통해 적절한 자원 어댑터가 구성됩니다.

EIS 바인딩의 주요 기능

EIS 가져오기는 SCA 모듈의 컴포넌트가 SCA 모듈 외부에서 정의된 EIS 응용프로그램을 사용할 수 있게 해주는 SCA(Service Component Architecture) 가져오기입니다. EIS 가져오기는 데이터를 SCA 컴포넌트에서 외부 EIS로 전송하는 데 사용되고 EIS 내보내기는 데이터를 외부 EIS에서 SCA 모듈로 전송하는 데 사용됩니다.

가져오기

EIS 가져오기의 역할은 SCA 컴포넌트와 외부 EIS 시스템 간의 차이점을 보완하는 것입니다. 외부 응용프로그램을 EIS 가져오기로 취급할 수 있습니다. 이런 경우 EIS 가져오기는 외부 EIS로 데이터를 전송하고 선택적으로 응답에 데이터를 수신합니다.

EIS 가져오기는 모듈의 외부 응용프로그램의 동일한 보기를 SCA 컴포넌트에 제공합니다. 이로 인해 컴포넌트가 일관된 SCA 모델을 사용하여 SAP, Siebel 또는 PeopleSoft와 같은 외부 EIS와 통신할 수 있습니다.

가져오기의 클라이언트측에는 하나 이상의 메소드를 사용하여(각 메소드가 데이터 오브젝트를 인수로 사용하고 값을 리턴함) EIS 가져오기 응용프로그램에서 공개한 인터페이스가 있습니다. 구현측에는 자원 어댑터가 구현한 공통 클라이언트 인터페이스(CCI)가 있습니다.

EIS 가져오기의 런타임 구현은 클라이언트 측 인터페이스와 CCI를 연결합니다. 가져오기는 인터페이스의 메소드 호출을 CCI의 호출에 맵핑합니다.

바인딩은 세 레벨에서 작성됩니다. 인터페이스 바인딩에서는 다음으로 포함된 메소드 바인딩을 사용하고 이 바인딩에서는 그 다음 데이터 바인딩을 사용합니다.

인터페이스 바인딩은 가져오기의 인터페이스를 응용프로그램을 제공하는 EIS 시스템에 대한 연결과 관련시킵니다. 이는 EIS의 특정 인스턴스가 인터페이스가 표시하는 응용프로그램 세트를 제공하고 연결이 이 인스턴스에 대한 액세스를 제공한다는 사실을 나타냅니다. 바인딩 요소에는 연결을 작성하는 데 충분한 정보가 포함된 특성이 들어 있습니다(이러한 특성은 javax.resource.spi.ManagedConnectionFactory 인스턴스의 일부임).

메소드 바인딩은 특정 상호작용을 사용하는 메소드와 EIS 시스템을 연관시킵니다. JCA의 경우, 상호작용의 특징은 javax.resource.cci.InteractionSpec 인터페이스 구현의 특성 세트가 있다는 점입니다. 메소드 바인딩의 상호작용 요소에 클래스 이름과 함께 이러한 특성이 들어 있어 상호작용을 수행하는 데 충분한 정보를 제공합니다. 메소드 바인딩은 EIS 표시에 대한 인터페이스 메소드 결과 및 인수 맵핑에 대해 설명하는 데이터 바인딩을 사용합니다.

EIS 가져오기의 런타임 시나리오는 다음과 같습니다.

1. SCA 프로그래밍 모델을 사용하여 가져오기 인터페이스의 메소드가 호출됩니다.
2. EIS 가져오기에 도달하는 요청에는 메소드의 이름과 해당 인수가 들어 있습니다.
3. 가져오기는 제일 먼저 인터페이스 바인딩 구현을 작성한 다음 가져오기 바인딩의 데이터를 사용하여 ConnectionFactory를 작성하고 이 둘을 연관시킵니다. 즉, 가져오기는 인터페이스 바인딩에서 setConnectionFactory를 호출합니다.
4. 호출한 메소드와 일치하는 메소드 바인딩 구현이 작성됩니다.
5. javax.resource.cci.InteractionSpec 인스턴스를 작성하여 채운 다음 데이터 바인딩을 사용하여 메소드 인수를 자원 어댑터에서 인식하는 형식으로 바인드합니다.
6. CCI 인터페이스를 사용하여 상호작용이 수행됩니다.
7. 호출이 리턴되면 데이터 바인딩을 사용하여 호출 결과가 작성되고 결과가 호출자에게 리턴됩니다.

내보내기

EIS 내보내기의 역할은 SCA 컴포넌트와 외부 EIS 간 차이점을 보완하는 것입니다. 외부 응용프로그램을 EIS 내보내기로 취급할 수 있습니다. 이런 경우 외부 응용프로그램은 데이터를 정기적인 공고 양식으로 전송합니다. EIS 내보내기를 EIS의 외부 요청을 청취하는 등록 응용프로그램으로 간주할 수 있습니다. EIS 내보내기를 사용하는 SCA 컴포넌트에서는 이를 로컬 응용프로그램으로 표시합니다.

EIS 내보내기는 모듈의 외부 응용프로그램의 동일한 보기를 SCA 컴포넌트에 제공합니다. 이로 인해 컴포넌트가 일관된 SCA 모델을 사용하여 SAP, Siebel 또는 PeopleSoft와 같은 EIS와 통신할 수 있습니다.

내보내기는 EIS에서 요청을 수신하는 리스너 구현 기능을 수행합니다. 리스너는 자원 어댑터 지정 리스너 인터페이스를 구현합니다. 내보내기에는 내보내기를 통해 EIS에 공개된 컴포넌트 구현 인터페이스도 들어 있습니다.

EIS 내보내기의 런타임 구현은 리스너와 컴포넌트 구현 인터페이스를 연결합니다. 내보내기는 EIS 요청을 컴포넌트에서 적합한 조작의 호출에 맵핑합니다. 바인딩은 세 레벨에서 작성됩니다. 하나는 리스너 바인딩이고, 이 바인딩에서는 다음으로 포함된 기본 메소드 바인딩을 사용하고, 이 바인딩에서는 데이터 바인딩을 사용합니다.

리스너 바인딩은 요청을 수신하는 리스너와 내보내기를 통해 공개된 컴포넌트를 관련시킵니다. 내보내기 정의에는 컴포넌트 이름이 들어 있으며 런타임에서 이 이름을 찾아 요청을 전달합니다.

기본 메소드 바인딩은 리스너가 수신한 이벤트 유형 또는 기본 메소드를 내보내기를 통해 공개된 컴포넌트가 구현한 조작에 연관시킵니다. 리스너에서 호출된 메소드와 이벤트 유형 간에는 아무 관련이 없으며 모든 이벤트는 하나 이상의 리스너 메소드를 통해 수신됩니다. 기본 메소드 바인딩은 내보내기에서 정의된 함수 선택기를 사용하여 EIS의 데이터 형식을 컴포넌트가 인식하는 형식으로 바인드하기 위해 인바운드 데이터 및 데이터 바인딩에서 기본 메소드 이름을 추출합니다.

EIS 내보내기의 런타임 시나리오는 다음과 같습니다.

1. EIS 요청이 리스너 구현에서 메소드 호출을 트리거합니다.
2. 리스너는 내보내기를 찾아서 이를 호출하여 여기에 모든 호출 인수를 전달합니다.
3. 내보내기는 리스너 바인딩 구현을 작성합니다.
4. 내보내기가 함수 선택기를 인스턴스화하여 리스너 바인딩에 설정합니다.
5. 내보내기가 기본 메소드 바인딩을 초기화하여 리스너 바인딩에 추가합니다. 각각의 기본 메소드 바인딩에서 데이터 바인딩도 초기화됩니다.
6. 내보내기는 리스너 바인딩을 호출합니다.
7. 리스너 바인딩이 내보낸 컴포넌트를 찾고 함수 선택기를 사용하여 기본 메소드 이름을 검색합니다.
8. 이 이름을 사용하여 기본 메소드 바인딩을 찾은 후 해당 바인딩이 대상 컴포넌트를 호출합니다.

어댑터 상호작용 양식을 사용하면 EIS 내보내기 바인딩에서 비동기적(기본값) 또는 동기적으로 대상 컴포넌트를 호출할 수 있습니다.

자원 어댑터

외부 서비스 마법사를 사용하여 가져오기 또는 내보내기를 개발하고 개발 중에 자원 어댑터를 포함합니다. CICS, IMS, JD Edwards, PeopleSoft, SAP 및 Siebel 시스템에 액세스하는 데 사용된 WebSphere Integration Developer와 함께 제공되는 어댑터는 개발 및 테스트에만 사용됩니다. 이 어댑터를 사용하여 응용프로그램을 개발하고 테스트한다는 의미입니다.

응용프로그램을 전개하면, 응용프로그램을 실행하기 위해 라이센스가 부여된 런타임 어댑터가 필요합니다. 하지만 서비스 빌드 시 어댑터를 서비스로 임베드할 수 있습니다. 어댑터 라이센스 부여를 통해 임베디드 어댑터를 라이센스가 부여된 런타임 어댑터로 사용할 수 있습니다. 이러한 어댑터는 JCA 1.5(Java EE Connector Architecture)를 준수합니다. 개방 표준인 JCA는 EIS 연결을 위한 Java EE 표준입니다. JCA는 관리된 프레임워크를 제공합니다. 즉, 서비스 품질(QoS)이 라이프사이클 관리 및 보안을 트랜잭션에 제공하는 Application Server에 의해 제공됩니다. Java용 IBM IMS 커넥터 및 IBM CICS ECI 자원 어댑터를 제외하고 EMD(Enterprise Metadata Discovery) 스펙도 준수합니다.

이전 어댑터 세트인 WebSphere Business Integration Adapter는 마법사에서도 지원합니다.

Java EE 자원

EIS 모듈 패턴을 따르는 SCA 모듈인 EIS 모듈을 Java EE 플랫폼에 전개할 수 있습니다.

EIS 모듈을 Java EE 플랫폼에 전개하면 실행할 준비가 된 응용프로그램이 EAR 파일로 패키지되어 서버에 전개됩니다. 모든 Java EE 아티팩트 및 자원이 작성되었습니다. 응용프로그램이 구성되어 실행할 준비가 되었습니다.

JCA 상호작용 스펙 및 연결 스펙 동적 특성

EIS 바인딩에서는 적절히 정의된 하위 데이터 오브젝트(페이로드와 공존)를 사용하여 지정된 InteractionSpec 및 ConnectionSpec에 대한 입력을 승인할 수 있습니다. 이는 동적 요청-응답이 InteractionSpec을 통해 자원 어댑터와 상호작용하고 ConnectionSpec을 통해 컴포넌트를 인증할 수 있게 합니다.

javax.cci.InteractionSpec에는 자원 어댑터를 사용하는 상호작용 요청 처리 방법에 대한 정보가 있습니다. 여기에는 요청 이후 상호작용이 이루어진 방법에 대한 정보가 있을 수도 있습니다. 이러한 상호작용을 통한 양방향 통신을 때로는 *통신(conversation)*이라고 부르기도 합니다.

EIS 바인딩에서는 자원 어댑터의 인수가 되는 페이로드에 properties라는 하위 데이터 오브젝트가 포함되어 있을 것으로 예상합니다. 이 특성 데이터 오브젝트에는 특정 형식의 상호작용 스펙 특성 이름이 포함된 이름/값 쌍이 포함됩니다. 형식 지정 규칙은 다음과 같습니다.

- 이름은 접두부 IS로 시작하고 그 뒤에 특성 이름이 와야 합니다. 예를 들어, InteractionId라는 JavaBeans 특성을 가진 상호작용 스펙에서는 특성 이름을 ISInteractionId로 지정합니다.
- 이름/값 쌍은 단순한 유형의 상호작용 스펙 특성의 이름과 값을 표시합니다.

이 예제에서 인터페이스는 조작의 입력이 Account 데이터 오브젝트가 되도록 지정합니다. 이 인터페이스는 값이 xyz인 workingSet라는 동적 InteractionSpec 특성을 전송하고 수신하기 위해 EIS 가져오기 바인딩 응용프로그램을 호출합니다.

서버의 비즈니스 그래프 또는 비즈니스 오브젝트에는 페이로드와 함께 프로토콜에 맞는 데이터를 전송할 수 있는 기본 properties 비즈니스 오브젝트가 포함되어 있습니다. 이 properties 비즈니스 오브젝트는 내장되어 있어 비즈니스 오브젝트 생성 시 XML 스키마에 지정할 필요가 없으며 작성하여 사용하기만 하면 됩니다. XML 스키마를 기반으로 하여 정의한 고유 데이터 유형이 있는 경우에는 예상 이름/값 쌍이 들어 있는 properties 요소를 지정해야 합니다.

```
BOFactory dataFactory = (BOFactory) ₩
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
 //skip to payload for non doc-lit
 DataObject docLitWrapper = dataFactory.createByElement /
 ("http://mytest/eis/Account", "AccountWrapper");
```

페이로드를 작성하십시오.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload


  //Construct properties data for dynamic interaction

  DataObject properties = account.createDataObject("properties");
```

이름 workingSet에 예상 값 xyz를 설정하십시오.

```
properties.setString("ISworkingSet", "xyz");


  //Invoke the service with argument

  Service accountImport = (Service) ₩
  serviceManager.locateService("AccountOutbound");
  DataObject result = accountImport.invoke("createAccount", docLitWrapper);

  //Get returned property
```

```
DataObject retProperties = result.getDataObject("properties");

String workingset = retProperties.getString("ISworkingSet");
```

동적 컴포넌트 인증에 ConnectionSpec 특성을 사용할 수 있습니다. 특성 이름 접두부가 IS가 아닌 CS여야 하는 점만 제외하면 위와 동일한 규칙이 적용됩니다. ConnectionSpec 특성은 양방향이 아닙니다. 동일한 properties 데이터 오브젝트에 IS와 CS 특성 둘 다 있을 수 있습니다.

ConnectionSpec 특성을 사용하려면 가져오기 바인딩에 지정된 resAuth를 Application으로 설정하십시오. 또한 자원 어댑터가 컴포넌트 권한을 지원하는지 확인하십시오. 자세한 내용은 8장의 J2EE 커넥터 아키텍처 스펙을 참조하십시오.

EIS 바인딩을 사용하는 외부 클라이언트

서버는 EIS 바인딩을 사용하여 외부 클라이언트에 메시지를 전송하거나 반대로 수신할 수 있습니다.

웹 포털 또는 EIS와 같은 외부 클라이언트는 서버의 SCA 모듈에 메시지를 전송하거나 서버의 컴포넌트에 의해 호출되어야 합니다.

클라이언트는 기타 응용프로그램과 마찬가지로 동적 호출 인터페이스(DII) 또는 Java 인터페이스를 사용하여 EIS 가져오기를 호출합니다.

1. 외부 클라이언트는 ServiceManager의 인스턴스를 작성하고 해당 참조 이름을 사용하여 EIS 가져오기를 찾습니다. 검색 결과 서비스 인터페이스가 구현됩니다.
2. 클라이언트는 데이터 오브젝트 스키마를 사용하여 동적으로 작성된 입력 인수(일반 데이터 오브젝트)를 작성합니다. 이 단계는 서비스 데이터 오브젝트(SDO) DataFactory 인터페이스 구현을 사용하여 수행됩니다.
3. 외부 클라이언트는 EIS를 호출하여 필요한 결과를 얻습니다.

또는 클라이언트는 Java 인터페이스를 사용하여 EIS 가져오기를 호출할 수 있습니다.

1. 클라이언트는 ServiceManager의 인스턴스를 작성하고 해당 참조 이름을 사용하여 EIS 가져오기를 찾습니다. 검색 결과 EIS 가져오기의 Java 인터페이스가 구현됩니다.
2. 클라이언트는 입력 인수 및 입력한 데이터 오브젝트를 작성합니다.
3. 클라이언트는 EIS를 호출하여 필요한 결과를 얻습니다.

EIS 내보내기 인터페이스는 내보낸 SCA 컴포넌트의 인터페이스를 정의하며 외부 EIS 응용프로그램에서 이 인터페이스를 사용할 수 있습니다. 이 인터페이스는 EIS 내보내기 응용프로그램 런타임 구현을 통해 SAP 또는 PeopleSoft와 같은 외부 응용프로그램이 호출하는 인터페이스로 간주될 수 있습니다.

내보내기에서는 EISExportBinding을 사용하여 내보낸 서비스를 외부 EIS 응용프로그램에 바인드합니다. 이를 통해 SCA 모듈에 포함된 응용프로그램을 등록하여 EIS 서비스 요청을 청취할 수 있습니다. EIS 내보내기 바인딩은 Java EE 커넥터 아키텍처 인터페이스를 사용하여 자원 어댑터가 인식한 인바운드 이벤트의 정의와 SCA 조작 호출 간의 맵핑을 지정합니다.

EISExportBinding에서는 외부 EIS 서비스가 Java EE 커넥터 아키텍처 1.5 인바운드 계약을 기반으로 해야 합니다. EISExportBinding에서는 바인딩 레벨 또는 메소드 레벨에서 데이터 핸들러 또는 데이터 바인딩을 지정해야 합니다.

JMS 바인딩

JMS(Java Message Service) 프로바이더는 Java Messaging Service API 및 프로그래밍 모델을 기반으로 메시징을 사용 가능하게 합니다. 이는 JMS 대상에 대한 연결을 작성하고 메시지를 전송 및 수신하기 위해 Java EE 연결 팩토리를 제공합니다.

세 개의 JMS 바인딩이 제공됩니다.

- JMS JCA 1.5와 호환되는 서비스 통합 버스(SIB) 프로바이더 바인딩(*JMS 바인딩*)
- JMS 1.1과 호환되는 일반, 비JCA JMS 바인딩(*일반 JMS 바인딩*)
- WebSphere MQ에 JMS 프로바이더 지원을 제공하고 Java EE 응용프로그램 상호운영성을 허용하는 WebSphere MQ JMS 바인딩(*WebSphere MQ JMS 바인딩*)

JMS 내보내기 및 가져오기 바인딩은 SCA(Service Component Architecture) 모듈이 외부 JMS 시스템을 호출하고 외부 JMS 시스템에서 메시지를 수신할 수 있게 합니다.

또한 WebSphere MQ 바인딩(*WebSphere MQ 바인딩*)이 지원되어 기본 MQ 사용자가 무작위 수신 및 송신 메시지 형식을 처리할 수 있게 합니다(WebSphere MQ가 필요함).

JMS 가져오기 및 내보내기 바인딩은 WebSphere Application Server의 일부인 JCA 1.5 기반 SIB JMS 프로바이더를 사용하여 JMS 응용프로그램과의 통합을 제공합니다. 다른 JCA 1.5 기반 JMS 자원 어댑터는 지원되지 않습니다.

JMS 바인딩 개요

JMS 바인딩은 SCA(Service Component Architecture) 환경과 JMS 시스템 간에 연결을 제공합니다.

JMS 바인딩

JMS 가져오기 및 JMS 내보내기 바인딩 둘 다의 주요 컴포넌트는 다음과 같습니다.

- 자원 어댑터: SCA 모듈과 외부 JMS 시스템 간에 관리된 양방향 연결을 사용 가능하게 합니다.
- 연결: 클라이언트와 프로바이더 응용프로그램 간의 가상 연결을 캡슐화합니다.

- 대상: 클라이언트가 생성하는 메시지의 대상 또는 사용하는 메시지의 소스를 지정하기 위해 사용합니다.
- 인증 데이터: 바인딩에 대한 보안 액세스에 사용됩니다.

JMS 가져오기 바인딩

JMS 가져오기 바인딩을 사용하면 SCA 모듈 내부에서 사용할 외부 JMS 응용프로그램을 가져올 수 있습니다. JMS 가져오기 바인딩은 SCA 모듈 내의 컴포넌트가 외부 JMS 응용프로그램에서 제공하는 서비스와 통신할 수 있게 합니다.

JMS 연결 팩토리를 사용하여 JMS 대상과 연관된 JMS 프로바이더에 대한 연결을 작성합니다. 기본 메시징 프로바이더의 JMS 연결 팩토리를 관리하려면 연결 팩토리 관리 오브젝트를 사용하십시오.

외부 JMS 시스템과의 상호작용은 요청을 송신하고 응답을 수신하는 대상의 사용을 포함합니다.

호출되는 조작 유형에 따라 두 가지 유형의 JMS 가져오기 바인딩 사용법 시나리오가 지원됩니다.

- 단방향: JMS 가져오기는 가져오기 바인딩에서 구성된 전송 대상에 메시지를 위치시킵니다. JMS 헤더의 replyTo 필드에는 아무 것도 설정되지 않습니다.
- 양방향(요청-응답): JMS 가져오기는 전송 대상에 메시지를 위치시킨 후 SCA 컴포넌트에서 수신하는 응답을 지속시킵니다.

  가져오기 바인딩이 구성되어(**응답 상관 설계** 필드를 WebSphere Integration Developer에서 사용) 응답 메시지 상관 ID가 요청 메시지 ID(기본값)나 요청 메시지 상관 ID에서 복사되었음을 예상할 수 있습니다.가져오기 바인딩은 임시 동적 응답 대상을 사용하여 응답을 요청과 관련시키도록 구성될 수도 있습니다. 임시 대상은 모든 요청에 대해 작성되고 가져오기는 응답을 수신하기 위해 이 대상을 사용합니다.

  아웃바운드 메시지의 replyTo 헤더 특성에 receive 대상이 설정됩니다. 수신 대상에서 청취를 위해 메시지 리스너가 전개되며 응답이 수신되면 메시지 리스너가 응답을 다시 컴포넌트로 전달합니다.

단방향 및 양방향 사용법 시나리오 둘 다의 경우 동적 및 정적 헤더 특성을 지정할 수 있습니다. 정적 특성은 JMS 가져오기 메소드 바인딩에서 설정될 수 있습니다. 이러한 특성 중 일부는 SCA JMS 런타임에서 특별한 의미를 갖습니다.

JMS가 비동기 바인딩임을 참고하십시오. 호출 컴포넌트가 JMS 가져오기를 동기적으로 호출하면(양방향 조작의 경우), JMS 서비스가 응답을 리턴할 때까지 호출 컴포넌트가 블록화됩니다.

그림 33에서는 가져오기가 외부 서비스에 링크되어 있는 방식을 표시합니다.

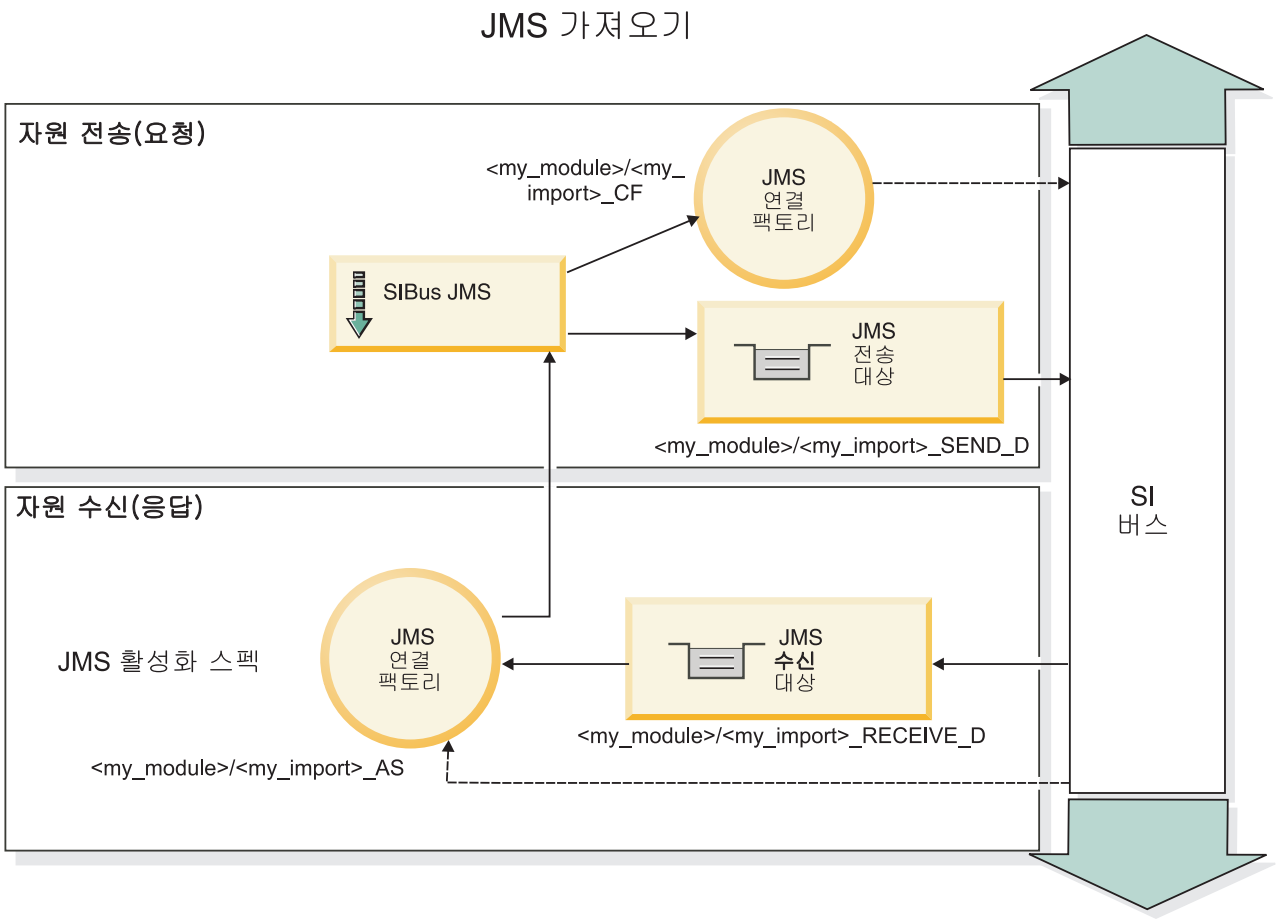


그림 33. JMS 가져오기 바인딩 자원

JMS 내보내기 바인딩

JMS 내보내기 바인딩은 외부 JMS 응용프로그램으로 서비스를 제공하기 위한 SCA 모듈 수단을 제공합니다.

JMS 내보내기의 일부인 연결은 구성 가능한 활성화 스펙입니다.

JMS 내보내기에는 전송 및 수신 대상이 있습니다.

- receive 대상은 대상 컴포넌트의 수신 메시지를 두는 장소입니다.
- send 대상은 수신 메시지가 replyTo 헤더 특성을 사용하여 대체하지 않는 한 응답이 전송되는 장소입니다.

내보내기 바인딩에 지정된 receive 대상으로 들어오는 요청을 청취하기 위해 메시지 리스너가 전개됩니다. 호출한 컴포넌트가 응답을 제공하는 경우 send 필드에서 지정한 대

상이 인바운드 요청에 응답을 전송하는 데 사용됩니다. 수신 메시지의 replyTo 필드에서 지정한 대상이 send에서 지정한 대상을 대체합니다.

그림 34에서는 외부 요청자가 내보내기에 링크되어 있는 방식을 표시합니다.

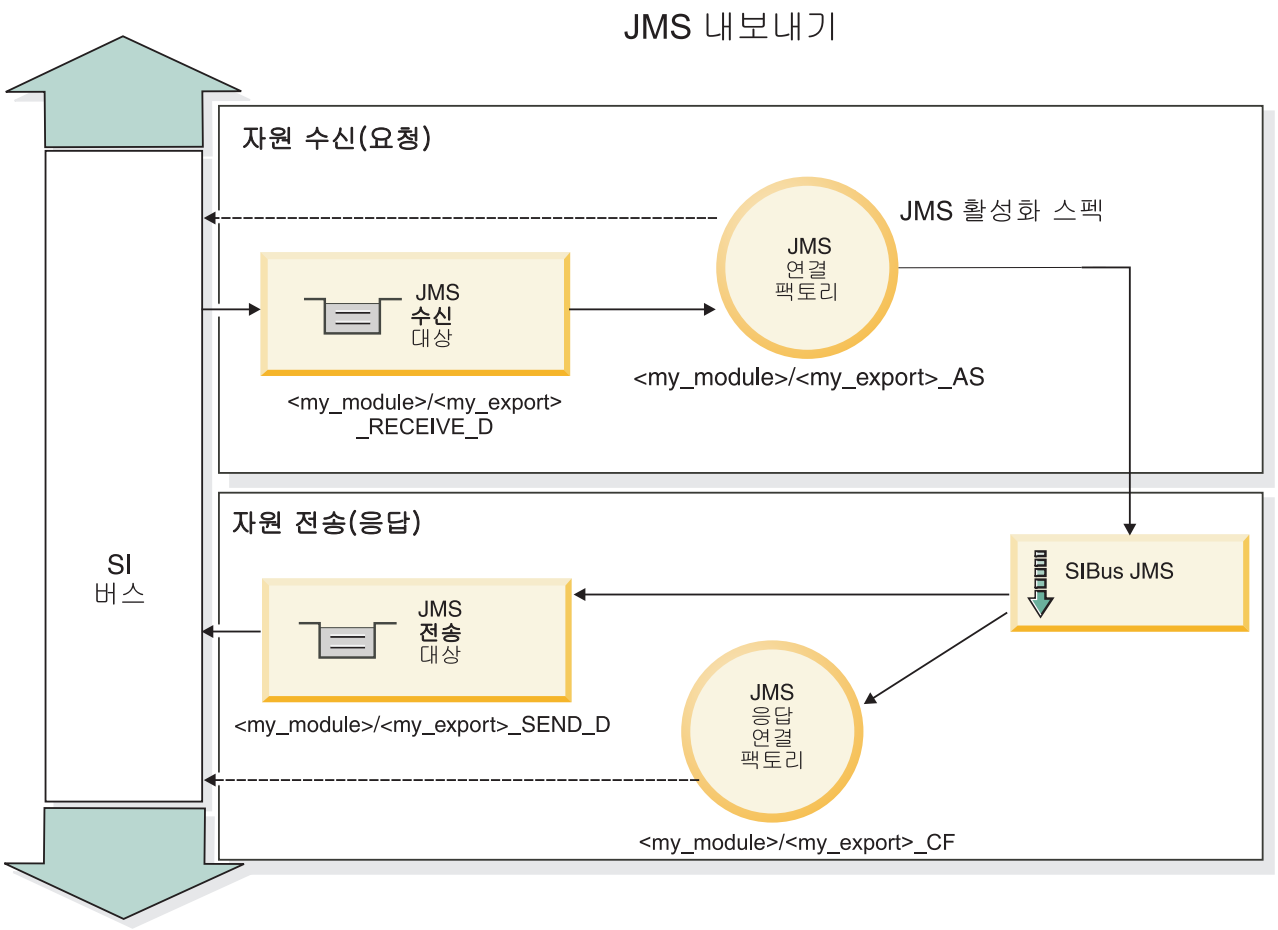


그림 34. JMS 내보내기 바인딩 자원

JMS 통합 및 자원 어댑터

JMS(Java Message Service)는 사용 가능한 JMS JCA 1.5 기반 자원 어댑터를 통한 통합을 제공합니다. 서비스 통합 버스(SIB) JMS 자원 어댑터에 JMS 통합을 위한 완전한 지원이 제공됩니다.

외부 JCA 1.5 호환 JMS 시스템과 통합하려면 JCA 1.5 자원 어댑터용 JMS 프로바이더를 사용하십시오. JCA 1.5와 호환되는 외부 서비스는 SIB JMS 자원 어댑터를 사용하여 SCA(Service Component Architecture) 컴포넌트와 통합하도록 메시지를 수신 및 전송할 수 있습니다.

기타 프로바이더에 맞는 JCA 1.5 자원 어댑터 사용은 지원되지 않습니다.

JMS 모듈을 Java SE 환경에 전개할 수 없습니다. 이러한 모듈은 Java EE 환경에만 전개 가능합니다.

JMS 바인딩의 주요 기능

JMS 가져오기 및 내보내기 바인딩의 주요 기능에는 헤더 및 작성된 Java EE 자원이 포함됩니다.

특수 헤더

특수 헤더 특성은 JMS 가져오기 및 내보내기에서 메시지 처리 방법을 대상에 알리는 데 사용됩니다.

예를 들어, TargetFunctionName은 기본 메소드에서 조작 메소드로 맵핑합니다.

Java EE 자원

JMS 가져오기 및 내보내기가 Java EE 환경에 전개되면 여러 Java EE 자원이 작성됩니다.

ConnectionFactory
클라이언트가 JMS 프로바이더에 연결을 작성하는 데 사용됩니다.

ActivationSpec
가져오기는 요청에 대한 응답을 수신하기 위해 이 자원을 사용합니다. 내보내기는 메시징 시스템과 상호작용할 때 메시지 리스너를 나타내는 메시지 엔드포인트를 구성하는 경우에 이 자원을 사용합니다.

대상

- 전송 대상: 가져오기의 경우 요청 또는 송신 메시지를 전송하는 곳입니다. 내보내기의 경우에는 응답 메시지를 전송할 대상입니다(수신 메시지에서 JMSReplyTo 헤더 필드로 대체되지 않은 경우).
- 수신 대상: 수신 메시지를 배치할 장소입니다. 수신 메시지는 응답이고 내보내기에서는 요청입니다.
- 콜백 대상: 상관 정보를 저장하는 데 사용되는 SCA JMS 시스템 대상입니다. 이 대상에서 읽거나 쓰지 마십시오.

설치 타스크에서는 ConnectionFactory 및 세 개의 대상을 작성합니다. 또한 ActivationSpec을 작성하여 런타임 메시지가 수신 대상에서 응답을 청취할 수 있게 합니다. 이들 자원의 특성은 가져오기 또는 내보내기 파일에 지정되어 있습니다.

JMS 헤더

JMS 메시지에는 JMS 시스템 헤더 및 다중 JMS 특성이라는 두 가지 유형의 헤더가 포함됩니다. 양쪽 유형의 헤더는 SMO(Service Message Object)의 중개 모듈이나 ContextService API를 사용하여 액세스될 수 있습니다.

JMS 시스템 헤더

JMS 시스템 헤더는 JMSHeader 요소에 의해 SMO에 표시되며, 여기에는 일반적으로 JMS 헤더에 있는 모든 필드가 포함되어 있습니다. 이들 필드는 중개(또는 ContextService)에서 수정될 수 있지만, SMO에서 설정되는 일부 JMS 시스템 헤더 필드는 시스템이나 정적 값으로 대체될 때 아웃바운드 JMS 메시지에서 전파되지 않습니다.

중개(또는 ContextService)에서 갱신될 수 있는 JMS 시스템 헤더의 키 필드는 다음과 같습니다.

- **JMSType** 및 **JMSCorrelationID** – 사전 정의된 특정 메시지 헤더 특성 값
- **JMSDeliveryMode** – 전달 모드의 값(지속적 또는 비지속적, 기본값은 지속적)
- **JMSPriority** – 우선순위 값(0 - 9; 기본값은 JMS_Default_Priority)

JMS 특성

JMS 특성은 특성 목록의 항목으로 SMO에 표시됩니다. 특성은 중개에서 또는 ContextService API를 사용하여 추가, 갱신 또는 삭제될 수 있습니다.

또한 특성은 JMS 바인딩에서 정적으로 설정될 수 있습니다. 정적으로 설정되는 특성은 동적으로 설정된 설정(동일한 이름을 가짐)을 대체합니다.

다른 바인딩(예: HTTP 바인딩)에서 전파된 사용자 특성은 JMS 특성으로 JMS 바인딩에서 출력됩니다.

헤더 전파 설정

인바운드 JMS 메시지에서 다운스트림 컴포넌트로 또는 업스트림 컴포넌트에서 아웃바운드 JMS로의 JMS 시스템 헤더 및 특성의 전파는 바인딩에서 전파 프로토콜 헤더 플래그에 의해 제어될 수 있습니다.

전파 프로토콜 헤더가 설정되면, 다음 목록에서 설명한 대로 헤더 정보는 메시지나 대상 컴포넌트로 플로우됩니다.

- JMS 내보내기 요청

  메시지에 수신된 JMS 헤더는 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다. 메시지에 수신된 JMS 특성은 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다.

- JMS 내보내기 응답

  컨텍스트 서비스에 설정된 모든 JMS 헤더 필드는 JMS 내보내기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다. 컨텍스트 서비스의 모든 특성 세트는 JMS 내보내기 바인딩의 정적 특성 세트로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다.

- JMS 가져오기 요청

  컨텍스트 서비스에 설정된 모든 JMS 헤더 필드는 JMS 가져오기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다. 컨텍스트 서비스에 설정된 모든 특성은 JMS 가져오기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다.

- JMS 가져오기 응답

  메시지에 수신된 JMS 헤더는 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다. 메시지에 수신된 JMS 특성은 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다.

JMS 임시 동적 응답 대상 상관 설계

임시 동적 응답 대상 상관 설계는 전송되는 요청마다 고유한 동적 큐 또는 토픽이 작성되도록 합니다.

가져오기에 지정된 정적 응답 대상은 임시 동적 대상 큐 또는 토픽의 네이처를 파생시키기 위해 사용됩니다. 이는 요청의 **ReplyTo** 필드에서 설정되며 JMS 가져오기는 해당 대상에서 응답을 청취합니다. 응답이 수신되면 비동기 처리를 위해 정적 응답 대상 큐에 다시 넣어집니다. 응답의 **CorrelationID** 필드는 사용되지 않으므로 설정하지 않아도 됩니다.

트랜잭션 문제

임시 동적 대상이 사용될 경우 응답은 전송된 응답과 같은 스레드에서 이용되어야 합니다. 요청은 글로벌 트랜잭션 외부에서 전송되어야 하며, 백엔드 서비스에서 수신되고 응답이 리턴되기 전에 커미트해야 합니다.

지속

임시 동적 큐는 수명이 짧은 엔티티이므로 정적 큐 또는 토픽과 연관되는 지속성의 동일 레벨이 보장되지 않습니다. 임시 동적 큐 또는 토픽은 서버 다시 시작과 메시지 발생 시 유지되지 못합니다. 메시지가 정적 응답 대상 큐에 다시 넣어지면 메시지에 정의된 지속을 유지합니다.

제한시간 초과

가져오기는 고정 시간 동안 임시 동적 응답 대상에 대해 응답을 수신하길 기다립니다. 이 시간 간격은 SCA 응답 만기 시간 규정자(설정된 경우)에서 가져옵니다. 설정되지 않으면 기본 시간인 60초가 사용됩니다. 대기 시간을 초과하면 가져오기는 `ServiceTimeoutRuntimeException`을 발생시킵니다.

외부 클라이언트

서버는 JMS 바인딩을 사용하여 외부 클라이언트에 메시지를 전송하거나 반대로 수신할 수 있습니다.

외부 클라이언트(예: 웹 포털 또는 엔터프라이즈 정보 시스템)가 서버의 SCA 모듈에 메시지를 전송하거나 서버의 컴포넌트에서 메시지를 호출할 수 있습니다.

JMS 내보내기 컴포넌트에서는 메시지 리스너를 전개하여 내보내기 바인딩에서 지정한 수신 대상에 수신되는 요청을 청취합니다. 호출한 응용프로그램이 응답을 제공하는 경우 인바운드 요청에 응답을 전송하기 위해 전송 필드에서 지정한 대상을 사용합니다. 따라서 외부 클라이언트가 내보내기 바인딩을 사용하여 응용프로그램을 호출할 수 있습니다.

JMS는 외부 클라이언트에 바인드를 가져오고 메시지를 전달할 수 있습니다. 해당 메시지는 외부 클라이언트의 응답을 요구하거나 그렇지 않을 수도 있습니다.

외부 클라이언트에 대한 작업:

외부 클라이언트, 즉 서버 외부에 있는 클라이언트가 서버에 설치된 응용프로그램과 상호작용해야 할 수도 있습니다.

이 태스크 정보

외부 클라이언트가 서버의 한 응용프로그램과 상호작용하려고 하는 매우 단순한 시나리오를 검토합니다. 그림은 일반적인 단순 시나리오를 나타냅니다.

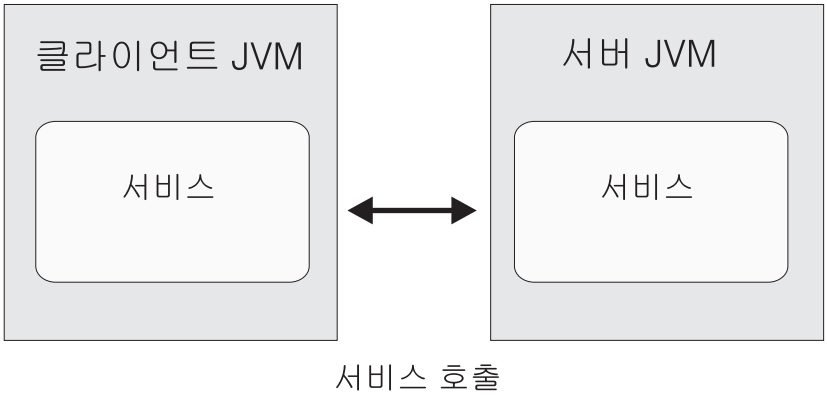


그림 35. 단순 사용 사례 시나리오: 외부 클라이언트와 서버 응용프로그램의 상호작용

SCA 응용프로그램은 JMS 바인딩에 내보내기를 포함합니다. 이렇게 하면 응용프로그램을 외부 클라이언트에서 사용할 수 있습니다.

서버와는 별개의 JVM(Java Virtual Machine)에 외부 클라이언트가 있는 경우 JMS 내보내기와 연결하고 상호작용하려면 몇 가지 단계를 수행해야 합니다. 클라이언트는 올바른 값이 있는 InitialContext를 획득한 다음 JNDI를 통해 자원을 찾습니다. 그런 다음, 클라이언트는 JMS 1.1 스펙 클라이언트를 사용하여 대상과 대상의 송신 및 수신 메시지에 액세스합니다.

런타임에서 자동으로 작성한 자원의 기본 JNDI 이름이 이 섹션의 구성 주제에 있습니다. 그러나 사전에 작성한 자원이 있는 경우에는 해당 JNDI 이름을 사용하십시오.

프로시저

1. JMS 대상 및 연결 팩토리를 구성하여 메시지를 전송하십시오.
2. JNDI 컨텍스트, SIB 자원 어댑터의 포트 및 메시징 부트스트랩 포트가 올바른지 확인하십시오.

   서버는 몇몇 기본 포트를 사용하지만 해당 시스템에 설치된 서버가 더 있는 경우에는 설치 시 대체 포트가 작성되어 다른 서버 인스턴스와 충돌하지 않도록 합니다. 관리 콘솔을 사용하여 서버에서 사용할 포트를 결정할 수 있습니다. **서버** → **Application Server** → ***your_server_ name*** → **구성**으로 이동한 다음 **통신** 아래 있는 **포트**를 클릭하십시오. 그러면 사용 중인 포트를 편집할 수 있습니다.
3. 클라이언트는 올바른 값을 가진 초기 컨텍스트를 획득한 다음 JNDI를 통해 자원을 찾습니다.
4. 클라이언트는 JMS 1.1 스펙을 사용하여 대상과 대상의 송신 및 수신 메시지에 액세스합니다.

JMS 바인딩 문제점 해결

JMS 바인딩에 대한 문제점을 진단하여 수정할 수 있습니다.

구현 예외

다양한 오류 조건에 대한 응답으로, JMS 가져오기 및 내보내기 구현에서는 두 가지 유형의 예외 중 하나가 리턴될 수 있습니다.

- 서비스 비즈니스 예외: 이 예외는 서비스 비즈니스 인터페이스(WSDL 포트 유형)에 지정된 결함이 발생한 경우에 리턴됩니다.
- 서비스 런타임 예외: 다른 모든 경우에 발생합니다. 대부분의 경우, cause 예외에는 원래 예외(JMSException)가 포함됩니다.

  예를 들어, 가져오기에서는 각 요청 메시지에 대해 하나의 응답 메시지만 예상합니다. 응답이 두 개 이상 도달하거나 늦은 응답(SCA 응답 만기가 만료됨)이 도달하면 서비스 런타임 예외가 처리됩니다. 트랜잭션이 롤백되고 응답 메시지가 큐에서 제거되거나 실패 이벤트 관리자에 의해 처리됩니다.

1차 실패 조건

JMS 바인딩의 1차 실패 조건은 트랜잭션 시맨틱, JMS 프로바이더 구성 또는 다른 컴포넌트에서 기존 작동에 대한 참조로 판별됩니다. 1차 실패 조건은 다음과 같습니다.

- JMS 프로바이더 또는 대상에 대한 연결 실패

  메시지 수신을 위한 JMS 프로바이더에 대한 연결 실패의 결과로 메시지 리스너 시작에 실패합니다. 이 조건은 WebSphere Application Server 로그에 로깅됩니다. 지속적 메시지는 성공적으로 검색(또는 만기)될 때까지 대상에 남아 있습니다.

  아웃바운드 메시지를 전송하기 위해 JMS 프로바이더에 연결하는 데 실패하면 전송을 제어하는 트랜잭션이 롤백됩니다.
- 인바운드 메시지 구문 분석 또는 아웃바운드 메시지 생성 실패

  데이터 바인딩 또는 데이터 핸들러에서 실패가 발생하면 작업을 제어하는 트랜잭션이 롤백됩니다.
- 아웃바운드 메시지 전송 실패

  메시지 전송에 실패하면 관련 트랜잭션이 롤백됩니다.
- 다중 또는 예기치 않은 늦은 응답 메시지

  가져오기에서는 각 요청 메시지에 대해 하나의 응답 메시지만 예상합니다. 또한 응답을 수신할 수 있는 유효한 기간은 요청 시 SCA 응답 만기 규정자에 의해 판별됩니다. 응답이 도달하거나 만기 시간이 초과되면 상관 레코드가 삭제됩니다. 응답 메시지가 예상외로 도달하거나 늦게 도달하면 서비스 런타임 예외가 처리됩니다.
- 임시 동적 응답 대상 상관 설계를 사용할 때 늦은 응답으로 야기된 서비스 제한시간 런타임 예외

JMS 가져오기는 SCA 응답 만기 규정자가 판별한 기간 이후에 제한시간이 초과되거나, 설정되지 않은 경우 60초가 기본값이 됩니다.

JMS 기반 SCA 메시지가 실패 이벤트 관리자에 표시되지 않음

SCA 메시지가 원래 JMS 상호작용 실패를 통해 발생된 경우, 실패 이벤트 관리자에서 이 메시지를 볼 수 있습니다. 이와 같은 메시지가 실패 이벤트 관리자에 표시되지 않으면, JMS 대상의 기본적인 SIB 대상에서 실패한 최대 전달 수의 값이 2 이상인지 확인하십시오. 이 값을 2 이상으로 설정하면 JMS 바인딩에 대한 SCA 호출 중 실패 이벤트 관리자와의 상호작용이 사용 가능하게 됩니다.

예외 처리

바인딩이 구성되는 방식은 데이터 핸들러 또는 데이터 바인딩에 의해 발생되는 예외가 처리되는 방식을 판별합니다. 또한 중개 플로우의 특성이 예외가 발생할 때 시스템의 동작을 기술합니다.

데이터 핸들러 또는 데이터 바인딩이 사용자 바인딩에 의해 호출될 때 여러 가지 문제점이 발생할 수 있습니다. 예를 들어, 데이터 핸들러는 손상 페이로드가 있는 메시지를 수신하거나 잘못된 형식이 있는 메시지를 읽으려고 시도합니다.

바인딩이 예외를 처리하는 방법은 데이터 핸들러 또는 데이터 바인딩을 구현하는 방법에 의해 결정됩니다. 권장되는 동작은 `DataBindingException`을 처리하는 데이터 바인딩을 설계하는 것입니다.

`DataBindingException`을 포함한 런타임 예외가 발생할 때:

- 중개 플로우가 트랜잭션이도록 구성되는 경우, 기본값으로 JMS 메시지 가 수동 재생이나 삭제를 위해 실패 이벤트 관리자에 저장됩니다.

  주: 실패 이벤트 관리자에 메시지를 저장하는 대신 롤백하도록 바인딩에서 복구 모드를 변경할 수 있습니다.
- 중개 플로우가 트랜잭션이 아닌 경우 예외가 로그되고 메시지가 유실됩니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 모든 데이터 핸들러 예외는 데이터 바인딩 예외에 포함됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

일반 JMS 바인딩

일반 JMS 바인딩은 써드파티 JMS 1.1 호환 프로바이더에 대한 연결을 제공합니다. 일반 JMS 바인딩은 JMS 바인딩과 유사합니다.

JMS 바인딩을 통해 제공되는 서비스를 사용하면 SCA(Service Component Architecture) 모듈이 외부 시스템을 호출하거나 외부 시스템에서 메시지를 수신할 수 있습니다. 시스템은 외부 JMS 시스템일 수 있습니다.

일반 JMS 바인딩을 사용하면 JMS 1.1을 지원하고 선택적 JMS Application Server 기능을 구현하는 비JCA 1.5 호환 JMS 프로바이더와 통합할 수 있습니다. 일반 JMS 바인딩은 JCA 1.5를 지원하지는 않지만 JMS 1.1 스펙의 Application Server 기능을 지원하는 JMS 프로바이더를 지원하는 JMS 프로바이더(Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic 및 WebSphere MQ)를 지원합니다. JCA 1.5 JMS 프로바이더인 WebSphere 임베디드 JMS 프로바이더(SIBJMS)는 이 바인딩에서 지원되지 않습니다. 해당 프로바이더를 사용하는 경우 78 페이지의 『JMS 바인딩』을 사용하십시오.

SCA 환경에서 비JCA 1.5 호환 JMS 기반 시스템과 통합할 때 이 일반 바인딩을 사용하십시오. 그러면 대상 외부 응용프로그램이 메시지를 수신 및 송신하여 SCA 컴포넌트와 통합할 수 있습니다.

일반 JMS 바인딩 개요

일반 JMS 바인딩은 SCA(Service Component Architecture) 환경과 JMS 시스템(JMS 1.1과 호환되고 선택적 JMS Application Server 기능을 구현) 간에 연결을 제공하는 비JCA JMS 바인딩입니다.

일반 JMS 바인딩

일반 JMS 가져오기 및 내보내기 바인딩의 주요 사항은 다음과 같습니다.

- 리스너 포트: 비JCA 기반 JMS 프로바이더가 메시지를 수신하고 이를 메시지 구동 Bean(MDB)에 디스패치 할 수 있도록 합니다.
- 연결: 클라이언트와 프로바이더 응용프로그램 간의 가상 연결을 캡슐화합니다.
- 대상: 클라이언트가 생성하는 메시지의 대상 또는 사용하는 메시지의 소스를 지정하기 위해 사용합니다.
- 인증 데이터: 바인딩에 대한 보안 액세스에 사용됩니다.

일반 JMS 가져오기 바인딩

일반 JMS 가져오기 바인딩은 SCA 모듈 내의 컴포넌트가 외부 비JCA 1.5 호환 JMS 프로바이더에서 제공하는 서비스와 통신할 수 있게 합니다.

JMS 가져오기의 연결 파트는 연결 팩토리입니다. 클라이언트가 프로바이더에 연결을 작성하는 데 사용하는 오브젝트인 연결 팩토리는 관리자가 정의한 연결 구성 매개변수 세트를 캡슐화합니다. 각 연결 팩토리는 ConnectionFactory, QueueConnectionFactory 또는 TopicConnectionFactory 인터페이스의 인스턴스입니다.

외부 JMS 시스템과의 상호작용은 요청을 송신하고 응답을 수신하는 대상의 사용을 포함합니다.

호출되는 조작 유형에 따라 두 가지 유형의 일반 JMS 가져오기 바인딩 사용법 시나리오가 지원됩니다.

- 단방향: 일반 JMS 가져오기는 가져오기 바인딩에서 구성된 전송 대상에 메시지를 위치시킵니다. JMS 헤더의 replyTo 필드에는 아무 것도 전송되지 않습니다.
- 양방향(요청-응답): 일반 JMS 가져오기는 전송 대상에 메시지를 위치시킨 후 SCA 컴포넌트에서 수신하는 응답을 지속시킵니다.

  아웃바운드 메시지의 replyTo 헤더 특성에 receive 대상이 설정됩니다. 수신 대상에서 청취를 위해 메시지 구동 Bean(MDB)이 전개되며 응답이 수신되면 MDB가 응답을 다시 컴포넌트로 전달합니다.

  가져오기 바인딩이 구성되어(**응답 상관 설계** 필드를 WebSphere Integration Developer에서 사용) 응답 메시지 상관 ID가 요청 메시지 ID(기본값)나 요청 메시지 상관 ID에서 복사되었음을 예상할 수 있습니다.

단방향 및 양방향 사용법 시나리오 둘 다의 경우 동적 및 정적 헤더 특성을 지정할 수 있습니다. 정적 특성은 일반 JMS 가져오기 메소드 바인딩에서 설정될 수 있습니다.

일반 JMS가 비동기 바인딩임을 참고하십시오. 호출 컴포넌트가 일반 JMS 가져오기를 동기적으로 호출하면(양방향 조작의 경우), JMS 서비스가 응답을 리턴할 때까지 호출 컴포넌트가 블록화됩니다.

91 페이지의 그림 36에서는 가져오기가 외부 서비스에 링크되어 있는 방식을 표시합니다.

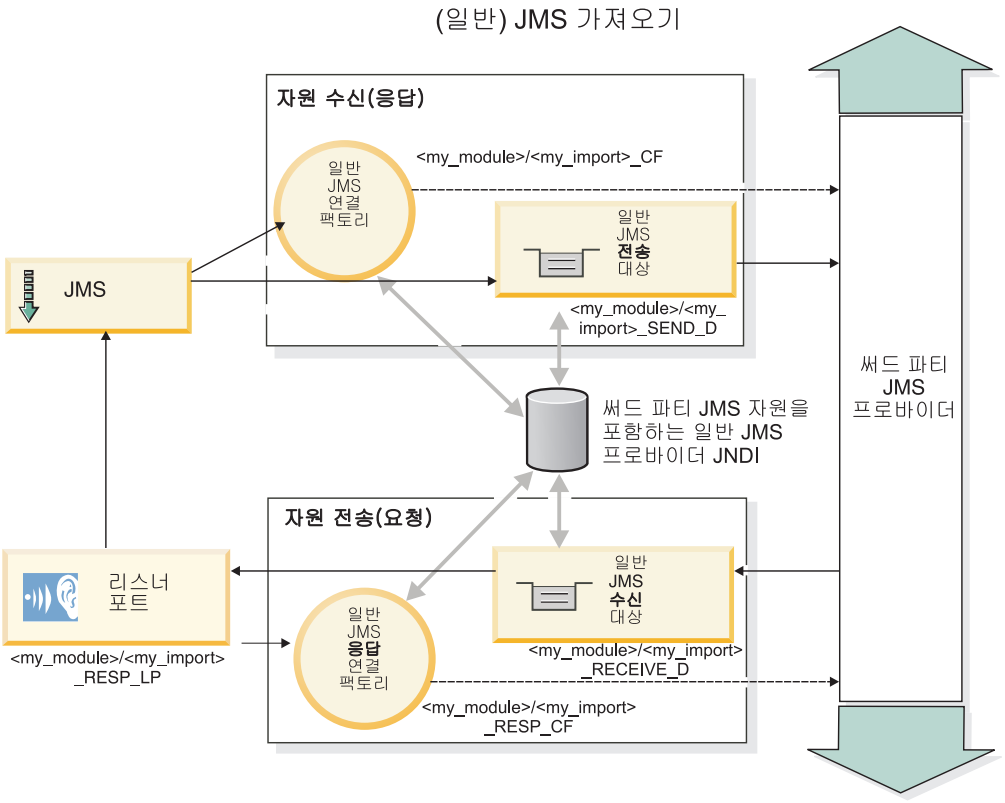


그림 36. 일반 JMS 가져오기 바인딩 자원

일반 JMS 내보내기 바인딩

일반 JMS 내보내기 바인딩은 외부 JMS 응용프로그램으로 서비스를 제공하기 위한 SCA 모듈 수단을 제공합니다.

JMS 내보내기의 연결 파트는 ConnectionFactory 및 ListenerPort로 구성됩니다.

일반 JMS 내보내기에는 전송 및 수신 대상이 있습니다.

- receive 대상은 대상 컴포넌트의 수신 메시지를 두는 장소입니다.
- send 대상은 수신 메시지가 replyTo 헤더 특성을 사용하여 대체하지 않는 한 응답이 전송되는 장소입니다.

내보내기 바인딩에서 지정한 receive 대상으로 들어오는 요청을 청취하기 위해 MDB가 전개됩니다.

- 호출한 컴포넌트가 응답을 제공하는 경우 send 필드에서 지정한 대상이 인바운드 요청에 응답을 전송하는 데 사용됩니다.
- 수신 메시지의 replyTo 필드에서 지정한 대상이 send 필드에서 지정한 대상을 대체합니다.
- 요청/응답 시나리오의 경우 가져오기 바인딩을 구성하여(WebSphere Integration Developer에서 **응답 상관 설계** 필드 사용) 응답이 응답 메시지의 correlation ID 필

드에 요청 message ID를 복사하거나 응답이 응답 메시지의 correlation ID 필드에 요청 correlation ID를 복사할 수 있습니다.

그림 37에서는 외부 요청자가 내보내기에 링크되어 있는 방식을 표시합니다.

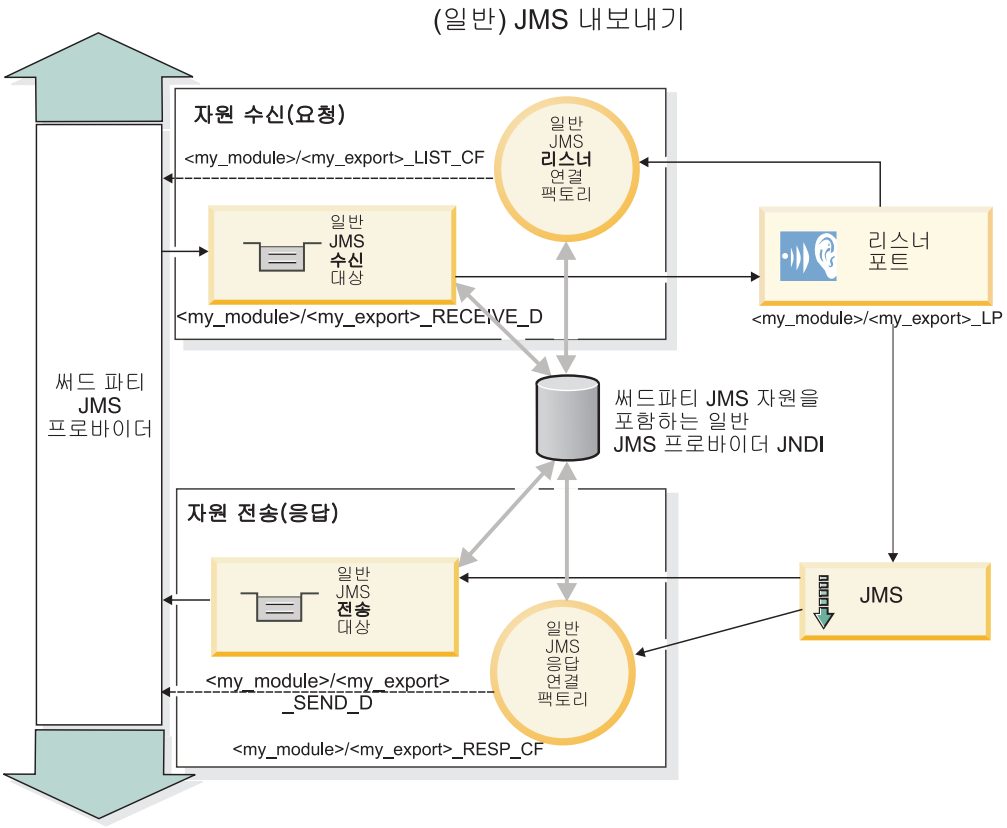


그림 37. 일반 JMS 내보내기 바인딩 자원

일반 JMS 바인딩의 주요 기능

일반 JMS 가져오기 및 내보내기 바인딩의 기능은 WebSphere 임베디드 JMS 및 MQ JMS 가져오기 바인딩의 기능과 일관됩니다. 주요 기능에는 헤더 정의 및 기존 Java EE 자원에 대한 액세스가 포함됩니다. 그러나 이는 일반 바인딩이므로 JMS 프로바이더에 따라 다른 연결 옵션이 제공되지는 않으며 이 바인딩에는 전개 및 설치 시 자원을 생성하는 기능에 한계가 있습니다.

일반 가져오기

MQ JMS 가져오기 응용프로그램과 마찬가지로 일반 JMS 구현은 비동기적이며 단방향, 양방향(요청-응답으로도 알려져 있음) 및 콜백의 세 가지 호출을 지원합니다.

JMS 가져오기가 전개되면 런타임 환경에서 제공하는 메시지 구동 Bean(MDB)이 전개됩니다. MDB는 요청 메시지에 대한 응답을 청취합니다. MDB는 JMS 메시지의 replyTo 헤더 필드에서 요청과 함께 전송된 대상과 연관이 있습니다(대상에서 청취함).

일반 내보내기

일반 JMS 내보내기 바인딩은 결과 리턴을 처리하는 면에서 EIS 내보내기 바인딩과 다릅니다. 일반 JMS 내보내기는 수신 메시지에서 지정된 replyTo 대상으로 응답을 명시적으로 전송합니다. 지정된 대상이 없는 경우 전송 대상이 사용됩니다.

일반 JMS 내보내기가 전개되면 메시지 구동 Bean(일반 JMS 가져오기에서 사용한 것과 다른 MDB)이 전개됩니다. 이 MDB는 수신 대상에서 수신 요청을 청취한 다음 SCA 런타임에서 요청을 처리하도록 디스패치합니다.

특수 헤더

특수 헤더 특성은 일반 JMS 가져오기 및 내보내기에서 메시지 처리 방법을 대상 바인딩에 알리는 데 사용됩니다.

예를 들어, 기본 함수 선택기가 TargetFunctionName 특성을 사용하여 호출 중인 내보내기 인터페이스에서 조작의 이름을 식별합니다.

주: 가져오기 바인딩은 TargetFunctionName 헤더를 각 조작의 조작 이름으로 설정하도록 구성할 수 있습니다.

Java EE 자원

JMS 바인딩이 Java EE 환경에 전개되면 여러 Java EE 자원이 작성됩니다.

- 가져오기의 수신(응답) 대상(양방향 전용) 및 내보내기의 수신(요청) 대상에서 청취에 사용되는 리스너 포트
- outboundConnection(가져오기) 및 inboundConnection(내보내기)에 사용되는 일반 JMS 연결 팩토리
- 전송(가져오기) 및 수신(내보내기, 양방향 전용) 대상의 일반 JMS 대상
- responseConnection에 사용되는 일반 JMS 연결 팩토리(양방향 전용 및 선택적, 그렇지 않은 경우에는 outboundConnection이 가져오기에 사용되고 inboundConnection이 내보내기에 사용됨)
- 수신(가져오기) 및 전송(내보내기) 대상의 일반 JMS 대상(양방향 전용)
- SIB 콜백 큐 대상에 액세스하는 데 사용되는 기본 메시징 프로바이더 콜백 JMS 대상(양방향 전용)
- 콜백 JMS 대상에 액세스하는 데 사용되는 기본 메시징 프로바이더 콜백 JMS 연결 팩토리(양방향 전용)

- 응답 처리 중에 사용하기 위해 요청 메시지에 대한 정보를 저장하는 데 사용되는 SIB 콜백 큐 대상(양방향 전용)

설치 타스크에서는 가져오기 및 내보내기 파일에 있는 정보를 바탕으로 ConnectionFactory, 세 가지 대상 및 ActivationSpec을 작성합니다.

일반 JMS 헤더

일반 JMS 헤더는 일반 JMS 메시지 특성의 모든 특성을 포함하는 서비스 데이터 오브젝트(SDO)입니다. 이러한 특성은 인바운드 메시지의 특성이거나 아웃바운드 메시지에 적용되는 특성일 수 있습니다.

JMS 메시지에는 JMS 시스템 헤더 및 다중 JMS 특성이라는 두 가지 유형의 헤더가 포함됩니다. 양쪽 유형의 헤더는 SMO(Service Message Object)의 중개 모듈이나 ContextService API를 사용하여 액세스될 수 있습니다.

다음 특성은 methodBinding에 대해 정적으로 설정됩니다.

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

일반 바인딩에서는 JMS 및 MQ JMS 바인딩과 동일한 방법으로 JMS 헤더와 특성의 동적 수정도 지원합니다.

일부 일반 JMS 프로바이더는 응용프로그램에서 설정할 수 있는 특성의 종류 및 조합의 종류에 제한을 두기도 합니다. 자세한 정보는 써드파티 제품 문서를 참조하십시오. 그러나 추가 특성이 methodBinding, ignoreInvalidOutboundJMSProperties에 추가되어 모든 예외를 전파할 수 있습니다.

일반 JMS 헤더 및 메시지 특성은 기본 Service Component Architecture SCDL 바인딩 스위치가 켜짐으로 설정된 경우에만 사용됩니다. 스위치가 켜짐으로 설정되면 컨텍스트 정보가 전파됩니다. 기본적으로 이 스위치는 켜짐 상태입니다. 컨텍스트 정보를 전파하지 않으려면 값을 `false`로 변경하십시오.

컨텍스트 전파가 사용 가능한 경우 헤더 정보를 메시지 또는 대상 컴포넌트로 플로우시킬 수 있습니다. 컨텍스트 전파를 설정 및 설정 해제하려면 가져오기 및 내보내기 바인딩의 contextPropagationEnabled 속성으로 `true` 또는 `false`를 지정하십시오. 예를 들어, 다음과 같습니다.

```
<esbBinding xsi:type="eis:JMSImportBinding"
contextProgagationEnabled="true">
```

기본값은 `true`입니다.

일반 JMS 바인딩 문제점 해결

일반 JMS 바인딩의 문제점을 진단하고 수정할 수 있습니다.

구현 예외

다양한 오류 조건에 대한 응답으로 일반 JMS 가져오기 및 내보내기 구현에서는 두 가지 유형의 예외 중 하나가 리턴될 수 있습니다.

- 서비스 비즈니스 예외: 이 예외는 서비스 비즈니스 인터페이스(WSDL 포트 유형)에 지정된 결함이 발생한 경우에 리턴됩니다.
- 서비스 런타임 예외: 다른 모든 경우에 발생합니다. 대부분의 경우, cause 예외에는 원래 예외(JMSException)가 포함됩니다.

일반 JMS 메시지 만기 문제점 해결

JMS 프로바이더에 의한 요청 메시지에는 만기 조건이 있습니다.

*요청 만기*는 요청 메시지에서 JMSExpiration 시간에 도달할 때 JMS 프로바이더에 의해 요청 메시지가 만기되는 것을 말합니다. 다른 JMS 바인딩에서와 같이, 일반 JMS 바인딩은 송신 요청의 경우와 같이 가져오기에서 제출된 콜백 메시지에 대해 만기를 설정하여 요청 만기를 처리합니다. 콜백 메시지의 만기 공고는 요청 메시지가 만기되었으며 비즈니스 예외 수단으로 클라이언트에 알려야 함을 나타냅니다.

그러나 콜백 대상이 써드파티 프로바이더로 이동하는 경우, 이 유형의 요청 만기는 지원되지 않습니다.

*응답 만기*는 응답 메시지에서 JMSExpiration 시간에 도달할 때 JMS 프로바이더에 의해 응답 메시지가 만기되는 것을 말합니다.

일반 JMS 바인딩에 대한 응답 만기는 지원되지 않습니다. 써드파티 JMS 프로바이더의 정확한 만기 작동이 정의되어 있지 않기 때문입니다. 그러나 응답이 수신된 경우 그 응답이 만기되지 않았는지 확인할 수 있습니다.

아웃바운드 요청 메시지의 경우, JMSExpiration 값은 대기한 시간과 asyncHeader에서 전달된 requestExpiration 값(설정된 경우)에서 계산됩니다.

일반 JMS 연결 팩토리 오류 문제점 해결

일반 JMS 프로바이더에서 특정 유형의 연결 팩토리를 정의하는 경우, 응용프로그램을 시작하려고 할 때 오류 메시지를 수신할 수 있습니다. 외부 연결 팩토리를 수정하여 이 문제점을 피할 수 있습니다.

응용프로그램을 실행할 때 다음과 같은 오류 메시지를 수신할 수 있습니다.

```
MDB Listener Port JMSConnectionFactory type does not match
JMSDestination type
```

이 문제점은 외부 연결 팩토리를 정의할 때 발생할 수 있습니다. 특히 JMS 1.1(단일화된) 연결 팩토리(즉, 지점간 및 공개/등록 통신 둘 다를 지원할 수 있는 연결 팩토리) 대신 JMS 1.0.2 주제 연결 팩토리를 작성할 때 예외가 발생할 수 있습니다.

이 문제를 해결하려면 다음 단계를 수행하십시오.

1. 사용 중인 일반 JMS 프로바이더에 액세스하십시오.
2. JMS 1.1(단일화된) 연결 팩토리로 정의한 JMS 1.0.2 주제 연결 팩토리를 바꾸십시오.

새로 정의된 JMS 1.1 연결 팩토리를 사용하여 응용프로그램을 실행하면 더 이상 오류 메시지가 수신되지 않아야 합니다.

일반 JMS 기반 SCA 메시지가 실패 이벤트 관리자에 표시되지 않음

SCA 메시지가 일반 JMS 상호작용 실패를 통해 발생된 경우, 실패 이벤트 관리자에서 이 메시지를 볼 수 있습니다. 이와 같은 메시지가 실패 이벤트 관리자에 표시되지 않으면 기본적인 리스너 포트의 최대 재시도 수 특성 값이 1 이상인지 확인하십시오. 이 값을 1 이상으로 설정하면 일반 JMS 바인딩에 대한 SCA 호출 동안 실패 이벤트 관리자와 상호작용할 수 있습니다.

예외 처리

바인딩이 구성되는 방식은 데이터 핸들러 또는 데이터 바인딩에 의해 발생되는 예외가 처리되는 방식을 판별합니다. 또한 중개 플로우의 특성이 예외가 발생할 때 시스템의 동작을 기술합니다.

데이터 핸들러 또는 데이터 바인딩이 사용자 바인딩에 의해 호출될 때 여러 가지 문제점이 발생할 수 있습니다. 예를 들어, 데이터 핸들러는 손상 페이로드가 있는 메시지를 수신하거나 잘못된 형식이 있는 메시지를 읽으려고 시도합니다.

바인딩이 예외를 처리하는 방법은 데이터 핸들러 또는 데이터 바인딩을 구현하는 방법에 의해 결정됩니다. 권장되는 동작은 `DataBindingException`을 처리하는 데이터 바인딩을 설계하는 것입니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 모든 데이터 핸들러 예외는 데이터 바인딩 예외에 포함됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

`DataBindingException` 예외를 포함한 런타임 예외가 발생할 때 다음을 수행하십시오.

- 중개 플로우가 트랜잭션이도록 구성되는 경우 JMS 메시지가 기본적으로 수동 재생이나 삭제를 위해 실패 이벤트 관리자에 저장됩니다.

주: 실패 이벤트 관리자에 메시지를 저장하는 대신 롤백하도록 바인딩에서 복구 모드를 변경할 수 있습니다.

- 중개 플로우가 트랜잭션이 아닌 경우 예외가 로그되고 메시지가 유실됩니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 데이터 핸들러 예외는 데이터 바인딩 예외에 생성됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

WebSphere MQ JMS 바인딩

WebSphere MQ JMS 바인딩을 사용하면 WebSphere MQ JMS 기반 프로바이더를 사용하는 외부 응용프로그램과 통합할 수 있습니다.

WebSphere MQ JMS 내보내기 및 가져오기 바인딩을 사용하여 사용자 서버 환경에서 외부 JMS 또는 MQ JMS 시스템과 바로 통합할 수 있습니다. 이렇게 하면 서비스 통합 버스의 MQ 링크 또는 클라이언트 링크 기능을 사용할 필요가 없습니다.

컴포넌트가 가져오기를 통해 WebSphere MQ JMS 기반 서비스와 상호작용하는 경우, WebSphere MQ JMS 가져오기 바인딩은 데이터를 전송할 대상 및 응답을 수신할 대상을 이용합니다. JMS 데이터 핸들러 또는 데이터 바인딩 Edge Component를 사용하여 데이터와 JMS 메시지 간의 변환을 수행합니다.

SCA 모듈이 WebSphere MQ JMS 클라이언트로 서비스 제공 시, WebSphere MQ JMS 내보내기 바인딩은 요청을 수신하고 응답을 전송할 수 있는 대상을 사용합니다. JMS 메시지로(부터)의 데이터 변환은 JMS 데이터 핸들러 또는 데이터 바인딩을 통해 수행됩니다.

함수 선택기는 대상 컴포넌트에서 호출할 조작에 맵핑을 제공합니다.

WebSphere MQ JMS 바인딩 개요

WebSphere MQ JMS 바인딩을 사용하면 WebSphere MQ JMS 프로바이더를 사용하는 외부 응용프로그램과 통합할 수 있습니다.

WebSphere MQ 관리 타스크

WebSphere MQ 시스템 관리자는 WebSphere MQ JMS 바인딩에서 해당 바인딩이 들어 있는 응용프로그램을 실행하기 전에 사용할 기본 WebSphere MQ 큐 관리자를 작성해야 합니다.

WebSphere MQ JMS 가져오기 바인딩

WebSphere MQ JMS 가져오기는 SCA 모듈 내의 컴포넌트가 WebSphere MQ JMS 기반 프로바이더가 제공하는 서비스와 통신할 수 있게 합니다. 지원되는 WebSphere MQ 버전을 사용 중이어야 합니다. 자세한 하드웨어 및 소프트웨어 요구사항은 IBM 지원 페이지에 있습니다.

호출되는 조작 유형에 따라 두 가지 유형의 WebSphere MQ JMS 가져오기 바인딩 사용법 시나리오가 지원됩니다.

- 단방향: WebSphere MQ JMS 가져오기는 가져오기 바인딩에서 구성된 전송 대상에 메시지를 놓습니다. JMS 헤더의 replyTo 필드로 아무것도 전송되지 않습니다.
- 양방향(요청-응답): WebSphere MQ JMS 가져오기는 전송 대상에 메시지를 위치시킵니다.

  receive 대상이 replyTo 헤더 필드에 설정됩니다. 수신 대상에서 청취를 위해 메시지 구동 Bean(MDB)이 전개되며 응답이 수신되면 MDB가 응답을 다시 컴포넌트로 전달합니다.

  가져오기 바인딩이 구성되어(**응답 상관 설계** 필드를 WebSphere Integration Developer에서 사용) 응답 메시지 상관 ID가 요청 메시지 ID(기본값)나 요청 메시지 상관 ID에서 복사되었음을 예상할 수 있습니다.

단방향 및 양방향 사용법 시나리오 모두에 동적 및 정적 헤더 특성을 지정할 수 있습니다. 정적 특성은 JMS 가져오기 메소드 바인딩에서 설정할 수 있습니다. 이러한 특성 중 일부는 SCA JMS 런타임에서 특별한 의미를 갖습니다.

WebSphere MQ JMS가 비동기 바인딩임을 참조하십시오. 호출 컴포넌트가 WebSphere MQ JMS 가져오기를 동기적으로 호출하면(양방향 조작의 경우), JMS 서비스가 응답을 리턴할 때까지 호출 컴포넌트가 블록화됩니다.

99 페이지의 그림 38에서는 가져오기가 외부 서비스에 링크되어 있는 방식을 표시합니다.

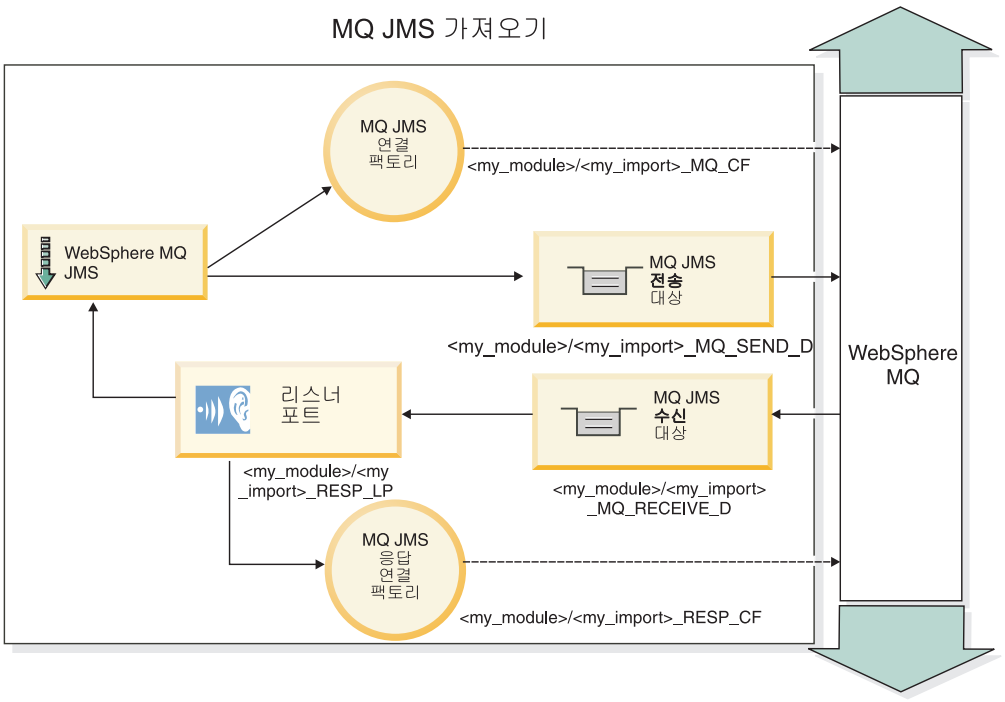


그림 38. WebSphere MQ JMS 가져오기 바인딩 자원

WebSphere MQ JMS 내보내기 바인딩

WebSphere MQ JMS 내보내기 바인딩은 WebSphere MQ 기반 JMS 프로바이더에서 외부 JMS 응용프로그램으로 서비스를 제공하기 위한 SCA 모듈 수단을 제공합니다.

내보내기 바인딩에서 지정한 receive 대상으로 들어오는 요청을 청취하기 위해 MDB가 전개됩니다. 호출한 컴포넌트가 응답을 제공하는 경우 send 필드에서 지정한 대상이 인바운드 요청에 응답을 전송하는 데 사용됩니다. 응답 메시지의 replyTo 필드에서 지정한 대상이 send 필드에서 지정한 대상을 대체합니다.

100 페이지의 그림 39에서는 외부 요청자가 내보내기에 링크되어 있는 방식을 표시합니다.

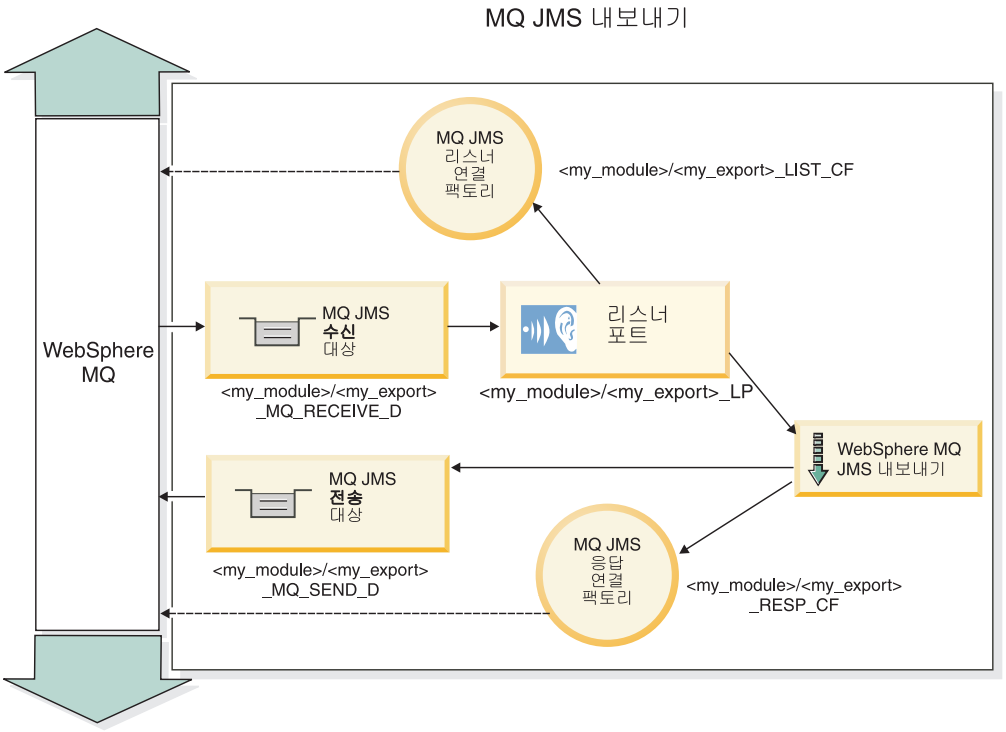


그림 39. WebSphere MQ JMS 내보내기 바인딩 자원

주: 99 페이지의 그림 38 및 그림 39에서는 WebSphere Process Server의 이전 버전에서 응용프로그램을 외부 서비스와 링크하는 방법을 표시합니다. WebSphere Process Server 버전 7.0을 위해 개발된 응용프로그램의 경우 활성화 스펙이 리스너 포트 및 연결 팩토리 대신 사용됩니다.

WebSphere MQ JMS 바인딩의 주요 기능

WebSphere MQ JMS 바인딩의 주요 기능에는 헤더, Java EE 아티팩트 및 작성된 Java EE 자원이 포함됩니다.

헤더

JMS 메시지 헤더에는 메시지를 식별하고 라우트하기 위해 클라이언트 및 프로바이더 둘 다 사용하는 값이 포함된 여러 사전 정의 필드가 있습니다. 바인딩 특성을 사용하여 이 헤더를 고정 값으로 구성하거나 헤더를 런타임 시 동적으로 지정할 수 있습니다.

JMSCorrelationID

관련 메시지에 대한 링크입니다. 일반적으로 이 필드는 응답 중인 메시지의 메시지 ID 문자열에 설정됩니다.

TargetFunctionName

이 헤더는 제공된 함수 선택기 중 하나에서 호출되는 조작을 식별하는 데 사용됩니다. 전송된 메시지의 TargetFunctionName JMS 헤더 특성을 JMS 내보내기로 설정하면 이 함수 선택기를 사용할 수 있습니다. JMS 클라이언트 응용프로그램에서

나 JMS가 바인딩된 가져오기를 이러한 내보내기에 연결할 때 이 특성을 직접 설정할 수 있습니다. 이 경우에는 인터페이스의 각 조작에 대한 TargetFunctionName 헤더를 조작의 이름으로 설정하도록 JMS 가져오기 바인딩을 구성해야 합니다.

상관 설계

WebSphere MQ JMS 바인딩은 요청 메시지와 응답 메시지를 상관시키는 방법을 판별하는 데 사용되는 여러 상관 설계를 제공합니다.

RequestMsgIDToCorrelID
JMSMessageID가 JMSCorrelationID 필드에 복사됩니다. 이는 기본 설정입니다.

RequestCorrelIDToCorrelID
JMSCorrelationID가 JMSCorrelationID 필드에 복사됩니다.

Java EE 자원

MQ JMS 가져오기가 Java EE 환경에 전개되면 여러 Java EE 자원이 작성됩니다.

매개변수

MQ 연결 팩토리
클라이언트가 MQ JMS 프로바이더에 연결을 작성하는 데 사용됩니다.

응답 연결 팩토리
전송 대상이 수신 대상과 다른 큐 관리자에 있는 경우 SCA MQ JMS 런타임에서 사용합니다.

활성화 스펙
MQ JMS 활성화 스펙은 하나 이상의 메시지 구동 Bean과 연관되며 이러한 Bean에 필요한 구성을 제공하여 메시지를 수신합니다.

대상

- 전송 대상:
  - 가져오기: 요청 또는 송신 메시지를 전송하는 장소입니다.
  - 내보내기: 수신 메시지의 JMSReplyTo 헤더 필드로 대체되지 않은 경우 응답 메시지를 전송할 장소입니다.
- 수신 대상:
  - 가져오기: 응답 또는 수신 메시지를 배치할 장소입니다.
  - 내보내기: 수신 또는 요청 메시지를 배치할 장소입니다.

JMS 헤더

JMS 메시지에는 JMS 시스템 헤더 및 다중 JMS 특성이라는 두 가지 유형의 헤더가 포함됩니다. 양쪽 유형의 헤더는 SMO(Service Message Object)의 중개 모듈이나 ContextService API를 사용하여 액세스될 수 있습니다.

JMS 시스템 헤더

JMS 시스템 헤더는 JMSHeader 요소에 의해 SMO에 표시되며, 여기에는 일반적으로 JMS 헤더에 있는 모든 필드가 포함되어 있습니다. 이들 필드는 중개(또는 ContextService)에서 수정될 수 있지만, SMO에서 설정되는 일부 JMS 시스템 헤더 필드는 시스템이나 정적 값으로 대체될 때 아웃바운드 JMS 메시지에서 전파되지 않습니다.

중개(또는 ContextService)에서 갱신될 수 있는 JMS 시스템 헤더의 키 필드는 다음과 같습니다.

- **JMSType** 및 **JMSCorrelationID** – 사전 정의된 특정 메시지 헤더 특성 값
- **JMSDeliveryMode** – 전달 모드의 값(지속적 또는 비지속적, 기본값은 지속적)
- **JMSPriority** – 우선순위 값(0 - 9; 기본값은 JMS_Default_Priority)

JMS 특성

JMS 특성은 특성 목록의 항목으로 SMO에 표시됩니다. 특성은 중개에서 또는 ContextService API를 사용하여 추가, 갱신 또는 삭제될 수 있습니다.

또한 특성은 JMS 바인딩에서 정적으로 설정될 수 있습니다. 정적으로 설정되는 특성은 동적으로 설정된 설정(동일한 이름을 가짐)을 대체합니다.

다른 바인딩(예: HTTP 바인딩)에서 전파된 사용자 특성은 JMS 특성으로 JMS 바인딩에서 출력됩니다.

헤더 전파 설정

인바운드 JMS 메시지에서 다운스트림 컴포넌트로 또는 업스트림 컴포넌트에서 아웃바운드 JMS로의 JMS 시스템 헤더 및 특성의 전파는 바인딩에서 전파 프로토콜 헤더 플래그에 의해 제어될 수 있습니다.

전파 프로토콜 헤더가 설정되면, 다음 목록에서 설명한 대로 헤더 정보는 메시지나 대상 컴포넌트로 플로우됩니다.

- JMS 내보내기 요청

  메시지에 수신된 JMS 헤더는 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다. 메시지에 수신된 JMS 특성은 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다.
- JMS 내보내기 응답

컨텍스트 서비스에 설정된 모든 JMS 헤더 필드는 JMS 내보내기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다. 컨텍스트 서비스의 모든 특성 세트는 JMS 내보내기 바인딩의 정적 특성 세트로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다.

- JMS 가져오기 요청

  컨텍스트 서비스에 설정된 모든 JMS 헤더 필드는 JMS 가져오기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다. 컨텍스트 서비스에 설정된 모든 특성은 JMS 가져오기 바인딩에 설정된 정적 특성으로 대체되지 않는 경우 아웃바운드 메시지에서 사용됩니다.

- JMS 가져오기 응답

  메시지에 수신된 JMS 헤더는 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다. 메시지에 수신된 JMS 특성은 컨텍스트 서비스 방식으로 대상 컴포넌트에 전달됩니다.

외부 클라이언트

서버는 WebSphere MQ JMS 바인딩을 사용하여 외부 클라이언트에 메시지를 전송하거나 반대로 메시지를 수신할 수 있습니다.

외부 클라이언트(예: 웹 포털 또는 엔터프라이즈 정보 시스템)는 응용프로그램의 SCA 컴포넌트에 내보내기 방식으로 메시지를 전송하거나 응용프로그램의 SCA 컴포넌트에 의해 가져오기 방식으로 호출될 수 있습니다.

WebSphere MQ JMS 내보내기 바인딩에서는 메시지 구동 Bean(MDB)을 전개하여 내보내기 바인딩에서 지정한 receive 대상에 수신되는 요청을 청취합니다. 호출한 응용프로그램이 응답을 제공하는 경우 send 필드에서 지정한 대상이 인바운드 요청에 응답을 전송하는 데 사용됩니다. 따라서 외부 클라이언트가 내보내기 바인딩을 통해 응용프로그램을 호출할 수 있습니다.

WebSphere MQ JMS는 외부 클라이언트에 바인드를 가져오고 메시지를 전달할 수 있습니다. 해당 메시지는 외부 클라이언트의 응답을 요구하거나 그렇지 않을 수도 있습니다.

WebSphere MQ를 사용하여 외부 클라이언트와 상호작용하는 방법에 대한 자세한 정보는 WebSphere MQ Information Center에 있습니다.

WebSphere MQ JMS 바인딩 문제점 해결

WebSphere MQ JMS 바인딩에 대한 문제점을 진단하여 수정할 수 있습니다.

구현 예외

다양한 오류 조건에 대한 응답으로 MQ JMS 가져오기 및 내보내기 구현에서는 두 가지 유형의 예외 중 하나가 리턴될 수 있습니다.

- 서비스 비즈니스 예외: 이 예외는 서비스 비즈니스 인터페이스(WSDL 포트 유형)에 지정된 결함이 발생한 경우에 리턴됩니다.
- 서비스 런타임 예외: 다른 모든 경우에 발생합니다. 대부분의 경우, cause 예외에는 원래 예외(JMSException)가 포함됩니다.

  예를 들어, 가져오기에서는 각 요청 메시지에 대해 하나의 응답 메시지만 예상합니다. 응답이 두 개 이상 도달하거나 늦은 응답(SCA 응답 만기가 만료됨)이 도달하면 서비스 런타임 예외가 처리됩니다. 트랜잭션이 롤백되고 응답 메시지가 큐에서 제거되거나 실패 이벤트 관리자에 의해 처리됩니다.

WebSphere MQ JMS 기반 SCA 메시지가 실패 이벤트 관리자에 표시되지 않음

SCA 메시지가 원래 WebSphere MQ 상호작용 실패를 통해 발생된 경우, 실패 이벤트 관리자에서 이 메시지를 볼 수 있습니다. 이와 같은 메시지가 실패 이벤트 관리자에 표시되지 않으면, 기본적인 리스너 포트의 최대 재시도 수 특성 값이 1 이상인지 확인하십시오, 이 값을 1 이상으로 설정하면 MQ JMS 바인딩에 대한 SCA 호출 중 실패 이벤트 관리자와의 상호작용이 사용 가능하게 됩니다.

잘못 사용되는 시나리오: WebSphere MQ 바인딩과의 비교

WebSphere MQ JMS 바인딩은 JMS 메시지 모델에 따라 메시지를 표시하는 WebSphere MQ에 대해 전개된 JMS 응용프로그램과 상호 운영되도록 설계되어 있습니다. 그러나 WebSphere MQ 가져오기 및 내보내기는 주로 기본 WebSphere MQ 응용프로그램과 상호 운영하고 WebSphere MQ 메시지 본문의 전체 컨텐츠를 중개에 표시하도록 설계됩니다.

WebSphere MQ 바인딩이 아닌 WebSphere MQ JMS 바인딩을 사용해서 다음 시나리오를 빌드해야 합니다.

- SCA 모듈에서 JMS 메시지 구동 Bean(MDB)을 호출하며, 여기서 MDB는 WebSphere MQ JMS 프로바이더에 대해 전개됩니다. WebSphere MQ JMS 가져오기를 사용하십시오.
- JMS를 통해 Java EE 컴포넌트 서블릿 또는 EJB에서 SCA 모듈을 호출할 수 있도록 허용합니다. WebSphere MQ JMS 가져오기를 사용하십시오.
- WebSphere MQ에서 이전 시에 JMS MapMessage의 컨텐츠를 중개합니다. 적합한 데이터 핸들러 또는 데이터 바인딩과 함께 WebSphere MQ JMS 내보내기 및 가져오기를 사용합니다.

WebSphere MQ 바인딩 및 WebSphere MQ JMS 바인딩이 상호 운영될 것으로 예상되는 상황이 있습니다. 특히, Java EE 및 비Java EE WebSphere MQ 응용프로그램 사이에서 브릿지하는 경우 적절한 데이터 바인딩 또는 중개 모듈(또는 둘 다)과 함께 WebSphere MQ 내보내기 및 WebSphere MQ JMS 가져오기(또는 반대로)를 사용하십시오.

예외 처리

바인딩이 구성되는 방식은 데이터 핸들러 또는 데이터 바인딩에 의해 발생되는 예외가 처리되는 방식을 판별합니다. 또한 중개 플로우의 특성이 예외가 발생할 때 시스템의 동작을 기술합니다.

데이터 핸들러 또는 데이터 바인딩이 사용자 바인딩에 의해 호출될 때 여러 가지 문제점이 발생할 수 있습니다. 예를 들어, 데이터 핸들러는 손상 페이로드가 있는 메시지를 수신하거나 잘못된 형식이 있는 메시지를 읽으려고 시도합니다.

바인딩이 예외를 처리하는 방법은 데이터 핸들러 또는 데이터 바인딩을 구현하는 방법에 의해 결정됩니다. 권장되는 동작은 `DataBindingException`을 처리하는 데이터 바인딩을 설계하는 것입니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 모든 데이터 핸들러 예외는 데이터 바인딩 예외에 포함됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

`DataBindingException` 예외를 포함한 런타임 예외가 발생할 때 다음을 수행하십시오.

- 중개 플로우가 트랜잭션이도록 구성되는 경우 JMS 메시지가 기본적으로 수동 재생이나 삭제를 위해 실패 이벤트 관리자에 저장됩니다.

  주: 실패 이벤트 관리자에 메시지를 저장하는 대신 롤백하도록 바인딩에서 복구 모드를 변경할 수 있습니다.
- 중개 플로우가 트랜잭션이 아닌 경우 예외가 로그되고 메시지가 유실됩니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 데이터 핸들러 예외는 데이터 바인딩 예외에 생성됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

WebSphere MQ 바인딩

WebSphere MQ 바인딩은 WebSphere MQ 응용프로그램과의 SCA(Service Component Architecture) 연결성을 제공합니다.

사용자의 서버 환경에서 WebSphere MQ 기반 시스템과 바로 통합하려면 WebSphere MQ 내보내기 및 가져오기 바인딩을 사용하십시오. 이렇게 하면 서비스 통합 버스의 MQ 링크 또는 클라이언트 링크 기능을 사용할 필요가 없습니다.

컴포넌트가 가져오기를 통해 WebSphere MQ 서비스와 상호작용할 때 WebSphere MQ 가져오기 바인딩은 데이터가 전송되는 큐와 응답을 수신하는 큐를 사용합니다.

SCA 모듈이 WebSphere MQ 클라이언트에 서비스를 제공할 때 WebSphere MQ 내보내기 바인딩은 요청을 수신하고 응답을 전송할 수 있는 큐를 사용합니다. 함수 선택기는 대상 컴포넌트에서 호출할 조작에 맵핑을 제공합니다.

페이로드 데이터와 MQ 메시지 간의 변환은 MQ 본문 데이터 핸들러나 데이터 바인딩을 이용하여 수행됩니다. 헤더 데이터와 MQ 메시지 간의 변환은 MQ 헤더 데이터 바인딩을 통해 완료됩니다.

지원되는 WebSphere MQ 버전에 대한 자세한 정보는 WebSphere Process Server 시스템 요구사항 웹 사이트를 참조하십시오.

WebSphere MQ 바인딩 개요

WebSphere MQ 바인딩을 사용하면 기본 MQ 기반 응용프로그램과 통합할 수 있습니다.

WebSphere MQ 관리 타스크

WebSphere MQ 시스템 관리자는 WebSphere MQ 바인딩에서 해당 바인딩이 들어 있는 응용프로그램을 실행하기 전에 사용할 기본 WebSphere MQ 큐 관리자를 작성해야 합니다.

WebSphere 관리 타스크

Websphere에서 MQ 자원 어댑터의 **기본 라이브러리 경로** 특성을 서버가 지원하는 WebSphere MQ 버전으로 설정하고 서버를 다시 시작해야 합니다. 이를 수행하면 지원되는 WebSphere MQ 버전의 라이브러리를 사용 중인지 확인합니다. 자세한 하드웨어 및 소프트웨어 요구사항은 IBM 지원 페이지에 있습니다.

WebSphere MQ 가져오기 바인딩

WebSphere MQ 가져오기 바인딩은 SCA 모듈 내의 컴포넌트가 WebSphere MQ 기반 응용프로그램에서 제공하는 서비스와 통신할 수 있게 합니다. 지원되는 WebSphere MQ 버전을 사용 중이어야 합니다. 자세한 하드웨어 및 소프트웨어 요구사항은 IBM 지원 페이지에 있습니다.

외부 WebSphere MQ 시스템과의 상호작용은 요청을 송신하고 응답을 수신하는 큐의 사용을 포함합니다.

호출되는 조작 유형에 따라 두 가지 유형의 WebSphere MQ 가져오기 바인딩 사용법 시나리오가 지원됩니다.

- 단방향: WebSphere MQ 가져오기는 가져오기 바인딩의 **전송 대상 큐** 필드에 구성된 큐에 메시지를 위치시킵니다. MQMD 헤더의 replyTo 필드에는 아무 것도 전송되지 않습니다.
- 양방향(요청-응답): WebSphere MQ 가져오기는 **전송 대상 큐** 필드에서 구성된 큐에 메시지를 위치시킵니다.

  receive 큐가 replyTo MQMD 헤더 필드에 설정됩니다. 수신 큐에서 청취를 위해 메시지 구동 Bean(MDB)이 전개되며 응답이 수신되면 MDB가 응답을 다시 컴포넌트로 전달합니다.

  가져오기 바인딩이 구성되어(**응답 상관 설계** 필드 사용) 응답 메시지 상관 ID가 요청 메시지 ID(기본값)나 요청 메시지 상관 ID에서 복사되었음을 예상할 수 있습니다.

WebSphere MQ가 비동기 바인딩이라는 것에 유의하는 것이 중요합니다. 호출 컴포넌트가 WebSphere MQ 가져오기를 동기적으로 호출하면(양방향 조작의 경우), WebSphere MQ 서비스가 응답을 리턴할 때까지 호출 컴포넌트가 블록화됩니다.

108 페이지의 그림 40에서는 가져오기가 외부 서비스에 링크되어 있는 방식을 표시합니다.

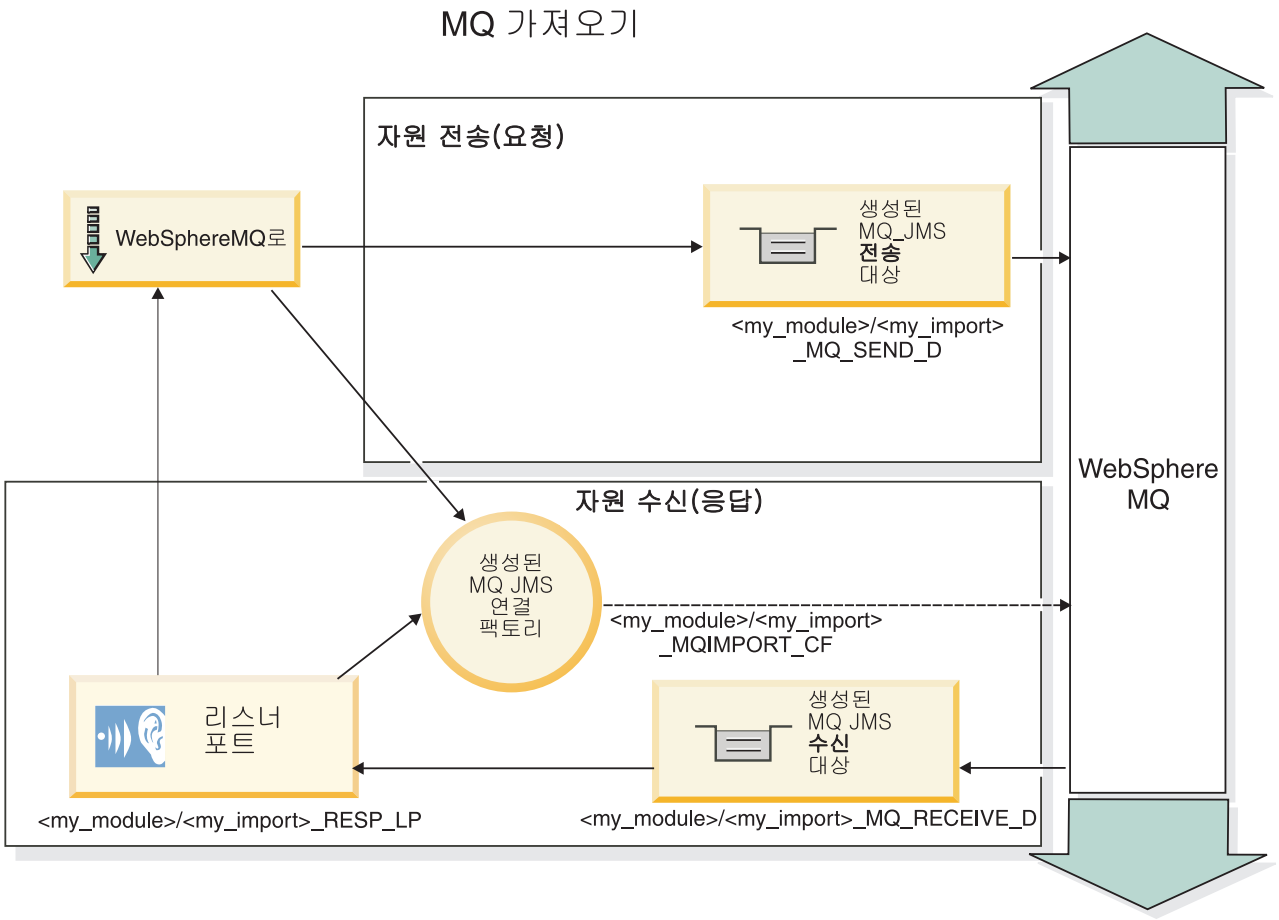


그림 40. WebSphere MQ 가져오기 바인딩 자원

WebSphere MQ 내보내기 바인딩

WebSphere MQ 내보내기 바인딩은 외부 WebSphere MQ 기반 응용프로그램에 서비스를 제공하기 위한 SCA 모듈의 수단을 제공합니다.

내보내기 바인딩에서 지정한 **수신 대상 큐**로의 요청을 청취하기 위해 MDB가 전개됩니다. 호출한 컴포넌트가 응답을 제공하는 경우 **전송 대상 큐** 필드에 지정된 큐가 인바운드 요청에 응답을 전송하는 데 사용됩니다. 응답 메시지의 replyTo 필드에 지정된 큐는 **전송 대상 큐**에 지정된 큐를 대체합니다.

109 페이지의 그림 41에서는 외부 요청자가 내보내기에 링크되어 있는 방식을 표시합니다.

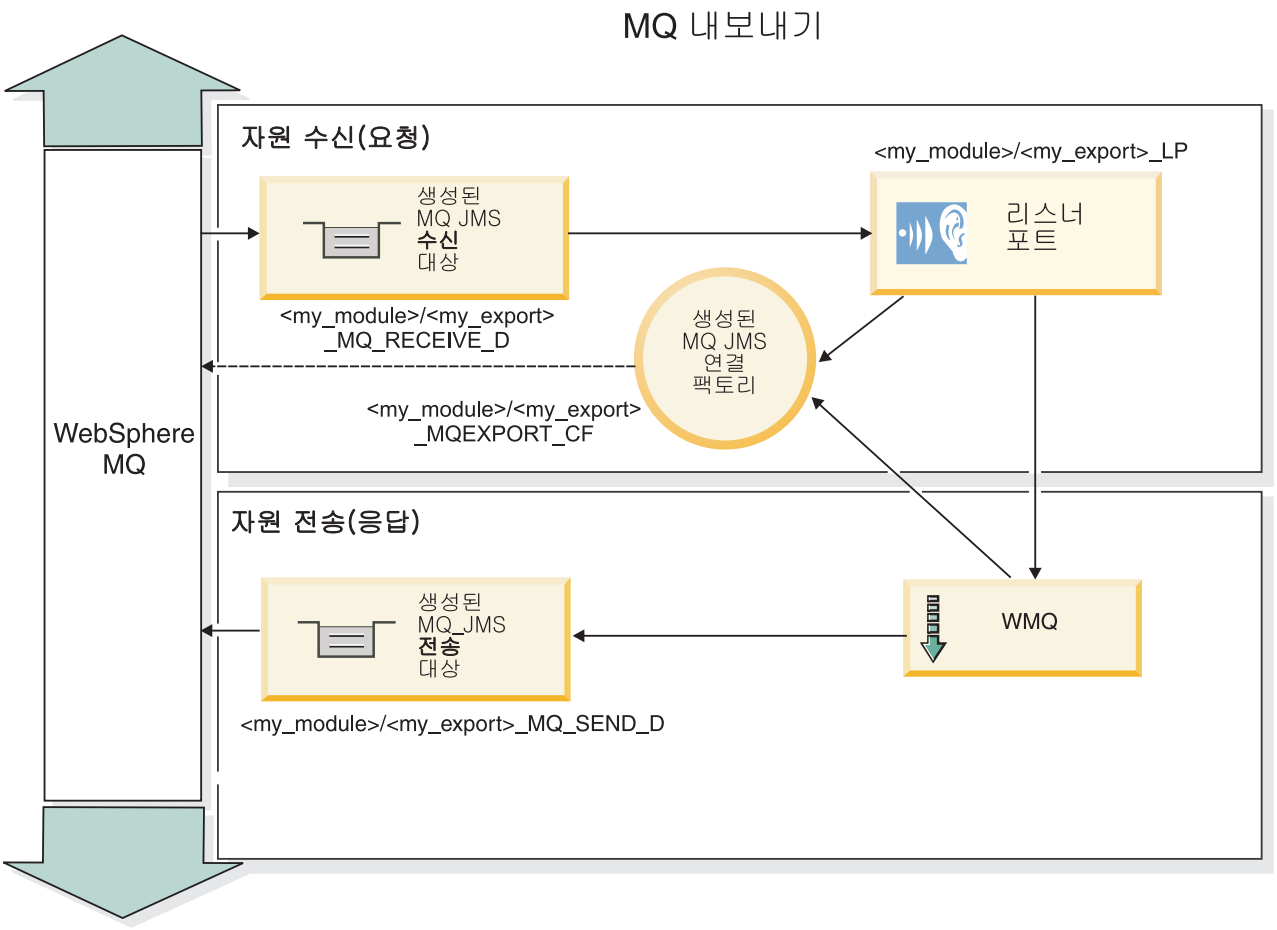


그림 41. WebSphere MQ 내보내기 바인딩 자원

주: 108 페이지의 그림 40 및 그림 41에서는 WebSphere Process Server의 이전 버전에서 응용프로그램을 외부 서비스와 링크하는 방법을 표시합니다. WebSphere Process Server 버전 7.0을 위해 개발된 응용프로그램의 경우 활성화 스펙이 리스너 포트 및 연결 팩토리 대신 사용됩니다.

WebSphere MQ 바인딩의 주요 기능

WebSphere MQ 바인딩의 주요 기능에는 헤더, Java EE 아티팩트 및 작성된 Java EE 자원이 포함됩니다.

상관 설계

WebSphere MQ 요청/응답 응용프로그램에서는 여러 기법 중 하나를 사용하여 MQMD의 MessageID 및 CorrelID 필드 주변에서 빌드된 요청과 응답 메시지를 상관시킬 수 있습니다. 대부분의 경우 요청자는 큐 관리자가 MessageID를 선택하게 하고 응답 응용프로그램이 이를 응답의 CorrelID에 복사하도록 합니다. 대개의 경우 요청자와 응답 응용프로그램은 사용 중인 상관의 종류를 묵시적으로 알고 있습니다. 경우에 따라 응답 응용프로그램은 요청의 Report 필드에 있는 여러 플래그를 사용하며 이 플래그에는 해당 필드를 처리하는 방법이 설명되어 있습니다.

다음 옵션을 사용하여 WebSphere MQ 메시지의 내보내기 바인딩을 구성할 수 있습니다.

응답 MsgId 옵션:

새 MsgID
큐 관리자가 응답에 적합한 고유 MsgId를 선택할 수 있게 합니다(기본값).

요청 MsgID에서 복사
요청의 MsgId 필드에서 MsgId 필드를 복사합니다.

SCA 메시지에서 복사
MsgId가 SCA 응답 메시지의 WebSphere MQ 헤더에 포함되어 전달되도록 설정하거나 값이 없는 경우에는 큐 관리자가 새 ID를 정의하도록 합니다.

보고서 옵션으로
MsgId 처리 방법에 대한 힌트를 얻기 위해 요청에 있는 MQMD의 Report 필드를 검사합니다. MQRO_NEW_MSG_ID 및 MQRO_PASS_MSG_ID 옵션이 지원되며 이는 각각 New MsgId 및 Copy from Request MsgID와 같이 작동합니다.

응답 CorrelId 옵션:

요청 MsgID에서 복사
요청의 MsgId 필드에서 CorrelId 필드를 복사합니다(기본값).

요청 CorrelID에서 복사
요청의 CorrelId 필드에서 CorrelId 필드를 복사합니다.

SCA 메시지에서 복사
CorrelId가 SCA 응답 메시지의 WebSphere MQ 헤더에 포함되어 전달되도록 설정하거나 값이 없는 경우에는 공백으로 둡니다.

보고서 옵션으로
CorrelId 처리 방법에 대한 힌트를 얻기 위해 요청에 있는 MQMD의 Report 필드를 검사합니다. MQRO_COPY_MSG_ID_TO_CORREL_ID 및 MQRO_PASS_CORREL_ID 옵션이 지원되며 이는 각각 Copy from Request MsgID 및 Copy from Request CorrelID와 같이 작동합니다.

다음 옵션을 사용하여 WebSphere MQ 메시지의 가져오기 바인딩을 구성할 수 있습니다.

요청 MsgId 옵션:

새 MsgID
큐 관리자가 요청에 적합한 고유 MsgId를 선택할 수 있게 합니다(기본값).

SCA 메시지에서 복사

MsgId가 SCA 요청 메시지의 WebSphere MQ 헤더에 포함되어 전달되도록 설정하거나 값이 없는 경우에는 큐 관리자가 새 ID를 정의하도록 합니다.

응답 상관 옵션:

응답에 MsgId에서 복사한 CorrelID가 있음

응답 메시지에 요청의 MsgId에 따라 설정된 CorrelId 필드가 있어야 합니다(기본값).

응답에 MsgId에서 복사한 MsgID가 있음

응답 메시지에 요청의 MsgId에 따라 설정된 MsgId 필드가 있어야 합니다(기본값).

응답에 CorrelId에서 복사한 CorrelID가 있음

응답 메시지에 요청의 CorrelId에 따라 설정된 CorrelId 필드가 있어야 합니다(기본값).

Java EE 자원

WebSphere MQ 바인딩이 Java EE 환경에 전개되면 여러 Java EE 자원이 작성됩니다.

매개변수

MQ 연결 팩토리

클라이언트가 WebSphere MQ 프로바이더에 연결을 작성하는 데 사용됩니다.

응답 연결 팩토리

전송 대상이 수신 대상과 다른 큐 관리자에 있는 경우 SCA MQ 런타임에서 사용합니다.

활성화 스펙

MQ JMS 활성화 스펙은 하나 이상의 메시지 구동 Bean과 연관되며 이러한 Bean에 필요한 구성을 제공하여 메시지를 수신합니다.

대상

- 전송 대상: 가져오기의 경우 요청 또는 송신 메시지를 전송하는 장소입니다. 내보내기의 경우에는 응답 메시지를 전송할 장소입니다(수신 메시지에서 MQMD ReplyTo 헤더 필드로 대체되지 않은 경우).
- 수신 대상: 응답/요청 또는 수신 메시지를 배치할 장소입니다.

WebSphere MQ 헤더

WebSphere MQ 헤더에는 SCA(Service Component Architecture) 메시지로 변환하기 위한 특정 규칙이 포함되어 있습니다.

WebSphere MQ 메시지는 시스템 헤더(MQMD), 0개 이상의 다른 MQ 헤더(시스템 또는 사용자 정의) 및 메시지 본문으로 구성됩니다. 메시지에 여러 개의 메시지 헤더가 있는 경우 헤더 순서가 중요합니다.

각 헤더에는 그 뒤에 있는 헤더 구조에 대해 설명하는 정보가 포함되어 있습니다. MQMD는 첫 번째 헤더를 설명합니다.

MQ 헤더의 구문 분석 방법

MQ 헤더 데이터 바인딩은 MQ 헤더를 구문 분석하는 데 사용됩니다. 다음 헤더는 자동으로 지원됩니다.

- MQRFH
- MQRFH2
- MQCIH
- MQIIH

MQH로 시작하는 헤더는 다르게 처리됩니다. 헤더의 특정 필드는 구문 분석되지 않으며, 구문 분석되지 않은 바이트로 남습니다.

다른 MQ 헤더의 경우, 헤더를 구문 분석하기 위해 사용자 정의 MQ 헤더 데이터 바인딩을 기록할 수 있습니다.

MQ 헤더의 액세스 방법

MQ 헤더는 다음 두 가지 방법 중 하나로 제품에서 액세스될 수 있습니다.

- 중개에서 SMO(service message object)를 통해
- ContextService API를 통해

MQ 헤더는 내부적으로 SMO MQHeader 요소와 함께 표시됩니다. MQHeader는 MQControl을 확장하는 헤더 데이터의 컨테이너이지만 모든 유형의 값 요소를 포함합니다. MQMD, MQControl(MQ 메시지 본문 제어 정보) 및 다른 MQ 헤더 목록을 포함합니다.

- MQMD는 본문의 구조 및 인코딩을 판별하는 정보를 제외한 WebSphere MQ 메시지 설명의 컨텐츠를 표시합니다.
- MQControl에는 메시지 본문의 구조 및 인코딩을 판별하는 정보가 포함되어 있습니다.
- MQHeader에는 MQHeader 오브젝트 목록이 포함되어 있습니다.

MQ 헤더 체인은 감겨 있지 않으므로, 각 MQ 헤더는 고유한 제어 정보(CCSID, 인코딩 및 형식)를 SMO 안에 갖습니다. 다른 헤더 데이터를 변경시키지 않고 손쉽게 헤더를 추가 또는 삭제할 수 있습니다.

MQMD의 필드 설정

중개에서 컨텍스트 API를 사용하여 또는 SMO(Service Message Object)를 통해 MQMD를 갱신할 수 있습니다. 다음 필드는 자동으로 아웃바운드 MQ 메시지로 전파됩니다.

- 인코딩
- CodedCharacterSet
- 형식
- 보고서
- 만기
- 피드백
- 우선순위
- 지속
- CorrelId
- MsgFlags

가져오기 또는 내보내기에 대한 MQ 바인딩을 구성하여 다음 특성을 아웃바운드 MQ 메시지로 전파하십시오.

MsgID
요청 메시지 ID를 SCA 메시지에서 복사로 설정하십시오.

MsgType
요청-응답 조작의 메시지 유형을 MQMT_DATAGRAM 또는 MQMT_REQUEST로 설정 선택란을 지우십시오.

ReplyToQ
요청 메시지 큐에 대한 응답 대체 선택란을 지우십시오.

ReplyToQMgr
요청 메시지 큐에 대한 응답 대체 선택란을 지우십시오.

버전 7.0부터는 JNDI 대상 정의에서 사용자 정의 특성을 사용하여 컨텍스트 필드를 대체할 수 있습니다. 다음 필드를 아웃바운드 MQ 메시지로 전파하는 전송 대상에서 사용자 정의 특성 MDCTX에 값 SET_IDENTITY_CONTEXT를 설정하십시오.

- UserIdentifier
- AppIdentityData

다음 특성을 아웃바운드 MQ 메시지로 전파하는 전송 대상에서 사용자 정의 특성 MDCTX에 값 SET_ALL_CONTEXT를 설정하십시오.

- UserIdentifier
- AppIdentityData

- PutApplType
- PutApplName
- ApplOriginData

일부 필드는 아웃바운드 MQ 메시지로 전파되지 않습니다. 다음 필드는 메시지를 전송하는 동안 대체됩니다.

- BackoutCount
- AccountingToken
- PutDate
- PutTime
- Offset
- OriginalLength

WebSphere MQ 바인딩에 MQCIH 정적으로 추가

WebSphere Process Server는 중개 모듈을 사용하지 않고 MQCIH 헤더 정보를 정적으로 추가하는 방식을 지원합니다.

MQCIH 헤더 정보를 메시지에 추가하는 다양한 방법이 있습니다(예: 헤더 Setter 중개 기본요소를 사용해서 추가). 이 방법은 추가 중개 모듈을 사용하지 않고 해당 헤더 정보를 정적으로 추가하는 데 유용합니다. CICS® 프로그램 이름, 트랜잭션 ID 및 기타 데이터 형식 헤더 세부사항을 포함하는 정적 헤더 정보를 정의하고 WebSphere MQ 바인딩의 파트로 추가할 수 있습니다.

WebSphere MQ, MQ CICS Bridge 및 CICS는 정적으로 추가될 MQCIH 헤더 정보에 대해 구성되어야 합니다.

WebSphere Integration Developer를 사용하여 MQCIH 헤더 정보에 필요한 정적 값으로 WebSphere MQ 가져오기를 구성할 수 있습니다.

메시지가 도달하고 해당 메시지가 WebSphere MQ 가져오기에 의해 처리되는 경우 MQCIH 헤더 정보가 메시지에 이미 있는지 여부를 확인하는 검사가 이루어집니다. MQCIH 헤더 정보가 메시지에 있는 경우 WebSphere MQ 가져오기에 정의된 정적 값을 사용하여 해당 메시지의 상응하는 동적 값을 대체합니다. MQCIH 헤더 정보가 메시지에 없는 경우 정보가 메시지에 작성되고 WebSphere MQ 가져오기에 정의된 정적 값이 추가됩니다.

WebSphere MQ 가져오기에 정의된 정적 값은 메소드에 대해 특정합니다. 동일한 WebSphere MQ 가져오기의 다른 메소드에 다른 정적 MQCIH 값을 지정할 수 있습니다.

이 기능은 WebSphere MQ 가져오기에 정의된 정적 값이 수신 메시지에 제공된 해당 값을 대체하기 때문에 MQCIH에 특정 헤더 정보가 포함되지 않은 경우 기본값 제공에 사용되지 않습니다.

외부 클라이언트

WebSphere Process Server는 WebSphere MQ 바인딩을 사용해서 외부 클라이언트로 메시지를 전송하거나 외부 클라이언트에서 메시지를 수신할 수 있습니다.

외부 클라이언트(예: 웹 포털 또는 엔터프라이즈 정보 시스템)는 응용프로그램의 SCA 컴포넌트에 내보내기 방식으로 메시지를 전송하거나 응용프로그램의 SCA 컴포넌트에 의해 가져오기 방식으로 호출될 수 있습니다.

WebSphere MQ 내보내기 바인딩에서는 메시지 구동 Bean(MDB)을 전개하여 내보내기 바인딩에서 지정한 receive 대상에 수신되는 요청을 청취합니다. 호출한 응용프로그램이 응답을 제공하는 경우 send 필드에서 지정한 대상이 인바운드 요청에 응답을 전송하는 데 사용됩니다. 따라서 외부 클라이언트가 내보내기 바인딩을 이용해 응용프로그램을 호출할 수 있습니다.

WebSphere MQ는 외부 클라이언트에 바인드를 가져오고 메시지를 전달할 수 있습니다. 해당 메시지는 외부 클라이언트의 응답을 요구하거나 그렇지 않을 수도 있습니다.

WebSphere MQ를 사용하여 외부 클라이언트와 상호작용하는 방법에 대한 자세한 정보는 WebSphere MQ Information Center에 있습니다.

WebSphere MQ 바인딩 문제점 해결

WebSphere MQ 바인딩에 대해 발생하는 결함 및 실패 조건을 진단하고 수정할 수 있습니다.

1차 실패 조건

WebSphere MQ 바인딩의 1차 실패 조건은 트랜잭션 시맨틱, WebSphere MQ 구성 또는 다른 컴포넌트에서 기존 작동에 대한 참조로 판별됩니다. 1차 실패 조건은 다음과 같습니다.

- WebSphere MQ 큐 관리자 또는 큐에 대한 연결 실패

  메시지 수신을 위한 WebSphere MQ에 대한 연결에 실패하면 MDB 리스너 포트가 시작되지 않습니다. 이 조건은 WebSphere Application Server 로그에 로깅됩니다. 지속적 메시지는 성공적으로 검색(또는 WebSphere MQ에 의해 만기)될 때까지 WebSphere MQ 큐에 유지됩니다.

  아웃바운드 메시지를 전송하기 위해 WebSphere MQ에 연결하는 데 실패하면 전송을 제어하는 트랜잭션이 롤백됩니다.
- 인바운드 메시지 구문 분석 또는 아웃바운드 메시지 생성 실패

데이터 바인딩에서 실패가 발생하면 작업을 제어하는 트랜잭션이 롤백됩니다.

- 아웃바운드 메시지 전송 실패

  메시지 전송에 실패하면 관련 트랜잭션이 롤백됩니다.

- 다중 또는 예기치 않은 응답 메시지

  가져오기에서는 각 요청 메시지에 대해 하나의 응답 메시지만 예상합니다. 응답이 두 개 이상 도달하거나 늦은 응답(SCA 응답 만기가 만료됨)이 도달하면 서비스 런타임 예외가 처리됩니다. 트랜잭션이 롤백되고 응답 메시지가 큐에서 제거되거나 실패 이벤트 관리자에 의해 처리됩니다.

잘못 사용되는 시나리오: WebSphere MQ JMS 바인딩과의 비교

WebSphere MQ 가져오기 및 내보내기는 주로 기본 WebSphere MQ 응용프로그램과 상호 운영하고 WebSphere MQ 메시지 본문의 전체 컨텐츠를 중개에 표시하도록 설계됩니다. 그러나 WebSphere MQ JMS 바인딩은 JMS 메시지 모델에 따라 메시지를 표시하는 WebSphere MQ에 대해 전개된 JMS 응용프로그램과 상호 운영하도록 설계됩니다.

WebSphere MQ 바인딩이 아닌 WebSphere MQ JMS 바인딩을 사용해서 다음 시나리오를 빌드해야 합니다.

- SCA 모듈에서 JMS 메시지 구동 Bean(MDB)을 호출하며, 여기서 MDB는 WebSphere MQ JMS 프로바이더에 대해 전개됩니다. WebSphere MQ JMS 가져오기를 사용하십시오.
- JMS를 통해 Java EE 컴포넌트 서블릿 또는 EJB에서 SCA 모듈을 호출할 수 있도록 허용합니다. WebSphere MQ JMS 가져오기를 사용하십시오.
- WebSphere MQ에서 이전 시에 JMS MapMessage의 컨텐츠를 중개합니다. 적합한 데이터 바인딩과 함께 WebSphere MQ JMS 내보내기 및 가져오기를 사용합니다.

WebSphere MQ 바인딩 및 WebSphere MQ JMS 바인딩이 상호 운영될 것으로 예상되는 상황이 있습니다. 특히, Java EE 및 비Java EE WebSphere MQ 응용프로그램 사이에서 브릿지하는 경우 적절한 데이터 바인딩 또는 중개 모듈(또는 둘 다)과 함께 WebSphere MQ 내보내기 및 WebSphere MQ JMS 가져오기(또는 반대로)를 사용하십시오.

전달되지 않은 메시지

WebSphere MQ가 메시지를 의도한 대상에 전달할 수 없는 경우(예를 들어, 구성 오류로 인해)에는 후보로 지명된 데드-레터 큐에 메시지를 대신 전송합니다.

이 때 메시지 본문의 시작 부분에 데드-레터 헤더가 추가됩니다. 이 헤더에는 실패 이유, 원래 대상 및 기타 정보가 포함되어 있습니다.

MQ 기반 SCA 메시지가 실패 이벤트 관리자에 표시되지 않음

SCA 메시지가 WebSphere MQ 상호작용 실패로 인해 발생한 경우 실패 이벤트 관리자에서 이러한 메시지를 찾을 수 있습니다. 이러한 메시지가 실패 이벤트 관리자에 표시되지 않으면 기본적인 WebSphere MQ 대상에 1 이상의 실패한 최대 전달 값이 있는지 확인하십시오. 이 값을 2 이상으로 설정하면 WebSphere MQ 바인딩에 대한 SCA 호출 동안 실패 이벤트 관리자와 상호작용할 수 있습니다.

MQ 실패 관리자가 잘못된 큐 관리자에 대해 재생됨

사전 정의된 연결 팩토리가 아웃바운드 연결에 사용되는 경우, 연결 특성은 인바운드 연결에 사용된 활성화 스펙에 정의된 것과 일치해야 합니다.

사전 정의된 연결 팩토리는 실패 이벤트를 재생할 때 연결을 작성하는 데 사용되므로 원래 메시지를 수신했던 것과 동일한 큐 관리자를 사용하도록 구성해야 합니다.

예외 처리

바인딩이 구성되는 방식은 데이터 핸들러 또는 데이터 바인딩에 의해 발생되는 예외가 처리되는 방식을 판별합니다. 또한 중개 플로우의 특성이 예외가 발생할 때 시스템의 동작을 기술합니다.

데이터 핸들러 또는 데이터 바인딩이 사용자 바인딩에 의해 호출될 때 여러 가지 문제점이 발생할 수 있습니다. 예를 들어, 데이터 핸들러는 손상 페이로드가 있는 메시지를 수신하거나 잘못된 형식이 있는 메시지를 읽으려고 시도합니다.

바인딩이 예외를 처리하는 방법은 데이터 핸들러 또는 데이터 바인딩을 구현하는 방법에 의해 결정됩니다. 권장되는 동작은 `DataBindingException`을 처리하는 데이터 바인딩을 설계하는 것입니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 모든 데이터 핸들러 예외는 데이터 바인딩 예외에 포함됩니다. 따라서 `DataHandlerException`이 사용자에게 `DataBindingException`으로 보고됩니다.

`DataBindingException` 예외를 포함한 런타임 예외가 발생할 때 다음을 수행하십시오.

- 중개 플로우가 트랜잭션이도록 구성되는 경우 JMS 메시지가 기본적으로 수동 재생이나 삭제를 위해 실패 이벤트 관리자에 저장됩니다.

  주: 실패 이벤트 관리자에 메시지를 저장하는 대신 롤백하도록 바인딩에서 복구 모드를 변경할 수 있습니다.
- 중개 플로우가 트랜잭션이 아닌 경우 예외가 로그되고 메시지가 유실됩니다.

데이터 핸들러의 상황도 비슷합니다. 데이터 핸들러가 데이터 바인딩에 의해 호출되므로 데이터 핸들러 예외는 데이터 바인딩 예외에 생성됩니다. 따라서 DataHandlerException이 사용자에게 DataBindingException으로 보고됩니다.

바인딩 제한사항

바인딩 사용에는 여기에 나열된 제한사항이 있습니다.

MQ 바인딩의 제한사항

MQ 바인딩 사용에는 여기에 나열된 제한사항이 있습니다.

publish-subscribe 메시지 분배 없음

메시지를 분배하는 publish-subscribe 메소드가 현재 자체적으로 publish-subscribe를 지원하는 WMQ를 통한 MQ 바인딩에서 지원되지 않습니다. 그러나 MQ JMS 바인딩은 이 분배 메소드를 지원합니다.

공유 수신 큐

다중 WebSphere MQ 내보내기 및 가져오기 바인딩은 구성된 수신 큐에 해당 가져오기 또는 내보내기에 대한 메시지가 있을 것을 예상합니다. 가져오기 및 내보내기 바인딩은 다음 사항을 고려하여 구성해야 합니다.

- MQ 가져오기 바인딩은 수신 큐의 모든 메시지를 보낸 요청에 대한 응답으로 가정하므로 각 MQ 가져오기에는 다른 수신 큐가 있어야 합니다. 둘 이상의 가져오기가 수신 큐를 공유하는 경우, 잘못된 가져오기가 응답을 수신할 수 있으므로 원본 요청 메시지와 상관되는 데 실패합니다.
- 각 MQ 내보내기에는 다른 수신 큐가 있어야 합니다. 그렇지 않으면 특정 요청 메시지를 가져올 내보내기를 예상할 수 없습니다.
- MQ 가져오기와 내보내기는 동일한 전송 큐를 가리킬 수 있습니다.

JMS, MQ JMS 및 일반 JMS 바인딩의 제한사항

JMS 및 MQ JMS 바인딩에는 제한사항이 있습니다.

기본 바인딩 생성에 내포된 사항

JMS, MQ JMS 및 일반 JMS 바인딩의 제한사항은 다음 절에서 설명합니다.

- 기본 바인딩 생성에 내포된 사항
- 응답 상관 설계
- 양방향 지원

바인딩을 생성할 때 몇몇 필드는 기본값으로 채워집니다(사용자 스스로 값을 입력할 것을 선택하지 않은 경우). 예를 들어 연결 팩토리 이름이 자동으로 작성됩니다. 사용자가 응용프로그램을 서버에 넣고 클라이언트가 원격으로 그 응용프로그램에 액세스할 것

을 아는 경우, 바인딩 작성 시 기본값을 사용하지 않고 JNDI 이름을 입력해야 합니다. 런타임 시 사용자가 관리 콘솔을 통해 이 값을 제어하려고 하기 때문입니다.

그러나 기본값을 승인한 후 나중에 원격 클라이언트에서 응용프로그램에 액세스할 수 없음을 발견한 경우에는 관리 콘솔을 사용하여 연결 팩토리 값을 명시적으로 설정할 수 있습니다. 연결 팩토리 설정에서 프로바이더 엔드포인트 필드를 찾고 <server_hostname>:7276(기본 포트 번호를 사용 중인 경우)과 같은 값을 추가하십시오.

응답 상관 설계

요청-응답 조작에서 메시지를 상관시키기 위해 CorrelationId 대 CorrelationId 응답 상관 설계를 사용하는 경우 메시지에 동적 상관 ID가 있어야 합니다.

중개 플로우 편집기를 사용하여 중개 모듈에서 동적 상관 ID를 작성하려면 JMS 바인딩과 함께 가져오기 전에 XSLT 노드를 추가하십시오. XSLT 맵핑 편집기를 여십시오. 알려진 Service Component Architecture 헤더는 대상 메시지에서 사용할 수 있게 됩니다. 소스 메시지에 고유 ID를 포함하는 필드를 끌어서 대상 메시지의 JMS 헤더에 있는 상관 ID에 놓으십시오.

양방향 지원

런타임 시 JNDI(Java Naming and Directory Interface) 이름에는 ASCII 문자만 지원됩니다.

공유 수신 큐

다중 내보내기 및 가져오기 바인딩은 구성된 수신 큐에 해당 가져오기 또는 내보내기에 대한 메시지가 있을 것을 예상합니다. 가져오기 및 내보내기 바인딩은 다음 사항을 고려하여 구성해야 합니다.

- 가져오기 바인딩은 수신 큐의 모든 메시지를 보낸 요청에 대한 응답으로 가정하므로 각 가져오기 바인딩에는 다른 수신 큐가 있어야 합니다. 둘 이상의 가져오기가 수신 큐를 공유하는 경우, 잘못된 가져오기가 응답을 수신할 수 있으므로 원본 요청 메시지와 상관되는 데 실패합니다.
- 각 내보내기에는 다른 수신 큐가 있어야 합니다. 그렇지 않으면 특정 요청 메시지를 가져올 내보내기를 예상할 수 없습니다.
- 가져오기와 내보내기는 동일한 전송 큐를 가리킬 수 있습니다.

제 3 장 프로그래밍 안내서 및 기술

이 절에는 프로그래밍 안내서 및 예제가 있습니다.

다음 하위 주제에서는 다양한 컴포넌트, 응용프로그램 및 비즈니스 통합 솔루션의 프로그래밍에 대한 정보를 제공합니다.

중요사항: WebSphere Process Server 및 WebSphere Enterprise Service Bus에서 지원하는 API(application programming interface) 및 SPI(system programming interfaces)의 세부사항에 대해서는 InfoCenter의 참조 절을 참조하십시오.

Service Component Architecture 프로그래밍

SCA(Service Component Architecture)는 SOA(Service Oriented Architecture)를 기초로 응용프로그램을 구성하기 위한 단순하면서 강력한 프로그래밍 모델을 제공합니다.

서비스 컴포넌트 정의 언어

서비스 컴포넌트 정의 언어(SCDL)는 모듈, 컴포넌트, 참조, 가져오기 및 내보내기와 같은 SCA(Service Component Architecture) 컴포넌트를 설명하기 위해 사용하는 XML 기반 언어입니다.

SCA에 존재하는 다양한 아티팩트 유형은 이 SOA(Service Oriented Architecture)의 기본 요구사항 중 일부를 지원하도록 설계되었습니다. SCA 사용을 시작하려면 SCA에 기본 서비스 컴포넌트를 정의하기 위한 메커니즘이 필요합니다. 서비스 컴포넌트 정의에 대한 메커니즘이 있으면 이 서비스를 현재 SCA 모듈 내부 또는 외부에서 클라이언트가 사용 가능하도록 만들 수 있는 것이 중요합니다. 또한 현재 SCA 모듈 외부에 있는 서비스를 가져오고 참조하도록 설계된 구성이 존재해야 합니다. 마지막으로, SCA는 서비스 및 모듈을 대형 응용프로그램 내에 작성하기 위한 구성을 제공합니다.

SCDL 정의는 여러 파일을 대상으로 구성됩니다. 예를 들어 신용카드 승인 응용프로그램에서, 인터페이스 및 구현에 대한 SCDL을 `CreditApproval.component`라고 하는 파일에 저장할 수 있습니다. 참조는 `CreditApproval.component` 파일(인라인)이나 모듈 루트에 있는 독립 `sca.references` 파일에 포함될 수 있습니다. 독립형 참조는 `sca.references` 파일에 위치합니다. 독립형 참조는 SCA 모듈을 호출하기 위해 동일한 SCA 모듈 내에서 비SCA 아티팩트(JSP)가 사용할 수 있습니다.

표 14. SCA 서비스 모듈을 구성하는 1차 아티팩트

아티팩트	SCDL 정의
모듈 정의	• 루트 SCA 프로젝트 JAR에 있는 `sca.module` 파일에 포함됩니다.

표 14. SCA 서비스 모듈을 구성하는 1차 아티팩트 (계속)

아티팩트	SCDL 정의
서비스 컴포넌트	• 모듈에는 0..n개의 서비스 정의가 포함될 수 있습니다. • 각 컴포넌트 정의는 <SERVICE_NAME>.component 파일에 포함됩니다.
가져오기	• 모듈에는 0..n개의 가져오기 정의가 포함될 수 있습니다. • 각 가져오기 정의는 <IMPORT_NAME>.import 파일에 포함됩니다.
내보내기	• 모듈에는 0..n개의 내보내기 정의가 포함될 수 있습니다. • 각 내보내기 정의는 <EXPORT_NAME>.export 파일에 포함됩니다.
참조	• 두 가지 유형의 참조 – 인라인(서비스 컴포넌트 정의 내에 포함됨) – 독립형 • 각 컴포넌트 정의는 <SERVICE_NAME>.reference 파일에 포함됩니다.
기타 아티팩트	• 기타 아티팩트로는 Java 클래스, WSDL 파일, 기타 아티팩트 XSD 파일, BPEL이 있습니다.

SCA 응용프로그램을 빌드할 때 WebSphere Integration Developer는 적절한 SCDL 정의를 생성합니다. 그러나 SCDL과의 기본적인 유사성은 전체 아키텍처를 이해하고 응용프로그램을 디버깅할 때 도움이 될 수 있습니다.

모듈 정의

SCA(Service Component Architecture)는 컴포넌트를 서비스 모듈에 패키징하기 위한 표준 전개 모델을 정의합니다. sca.module 파일에는 모듈 정의가 포함됩니다.

SCA 모듈은 단지 다른 유형의 패키지가 아닙니다. WebSphere Process Server에서, SCA 서비스 모듈은 Java EE EAR 파일과 몇몇 다른 Java EE 서브모듈과 같습니다. Java EE 요소(예: WAR 파일)는 SCA 모듈과 함께 패키지로 묶을 수 있습니다. SCA가 아닌 아티팩트(JSP 및 기타)도 SCA 모듈과 함께 패키지로 묶어서, 독립형 참조라고 하는 특수한 참조 유형을 사용하여 SCA 클라이언트 프로그래밍 모델을 통해 SCA 서비스를 호출하게 합니다.

다음은 sca.module 파일의 예제입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
name="CreditApproval"/>
```

다이어그램은 편집기에서 볼 수 있는 연관된 SCDL 모듈 정의와 함께 WebSphere Integration Developer의 모듈을 표시합니다. 이 예제에서 모듈 유형은 중개 모듈입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:mt="http://www.ibm.com/xmlns/prod/websphere/scdl/moduletype/7.0.0"
  Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"name="StockQuote">
  <mt:moduleType type="com.ibm.ws.sca.scdl.moduletype.mediation" version="1.0.0"/>
</Scdl:module>
```

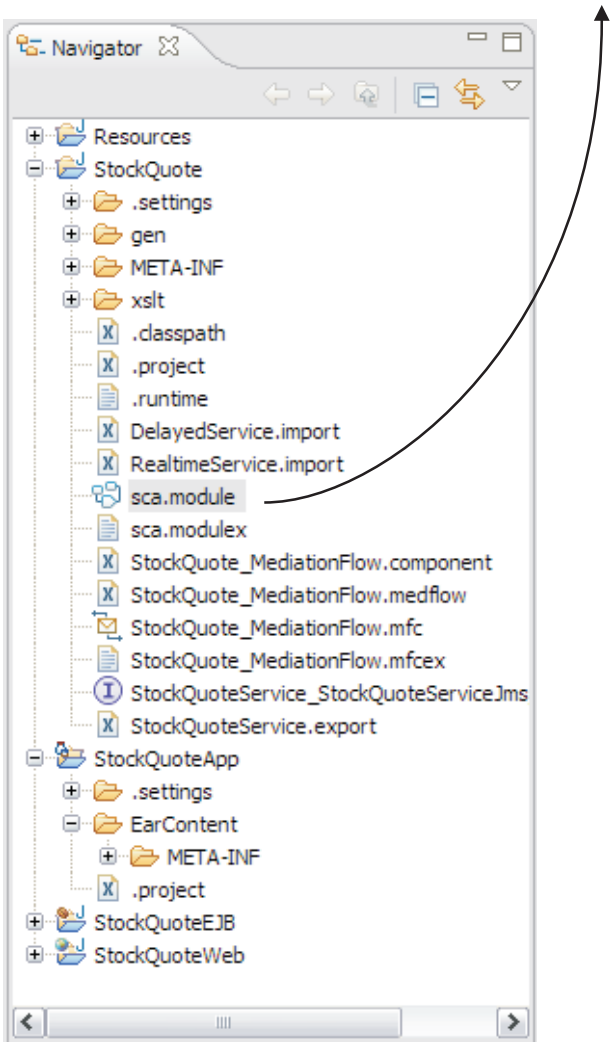


그림 42. WebSphere Integration Developer의 모듈과 모듈 정의 간의 관계 표시

컴포넌트 정의

서비스 컴포넌트 정의는 `<SERVICE_NAME>.component` 파일에 포함됩니다. 해당 연관된 종속성과 함께 SCA 컴포넌트는 정의되어 전개 가능한 단위로 함께 패키징될 수 있습니다.

다음 그림은 서비스 컴포넌트 정의에서의 자세한 형태를 제공합니다. 각 서비스 컴포넌트는 SCA 모듈 내에 고유한 이름을 가지고 있고 모듈 루트에 상대적인 파일 경로와 일치해야 합니다. 이전 슬라이드에 명시된 것처럼, 서비스 컴포넌트 정의는 `<SERVICE_NAME>.component` 파일에 포함됩니다.

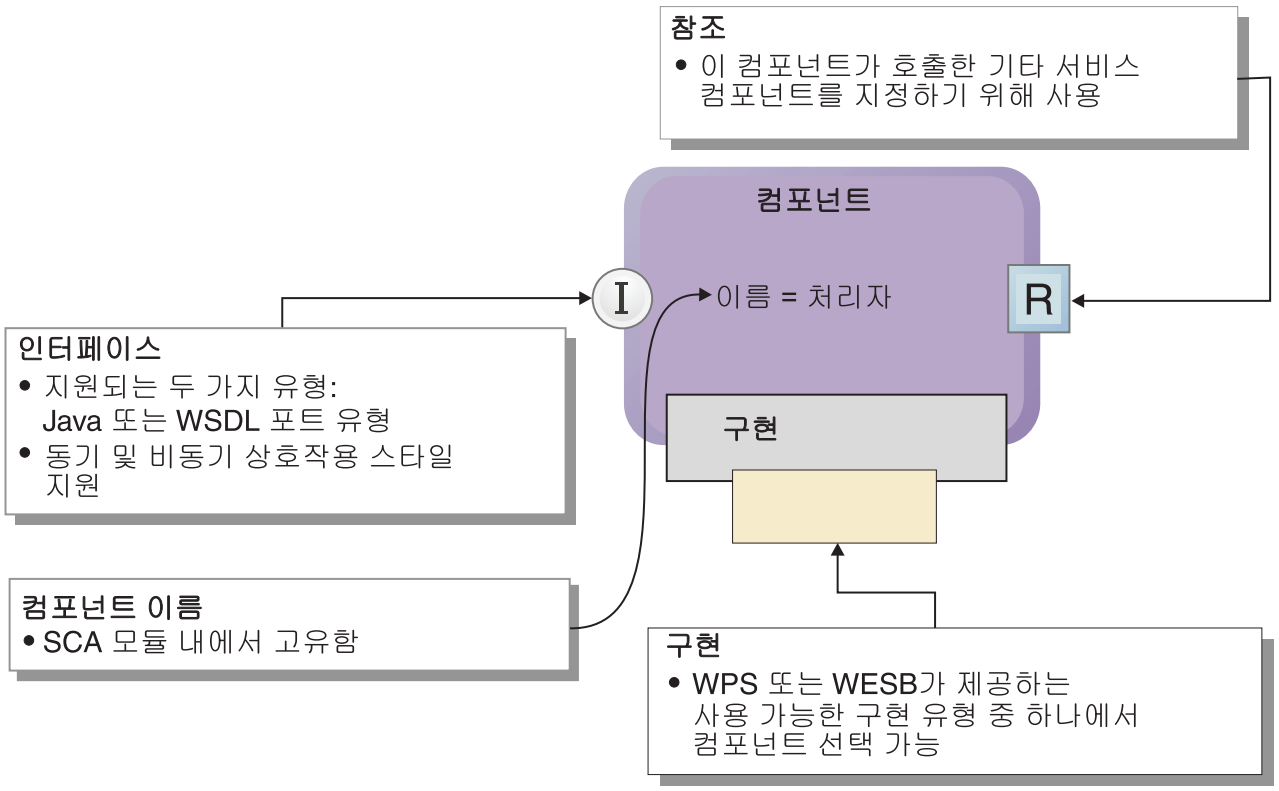
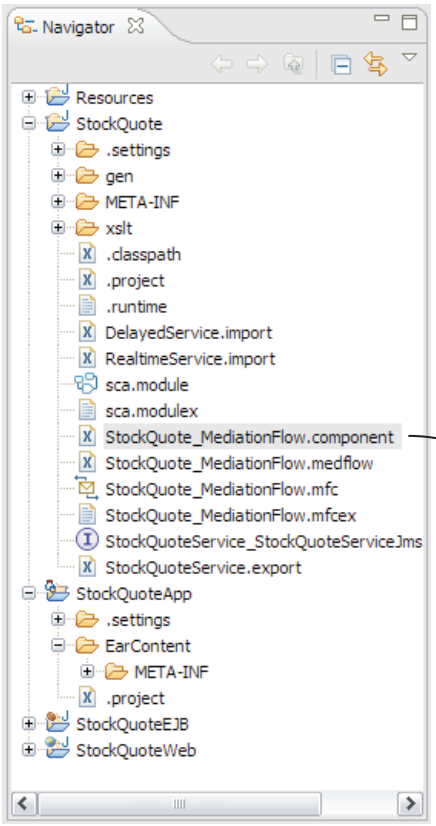


그림 43. 컴포넌트 이름, 구현, 인터페이스 및 참조를 포함하는 서비스 컴포넌트 정의

각 서비스 컴포넌트는 SCA 모듈 내에 고유한 이름을 가지고 있고 모듈 루트에 상대적인 파일 경로와 일치해야 합니다. 다음으로, 각 서비스 컴포넌트에는 하나 이상의 인터페이스가 연관될 수 있습니다. 이 인터페이스는 Java 또는 WSDL 포트 유형 인터페이스 정의가 될 수 있습니다. 서비스 컴포넌트와 연관된 인터페이스는 서비스를 호출하는 클라이언트와의 동기 또는 비동기 상호작용 스타일을 지원할 수 있습니다.

각 서비스 컴포넌트는 구현 정의에 지정된 다양한 방식으로 구현될 수 있습니다. 마지막으로, 서비스 컴포넌트는 현재 서비스 모듈에 정의된 다른 서비스 컴포넌트 또는 가져오기를 호출할 수 있습니다. 이 경우, 사용할 서비스를 표시하기 위해 적절한 참조를 정의해야 합니다. 종종 이 유형의 참조는 독립형 참조 파일에 배치될 수도 있지만 서비스 컴포넌트 정의에 일렬로 놓입니다. 각각의 서비스 컴포넌트 정의는 정의되는 서비스 컴포넌트가 호출하는 다른 서비스에 대한 0개 이상의 참조를 보유할 수 있습니다.

다음은 StockQuote_MediationFlow 컴포넌트에 대한 정의를 표시하는 예제입니다. 컴포넌트에는 WSDL 인터페이스, 두 개의 참조 및 구현에 대한 정의가 포함됩니다.




```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

그림 44. 서비스 컴포넌트 정의 언어로 된 컴포넌트 정의 예제

가져오기 정의

가져오기 정의는 <IMPORT_NAME>.import 파일에 포함됩니다. SCA 가져오기는 SCA 모듈의 클라이언트가 현재 SCA 모듈 외부에 있는 서비스에 액세스할 수 있도록 합니다.

서비스 컴포넌트와 같이, 가져오기는 연관되는 1..N개 인터페이스 세트와 이름을 보유합니다. 가져오기는 또한 외부 서비스가 현재 모듈에 바인드되는 방법을 설명하기 위해 사용되는 바인딩 속성도 보유합니다. 공통 바인딩 유형은 다이어그램에 표시됩니다.

가져오기는 SCA 모듈에서 서비스 컴포넌트의 특수 유형으로 생각할 수 있습니다. 가져오기는 서비스 참조에 대한 연결 정의에서 유효한 대상입니다. 이는 참조사항이 가져오기 또는 다른 서비스 컴포넌트를 지시하는지 여부에 관계없이 대상 서비스를 호출하는 클라이언트에 대해 클라이언트 프로그래밍 모델이 동일함을 의미합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns1="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        xmlns:webservice="http://www.ibm.com/xmlns/prod/websphere/scdl/webservice/7.0.0"
        xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="DelayedService" name="DelayedService">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:DelayedServicePortType">
   <method name="getQuote"/>
  </interface>
 </interfaces>
 <esbBinding xsi:type="webservice:WebServiceImportBinding"
        endpoint="http://localhost:9080/DelayedService/services/DelayedServiceSOAP"
        port="ns1:DelayedServiceSOAP" service="ns1:DelayedService"/>
</scdl:import>
```

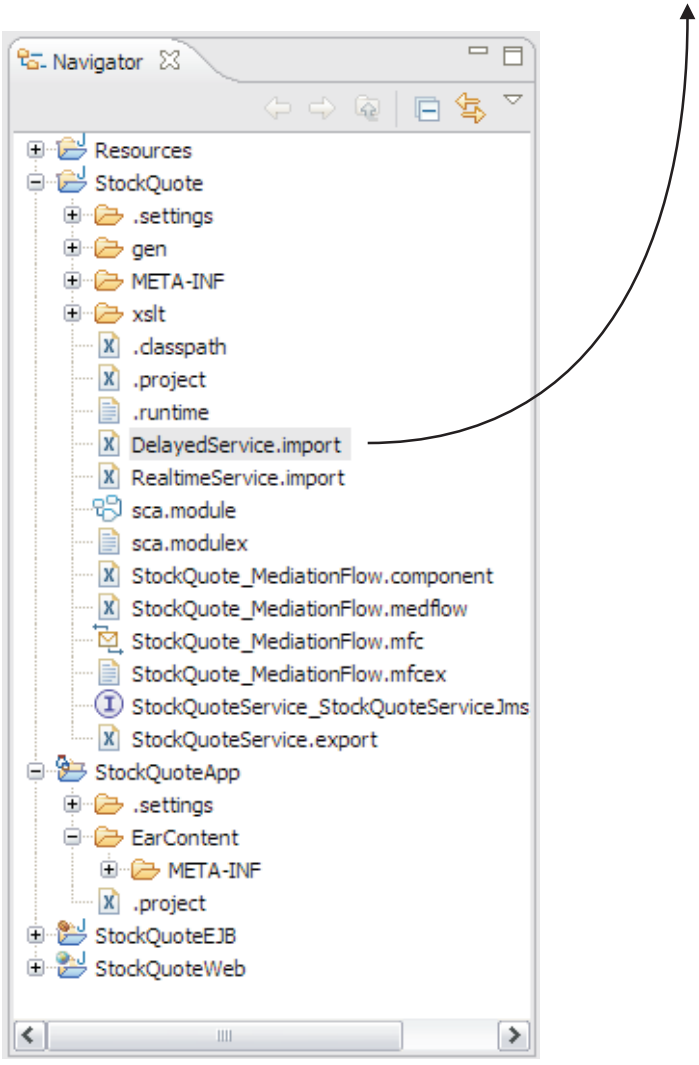
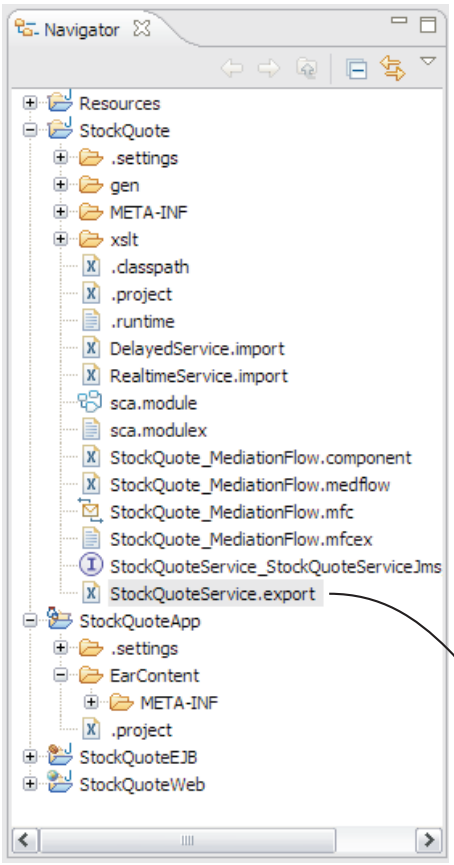


그림 45. 서비스 컴포넌트 정의 언어로 된 가져오기 정의의 예제

내보내기 정의

내보내기 정의는 <EXPORT_NAME>.export 파일에 포함됩니다. SCA 내보내기는 현재 SCA 모듈 외부에서 클라이언트가 사용하도록 SCA 모듈에 정의된 서비스 컴포넌트에 대한 액세스를 제공합니다.

내보내기 컴포넌트에는 내보낼 서비스 컴포넌트의 이름을 지정하는 이름 및 대상 속성이 포함됩니다. 가져오기 컴포넌트와 같이, 내보내기에는 서비스가 외부적으로 바인드되는 방법을 표시하는 바인딩 속성이 있습니다. 공통 바인딩 유형은 다이어그램에 표시됩니다.




```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:_="http://Resources/StockQuoteService/Binding"
xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        xmlns:webservice="http://www.ibm.com/xmlns/prod/websphere/scdl/webservice/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="StockQuoteService" name="StockQuoteService" target="StockQuote_MediationFlow">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService">
   <method name="getQuote"/>
  </interface>
 </interfaces>
```

그림 46. 서비스 컴포넌트 정의 언어로 된 내보내기 정의 예제

참조 정의

서비스 컴포넌트를 호출하는 SCA 및 비SCA 클라이언트에는 호출하기 위해 해당 서비스에 대한 참조가 필요합니다. 참조는 `sca.reference` 파일에서 독립형으로, 또는 서비스 작성 정의 내에서 인라인으로 정의될 수 있습니다.

각 참조에는 클라이언트가 클라이언트 프로그래밍 모델을 사용하여 적절한 서비스를 찾기 위해 사용하는 이름이 있습니다. 이름 외에도, 참조에는 인터페이스 요소가 포함됩

니다. 참조에 대한 다중성은 이 참조를 소스로 이름 지정할 수 있는 연결 정의 수를 표시합니다. 마지막으로, 연결 정의는 참조를 분석하는 가져오기나 대상 서비스 컴포넌트의 이름을 지정합니다.

참조를 정의하는 방법은 두 가지입니다. 첫 번째 방법은 서비스 컴포넌트 정의에 참조를 인라인하는 것입니다. 이 접근방식을 사용하면, 참조가 포함된 서비스 컴포넌트에만 참조를 사용할 수 있습니다. 다른 접근방식은 독립형 참조 파일에 참조 정의를 포함시키는 것입니다. 이 접근방식의 경우 참조는 비SCA 클라이언트나 모듈 내의 다른 컴포넌트에서 사용할 수 있습니다. 독립형 참조 파일에서 참조를 사용하는 비SCA 컴포넌트의 예로는 특정 서비스를 호출할 수 있는 기능이 필요한 JSP와 같은 사용자 인터페이스 컴포넌트가 있습니다. 서비스 컴포넌트를 호출하기 위해, 클라이언트는 SCA 런타임을 사용하여 호출할 적절한 서비스를 찾을 수 있도록 참조를 필요로 합니다.

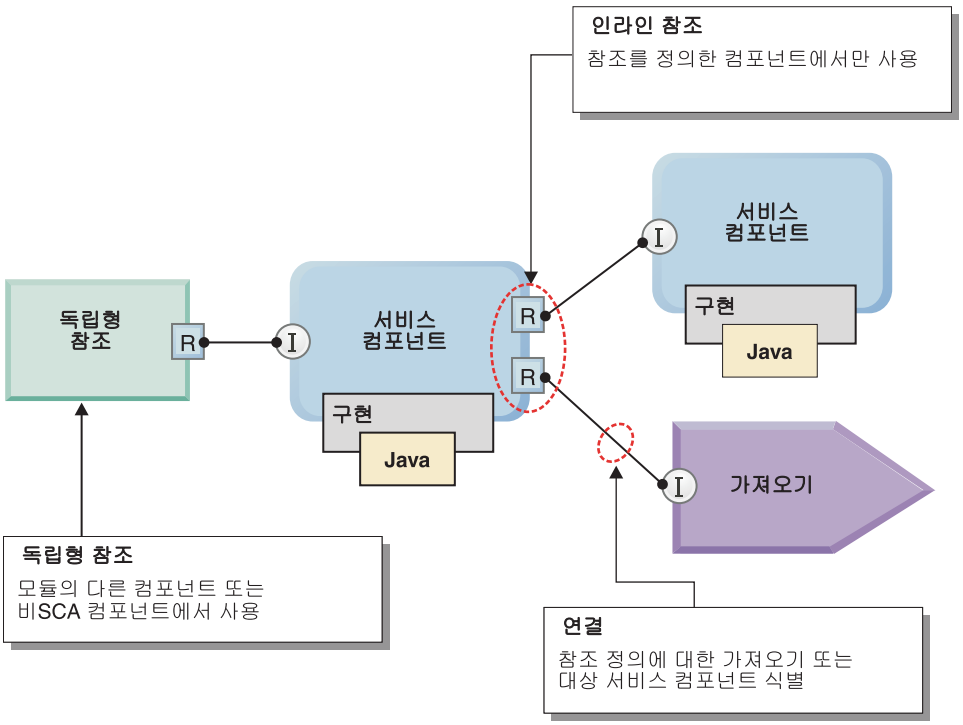


그림 47. 클라이언트는 독립형 또는 인라인 참조를 사용하여 서비스 컴포넌트를 호출할 수 있음

다음은 두 개의 인라인 참조인 DelayedServicePortTypePartner 및 RealtimeServicePortTypePartner를 포함하는 컴포넌트에 대한 정의를 보여주는 예제입니다. 컴포넌트에는 WSDL 인터페이스, 두 개의 참조 및 구현에 대한 정의가 포함됩니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        Xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        Xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </Interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

그림 48. 인라인 참조 정의 예제

SCA 프로그래밍 모델 기본 원칙

소프트웨어 *컴포넌트*의 개념은 SCA(Service Component Architecture) 프로그래밍 모델의 기초를 형성합니다. 컴포넌트는 일부 논리를 구현하고 인터페이스를 통해 기타 컴포넌트를 사용할 수 있도록 하는 단위입니다. 또한 컴포넌트는 다른 컴포넌트에서 서비스를 사용할 수 있도록 요청합니다. 이 경우, 컴포넌트는 해당 서비스에 대한 *참조*를 나타냅니다.

SCA에서, 모든 컴포넌트는 최소한 하나의 인터페이스를 나타내야 합니다. 132 페이지의 그림 49에 표시된 어셈블리 다이어그램에는 세 개의 컴포넌트가 있습니다. 각 컴포넌트에는 원 안의 문자 I로 표시되는 인터페이스가 있습니다. 컴포넌트는 또한 다른 컴포넌트를 참조할 수 있습니다. 참조는 정사각형 안의 문자 R로 표시됩니다. 참조 및 인터페이스는 어셈블리 다이어그램에서 링크됩니다. 기본적으로, 통합 개발자는 필요한 로직을 구현하는 컴포넌트의 인터페이스와 컴포넌트를 연결함으로써 참조를 "결정"합니다.

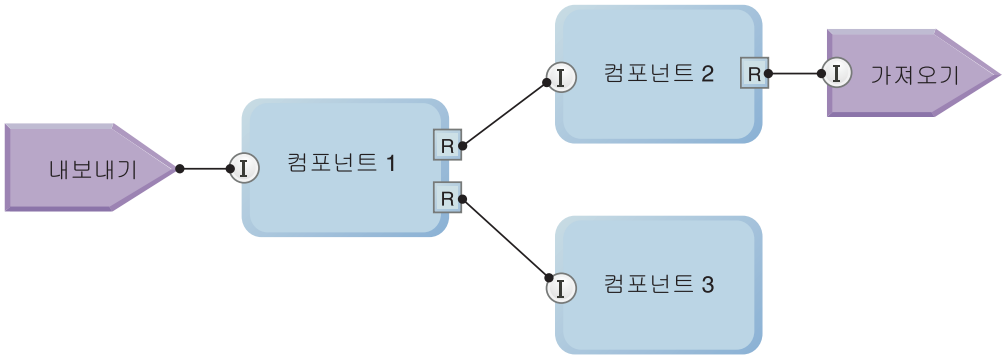


그림 49. 어셈블리 다이어그램

SCA 컴포넌트 호출

호출되는 서비스에 대한 액세스를 제공하기 위해, SCA 프로그래밍 모델은 *ServiceManager* 클래스를 포함하므로 개발자가 사용 가능한 서비스를 이름별로 찾을 수 있습니다. 여기에는 서비스 검색을 예시하는 일반적인 Java 코드 단편이 있습니다. ServiceManager는 시스템이 제공하는 서비스인 BOFactory 서비스에 대한 참조를 얻는 데 사용됩니다.

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
        smgr.locateService("com/ibm/websphere/bo/BOFactory");
```

주: ServiceManager의 패키지는 `com.ibm.websphere.sca`입니다.

locateService 메소드에서 참조되는 서비스의 이름을 지정하여 개발자는 자신의 서비스에 대한 참조를 얻기 위해 유사한 메커니즘을 사용할 수 있습니다. ServiceManager 클래스를 사용하여 서비스에 대한 참조를 얻은 후에, 호출 프로토콜 및 구현 유형과 독립적인 방식으로 해당 서비스에서 사용 가능한 조작을 호출할 수 있습니다.

세 가지 다른 호출 스타일을 사용하여 SCA 컴포넌트를 호출할 수 있습니다.

- **동기 호출**: 이 호출 스타일 사용 시, 호출자는 동기적으로 리턴되는 응답을 대기합니다. 이 스타일은 전통적인 호출 메커니즘입니다.
- **비동기 호출**: 이 메커니즘을 통해 호출자는 올바른 방식으로 응답이 생성될 때까지 대기하지 않고 서비스를 호출할 수 있습니다. 응답을 얻는 대신, 호출자는 응답을 검색하기 위해 나중에 사용될 수 있는 "티켓"을 얻습니다. 호출자는 이 목적으로 수신자(callee)가 제공해야 하는 특수 조작을 호출하여 응답을 검색합니다.
- **콜백을 이용한 비동기 호출**: 이 호출 스타일은 이전 호출 스타일과 유사하지만 응답을 리턴하는 책임을 수신자에게 위임합니다. 응답이 준비되었을 때 수신자가 호출할 수 있는 특정 조작(콜백 조작)을 호출자가 나타낼 필요가 있습니다.

- **지연된 응답을 이용한 비동기 호출**: 이 호출 스타일에서 클라이언트는 서비스를 호출한 다음 나중에 클라이언트가 응답을 캡처하도록 요청할 때까지 처리를 계속합니다.

가져오기

레거시 응용프로그램이나 기타 외부 구현과 같이, 외부 시스템에서 사용 가능한 기능이나 컴포넌트에서 비즈니스 로직을 제공하는 경우가 있습니다. 이 경우, 통합 개발자는 외부 구현을 "가리키는" 컴포넌트에 대한 참조를 연결하는 데 필요한 구현을 포함하는 컴포넌트에 대한 참조를 연결하여 참조를 해결할 수 없습니다. 그러한 컴포넌트를 *가져오기*라고 합니다. 가져오기 정의 시, 위치 및 호출 프로토콜의 측면에서 외부 서비스가 액세스될 수 있는 방법을 지정해야 합니다.

내보내기

유사하게 컴포넌트가 외부 응용프로그램에 의해서 액세스되어야 하는 경우가 종종 발생하는데 이때 액세스가 가능하도록 설정해야 합니다. 논리를 "외부 세계"에 표시하는 특수한 컴포넌트를 사용해서 액세스가 가능하도록 합니다. 이러한 컴포넌트를 *내보내기*라고 합니다. 내보내기도 동기 또는 비동기적으로 호출될 수 있습니다.

독립형 참조

WebSphere Process Server에서 SCA 서비스 모듈은 여러 다른 Java EE 서브모듈을 포함하는 Java EE EAR 파일로 패키지됩니다. WAR 파일과 같은 Java EE 요소는 SCA 모듈과 함께 패키지될 수 있습니다. JSP와 같은 비SCA 아티팩트는 SCA 서비스 모듈과 함께 패키지될 수도 있습니다. 이 패키징은 독립형 참조라고 하는 특수한 컴포넌트 유형을 사용하여 SCA 클라이언트 프로그래밍 모델을 통해 SCA 서비스를 호출하게 합니다.

SCA 프로그래밍 모델은 매우 설명적입니다. 통합 개발자는 어셈블리 다이어그램에서 직접 선언적인 방식으로 동기 또는 비동기적으로 호출되어야 하는지 여부와 관계없이 보안 신임의 호출, 전파의 트랜잭션 동작과 같은 측면을 구성할 수 있습니다. 개발자가 아닌 SCA 런타임이 이 수정자에서 지정된 동작을 구현하는 데 관여합니다. SCA의 선언적인 유연성은 이 프로그래밍 모델에서 가장 강력한 기능 중 하나입니다. 개발자는 비동기 호출 메커니즘을 수용할 수 있는 것과 같이, 주소 지정 기술 측면에 중점을 두기보다는 비즈니스 로직 구현에 집중할 수 있습니다. 이러한 모든 측면은 SCA 런타임에 의해 자동으로 수행됩니다.

규정자

규정자는 서비스 클라이언트와 대상 서비스 간에 상호작용을 제어합니다. 규정자는 서비스 컴포넌트 참조, 인터페이스 및 구현에서 지정될 수 있으며 보통 구현과는 무관합니다.

규정자의 다른 카테고리는 다음을 포함합니다.

- 트랜잭션 컨텍스트가 SCA 호출에서 핸들되는 방식을 지정하는 트랜잭션
- 활동 세션 컨텍스트가 전파되는 방식을 지정하는 활동 세션
- 사용 권한을 지정하는 보안
- 비동기 메시지 전달을 위한 규칙을 제공하는 비동기 신뢰도

SCA는 프로그래밍을 수행하거나 서비스 구현 코드를 변경하지 않고 서비스 품질(QoS) 규정자가 선언적으로 컴포넌트에 적용되게 합니다. WebSphere Integration Developer를 사용해서 서비스 규정자를 추가할 수 있습니다. 일반적으로 솔루션 전개를 고려할 준비가 된 경우에 QoS 규정자를 적용합니다.

클라이언트 프로그래밍 모델

SCA 클라이언트 프로그래밍 모델은 서비스를 찾고, 데이터 오브젝트를 작성하며, 서비스를 호출하고, 호출된 컴포넌트에 의해 발생되는 예외를 처리하도록 설계되었습니다.

SCA 클라이언트 프로그래밍 모델은 클라이언트에 대해 두 가지의 1차 기능을 제공합니다. 프로그래밍 모델은 클라이언트가 현재 모듈 내에서 서비스를 찾을 수 있도록 하는 인터페이스를 노출하고, 서비스를 찾으면 클라이언트 프로그래밍 모델은 클라이언트가 해당 서비스에 대해 조작을 호출할 수 있는 방법을 제공합니다.

클라이언트는 ServiceManager 클래스를 사용하여 서비스를 찾습니다. 서비스에 대해 원하는 검색 범위에 따라 ServiceManager 클래스를 인스턴스화할 몇 가지 방법이 있습니다.

클라이언트가 서비스를 찾기 위해 유의해야 하는 주요 인터페이스는 `com.ibm.websphere.sca.ServiceManager`입니다. 이 인터페이스에는 요청된 서비스에 대한 서비스 구현으로 참조를 리턴하는 locateService 메소드가 포함됩니다. locateService 메소드로 전달되는 문자열 매개변수는 클라이언트가 찾으려고 하는 서비스에 대한 참조 이름을 나타냅니다. SCA 프로그래밍 모델에 대한 Java 문서는 WebSphere Process Server Information Center에 포함되며, Java 문서를 WebSphere Process Server 설치의 일부로 설치할 것을 선택한 경우에도 포함됩니다.

클라이언트가 적절한 서비스를 찾으면, 서비스가 제공하는 메소드 또는 조작에 대해 호출을 작성하기 위해 사용할 수 있는 두 가지 유형의 호출 모델이 있습니다. 먼저, *동적 호출 스타일*의 상호작용이 있습니다. 이 스타일의 상호작용에 대한 주요 인터페이스는 `com.ibm.websphere.sca.Service`입니다. 이 인터페이스의 invoke() 메소드는 해당 조작을 호출하기 위해 필요한 매개변수와 함께 호출하려고 하는 조작의 이름을 사용합니다.

```
public Interface MyService {
public String someMethod(String input);
```

```
Service myService = (Service) serviceManager.locateService(myService);
DataObject input = ...
DataObject result = (DataObject) myService.invoke(someMethod, input);
```

클라이언트는 또한 *정적(유형 안전)* 호출 메소드를 사용하여 서비스와 연관된 특정 조작을 호출할 수 있습니다. 이 유형의 호출은 Java로 지정된 인터페이스 정의에 대해서만 작동합니다. 이 상황에서, 클라이언트는 locateService() 호출에서 적절한 인터페이스로의 리턴을 캐스트하고 해당 인터페이스에 대해 적절한 유형 안전성(type safe) 메소드 호출을 진행할 수 있습니다.

```
public Interface MyService {
public String someMethod(String input);

MyService myService = (MyService) serviceManager.locateService(myService);
String input = ...
String result = myService.someMethod(input);
```

인터페이스

서비스 컴포넌트에는 연관된 하나 이상의 인터페이스가 있습니다. 서비스 컴포넌트와 연관된 인터페이스는 이 서비스와 연관된 비즈니스 조작을 알립니다.

모든 컴포넌트에는 WSDL 유형의 인터페이스가 있습니다. Java 컴포넌트만 Java 유형 인터페이스를 지원합니다. 컴포넌트, 가져오기 또는 내보내기에 둘 이상의 인터페이스가 있는 경우 모든 인터페이스는 동일한 유형이어야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://HelloWorldLibrary/HelloWorld" xmlns:ns2="http://HelloService/HelloService"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="HelloWorldMediation" name="HelloWorldMediation">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:HelloWorld">
   <scdl:interfaceQualifier xsi:type="scdl:JoinTransaction" value="true"/>
  </Interface>
 </interfaces>
 <references>
  <reference name="HelloServicePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:HelloService"/>
   <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction" value="false"/>
   <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
   <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt" value="commit"/>
   <wire target="HelloServiceImport"/>
  </Reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="HelloWorldMediation.mfc">
  <scdl:implementationQualifier xsi:type="scdl:Transaction" value="global"/>7
```

그림 50. SCDL 컴포넌트 정의 내의 인터페이스 정의 예제

이 인터페이스는 Java 인터페이스나 WSDL 포트 유형 인터페이스로 지정할 수 있습니다. 그러나 동일한 서비스 컴포넌트 정의에 대해 Java 및 WSDL 포트 유형 인터페이

스를 혼합할 수 없습니다. 이러한 인터페이스에 대한 인수 및 리턴 유형은 단순 Java 유형, Java 클래스, Service DataObject 또는 XML 스키마(WSDL 포트 유형 인터페이스의 경우)로 지정됩니다.

컴포넌트는 동기식 또는 비동기식으로 호출될 수 있습니다. 이는 구현의 동기식 또는 비동기식 여부에는 관계없습니다. 컴포넌트 인터페이스는 동기식 양식으로 정의되며 비동기식 지원도 생성됩니다. 선호하는 상호작용 스타일을 동기식 또는 비동기식으로 지정할 수 있습니다. 비동기 유형은 인터페이스 사용자에게 완료하는 데 많은 시간이 소요될 수 있는 하나 이상의 조작이 포함되어 있음을 알립니다. 결과적으로, 호출하는 서비스는 조작이 완료되고 해당 응답을 전송하기를 기다리는 동안 트랜잭션을 열린 상태로 유지하면 안됩니다. 상호작용 스타일은 인터페이스의 모든 조작에 적용됩니다.

다른 모듈에 있는 컴포넌트는 해당 인터페이스를 보유하는 내보내기로 서비스를 공개하고 필수 어셈블리 다이어그램으로 내보내기를 끌어와서 함께 연결하여 가져오기를 작성할 수 있습니다. 컴포넌트를 연결할 때 구현, 상대 참조 및 컴포넌트의 인터페이스에 서비스 품질 규정자를 지정할 수도 있습니다.

가져오기에는 해당되는 원격 서비스를 호출할 수 있도록 연관되어 있는 원격 서비스의 인터페이스 서브세트와 동일한 인터페이스가 있습니다. 가져오기는 로컬 컴포넌트와 정확히 동일한 방식으로 응용프로그램에서 사용됩니다. 이로서 해당 위치 또는 구현에 관계없이 모든 기능에 대해 일정한 어셈블리 모델이 제공됩니다. 가져오기 바인딩은 개발할 때 정의하지 않아도 됩니다. 전개 시 수행될 수 있습니다.

내보내기에는 공개된 서비스를 호출할 수 있도록 연관되어 있는 컴포넌트의 인터페이스 서브세트와 동일한 인터페이스가 있습니다. 다른 모듈에서 어셈블리 다이어그램으로 끌어온 내보내기는 자동으로 가져오기를 작성합니다.

서비스 모듈 개발

서비스 컴포넌트는 서비스 모듈에 포함되어 있어야 합니다. 서비스 컴포넌트를 포함하도록 서비스 모듈을 개발하면 다른 모듈에 서비스를 제공할 수 있습니다.

시작하기 전에

이 타스크에서는 요구사항 분석이 다른 모듈에서 사용하도록 서비스 컴포넌트를 구현하는 것이 유용하다는 것을 보여주고 있다고 가정합니다.

이 태스크 정보

요구사항을 분석한 후, 서비스 컴포넌트를 제공하고 사용하는 것이 정보 처리에 효율적인 방법이라는 것을 결정할 수 있습니다. 재사용 가능한 서비스 컴포넌트가 사용자 환경에 유용하다고 판별되면 서비스 컴포넌트를 포함하는 서비스 모듈을 작성하십시오.

프로시저

1. 다른 모듈이 사용할 수 있는 서비스 컴포넌트를 식별하십시오.

   서비스 컴포넌트를 식별한 후 『서비스 컴포넌트 개발』을 계속 수행하십시오.
2. 다른 서비스 모듈에 있는 서비스 컴포넌트를 사용할 수 있는 응용프로그램 내에서 서비스 컴포넌트를 식별하십시오.

   서비스 컴포넌트와 대상 컴포넌트를 식별한 후 『컴포넌트 호출』 또는 『동적으로 컴포넌트 호출』을 계속 수행하십시오.
3. 클라이언트 컴포넌트와 대상 컴포넌트를 연결하십시오.

모듈 개발 개요:

모듈은 WebSphere Process Server 응용프로그램의 기본 전개 단위입니다. 모듈은 응용프로그램에서 사용하는 컴포넌트, 라이브러리 및 스테이징 모듈을 포함할 수 있습니다.

모듈 개발은 응용프로그램에서 필요한 컴포넌트, 스테이징 모듈 및 라이브러리(모듈에서 참조되는 라이브러리 콜렉션)를 프로덕션 서버에서 사용할 수 있는지 확인하는 데 관련이 있습니다.

WebSphere Integration Developer는 WebSphere Process Server로 전개하기 위한 모듈을 개발하기 위한 기본 도구입니다. 다른 환경에서 모듈을 개발할 수 있지만, WebSphere Integration Developer를 사용하는 것이 가장 좋습니다.

WebSphere Process Server는 비즈니스 서비스의 모듈과 중개 모듈을 지원합니다. 모듈 및 중개 모듈 둘 다 SCA(Service Component Architecture) 모듈 유형입니다. 중개 모듈을 사용하면 서비스 호출을 대상이 이해하는 형식으로 변환하여 응용프로그램 간 통신을 할 수 있으며 목표로 요청을 전달하고 그 결과를 작성자에게 리턴할 수 있습니다. 비즈니스 서비스의 모듈은 비즈니스 프로세스의 로직을 구현합니다. 그러나 모듈은 중개 모듈에서 패키지될 수 있는 동일한 중개 로직을 포함할 수도 있습니다.

다음 절에서는 WebSphere Process Server의 모듈을 구현하고 갱신하는 방법에 대해 설명합니다.

컴포넌트

SCA 모듈은 컴포넌트는 재사용 가능한 비즈니스 로직을 캡슐화하는 기본 빌딩 블록인 컴포넌트를 포함합니다. 컴포넌트는 서비스를 제공하고 이용하며 인터페이스, 참조 및 구현과 연관됩니다. 인터페이스는 서비스 컴포넌트와 호출 컴포넌트 간의 계약을 정의합니다.

WebSphere Process Server를 사용하여, 모듈은 다른 모듈에서 서비스 컴포넌트를 사용할 수 있도록 내보내거나 사용할 서비스 컴포넌트를 가져올 수 있습니다. 서비스 컴

포넌트를 호출하기 위해 호출 모듈은 서비스 컴포넌트에 대한 인터페이스를 참조합니다. 인터페이스 참조는 호출 모듈에서 연관되는 인터페이스 참조를 구성하여 해석됩니다.

모듈을 개발하려면 다음 활동을 실행해야 합니다.

1. 모듈에 컴포넌트에 대한 인터페이스를 정의하거나 식별하십시오.
2. 컴포넌트에서 사용한 비즈니스 오브젝트를 정의하거나 조작하십시오.
3. 인터페이스를 통해 컴포넌트를 정의하거나 수정하십시오.

   주: 인터페이스를 통해 컴포넌트가 정의됩니다.
4. 옵션: 서비스 컴포넌트를 내보내거나 가져오십시오.
5. 런타임에 전개할 엔터프라이즈 아카이브(EAR) 파일을 작성하십시오. WebSphere Integration Developer의 내보내기 EAR 기능이나 serviceDeploy 명령 중 하나를 사용하여 파일을 작성합니다.

개발 유형

WebSphere Process Server는 서비스 지향 프로그래밍 패러다임을 이용하도록 컴포넌트 프로그래밍 모델을 제공합니다. 이 모델을 사용하기 위해 프로바이더는 서비스 컴포넌트의 인터페이스를 내보내서 처리자가 이들 인터페이스를 가져와 이 서비스 컴포넌트를 마치 로컬에 있는 것처럼 사용할 수 있도록 합니다. 개발자는 엄격한 유형의 인터페이스 또는 동적 유형의 인터페이스를 사용하여 서비스 컴포넌트를 구현하거나 호출합니다. 인터페이스와 해당 메소드는 Information Center의 참조 절에 설명되어 있습니다.

서버에 서비스 모듈을 설치한 후 관리 콘솔을 사용하여 응용프로그램에서 참조하도록 대상 컴포넌트를 변경할 수 있습니다. 새로 지정한 대상은 동일한 비즈니스 오브젝트 유형을 허용해야 하며 응용프로그램의 참조에서 요청하는 동일한 조작을 수행해야 합니다.

서비스 컴포넌트 개발 고려사항

서비스 컴포넌트를 개발할 때 다음 사항을 확인하십시오.

- 이 서비스 컴포넌트를 다른 모듈로 내보내어 사용하시겠습니까?

  그런 경우, 컴포넌트에 대해 정의하는 인터페이스가 다른 모듈에서도 사용될 수 있도록 해야 합니다.
- 서비스 컴포넌트를 실행하는 데 비교적 오랜 시간이 소요됩니까?

  그런 경우, 서비스 컴포넌트에 대한 비동기 인터페이스의 구현을 고려하십시오.
- 서비스 컴포넌트를 분산시키는 것이 유용합니까?

  그런 경우, 서버의 클러스터에 전개되는 서비스 모듈의 서비스 컴포넌트 사본을 작성하여 병렬 처리를 이용할 것을 고려하십시오.

- 응용프로그램에서 1단계 및 2단계 커미트 자원의 혼합이 필요합니까?

  그런 경우, 응용프로그램에 대한 마지막 참여자 지원을 사용할 수 있게 해야 합니다.

  주: WebSphere Integration Developer를 사용하여 응용프로그램을 작성하거나 serviceDeploy 명령을 사용하여 설치 가능한 EAR 파일을 작성할 경우, 이 도구는 자동으로 응용프로그램에 대한 지원을 사용 가능하게 합니다. WebSphere Application Server Network Deployment Information Center의 『동일한 트랜잭션에서 1단계 및 2단계 커미트 자원 사용』 주제를 참조하십시오.

서비스 컴포넌트 개발:

서버에 있는 다중 응용프로그램에 재사용 가능한 로직을 제공하도록 서비스 컴포넌트를 개발하십시오.

시작하기 전에

이 타스크는 다중 모듈에 유용한 처리를 이미 개발 및 식별한 것으로 가정합니다.

이 태스크 정보

다중 모듈이 하나의 서비스 컴포넌트를 사용할 수 있습니다. 서비스 컴포넌트를 내보내면 인터페이스를 통해 서비스 컴포넌트를 참조하는 다른 모듈에서도 사용할 수 있습니다. 이 타스크는 서비스 컴포넌트를 빌드하여 다른 모듈이 사용할 수 있도록 하는 방법을 설명합니다.

주: 단일 서비스 컴포넌트에 다중 인터페이스를 포함시킬 수 있습니다.

프로시저

1. 호출자와 서비스 컴포넌트 간에 데이터를 이동시키기 위한 데이터 오브젝트를 정의하십시오.

   데이터 오브젝트 및 해당 유형은 호출자와 서비스 컴포넌트 간의 인터페이스의 일부분입니다.
2. 호출자가 서비스 컴포넌트를 참조하는 데 사용할 인터페이스를 정의하십시오.

   이 인터페이스 정의는 서비스 컴포넌트 이름을 지정하고 서비스 컴포넌트에서 사용할 수 있는 모든 메소드를 표시합니다.
3. 서비스 호출을 구현하는 클래스를 생성하십시오.
4. 생성된 클래스의 구현을 개발하십시오.
5. 컴포넌트 인터페이스 및 구현을 .java 확장자를 갖는 파일로 저장하십시오.
6. 서비스 모듈 및 필수 자원을 JAR 파일로 패키지하십시오.

이 Information Center의 『프로덕션 서버로 모듈 전개』에서 139 페이지의 6 - 8 단계의 설명을 참조하십시오.

7. serviceDeploy 명령을 실행하여 응용프로그램을 포함하는 설치 가능한 EAR 파일을 작성하십시오.
8. 서버 노드에 응용프로그램을 설치하십시오.
9. 옵션: 다른 서비스 모듈에 있는 서비스 컴포넌트를 호출하는 경우 호출자와 해당 서비스 컴포넌트 간의 연결을 구성하십시오.

   이 Information Center의 『관리』 절에 연결 구성 방법이 설명되어 있습니다.

컴포넌트 개발 예제

이 예에서는 단일 메소드인 CustomerInfo를 구현하는 동기 서비스 컴포넌트를 보여줍니다. 첫 번째 절은 getCustomerInfo 메소드를 구현하는 서비스 컴포넌트에 대한 인터페이스를 정의합니다.

```
public interface CustomerInfo {
 public Customer getCustomerInfo(String customerID);
}
```

다음 코드 블록은 서비스 컴포넌트를 구현합니다.

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

x

다음 절은 StockQuote와 연관된 클래스의 구현입니다.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

다음에 수행할 작업

서비스를 호출하십시오.

컴포넌트 호출:

모듈이 있는 컴포넌트는 WebSphere Process Server 클러스터의 모든 노드에서 컴포넌트를 사용할 수 있습니다.

시작하기 전에

컴포넌트를 호출하기 전에 컴포넌트가 들어 있는 모듈이 WebSphere Process Server에 설치되어 있는지 확인하십시오.

이 태스크 정보

컴포넌트는 컴포넌트 이름을 사용하고 컴포넌트가 예상하는 데이터 유형을 전달하여 WebSphere Process Server 클러스터에서 사용할 수 있는 모든 서비스 컴포넌트를 사용할 수 있습니다. 이 환경에서 컴포넌트를 호출하는 것은 필요한 컴포넌트에 대한 참조를 찾아 작성하는 것을 포함합니다.

주: 모듈에 있는 컴포넌트는 같은 모듈에 있는 컴포넌트를 호출할 수 있는데, 이를 모듈 내 호출이라고 합니다. 제공 컴포넌트에 있는 인터페이스를 내보내고 이 인터페이스를 호출 컴포넌트에 가져옴으로써 외부 호출(모듈 간 호출)을 구현합니다.

중요사항: 호출 모듈이 실행 중인 서버와 다른 서버에 있는 컴포넌트를 호출할 때 이들 서버에 추가 구성을 수행해야 합니다. 필요한 구성은 컴포넌트가 비동기 또는 동기로 호출되는지에 따라 다릅니다. 이 경우 Application Server를 구성하는 방법은 관련 타스크에 설명되어 있습니다.

프로시저

1. 호출 모듈에서 필요한 컴포넌트를 결정하십시오.

   컴포넌트의 인터페이스 이름 및 인터페이스에 필요한 데이터 유형을 참고하십시오.
2. 데이터 오브젝트를 정의하십시오.

   입력 또는 리턴이 Java 클래스일 수는 있지만 서비스 데이터 오브젝트가 최적입니다.
3. 컴포넌트를 찾으십시오.
   a. ServiceManager 클래스를 사용하여 호출 모듈에 사용 가능한 참조를 얻으십시오.
   b. locateService() 메소드를 사용하여 컴포넌트를 찾으십시오.

컴포넌트에 따라 인터페이스는 WSDL(Web Service Descriptor Language) 포트 유형 또는 Java 인터페이스가 될 수 있습니다.

4. 컴포넌트를 동기적으로 호출하십시오.

   Java 인터페이스를 통해 컴포넌트를 호출하거나 invoke() 메소드를 사용하여 컴포넌트를 동적으로 호출할 수 있습니다.

5. 리턴을 처리하십시오.

   컴포넌트는 예외를 생성할 수 있으므로 클라이언트가 이러한 가능성을 처리할 수 있어야 합니다.

호출 컴포넌트의 예제

다음 예에서 ServiceManager 클래스를 작성합니다.

```
ServiceManager serviceManager = new ServiceManager();
```

다음 예제에서는 ServiceManager 클래스를 사용하여 컴포넌트 참조를 포함하는 파일에서 컴포넌트 목록을 가져옵니다.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

다음 코드는 StockQuote Java 인터페이스를 구현하는 컴포넌트를 찾습니다.

```
StockQuote stockQuote =
(StockQuote)serviceManager.locateService("stockQuote");
```

다음 코드는 Java 또는 WSDL 포트 유형 인터페이스를 구현하는 컴포넌트를 찾습니다. 호출 모듈은 Service 인터페이스를 사용하여 컴포넌트와 상호작용합니다.

팁: 컴포넌트가 Java 인터페이스를 구현하는 경우 컴포넌트는 인터페이스 또는 invoke() 메소드를 통해 호출될 수 있습니다.

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

다음 예에서 다른 컴포넌트를 호출하는 코드인 MyValue를 보여줍니다.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

         // invoke
         CustomerInfo cInfo =
     (StockQuote)serviceManager.locateService("stockQuote");
         customer = cInfo.getCustomerInfo(customerID);
```

```
     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuote sQuote =
      (StockQuote)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuote(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // assign
          value = quote * customer.getNumShares();
     } else {

         // throw
         throw new MyValueException(customer.getErrorMsg());
     }
     // reply
         return value;
 }
}
```

다음에 수행할 작업

호출 모듈 참조와 컴포넌트 인터페이스의 연결을 구성합니다.

동적으로 컴포넌트 호출:

모듈이 WSDL(Web Service Descriptor Language) 포트 유형 인터페이스가 포함된 컴포넌트를 호출할 때 모듈은 invoke() 메소드를 사용하여 동적으로 컴포넌트를 호출해야 합니다.

시작하기 전에

이 타스크에서는 호출 컴포넌트가 컴포넌트를 동적으로 호출한다고 가정합니다.

이 태스크 정보

WSDL 포트 유형 인터페이스에서 호출 컴포넌트는 invoke() 메소드를 사용하여 컴포넌트를 호출해야 합니다. 이러한 방법으로 호출 모듈은 Java 인터페이스가 있는 컴포넌트를 호출할 수도 있습니다.

프로시저

1. 필수 컴포넌트를 포함하는 모듈을 판별하십시오.
2. 컴포넌트에서 필요한 배열을 판별하십시오.

   입력 배열은 다음 세 유형 중 하나입니다.
   - 기본 대문자 Java 유형 또는 유형의 배열
   - 정규 Java 클래스 또는 클래스의 배열
   - 서비스 데이터 오브젝트(SDO)

3. 컴포넌트의 응답을 포함하도록 배열을 정의하십시오.

   응답 배열은 입력 배열과 동일한 유형일 수 있습니다.
4. invoke() 메소드를 사용하여 필수 컴포넌트를 호출하고 배열 오브젝트를 컴포넌트로 전달하십시오.
5. 결과를 처리하십시오.

컴포넌트 동적 호출 예

다음 예에서는 모듈이 invoke() 메소드를 사용하여 기본 대문자 Java 데이터 유형을 사용하는 컴포넌트를 호출합니다.

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();

  BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

  DataObject paramObject = boFactory.createbyType(t);

  paramObject.set(0,"input1")
  paramObject.set(1,"input2")
  paramObject.set(2,"input3")

  service.invoke("methodMultiParamater",paramObject);
```

다음 예는 WSDL 포트 유형 인터페이스를 대상으로 가지는 호출 메소드를 사용합니다.

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

 DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
   dob.setString("attribute1", stringArg);

 DataObject wrapBo = factory.createByElement
  ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
   wrapBo.set("input1", dob); //wrapBo encapsulates all the parameters of methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

 DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

호출 스타일

SCA에서, 동기 및 비동기 프로그래밍 스타일을 사용하여 서비스 컴포넌트를 호출할 수 있습니다. 서비스 컴포넌트 및 모듈 사이의 비동기 채널이 시스템의 전체 처리량 및 유연성을 증가시킬 수 있는 전체 솔루션으로 모듈을 어셈블할 수 있습니다.

컴포넌트는 서비스가 사용되거나 호출될 수 있도록 해당 응용프로그램 비즈니스 로직에 비즈니스 레벨 인터페이스를 노출합니다. 컴포넌트의 인터페이스는 호출할 수 있는 조작과 전달되는 데이터(예: 입력 인수, 리턴값 및 예외)를 정의합니다. 가져오기 및 내보내기에도 공개된 서비스를 호출할 수 있도록 인터페이스가 있습니다.

모든 컴포넌트에는 WSDL 유형의 인터페이스가 있습니다. Java 컴포넌트만 Java 유형 인터페이스를 지원합니다. 컴포넌트, 가져오기 또는 내보내기에 둘 이상의 인터페이스가 있는 경우 모든 인터페이스는 동일한 유형이어야 합니다.

컴포넌트는 동기식 또는 비동기식으로 호출될 수 있습니다. 이는 구현의 동기식 또는 비동기식 여부에는 관계가 없습니다. 컴포넌트 인터페이스는 동기식 양식으로 정의되며 비동기식 지원도 생성됩니다. 선호하는 상호작용 스타일을 동기식 또는 비동기식으로 지정할 수 있습니다. 비동기 유형은 인터페이스 사용자에게 완료하는 데 많은 시간이 소요될 수 있는 하나 이상의 조작이 포함되어 있음을 알립니다. 결과적으로, 호출하는 서비스는 조작이 완료되고 해당 응답을 전송하기를 기다리는 동안 트랜잭션을 열린 상태로 유지하면 안됩니다. 상호작용 스타일은 인터페이스의 모든 조작에 적용됩니다.

WebSphere Integration Developer에서 응용프로그램을 작성할 때, 각 컴포넌트가 서로 호출하기 위해 사용하는 호출 스타일을 명시적으로 설정하는 것이 최상입니다. 최소한, 사용자는 성능 분석을 수행하거나 오류 처리 전략을 개발하는 동안 응용프로그램 전체적으로 사용되는 호출 스타일을 알려고 합니다. 트랜잭션 경계를 고려하거나 설정할 때 응용프로그램에서의 상호작용을 이해해야 합니다. 사용자는 종종 컴포넌트 사이의 호출 스타일을 설정하거나 판별하는 것이 보여지는 것만큼 쉽지 않다는 것을 알게 됩니다. 이 절에서는 응용프로그램의 특정한 특성을 기초로 런타임 시 사용되는 호출 스타일을 설정하거나 판별하는 방법을 설명합니다.

SCA가 제공하는 호출 스타일은 다음과 같습니다.

- 동기
- 단방향 조작을 사용하는 비동기
- 콜백을 사용하는 비동기
- 지연된 응답이 있는 비동기

호출을 작성하기 위해 Java 구현에서 사용되는 SCA API는 런타임 시 호출 스타일을 판별합니다.

- `invoke()`: 동기
- `invokeAsync()`: 비동기
- `invokeAsyncWithCallback()`: 비동기

일반적으로, 하나의 컴포넌트(소스 또는 클라이언트)에서 다른(대상) 컴포넌트로의 상호작용을 고려할 때 서비스 클라이언트는 사용되는 호출 유형을 판별합니다. 예를 들어, 소스 컴포넌트가 Java™ 컴포넌트인 경우 소스에서 대상으로의 호출 스타일은 구현에서 사용하는 특정 SCA 호출 API(예: invoke(), invokeAsync() 또는 invokeAsyncWithCallback())에 의해 판별됩니다. WebSphere Process Server에서 제공되는 다른 컴포넌트 각각에는 호출의 동기 또는 비동기 여부를 판별하기 위해 사용하는 규칙 세트가 있습니다.

일부 컴포넌트/가져오기는 비동기로 간주됩니다.

- 장기 실행 BPEL
- 휴먼 타스크
- MQ/MQ 가져오기
- JMS/일반 가져오기
- JMS/JMS 가져오기

이 유형의 가져오기나 컴포넌트에 대한 모든 호출은 비동기 호출이어야 합니다. 호출하는 컴포넌트(또는 소스)가 상호작용을 동기식으로 시작하는 경우 SCA는 상호작용을 비동기로 전환합니다.

동기 호출:

서비스 컴포넌트 인터페이스(SCA)는 항상 동기 양식으로 정의됩니다. 동기 인터페이스마다 하나 이상의 비동기 인터페이스가 생성될 수 있습니다.

서비스 컴포넌트가 동기식으로 호출될 때 클라이언트(처리자) 및 서비스 프로바이더 모두 동일한 스레드에서 실행됩니다. WebSphere Process Server 내에서 호출하는 컴포넌트는 프로바이더에서 응답이 수신될 때까지 블록화됩니다.

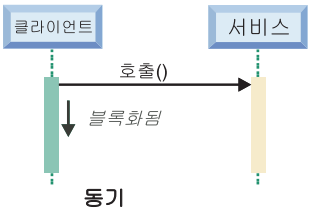


그림 51. 동기 호출

비동기 호출:

WebSphere Process Server는 비동기 응용프로그램 개발을 위한 강력한 프로그래밍 모델을 전달합니다. SCA에서의 비동기 호출을 사용하는 경우, 세 가지 유형의 비동기 상호작용 스타일을 사용할 수 있습니다. 세 가지 유형은 단방향, 지연된 응답 및 콜백이 있는 요청입니다. 세 가지 유형의 비동기 호출 모두를 사용하는 경우, 클라이언트는 `invokeAsync()` 호출 시 즉시 SCA 런타임으로부터 다시 제어를 받습니다.

Java를 사용하여 비동기 프로그램을 개발하도록 공개된 해당 API 및 도구 외에, WebSphere Process Server는 많은 내장 비동기 메시징 바인딩 및 내장 비동기 컴포넌트와 함께 제공됩니다.

클라이언트가 나중에 응답을 캡처할 수 있는 방법은 세 가지입니다. 첫 번째, 클라이언트는 응답을 완전히 버릴 것을 선택하거나 void 메소드에 대한 호출인지 여부를 선택

할 수 있습니다. 이 경우, 비동기 호출을 *단방향*이라고 합니다. 다른 옵션은 클라이언트가 `invokeAsync()`를 호출하고 나중에 클라이언트가 응답을 캡처하기 위해 요청을 작성할 때까지 계속 처리하도록 하는 것입니다. 이러한 시나리오를 *지연되는 응답*이라고 합니다. 마지막으로, 클라이언트에게 비동기식의 *콜백이 있는 요청*을 수행하는 옵션도 제공됩니다. 이 옵션을 수행하려면 클라이언트가 먼저 ServiceCallback 인터페이스를 구현해야 합니다. 그런 다음 `invokeAsync()`를 호출하면 SCA 런타임은 ServiceCallback 핸들러에 콜백을 제공하여 클라이언트에 응답을 제공합니다.

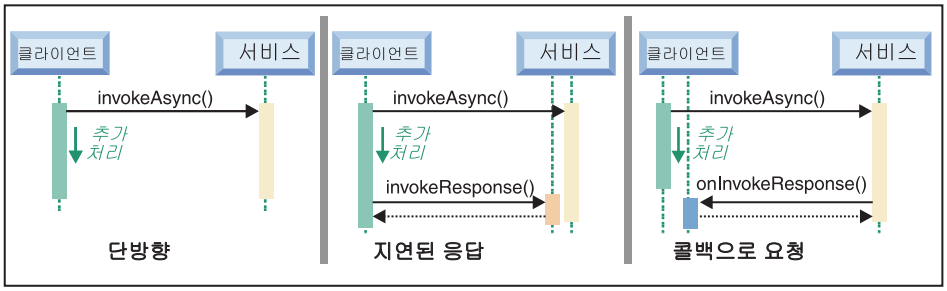


그림 52. 비동기 호출 모델

SCA 인터페이스는 항상 동기 양식으로 정의됩니다. 동기 인터페이스마다 하나 이상의 비동기 인터페이스가 생성될 수 있습니다. 클라이언트가 콜백 메커니즘을 선택한 경우 클라이언트 컴포넌트는 `<interface name>.Callback.java` 클래스를 구현해야 합니다. 이 클래스의 인터페이스는 클라이언트가 사용하려는 실제 컴포넌트의 인터페이스에서 파생됩니다.

SCA 상호작용:

SCA는 모듈의 동기 및 비동기 호출을 지원합니다. 개발자는 SCA 상호작용에 대한 호출 메소드 및 적절한 인터페이스를 선택하는 옵션을 갖습니다.

다이어그램은 다른 인터페이스 유형, 지원되는 호출 메소드 및 모델과, 클라이언트 및 서비스 사이에 데이터가 전달되는 방식을 요약합니다.

동기 호출의 경우 데이터가 동일한 SCA 모듈의 참조에 의해 전달되는 반면 비동기 호출의 경우에는 데이터가 값에 의해 전달됩니다. 테이블은 인터페이스 유형을 기반으로 하는 동적 호출 또는 유형 안정성을 사용할 수 있을 때 요약할 수도 있습니다. 동적 호출 메소드는 항상 WSDL 포트 유형 또는 Java 인터페이스에 대해 사용 가능합니다. 그러나 유형 안정성 호출 메소드를 클라이언트에 사용 가능하도록 하려면 Java 인터페이스 유형이 적절한 클라이언트 참조에서 인터페이스 정의에 대해 사용되어야 합니다.

인터페이스 유형	호출 모델				호출 메소드	
	동기	단방향	지연된 응답	콜백으로 요청	동적	유형 안정성
WSDL 포트 유형	●	■	■	■	예	아니오
Java 인터페이스	●	■	■	■	예	예

● 동일한 SCA 모듈의 참조에 의해 전달된 데이터

■ 값에 의해 전달된 데이터

그림 53. 데이터 전달을 위해 지원되는 메소드와 함께 제공되는 호출 모델의 요약

동적 클라이언트 호출

동적 클라이언트 호출을 사용할 때 동기 및 비동기 상호작용 모두를 지원하기 위해 필요한 많은 키 메소드 및 인터페이스가 있습니다.

표 15. 동적 클라이언트 호출을 위한 키 메소드 및 인터페이스의 요약

인터페이스	메소드	설명
서비스	Object invoke(Service String, Object)	동기 서비스 요청 호출에 사용
	Ticket invokeAsync(String, Object)	**단방향** 또는 **지연된 응답** 비동기 서비스 요청 호출에 사용
	Ticket invokeAsyncWithCallback(String, Object)	**콜백이 있는 요청** 비동기 서비스 요청 호출에 사용. 클라이언트는 ServiceCallback 인터페이스를 구현해야 합니다.
	Object invokeResponse(Ticket, long)	**지연된 응답** 호출의 경우 응답을 얻기 위해 사용
ServiceCallback	void onInvokeResponse(Ticket, Object, Exception)	콜백 인터페이스는 **콜백이 있는 요청** 비동기 서비스 호출을 사용하여 클라이언트가 구현해야 합니다.

동기 호출에 대한 예외 처리:

서비스 컴포넌트가 동기식으로 호출될 때 클라이언트 및 서비스 프로바이더 모두 동일한 스레드에서 실행됩니다. 대상은 클라이언트에 응답 메시지 또는 예외를 리턴하거나 어떤 것도 리턴하지 않을 수 있습니다(단방향 조작에서). 결과가 예외인 경우 비즈니스 예외 또는 시스템 예외일 수 있습니다. 이 경우의 클라이언트는 응용프로그램 코드나 시스템 코드 양식일 수 있습니다.

다음은 JType 인터페이스에 대해 선언된 Java 컴포넌트를 호출하는 샘플 클라이언트입니다. 인터페이스에는 다음과 같이 하나의 메소드가 선언되어 있습니다.

```
public interface StockQuote {
 float getQuote(String symbol) throws InvalidSymbolException;
}
```

클라이언트 코드는 다음과 같습니다.

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (InvalidSymbolException s) {
 System.out.println(This is business

 exception declared in the Java interface.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Unchecked system exception detected);
}
```

시나리오에서, 첫 번째 예외 InvalidSymbolException은 요청이 서비스 프로바이더에 도달했지만 클라이언트 입력을 인식하지 못함을 표시합니다. 서비스 프로바이더는 제공된 기호가 유효하지 않음을 알리는 비즈니스 예외를 처리합니다. 이 비즈니스 예외는 메소드 서명에 의해 선언되는 유일한 예외입니다.

InvalidSymbolException과 유사한 JType 예외는 JType 참조를 사용하는 클라이언트에 대해서만 발생합니다.

선언된 비즈니스 예외 외에도, 클라이언트는 시스템 예외를 수신할 수 있습니다. 예를 들어, 증권 거래소 시스템에서 문제점이 발생한 경우 서비스가 확인되지 않은 예외로 견적가를 얻는 데 실패할 수 있습니다. 서비스에서 이러한 예외가 발생되면 ServiceRuntimeException은 클라이언트에 리턴되고, 클라이언트는 기본적인 원인을 판별하려고 할 수 있습니다. 다음 코드 스니펫은 이러한 정보를 얻을 수 있는 방법을 보여줍니다.

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (ServiceRuntimeException e) {
 Throwable t = e.getCause();
 if (t instanceof RemoteException) {
 system.out.println(System ran into RemoteException.

 Details as follows:  + e.toString());
}
```

비동기 호출에 대한 예외 처리:

서비스 컴포넌트가 비동기식으로 호출될 때 클라이언트 및 서비스 프로바이더는 다른 스레드에서 실행되며 두 스레드 중 하나에서 오류 상태가 발생할 수 있습니다. 클라이언트에서 호출 중에 시스템 예외가 발생할 수 있거나, 서비스 프로바이더에서 요청에 서비스를 제공하는 동안 비즈니스 또는 시스템 예외가 발생할 수 있습니다.

WebSphere Process Server에는 항상 동기 인터페이스의 비동기 상대가 있습니다.

다음은 비동기 인터페이스의 예제입니다.

```
public interface StockQuoteAsync {
public Ticket getQuoteAsync(String arg0);
public Ticket getQuoteAsyncWithCallback(String arg0);
public float getQuoteResponse(Ticket ticket,
long timeout) throws InvalidSymbolException;
}
```

다음은 지연된 응답에 대해 호출 패턴을 사용하는 호출의 클라이언트 코드입니다.

```
Ticket ticket = stockQuote.getQuoteAsync(symbol);
try {
 quote = stockQuote.getQuoteResponse(ticket, Service.WAIT);
 } catch (InvalidSymbolException s) {
 System.out.println(This is business exception declared in the interface.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Unchecked system exception detected);
}
```

동기 호출과 같이, InvalidSymbolException은 요청이 서비스 프로바이더에 도달했지만 시스템이 유효하지 않음을 알리는 비즈니스 예외가 발생했음을 표시합니다. 이 비즈니스 예외는 인터페이스에 의해 선언되는 유일한 예외입니다. InvalidSymbolException과 유사한 JType 예외는 JType 참조를 사용하는 클라이언트에 대해서만 발생합니다.

선언된 비즈니스 예외 외에도, 클라이언트는 메시지를 전송하는 중에 발생하는 연결 오류와 같은 시스템 예외를 수신할 수 있습니다. 클라이언트는 서비스 스레드(비동기 호출의 서비스 측 스레드)에서 발생하는 시스템 예외를 수신할 수 없습니다. SCA 비동기 프로그래밍 모델에 따라, 대상 컴포넌트에 발생하는 런타임 예외는 소스 컴포넌트에 리턴되지 않습니다.

비동기 예외 처리 시 예외 사례

SCA 비동기 프로그래밍 모델 규칙에 대해, 대상 컴포넌트에 발생하는 런타임 예외가 소스 컴포넌트로 리턴되지 않는 하나의 예외가 있습니다. 소스 컴포넌트가 비즈니스 프로세스 컴포넌트나 스태프 프로세스인 경우, 대상 서비스 컴포넌트에서 발생하는 시스템 예외는 호출자에게 리턴됩니다. 이 성능은 BPEL 클라이언트가 시스템 예외를 리턴할 경우 비즈니스 프로세스 디자이너가 시스템 예외를 모델링하고 포착하며 오류 논리를 실행하도록 합니다.

다른 서버에서 서비스를 호출할 경우의 고려사항:

SOA(Service Oriented Architecture)의 이점 중 하나는 처리자가 다른 서비스 모듈에 존재하는 서비스를 사용하도록 하는 기능입니다. 워크로드 밸런싱을 공정하게 하기 위해 셀 내의 다른 서버에 응용프로그램을 설치할 수 있고 그 응용프로그램이 다른 실제 서버에 상주할 수 있습니다.

WebSphere Process Server의 장점 중 하나는 셀 내의 여러 서버에서 응용프로그램 워크로드를 분산시킬 수 있는 기능입니다. 이러한 분산으로 셀 내의 다양한 서버 사이에 워크로드 밸런싱이 한층 나아지도록 하고 컴퓨팅 자원의 유지보수성이 최대화됩니다. 이

는 서버 내에 응용프로그램 또는 서비스 사본이 하나만 있기 때문입니다. 따라서, 서버 A의 응용프로그램은 서비스가 셀 내의 서버 B에 설치되도록 요구할 수 있습니다. 이 방식으로 서비스를 사용하려면 서버 사이에 통신을 구성해야 합니다. 수행하는 구성 유형은 호출하는 서비스 컴포넌트가 서비스를 비동기식 또는 동기식으로 호출하는지 여부에 따라 다릅니다.

관련 주제는 비동기 및 동기 호출 둘 다에 대해 시스템을 구성하는 방법을 설명합니다.

비동기적으로 서비스를 호출하기 위한 서버 구성:

다른 서버의 서비스 컴포넌트가 통신할 수 있도록 하려면 유사하게 서버를 구성해야 합니다. 이 주제는 다른 서버에서 비동기적으로 서비스를 호출하는 응용프로그램에 대해 통신할 수 있도록 수행하는 구성을 설명합니다.

시작하기 전에

타스크는 통신을 구성한 시스템에 WebSphere Process Server를 이미 설치했지만 관련된 응용프로그램은 아직 설치하지 않았다고 가정합니다. 관련된 서버 모두에 대한 구성을 조사하고 변경할 수 있는 관리 콘솔을 사용 중입니다.

이 태스크 정보

다른 시스템에 설치된 서비스 컴포넌트의 서비스를 요청하는 응용프로그램을 설치하기 전에 시스템을 구성해야 시스템이 요청과 통신할 수 있습니다. 비동기 호출을 사용하는 서비스 모듈의 경우 처리는 외부 버스 및 서비스 통합 버스(SIB) 중개와 관련됩니다.

주: 이 타스크를 위해 호출 서비스 모듈은 시스템 A에 상주하고 대상은 시스템 B에 상주합니다.

이 타스크를 위해 그림 54에는 구성에서 사용할 정보가 포함됩니다.

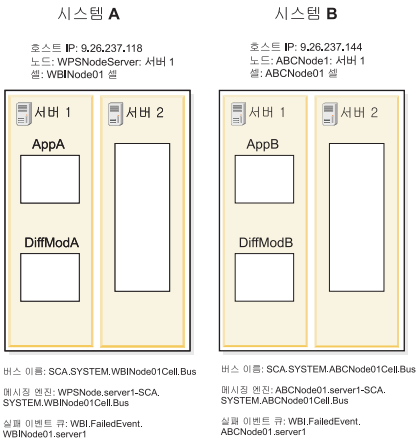


그림 54. 다른 시스템에서 서비스 호출

주: 간단하게 모든 셀의 이 통신에 관련된 서버만 표시되고 각 서버는 다른 물리적 시스템에 상주합니다.

프로시저

1. 통신에 관련된 각 서버에 대한 정보를 수집하십시오. 작성자 및 대상 서버 모두에 대해 다음 정보가 필요합니다.
   - 호스트 IP 주소
   - 셀
   - 노드
   - 서버
   - 버스 이름
   - 메시징 엔진
   - 실패 이벤트 큐 이름
2. 응용프로그램을 설치하십시오.
3. 기타 서버를 가리키는 각 서버에 외부 버스를 작성하고 라우팅 정의 유형을 직접, 서비스 통합 버스 링크로 설정하십시오.

   자세한 정보는 WebSphere Application Server Network Deployment 버전 7 Information Center의 지점간 메시징을 사용하기 위해 서비스 통합 버스 연결 주제를 참조하십시오.

   예를 들어, 시스템 A의 SIB 중개 링크 및 외부 버스의 구성은 다음과 같습니다.

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.ABCNode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   ABCNode01.server1-SCA.SYSTEM.ABCNode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.144:7277:BootstrapBasicMessaging
   ```

   시스템 B의 SIB 중개 링크 및 외부 버스의 구성은 다음과 같습니다.

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.WBINode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   WPSNode.server1-SCA.SYSTEM.WBINode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.118:7276:BootstrapBasicMessaging
   ```

   주의: 부트스트랩의 포트 번호는 SIB 엔드포인트 주소 포트입니다. 보안이 사용 가능할 경우 보안 SIB 엔드포인트 주소 포트를 사용해야 합니다.
4. 서버를 다시 시작해서 SIB 중개 링크를 동기화하십시오.

   다음과 같은 메시지가 표시됩니다.

```
[9/25/09 8:04:23:406 CDT] 00000034 SibMessage I [:] CWSIT0032I:
The service integration bus link TestCrossCell from messaging engine
WPSNode01.server1-SCA.SYSTEM.WPSNode01Cell.Bus
in bus SCA.SYSTEM.WPSNode01Cell.Bus
to messaging engine ABCNode01.server1-SCA.SYSTEM.
ABCNode01Cell.Bus in bus SCA.SYSTEM.
ABCNode01Cell.Bus started.
```

5. 모든 서비스 모듈에 대한 대상을 표시하십시오.
6. 기타 시스템의 대상에 연결되어야 하는 호출 서비스 모듈의 전송 대상 기본 전달 경로를 수정하십시오.

   응용프로그램 > SCA 모듈을 선택하고 모듈을 선택한 후 **SCA 시스템 버스 대상**을 클릭하십시오.

   연결할 대상에는 대상 이름에 importlink가 있습니다. 예를 들어, 시스템 A에서 대상은 sca/AppA/importlink/test/sca/cros/simple/custinfo/CustomerInfo입니다. 대상 이름 앞에 외부 버스 이름을 붙여서 경로를 수정하십시오. 예제에서 두 번째 시스템에 대한 외부 버스 이름은 SCA.SYSTEM.ABCNode01Cell.Bus입니다. 결과는 다음과 같습니다.

   ```
   SCA.SYSTEM.ABCNode01Cell.Bus:sca/AppA/importlink/
   test/sca/cros/simple/custinfo/CustomerInfo
   ```

7. 옵션: 시스템에서 보안을 사용할 수 있는 경우 전송자 역할을 외부 버스에 추가하십시오. 운영 체제 명령 프롬프트에서 각 응용프로그램이 양쪽 시스템 모두에서 사용하는 사용자를 정의해야 합니다. 역할을 추가할 명령은 다음과 같습니다.

   ```
   wsadmin $AdminTask addUserToForeignBusRole -bus busName
     -foreignBus foreignBusName -role roleName -user userName
   ```

   busName
   명령을 입력하는 시스템의 버스 이름입니다.

   foreignBusName
   사용자를 추가하는 외부 버스입니다.

   userName
   외부 버스에 추가할 사용자 ID입니다.

다음에 수행할 작업

응용프로그램을 시작하십시오.

서비스를 동기식으로 호출하도록 서버 구성:

서비스 컴포넌트가 다른 서비스 컴포넌트를 동기식으로 호출하는 경우, 대상을 실행 중인 시스템을 지시하도록 호출하는 서비스 컴포넌트를 구성해야 하므로, 대상 서비스는 호출하는 서비스 컴포넌트에 결과를 전달할 수 있습니다.

시작하기 전에

타스크는 통신을 구성한 시스템에 WebSphere Process Server를 이미 설치했지만 관련된 응용프로그램은 아직 설치하지 않았다고 가정합니다. 관련된 서버 모두에 대한 구성을 조사하고 변경할 수 있는 관리 콘솔을 사용 중입니다.

이 태스크 정보

다른 서비스를 동기식으로 호출하는 서비스 컴포넌트는 단지 대상 시스템의 JNDI(Java Naming and Directory Interface) 이름에서 호출하는 시스템의 JNDI 이름으로의 내보내기를 구성하여 대상과 통신할 수 있습니다.

주: 이 타스크를 위해 호출 서비스 모듈은 시스템 A에 상주하고 대상은 시스템 B에 상주합니다.

이 타스크의 목적에 따라, 그림 55에는 구성에 사용할 정보가 있습니다.

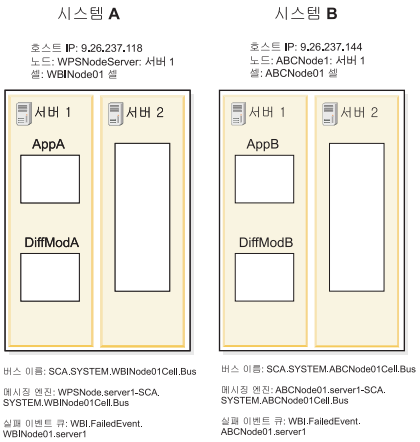


그림 55. 다른 시스템에서 서비스 호출

주: 간단하게 모든 셀의 이 통신에 관련된 서버만 표시되고 각 서버는 다른 물리적 시스템에 상주합니다.

프로시저

1. 각 서버에 응용프로그램을 설치하십시오.
2. 호출하는 시스템(예제에서 시스템 A)에 대상 시스템에 대한 내보내기를 지시하는 새 네임스페이스 바인딩을 작성하십시오.

   네임스페이스 바인딩 패널에서 셀 범위를 선택하고 **적용**을 클릭하십시오. 변경된 범위가 있으면 화면에서 **새로 작성**을 클릭하여 새 바인딩을 작성하십시오.

   마법사에서 다음을 지정하십시오(값은 예제 구성에 적절함).

   a. 바인딩 유형은 CORBA입니다.

b. 기본 특성은 다음과 같습니다.

- 바인딩 ID는 고유한 문자열입니다. 예제의 경우 `sca_import_test_sca_cross_simple_custinfo_CustomerInfo`입니다.
- 네임스페이스의 이름은 대상 시스템에서 호출하는 EJB(Enterprise Java Bean)의 JNDI 이름입니다. 예를 들면 다음과 같습니다.

  ```
  sca/AppB/export/test/sca/cros/simple/custinfo/CustomerInfo
  ```

  이는 대상 시스템의 내보내기 인터페이스에 이름을 지정합니다.
- Corbaname URL은 대상 시스템에서 네이밍 서비스의 IP 주소 및 포트 번호입니다.

  `corbaname:iiop:`*host*`:`*port*`/`
  NameServiceServerRoot`#<`*package.qualified.interface*`>`

  주: WebSphere Process Server의 경우 포트는 `BOOTSTRAP_ADDRESS`입니다.

  예를 들어, 다음과 같습니다.

  ```
  corbaname:iiop:9.26.237.144:2809/NameServiceServerRoot#sca/
    AppB/export/test/sca/cros/simple/custinfo/CustomerInfo
  ```

완료된 경우 **다음**을 클릭하고 **요약** 페이지에서 값을 확인하십시오. 확인 후에는 **완료**를 클릭하십시오.

시스템은 새 바인딩을 표시합니다.

3. **저장**을 클릭하여 변경사항을 저장하십시오.
4. 교차 셀 구성이 이름이 같은 동일 호스트의 서버로 구성되고 사용자가 `NameNotFoundException`과 함께 JNDI 검색 실패를 발견하는 경우 사용자는 시스템 특성을 설정해야 합니다.

   동일한 호스트에서 실행 중인 동일한 이름의 두 서버가 상호 운영에 사용됨 하위 표제에서 응용프로그램 액세스 문제점에 있는 지시사항을 따르십시오.

다음에 수행할 작업

응용프로그램을 시작하십시오. 시스템 A의 서비스 컴포넌트는 이제 시스템 B에서 서비스를 동기식으로 호출할 수 있습니다.

규정자

규정자를 사용하면 개발자가 WebSphere Process Server 런타임에 서비스 품질 요구사항을 위치시킬 수 있기 때문에 규정자는 SCA의 중요한 파트입니다.

SCA에서 사용할 수 있는 규정자의 여러 다른 카테고리가 있습니다. 이러한 규정자 카테고리는 트랜잭션, 활동 세션, 보안 및 비동기 신뢰도입니다.

각 SCA 규정자에는 규정자가 지정할 수 있는 컴포넌트의 SCDL 정의에 특정한 범위가 있습니다. 예를 들어, 일부 규정자가 참조 레벨에서 지정될 수 있는 반면 나머지 규정자는 인터페이스 또는 구현 레벨에서만 유효합니다. 테이블에는 사용 가능한 다양한 규정자 및 각 규정자에 대한 유효한 범위가 나열됩니다. 규정자는 트랜잭션 또는 보안과 같이 규정자가 제공한 서비스 품질 유형에 의해서 정렬됩니다.

표 16. 규정자 요약

유형	규정자	범위	설명
트랜잭션	트랜잭션	구현	**글로벌** – 글로벌 트랜잭션은 컴포넌트를 실행하기 위해 있어야 합니다. **로컬** – 글로벌 트랜잭션은 컴포넌트를 실행하기 위해 존재하지 않아야 합니다. **모두** – 컴포넌트는 트랜잭션 상태에 따라 영향을 받지 않습니다. **로컬 응용프로그램** – 컴포넌트 처리는 응용프로그램이 관리하는 WebSphere 로컬 트랜잭션 포함에서 발생합니다.
	joinTransaction	인터페이스	**true** – 호스팅 컨테이너가 클라이언트 트랜잭션과 결합합니다. **false**(기본값) – 호스팅 컨테이너가 클라이언트 트랜잭션과 결합하지 않습니다.
	suspendTransaction	참조	**true** – 대상 컴포넌트의 동기 호출은 클라이언트 글로벌 트랜잭션에서 실행되지 않습니다. **false** – 대상 컴포넌트의 동기 호출은 클라이언트 글로벌 트랜잭션에서 실행됩니다.
	deliverAsyncAt	참조	**호출** – 대상 서비스의 동기 호출이 즉시 발생합니다. **커미트** – 대상 서비스의 비동기 호출이 글로벌 트랜잭션의 파트로 발생합니다.

표 16. 규정자 요약 (계속)

유형	규정자	범위	설명
비동기 응답	신뢰도	참조	비동기 메시지 전달에 대한 서비스 품질 레벨을 지정합니다. 신뢰도는 다음 값 중 하나가 될 수 있습니다: **bestEffort** 또는 **assured**
	requestExpiration	참조	전달되지 않은 경우 비동기 요청이 버려지는 시간 길이(밀리초)를 지정합니다.
	responseExpiration	참조	요청이 전송된 시간과 응답 또는 콜백을 수신한 시간 사이의 지속 기간(밀리초)을 지정합니다.
보안	securityIdentity	구현	**권한**은 런타임 시 구현이 실행하는 ID에 대한 논리적 이름을 지정합니다.
	securityPermission	인터페이스, 인터페이스, 메소드	호출자 ID에는 인터페이스 또는 메소드를 실행하기 위한 권한을 갖기 위해 이 규정자에서 지정된 역할이 있어야 합니다.

표 16. 규정자 요약 (계속)

유형	규정자	범위	설명
활동 세션	activitySession	구현	**true** – 이 컴포넌트를 실행하기 위해 확립된 ActivitySession이어야 합니다. **false** – 컴포넌트는 활동 세션이 없는 곳에서 실행됩니다.**모두** – 컴포넌트는 ActivitySession의 존재 또는 부재에 영향을 받지 않습니다.
	joinActivitySession	인터페이스	**true** – 호스팅 컨테이너가 클라이언트 ActivitySession과 결합합니다. **false** – 호스팅 컨테이너가 클라이언트 ActivitySession과 결합하지 않습니다.
	suspendActivitySession	참조	**true** – 대상 컴포넌트의 메소드는 클라이언트 ActivitySession의 파트로 실행되지 않습니다. **false** – 대상 컴포넌트의 메소드는 클라이언트 ActivitySession의 파트로 실행됩니다.

트랜잭션 규정자

트랜잭션 규정자를 사용하면 개발자가 SCA 모듈의 서비스 컴포넌트에 대한 특정 트랜잭션 환경을 요청할 수 있습니다. 다음은 이러한 규정자의 요약입니다.

트랜잭션

트랜잭션 규정자는 n 서비스 컴포넌트의 구현 범위에서 설정됩니다. 이 규정자는 '글로벌', '로컬'(기본값) 또는 '모두'로 설정될 수 있습니다. 글로벌로 설정된 경우 컴포넌트는 글로벌 트랜잭션의 컨텍스트에서 실행됩니다. 글로벌 트랜잭션이 호출에 있는 경우 컴포넌트가 이 글로벌 트랜잭션 범위에 추가됩니다. 로컬로 설정된 경우 컴포넌트가 로컬 트랜잭션의 컨텍스트에서 실행됩니다. 마지막으로 값이 모두로 설정된 후 글로벌 트랜잭션이 있는 경우 컴포넌트가 현재 글로벌 트랜잭션 범위를 결합합니다. 그러나 글로벌 트랜잭션이 없는 경우 컴포넌트가 로컬 트랜잭션의 컨텍스트에서 실행됩니다.

joinTransaction

joinTransaction 규정자는 서비스 컴포넌트의 인터페이스 범위에서 설정됩니다. 이 규정자는 true 또는 false(기본값이 되는 false)로 설정될 수 있습니다. true로 설정된 경우 인터페이스 경계에서 글로벌 트랜잭션(있는 경우)을 일시중단하지 않도록 런타임을 지시합니다. false로 설정된 경우 인터페이스 경계에서 글로벌 트랜잭션(있는 경우)을 일시중단하도록 런타임을 지시합니다. 인터페이스에 joinTransaction 트랜잭션 규정자를 표시하면 어셈블된 응용프로그램이 필요에 따라 수행하는지 확인하기 위해 어셈블러 또는 개발자가 사용할 수 있는 메타데이터를 제공합니다. 대상 컴포넌트와 전파된 트랜잭션의 연합 여부를 추론하는 것은 동적 런타임 이외에 어셈블러와 개발자에게 달려있습니다.

suspendTransaction

suspendTransaction 규정자는 서비스 컴포넌트의 참조 레벨에 설정되고 참조와 연관된 대상 서비스를 호출하기 전에 글로벌 트랜잭션을 일시중단해야 하는지 여부를 식별합니다. 이 규정자는 true 또는 false(기본값)로 설정될 수 있습니다.

deliveryAsyncAt

suspendTransaction에 흔히 있는 일이지만 동기 유형보다 비동기 상호작용에 적용되는 것을 제외하면 deliveryAsyncAt 규정자는 suspendTransaction 규정자와 유사합니다. deliverAsyncAt 규정자에는 호출(기본값) 또는 커미트의 값이 있을 수 있습니다. 호출로 설정된 경우 이는 호출이 작성될 때 비동기 상호작용에 대한 메시지가 즉시 큐에 커미트되어야 한다는 것을 런타임에 표시합니다. 커미트 값은 현재 작업 단위와 연관된 트랜잭션의 파트로 메시지가 큐에 커미트되어야 한다는 것을 표시합니다.

비동기 응답 규정자

비동기 응답에 대한 서비스 품질을 표시하기 위해 사용할 수 있는 세 개의 규정자가 있습니다. 모든 비동기 응답 규정자는 참조 범위에 지정됩니다. 다음은 비동기 응답 규정자의 요약입니다.

신뢰도 신뢰도 규정자는 비동기 메시지 전달에 대한 서비스 품질 레벨을 지정하는 데 사용됩니다. 신뢰도는 최상의 노력 또는 보증(기본값)으로 설정될 수 있습니다.

requestExpiration

requestExpiration 규정자는 아직 전달되지 않은 경우 런타임이 비동기 요청에서 보류해야 하는 시간 길이를 지정하는 데 사용됩니다. 이 규정자에 대해 표시된 시간(밀리초) 이후에 이 요청이 버려집니다.

responseExpiration

런타임이 비동기 응답을 유지해야 하거나 콜백을 제공해야 하는 시간의 길이를 지정하는 데 responseExpiration 규정자가 사용됩니다. 이 규정자에 대한 값은 밀리초로 제공됩니다.

보안 규정자

보안과 연관된 서비스 품질을 표시하기 위해 사용할 수 있는 두 개의 규정자가 있습니다. 다음은 이러한 규정자의 요약입니다.

securityIdentity

서비스 컴포넌트에 대한 구현이 런타임에서 실행되어야 하는 보안 ID를 지정하는 데 securityIdentity 규정자가 사용됩니다. 이 규정자는 서비스 컴포넌트에 대한 구현 범위에 놓여야 하며 제공된 값은 컴포넌트가 실행될 ID에 대한 논리 이름과 일치해야 합니다.

securityPermission

securityPermission 규정자는 인터페이스를 포함하는 인터페이스 레벨 또는 메소드 레벨에 지정됩니다. 이 규정자에 대한 값은 이 서비스의 호출자에 서비스를 호출하기 위해 지정된 역할이 있어야 한다는 것을 표시합니다.

securityPermission 및 securityIndentity 모두의 경우 이러한 규정자에 대한 기본 구현은 기존 Java EE 개념을 기반으로 합니다.

활동 세션 규정자

활동 세션 규정자 세트는 이전에 소개된 트랜잭션 규정자와 유사합니다. ActivitySession 서비스는 글로벌 트랜잭션과 비교될 때 대체 작업 단위를 제공할 수 있는 WebSphere 프로그래밍 모델 확장자입니다. 실제로 활동 세션 컨텍스트는 글로벌 트랜잭션보다 더 오래 지속될 수 있고 글로벌 트랜잭션도 포함할 수 있습니다. 다음은 활동 세션 규정자의 요약입니다.

joinActivitySession

joinActivitySession 규정자는 인터페이스 레벨에서 설정되고 컴포넌트가 클라이언트 호출자의 활동 세션에 결합하는지 여부를 표시합니다. 이 규정자에 대한 두 개의 값(true 및 false(기본값))이 있습니다. true로 설정된 경우 컴포넌트가 호출될 때 컴포넌트가 있으면 런타임이 활동 세션을 일시중단하지 말아야 함을 표시합니다. false로 설정된 경우 컴포넌트를 호출하기 전에 활동 세션이 일시중단되어야 함을 표시합니다.

activitySession

activitySession 규정자는 구현 레벨에 지정되고 연관된 서비스 컴포넌트를 실행하기 위한 활동 세션의 존재 여부를 표시하는 데 사용됩니다. 이 규정자는 'true', 'false' 또는 '모두'(기본값)로 설정될 수 있습니다. true로 설정된 경우 컴포넌트가 활동 세션의 파트로 실행됨을 표시합니다. false로 설정된 경우 컴포넌트는 활동 세션의 파트로 실행되지 말아야 합니다. 이는 컴포넌트에 지정된 인터페이스에 대해 joinActivitySession이 false로 설정되어야 함을 의미합니다. 마지막으로 이 규정자가 모두로 설정된 경우, 컴포넌트가 활동 세션의 파트로 실행됩니다(존재하는 경우). 그렇지 않으면 실행되지 않습니다.

suspendActivitySession

suspendActivitySession 규정자는 참조 레벨에 설정되고 참조와 연관된 대상 서비스가 호출 활동 세션의 파트 또는 그렇지 않은 경우로 호출되는지 여부를 표시하는 데 사용됩니다. true로 설정된 경우 활동 세션은 일시중단되고 대상 컴포넌트의 메소드는 클라이언트 활동 세션의 파트로 실행되지 않습니다. false(기본값)로 설정된 경우 활동 세션은 일시중단되지 않고 대상 컴포넌트의 메소드는 클라이언트 ActivitySession의 파트로 실행됩니다.

SCA 프로그래밍 기술

이 절은 Service Component Architecture 프로그래밍 기술의 예제를 제공합니다.

Java에서 사용하는 런타임 규칙을 서비스 데이터 오브젝트로 변환

생성된 코드를 올바로 대체하거나 Java에서 서비스 데이터 오브젝트(SDO)로의 변환과 관련된 가능한 런타임 예외를 판별하려면 먼저 관련 규칙을 이해하는 것이 중요합니다. 대부분의 변환은 단순하지만 생성된 코드를 변환하는 경우 런타임에서 최상의 가능성을 제공하는 일부 복잡한 경우도 있습니다.

기본 유형 및 클래스

런타임에서는 서비스 데이터 오브젝트와 기본 Java 유형 및 클래스 간 직접 변환을 수행합니다. 다음은 기본 유형 및 클래스입니다.

- Char 또는 java.lang.Character
- Boolean
- Java.lang.Boolean
- Byte 또는 java.lang.Byte
- Short 또는 java.lang.Short
- Int 또는 java.lang.Integer
- Long 또는 java.lang.Long
- Float 또는 java.lang.Float
- Double 또는 java.lang.Double
- Java.lang.String
- Java.math.BigInteger
- Java.math.BigDecimal
- Java.util.Calendar
- Java.util.Date
- Java.xml.namespace.QName
- Java.net.URI
- Byte[]

사용자 정의 Java 클래스 및 배열

Java 클래스 또는 배열을 SDO로 변환하는 경우 런타임에서 Java 유 형의 패키지 이름을 반전시켜 생성된 URI를 포함하고 Java 클래스 이름과 동일한 유형의 데이터 오브젝트 가 작성됩니다. 예를 들어, Java 클래스 com.ibm.xsd.Customer는 Customer 유형의 URI http://xsd.ibm.com을 포함하는 SDO로 변환됩니다. 그러면 런타임에서 Java 클래스 구 성원의 컨텐츠를 검사하고 SDO의 특성에 값을 지정합니다.

SDO에서 Java 유형으로 변환하는 경우 런타임에서는 URI를 반전시켜 패키지 이름을 생성하고 유형 이름을 SDO 유형과 일치시킵니다. 예를 들어, 유형이 Customer이고 URI가 http://xsd.ibm.com인 데이터 오브젝트는 Java 패키지 com.ibm.xsd.Customer의 인스턴스를 생성합니다. 그러면 런타임에서 SDO에 있는 특성의 값을 추출하여 해당 특성을 Java 클래스 인스턴스의 필드에 지정합니다.

Java 클래스가 사용자 정의 인터페이스인 경우 런타임에서 인스턴스화할 수 있는 구체적(concrete) 클래스를 제공하고 생성된 코드를 대체해야 합니다. 런타임에서 구체적(concrete) 클래스를 작성할 수 없는 경우 예외가 발생합니다.

Java.lang.Object

Java 유형이 java.lang.Object인 경우 생성된 유형은 xsd:anyType입니다. 모듈은 모든 SDO에서 이 인터페이스를 호출할 수 있습니다. 런타임에서 해당 클래스를 찾을 수 있다면 사용자 정의 Java 클래스 및 배열과 같은 방식으로 런타임에서 구 체적(concrete) 클래스를 인스턴스화합니다. 그렇지 않으면 SDO를 Java 인터페이스로 전달합니다.

메소드가 java.lang.Object 유형을 리턴하는 경우 메소드가 구체적(concrete) 유형을 리턴할 때만 런타임에서 SDO로 변환됩니다. 런타임에서는 사용자 정의 Java 클래스 및 배열을 SDO로 변환하는 방법(다음 단락에서 설명함)과 비슷한 변환을 사용합니다.

Java 클래스 또는 배열을 SDO로 변환하는 경우 런타임에서 Java 유 형의 패키지 이름을 반전시켜 생성된 URI를 포함하고 Java 클래스 이름과 동일한 유형의 데이터 오브젝트 가 작성됩니다. 예를 들어, Java 클래스 com.ibm.xsd.Customer는 Customer 유형의 URI http://xsd.ibm.com을 포함하는 SDO로 변환됩니다. 그러면 런타임에서 Java 클래스 구 성원의 컨텐츠를 검사하고 SDO의 특성에 값을 지정합니다.

두 경우 모두 런타임에서 변환을 완료 할 수 없으면 예외가 발생합니다.

Java.util 컨테이너 클래스

Vector, HashMap, HashSet 등과 같이 구체적(concrete) Java 컨테이너 클래스로 변환하는 경우 런타임에서는 적절한 컨테이너 클래스를 인스턴스화합니다. 런타임에서는 사용자 정의 Java 클래스 및 배열에서 컨테이너 클래스를 채우는 경우 사용하는 방법과 유사한 메소드를 사용합니다. 런타임에서 구체적(concrete) Java 클래스를 찾지 못하면 컨테이너 클래스를 SDO로 채웁니다.

구체적 Java 컨테이너 클래스를 SDO로 변환할 때에는 런타임에 『Java를 XML로 변환』에 표시된 생성된 스키 마를 사용합니다.

Java.util 인터페이스

java.util 패키지의 특정 컨테이너 인터페이스의 경우 런타임에서는 다음과 같은 구체적(concrete) 클래스를 인스턴스화합니다.

표 17. WSDL 유형을 Java 클래스로 변환

인터페이스	기본 구체적(concrete) 클래스
콜렉션	HashSet
맵	HashMap
목록	ArrayList
세트	HashSet

서비스 데이터 오브젝트와 Java 간 변환 대체

서비스 데이터 오브젝트(SDO)와 Java 유형 오브젝트 간에 시스템이 작성하는 변환이 사용자 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 수행하면 기본 구현을 사용자만의 구현으로 바꿀 수 있습니다.

시작하기 전에

WebSphere Integration Developer 또는 genMapper 명령을 사용하여 WSDL이 Java 유형으로 변환되어 생성되었는지 확인하십시오.

이 태스크 정보

생성된 코드를 사용자 요구에 맞는 코드로 바꾸어 WSDL 유형을 Java 유형으로 맵핑하는 생성된 컴포넌트를 대체합니다. 사용자가 고유한 Java 클래스를 정의한 경우 고유한 맵을 사용하는 방법을 고려하십시오. 이 프로시저를 수행하여 변경할 수 있습니다.

프로시저

1. 생성된 컴포넌트를 찾으십시오. 컴포넌트 이름은 *java_class*Mapper.component입니다.
2. 문서 편집기를 사용하여 컴포넌트를 편집하십시오.
3. 생성된 코드를 주석 처리하고 사용자만의 메소드를 제공하십시오.

   컴포넌트 구현이 포함된 파일 이름은 변경하지 마십시오.

예

다음은 변경할 생성된 컴포넌트에 대한 예제입니다.

```
 private Object datatojava_get_customerAcct(DataObject myCustomerID,
    String integer)
{

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

다음에 수행할 작업

포함하는 모듈이 있는 디렉토리에 컴포넌트 및 기타 파일을 복사한 후 WebSphere Integration Developer의 컴포넌트에 연결하거나 serviceDeploy 명령을 사용하여 EAR(Enterprise Archive) 파일을 생성하십시오.

생성된 Service Component Architecture 구현 대체

Java 코드와 서비스 데이터 오브젝트(SDO) 간에 시스템이 작성하는 변환이 사용자의 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 수행하면 기본 SCA(Service Component Architecture) 구현을 사용자의 고유한 구현으로 바꿀 수 있습니다.

시작하기 전에

WebSphere Integration Developer 또는 genMapper 명령을 사용하여 Java 유형이 WSDL(Web Services Definition Language)로 변환되어 생성되었는지 확인하십시오.

이 태스크 정보

생성된 코드를 사용자 요구에 맞는 코드로 바꾸어 Java 유형을 WSDL 유형으로 맵핑하는 생성된 컴포넌트를 대체합니다. 사용자가 고유한 Java 클래스를 정의한 경우 고유한 맵을 사용하는 방법을 고려하십시오. 이 프로시저를 수행하여 변경할 수 있습니다.

프로시저

1. 생성된 컴포넌트를 찾으십시오. 컴포넌트 이름은 *java_class*Mapper.component입니다.
2. 문서 편집기를 사용하여 컴포넌트를 편집하십시오.
3. 생성된 코드를 주석 처리하고 사용자만의 메소드를 제공하십시오.

   컴포넌트 구현이 포함된 파일 이름은 변경하지 마십시오.

예

다음은 변경할 생성된 컴포넌트에 대한 예제입니다.

```
private DataObject javatodata_setAccount_output(Object myAccount) {

  // You can override this code for custom mapping.
  // Comment out this code and write custom code.

  // You can also change the Java type that is passed to the
  // converter, which the converter tries to create.


 return SDOJavaObjectMediator.java2Data(myAccount);


}
```

다음에 수행할 작업

포함하는 모듈이 있는 디렉토리에 컴포넌트 및 기타 파일을 복사한 후 WebSphere Integration Developer의 컴포넌트에 연결하거나 serviceDeploy 명령을 사용하여 EAR(Enterprise Archive) 파일을 생성하십시오.

비SCA 내보내기 바인딩에서 프로토콜 헤더 전파

컨텍스트 서비스는 SCA(Service Component Architecture) 호출 경로를 따라 컨텍스트(JMS 헤더와 같은 프로토콜 헤더, 계정 ID와 같은 사용자 컨텍스트 포함)를 전파합니다. 컨텍스트 서비스는 API 및 구성 가능한 설정 세트를 제공합니다.

컨텍스트 서비스 전파가 양방향이면 응답 컨텍스트는 항상 현재 컨텍스트를 겹쳐씁니다. 하나의 SCA 컴포넌트에서 다른 컴포넌트로 호출 실행 시 응답에 다른 컨텍스트가 포함됩니다. 서비스 컴포넌트에 수신 컨텍스트가 있지만 다른 서비스 호출 시 또 다른 서비스가 원래의 전송 컨텍스트를 겹쳐씁니다. 응답 컨텍스트는 새 컨텍스트가 됩니다.

컨텍스트 서비스 전파가 단방향이면 원래의 컨텍스트는 동일하게 남습니다.

컨텍스트 서비스의 라이프사이클은 호출과 연관됩니다. 요청은 연관된 컨텍스트와 해당 특정 요청의 처리에 컨텍스트가 속한 라이프사이클을 갖습니다. 해당 요청 처리가 완료되면, 해당 컨텍스트의 라이프사이클이 끝납니다.

단기 실행 BPEL(Business Process Execution Language) 프로세스의 경우, 응답 컨텍스트가 요청 컨텍스트를 겹쳐씁니다. 첫 번째 요청에서 다시 응답 컨텍스트를 얻고 다음 요청으로 푸시합니다. 장기 실행 중 BPEL 프로세스의 경우, BPEL 프레임워크가 응답 컨텍스트를 버립니다. 원본 컨텍스트를 저장하고 다른 송신 호출 시 해당 컨텍스트를 사용합니다.

컨텍스트 서비스에는 바인딩 동작을 지시하는 테이블과 구성 가능한 규칙이 있습니다. 자세한 정보는 참조 절에서 사용 가능한 생성 API 및 SPI 문서를 참조하십시오. WebSphere® Integration Developer를 개발하는 동안, 가져오기-내보내기 특성에 컨텍스트 서비스를 설정할 수 있습니다. 자세한 정보는 WebSphere Integration Developer Information Center에서 가져오기 및 내보내기 바인딩 정보를 참조하십시오.

비즈니스 오브젝트 프로그래밍

비즈니스 오브젝트는 고객 또는 송장과 같은 응용프로그램 데이터에 대한 컨테이너입니다. 데이터는 비즈니스 오브젝트 방식으로 컴포넌트 사이에 교환됩니다. 비즈니스 오브젝트의 기본적인 구조는 XSD(XML Schema Definition)이며, 비즈니스 오브젝트에 대한 프로그램식 액세스는 WebSphere의 비즈니스 오브젝트 인터페이스를 통해 제공됩니다. 전체적으로, 이러한 비즈니스 오브젝트 측면, 구조적 표시, 프로그램식 인터페이스, 그리고 SCA(Service Component Architecture) 내에서의 동작 및 조작은 사용자 솔루션에서 비즈니스 데이터를 설명하고 전달하기 위한 일관성 있는 강력한 수단을 제공하는 비즈니스 오브젝트 프레임워크입니다.

이 안내서는 일부 기능에 대한 스키마 구성 처리에서 문제점 영역에 대한 설명을 비롯하여 프로그래밍 비즈니스 오브젝트에 대한 정보를 제공합니다. 비즈니스 오브젝트를 정의하는 방법, 비즈니스 오브젝트 개발 지침 및 비즈니스 오브젝트 프로그래밍 API 사용 방법에 대한 정보는 "관련 정보" 절에 있는 기사를 참조하십시오.

관련 정보

Web Services Description Language(WSDL) 1.1

서비스 데이터 오브젝트 소개

WebSphere Process Server의 비즈니스 오브젝트 점검

프로그래밍 모델

비즈니스 오브젝트 프로그래밍 모델 절에서는 IBM 비즈니스 오브젝트 프레임워크 내에서 데이터의 기본 유형을 캡슐화하는 방법을 설명합니다. 비즈니스 오브젝트의 작성 및 처리를 용이하게 하기 위해 Java 서비스 세트를 제공해서 비즈니스 오브젝트 프레임워크가 서비스 데이터 오브젝트 스펙을 확장합니다.

IBM 비즈니스 오브젝트 프레임워크에 대한 작업

비즈니스 오브젝트 프레임워크는 런타임의 데이터가 응용프로그램에 의해 모델링되고 런타임으로 통합되며 메모리에 표시되는 방법에 대해 설명합니다.

표 1은 데이터의 기본 유형이 비즈니스 오브젝트 프레임워크에서 구현되는 방법을 요약합니다.

표 18. 데이터 추상 및 해당하는 구현

데이터 추상	구현	설명
인스턴스 데이터	비즈니스 오브젝트	비즈니스 오브젝트는 비즈니스 엔티티를 표시하거나 리터럴 메시지 정의를 문서화하거나 스칼라 특성이 있는 단순한 기본 오브젝트에서 오브젝트의 크고 복잡한 계층 구조 또는 그래프에 이르기까지 모든 것을 사용 가능하게 하는 1차 메커니즘입니다. 비즈니스 오브젝트는 SDO DataObject 개념에 대한 직접적인 결과입니다.
인스턴스 메타데이터	비즈니스 그래프	비즈니스 그래프는 비즈니스 그래프에서 비즈니스 오브젝트와 관련된 변경 요약 및 이벤트 요약 정보 전달과 같은 추가 기능을 제공하기 위해 단순 비즈니스 오브젝트 또는 비즈니스 오브젝트의 계층 구조 주변에 추가된 랩퍼입니다. 비즈니스 그래프는 비즈니스 그래프가 단일 변경 요약 헤더 이상을 제공하는 경우를 제외하고 SDO DataGraph 개념에 대해 직접적인 결과입니다.
유형 메타데이터	엔터프라이즈 메타데이터 비즈니스 오브젝트 유형 메타데이터	비즈니스 오브젝트 유형 메타데이터는 해당 값을 런타임에서 향상시키기 위해 비즈니스 오브젝트 정의에 추가할 수 있는 메타데이터입니다. 이러한 메타데이터 항목은 잘 알려진 `xs:annotation` 및 `xs:appinfo` 요소로 비즈니스 오브젝트의 XSD(XML Schema Definition)에 추가됩니다.
서비스	비즈니스 오브젝트 서비스(API)	비즈니스 오브젝트 서비스는 서비스 데이터 오브젝트에서 제공되는 기본 성능 중 최우선으로 제공되는 성능 세트입니다. 작성, 복사, 동등화 및 직렬화와 같은 서비스가 이에 해당합니다. 이러한 API는 `com.ibm.websphere.bo` 패키지에서 찾을 수 있습니다.

WebSphere Process Server 비즈니스 오브젝트 프레임워크는 SDO 표준의 확장자입니다. 따라서 WebSphere Process Server 컴포넌트 간에 교환되는 비즈니스 오브젝트는 `commonj.sdo.DataObject` 클래스의 인스턴스입니다. 그러나 WebSphere Process Server 비즈니스 오브젝트 프레임워크는 기본 DataObject 기능성을 간소화하고 강화하는 여러 서비스 및 기능을 추가합니다.

비즈니스 오브젝트의 작성 및 처리를 용이하게 하기 위해 Java 서비스 세트를 제공해서 WebSphere 비즈니스 오브젝트 프레임워크가 SDO 스펙을 확장합니다. 이 서비스는 com.ibm.websphere.bo라는 패키지의 파트입니다.

- **BOFactory**: 비즈니스 오브젝트의 인스턴스를 작성하는 다양한 방법을 제공하는 키 서비스.
- **BOXMLSerializer**: XML 형식으로 비즈니스 오브젝트의 컨텐츠를 스트림에 작성하거나 스트림에서 비즈니스 오브젝트를 "확장(inflate)"하는 방법을 제공합니다.
- **BOCopy**: 비즈니스 오브젝트의 사본을 작성하는 메소드를 제공합니다("deep" 및 "shallow" 시맨틱).
- **BODataObject**: 변경 요약, 비즈니스 그래프 및 이벤트 요약과 같이, 비즈니스 오브젝트의 데이터 오브젝트 측면에 대한 액세스를 제공합니다.
- **BOXMLDocument**: XML 문서와 같이 비즈니스 오브젝트를 처리하게 하는 서비스에 대한 프론트 엔드.
- **BOChangeSummary** 및 **BOEventSummary**: 비즈니스 오브젝트의 이벤트 요약 부분과 변경 요약의 처리와 액세스를 단순화합니다.
- **BOEquality**: 두 개의 비즈니스 오브젝트에 동일한 정보가 들어 있는지 여부를 판별할 수 있게 하는 서비스. deep 및 샐로우(shallow) 등식 둘 다를 지원합니다.
- **BOType** 및 **BOTypeMetaData**: 이러한 서비스는 commonj.sdo 유형의 인스턴스를 구체화하며 연관된 메타데이터를 처리하게 합니다. 그런 다음, 비즈니스 오브젝트 "by type"을 작성하기 위해 유형 인스턴스를 사용할 수 있습니다.
- **BOInstanceValidator** : 비즈니스 오브젝트의 데이터가 XSD에 따르는지 여부를 알기 위해 유효성 검증합니다.

비즈니스 오브젝트 모델링

WebSphere Process Server 런타임을 통해 흐르는 비즈니스 오브젝트 데이터는 XML 스키마를 사용하여 모델링됩니다. XML 스키마는 문서 유형 정의(DTD)의 대안으로, 데이터 유형 지정, 상속 및 프리젠테이션 영역에서 기능을 확장하기 위해 사용할 수 있습니다. XML 스키마는 업계 표준이고 광범위하게 채택되며 플랫폼 및 언어 중립적인 데이터 유형 모델링 양식을 제공합니다.

대상 네임스페이스 정의:

XML이 해결할 수 있는 대부분의 비즈니스 및 통신 문제점에는 몇몇 XML 어휘의 조합이 필요합니다. XML에는 다른 네임스페이스(예: 다른 업계에 적용되는 네임스페이스)에 할당되도록 이름을 규정하기 위한 메커니즘이 있습니다. XML에서, URI(uniform resource identifier)는 XML 스키마의 요소, 속성 및 유형 정의와 연관될 고유 이름을 제공합니다.

비즈니스 오브젝트 대상 네임스페이스에 대한 두 가지 요구사항이 있습니다.

- 비즈니스 프로세스 관리 런타임 및 도구에서는 대상 네임스페이스가 http://www.foo.com/xyz 대 urn:foo:com:xyz와 유사해야 합니다.
- 비즈니스 오브젝트 프레임워크는 비즈니스 오브젝트에 대한 대상 네임스페이스를 요구합니다.

비즈니스 오브젝트 정의:

WebSphere Process Server는 비즈니스 오브젝트를 정의하거나 가져오기 위한 융통성 있는 메커니즘을 제공합니다.

기본적으로 WebSphere Process Server가 비즈니스 오브젝트 정의로 인식하는 세 가지의 다른 XML 스키마 양식이 있습니다.

- 최상위 레벨 복합 유형 정의
- 최상위 레벨 익명 복합 유형 정의
- 이름 지정된 복합 유형을 참조하는 최상위 레벨 요소

최상위 레벨 복합 유형 정의

다음은 최상위 레벨 복합 유형 정의 예제입니다.

```
<complexType name="ProductType">
 <sequence>
  <element name="name" type="string"/>
  <element name="color" type="string" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

모두 복합 유형 정의만 사용하여 정의된 비즈니스 오브젝트를 가져오거나 정의하는 것은 가장 융통성 있고 관리 가능한 설계입니다. 이 모델의 업사이드는 재사용할 수 있도록 하는 유형 라이브러리입니다. 재사용은 세 개의 다른 메소드에 의해 가능합니다.

- 먼저, 복합 유형 파생 모델(확장 또는 제한사항의)을 사용하여 새 유형을 작성할 수 있습니다.
- 두 번째로, 기존의 복합 유형과 사용 가능한 단순 유형을 기본으로 사용하여 새 총계 유형을 작성할 수 있습니다.
- 세 번째로, 총계 복합 유형과 같이 새 복합 문서 정의를 작성할 수 있습니다.

복합 유형으로 정의된 비즈니스 오브젝트의 다른 내재된 사항은, WS-I 준수를 유지보수하기 위해 런타임 내에서 데이터 전송을 위해 JService 컴포넌트 종류에서 복합 유형이 사용될 경우 이름 지정된 복합 유형을 참조하는 요소를 작성해야 합니다.

최상위 레벨 익명 복합 유형 정의

다음은 최상위 레벨 익명 복합 유형 정의 예제입니다.

```
<element name="Product">
 <complexType>
  <sequence>
   <element name="name" type="string"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

가져온 비즈니스 오브젝트가 모두 익명 요소 정의인 경우 JService 호출에 포함될 준비가 된 것입니다. 그러나 이러한 오브젝트는 원래 재사용할 수 없습니다.

이름 지정된 유형을 참조하는 최상위 레벨 요소

다음은 이름 지정된 유형을 참조하는 최상위 레벨 요소의 예제입니다.

```
<element name="product" type="prod:ProdType"/>
```

이름 지정된 복합 유형을 참조하는 비즈니스 오브젝트는 요소 정의가 필요한 WSDL 조작을 이미 정의한 환경에 종종 있을 수 있습니다. 이러한 시나리오에서, 복합 유형 및 요소 정의의 가능한 처리 조치를 고려하는 것은 중요합니다.

- 요소는 동일한 XML 스키마 파일에서 복합 유형 정의와 같이 상주할 수 있습니다.
- 요소는 WSDL 파일에서 임베드된 복합 유형 정의와 같이 상주할 수 있습니다.
- 요소는 XML 스키마 A.xsd에서 정의될 수 있고, 해당 복합 유형 정의는 XML 스키마 파일 B.xsd에 정의됩니다.
- 요소는 WSDL 파일에서 임베드되어, XML 스키마 파일에 정의된 복합 유형 정의를 참조할 수 있습니다.

예제

예제는 함께 결합된 비즈니스 오브젝트를 정의하기 위한 모든 메커니즘을 보여줍니다.

```
<schema
 targetNamespace="http://www.app.com/Address"
 xmlns:addr="http://www.app.com/Address"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <complexType name="Address">
  <sequence>
   <element name="street1" type="string"/>
   <element name="street2" type="string"/>
   <element name="city" type="string"/>
   <element name="state" type="string"/>
   <element name="zip" type="string"/>
  </sequence>
 </complexType>

 <element name="homeAddress" type="addr:Address"/>
 <element name="workAddress" type="addr:Address"/>
 <element name="otherAddress" type="addr:Address"/>
```

```
 <element name="individualContact">
  <complexType>
   <sequence>
    <element name="firstName" type="string"/>
    <element name="lastName" type="string"/>
    <element ref="addr:HomeAddress"/>
    <element ref="addr:WorkAddress"/>
    <element ref="addr:OtherAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="businessContact">
  <complexType>
   <sequence>
    <element name="name" type="string"/>
    <element ref="addr:WorkAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="chairmanOfTheBoard">
  <complexType>
   <sequence>
    <element name="startDate" type="date"/>
    <element ref="addr:IndividualContact"/>
    <element ref="addr:BusinessContact"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

다음 지침은 비즈니스 오브젝트를 정의하기 위해 선호하는 방법입니다.

- 요소는 이름 지정된 유형을 사용하여 정의됩니다. 익명 유형은 금지됩니다.
- 요소와 복합 유형 정의는 동일한 XML 스키마 또는 WSDL 파일에 있지 않습니다. 이 사례에서 유형 재사용은 금지됩니다.
- 복합 유형은 WSDL 정의가 아닌 XML 스키마 파일에서 정의되어 개념과 같은 유형 라이브러리를 작성합니다. 다시 한 번 이 유형의 정의에서 복합 유형의 재사용이 가능하며 권장합니다.
- 요소 정의는 필요에 따라 단일 복합 유형 정의를 참조하기 위해 빌드됩니다. 예를 들어, WSDL 내부의 요소 정의는 권장되는 패턴입니다.
- 요소 정의는 일반적으로 해당되는 복합 유형 정의와 동일한 대상 네임스페이스를 사용합니다.

비즈니스 오브젝트 특성 정의:

XML 스키마는 비즈니스 오브젝트를 빌드하기 위해 사용되는 복합 유형, 단순 유형 및 속성을 제공합니다.

복합 유형 정의, 익명 복합 유형 정의, 그리고 복합 유형 정의를 참조하는 요소는 외부 비즈니스 오브젝트를 정의하는 데 사용됩니다. term 특성은 비즈니스 오브젝트 내에서 데이터를 정의하는 데 사용됩니다. term은 서비스 데이터 오브젝트 term 특성에서 파생되며 `commonj.sdo.Property` 인터페이스에 의해 정의됩니다. 속성 개념과 같습니다.

특성은 단순하거나 복잡할 수 있습니다. 단순 특성은 XML 스키마 속성으로, 또는 XML 스키마 단순 유형의 XML 스키마 요소로 정의될 수 있습니다. 복잡한 특성은 다른 비즈니스 오브젝트를 참조하거나 현재 비즈니스 오브젝트에서 복잡한 구조를 정의할 수 있습니다.

전체 XML 스키마 유형 시스템이 지원됩니다.

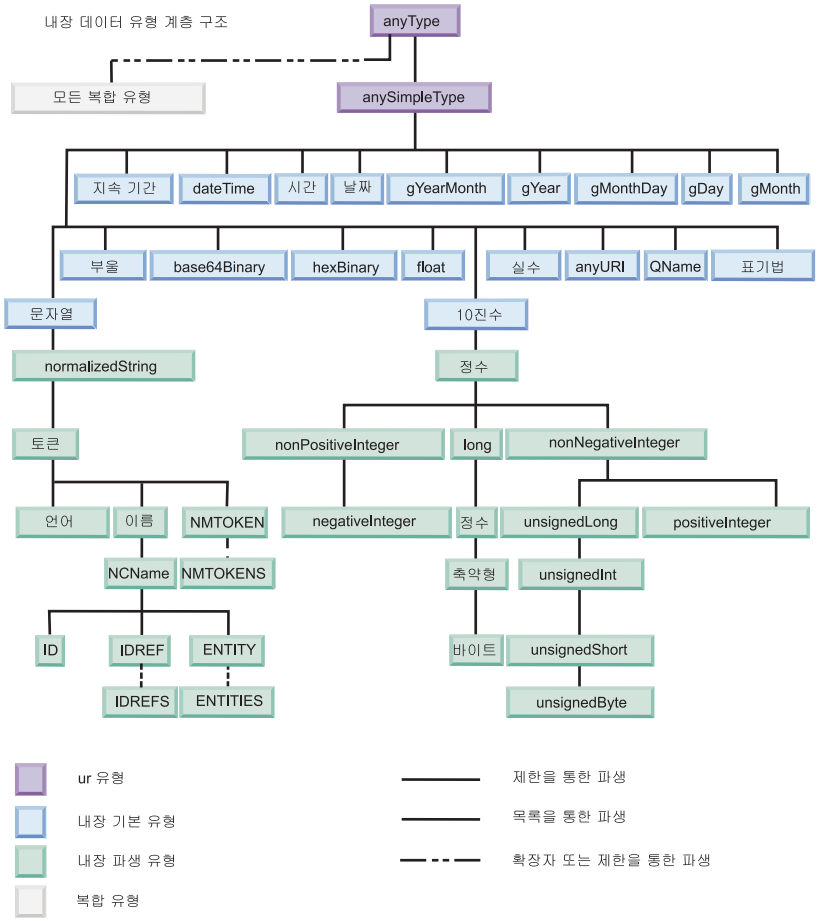


그림 56. XML 스키마 단순 유형

지원되는 XSD 및 WSDL 아티팩트:

WSDL 또는 스키마를 WebSphere Integration Developer의 프로젝트로 가져오는 경우, WSDL 또는 스키마를 통해 렌더링된 비즈니스 오브젝트는 모듈 개발에 사용할 수 있습니다. 그러나 스키마의 특정 아티팩트만 *비즈니스 오브젝트*로 렌더링된다는 점에 유의하십시오(예: 루트/최상위 레벨 요소 및 이름 지정된 복합 유형). 중첩된 익명 복합 유

형과 같은 특정 아티팩트는 비즈니스 오브젝트로 렌더링되지 *않습니다*. 이러한 제한사항은 XML 스키마에서 액세스할 수 있는 아티팩트의 결과입니다. 예를 들어, 하나의 비즈니스 오브젝트만 결과로 생성된 스키마를 가져오는 경우 나머지 요소는 익명 복합 유형일 수 있습니다. 다음 정보는 비즈니스 오브젝트를 생성하는 XSD 및 WSDL 아티팩트를 자세히 설명합니다.

가져온 XSD 정의의 비즈니스 오브젝트

XML 스키마를 프로젝트로 가져온 경우, 특정 아티팩트만 비즈니스 오브젝트로 렌더링됩니다. 다음 목록은 작성 및 런타임 시 지원되는 아티팩트를 보여줍니다.

작성 시 비즈니스 오브젝트를 결과로 생성하는 XSD 아티팩트

- 루트 레벨에서 정의된 복합 유형
- 루트 레벨에서 정의된 익명 복합 유형의 요소

이 아티팩트는 작성 시 비즈니스 오브젝트에 의해 참조될 수 있는 사용자 정의 단순 유형을 발생시킵니다.

- 루트 레벨에서 정의된 단순 유형
- 루트 레벨에서 정의된 익명 단순 유형의 요소

가져온 WSDL 파일의 비즈니스 오브젝트

인라인 XSD 스키마를 포함하는 WSDL 정의를 프로젝트로 가져온 경우, 특정 아티팩트만 비즈니스 오브젝트로 렌더링됩니다. 다음 목록은 작성 및 런타임 시 지원되는 아티팩트를 보여줍니다.

작성 시 비즈니스 오브젝트를 결과로 생성하는 인라인 XSD 아티팩트

- 루트 레벨에서 정의된 복합 유형
- 요소는 익명 복합 유형으로 루트 레벨에서 정의되었으며 요소의 이름에 조작/메시지의 이름이 포함되지 않음(이 요소는 WebSphere Integration Developer가 자동으로 언랩핑하는 doc-lit-wrapped 요소가 될 수 있으므로)

이 아티팩트는 작성 시 비즈니스 오브젝트에 의해 참조될 수 있는 사용자 정의 단순 유형을 발생시킵니다.

- 루트 레벨에서 정의된 단순 유형
- 루트 레벨에서 정의된 익명 단순 유형의 요소

XSD 아티팩트의 런타임 비즈니스 오브젝트

이 아티팩트는 런타임 시 비즈니스 오브젝트를 생성합니다.

- 루트 레벨에서 정의된 복합 유형
- 루트 레벨에서 정의된 익명 복합 유형의 요소

- 복합 유형을 참조하는, 루트 레벨에서 정의된 요소

WSDL 파일의 런타임 비즈니스 오브젝트

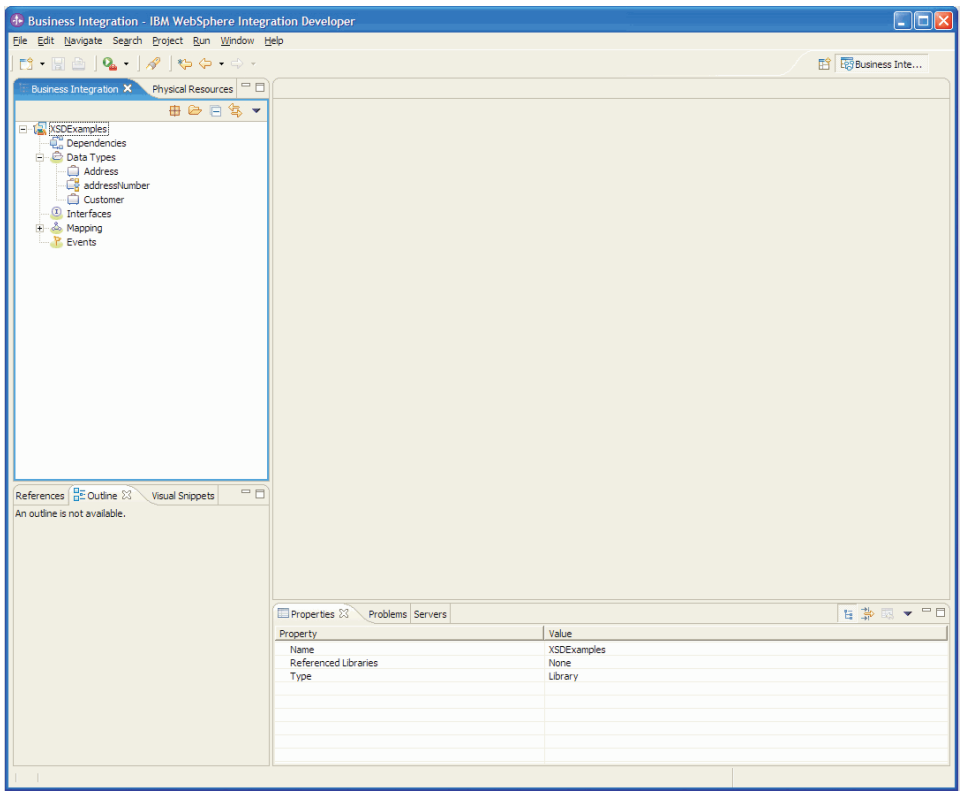
이 아티팩트는 런타임 시 비즈니스 오브젝트를 생성합니다.

- 루트 레벨에서 정의된 복합 유형
- 요소는 익명 복합 유형으로 루트 레벨에서 정의되었으며 요소의 이름에 조작/메시지의 이름이 포함되지 않음(이 요소는 WebSphere Integration Developer가 자동으로 언랩핑하는 doc-lit-wrapped 요소가 될 수 있으므로)
- 복합 유형을 참조하는, 루트 레벨에서 정의된 요소

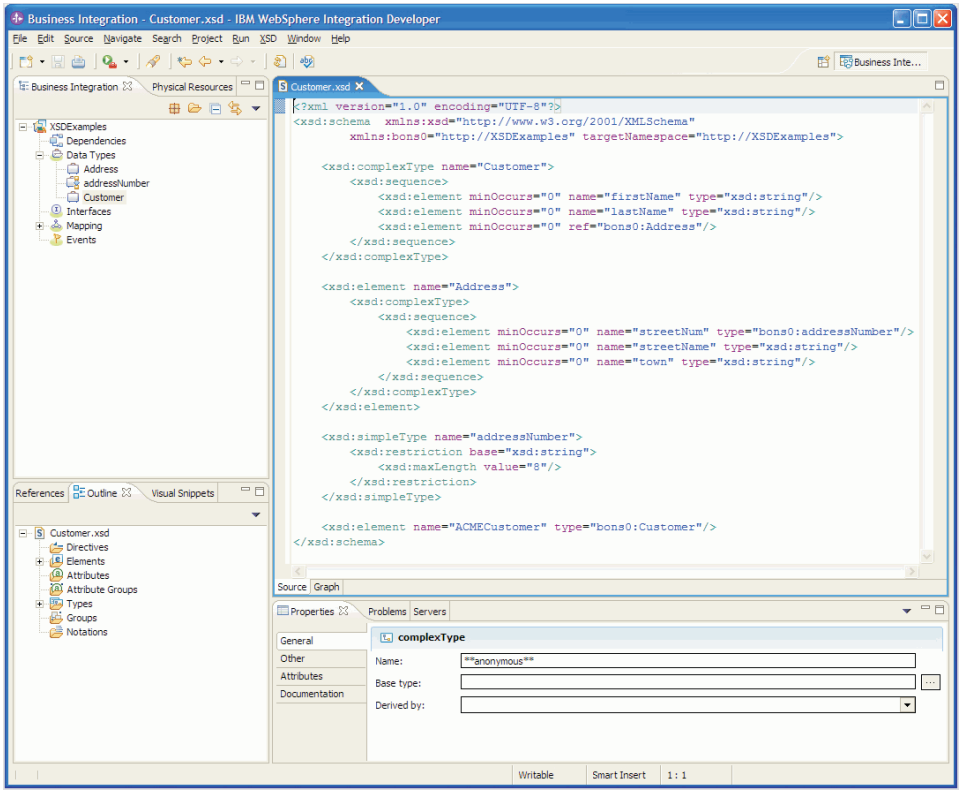
예제

1. **XSD 예제(작성 시 지원됨)**

   이 예제는 비즈니스 오브젝트가 표시되는 비즈니스 통합 보기의 프로젝트(XSDExamples)를 보여줍니다.



   이 보기는 XSD 스키마 편집기에 있는 Customer.xsd 파일을 보여줍니다.



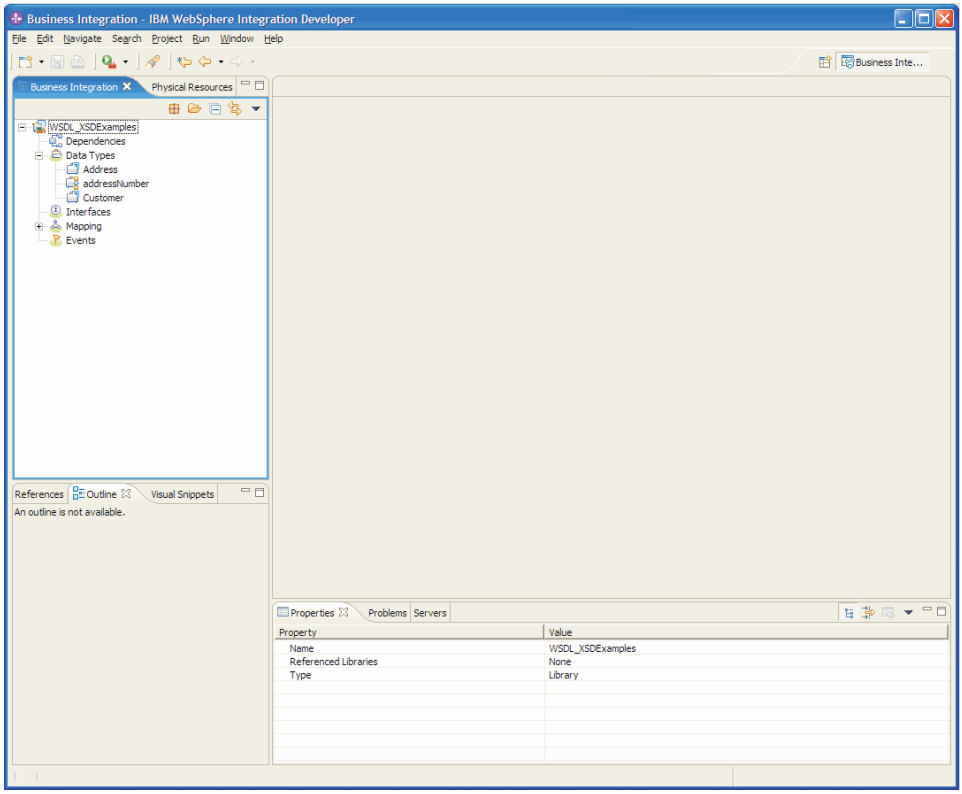
위의 예제에서는 다음을 지원합니다.

표 19. XSD 아티팩트 지원

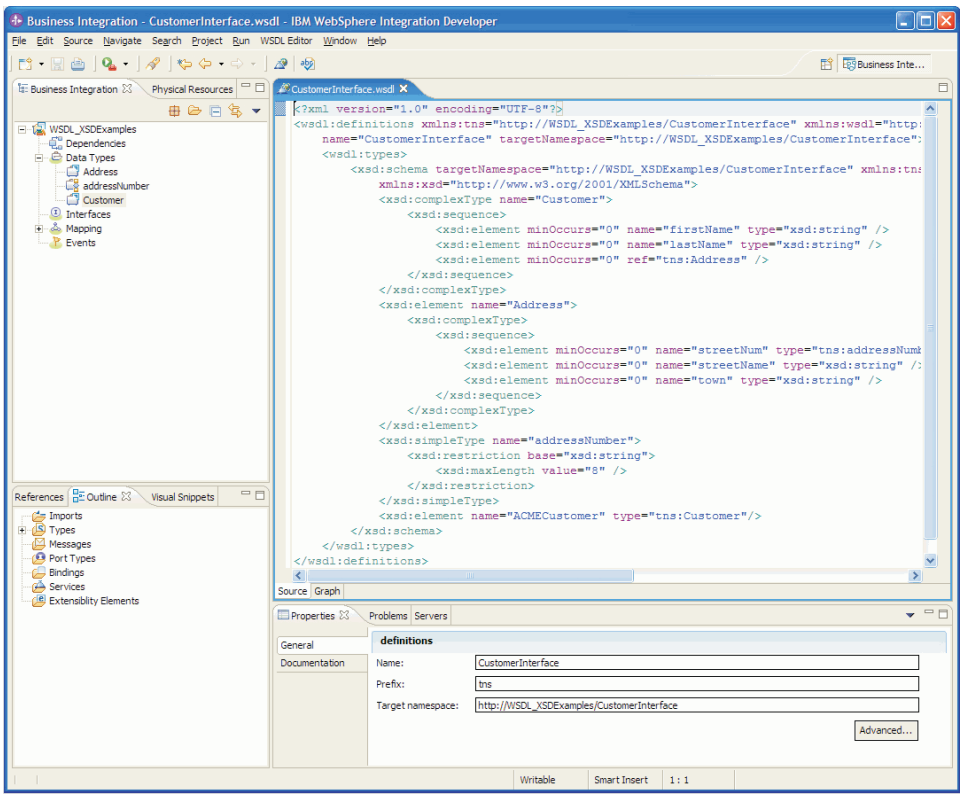
XSD 지원	**위의 예제에 있는 XSD 아티팩트**
루트 레벨에서 정의된 복합 유형	Customer
루트 레벨에서 정의된 익명 복합 유형의 요소	Address
루트 레벨에서 정의된 사용자 정의 단순 유형의 요소	addressNumber

2. WSDL 예제(작성 시 지원됨)

이 예제는 비즈니스 오브젝트가 표시되는 비즈니스 통합 보기의 프로젝트(WSDL_XSDExamples)를 보여줍니다.



이 화면 캡처는 WSDL 편집기에서 열린 CustomerInterface.wsdl 파일을 보여줍니다.



위의 예제에서는 다음을 지원합니다.

표 20. WSDL 아티팩트 지원

WSDL 지원	**위의 예제에 있는 인라인 XSD 아티팩트**
루트 레벨에서 정의된 복합 유형	Customer
요소는 익명 복합 유형으로 루트 레벨에서 정의되었으며 요소의 이름에 조작/메시지의 이름이 포함되지 않음(이 요소는 WebSphere Integration Developer가 자동으로 언랩핑하는 doc-lit-wrapped 요소가 될 수 있으므로)	Address
루트 레벨에서 정의된 사용자 정의 단순 유형의 요소	addressNumber

3. 런타임 예제

위의 예제는 다음 런타임 지원을 보여줍니다.

표 21. 런타임 아티팩트 지원

런타임 지원	**위의 예제에 있는 XSD 또는 인라인 XSD 아티팩트**
위의 예제 1 및 2에 있는 모든 것(단순 유형은 비즈니스 오브젝트가 아니므로 addressNumber는 제외)	위 참조(예제 1 및 2)
복합 유형을 참조하는, 루트 레벨에서 정의된 요소	ACMECustomer(예제 1 및 2에 표시됨)

플랫 및 계층 구조 비즈니스 오브젝트:

비즈니스 오브젝트는 플랫 또는 계층 구조로 모델링할 수 있습니다.

플랫 비즈니스 오브젝트

*플랫 비즈니스 오브젝트*에는 하나 이상의 단순 속성과 지원되는 verb 목록이 포함됩니다. 단순 속성은 String, Integer 또는 Date와 같은 하나의 값을 표시합니다. 모든 단순 속성에는 단일 카디널리티가 있습니다. 비즈니스 오브젝트가 응용프로그램 특정 비즈니스 오브젝트인 경우 플랫 비즈니스 오브젝트는 응용프로그램이나 기술 표준에서 하나의 엔티티를 표시할 수 있습니다.

계층 구조 비즈니스 오브젝트

계층 구조 비즈니스 오브젝트 정의는 여러 관련 엔티티의 구조를 정의하여, 각각의 개별 엔티티 뿐만 아니라 엔티티 간의 관계 측면도 캡슐화합니다. 하나 이상의 단순 속성을 포함하는 것 외에, 계층 구조 비즈니스 오브젝트에는 복잡한 하나 이상의 속성이 있습니다(즉, 속성 자체에 *하위 비즈니스 오브젝트*라는 하나 이상의 비즈니스 오브젝트가 포함됨). 복합 속성을 포함하는 비즈니스 오브젝트를 *상위 비즈니스 오브젝트*라고 합니다.

상위와 하위 비즈니스 오브젝트 간의 관계에는 두 가지 유형이 있습니다.

- 단일 카디널리티 - 상위 비즈니스 오브젝트의 속성이 하나의 하위 비즈니스 오브젝트를 표시하는 경우. 속성의 유형은 하위 비즈니스 오브젝트의 이름으로 설정되고 카디널리티는 1로 설정됩니다.
- 다중 카디널리티 - 상위 비즈니스 오브젝트의 속성이 하위 비즈니스 오브젝트 배열을 표시하는 경우. 속성의 유형은 하위 비즈니스 오브젝트의 이름으로 설정되고 카디널리티는 *n*으로 설정됩니다.

각 하위 비즈니스 오브젝트에는 하위 비즈니스 오브젝트나 비즈니스 오브젝트 배열 등이 포함되는 속성이 포함될 수 있습니다. 계층 구조의 맨 위에 있는 비즈니스 오브젝트(자체에 상위 비즈니스 오브젝트가 없음)는 *최상위 레벨 비즈니스 오브젝트*라고 합니다. 단일 비즈니스 오브젝트가 포함할 수 있는(또는 단일 비즈니스 오브젝트를 포함할 수 있는) 하위 비즈니스 오브젝트에 관계없이, 단일 비즈니스 오브젝트를 *개별 비즈니스 오브젝트*라고 합니다.

예제

다음 예제는 플랫 비즈니스 오브젝트와 계층 구조 비즈니스 오브젝트 사이의 차이를 보여주는 데 도움이 됩니다. 다이어그램에는 Product라고 하는 플랫 비즈니스 오브젝트가 포함됩니다. 비즈니스 오브젝트는 서비스 데이터 오브젝트 유형이 `commonj.sdo.DataObject`인 메모리에 표시됩니다(정적으로 생성되지 않은 경우). 이

플랫 비즈니스 오브젝트에는 XML 스키마 단순 유형으로 모델링되는 속성 세트와 단순 유형 목록으로 모델링되는 속성이 있습니다.

다이어그램은 또한 ProductCategory 비즈니스 오브젝트와의 조합으로, 한층 복잡한 계층 구조 비즈니스 오브젝트를 작성하기 위한 Product 비즈니스 오브젝트도 보여줍니다. 이 비즈니스 오브젝트에는 최상위 레벨 비즈니스 오브젝트(ProductCategory)와 포함된 비즈니스 오브젝트(Product)가 있습니다.

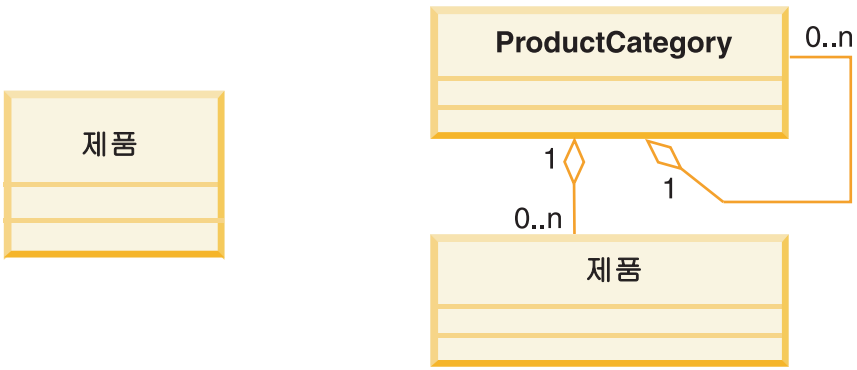


그림 57. 플랫 및 계층 구조 비즈니스 오브젝트의 비교

다음은 Product 비즈니스 오브젝트에 대한 플랫 비즈니스 오브젝트 정의의 예제입니다. Product 비즈니스 오브젝트는 XML 스키마 단순 유형 `xs:string` 및 `xs:int`로 유형이 지정된 두 개의 특성 *Name* 및 *Inventory*를 정의합니다. 또한 Product는 `xs:string` 단순 유형의 목록인 목록 특성 *Color*의 정의를 보여줍니다.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <complexType name="Product">
  <sequence>
   <element name="name" type="string"/>
   <element name="inventory" type="int"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</schema>
```

다음은 계층 구조 비즈니스 오브젝트인 ProductCategory의 예제입니다. 정의는 두 개의 다른 비즈니스 오브젝트인 ProductCategory 및 Product를 정의합니다. 계층 구조 ProductCategory 비즈니스 오브젝트는 특성 *Name*을 정의하고 `Product` 또는 `ProductCategory` 유형의 비즈니스 오브젝트 목록을 정의합니다.

```
<schema>
 targetNamespace="http://www.scm.com/ProductCategoryTypes"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">
 elementFormDefault="qualified">

 <complexType name="ProductCategory">
  <sequence>
   <element name="name" type="string"/>
   <choice>
    <element name=="productCategory"
     type="pc:ProductCategory"
     maxOccurs="unbounded"/>
    <element name=="product"
     type="pc:Product"
     maxOccurs="unbounded"/>
   </choice>
  </sequence>
 </complexType>

 <complexType name="Product">
  <sequence>
   <element name="name" type="string"/>
   <element name=="inventory" type="int"/>
   <element name=="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>

</schema>
```

비즈니스 오브젝트 특성:

비즈니스 오브젝트에는 비즈니스 오브젝트 프레임워크 내에서의 사용을 확장시키는 고유 특성이 있습니다.

카디널리티

특성의 카디널리티는 단순 및 복합 유형에 대해 표준 XML 스키마 minOccurs 및 maxOccurs 패싯에 의해 정의 되고 속성에 대해 use 속성으로 정의됩니다.

기본 특성 값

비즈니스 오브젝트의 속성 및 단순 유형에 대한 XML 스키마의 기본값을 제공하기 위한 성능은 비즈니스 오브젝트 프레임워크에 의해 지원됩니다. 이 지원은 비즈니스 오브젝트의 단순 특성 유형이 기본값을 반영하는 경우 작성할 때 표시됩니다.

Nillable

요소는 XML 스키마에서 nillable 상태가 되도록 정의될 수 있습니다. 비즈니스 오브젝트 프레임워크는 *nillable* 상태의 특성이 런타임 시 해당 값을 Null로 설정할 수 있도록 합니다.

키 정의

관계, 순서 지정 및 분리와 같은 여러 서브시스템에서 비즈니스 오브젝트 키 정보를 사용할 수 있습니다. 그러나 이러한 각 서브시스템은 비즈니스 오브젝트 키 정의와 독립적으로 자체 키 메커니즘을 정의할 수 있습니다. 비즈니스 오브젝트에서 활용되는 기본적인 모델 언어는 XML 스키마이므로 키 정의에 대한 첫 번째 클래스 지원이 모델링 언어 내에 존재합니다. 그러나 모델링 언어 내에서의 이러한 지원이 SDO 런타임에서 완전히 지원되는 것은 아닙니다.

xs:ID, xs:IDREF 및 xs:IDREFS

이 유형은 원래 DTD의 업그레이드 경로를 제공하기 위해 XML 스키마에 추가되었습니다. 각각의 복합 유형에는 xs:ID로 유형이 지정된 하나의 요소/속성이 있거나 전혀 없을 수 있습니다. 데이터베이스의 1차 키(예: 테이블 범위에 대해 고유해야 함)와 반대로 ID는 전체 문서에 대해 고유해야 합니다. 예로서, 준수 문서는 Product 및 ProductCategory 둘 다를 식별하기 위해 동일한 ID 값을 사용할 수 없습니다. 요소는 주로 키 값 앞에 복합 유형 이름을 추가하여 이 제한사항을 해결합니다. IDREF 유형 지정된 속성에는 현재 문서에 있는 ID 값 중 하나와 일치하는 값이 있어야 합니다. 또한 XML 스키마는 ID 참조 목록(xs:IDREFs)을 포함하도록 유형을 지정할 수 있는 요소에 해당되는 구성에 대해 제공됩니다.

xs:unique, xs:key, xs:keyref

XML 스키마는 키 정의 및 키 참조를 사용 가능하도록 하는 새 스타일을 도입했습니다. 사용자는 xs:unique 구성을 사용하여 특정 요소 범위(전체 문서를 나타내는) 내에서 고유해야 하는 하나 이상의 필드를 요소에 정의할 수 있습니다. xs:key 구성은 xs:unique의 변형으로, 참조된 요소가 필요한 추가 제한조건이 있습니다. xs:keyref 구성은 요소의 값이 이름 지정된 키 또는 고유 구성이어야 하는지 식별하기 위해 사용됩니다.

unique, key 및 keyref 구성에는 ID, IDREF 및 IDREFS 세트에 비해 다음과 같은 몇 가지의 장점이 있습니다.

- 복합 키를 정의할 수 있습니다.
- 문서의 부분에 상대적인 고유 제한조건을 정의할 수 있습니다.

비즈니스 오브젝트가 반드시 정의된 키를 보유할 필요는 없지만 보유할 것을 권장합니다. 키를 정의하지 않는 비즈니스 오브젝트는 응용프로그램에서 사용할 수 있습니다. 이 시나리오는 많은 Java EE 중심 응용프로그램 사용 모델에서 공통적인 사용 모델입니다. 이 때 JavaBean은 키 스펙 없이 서블릿과 EJB 컨테이너 사이에 앞뒤로 전달됩니다. 그러나 키를 정의하지 않는 비즈니스 오브젝트는 키가 필요한 서브시스템과 상호작용할 수 없습니다. 이 상황은 WebSphere Process Server 서비스 품질을 이용할 수 있는 기능을 제한합니다.

비즈니스 그래프 모델링

비즈니스 그래프는 SDO DataGraph가 SDO DataObject와 관련되는 것과 동일한 방식으로 비즈니스 오브젝트에 관련됩니다. 최상위 레벨 비즈니스 오브젝트가 WebSphere Process Server에서 제공되는 서비스를 사용할 수 있도록 강화되어야 하는 경우 비즈니스 그래프로 랩핑됩니다. 비즈니스 그래프 랩퍼는 비즈니스 그래프가 직렬화될 때 논리적으로 메모리에, 또는 물리적으로 정보를 저장하기 위해 데이터 헤더를 추가하여 추가 값을 제공합니다.

주: WebSphere InterChange Server에서 응용프로그램을 이주하거나 어댑터를 이주하는 경우 비즈니스 그래프를 사용해야 할 수도 있습니다.

비즈니스 그래프 사용 모델:

두 가지의 1차 사용 모델은 비즈니스 그래프가 제공하는 기본적인 기능인 델타 지원과 이후 이미지를 표시합니다.

*델타 지원*은 SDO 1.0에서 사용 가능한 기능으로, 비즈니스 오브젝트 그래프 변경사항이 *변경 요약*이라고 하는 특수 헤더에서 캡처됩니다.

*이후 이미지*는 보통 EIS에서 해당 데이터를 변경한 결과로, EIS 시스템에서 비즈니스 데이터의 현재 상태를 캡처하는 비즈니스 그래프입니다. 이후 이미지를 통해 EIS 시스템에서 변경사항을 캡처하여 런타임에 공개할 수 있습니다.

이러한 두 가지의 기본적인 개념을 제공하기 위해, 다음 개념에 대해 비즈니스 그래프가 도입되어 제공됩니다.

- *템플리트화된 비즈니스 그래프*는 비즈니스 그래프가 랩핑되는 비즈니스 오브젝트 그래프의 유형에 대해 특별히 유형 지정되는 비즈니스 그래프입니다.
- *변경 요약*은 델타 및 이후 이미지 사용 모델 둘 다에 대해 내재적 변경사항과 명시적 변경사항을 캡처하도록 제공됩니다.
- *명시적 변경 요약* 지원은 어댑터, 중개자, 맵 및 관계와 같은 BPM 컴포넌트가 변경 요약 헤더를 명시적으로 수정할 수 있도록 하는 비즈니스 그래프 프로그래밍 인터페이스에서 제공됩니다.
- *이벤트 요약*은 비즈니스 오브젝트 그래프의 데이터에 대한 인스턴스 기반 어노테이션의 캡처를 지원하기 위해, 특히 오브젝트 이벤트 ID를 포함하기 위해 제공됩니다.
- *Verb* 지원은 런타임의 컴포넌트가 부가 가치 기능을 수행하기 위해 이벤트 유형을 키 오프할 수 있도록 하는 비즈니스 그래프에 의해 제공됩니다.
- *지원되는 verb*는 비즈니스 그래프에 대해 지정될 수 있는 허용 가능한 verb 세트를 제한하고 확장하는 표기입니다.
- *오브젝트 이벤트 ID*는 그래프의 모든 오브젝트가 고유하게 식별될 수 있도록 하는 비즈니스 그래프에 의해 지원됩니다. 이 성능은 부가 가치 기능을 제공하는 일부 컴포넌트에 필요합니다.

비즈니스 그래프 모델 정의:

외부에서 개발된 비즈니스 오브젝트 모델에 대해 비침입 상태를 유지하기 위해 원래 비즈니스 오브젝트 주변에 비즈니스 그래프 기능이 랩핑됩니다. 템플리트화된 비즈니스 그래프를 이름 지정한 패턴은 원래 비즈니스 오브젝트를 내용이 많아진 비즈니스 그래프 스키마로 랩핑하는 데 사용됩니다.

*템플리트화된 비즈니스 그래프*는 비즈니스 오브젝트 프레임워크 런타임에 의해 제공되는 비즈니스 그래프 복합 유형을 확장하고 원래 비즈니스 오브젝트로 위임하는 요소를 추가하여 작성됩니다. 다이어그램은 비즈니스 그래프에 대한 UML 모델을 표시합니다. 비즈니스 그래프는 최상위 레벨 비즈니스 오브젝트에 추가되는 표준 헤더 세트만을 제공하는 추상적인 그래프입니다.

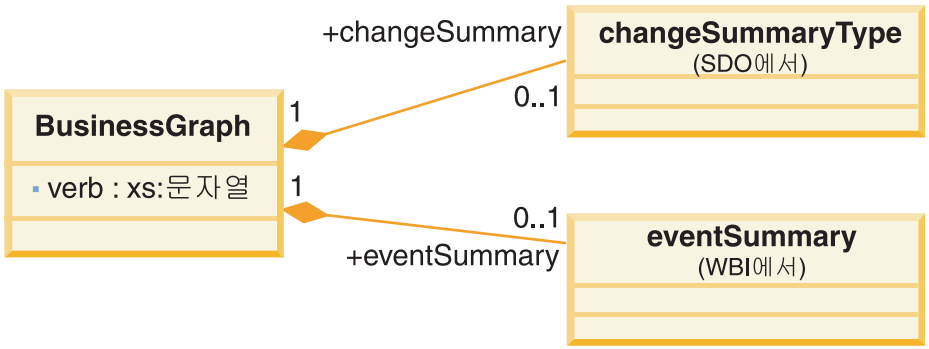


그림 58. 비즈니스 그래프 복합 유형

추상 비즈니스 그래프 XML 스키마 복합 유형의 XML 스키마 모델

```
<schema
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:sdo="commonj.sdo"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="commonj.sdo" schemaLocation="DataGraph.xsd"/>

 <complexType name="BusinessGraph" abstract="true">
  <sequence>
   <element name="changeSummary" type="sdo:ChangeSummaryType"
    minOccurs="0" maxOccurs="1"/>
   <element name="eventSummary" type="bo:EventSummary"
    minOccurs="0" maxOccurs="1"/>
   <element name="property" type="bo:ValuesType"
    minOccurs="0"/>
```

```
 </sequence>
 <anyAttribute namespace="##other" processContents="lax"/>
</complexType>

<complexType name="EventSummary">
 <sequence>
  <any namespace="##any" processContents="lax"
   minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>

<complexType name="ValuesType">
 <complexContent>
  <extension base="ecore:EClass"/>
 </complexContent>
</complexType>

<attribute name="name" type="string"/>
</schema>
```

비즈니스 그래프는 단지 최상위 레벨 개념입니다. 기존의 최상위 레벨 비즈니스 오브젝트에 헤더 세트를 추가하기 위해 존재하므로, 비즈니스 오브젝트에서와 같이 순환 설계 패턴으로 모델링될 수 없습니다. 비즈니스 그래프는 모든 비즈니스 오브젝트에 적용될 수 있지만, 적용 시 해당 비즈니스 오브젝트는 최상위 레벨 비즈니스 오브젝트가 됩니다.

다음은 ProductCategory라고 하는 비즈니스 오브젝트를 랩핑하는 비즈니스 그래프의 예제입니다. ProductCategory는 Product라고 하는 하위 비즈니스 오브젝트를 포함하는 계층 구조 비즈니스 오브젝트입니다.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes

 /ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="http://www.scm.com/ProductCategoryTypes"
  schemaLocation="ProductCategoryTypes.xsd"/>

 <complexType name="ProductCategoryBG">
  <complexContent>
   <extension base="bo:BusinessGraph">
    <sequence>
     <element name="verb" minOccurs="0" maxOccurs="1"/>
     <element name="productCategory"
      type="pc:ProductCategory"
      minOccurs="0" maxOccurs="1"/>
```

```
    </sequence>
   </extension>
  <complexContent>
 <complexType>
</schema>
```

생성된 템플리트화 비즈니스 그래프에 대한 권장 패턴은 다음과 같습니다.

- 템플리트화된 비즈니스 그래프는 제한조건에 의해 bo:BusinessGraph 스키마를 확장하는 이름 지정된 복합 유형을 사용하여 정의됩니다(이 패턴은 허용 가능한 Verb 값에 대한 제한사항을 제공함).
- 템플리트화된 비즈니스 그래프의 이름은 문자열 "BG가 추가된 최상위 레벨 비즈니스 오브젝트의 이름입니다.
- 템플리트화된 비즈니스 그래프의 대상 네임스페이스는 템플리트화된 비즈니스 그래프의 복합 유형에 의해 이름 지정되는 랩핑된 비즈니스 오브젝트의 대상 네임스페이스로 구성되며 뒤에 "/가 붙습니다.
- 템플리트화된 비즈니스 그래프 복합 유형 정의는 해당 이름이 복합 유형의 이름에 해당되는 고유 XML 스키마 파일에 위치합니다.

비즈니스 그래프 모델 인스턴스:

최상위 레벨 비즈니스 그래프는 비즈니스 오브젝트와 아주 동일하게, SDO 1.0 DataObject(특히 클래스 commonj.sdo.DataObject)로 메모리에 표시됩니다.

비즈니스 그래프는 비즈니스 그래프 포함 오브젝트와 그 이상으로 구성됩니다. 또한 두 개의 헤더와 최상위 레벨 비즈니스 오브젝트도 포함됩니다. 어떤 헤더도 메모리에서 DataObject로 표시되지 않으며, 어느 헤더도 DataObject API 액세스 메커니즘을 사용하지 않습니다.

요약 헤더 변경

비즈니스 그래프에 의해 제공되는 기능은 비즈니스 그래프가 서로 다른 몇몇 비즈니스 프로세스 사이에 전달되는 대로 비즈니스 그래프에서 비즈니스 오브젝트에 대한 변경사항을 내부적으로 추적할 수 있는 기능입니다. 각각의 프로세스가 비즈니스 그래프를 변경하므로, 메모리에 변경 로그가 생성됩니다. 비즈니스 그래프가 직렬화되면 다음 프로세스가 비즈니스 그래프에 대해 작성된 변경사항 유형을 볼 수 있도록 하는 형식으로 변경 로그가 작성됩니다. 이 기술을 사용하면 어댑터 및 데이터 중개자 서비스가 초점을 맞춰야 하는 데이터를 최적화하여 지속적인 데이터 저장소를 효율적으로 갱신할 수 있습니다.

또한 변경 요약은 어댑터가 EIS 시스템에서 갱신된 데이터를 설명하기 위해 이후 이미지 이벤트를 생성할 때도 사용됩니다. 특히, 오브젝트 및 특성 변경사항에 어노테이션을 작성하기 위한 변경 요약 기능은 이후 이미지 사용 모델(그리고 비즈니스 그래프에 대한 verb)에서 사용됩니다.

내부 변경 요약 사용법

응용프로그램이 비즈니스 오브젝트를 변경할 경우, 응용프로그램은 변경 로깅을 작동시킬 수 있으므로 변경 이벤트가 자동으로 변경 요약에 로그됩니다. 변경 요약은 최상위 레벨 비즈니스 오브젝트의 피어 레벨에서 비즈니스 그래프에 포함됩니다. 변경 이벤트는 비즈니스 오브젝트와 비즈니스 오브젝트 특성에 대해 두 레벨에서 정의됩니다. 비즈니스 오브젝트에는 변경 작성, 갱신 및 삭제 유형이 있고 특성에는 변경 설정 및 설정 해제 유형이 있습니다. 변경 이벤트는 비즈니스 그래프의 비즈니스 오브젝트 부분에 대한 수정사항만 추적합니다. 이벤트 요약의 변경사항으로 인해 비즈니스 그래프에 대한 변경 요약의 내부 갱신이 발생하지는 않습니다.

명시적 변경 요약 수정사항

변경 요약 헤더에 명시적으로 쓸 수 있는 기능이 필요한 WebSphere Process Server 컴포넌트에 몇 가지의 사용 모델이 있습니다. 예를 들어, EIS 이벤트를 생성하는 어댑터는 명시적으로 오브젝트 변경 유형을 작성하고 잠재적으로는 특성 변경 유형을 작성합니다. 응용프로그램 특정 비즈니스 오브젝트/일반 비즈니스 오브젝트(ASBO/GBO) 맵은 변경 요약을 하나의 비즈니스 그래프에서 다른 비즈니스 그래프로 변환하여, 출력 비즈니스 그래프에 변경 요약의 새 버전을 작성합니다. 이 기능은 비즈니스 오브젝트 프레임워크에 의해 제공됩니다.

이벤트 요약 헤더

이벤트 요약은 런타임에 표시되는 오브젝트의 인스턴스를 고유하게 식별하기 위해 사용되는 메커니즘인 **ObjectEventID**를 제공합니다. 이 정보는 이벤트 요약으로 전달됩니다. 고유 ID가 비즈니스 그래프의 비즈니스 오브젝트 계층 구조에서 지정된 DataObject와 연관됩니다.

이벤트 정보도 이벤트 요약으로 전달될 수 있습니다. 이 정보는 비즈니스 그래프에 대한 비즈니스 오브젝트 계층 구조에서 각 오브젝트와 연관되는 추가 메타데이터를 추가하기 위해 사용할 수 있는 문자열입니다. 이벤트 정보에 대한 하나의 가능한 사용 모델은 변경 요약에서 지원되는 표준 작성, 갱신 및 삭제 verb가 아닌 다른 verb로 포함된 비즈니스 오브젝트를 마크업하는 것입니다.

Verb 헤더

비즈니스 그래프에 Verb가 설정된 경우 비즈니스 그래프의 비즈니스 오브젝트 데이터 부분이, 설정된 EIS 이후 이미지 데이터를 전달합니다. Verb에 값이 포함된 경우 이후 이미지 이벤트의 정밀도에 대한 세 가지 가능성이 있습니다.

- 변경 요약이 비어 있습니다. 이 상황은 EIS가 데이터 세트에 대한 갱신 유형은 알지만 그래프의 오브젝트가 작성, 갱신 또는 삭제된 세부사항은 가지고 있지 않은 경우에 발생합니다. 결과적으로, 다운스트림 중개, 맵, 관계 또는 어댑터가 수행할 실제 갱신을 판별하기 위해 비즈니스 그래프의 정보와 추가 데이터를 사용해야 합니다.

- 변경 요약에 오브젝트 레벨 변경 이벤트 어노테이션이 있습니다. 이러한 경우는 EIS 시스템이 비즈니스 그래프의 각 오브젝트에 대해 발생한 사항을 인식하지만 오브젝트의 특정한 특성이 갱신되었는지 여부를 판별하기에는 정밀도가 부족한 경우에 일반적입니다.
- 변경 요약에 오브젝트 레벨 변경 이벤트 어노테이션이 있고 특성 레벨 get/set 어노테이션이 있습니다. 이 상황은 가장 정밀한 경우로, 모든 어댑터는 해당되는 EIS 시스템이 적절한 데이터를 사용 가능하도록 만들 경우 이 레벨의 이후 이미지를 얻으려고 합니다. 완전하게 지정된 이후 이미지의 장점은 특성 레벨 내부 특성 변경 이벤트 관리가 발생하도록 하는 것입니다. 따라서 이후 이미지 비즈니스 그래프가, 연결이 끊긴 델타 기반 비즈니스 그래프와 상호작용하는 것이 한층 쉬워집니다.

비즈니스 오브젝트 유형 메타데이터 모델링

비즈니스 오브젝트 유형 메타데이터를 비즈니스 오브젝트 정의에 추가하여 런타임 시 가치를 향상시킬 수 있습니다. 비즈니스 오브젝트 프레임워크를 통해 사용자는 비즈니스 오브젝트 유형 메타데이터를 설계, 어노테이트 및 변환할 수 있습니다.

비즈니스 오브젝트 프레임워크는 다음을 가능하게 하는 메커니즘을 제공합니다.

- 일관성이 있고 상대적으로 비침입 형태로 메타데이터를 비즈니스 오브젝트에 혼합할 수 있습니다.
- 개발자에 대한 규범적 정책에서 복잡한 어노테이션 구조를 정의할 수 있습니다.
- 비즈니스 오브젝트 개발자 및 전개자에 대한 규범적 정책에서 사전 정의된 복잡한 어노테이션 구조를 준수하는 인스턴스 메타데이터로 비즈니스 오브젝트 어노테이션을 작성할 수 있습니다.
- 런타임 시 어노테이션을 변환하기 위한 API 세트가 쉽게 데이터 오브젝트 구조를 사용할 수 있습니다.

이 프로세스에는 세 가지 이상의 역할이 관련됩니다.

- 1차 역할은 비즈니스 오브젝트 유형 메타데이터 디자이너 역할입니다. 이 역할은 메타데이터 구조를 설계합니다. 예를 들어, 비즈니스 오브젝트 프레임워크는 이 역할을 수행하고 비즈니스 오브젝트에 어노테이션을 작성하기 위해 몇 가지 메타데이터 특성을 정의합니다. PeopleSoft, Siebel 및 SAP와 같은 어댑터가 메타데이터 디자이너의 역할을 수행하여 비즈니스 오브젝트에 응용프로그램 특정 정보로 어노테이션을 작성할 것으로 예상됩니다.
- 두 번째 역할은 비즈니스 오브젝트 디자이너 또는 전개자입니다. 이 역할은 비즈니스 오브젝트 유형 메타데이터의 구조와, 비즈니스 오브젝트 유형 메타데이터 프레임워크에 의해 정의된 정책을 사용하여 비즈니스 오브젝트 정의에 메타데이터로 어노테이션을 작성합니다.

- 비즈니스 오브젝트에 올바르게 어노테이션이 작성된 경우, 세 번째 역할은 비즈니스 오브젝트 메타데이터 API를 사용하여 유효성을 검증하고 런타임 시 메타데이터를 검사한 후 탐색 가능하고 유용한 데이터 오브젝트 그래프 구조로 바꿀 수 있습니다.

또한 비즈니스 오브젝트 프레임워크는 다음 메타데이터 기능에 대해 제공됩니다.

- 복합 1차 키 및 외부 키 특성 정의
- 최상위 레벨에서 지원하는 verb 메타데이터 어노테이션

세부사항은 기타 주제를 참조하십시오.

비즈니스 오브젝트 유형 메타데이터 표시:

비즈니스 오브젝트 프레임워크는 비즈니스 오브젝트 유형 메타데이터가 하향식으로 또는 외부적으로 개발된, 가져온 스키마로 혼합될 수 있는 메커니즘을 정의합니다.

비즈니스 오브젝트 유형 메타데이터에서 혼합하기 위해 사용되는 메커니즘은 XML 스키마 어노테이션 및 `appInfo` 구조입니다. 이 정책을 사용하려면 원래 스키마 정의를 수정해야 하지만, 이 정책은 어노테이션 및 `appinfo`가 보기를 쉽게 토글하거나 완전하게 제거될 수 있도록 하는 방식으로 수정됩니다. 이러한 식별 설계는 값이 비즈니스 오브젝트 유형 메타데이터를 정의하는 대상 네임스페이스인 `xs:appinfo` 태그의 소스 속성을 사용하여 수행됩니다.

예를 들어, 다음 스키마를 런타임에 가져왔다고 가정합니다. 스키마는 고객 어노테이션을 포함하는 Product라고 하는 비즈니스 오브젝트를 설명합니다.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string"

   default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>
```

```
 <simpleType name="Sku">
  <restriction base="string">
   <pattern value="₩d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

비즈니스 오브젝트 유형 메타데이터 설계:

비즈니스 오브젝트 메타데이터를 수용하기 위해 메타데이터 구조를 설계할 수 있습니다.

이 태스크 정보

메타데이터 구조를 설계하려면 다음 단계를 수행하십시오.

프로시저

1. 유효한 대상 네임스페이스가 있는 XML 스키마 파일을 작성하십시오. 이 단계는 인스턴스 메타데이터가 비즈니스 오브젝트 정의에 추가될 때 사용됩니다.
2. 비즈니스 오브젝트에 추가할 수 있는 메타데이터의 각 독립 조각을 정의하려면 이름 지정된 복합 유형을 정의하십시오. 복합 유형 정의는 인스턴스 메타데이터를 읽기 위해 사용되는 동적으로 유형 지정된 DataObject의 구조를 정의하는 데 사용됩니다. 복합 유형의 이름은 인스턴스 메타데이터가 비즈니스 오브젝트 정의에 추가될 때 사용됩니다.

예

이 예제에서는 PeopleSoft 어댑터용으로 개발된 비즈니스 오브젝트 유형 메타데이터의 일부를 보여줍니다.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <complexType name="PSFTBODefinitionASI">
  <sequence>
   <element name="hostname" type="string"/>
   <element name="ipaddress" type="string"/>
  </sequence>
 </complexType>
</schema>
```

비즈니스 오브젝트 정의에 어노테이션 작성:

비즈니스 오브젝트 프레임워크에 의해 지원되는 구문을 사용하여 비즈니스 오브젝트 정의에 어노테이션을 작성할 수 있습니다.

이 태스크 정보

비즈니스 오브젝트 프레임워크는 인스턴스 메타데이터 데이터가 비즈니스 오브젝트 정의에 첨부되는 메커니즘을 정의하려고 하지 않습니다. 그러나 인스턴스 메타데이터의 구문을 정의합니다.

비즈니스 오브젝트 정의에 어노테이션을 작성하려면 다음 설명대로 수행하십시오.

프로시저

1. 인스턴스 메타데이터가 첨부될 비즈니스 오브젝트 정의의 일부에 아직 어노테이션이 없는 경우 추가하도록 하십시오. 이미 어노테이션이 있으면 기존 어노테이션을 이용하십시오.
2. xs:annotation 태그 내에서 xs:appinfo 태그를 작성하고, 추가되는 비즈니스 오브젝트 유형 메타데이터가 정의하는 네임스페이스에 대해 소스 속성을 정의하십시오.
3. xs:appinfo 태그 내에서, 대상 네임스페이스 접두부와 뒤에 오는 복합 유형 정의 이름을 사용하여 QName을 정의하십시오. xmlns:와 이전에 사용된 대상 네임스페이스 접두부로 정의된 속성 QName을 추가하십시오. 이 QName 속성의 값을 사용되는 비즈니스 오브젝트 유형 메타데이터의 대상 네임스페이스로 설정하십시오.
4. 비즈니스 오브젝트 유형 메타데이터 정의에 의해 지시된 구조 및 인스턴스 데이터를 추가하십시오. 모든 태그를 적절하게 닫으십시오.

예

오브젝트를 가져오면 이 오브젝트는 PeopleSoft 어노테이션 세트가 필요하다고 식별됩니다. 이 예제는 추가된 PeopleSoft 메타데이터와 동일한 비즈니스 오브젝트 정의를 보여줍니다.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appinfo source="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
    <psft:PSFTMetadata
     xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
     <hostname>mumbai</hostname>
     <ipaddress>9.29.1.1</ipaddress>
    <psft:PSFTMetadata>
   </appInfo>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
```

```
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string" default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>

 <simpleType name="Sku">
  <restriction base="string">
   <pattern value="₩d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

어노테이션을 DataObject로 변환:

비즈니스 오브젝트 프레임워크는 어노테이션을 사용 가능한 DataObject 구조로 변환할 수 있는 성능을 제공합니다.

이 태스크 정보

비즈니스 오브젝트 정의와 연관되는 어노테이션은 SDO 구현 특정의 API 세트를 사용하여 런타임 시 읽을 수 있습니다. 그러나 이 API에 대한 문제점은 2진 대형 오브젝트(BLOB)를 리턴한다는 것입니다. 그러나 비즈니스 오브젝트 프레임워크는 (권장되는 어노테이션 패턴을 따른 경우) BLOB를 읽고, 유효성을 검증한 후 사용 가능한 DataObject 구조로 변환하는 유틸리티를 제공합니다.

어노테이션을 DataObject로 변환하려면 다음을 수행하십시오.

프로시저

1. 어노테이션을 가져오십시오.
2. BOTypeMetadata를 사용하여 해당 SDO DataObject로 변환하십시오. BOTypeMetadata 구현은 BOTypeMetadata.INSTANCE 인스턴스를 사용하여 싱글톤으로 사용 가능합니다.

예

다음 예제는 API를 사용하여 어노테이션을 가져온 후 BOTypeMetadata API를 사용하여 해당 어노테이션을 SDO DataObject로 변환하는 방법을 보여줍니다. 메타데이터는 BOTypeMetadata.xsd에서 정의됩니다.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
```

```
 <complexType name="VerbInfo">
  <sequence>
   <element name="verbInfo" type="string"/>
  </sequence>
 </complexType>
</schema>
```

비즈니스 오브젝트 프레임워크의 기능 중 하나는 지원되는 추가 verb 및 동반 메타데이터를 추가하여 런타임 시 메타 구동 기능을 사용할 수 있도록 하는 것입니다. 이 기능은 verb에 추가 메타데이터 어노테이션을 작성하기 위해 verb 및 VerbInfo 메타데이터를 사용함으로써 지원됩니다. 다음 예제는 VerbInfo 메타데이터가 가능한 각각의 Verb 값으로 추가되는 위치를 보여줍니다.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:sdo="commonj.sdo"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="commonj.sdo"
  schemaLocation="DataGraph.xsd"/>

 <import namespace="http://www.scm.com/ProductCategoryTypes"
  schemaLocation="ProductCategoryTypes.xsd"/>

 <complexType name="ProductCategoryBG">
  <complexContent>
   <extension base="bo:BusinessGraph">
    <sequence>

     <element name="verb"
      minOccurs="0" maxOccurs="1">
      <simpleType>
       <restriction base="string">
        <enumeration value="Create">
         <annotation>
          <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <botm:VerbInfo
            xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
            <verbInfo>Metadata relating to Create</verbInfo>
           </botm:VerbInfo>
          </appinfo>
         </annotation>
        </enumeration>
        <enumeration value="Retrieve">
         <annotation>
          <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <botm:VerbInfo
            xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
            <verbInfo>Metadata relating to Retrieve</verbInfo>
           </botm:VerbInfo>
          </appinfo>
         </annotation>
        </enumeration>
       </restriction>
      </simpleType>
     </element>

     <element name="productCategory" type="pc:ProductCategory"
      minOccurs="0" maxOccurs="1"/>

    </sequence>
```

```
  </extension>
 </complexContent>
</complexType>

<element name="productCategoryBG" type="pcbg:ProductCategoryBG"/>

</schema>
```

비즈니스 오브젝트 서비스를 사용한 프로그래밍

비즈니스 오브젝트의 작성 및 조작을 용이하게 하기 위해, 비즈니스 오브젝트 프레임워크는 Java 서비스 세트를 제공하여 SDO 스펙을 확장합니다. 이 서비스는 com.ibm.websphere.bo라는 패키지의 파트입니다.

비즈니스 오브젝트 서비스의 간략한 설명입니다.

표 22. 비즈니스 오브젝트 서비스

서비스	설명
BOChangeSummary	비즈니스 그래프의 변경 요약 헤더를 관리하도록 SDO 변경 요약 인터페이스에 대한 확장사항을 제공합니다.
BOCopy	비즈니스 오브젝트 그래프를 포함하는 비즈니스 그래프나 비즈니스 오브젝트 그래프 복사를 용이하게 합니다.
BODataObject	SDO 데이터 오브젝트 인터페이스에 대한 확장사항을 제공합니다.
BOEquality	두 개의 비즈니스 그래프 또는 비즈니스 오브젝트가 동일한지 판별할 수 있는 기능을 제공합니다.
BOEventSummary	비즈니스 그래프 이벤트 요약 헤더의 컨텐츠를 관리하기 위한 인터페이스를 제공합니다.
BOFactory	비즈니스 그래프 또는 비즈니스 오브젝트를 작성하기 위한 성능을 제공합니다.
BOType	Class.forName()이 Java 클래스 이름에 대해 제공하는 것을 미러링하는 비즈니스 오브젝트 또는 비즈니스 그래프의 SDO 유형을 얻기 위한 메커니즘을 제공합니다.
BOTypeMetadata	BO 유형 메타데이터 패턴을 준수하고 이 패턴을 SDO DataObject 세트로 변환하는(그런 다음 예약 변환을 수행하는) 어노테이션 2진 대형 오브젝트(BLOB)를 가져오는 성능을 제공합니다.
BOXMLDocument/BOXMLSerializer	메모리에서 XML 문서를 작성 및 표시하고 XML 문서를 직렬화 및 직렬화 해제하기 위한 메커니즘을 제공합니다.

표 22. 비즈니스 오브젝트 서비스 (계속)

서비스	설명
BOInstanceValidator	해당되는 XSD 정의에 대해 비즈니스 오브젝트 인스턴스의 유효성을 검증하기 위한 기능을 제공합니다. 비즈니스 오브젝트는 다양한 양식으로 제시될 수 있습니다. 단순 비즈니스 오브젝트이거나 풍부해진 비즈니스 그래프 모델에 의해 랩핑될 수 있습니다. 특정 비즈니스 통합 시나리오에서, 비즈니스 오브젝트는 삭제된 변경 요약 섹션에 있습니다. 이 비즈니스 오브젝트는 다운스트림 비즈니스 논리를 구동합니다. 비즈니스 오브젝트의 정확성은 모든 경우에 확인되어야 합니다. BOInstanceValidator에 대해 지원되는 두 가지 스타일이 있습니다. • 명시적 프로그램 방식 유효성 검증: 프로그래밍 API 세트를 통해 비즈니스 오브젝트의 유효성을 검증하기 위한 시스템 서비스가 제공됩니다. • 내부 인터페이스 유효성 검증: 이 유효성 검증은 SCA 인터페이스 규정자를 통해 대상 인터페이스의 WebSphere Integration Developer를 통해 사용 가능/사용 불가능하게 됩니다.

XML 문서 유효성 검증

XML 문서 및 비즈니스 오브젝트는 유효성 검증 서비스를 사용하여 유효성이 검증될 수 있습니다.

또한 기타 서비스는 특정한 최소 표준을 요구하거나 런타임 예외를 처리합니다. 이 중 하나는 BOXMLSerializer입니다.

서비스 요청에 따라 XML 문서를 처리하기 전에 BOXMLSerializer를 사용하여 해당 문서를 유효성 검증할 수 있습니다. BOXMLSerializer는 XML 문서의 구조를 유효성 검증하여 다음 유형의 오류가 존재하는지 판별하십시오.

- 유효하지 않은 XML 문서(예: 특정 요소 태그가 누락되어 있는 문서).
- 잘 구성되지 않은 XML 문서(예: 누락된 닫기 태그를 포함하는 문서).
- 구문 분석 오류(예: 엔티티 선언의 오류)를 포함하는 문서.

BOXMLSerializer에서 오류가 발견되는 경우 문제점 세부사항과 함께 예외가 처리됩니다.

다음 서비스에 대한 XML 문서의 가져오기 및 내보내기에 대해 유효성 검증을 수행할 수 있습니다.

- HTTP
- JAXRPC 웹 서비스
- JAX-WS 웹 서비스

- JMS 서비스
- MQ 서비스

HTTP, JAXRPC 및 JAX-WS 서비스에 대한 BOXMLSerializer는 다음 방식으로 예외를 생성합니다.

- 가져오기 –
  1. SCA 컴포넌트가 서비스를 호출합니다.
  2. 서비스가 대상 URL을 호출합니다.
  3. 대상 URL이 유효하지 않은 XML 예외로 응답합니다.
  4. 런타임 예외 및 메시지와 함께 서비스가 실패합니다.
- 내보내기 –
  1. 서비스 클라이언트가 서비스 내보내기를 호출합니다.
  2. 서비스 클라이언트가 유효하지 않은 XML을 전송합니다.
  3. 서비스에 대한 내보내기가 실패하고 예외 및 메시지를 생성합니다.

JMS 및 MQ 메시징 서비스에 대한 예외는 다음 방식으로 생성됩니다.

- 가져오기 –
  1. 가져오기가 JMS 또는 MQ 서비스를 호출합니다.
  2. 서비스가 응답을 리턴합니다.
  3. 서비스가 유효하지 않은 XML 예외를 리턴합니다.
  4. 가져오기가 실패하고 메시지를 생성합니다.
- 내보내기 –
  1. MQ 또는 JMS 클라이언트가 내보내기를 호출합니다.
  2. 클라이언트가 유효하지 않은 XML을 전송합니다.
  3. 내보내기가 실패하고 예외 및 메시지를 생성합니다.

XML 유효성 검증 예외로 생성된 모든 메시지에 대한 로그를 볼 수 있습니다. 아래 예제는 BOXMLSerializer에 의해 유효성 검증된 부적절한 XML 코딩으로 생성된 메시지입니다.

- JAXWS 가져오기

```
javax.xml.ws.WebServiceException: org.apache.axiom.om.OMException:
 javax.xml.stream.XMLStreamException: Element type "TestResponse" must be
 followed by either attribute specifications, ">" or "/>".

javax.xml.ws.WebServiceException: org.apache.axiom.soap.SOAPProcessingException:
 First Element must contain the local name, Envelope
```

- JAXRPC 가져오기

```
[9/11/08 15:16:27:417 CDT] 0000003e ExceptionUtil E
 CNTR0020E: EJB threw an unexpected (non-declared)
  exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
```

```
 Exception data: WebServicesFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
faultString: org.xml.sax.SAXParseException: Element type "TestResponse"
  must be followed by either
 attribute specifications, ">" or "/>". Message being parsed:
   <?xml version="1.0"?><TestResponse
 xmlns="http://WSXMLValidation"<firstName>Bob</firstName>
   <lastName>Smith</lastName></TestResponse>
faultActor: null
faultDetail:
[9/11/08 15:16:35:135 CDT] 0000003f ExceptionUtil E   CNTR0020E: EJB threw an
  unexpected (non-declared) exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
 Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXException: WSWS3066E: Error: Expected 'envelope'
   but found  TestResponse
  Message being parsed: <?xml version="1.0"?><TestResponse
   xmlns="http://WSXMLValidation">
  <firstName>Bob</firstName><middleName>John</middleName>
   <lastName>Smith</lastName>
   </TestResponse>
 faultActor: null
 faultDetail:
```

- JAXRPC/JAXWS 내보내기

```
[9/11/08 15:35:13:401 CDT] 00000064 WebServicesSe E
  com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
  getSoapAction WSWS3112E:
   Error: Generating WebServicesFault due to missing SOAPAction.
                    WebServicesFault
 faultCode: Client.NoSOAPAction
 faultString: WSWS3147E: Error: no SOAPAction header!
 faultActor: null
 faultDetail:
```

유효성 검증 서비스에 대한 자세한 정보는 참조 절에 생성된 API 및 SPI 문서에서 BOInstanceValidator 인터페이스를 참조하십시오.

프로그래밍 기술

이 기술은 비즈니스 오브젝트 프레임워크를 사용하여 비즈니스 오브젝트를 효율적으로 프로그래밍하는 방법을 보여줍니다.

비즈니스 오브젝트의 배열

요소가 데이터 인스턴스를 두 개 이상 포함할 수 있도록 비즈니스 오브젝트에서 요소의 배열을 정의할 수 있습니다.

목록 유형을 사용하여 비즈니스 오브젝트에 이름 지정된 단일 요소의 배열을 작성할 수 있습니다. 이 경우 해당 요소를 사용하여 데이터의 다중 인스턴스를 포함할 수 있습니다. 예를 들어, 배열을 사용하여 비즈니스 오브젝트 랩퍼에 문자열로 정의되어 있는 telephone이라는 요소 안에 여러 개의 전화 번호를 저장할 수 있습니다. 또한 maxOccurs 값을 사용하는 데이터 인스턴스 수를 지정하여 배열의 *크기*를 정의할 수 있습니다. 다음 예제 코드는 해당 요소의 데이터에 대해 세 개의 인스턴스를 보유하는 배열의 작성 방법을 표시합니다.

```
<xsd:element name="telephone" type="xsd:string" maxOccurs="3"/>
```

이 결과로 최대 세 개의 데이터 인스턴스를 보유할 수 있는 요소 telephone의 목록 색인이 작성됩니다. 또한 배열에 항목을 하나만 포함하려는 경우 값 minOccurs를 사용할 수 있습니다.

결과 배열은 다음 두 항목으로 구성됩니다.

- 배열의 컨텐츠
- 배열 자체.

하지만 이 배열을 작성하기 위해서는 랩퍼를 정의하여 중간 단계를 수행해야 합니다. 이 랩퍼는 사실상 요소의 특성을 배열 오브젝트로 바꿉니다. 위 예제에서 ArrayOfTelephone 오브젝트를 작성하여 요소 telephone을 배열로 정의할 수 있습니다. 다음 코드 예제는 이 타스크를 수행하는 방법을 보여줍니다.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="Customer">
   <xsd:complexType>
    <xsd:sequence>
 <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="ArrayOfTelephone" type="ArrayOfTelephone"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>

    <xsd:complexType name="ArrayOfTelephone">
 <xsd:sequence maxOccurs="3">
        <xsd:element name="telephone" type="xsd:string" nillable="true"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

telephone 요소는 이제 ArrayOfTelephone 랩퍼 오브젝트의 하위로 표시됩니다.

위 예제에서 telephone 요소가 nillable이라는 특성을 포함한다는 점에 유의하십시오. 배열 색인의 특정 항목에 데이터를 포함하지 않으려면 이 특성을 true로 설정할 수 있습니다. 다음 예제 코드는 배열에 데이터를 나타낼 수 있는 방법을 표시합니다.

```
<Customer>
   <name>Bob</name>
   <ArrayOfTelephone>
     <telephone>111-1111</telephone>
     <telephone xsi:nil="true"/>
     <telephone>333-3333</telephone>
   </ArrayOfTelephone>
 </Customer>
```

이 경우, telephone 요소의 배열 색인에서 첫 번째 및 세 번째 항목은 데이터를 포함하고 두 번째 항목은 데이터를 포함하지 않습니다. telephone 요소에 대해 nillable 특성을 사용하지 않은 경우 처음 두 요소에 데이터가 포함되어 있어야 합니다.

비즈니스 오브젝트 배열의 순서를 처리할 대체 메소드로 WebSphere Process Server에 서비스 데이터 오브젝트(SDO) 순서 API를 사용할 수 있습니다. 다음 예제 코드는 위에 표시된 것과 동일한 데이터로 telephone 요소의 배열을 작성합니다.

```
DataObject customer = ...
customer.setString("name", "Bob");

DataObject tele_array = customer.createDataObject("ArrayOfTelephone");
Sequence seq = tele_array.getSequence();  // The array is sequenced
seq.add("telephone", "111-1111");
seq.add("telephone", null);
seq.add("telephone", "333-3333");
```

아래 예제와 유사한 코드를 사용하여 주어진 요소 배열 색인의 데이터를 리턴할 수 있습니다.

```
String tele3 = tele_array.get("telephone[3]");  // tele3 = "333-3333"
```

이 예제에서 tele3이라는 문자열은 데이터 "333-3333"을 리턴합니다.

JMS 또는 MQ 메시지 큐에 놓인 구분된 데이터 또는 고정 너비를 사용하여 목록 색인에 배열의 데이터 항목을 입력할 수 있습니다. 또한 올바르게 형식화된 데이터를 포함하는 일반 텍스트 파일을 사용하여 이 타스크를 수행할 수 있습니다.

중첩 비즈니스 오브젝트 작성

setWithCreate 기능을 사용하여 상위 비즈니스 오브젝트 내에서 중첩 비즈니스 오브젝트를 작성할 수 있습니다.

중간 하위 오브젝트를 자세히 설명하는 코드를 작성하지 않아도 상위 비즈니스 오브젝트에서 중첩 비즈니스 오브젝트를 작성할 수 있습니다. 예를 들어, 상위 오브젝트의 한 레벨 아래에 종속 비즈니스를 정의하지 않아도 해당 상위 오브젝트의 두 레벨 아래에 중첩 비즈니스 오브젝트를 설정할 수 있습니다. setWithCreate 기능을 사용하여 다음에 대해 이 타스크를 수행하십시오.

- 단일 인스턴스
- 다중 인스턴스
- 와일드카드 값
- 모델 그룹

다음 주제에서는 각각을 수행하는 방법에 대해 설명합니다.

중첩 비즈니스 오브젝트의 단일 인스턴스:

setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 단일 인스턴스를 작성하십시오.

시작하기 전에

아래의 예제 코드는 써드 레벨(하위의 하위) 오브젝트를 작성하기 위해 상위 레벨(상위) 오브젝트에서 중간(하위) 오브젝트의 코드를 정상적으로 작성하는 방법을 표시합니다. XSD 파일은 다음과 같이 표시됩니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

이 태스크 정보

전형적인 "하향식" 메소드를 사용하여 비즈니스 오브젝트 데이터를 설정한 경우 하위의 하위 오브젝트에 데이터를 설정하기 전에 하위 및 하위의 하위 오브젝트를 지정하는 다음 코드를 처리해야 합니다.

```
DataObject parent = ...
DataObject child = parent.createDataObject("child");
DataObject grandchild = child.createDataObject("grandChild");
grandchild.setString("name", "Bob");
```

setWithCreate 기능과 함께 더 효율적인 메소드를 사용하여 중간 하위 오브젝트를 지정하지 않아도 하위의 하위 오브젝트를 정의하고 해당 데이터를 설정하는 작업을 동시에 수행할 수 있습니다. 다음 예제 코드는 이 타스크를 수행하는 방법을 보여줍니다.

```
DataObject parent = ...
parent.setString("child/grandchild/name", "Bob");
```

결과

하위 레벨 비즈니스 오브젝트 데이터는 중간 레벨 비즈니스 오브젝트를 참조하지 않아도 설정됩니다. 경로가 유효하지 않은 경우 예외가 발생합니다.

중첩 비즈니스 오브젝트의 다중 인스턴스 작성:

setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 다중 인스턴스를 작성하십시오.

시작하기 전에

아래 예제 XSD 파일은 맨 위(상위) 비즈니스 오브젝트의 한 레벨(하위) 및 두 레벨(하위의 하위) 아래에 중첩 오브젝트를 포함합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child" maxOccurs="5"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

상위 오브젝트는 maxOccurs 값에 지정된 대로 최대 5개의 하위 오브젝트를 포함할 수 있습니다.

이 태스크 정보

배열의 누락된 순서에 대해 허용하지 않는 더 엄격한 정책에 따라 목록을 작성할 수 있습니다. setWithGet 메소드를 사용함과 동시에 특정 목록 색인 항목에 표시되는 데이터를 지정할 수 있습니다.

```
DataObject parent = ...
parent.setString("child[3]/grandchild/name", "Bob");
```

이 경우 결과 배열의 *크기*는 3이지만 child[1] 및 child[2] 목록 색인 항목의 값은 정의되지 않습니다. 항목은 널값이 되거나 연관된 데이터 값을 가질 수도 있습니다. 위 시나리오에서는 처음 두 개의 배열 색인 항목 값이 정의되지 않아서 예외가 처리됩니다.

목록의 색인에 값을 정의하여 이 상황을 개선할 수 있습니다. 색인 항목이 배열의 기존 요소를 참조하는 경우와 해당 요소가 널(null)이 아닌 경우(즉, 데이터가 포함됨) 사용됩니다. 널이면 요소가 작성되고 사용됩니다. 목록의 색인이 목록의 *크기*보다 더 큰 값이면 새 값이 작성되고 추가됩니다. 다음 예제 코드는 *크기*가 2인 목록에서 발생하는 상황을 보여줍니다. 여기서, child[1]은 널로 지정되어 있고 child[2]는 데이터를 포함합니다.

```
DataObject parent = ...
// child[1] = null
// child[2] = existing Child
// This code will work because child[1] is null and will be created.
parent.setString("child[1]/grandchild/name", "Bob");

// This code will work because child[2] exists and will be used.
parent.setString("child[2]/grandchild/name", "Dan");

// This code will work because the child list is of size 2, and adding
// one more list item will increase the list size.
parent.setString("child[3]/grandchild/name", "Sam");
```

결과

두 개의 *기존* 항목 값을 대체하고 목록 색인에 세 번째 항목을 추가했습니다. 그러나 *크기*가 4가 아니거나 maxOccurs에 지정된 *크기*보다 큰 또 다른 항목을 추가하는 경우 예외가 처리됩니다. 이 메소드의 더 엄격한 정책은 다음 예제 코드에서 설명합니다.

주: 아래 코드는 위의 기존 코드에 추가되는 것으로 가정합니다.

```
// This code will throw an exception because the list is of size 3
// and you have not created an item to increase the size to 4.
parent.setString("child[5]/grandchild/name", "Billy");
```

와일드카드에 정의된 중첩 비즈니스 오브젝트 사용:

상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

이 태스크 정보

단일 또는 다중 인스턴스에 대해 중첩 비즈니스 오브젝트를 정의하는 데 사용된 setWithCreate 기능은 서비스 데이터 오브젝트에 와일드카드 값 xsd:any를 사용할 경우에 작동하지 않습니다. 이에 대해서는 다음 예제 코드에서 설명합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="xsd:anyType"/>
```

```
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

결과

하위 데이터 오브젝트가 존재하지 않는 경우 예외가 처리됩니다.

모델 그룹의 비즈니스 오브젝트 사용:

모델 그룹의 파트인 중첩 비즈니스 오브젝트에 대해 작업할 때 모델 그룹 경로 패턴을 작성합니다.

이 태스크 정보

모델 그룹은 상위 비즈니스 오브젝트에서 비즈니스 오브젝트를 작성하기 위해 사용할 수 있는 xsd:choice 태그를 사용합니다. 그러나 비즈니스 오브젝트 프레임워크는 예외를 생성할 수 있는 네이밍 충돌을 야기할 수 있습니다. 다음 예제 코드는 네이밍 충돌이 발생하는 방식을 설명합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

주: "child1" 및 "child2"라는 요소의 다중 인스턴스가 있을 수 있습니다.

모델 그룹에 대한 SDO(Service Data Object) 경로 패턴을 사용하여 이러한 충돌을 해결하십시오.

결과

다음 예제 코드에 표시된 대로 모델 그룹을 처리하는 데 사용되는 SDO 경로 패턴을 사용하는 배열을 가져올 수 있습니다.

```
set("child1/grandchild/name", "Bob");

set("child11/grandchild/name", "Joe");
```

동일하게 이름 지정된 요소 구별

비즈니스 오브젝트 요소 및 속성에 고유한 이름을 지정해야 합니다.

서비스 데이터 오브젝트(SDO) 프레임워크에서 요소 및 속성은 특성으로 작성됩니다. 다음 코드 예제에서는 XSD가 foo로 이름 지정된 하나의 특성이 있는 유형을 작성합니다.

```
<xsd:complexType name="ElementFoo">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeFoo">
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

이런 경우 XML 경로 언어(XPath)를 사용하여 특성에 액세스할 수 있습니다. 그러나 다음 예제에서와 같이 유효한 스키마 유형에 동일한 이름을 가진 속성 및 요소가 있을 수 있습니다.

```
<xsd:complexType name="DuplicateNames">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

XPath에서 동일한 이름이 지정된 요소를 속성과 구별할 수 있어야 합니다. 이름 중 하나를 앳 부호(@)로 시작하여 이를 구별합니다. 다음 스니펫은 동일하게 이름이 지정된 요소 및 속성에 액세스하는 방법을 보여 줍니다.

```
1 DataObject duplicateNames = ...

2 // Displays "elem_value"
3 System.out.println(duplicateNames.get("foo"));

4 // Displays "attr_value"
5 System.out.println(duplicateNames.get("@foo"));
```

SDO XPath인 문자열 값을 취하는 모든 메소드에 이 네이밍 설계를 사용하십시오.

모델 그룹 지원(모두, 선택사항, 순서 및 그룹 참조):

SDO 스펙을 적소에 펼치고 유형 또는 특성에 대해 설명하지 않으려면 모델 그룹(모두, 선택사항, 순서 및 그룹 참조)이 필요합니다.

기본적으로 이는 동일한 포함 구조에 있는 모든 해당 구조가 "단일화"됨을 의미합니다. 이 "단일화"는 모든 하위 구조를 동일한 레벨에 놓습니다. 이로 인해 단일화된 데이터

에서 구조가 파생된 SDO에 중복 네이밍 문제가 발생할 수 있습니다. XSD에서 그룹을 단일화하지 않는 경우 여전히 서로 다른 상위 구조에 중복되는 항목이 각각 포함되어 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

option1 및 option2의 다중 어커런스가 서로 다른 선택사항 블록에 들어 있고 이들 사이에 독립적인 요소도 있으므로 XSD 및 XML에서 이를 구별하는 데 문제가 없습니다. 그러나 SDO에서 이 그룹을 단일화하면 모든 옵션 특성이 동일한 MultipleGroup 컨테이너에 있게 됩니다.

중복된 이름이 없더라도 이들 그룹을 단일화함으로써 발생하는 시맨틱 문제도 있습니다(예: XSD).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://SimpleChoice">
  <xsd:complexType name="SimpleChoice">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

표준 및 산업 스키마와 같은 여러 경우에 사용자가 작업 중인 XSD를 제어하지는 않으므로 사용자에게 중복된 이름을 바꾸거나 XSD에 특수한 어노테이션을 추가하도록 요청하는 것은 바람직하지 않습니다.

모든 특성에 일관성이 있도록 하기 위해 비즈니스 오브젝트에는 XPath를 통해 중복된 이름을 가진 특성의 개별 어커런스에 액세스할 수 있는 메소드가 포함되어 있습니다. 비즈니스 오브젝트 프레임워크 이름 지정 규칙에 따라, 발견된 모든 중복 특성 이름은 사용되지 않은 다음 숫자가 해당 이름에 추가되도록 합니다. 예를 들어, XSD는 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://TieredGroup">
  <xsd:complexType name="TieredGroup">
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:sequence>
          <xsd:element name="low" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:choice minOccurs="0">
            <xsd:element name="width" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
        <xsd:element name="high" minOccurs="1"
    maxOccurs="1" type="xsd:string"/>
          <xsd:sequence>
            <xsd:element name="width" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="center" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="width" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

위 XSD는 다음 DataObject 모델을 생성합니다.

```
DataObject - TieredGroup
 Property[0] - low - string
 Property[1] - width - string
 Property[2] - high - string
 Property[3] - high1 - string
 Property[4] - width1 - string
 Property[5] - high2 - string
 Property[6] - center - string
 Property[7] - width2 - string
```

여기서 **width**, **width1** 및 **width2**는 **high**, **high1** 및 **high2**와 마찬가지로 첫 번째 width에서 시작하여 XSD에서 아래 방향으로 이름 지정된 특성 width의 이름입니다.

이러한 새 특성 이름은 참조 및 XPath로만 사용되는 이름이며 직렬화된 컨텐츠에는 영향을 주지 않습니다. 직렬화된 XML에 표시되는 이들 각 특성의 "참" 이름은 XSD에 주어진 값입니다. XML 인스턴스는 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:TieredGroup xsi:type="p:TieredGroup"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xmlns:p="http://TieredGroup">
  <width>foo</width>
  <high>bar</high>
</p:TieredGroup>
```

이러한 특성에 액세스하기 위해 다음 코드를 사용할 수 있습니다.

```
DataObject tieredGroup = ...

// Displays "foo"
System.out.println(tieredGroup.get("width1"));

// Displays "bar"
System.out.println(tieredGroup.get("high2"));
```

동일하게 이름 지정된 특성 구별

동일한 네임스페이스를 갖는 다중 XSD가 동일하게 이름 지정된 유형을 정의하는 경우 잘못된 유형을 우연히 참조할 수 있습니다.

Address1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="city" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

Address2.xsd:
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="state" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

비즈니스 오브젝트는 BOFactory.create() API를 통해 어떤 글로벌 XSD 구조(예: complexType, simpleType, 요소, 속성 등)에도 중복된 이름을 지원하지 않습니다. 다음 예제에서와 같이 올바른 API를 사용하면 다른 구조의 하위 구조로 중복 글로벌 구조를 작성할 수 있습니다.

Customer1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer1"
 targetNamespace="http://Customer1">
  <xsd:import schemaLocation="./Address1.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
```

```
      <xsd:element minOccurs="0" name="address" type="Address"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Customer2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer2"
 targetNamespace="http://Customer2">
  <xsd:import schemaLocation="./Address2.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

고객 주소 필드를 둘 다 채우고 BOFactory.create()를 호출하여 주소를 작성하는 경우 생성되는 하위 비즈니스 오브젝트 유형이 잘못 설정될 수 있습니다. 고객 DataObject에 대한 createDataObject("address") API를 호출하여 이를 방지할 수 있습니다. 비즈니스 오브젝트는 가져오기의 schemaLocation을 따르므로 올바른 유형의 하위 유형을 생성하도록 보장할 수 있습니다.

```
DataObject customer1 = ...

 // Incorrect way to create Address child
 // This may create a type of Address1.xsd Address

 or maybe Address2.xsd Address
 DataObject incorrect = boFactory.create("", "Address");
 customer1.set("address", incorrect);

 // Correct way to create Address child
 // This is guaranteed to create a type of Address1.xsd Address
 customer1.createDataObject("address");
```

마침표를 포함하는 특성 이름 해석

XSD의 특성 이름에 마침표(".")가 있을 수 있습니다. 이는 여러 유효 문자 중 하나로서 SDO에서는 다중 카디널리티의 특성에서 색인화를 표시하는 데도 사용됩니다. 마침표는 특정 상황에서 해석 문제를 발생시킬 수 있습니다.

서비스 데이터 오브젝트(SDO)의 특성 이름은 XSD에서 생성된 요소 및 속성 이름을 기본으로 합니다. 비즈니스 오브젝트에서는 마침표(".") 문자를 적절히 처리할 수 있으나 예외인 경우가 하나 있습니다. XSD에 "<name>.<#>"으로 이름 지정된 단일 카디널리티 특성 및 "<name>"으로 이름 지정된 다중 카디널리티 특성이 있는 경우입니다.

"foo.0"으로 이름 지정된 단일 카디널리티 특성 및 "foo"로 이름 지정된 다중 카디널리티 특성이 있는 경우 "foo.0"과 같은 XPath에서는 특성을 올바로 해석할 수 없습니다.

이런 경우 "foo.0"이라는 단일 카디널리티 특성이 해석됩니다. 이러한 어커런스가 드물기는 하지만 "foo[1]" 구문을 사용하여 다중 카디널리티 특성에 액세스하면 이러한 어커런스가 전혀 발생하지 않습니다. SDO에서는 색인화에 마침표(".") 구문을 지원하지 않으므로 색인화하려면 대괄호("[]")를 사용해야 합니다.

xsi:type을 사용하여 유니온 직렬화 및 직렬화 해제:

XSD에서 유니온은 구성원으로 알려진 여러 단순 데이터 유형의 렉시칼 영역을 병합하는 방법입니다.

다음 XSD 예제는 정수 및 날짜 구성원을 가진 유니온을 표시합니다.

```
<xsd:simpleType name="integerOrDate">
 <xsd:union memberTypes="xsd:integer xsd:date"/>
</xsd:simpleType>
```

이러한 다중 입력은 직렬화 해제 중 및 데이터 조작 시 혼란을 일으킬 수 있습니다.

비즈니스 오브젝트는 SDO가 직렬화 시 xsi:type을 사용하도록 지원하며 XML 데이터에 xsi:type이 없는 경우에는 직렬화 해제 시 유형을 판별하는 데 사용되는 것과 동일한 알고리즘을 따릅니다.

데이터(다음 예제의 숫자 "42")가 정수로 직렬화 해제되도록 하기 위해 입력 XML에 지정된 xsi:type을 사용할 수 있습니다. 문자열 앞에 정수가 위치하도록 XSD의 유니온 구성원 목록의 순서를 지정할 수도 있습니다. 다음 예는 두 가지 메소드를 구현하는 방법을 표시합니다.

```
<integerOrString xsi:type="xsd:integer">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:integer xsd:string"/>
</xsd:simpleType>
```

이와 마찬가지로 데이터를 문자열로 직렬화 해제하려는 경우에는 다음과 같이 변경하면 됩니다.

```
<integerOrString xsi:type="xsd:string">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:string xsd:integer"/>
</xsd:simpleType>
```

유니온의 첫 번째 구성원이 문자열 유형인 경우에는 정보가 전혀 유실되지 않습니다. xsi:type이 아닌 알고리즘으로 항상 선택되는 모든 데이터도 보존할 수 있습니다. 문자열 이외의 유형을 사용하려면 XML에서 xsi:type을 사용하거나 XSD에서 구성원 유형을 다시 정렬하여 다른 구성원이 데이터를 승인할 수 있게 해야 합니다.

널 비즈니스 오브젝트에 대한 지원

이 시나리오에는 SOAP 메시지 안에서 랩핑된 XML을 통해 WebSphere Process Server와 통신하여 외부 시스템이 관련됩니다. 둘러싸인 요소가 nillible이고 `xsi:nil="true"`를 보유하는 경우 WebSphere Process Server에서 작성되는 결과 DataObject는 널(null)입니다.

다음은 nillible 요소가 있는 XML 메시지를 보여주는 예제입니다.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body>
  <p:Employee xmlns:p="http://www.mycompany.com"

  xmlns:xsi="http://www.w3.org"
   xsi:nil="true"/>
 </soap:Body>
</soap:Envelope>
```

여기서 `Employee`는 다음과 같이 정의됩니다.

```
<element name="Employee" nillable="true">
...
</element>
```

생성되고 내보내기에서 송신되는 비즈니스 오브젝트는 이 경우에 널(null)입니다. 예를 들어, 다운스트림 컴포넌트에 호출되는 조작이 있는 경우 해당 조작의 입력은 널입니다.

주: 비즈니스 오브젝트 맵은 널 오브젝트의 필드를 맵핑할 수 없으므로 이 오브젝트는 비즈니스 오브젝트 맵으로 전달될 수 없습니다.

순서 오브젝트를 사용하여 데이터 순서 설정

일부 XSD를 정의하는 방법으로 인해 XML에서 발생하는 데이터 순서에 특수한 중요성이 부여됩니다.

XSD에서의 순서 중요성에 대한 예로는 혼합 컨텐츠를 들 수 있습니다. 한 요소의 이전 또는 이후에 텍스트 데이터가 표시되는 경우 다른 위치에 표시되는 경우와는 다른 의미를 가질 수 있습니다. 이런 경우 SDO가 순서로 알려진 오브젝트를 생성하며 이는 순서를 지정하여 데이터를 설정하는 데 사용됩니다.

SDO 순서를 XSD 순서와 혼동해서는 안 됩니다. XSD 순서는 SDO 모델을 생성하기 전에 단일화되는 모델 그룹입니다. XSD 순서 표시는 SDO 순서 표시와 관련이 없습니다.

XSD의 다음 조건은 SDO 순서를 생성합니다.

혼합 컨텐츠가 포함된 complexType:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://MixedContent"
 targetNamespace="http://MixedContent">
  <xsd:complexType name="MixedContent" mixed="true">
    <xsd:sequence>
      <xsd:element name="element1" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element2" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element3" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MixedContent" type="tns:MixedContent"/>
</xsd:schema>
```

<any/> 태그가 하나 이상 있는 스키마:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <xsd:any/>
      <xsd:element name="marker1" type="xsd:string"/>
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

모델 그룹 배열(maxOccurs > 1인 모두, 선택사항, 순서 및 그룹 참조):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence maxOccurs="3">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

둘 이상의 요소가 포함된 maxOccurs <= 1인 <all/> 태그:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://All">
  <xsd:complexType name="All">
    <xsd:all>
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>
```

<any/>와 순서를 함께 사용하는 것에 대한 특정 정보는 이 페이지 맨 아래 나열되어 있는 주제에서 설명합니다. 이 절의 나머지 부분에 표시된 일반 정보에서는 기타 순서 조건에서 작업하는 방법에 대해 설명하며, 이는 여전히 <any/>에도 적용됩니다.

내 DataObject에 순서가 있는지 확인하는 방법:

DataObject에 순서가 있는지 판별하기 위해 선택할 수 있는 두 가지 간단한 API가 있습니다. DataObject noSequence 및 DataObject withSequence입니다.

다음 예제에서와 같이 DataObject noSequence 및 DataObject withSequence를 사용할 수 있습니다.

```
DataObject noSequence = ...
DataObject withSequence = ...

// Displays false
System.out.println(noSequence.getType().isSequenced());

// Displays true
System.out.println(withSequence.getType().isSequenced());

// Displays true
System.out.println(noSequence.getSequence() == null);

// Displays false
System.out.println(withSequence.getSequence() == null);
```

DataObject에 순서가 있는지 파악해야 하는 이유:

순서가 있는 DataObject에 대해 작업 중인 경우 데이터가 설정된 순서를 알아야 합니다. 이 때문에 값이 설정된 순서에 주의를 기울여야 합니다.

순서가 지정되지 않은 DataObject에서는 임의의 순서로 설정에 액세스할 수 있습니다. 이는 모든 키가 동일한 값으로 설정된 맵과 같이 작동합니다. 키가 설정된 순서는 중요하지 않으며 맵의 데이터는 서로 같고 XML에 동일하게 직렬화됩니다.

DataObject에 순서가 지정된 경우 목록에 데이터를 추가하는 것과 마찬가지로 데이터가 설정된 순서가 순서(Sequence)에 레코드됩니다. 이로 인해 이름/값 쌍별(DataObject API) 및 데이터가 설정된 순서별(Sequence API)의 두 가지 방법으로 데이터에 액세스할 수 있습니다. DataObject set(...) 또는 Sequence add(...) API를 사용하여 구조를 유지할 수 있습니다. 이러한 순서 지정은 XML을 직렬화하는 방법에 영향을 미칩니다.

예를 들면, 아래 <all/> 태그 XSD와 같습니다. set 메소드를 다음 순서로 호출하면 직렬화 시 다음과 같은 XML이 생성됩니다.

```
DataObject all = ...
all.set("element1", "foo");
all.set("element2", "bar");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
```

```
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element1>foo</element1>
  <element2>bar</element2>
</p:All>
```

이와 달리 set 메소드를 반대 순서로 호출하면 비즈니스 오브젝트 직렬화 시 다음과 같은 XML이 생성됩니다.

```
 DataObject all = ...
 all.set("element2", "bar");
 all.set("element1", "foo");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element2>bar</element2>
  <element1>foo</element1>
</p:All>
```

순서(Sequence)의 순서를 변경해야 하는 경우에는 순서 클래스에 있는 기본 add, remove 및 move 메소드를 사용하여 사용자가 순서를 변경할 수 있습니다.

혼합 컨텐츠에 대한 작업 방법:

혼합 컨텐츠의 경우 순서에 텍스트를 추가하는 데 사용되는 특정 API(addText(...))가 있습니다.

기타 모든 API에서는 특성에 대한 작업과 동일하게 텍스트에 대한 작업이 이루어집니다. getProperty(int) API는 혼합 컨텐츠 텍스트 데이터에 널을 리턴합니다. 다음 혼합 컨텐츠 코드 예제를 사용하여 DataObject에서 모든 혼합 컨텐츠 텍스트를 인쇄할 수 있습니다.

```
 DataObject mixedContent = ...
 Sequence seq = mixedContent.getSequence();

 for (int i=0; i < seq.size(); i++)
 {
     Property prop = seq.getProperty(i);
     Object value = seq.getValue(i);

     if (prop == null)
     {
         System.out.println("Found mixed content text: "+value);
     }
     else
     {
         System.out.println("Found Property "+prop.getName()+": "+value);
     }
 }
```

모델 그룹 배열에 대한 작업 방법:

모델 그룹에 maxOccurs > 1인 값이 있는 경우 모델 그룹 배열이 작성됩니다.

모델 그룹이 단일화되고 DataObject에 표시되지 않으므로 모델 그룹 내부의 특성이 아직 참이 아닌 경우 해당 isMany() 메소드가 참을 리턴하도록 모델 그룹 내부의 특성은 다중 카디낼리티 특성이 됩니다. 해당 minOccurs 및 maxOccurs 패싯은 포함된 모델 그룹의 패싯 만큼 배가됩니다. 선택사항은 기타 모델 그룹과 동일한 방법으로 maxOccurs 패싯을 배가시키지만, 선택사항의 어떤 데이터도 선택되지 않기 때문에 minOccurs의 곱셈 값으로 항상 0을 사용합니다.

예를 들어, 다음 XSD에 모델 그룹 배열이 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

설명한 바와 같이 get(...) 액세서가 목록을 리턴할 수 있도록 **element1** 및 **element2**가 다중 카디낼리티가 됩니다. **Element1**에는 minOccurs 기본값 1 및 maxOccurs 기본값 1이 있습니다. **Element2**에는 minOccurs 값 0 및 maxOccurs 값 3이 있습니다. 다음 예제에서 해당 새 minOccurs 및 maxOccurs는 다음과 같습니다.

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(2*1)=2 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(2*0)=0 - maxOccurs=(5*3)=15
```

유형이 선택사항인 경우에는 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:choice minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

위 예제에서는 매번 **element1**만 선택하거나 매번 **element2**만 선택할 수 있는 선택사항이 배제되었기 때문에 다음과 같은 minOccurs를 생성하여 둘 다 0개의 어커런스를 가질 수 있도록 하는 유효성 검증에 패스할 수 있게 합니다.

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(0*1)=0 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(0*0)=0 - maxOccurs=(5*3)=15
```

데이터 유형 사용

이 절은 데이터 유형 사용에 대한 프로그래밍 기술을 제공합니다.

단순 유형에 대해 AnySimpleType 사용:

AnySimpleType은 SDO API에서 기타 다른 단순 유형(문자열, 정수, 부울 등)과 유사하게 처리됩니다.

anySimpleType과 기타 단순 유형 간의 유일한 차이점은 해당 인스턴스 데이터 및 직렬화/직렬화 해제뿐입니다. 이는 비즈니스 오브젝트에만 해당되는 내부 개념이며 데이터와 필드 간의 맵핑 방향이 유효한지 여부를 판별하는 데 사용됩니다. 문자열 유형에 대해 set(...) 메소드가 호출된 경우 데이터는 제일 먼저 문자열로 변환되고 원래 데이터 유형은 유실됩니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://StringType">
  <xsd:complexType name="StringType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject stringType = ...

 // Set the data to a String
 stringType.set("foo", "bar");

 // The instance data will always be type String,

 regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Set the data to an Integer
 stringType.set("foo", new Integer(42));

 // The instance data will always be type String,

 regardless of the data set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());
```

대신 anySimpleType은 다음과 같이 설정되어 있는 원래의 데이터 유형을 느슨하게 만들지 않습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnySimpleType">
  <xsd:complexType name="AnySimpleType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:anySimpleType"/>
   </xsd:sequence>
```

```
  </xsd:complexType>
</xsd:schema>

 DataObject anySimpleType = ...

 // Set the data to a String
 stringType.set("foo", "bar");

 // The instance data will always be of the type of date used in the set
 // Displays "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Set the data to an Integer
 stringType.set("foo", new Integer(42));

 // The instance data will always be of the type of date used in the set
 // Displays "java.lang.Integer"
 System.out.println(stringType.get("foo").getClass().getName());
```

이 데이터 유형은 또한 직렬화 및 직렬화 해제 동안에도 xsi:type으로 유지됩니다. 따라서 anySimpleType 요소를 직렬화할 때마다 이 요소에는 해당 Java 유형을 기본으로 SDO 스펙에 정의된 것과 일치하는 xsi:type이 있습니다.

다음 예제에서는 데이터가 다음과 같이 표시되도록 위의 비즈니스 오브젝트를 직렬화합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:StringType xsi:type="p:StringType"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:p="http://StringType">
  <foo xsi:type="xsd:int">42</foo>
<p:StringType></p:StringType>
```

직렬화 해제 시 xsi:type을 사용하여 데이터를 적합한 Java 인스턴스 클래스로 로드합니다. xsi:type이 지정되지 않은 경우 기본 직렬화 해제 유형은 문자열이 됩니다.

기타 단순 유형의 경우 맵핑 가능성을 판별하는 것은 상수입니다. 예를 들어, 부울은 항상 문자열에 맵핑할 수 있습니다. AnySimpleType에는 모든 단순 유형이 포함될 수 있으나, 이 때문에 필드의 인스턴스 데이터에 따라 맵핑할 수 있는지 여부가 결정됩니다.

특성 유형의 URI 및 이름을 사용하여 특성 유형이 anySimpleType인지 판별할 수 있습니다. URI는 "commonj.sdo"이고 이름은 "Object"입니다. 데이터가 anySimpleType에 삽입할 수 있는 유효한 데이터인지 판별하려면 DataObject의 인스턴스가 아닌지 확인하십시오. 문자열로 표시할 수 있고 DataObject가 아닌 모든 데이터를 anySimpleType 필드에 설정할 수 있습니다.

이로 인해 다음과 같은 맵핑 규칙이 적용됩니다.

- anySimpleType은 항상 anySimpleType에 맵핑할 수 있습니다.
- 기타 모든 단순 유형은 항상 anySimpleType에 맵핑할 수 있습니다.

- 모든 단순 유형을 문자열로 변환할 수 있어야 하므로 anySimpleType은 항상 문자열에 맵핑할 수 있습니다.
- anySimpleType은 비즈니스 오브젝트의 해당 값에 따라 기타 단순 유형에 맵핑할 수 있습니다. 이는 해당 맵핑을 설계 시 판별할 수 없으며 런타임 시에만 판별할 수 있음을 의미합니다.

관련 정보


xs:any에서의 지정 및 xs:any에 대한 지정

복합 유형에 대해 AnyType 사용:

anyType 태그는 SDO API에서 기타 다른 복합 유형과 비슷하게 처리됩니다.

anyType과 기타 복합 유형 간의 유일한 차이점은 인스턴스 데이터 및 직렬화/직렬화 해제(비즈니스 오브젝트에만 해당되는 내부 개념), 그리고 데이터와 필드 간의 맵핑 방향이 유효한지 여부를 판별하는 데만 있습니다. 복합 유형에는 고객, 주소 등의 단일 유형만 포함됩니다. 그러나 anyType에는 유형에 관계 없이 모든 DataObject가 포함됩니다. maxOccurs > 1인 경우 목록의 각 DataObject 유형이 다를 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnyType">
  <xsd:complexType name="AnyType">
    <xsd:sequence>
      <xsd:element name="person" type="xsd:anyType"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://Customer">
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Employee" targetNamespace="http://Employee">
  <xsd:complexType name="Employee">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anyType = ...
 DataObject customer = ...
 DataObject employee = ...
```

```
// Set the person to a Customer
anyType.set("person", customer);

// The instance data will be a Customer
// Displays "Customer"
System.out.println(anyType.getDataObject("person").getName());

// Set the person to an Employee
anyType.set("person", employee);

// The instance data will be an Employee
// Displays "Employee"
System.out.println(anyType.getDataObject("person").getName());
```

anySimpleType과 유사하게 anyType은 직렬화 중에 xsi:type을 사용하여 직렬화 해제 시 의도한 DataObject 유형이 유지보수되도록 합니다. 따라서 유형을 "Customer"로 설정하는 경우 XML은 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:customer="http://Customer"
    xmlns:p="http://AnyType">
  <person xsi:type="customer:Customer">
    <name>foo</name>
  </person>
</p:AnyType>
```

유형을 "Employee"로 설정하는 경우는 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:employee="http://Employee"
    xmlns:p="http://AnyType">
  <person xsi:type="employee:Employee">
    <id>foo</id>
  </person>
</p:AnyType>
```

AnyType을 사용하면 랩퍼 DataObject를 통해 단순 유형 값을 설정할 수도 있습니다. 이러한 랩퍼 DataObject에는 단순 유형 값을 보유한 "value"(요소)라는 단일 특성이 있습니다. get<Type>/set<Type> API 사용 시 이들 단순 유형 및 랩퍼 DataObject를 자동으로 랩핑하거나 랩핑 해제하도록 SDO API가 대체되었습니다. 비유형 캐스팅 get/set API는 이러한 랩핑을 수행하지 않습니다.

```
DataObject anyType = ...

// Calling a set<Type> API on an anyType Property causes automatic
// creation of a wrapper DataObject
anyType.setString("person", "foo");

// The regular get/set APIs are not overridden, so they will return
// the wrapper DataObject
```

```
DataObject wrapped = anyType.get("person");

// The wrapped DataObject will have the "value" Property
// Displays "foo"
System.out.println(wrapped.getString("value"));

// The get<Type> API will automatically unwrap the DataObject
// Displays "foo"
System.out.println(anyType.getString("person"));
```

랩퍼 DataObject를 직렬화하는 경우 xsi:type 필드의 XSD 유형에 대한 Java 인스턴스 클래스의 anySimpleType 맵핑과 유사하게 직렬화됩니다. 따라서 이 설정은 다음과 같이 직렬화됩니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd=" http://www.w3.org/2001/XMLSchema"
 xmlns:p="http://AnyType">
  <person xsi:type="xsd:string">foo</person>
</p:AnyType>
```

xsi:type이 주어지지 않거나 잘못된 xsi:type이 주어진 경우에는 예외가 처리됩니다. 자동 랩핑 이외에도 BOFactory createDataTypeWrapper(유형, 오브젝트)를 통해 set() API에서 사용하도록 랩퍼를 수동으로 작성할 수 있습니다. 여기서, 유형은 랩핑할 데이터의 SDO 단순 유형이고 오브젝트는 랩핑할 데이터입니다.

```
Type stringType = boType.getType("http://www.w3.org/2001/XMLSchema"
, "string");
DataObject stringType = boFactory.createByMessage(stringType, "foo");
```

DataObject가 랩퍼 유형인지 판별하기 위해 BOType isDataTypeWrapper(유형)를 호출할 수 있습니다.

```
DataObject stringType = ...
boolean isWrapper = boType.isDataTypeWrapper(stringType.getType());
```

기타 복합 유형의 경우 한 필드에서 다른 필드로 데이터를 이동시키려면 데이터 유형이 동일해야 합니다. AnyType에는 모든 복합 유형이 포함될 수 있으나, 이 때문에 필드의 인스턴스 데이터에 따라 맵핑하지 않고도 바로 이동할 수 있는지 여부가 결정됩니다.

특성 유형의 URI 및 이름을 사용하여 특성 유형이 anyType인지 판별할 수 있습니다. URI는 "commonj.sdo"이고 이름은 "DataObject"입니다. 모든 데이터를 anyType에 삽입할 수 있습니다. 이로 인해 다음과 같은 맵핑 규칙이 적용됩니다.

- anyType은 항상 anyType에 맵핑할 수 있습니다.
- 모든 복합 유형은 항상 anyType에 맵핑할 수 있습니다.
- 모든 단순 유형은 항상 anyType에 맵핑할 수 있습니다.

- anyType은 BO 인스턴스의 해당 값에 따라 기타 단순 또는 복합 유형에 맵핑할 수 있습니다. 이는 해당 맵핑을 설계 시 판별할 수 없으며 런타임 시에만 판별할 수 있음을 의미합니다.

Any를 사용하여 복합 유형에 대한 글로벌 요소 설정:

<any/> 태그를 사용하여 복합 유형에 글로벌 요소를 설정할 수 있습니다.

태그의 어커런스는 DataObject Type isOpen() 메소드와 isSequenced() 메소드가 참을 리턴하게 합니다. any 태그에서 maxOccurs의 값이 1보다 큰 경우 DataObject의 구조에 아무 영향도 주지 않습니다. 유효성 검증 시 정보로만 사용됩니다. 마찬가지로 임의의 유형의 다중 any 태그 어커런스는 DataObject의 구조를 변경시키지 않습니다. 설정된 열린 데이터 위치의 유효성을 검증하는 데만 사용됩니다.

내 DataObject에 태그가 있는지 확인하는 방법:

인스턴스 특성을 확인하여 열린 특성 중에 속성이 있는지 확인함으로써 DataObject의 인스턴스에 any 값이 설정되었는지 여부를 간단히 판별할 수 있습니다.

DataObject는 DataObject 유형에 any 태그가 있는지 판별할 수 있는 메커니즘을 제공하지 않습니다. DataObject에는 any 및 anyAttribute 둘 다에 적용되고 모든 특성을 자유롭게 추가할 수 있도록 허용하는 "열림" 개념만 있습니다. any 태그가 있는 경우 DataObject에서 isOpen() = true, isSequenced() = true가 되지만, anyAttribute 태그 및 순서 주제에서 설명한 순서 지정 이유 중 하나가 있을 수도 있습니다. 다음 예제는 이러한 개념에 대해 설명합니다.

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// any values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have any values set

// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Elements
for (int i=definedPropertyCount; i < instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isElement(prop))
    {
        return true;  // Found an any value
    }
```

```
}

return false;  // Does not have any values set
```

Any 값 가져오기/설정 방법:

Any 필드에 설정된 데이터 이름을 알고 있는 경우 기타 요소 값과 동일한 방법으로 데이터 가져오기를 수행할 수 있습니다.

XPath "<이름>"을 사용하여 가져오기를 수행할 수 있으며 가져오기가 해결됩니다. 이름을 알 수 없는 경우에는 앞에서와 마찬가지로 인스턴스 특성을 확인하여 값을 찾을 수 있습니다. any 태그가 여러 개 있거나 maxOccurs > 1인 any 태그가 있는 경우 데이터를 가져온 원본 any 태그를 판별하는 일이 중요하면 DataObject 순서를 대신 사용해야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <!-- Handle all these any one way -->
      <xsd:any maxOccurs="3"/>
      <xsd:element name="marker1" type="xsd:string"/>
      <!-- Handle this any in another -->
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

<any/> 태그로 인해 DataObject 순서가 지정되므로 any 특성 위치의 순서를 확인하여 설정된 any 값을 판별할 수 있습니다.

다음 코드를 사용하여 다음과 같은 XSD의 인스턴스 데이터가 속하는 any 태그를 판별할 수 있습니다.

```
DataObject anyElemAny = ...
Seqeuence seq = anyElemAny.getSequence();

// Until we encounter the marker1 element, all the open data
// found belongs to the first any tag
boolean foundMarker1 = false;

for (int i=0; i<seq.size(); i++)
{
    Property prop = seq.getProperty(i);

    // Check to see if the property is an open property
    if (prop.isOpenContent())
    {
        if (!foundMarker1)
        {
            // Must be the first any because it occurs
            // before the marker1 element
```

```
            System.out.println("Found first any data:

     "+seq.getValue(i));
        }
        else
        {
            // Must be the second any because it occurs
            // after the marker1 element
            System.out.println("Found second any data:

     "+seq.getValue(i));
         }
    }
    else
    {
        // Must be the marker1 element
        System.out.println("Found marker1 data: "+seq.getValue(i));
        foundMarker1 = true;
    }
}
```

글로벌 요소 특성을 작성하고 해당 값을 순서에 추가하여 <any/> 값을 설정합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://GlobalElems"
 targetNamespace="http://GlobalElems">
  <xsd:element name="globalElement1" type="xsd:string"/>
  <xsd:element name="globalElement2" type="xsd:string"/>
</xsd:schema>

 DataObject anyElemAny = ...
 Seqeuence seq = anyElemAny.getSequence();

 // Get the global element Property for globalElement1
 Property globalProp1 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement1", true);

 // Get the global element Property for globalElement2
 Property globalProp2 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement2", true);

 // Add the data to the sequence for the first any
 seq.add(globalProp1, "foo");
 seq.add(globalProp1, "bar");

 // Add the data for the marker1
 seq.add("marker1", "separator");  // or anyElemAny

 .set("marker1", "separator")

 // Add the data to the sequence for the second any
 seq.add(globalProp2, "baz");

 // The data can now be accessed by a get call
 System.out.println(dobj.get("globalElement1"); // Displays "[foo, bar]"
 System.out.println(dobj.get("marker1");        // Displays "separator"
 System.out.println(dobj.get("globalElement2"); // Displays "baz"
```

Any에 있는 데이터에 대해 유효한 맵핑:

<any/> 태그는 이름/값 쌍 세트입니다. 설계 시 <any/>에 대해 판별할 수 있는 유효한 맵핑은 maxOccurs 값이 동일한 다른 <any/> 또는 anyType뿐입니다.

개별적으로, any에 사용할 DataObject의 인스턴스에 포함된 값은 모든 복합 유형 맵핑 규칙을 따르는 기본 복합 유형입니다. 이러한 복합 유형의 일부는 단순 유형으로 랩핑되어 단순 유형 맵핑 규칙을 따를 수도 있습니다.

AnyAttribute를 사용하여 복합 유형에 대한 글로벌 속성 설정:

<anyAttribute/> 태그를 사용하면 임의의 수의 글로벌 속성을 복합 유형에 설정할 수 있습니다.

<any/> 태그와 마찬가지로 <anyAttribute/> 태그의 어커런스는 DataObject Type isOpen() 메소드가 참을 리턴하게 합니다. 그러나 XSD의 속성이 순서가 지정된 구조가 아니기 때문에 <any/> 태그와는 달리 <anyAttribute/> 태그로 인해 DataObject의 순서가 지정되지는 않습니다.

내 DataObject에 anyAttribute 태그가 있는지 확인하는 방법:

인스턴스 특성을 확인하여 열린 특성 중에 속성이 있는지 확인함으로써 DataObject의 인스턴스에 anyAttribute 값이 설정되었는지 여부를 간단히 판별할 수 있습니다.

DataObject는 DataObject 유형에 anyAttribute 태그가 있는지 판별할 수 있는 메커니즘을 제공하지 않습니다. DataObject에는 any 및 <anyAttribute/> 둘 다에 적용되고 모든 특성을 자유롭게 추가할 수 있도록 허용하는 "열림" 개념만 있습니다. DataObject에서 isOpen() = true, isSequenced() = false인 경우는 참이지만 이 경우 anyAttribute 태그가 있어야 하며 isOpen() = true, isSequenced() = true인 경우에는 DataObject 유형에 anyAttribute 태그가 있을 수도 있습니다.

DataObject는 메타데이터 조회 메소드를 제공하여 DataObject를 생성하는 데 사용된 XSD 구조에 대한 해당 질문 및 기타 질문에 프로그래밍을 통해 응답합니다. anyAttribute 태그가 있는지 확인해야 하는 경우 InfoSet 모델을 조회할 수 있습니다. anyAttribute는 단일 속성이며 참 또는 참이 아님 둘 중 하나이므로 비즈니스 오브젝트 또한 BOXSDHelper hasAnyAttribute(Type) 메소드를 제공하여 해당 DataObject에 대해 열린 속성을 설정하면 유효한 결과가 생성되는지 여부를 판별할 수 있도록 합니다. 다음 코드 예제는 이러한 개념에 대해 설명합니다.

```
DataObject dobj = ...

// Check to see if the type is open, if it isn't then it can't have
// anyAttribute values set in it.
boolean isOpen = dobj.getType().isOpen();

if (!isOpen) return false;  // Does not have anyAttribute values set
```

```
// Open Properties are added to the Instance Property list, but not
// the Property list, so comparing their sizes can easily determine
// if any open data is set
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// If equal, does not have any open content set
if (instancePropertyCount == definedPropertyCount) return false;

// Check the open content Properties to determine if any are Attributes
for (int i=definedPropertyCount; i<instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    if (boXsdHelper.isAttribute(prop))
    {
        return true;  // Found an anyAttribute value
    }
}

return false;  // Does not have anyAttribute values set
```

anyAttribute 값 가져오기/설정 방법:

<anyAttribute/> 값 설정은 <any/>를 설정하는 것과 동일한 방법으로 수행되지만 글로벌 요소 대신 글로벌 속성을 사용합니다.

anyAttribute 필드에 설정된 데이터 이름을 알고 있는 경우 기타 속성 값과 동일한 방법으로 데이터 가져오기를 수행할 수 있습니다. XPath "@<name>"을 사용하여 가져오기를 수행할 수 있으며 가져오기가 해결됩니다. 이름을 알 수 없는 경우 위의 코드를 사용하여 값을 반복시키고 차례로 값에 액세스할 수 있습니다. 다음 코드 예제는 이 작업을 수행하는 방법을 보여줍니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyAttrOnlyMixed"
 targetNamespace="http://AnyAttrOnly">
  <xsd:complexType name="AnyAttrOnly">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://GlobalAttrs">
  <xsd:attribute name="globalAttribute" type="xsd:string"/>
</xsd:schema>

 DataObject dobj = ...

 // Get the global attribute Property that is going to be set
 Property globalProp = boXsdHelper.getGlobalProperty(http://GlobalAttrs,
 "globalAttribute", false);
```

```
// Set the value on the dobj, just like any other data
dobj.set(globalProp, "foo");

// The data can now be accessed by a get call
System.out.println(dobj.get("@globalAttribute")); // Displays "foo"
```

anyAttribute에 있는 데이터에 대해 유효한 맵핑:

AnyAttribute 태그는 모든 태그와 유사하며 이름/값 쌍 세트입니다. 따라서 anyAttribute에 유효한 맵핑은 다른 anyAttribute뿐입니다.

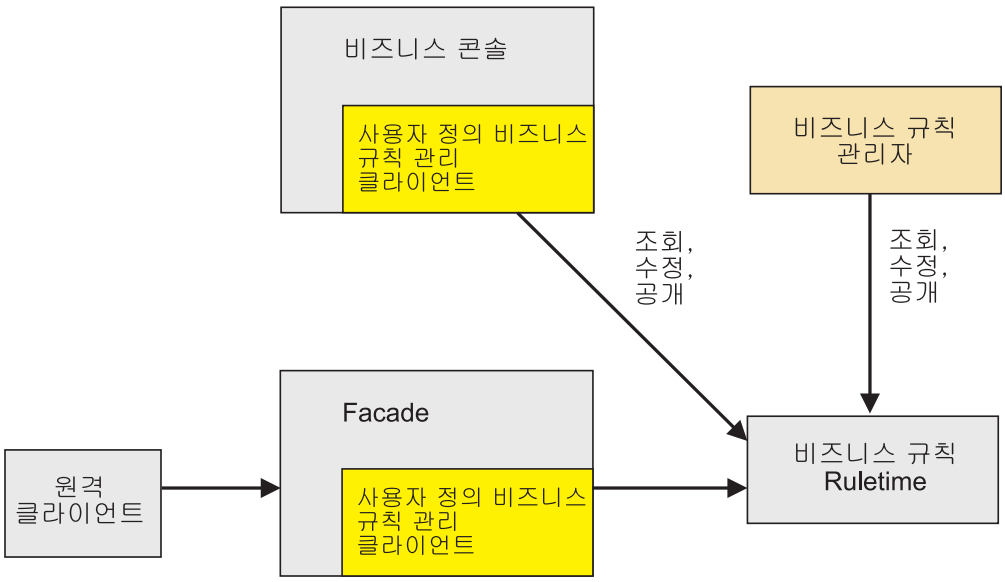
개별적으로, anyAttribute 데이터에 포함된 값은 모든 단순 유형 맵핑 규칙을 따르는 기본 단순 유형입니다.

비즈니스 규칙 관리 프로그래밍

사용자 정의 관리 클라이언트의 빌드를 위해 또는 비즈니스 규칙에 대한 변경을 자동화하기 위해 공용 비즈니스 규칙 관리 클래스가 제공됩니다.

단일 클라이언트에서 모든 컴포넌트를 관리하기 위해 비즈니스 프로세스나 휴먼 타스크를 위한 다른 관리 기능과 결합된 웹 응용프로그램에서 비즈니스 규칙 관리 클래스를 사용할 수 있었습니다. 모든 사용자 정의 관리 클라이언트는 WebSphere Process Server와 함께 포함된 비즈니스 규칙 관리자 웹 응용프로그램과 함께 사용될 수 있습니다. 응용프로그램 내에서 비즈니스 규칙에 대한 변경사항을 자동화하기 위해 클래스를 사용할 수 있었습니다. 예를 들어, 비즈니스 규칙을 사용 중인 비즈니스 프로세스의 결과가 일부 임계값 또는 한계를 초과할 때 비즈니스 규칙을 변경할 수 있었습니다.

WebSphere Process Server에 설치된 응용프로그램에서 비즈니스 규칙 관리 클래스를 사용할 수 있어야 합니다. 클래스는 원격 인터페이스를 제공하지 않지만, 원격 실행을 위해 특정 프로토콜에서 표시되는 facade에서 클래스를 줄 바꾸기할 수 있습니다.



이 프로그래밍 안내서는 두 개의 기본 절과 부록으로 구성됩니다. 첫 번째 절에서는 프로그래밍 모델과 여러 클래스 사용 방법을 설명합니다. 클래스 간의 관계를 표시하기 위해 클래스 다이어그램을 제공합니다. 두 번째 절에서는 비즈니스 규칙 그룹 검색, 새 규칙 대상 스케줄 및 규칙 세트나 의사결정 테이블 수정과 같은 조치를 수행하기 위해 클래스 사용에 대한 예제를 제공합니다. 부록에는 일반 조작을 단순화시키기 위해 예제에서 사용된 추가 클래스가 있으며, 와일드카드를 사용하여 비즈니스 규칙 그룹을 검색하기 위해 복합 조회를 작성하는 추가 예제가 있습니다.

WebSphere Integration Developer v6.1과 함께 포함된 WebSphere Process Server v6.1 및 테스트 환경 둘 다에 포함된 Javadoc HTML 형식에서도 클래스에 대한 이 프로그램 안내서 정보가 제공됩니다. 이 Javadoc 문서는 ${WebSphere Process Server Install Directory}₩web₩apidocs 또는 ${WebSphere Integration Developer Install Directory}₩runtimes₩bi_v61₩web₩apidocs에서 사용 가능합니다. com.ibm.wbiserver.brules.mgmt.* 패키지에는 모든 정보가 포함되어 있습니다.

프로그래밍 모델

WebSphere 비즈니스 통합 비즈니스 규칙은 두 가지 다른 작성자 도구로 작성되며 규칙 런타임으로 발행됩니다. 세 가지 모두 비즈니스 규칙 아티팩트의 동일한 모델을 공유합니다.

모델 공유는 이후의 유지보수 용이성을 위해서만이 아니라 일반 사용자의 일관된 프로그래밍 모델을 위해서도 중요하게 고려되었습니다. 이러한 모델 공유를 위해서는 데스크탑 도구화, 런타임 실행 및 작성의 필요성을 절충해야 했습니다. 이는 모두 각 환경을 충족시키는 명확한 요구사항 세트를 지니고 있으며 이러한 요구사항이 충돌하는 경우가 있기 때문입니다. 전체 프로그래밍 모델의 일부로 아래 설명된 아티팩트는 여러 환경의 요구사항을 충족시키는 밸런스를 나타냅니다.

비즈니스 규칙의 수정은 조작 선택사항 테이블(유효 날짜 및 대상)은 물론 규칙 세트 및 의사결정 테이블의 템플리트를 이용하여 정의된 항목만으로 제한됩니다. 새 규칙 세트 및 의사결정 테이블의 작성은 기존 규칙 세트나 의사결정 테이블의 사본을 통해서만 지원됩니다. 비즈니스 규칙 그룹 컴포넌트 자체는 사용자 정의 특성 및 설명 값의 예외를 이용하는 런타임의 동적 작성에는 적합하지 않습니다. 컴포넌트에 필요한 변경사항은(예: 새 조작 추가) WebSphere Integration Developer를 사용하여 수행된 후 서버에서 재전개 또는 재설치되어야 합니다.

비즈니스 규칙 그룹

BusinessRuleGroup 클래스는 비즈니스 규칙 그룹 컴포넌트를 표시합니다. BusinessRuleGroup 클래스는 규칙 세트와 의사결정 테이블을 포함하는 루트 오브젝트로 고려될 수 있습니다.

규칙 세트 및 의사결정 테이블은 연관된 비즈니스 규칙 그룹을 통해서만 접근될 수 있습니다. 비즈니스 규칙 그룹에 대한 정보를 검색하고 규칙 세트 및 의사결정 테이블에 접근하기 위해 클래스에 대한 메소드가 제공됩니다. 메소드를 통해 다음 정보가 검색될 수 있습니다.

- 대상 네임스페이스
- 비즈니스 규칙 그룹의 이름
- 표시 이름
- 이름/표시 이름 동기화
- 설명
- 날짜를 UTC 형식으로 표시할지, 시스템 로컬로 표시할지 여부를 나타내는 프리젠테이션 시간대
- 비즈니스 규칙 그룹과 연관된 인터페이스에 정의된 조작
- 비즈니스 규칙 그룹에 정의된 사용자 정의 특성

비즈니스 규칙 그룹과 연관된 여러 규칙 세트 및 의사결정 테이블은 비즈니스 규칙 그룹의 조작을 통해 접근될 수 있습니다.

또한 비즈니스 규칙 그룹에서 정보가 갱신되게 하는 메소드도 있습니다. 메소드를 통해, 다음 정보를 갱신할 수 있습니다.

- 설명
- 표시 이름
- 이름/표시 이름 동기화
- 비즈니스 규칙 그룹에 정의된 사용자 정의 특성

비즈니스 규칙 그룹의 표시 이름을 명시적으로 설정하거나 setDisplayNameIsSynchronizedToName 메소드를 사용하여 이름 값으로 설정될 수 있습니다.

비즈니스 규칙 그룹 컴포넌트 정의의 일부가 되도록 다른 값을 수정할 수 없으며 이들 값에 대한 변경은 재설치와 마찬가지로 재전개를 필요로 합니다.

비즈니스 규칙 그룹 클래스는 새로 고치기 메소드를 제공합니다. 이 메소드는 비즈니스 규칙이 저장되고 비즈니스 규칙 그룹과 지속되는 정보와 연관된 모든 규칙 세트와 의사 결정 테이블을 리턴하는 지속적 기억장치나 저장소에 대한 호출을 작성합니다. 리턴되는 비즈니스 규칙 그룹은 최신 사본이며 이전 오브젝트는 사용하지 않습니다.

비즈니스 규칙 그룹 인스턴스가 현재 런타임에서 지원하지 않는 버전인지 여부를 알리기 위해 isShell 메소드를 사용할 수 있습니다. 예를 들어, 현재 비즈니스 규칙 관리 클래스를 이용하여 웹 클라이언트를 작성하고, 클래스에서 지원하지 않은 비즈니스 규칙 그룹에 장래 새 기능을 추가하면, 비즈니스 규칙 그룹이 검색될 때 쉘 비즈니스 규칙 그룹이 작성됩니다. 이렇게 하면 웹 클라이언트는 지원되는 비즈니스 규칙으로 계속 작업할 수 있으며 제한된 속성과 기능을 이용하여 여전히 비즈니스 규칙 그룹을 검색할 수 있습니다. isShell이 참이면, getName, getTargetNameSpace, getProperties, getPropertyValue 및 getProperty 메소드만이 값을 리턴합니다. 다른 모든 메소드는 UnsupportedOperationException을 유발합니다. 그리고 isShell 메소드 사용 시, 지원되는 버전인지 여부를 결정하기 위해 BusinessRuleGroup의 유형이 BusinessRuleGroupShell의 인스턴스인지 여부를 검사합니다.

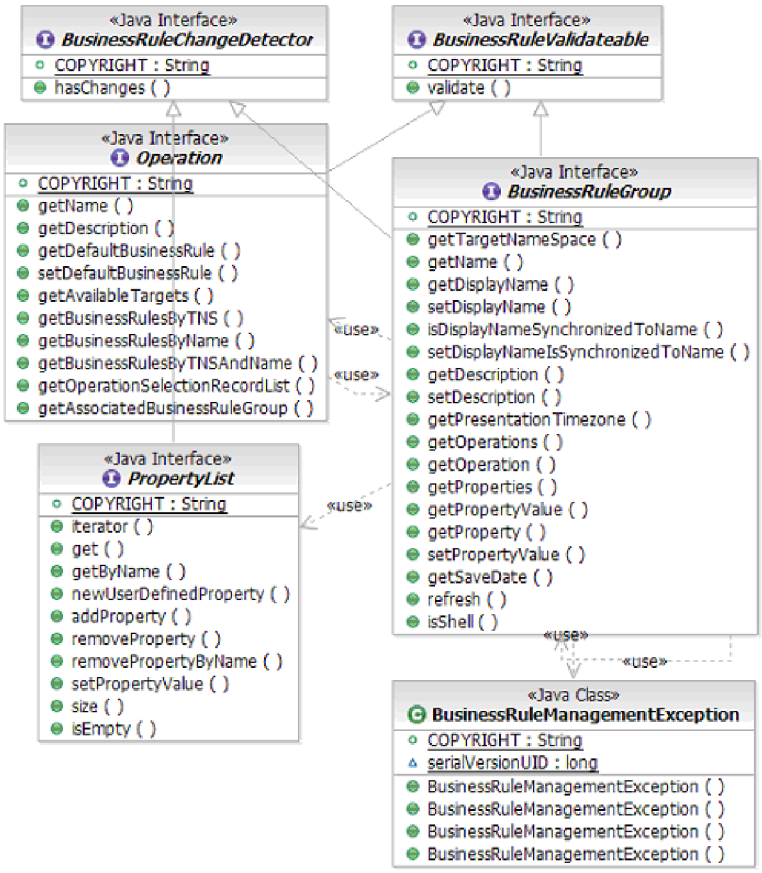


그림 59. BusinessRuleGroup 및 관련 클래스의 클래스 다이어그램

비즈니스 규칙 그룹 특성

비즈니스 규칙 그룹의 특성은 비즈니스 규칙 그룹의 관리를 위해 사용됩니다. 비즈니스 규칙 그룹의 특성 세트는 표시된 후 수정되는 비즈니스 규칙 그룹의 서브세트만을 리턴하기 위해 조회에서 사용될 수 있습니다.

모든 특성은 유형 문자열이며 이름-값 쌍으로 정의됩니다. 각 특성은 비즈니스 규칙 그룹에 한 번만 정의될 수 있습니다. 정의된 각 특성의 경우, 값도 정의되어야 합니다. 특성 값은 비어 있는 문자열이거나 길이가 0일 수 있으나 널일 수 없습니다. 특성을 널로 설정하면 특성을 삭제하는 것과 동일합니다.

비즈니스 규칙 그룹의 특성은 런타임 시 규칙 세트나 의사결정 테이블에서도 액세스될 수 있습니다. 이것은 비즈니스 규칙 그룹에서 단일 값을 설정하여 비즈니스 규칙 그룹의 의사결정 테이블이나 다중 규칙 세트 내에서 사용될 수 있게 합니다. 비즈니스 규칙 그룹에 정의된 특성만이 둘러싼 규칙 세트와 의사결정 테이블에 사용 가능합니다.

두 가지 유형의 특성, 시스템 및 사용자 정의가 있습니다. 시스템이나 사용자 정의 특성의 수는 비즈니스 규칙 그룹에 제한되지 않습니다. 시스템 특성은 규칙 로직 정의에

사용되는 규칙 모델의 버전과 같이 특정 컴포넌트 정보를 보유하기 위해 사용됩니다. 이 시스템 정보는 이들 필드에서 조회를 허용하도록 특성에 표시되어 있습니다. 시스템 특성은 접두부 IBMSystem으로 시작하며 비즈니스 규칙 그룹 및 특성 클래스에서 읽기 전용입니다. 시스템 특성은 추가, 변경 또는 삭제될 수 없습니다. 시스템 특성의 예제는 다음과 같습니다.

특성 이름	특성 값
IBMSystemVersion	6.2.0

주: 비즈니스 규칙 그룹의 이름, 네임스페이스 및 표시 이름의 값은 조회 목적으로 시스템 특성으로 취급되며, getProperties 메소드를 이용하여 비즈니스 규칙 그룹에 대해 검색될 수 있는 특성 목록의 일부입니다. 그러나 이들 특성은 비즈니스 규칙 그룹 아티팩트에서 실제 특성 요소로 정의되며, 비즈니스 규칙 그룹에서 독립된 고유의 요소로 정의되면 WebSphere Integration Developer에서 특성으로 보이지 않습니다. 단지 더 많은 조회 옵션을 위해 제공됩니다.

사용자 정의 특성은 고객 고유의 정보를 보유하기 위해 사용되며 비즈니스 규칙 그룹의 조회에서도 사용될 수 있습니다. 사용자 정의 특성은 읽기-쓰기에 사용 가능합니다.

비즈니스 규칙 그룹의 특성은 개별적으로 또는 목록으로(PropertyList 오브젝트) 검색될 수 있습니다. PropertyList를 이용 시, 개별 특성을 검색하고 사용자 정의 특성을 추가 및 제거하기 위해 메소드가 제공됩니다.

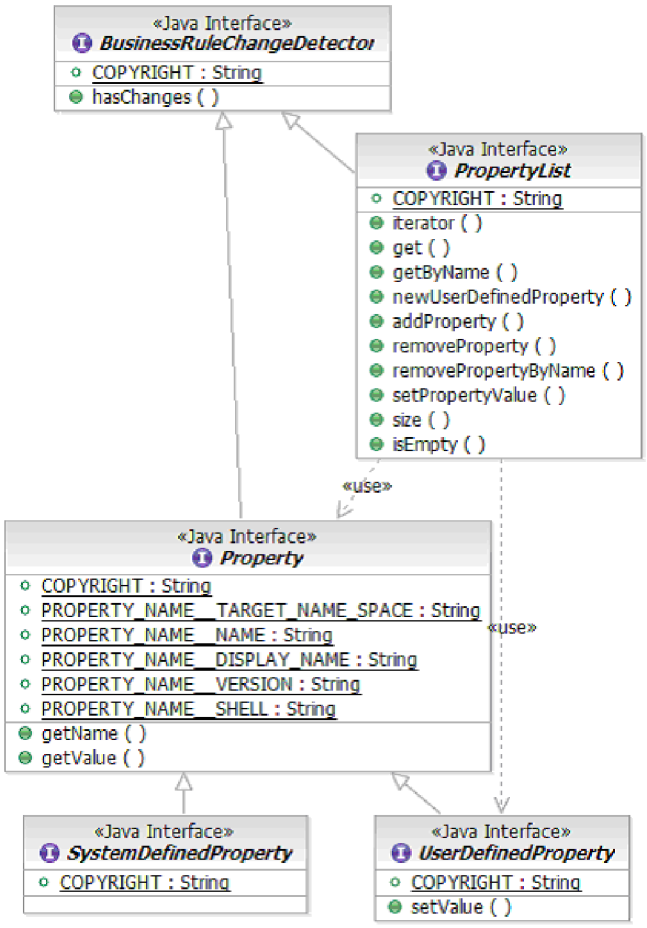


그림 60. 특성 및 관련 클래스의 클래스 다이어그램

조작

조작은 수정하려는 각 규칙 세트 및 의사결정 테이블을 접근하는 시작점입니다. 비즈니스 규칙 그룹의 조작은 비즈니스 규칙 그룹 컴포넌트와 연관된 WSDL에 나열된 조작과 일치합니다.

각 조작의 경우, 각각 비즈니스 규칙(규칙 세트 또는 의사결정 테이블)인 여러 대상이 있습니다.

- 기본값 대상(선택적)
- 날짜/시간 범위로 예정된 대상 목록(OperationSelectionRecord)
- 해당 조작에 사용될 수 있는 사용 가능한 모든 대상 목록

각 조작은 최소 하나의 비즈니스 규칙 대상이 지정되어야 합니다. 이 대상은 활성 상태가 되도록 예정될 때 특정 시작 날짜 및 종료 날짜를 갖는 OperationSelectionRecord가 될 수 있습니다. 또한 일치하는 예정된 비즈니스 규칙 대상이 없을 때 실행 중 사용되는 단일 기본값 대상 세트를 가질 수 있습니다. Operation

클래스는 예정된 비즈니스 규칙 대상 목록(OperationSelectionRecordList) 검색은 물론 기본값 비즈니스 규칙 대상을 검색하고 설정하는 메소드를 제공합니다. 기본값 비즈니스 규칙 대상 및 예정된 비즈니스 규칙 대상 이외에, 조작에 사용 가능한 모든 비즈니스 규칙 대상 목록이 있습니다. 이 목록은 이 조작에 예정되지 않은 다른 모든 규칙 세트나 의사결정 테이블은 물론 기본값 비즈니스 규칙 대상과 예정된 비즈니스 규칙 대상을 포함합니다. 예정되지 않은 규칙 세트나 의사결정 테이블은 내재적으로 조작 정보를 공유한다는 사실에 의해 사용 가능한 대상 목록을 통해 조작과 연관됩니다. 모든 비즈니스 규칙은 해당 조작의 입출력 메시지를 지원해야 합니다. 인터페이스에 고유한 각 조작에서, 조작의 규칙 세트 및 의사결정 테이블은 다른 조작의 규칙 세트와 의사결정 테이블과 달리 고유합니다.

사용 가능한 대상 목록에서 다른 규칙 세트와 의사결정 테이블 모두가 OperationSelectionRecord의 작성을 통해 활성화되도록 예정될 수 있습니다. 사용 가능한 대상 목록의 특정 규칙 세트나 의사결정 테이블과 함께, 시작 날짜 및 종료 날짜가 지정되어야 합니다. 시작 날짜는 종료 날짜 이전이어야 합니다. 날짜는 과거 및 장래는 물론 현재 날짜를 포괄하는 시간일 수 있습니다. 날짜의 시간 범위가 OperationSelectionRecordList에 추가되고 공개되고 나면 다른 OperationSelectionRecords와 겹칠 수 없습니다. 시작 날짜 및 종료 날짜 값은 java.util.Date 유형입니다. 지정된 모든 값은 java.util.Date 클래스에 따라 UTC 값으로 취급됩니다. OperationSelectionRecord 완료 시, OperationSelectionRecordList에 추가되어 다른 비즈니스 규칙 대상과 함께 예정될 수 있습니다. 여러 OperationSelectionRecords의 시간 범위 내에 간격이 있을 수 있습니다. 실행 중 간격이 벌어지면, 기본값 대상이 사용됩니다. 지정된 기본값 대상이 없으면, 예외가 발생합니다. 항상 기본값 비즈니스 규칙 대상을 지정하는 것이 좋습니다.

예정된 비즈니스 규칙 대상은 OperationSelectionRecordList에서 OperationSelectionRecord를 제거하여 예정된 대상 목록에서 제거될 수 있습니다. OperationSelectionRecord를 제거한다고 해서 사용 가능한 비즈니스 규칙 대상 목록에서 비즈니스 규칙 대상을 제거하지는 않으며 예정된 동일한 비즈니스 규칙 대상을 갖는 다른 OperationSelectionRecords를 제거하지 않습니다.

OperationSelectionRecordList을 통해 의사결정 테이블이나 규칙 세트를 검색하는 것 이외에, Operation 클래스는 이름이나 대상 네임스페이스 특성 값으로 비즈니스 규칙 대상을 검색하도록 허용합니다. Operation 클래스의 메소드를 통해, 해당 조작의 사용 가능한 대상에 표시된 규칙 세트와 의사결정 테이블이 조회될 수 있습니다. 일치하는 이름과 대상 네임스페이스를 가질 수 있는 규칙 세트와 의사결정 테이블은 다른 조작이 사용 가능한 대상 목록의 일부이지만 결과 세트에 포함되지 않습니다. 편의상, getBusinessRulesByName, getBusinessRulesByTNS 및 getBusinessRulesByTNSAndName 메소드는 특정 규칙 세트와 의사결정 테이블 검색을 단순화시킵니다.

Operation 클래스는 다음을 지원하는 메소드를 제공합니다.

- 조작 이름 검색
- 조작 설명 검색
- 기본값 비즈니스 규칙 대상 검색 및 설정
- 예정된 비즈니스 규칙 대상 검색(OperationSelectionRecordList)
- 사용 가능한 모든 비즈니스 규칙 대상 검색
- 이름이나 대상 네임스페이스에 의해 사용 가능한 모든 대상 목록에서 규칙 세트나 의사결정 테이블 검색
- 조작이 연관된 비즈니스 규칙 그룹 검색

OperationSelectionRecordList 클래스는 다음을 지원하는 메소드를 제공합니다.

- 색인 값에 의해 특정 OperationSelectionRecord 검색
- 색인 값에 의해 특정 OperationSelectionRecord 제거
- 새로운 OperationSelectionRecord를 목록에 추가

OperationSelectionRecord 클래스는 다음을 지원하는 메소드를 제공합니다.

- 시작 날짜 검색 및 설정
- 종료 날짜 검색 및 설정
- 비즈니스 규칙 대상 검색 및 설정
- OperationSelectionRecord가 연관된 조작 검색

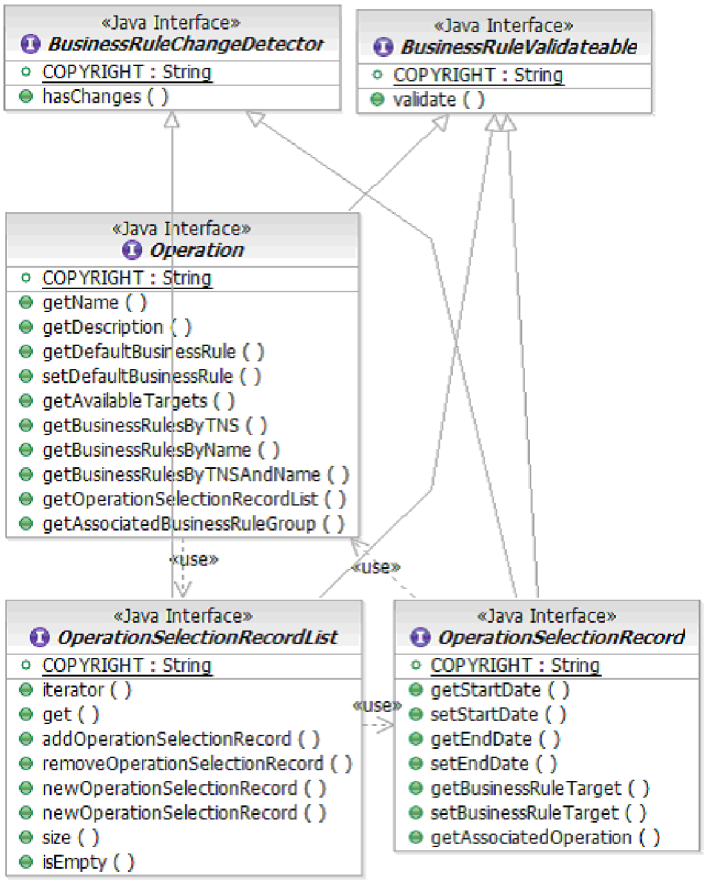


그림 61. 조작 및 관련 클래스의 클래스 다이어그램

비즈니스 규칙

RuleSet 및 DecisionTable 클래스는 규칙 세트 및 의사결정 테이블 둘 다에서 사용 가능한 정보를 제공하는 메소드를 사용하여 일반 BusinessRule 클래스를 기반으로 합니다.

비즈니스 규칙 그룹 아티팩트와 유사하게, 규칙 세트 및 의사결정 테이블에는 이름과 대상 네임스페이스가 있습니다. 다른 규칙 세트와 의사결정 테이블과 비교할 때 이들 값의 조합은 고유해야 합니다. 예를 들어, 두 개의 규칙 세트는 동일한 대상 네임스페이스 값을 공유할 수 있지만 다른 이름이나 규칙 세트를 가져야 하며 의사결정 테이블은 동일한 이름을 가질 수 있지만 다른 대상 네임스페이스 값을 갖습니다.

템플리트에서 구성된 규칙의 여러 매개변수 값을 사용하여 특정 시간에 유사한 규칙이 예정되는 상황에서 기존 비즈니스 규칙으로부터 비즈니스 규칙 사본을 작성할 수 있습니다. 비즈니스 규칙을 구현하기 위해 Java 클래스를 되돌리면 새 규칙을 완전히 작성할 수 없습니다. Java 클래스 되돌리기는 전개 시간에만 작성됩니다. 새 규칙이 작성되면, 원래 규칙과 연관된 조작에 사용 가능한 대상 목록에 추가됩니다. 추가 규칙은 지속하지 않지만 조작이 연관된 비즈니스 규칙 그룹이 공개될 때까지 지속됩니다.

새 비즈니스 규칙은 원래 규칙과는 다른 대상 네임스페이스이거나 이름입니다. 이름 및 네임스페이스의 조합이 비즈니스 규칙을 식별하기 위해 키 값을 제공하는 것과 같이 새 비즈니스 규칙의 표시 이름은 원래 규칙과 동일하게 남아 있을 수 있습니다. 비즈니스 규칙 내에서, 템플리트로 정의된 여러 매개변수 값은 수정될 수 있습니다. 특정 시간에 비즈니스 규칙을 스케줄링하는 것은 OperationSelectionRecordList를 사용하거나 비즈니스 규칙과 연관된 조작을 사용하는 기본값 대상으로서 수행될 수 있습니다.

BusinessRule 클래스는 다음을 지원하는 메소드를 제공합니다.

- 대상 네임스페이스 검색
- 규칙 세트 또는 의사결정 테이블 이름 검색
- 규칙 세트 또는 의사결정 테이블의 표시 이름 검색 및 설정
- 비즈니스 규칙 유형(규칙 세트 또는 의사결정 테이블) 검색
- 비즈니스 규칙의 설명 검색 및 설정
- 비즈니스 규칙이 연관된 조작 검색
- 여러 이름 및/또는 대상 네임스페이스를 사용하여 비즈니스 규칙 사본 작성

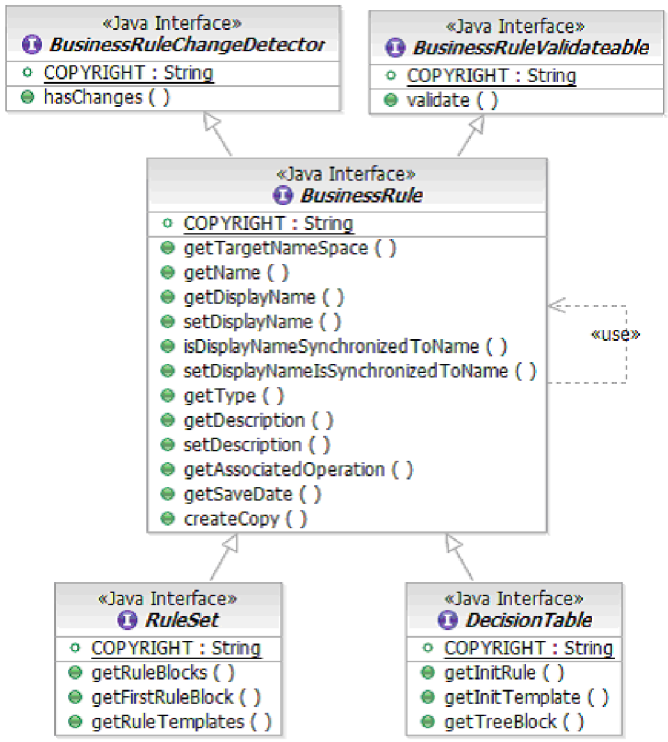


그림 62. BusinessRule 및 관련 클래스의 클래스 다이어그램

규칙 세트

규칙 세트는 비즈니스 규칙의 한 유형입니다. 여러 조건부 값에 근거하여 다중 규칙을 실행할 필요가 있으면 규칙 세트가 보통 사용됩니다. 규칙 세트는 규칙 블록 및 규칙 템플리트로 구성됩니다. 규칙 블록(RuleBlock)에는 여러 if-then과 규칙 세트의 로직을 작성하는 조치 규칙이 포함되어 있습니다.

RuleSet 클래스는 다음을 지원하는 메소드를 제공합니다.

- 규칙 세트의 규칙 블록 목록 검색
- 규칙 세트에 정의된 규칙 템플리트의 목록 검색

현재 각 규칙 세트는 오직 하나의 규칙 블록을 가질 수 있지만, 다중 규칙 템플리트를 규칙 세트에 정의할 수 있습니다. 규칙 블록에는 규칙 세트가 호출될 때 실행될 규칙 세트가 포함되어 있습니다. 규칙 블록은 규칙 순서를 수정하도록 허용합니다. 규칙 블록에는 최소 하나의 규칙이 정의되어야 합니다. 규칙(Rule)은 템플리트 인스턴스 규칙(TemplateInstanceRule)으로 정의되거나 하드코딩되어야 합니다. if-then 또는 조치 규칙이 템플리트를 이용하여 정의된 경우, 규칙 블록에서 제거될 수 있습니다. 새로운 규칙 인스턴스가 템플리트를 이용하여 작성된 경우, 규칙 블록에 추가될 수 있습니다.

규칙이 하드코딩되고 템플리트를 이용하여 정의되지 않은 경우, 규칙 블록에서 수정되거나 제거될 수 없습니다. 이러한 규칙에서 예상할 수 있는 것은 규칙 세트 로직의 일부로 항상 규칙이 설계되었으며 로직 내에서 변경 또는 반복되지 않는다는 것입니다.

새 규칙이 템플리트를 이용하여 작성되면, 고유한 이름 값을 가져야 합니다. 규칙을 작성하기 전에 먼저 기존 규칙 목록을 검색하고 확인할 수 있습니다.

하드코딩된 if-then 및 조치 규칙의 경우, 이름 및 프리젠테이션만이 검색될 수 있습니다. 프리젠테이션은 클라이언트 응용프로그램에서 규칙에 대한 정보를 표시하기 위해 사용할 수 있는 문자열입니다. 템플리트를 이용하여 정의된 if-then 또는 조치 규칙의 경우, 추가 정보와 마찬가지로 이름 및 조치를 검색할 수 있습니다. 특정 매개변수 값을 검색하고 변경할 수 있습니다. 규칙 세트에 정의된 템플리트(RuleSetRuleTemplate) 사용 시 규칙의 다른 인스턴스는 규칙 세트 내에서 작성할 수 있으며 매개변수 값을 설정할 수 있습니다. 예를 들어, 특정 상태 레벨의 고객이 특정량의 할인을 받는 규칙이 있습니다. 이 로직은 단일 규칙 템플리트로 정의된 다음 상태 레벨(gold, silver, bronze 등) 및 할인량(15%, 10%, 5%)에 대해 변경된 매개변수 값으로 반복될 수 있었습니다.

템플리트를 이용하여 정의된 규칙의 매개변수는 규칙의 인스턴스에 특정합니다. 템플리트만이 표준 프리젠테이션과 규칙의 매개변수 수를 정의합니다. 템플리트로 정의된 각 규칙은 다른 고객 상태의 할인에 대한 예에서 설명한 대로 다른 값을 가질 수 있습니다.

RuleBlock 클래스는 다음을 지원하는 메소드를 제공합니다.

- 색인 값에 의해 검색

- 템플리트를 이용하여 정의된 규칙 추가
- 템플리트를 이용하여 정의된 규칙 제거
- 한 위치씩 또는 특정 색인 위치로 규칙 순서 수정

RuleSetRule 클래스는 다음을 지원하는 메소드를 제공합니다.

- 규칙의 이름 검색
- 규칙의 표시 이름 검색
- 사용자 프리젠테이션 검색
- 규칙 블록 검색

RuleSetRuleTemplate 클래스는 다음을 지원하는 메소드를 제공합니다.

- 이 템플리트 정의에서 규칙 템플리트 인스턴스 작성
- 상위 규칙 세트 검색

TemplateInstanceRule 클래스는 다음을 지원하는 메소드를 제공합니다.

- 규칙의 매개변수 검색
- 규칙을 정의한 템플리트 정의 검색

Template 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트 ID 검색
- 이름 검색
- 표시 이름 검색 및 설정
- 설명 검색 및 설정
- 이 템플리트의 매개변수 검색
- 사용자 프리젠테이션 검색

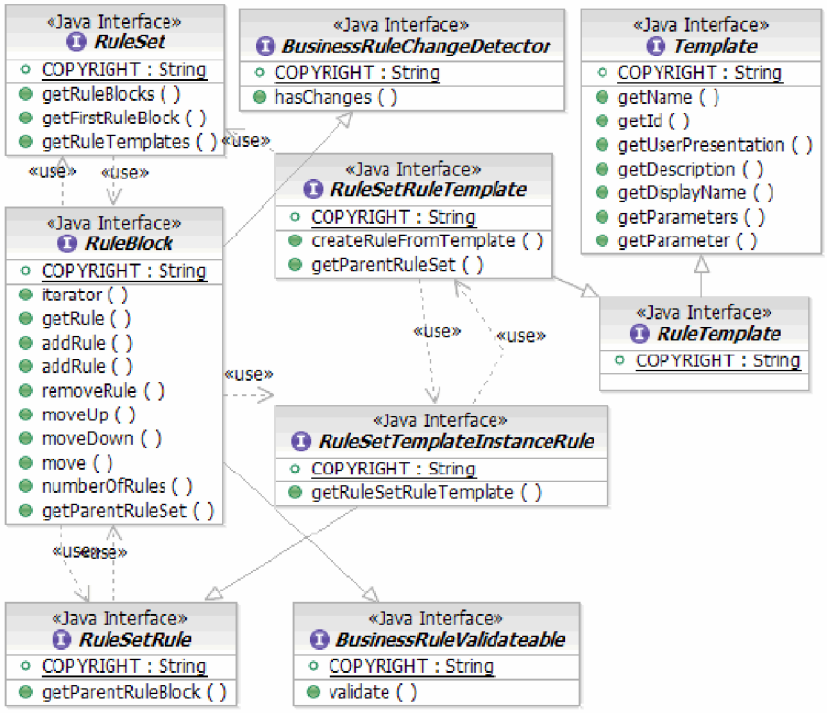


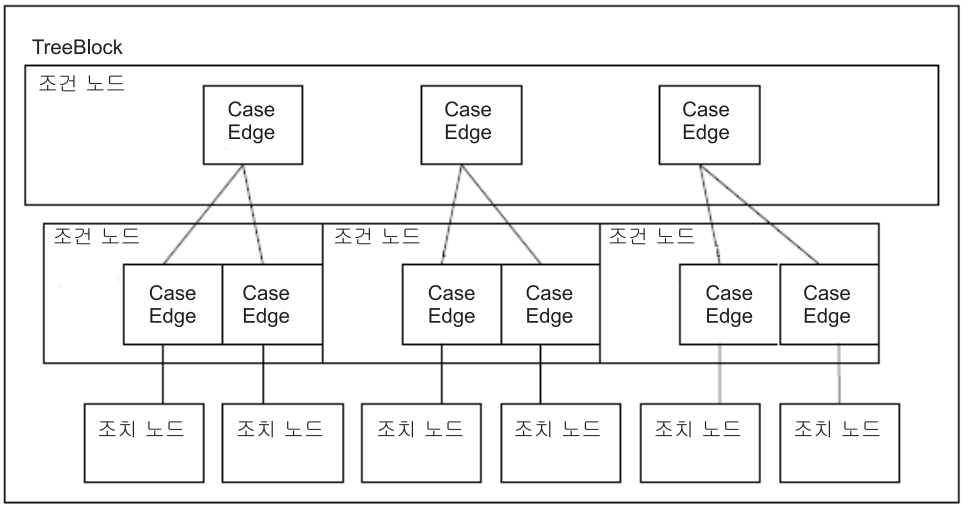
그림 63. BusinessRule 및 관련 클래스의 클래스 다이어그램

의사결정 테이블

의사결정 테이블은 관리되고 수정될 수 있는 비즈니스 규칙의 다른 유형입니다. 평가되어야 하는 일관된 수의 조건과 조건 충족 시 발행되는 특정 조치 세트가 있으면 일반적으로 의사결정 테이블이 사용됩니다.

의사결정 테이블은 결정 트리와 유사하지만 균형이 유지됩니다. 의사결정 테이블은 항상 어떤 분기 세트가 참으로 해석되는지에 관계 없이 수행되는 조치와 평가되는 동일한 수의 조건을 갖습니다. 결정 트리는 다른 분기보다 평가할 조건이 더 많은 분기 하나를 가질 수 있습니다.

의사결정 테이블은 노드 트리로 구조화되며 `TreeBlock`으로 정의됩니다. `TreeBlock`을 구성하는 여러 `TreeNodes`가 있습니다. `TreeNodes`는 조건 노드이거나 조치 노드일 수 있습니다. 조건 노드는 평가 분기입니다. 분기 끝에는 적절한 트리 조치를 발행시키는 조치 노드가 있으며 모든 조건을 참으로 평가해야 합니다. 조건 노드 레벨은 여러 종류가 가능하나 조치 노드 레벨은 한 가지만 가능합니다.



테이블의 조건을 검사하기 전에 발행될 수 있는 초기화 규칙(init 규칙)도 의사결정 테이블에 있을 수 있습니다.

DecisionTable 클래스는 다음을 지원하는 메소드를 제공합니다.

- 트리 노드(조건 및 조치 노드)의 트리 블록 검색
- init 규칙 인스턴스 검색
- 정의된 경우 init 규칙 템플리트 검색

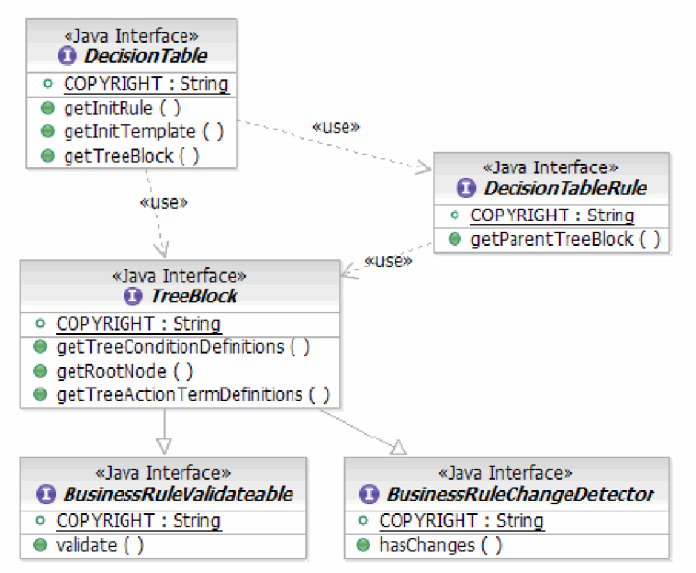
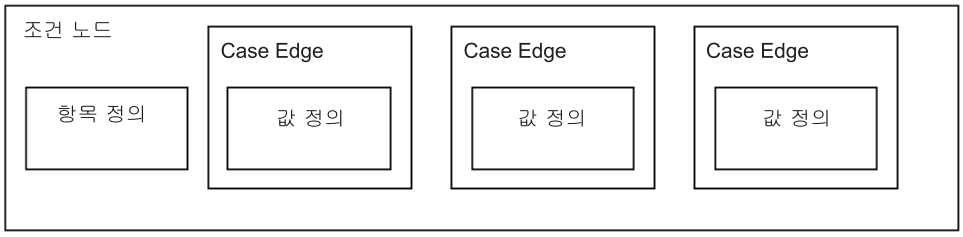


그림 64. DecisionTable 및 관련 클래스의 클래스 다이어그램

의사결정 테이블의 TreeBlock에는 여러 조건 및 조치 노드가 포함되어 있습니다. 각 조건 노드(ConditionNode)에는 항목 정의(TreeConditionTermDefinition)가 있으

며 임의 수의 Case Edge(CaseEdge)가 있습니다. 항목 정의에는 조건식의 왼쪽 피연산자가 포함됩니다. Case Edge에는 조건 표현식에서 사용되는 다른 오른쪽 피연산자인 값 정의가 포함됩니다. 예를 들어, 표현식(status == "gold")에서 항목 정의는 "status"이며 "gold"는 Case Edge에서의 값 정의입니다. 조건 노드의 모든 Case Edge의 경우, 항목 정의를 공유하며 값만 다릅니다(TreeConditionValueDefinition). 예제를 이용하여 계속하면, 조건 노드의 다른 Case Edge는 값 "silver"를 가질 수 있습니다. 이것 역시 표현식에서 사용됩니다(status == "silver"). 이 동작의 유일한 예외는 조건 노드에 대해 otherwise가 정의된 경우입니다. otherwise 절 이용 시, 조건 노드 내의 다른 모든 Case Edge가 거짓으로 평가되면 사용되는 값 정의는 없습니다. otherwise가 Case Edge가 아니면, 검색될 수 있는 TreeNode가 있습니다.



항목 정의의 경우, 사용자 프리젠테이션이 검색되고 클라이언트 응용프로그램에서 사용될 수 있습니다. 항목 정의의 프리젠테이션은 보통 왼쪽 피연산자만의 표시이며(예제의 상태) 플레이스홀더를 포함하지 않습니다. Case Edge의 경우, 값 정의를 위해 템플리트를 사용할 수 있습니다(TreeConditionValueTemplate). 템플리트 값 정의 인스턴스(TemplateInstanceExpression)는 실행에 사용되는 매개변수 값을 보유하며 수정될 수 있습니다. 템플리트를 이용하여 정의되지 않은 TreeConditionValueDefinition의 값 템플리트 정의를 검색하기 위한 시도가 있는 경우, 널값이 리턴됩니다. 값 조건을 정의하기 위해 템플리트를 사용하지 않은 경우, 사용자 프리젠테이션이 작성 시간에 지정된 경우 여전히 클라이언트 응용프로그램에서 검색 및 사용될 수 있습니다.

TreeBlock 클래스는 다음을 지원하는 메소드를 제공합니다.

- 트리의 루트 노드 검색
- 트리 블록의 조건 항목 정의 검색
- 트리 블록의 조치 항목 정의 검색

트리의 루트 노드는 TreeNode 유형이며 여기에서 의사결정 테이블의 탐색이 일어날 수 있습니다. TreeNode 클래스는 다음을 지원하는 메소드를 제공합니다.

- 노드가 otherwise 절인지 여부 판별
- 현재 트리 노드(조건 또는 조치 노드)의 상위 노드 검색
- 현재 트리 노드를 포함하는 트리의 루트 노드 검색

ConditionNode 클래스는 다음을 지원하는 메소드를 제공합니다.

- Case Edge 검색
- 항목 정의 검색
- otherwise 케이스 검색
- 조건 노드에 대한 Case Edge의 값 조건에 대한 템플리트 검색
- 템플리트에 근거한 조건 값을 노드에 추가
- 템플리트에 근거한 조건 값 제거

CaseEdge 클래스는 다음을 지원하는 메소드를 제공합니다.

- 값 정의에 사용 가능한 값 템플리트의 목록 검색
- 하위 노드 검색(조건 또는 조치 노드)
- 값 정의와 연관된 템플리트 정의의 인스턴스 검색
- 템플리트를 검색하지 않고 직접 값 정의 검색
- 특정 템플리트 인스턴스 정의를 사용하도록 정의 값 설정

TreeConditionTermDefinition 클래스는 다음을 지원하는 메소드를 제공합니다.

- 조건 노드용으로 정의된 값 정의 템플리트 검색
- 조건 항목의 사용자 프리젠테이션 검색

TreeConditionDefinition 클래스는 다음을 지원하는 메소드를 제공합니다.

- 조건 노드의 항목 정의 검색
- 모든 Case Edge에서 조건 노드의 조건 값 정의 검색
- 방위 검색(행 또는 열)

TreeConditionValueDefinition 클래스는 다음을 지원하는 메소드를 제공합니다.

- 값에 정의된 특정 템플리트 인스턴스 표현식 검색
- 사용자 검색

Template 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트의 시스템 ID 검색
- 템플리트의 이름 검색
- 템플리트에 정의된 매개변수 검색
- 템플리트의 프리젠테이션 검색

TreeConditionValueTemplate 클래스는 다음을 지원하는 메소드를 제공합니다.

- 새 템플리트 조건 값 인스턴스 작성

TemplateInstanceExpression 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트 인스턴스의 매개변수 검색
- 인스턴스를 정의하는 데 사용되는 템플리트(의사결정 테이블에 있는 Case Edge의 경우에서 TreeConditionValueTemplate) 검색

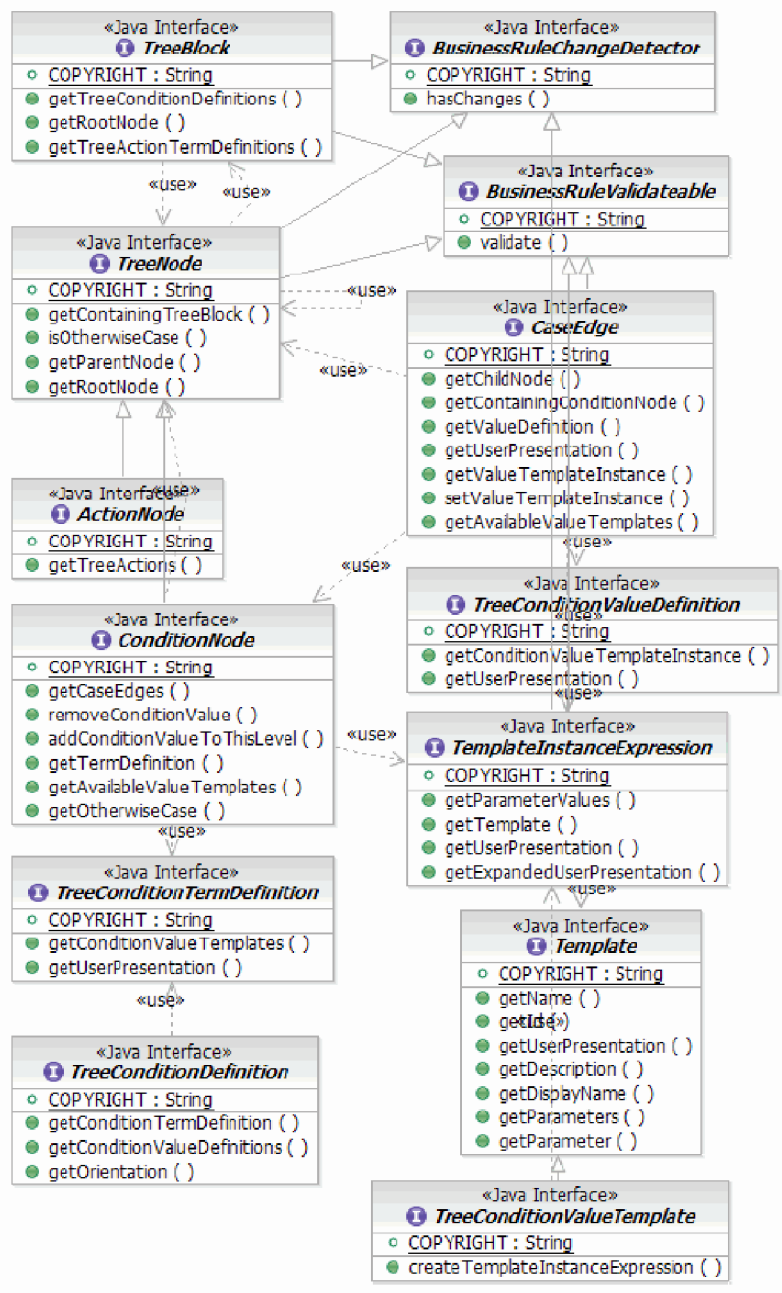


그림 65. TreeNode 및 관련 클래스의 클래스 다이어그램

새 Case Edge가 조건 노드에 추가될 때, 새 Case Edge는 템플리트를 사용하여 값을 정의해야 합니다. 예를 들어, "bronze"의 새 Case Edge가 'status' 검사를 위해 추가된 경우 해당 템플리트(`TreeConditionValueTemplate`)를 사용하여 새 TemplateInstanceExpression을 작성하고 매개변수 값을 "bronze"로 설정해야 합니다.

새 Case Edge가 추가되면, 자동으로 하위 조건 노드를 추가합니다. 이 하위 조건 노드에는 동일한 레벨에서 조건 노드에 정의된 Case Edge 정의에 근거한 Case Edge가 포함되어 있습니다. 템플리트나 하드 코딩된 값이 Case Edge에 사용되면 해당 템플리트나 값은 하위 조건 노드의 Case Edge에서도 사용됩니다. 자동으로 추가된 하위 조건 노드는 자동으로 작성된 고유한 하위 조건 노드를 갖습니다. 이러한 하위 조건 노드에는 하위 조건 노드가 포함되며 모든 레벨의 조건 노드가 재작성될 때까지 이러한 포함 관계가 유지됩니다.

조건 노드 이외에, 의사결정 테이블 및 특히 트리 블록도 조치 노드(ActionNode)의 레벨을 포함합니다. 조치 노드는 리프 노드이며 조건 노드의 분기 및 Case Edge의 끝에 있습니다. Case Edge의 모든 조건 값이 참으로 해석되면, 조치 노드가 접근됩니다. 조치 노드에는 최소 하나의 조치(TreeAction)가 정의됩니다. 조치의 경우, 항목 정의와 값 정의가 있습니다. 조건 노드에서와 같이 항목 정의(TreeActionTermDefinition)는 표현식의 왼쪽이며 값 정의(TemplateInstanceExpression)는 표현식의 오른쪽입니다. 예를 들어, status를 검사 중인 여러 조건 노드의 경우 discount를 정의하는 조치가 있을 수 있습니다. 조건이 (status == "gold")이면, 조치는 (discountValue = 0.90)일 수 있습니다. 조치의 경우 "discountValue"는 항목 정의이며 "= 0.90"은 값 정의입니다.

트리 조치의 항목 정의는 다른 조치 노드의 다른 트리 조치와 공유됩니다. Case Edge의 모든 분기는 조치에 접근하므로, 동일한 항목 정의가 사용됩니다. 그러나 값 정의는 트리 조치 및 조치 노드마다 다를 수 있습니다. 예를 들어, "gold"의 status는 "0.90"일 수 있지만, "silver"의 status에 대한 "discountValue"는 "0.95"일 수 있습니다.

조치 노드는 별도의 항목 정의와 별도의 값 정의가 있는 다중 트리 조치를 가질 수 있습니다. 예를 들어, discount가 임대하는 차에 대한 것으로 판별되는 경우, discountValue를 설정하는 것 이외에 차의 특정 레벨을 지정하려고 할 수 있습니다. status가 "gold"인 경우 "carSize" 항목을 "full size"로 설정하고 "discountValue"를 "0.90"으로 설정하도록 다른 트리 조치를 작성할 수 있습니다.

트리 조치의 값 정의는 템플리트(TreeActionValueTemplate)에서 작성될 수 있습니다. 템플리트 정의는 표현식의 매개변수가 있는 표현식(TemplateInstanceExpression)을 포함합니다.

매개변수 변경 이외에, 전체 값 정의는 트리 조치용으로 정의된 다른 템플리트를 이용하여 작성된 새 값 정의 인스턴스를 이용하여 수정될 수 있습니다.

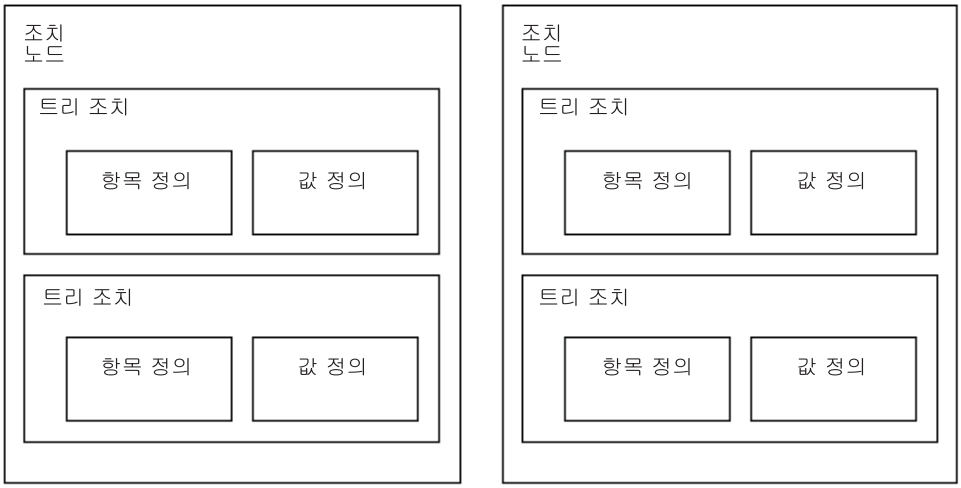
값 정의는 템플리트로 작성된 경우에만 변경될 수 있습니다. 클라이언트 응용프로그램의 경우, 사용자 프리젠테이션이 작성 시간에 지정된 경우 디스플레이에서 사용될 수 있습니다.

트리 조치의 항목 정의에서, 사용자 프리젠테이션이 지정되면 클라이언트 응용프로그램에 의해서도 사용될 수 있습니다.

새 Case Edge가 조건 노드에 추가되고 다른 하위 조건 노드가 작성되면, 조치 노드도 작성됩니다. 해당 레벨에 대해 이미 정의된 Case Edge의 정의에 근거하여 작성된 하위 조건 노드 및 Case Edge와 달리 조치 노드는 자동으로 기존 설계를 상속하지 않습니다. 비어 있는 플레이스홀더 TreeActions만이 조치 노드에서 작성됩니다. 템플리트(TreeActionValueTemplate)는 조치 노드의 최소 하나의 항목 정의에 대한 TemplateInstanceExpression을 작성하여 조치 정의를 완료하는 데 사용되어야 합니다. 트리 조치가 TemplateInstanceExpression으로 설정될 때까지, 트리 조치는 사용자 프리젠테이션 값 및 템플리트 인스턴스 값에 대해 지정된 널값을 가집니다.

새 ActionNodes 결과를 가져오는 새 조건 작성 시, 조치 노드가 즉시 상위 조건 노드의 기존 조치 오른쪽에 추가됩니다. 예를 들어, "ruby"의 status가 의사결정 테이블에 추가되고 특정 discount를 가져야 하는 경우, status를 검사하는 조건은 "gold", "silver" 및 "bronze"의 오른쪽에 추가됩니다. "ruby"의 discount 조치 노드는 "gold", "silver" 및 "bronze" Case Edge에 해당하는 조치 노드의 오른쪽에 추가됩니다.

조치 노드의 새 트리 조치 설정 시, 가장 하위 Case Edge의 가장 오른쪽 조치에 보이는 알고리즘은 비어 있는 트리 조치를 이용하는 조치 노드를 리턴합니다. 트리 조치가 사용자 프리젠테이션 값 및 템플리트 인스턴스 값에 대해 널값을 가지는지 검사될 수도 있습니다. 트리 조치를 얻고 나면, TreeActionValueTemplate의 올바른 인스턴스를 이용하여 설정될 수 있습니다.



ActionNode 클래스는 다음을 지원하는 메소드를 제공합니다.

- 정의된 트리 조치 목록 검색

TreeAction 클래스는 다음을 지원하는 메소드를 제공합니다.

- 트리 조치에 정의된 사용 가능한 값 템플리트 목록 검색
- 항목 정의 검색

- 트리 조치에 정의된 값 템플리트 인스턴스 검색
- 값 템플리트가 사용되지 않은 경우 값의 사용자 프리젠테이션 검색
- 조치가 SCA 서비스 호출인지 검사(isValueNotApplicable 메소드)
- 새 인스턴스를 이용하여 값 템플리트 인스턴스 바꾸기

TreeActionTermDefinition 클래스는 다음을 지원하는 메소드를 제공합니다.

- 항목 값 정의의 사용자 프리젠테이션 검색
- 트리 조치에 사용 가능한 값 템플리트 목록 검색
- 조치가 SCA 서비스 호출인지 검사(isTermNotApplicable 메소드)

Template 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트의 시스템 ID 검색
- 템플리트의 이름 검색
- 템플리트에 정의된 매개변수 검색
- 템플리트의 프리젠테이션 검색

TreeActionValueTemplate 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트 정의에서 새 값 템플리트 인스턴스 작성

TemplateInstanceExpression 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트 인스턴스의 매개변수 검색
- 인스턴스를 정의하는 데 사용되는 템플리트(의사결정 테이블에 있는 트리 조치의 경우에서 템플리트(TreeActionValueTemplate) 검색

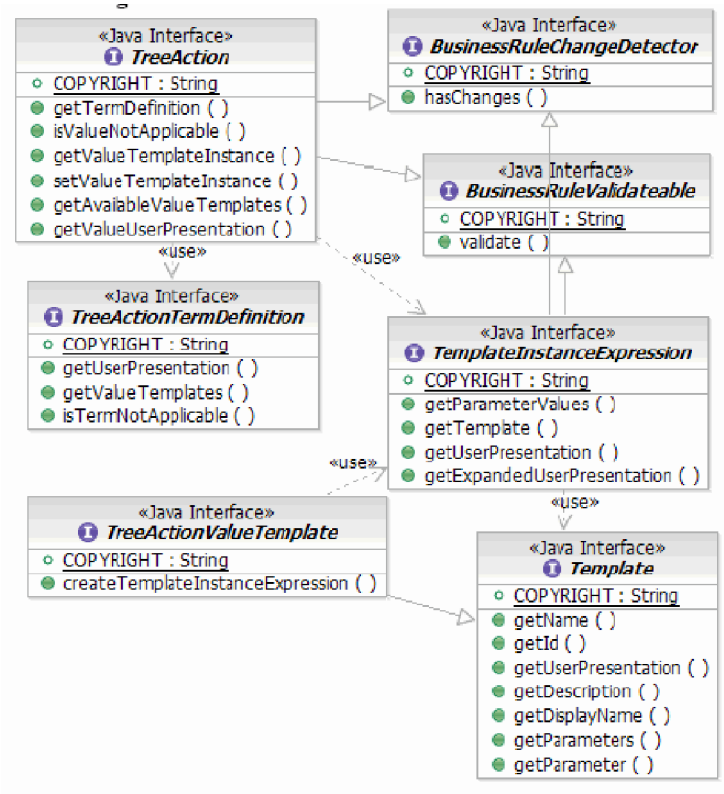


그림 66. TreeAction 및 관련 클래스의 클래스 다이어그램

의사결정 테이블의 init 규칙 정의는 규칙 세트의 규칙과 동일한 구조를 따릅니다. init 규칙은 템플리트(DecisionTableRuleTemplate)를 이용하여 정의될 수 있습니다.

init 규칙이 작성 시간에 작성되지 않은 경우, 규칙이 전개되고 나면 추가될 수 없습니다.

Rule 클래스는 다음을 지원하는 메소드를 제공합니다.

- 규칙의 이름 검색
- 규칙의 사용자 프리젠테이션 검색
- 채워진 규칙에 대해 다른 매개변수를 이용하는 규칙의 사용자 프리젠테이션 검색

DecisionTableRule 클래스는 다음을 지원하는 메소드를 제공합니다.

- init 규칙을 포함하는 트리 블록 검색

DecisionTableRuleTemplate 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트를 포함하는 의사결정 테이블 검색

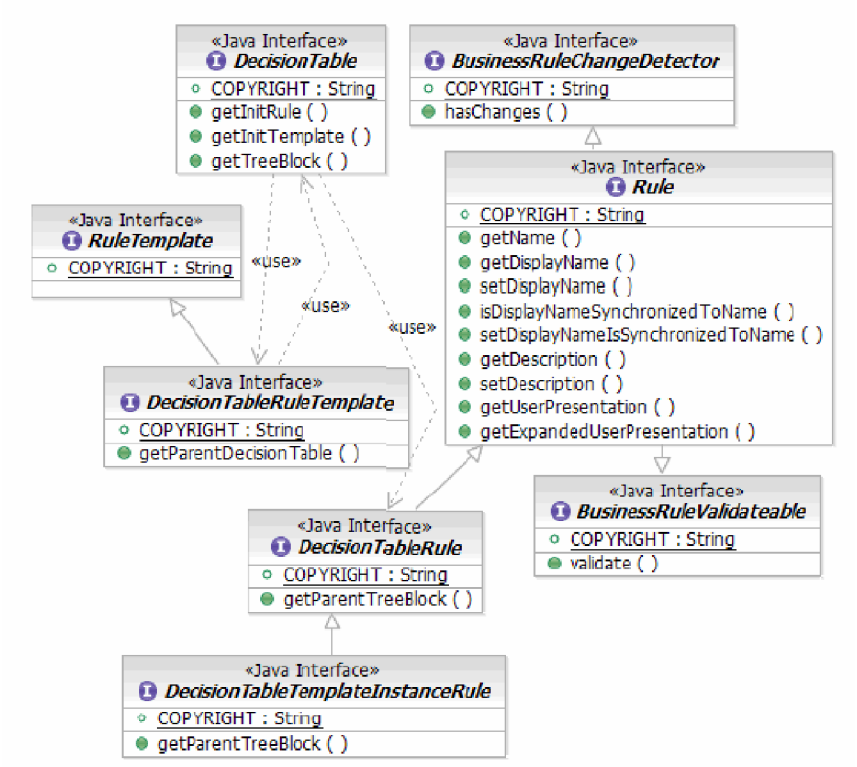


그림 67. DecisionTableRule 및 관련 클래스의 클래스 다이어그램

템플리트 및 매개변수

규칙 세트 및 의사결정 테이블의 템플리트는 공통 정의에 기반합니다. 템플리트에는 매개변수와 사용자 프리젠테이션이 있습니다. 규칙을 전개하고 난 후에는 규칙을 변경할 수 있도록 템플리트 매개변수 값을 정의합니다.

사용자 프리젠테이션 값은 사용자가 친숙한 방식으로 규칙 및 여러 매개변수를 표시하기 위해 사용될 수 있는 문자열 값을 제공합니다. 문자열인 사용자 프리젠테이션에는 여러 매개변수 값을 바꾸고 제대로 표시하는 플레이스홀더가 있습니다. 플레이스홀더는 {<parameter index>} 형식으로 되어 있습니다. 예를 들어, init 규칙은 "기준 할인은 {0} %입니다 "이며, {0} 플레이스홀더는 매개변수 값으로 대체될 수 있습니다. 규칙 또는 템플리트 정의의 프리젠테이션 문자열은 변경될 수 없습니다. 그러나 플레이스홀더 값은 템플리트의 정의마다 클라이언트 응용프로그램의 매개변수 값으로 수정될 수 있습니다. 다른 템플리트는 편의 메소드(`getExpandedUserPresentation`)를 포함하며 이는 모든 매개변수 값이 문자열에서 제대로 위치한 문자열을 리턴합니다.

모든 매개변수 값은 특정 데이터 유형을 가지지만, 매개변수 값 검색 및 설정 시 문자열 오브젝트가 사용됩니다. 값을 사용자 프리젠테이션으로 대체하고 매개변수를 새 값으로 설정할 때 문자열로서 매개변수 값을 취급할 수 있습니다. 실행 시간에 제대로 규칙을 발행하기 위해 매개변수가 런타임 시 올바른 데이터 유형으로 변환됩니다. 유효성

검증 동안 매개변수 값을 데이터 유형과 비교하여 올바른지 확인합니다. 예를 들어, 매개변수가 boolean 유형이며 "T"로 설정되면, 유효성 검증 시 이 값을 인식하지 않으며 문제점을 리턴합니다.

템플리트 정의에서, 매개변수 값은 제한조건으로 제한됩니다. 제한조건은 범위나 열거로 정의될 수 있습니다. 규칙이 유효성 검증되면 매개변수의 제한조건이 시행됩니다. 값 정의를 정의하기 위해 템플리트를 사용하지 않은 경우, 사용자 프리젠테이션만이 사용 가능합니다. 값 정의는 템플리트 및 사용자 프리젠테이션 둘 다를 가질 수 없습니다. 템플리트를 사용해야 하는 경우, 템플리트 정의의 프리젠테이션이 사용 가능한 유일한 프리젠테이션입니다.

Template 클래스는 다음을 지원하는 메소드를 제공합니다.

- 템플리트 ID 검색
- 이름 검색
- 매개변수 검색
- 사용자 프리젠테이션 검색

Parameter 클래스는 다음을 지원하는 메소드를 제공합니다.

- 매개변수 이름 검색
- 매개변수 데이터 유형 검색
- 매개변수의 제한조건 검색
- 매개변수를 정의하는 템플리트 검색
- 매개변수 값 작성

ParameterValue 클래스는 다음을 지원하는 메소드를 제공합니다.

- 매개변수 이름 검색
- 매개변수 값 검색
- 매개변수 값 설정

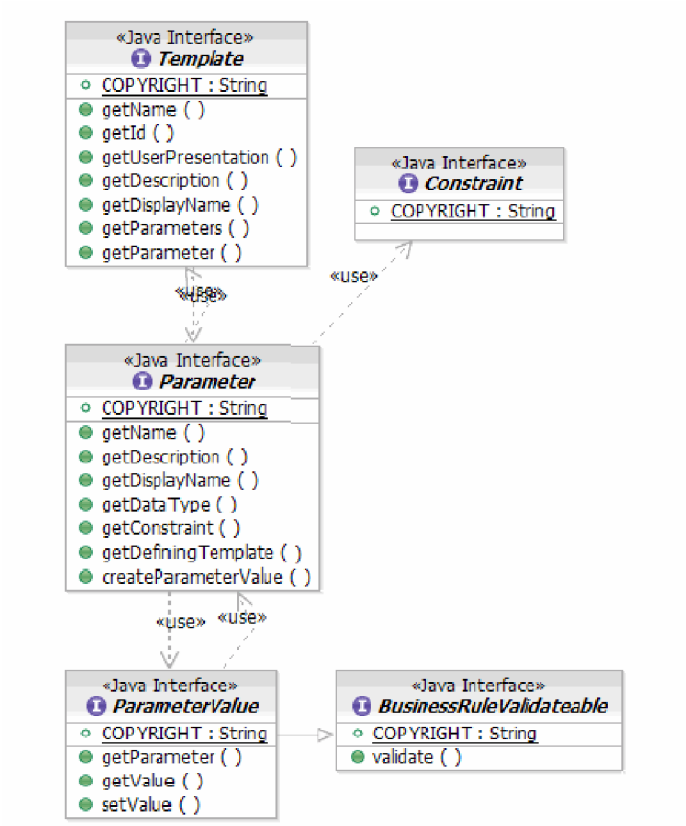


그림 68. 템플리트 및 매개변수와 관련 클래스의 클래스 다이어그램

유효성 검증

아티팩트를 공개하기 전에 정확성과 완전성을 확인하도록 아티팩트에 허용하는 유효성 검증 메소드가 여러 기본 오브젝트에 있습니다.

API 클래스를 통해 변경 시 발생하는 유효성 검증은 serviceDeploy 동안 발생하거나 WebSphere Integration Developer에서 아티팩트를 편집할 때 발생하는 전체 유효성 검증에 대해 유일하게 적합한 서브세트입니다. 이것은 런타임 시 편집 가능한 사항을 제한할 때 비즈니스 규칙 그룹에 이미 있는 제한조건으로 인한 것입니다. 클래스의 사용자는 필요할 때마다 비즈니스 규칙 그룹 선택사항 테이블, 규칙 세트 또는 의사결정 테이블을 유효성 검증할 수 있습니다(규칙 그룹 컴포넌트 자체는 런타임 시 편집 가능하지 않습니다). 비즈니스 규칙 그룹이 공개되면 규칙 그룹 선택사항 테이블, 규칙 세트 및 의사결정 테이블은 저장소로 공개되기 전에 유효성 검증됩니다.

아티팩트가 유효하지 않은 경우, 유효성 검증 문제점 목록과 함께 ValidationException이 발생합니다. 여러 유효성 검증 문제점이 예외 처리 절에 설명되어 있습니다.

변경사항 추적

모든 오브젝트에 대해, 오브젝트와 오브젝트를 포함하는 것에 대한 수정사항이 있는지 여부를 확인하는 hasChanges 메소드가 사용 가능합니다.

이 메소드는 변경사항을 확인하고 변경된 항목이 있는 경우 비즈니스 규칙 그룹을 공개만 할 수 있습니다.

BusinessRuleManager

BusinessRuleManager 클래스는 비즈니스 규칙 그룹, 규칙 그룹 및 의사결정 테이블을 이용하여 작업하는 기본 클래스입니다.

BusinessRuleManager에는 이름, 대상 네임스페이스 또는 사용자 정의 특성에 의해 비즈니스 규칙을 검색하는 메소드가 있습니다. 또한 비즈니스 규칙 그룹, 규칙 세트 또는 의사결정 테이블에 작성한 변경사항을 공개하는 메소드가 있습니다.

BusinessRuleManager 클래스는 다음을 지원하는 메소드를 제공합니다.

- 모든 비즈니스 규칙 그룹 검색
- 특정 대상 네임스페이스의 비즈니스 규칙 그룹 검색
- 특정 이름의 비즈니스 규칙 그룹 검색
- 특정 이름 및 대상 네임스페이스의 비즈니스 규칙 그룹 검색
- 특정 특성이 들어 있는 비즈니스 규칙 그룹 검색
- 특정 특성이 들어 있는 비즈니스 규칙 그룹 검색
- 비즈니스 규칙 그룹 공개

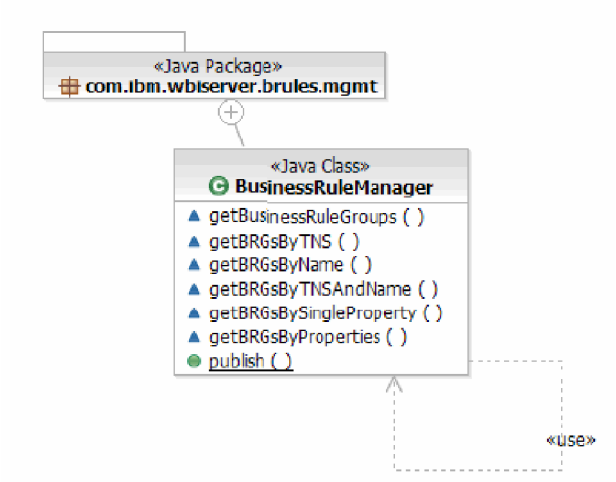


그림 69. BusinessRuleManager 및 패키지의 클래스 다이어그램

규칙 그룹 컴포넌트 조회

규칙 그룹 컴포넌트는 클래스에서 리턴되는 비즈니스 규칙 그룹의 목록을 좁히는 데 사용되는 사용자 정의 특성(이름/값 쌍)을 가질 수 있습니다. 조회 및 임의 조합에서 사용될 수 있는 필드는 다음과 같습니다.

- 비즈니스 규칙 그룹 컴포넌트 대상 네임스페이스
- 비즈니스 규칙 그룹 컴포넌트 이름
- 특성 이름
- 특성 값

각 특성은 비즈니스 규칙 그룹 컴포넌트에 한 번만 정의될 수 있습니다.

이 클래스에서 지원하는 조회 기능은 전체 SQL 언어의 축소 서브세트입니다. 사용자는 SQL 문을 제공하지 않지만, 노드 양식으로 다중-특성 조회의 정보가 있는 트리 구조나 단일 특성의 매개변수로 값을 제공합니다. 모두 `QueryNode` 인터페이스를 구현하는 논리 연산자 노드 및 특성 조회 노드가 있습니다. 논리 연산자 노드는 부울 연산자(AND, OR, NOT)를 지정합니다. 이것은 `QueryNodeFactory`를 통해 작성됩니다. 이러한 논리 연산자 노드의 작성의 일부로 왼쪽 및 오른쪽 연산자는 추가 `QueryNode` 클래스를 이용하여 지정되어야 합니다. 이 노드는 특성 조회 노드나 다른 논리 연산자 노드 중 하나일 수 있습니다. 특성 조회 노드가 전달되면, 특성 이름, 값 및 연산자(EQUAL (==), NOT_EQUAL (!=), LIKE 또는 NOTLIKE)를 포함합니다. 전체 `QueryNode`는 클래스에 의해 구문 분석되며 조회는 지속적 기억장치의 기초 데이터에서 수행됩니다.

와일드카드 검색은 LIKE 및 NOTLIKE 연산자가 사용될 때 지원됩니다. %' 및 '_' 문자 둘 다는 와일드카드 검색에서 지원됩니다. %' 문자는 알려지지 않고 검색 시 고려되지 않는 무한 개의 문자가 있을 때 사용됩니다. 예를 들어, 부서 이름과 "North"로 시작하는 값의 특성을 갖는 모든 비즈니스 규칙 그룹에 대한 검색이 수행된 경우, 값은 "North%"로 지정됩니다. 다른 예로, 모든 부서가 "Region"으로 끝나는 값을 갖는다고 가정하십시오. 값은 "%Region"입니다. '%' 문자는 문자열의 중간에도 사용할 수 있습니다. 예를 들어, 값이 "NorthCentralRegion", "NorthEastRegion" 및 "NorthWestRegion"인 특성을 갖는 비즈니스 규칙 그룹이 있는 경우, "North%Region"이 지정될 수 있습니다.

'_' 문자는 알려지지 않은 단일 문자가 있거나 검색 시 고려되지 않을 때 사용됩니다. 예를 들어, 값이 "Dept1North", "Dept2North", "Dept3North" 및 "Dept4North"인 부서 특성을 갖는 모든 비즈니스 규칙 그룹이 검색되는 경우, "Dept_North"가 지정될 수 있으며 이러한 특성을 갖는 4개의 비즈니스 규칙 그룹 모두가 리턴됩니다. '_' 문자는 무시할 단일 문자를 나타내는 각 인스턴스의 검색 값에서 여러 번 사용될 수 있습니다. '_' 문자는 값의 처음이나 끝에서 사용될 수 있습니다. 예를 들어, 두 개의 문자가 값에서 무시된 경우, 두 개의 '_'가 "Dept__outh"와 같이 사용될 수 있습니다.

'%' 및 '_'를 와일드카드가 아닌 리터럴 문자로 취급하려면 '₩' 이스케이프 문자가 '%'나 '_' 앞에 지정되어야 합니다. 예를 들어, 특성 이름이 "%Discount"인 경우, 조회에서 사용하려면 "₩%Discount"를 지정해야 합니다. '₩' 문자가 리터럴 문자로 사용되는 경우, "Orders₩₩Customer"와 같이 다른 '₩' 이스케이프 문자가 사용되어야 합니다. 단일 '₩' 문자 다음에 '%', '_' 또는 '₩'가 없는 경우, IllegalArgumentException이 발생합니다.

와일드카드 문자는 왼쪽 연산자(특성 값)에서만 사용될 수 있습니다. 와일드카드는 특성 이름에서 사용될 수 없습니다.

특성과 일치하지 않는 값을 검색하거나 특정 특성 값을 검색하는 중, 특성이 없는 경우 아티팩트가 검색 고려에서 무시됩니다. 예를 들어, 세 개의 비즈니스 규칙 그룹(A, B 및 C)이 있으며 오직 두 개만(A 및 B) 특성 이름이 "Department"이며 다른 값(각각 "Accounting" 및 "Shipping")을 갖는 경우 "Accounting"의 "Department" 특성을 갖지 않는 모든 비즈니스 규칙 그룹의 검색에서 "Department" 특성이 정의되었지만 "Accounting"과 같지 않은 비즈니스 규칙 그룹만 리턴합니다(비즈니스 규칙 그룹 B). "Department" 특성을 갖지 않는 비즈니스 규칙 그룹은(C) 정의된 특성이 없으므로 리턴되지 않습니다.

검색을 위한 특성 사용 시, 이름과 아티팩트의 네임스페이스에 근거해 검색하는 데 사용될 수 있는 두 개의 특정 특성 이름 *IBMSystemName* 및 *IBMSystemTargetNameSpace*가 있습니다. 또한 이들 값은 `getName` 및 `getTargetNameSpace` 메소드를 이용하여 검색될 수 있습니다.

클래스는 조회를 위한 다음 메소드를 지원합니다.

```
List getBRGsByTNS (string tNSName, Operator op, int skip, int threshold)
List getBRGByName(string Name, Operator op, int skip, int threshold)
List getBRGsByTNSAndName (string tNSName, Operator, tNSOp, string
 name, Operator nameOp, int skip, int threshold)
List getBRGsBySingleProperty (string propertyName, string propertyValue,
 Operator op, int skip, int threshold)
List getBRGsByProperties (QueryNode queryTree, int skip, int threshold)
```

'skip' 및 'threshold' 매개변수는 지정된 임계값까지 부분 결과 목록을 페치하는 기능을 사용자에게 제공합니다. 이들 매개변수 둘 다에서 0 값은 전체 결과 목록을 리턴합니다. 커서는 조회 호출의 결과 세트에서 유지되지 않습니다. 건너뛰기 값이 사용되는 경우, 차후 요청이 이전 결과 세트에 있던 비즈니스 규칙 그룹을 리턴하는 것과 같이 결과 세트에 추가나 삭제가 수행되는 것이 가능합니다.

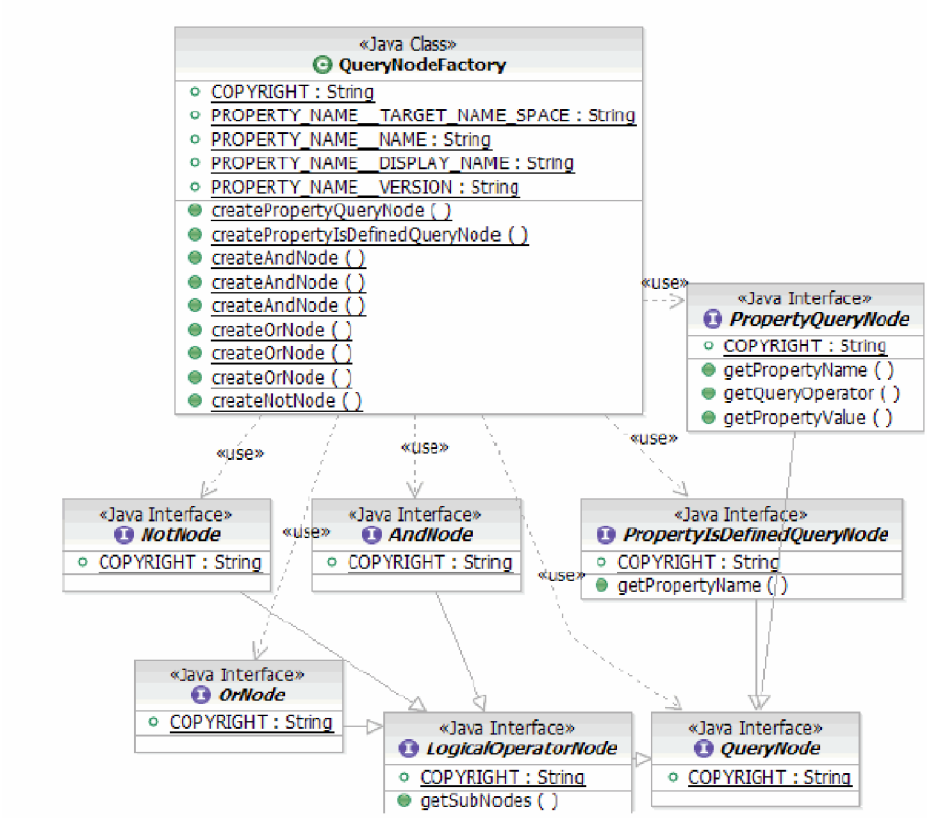


그림 70. QueryNodeFactory 및 관련 클래스의 클래스 다이어그램

트리의 노드는 부울 연산자, 와일드카드(% 및 이스케이프) 및 특성/값 쌍을 사용하여 검색 표현식을 지정할 수 있게 합니다. 연산자만이 값에 유효하며, 특성의 연산자는 항상 같음(==)입니다.

공개

비즈니스 규칙 공개는 비즈니스 규칙 그룹 컴포넌트 레벨에서 수행됩니다. 사용자는 1...n 비즈니스 규칙 그룹 컴포넌트를 공개할 수 있습니다. 공개 조작 이전에, 비즈니스 규칙 그룹과 비즈니스 규칙 그룹에 포함된 여러 오브젝트(조작 선택 테이블, 규칙 세트, 의사 결정 테이블 등)에 대해 유효성 검증 조치가 수행됩니다. 각 공개 요청은 단일 트랜잭션 내에서 일어나며 유효성 검증 또는 데이터베이스 공개 중 예외가 발생하면 트랜잭션이 롤백되며 비즈니스 규칙 그룹에 대한 어떤 변경사항도 저장소에 공개되지 않습니다. 이것은 단일 컴포넌트 내에서 변경사항이 서로 종속되게 하거나(예: 조작 선택 테이블 및 규칙 세트) 컴포넌트 간의 종속성이 하나의 원자 조작 내에서 일어나게 합니다.

공개 시간에 공개되는 항목이 다른 트랜잭션에 의해 변경되지 않았음을 확인하는 검사가 이루어집니다. 충돌의 가능성을 줄이기 위해, 공개 메소드는 변경 여부에 관계없이 모든 아티팩트를 공개하거나 비즈니스 규칙 그룹 내에서 변경된 아티팩트만을 공개할지 여부를 사용자가 선택하게 합니다. 기본값 동작은 모든 아티팩트를 공개하는 것입니다. 옵션이 모든 아티팩트를 공개하도록 설정되고 그 사이에 다른 트랜잭션이 아티팩트를

변경한 경우, ChangeConflictException이 발생합니다. 변경한 아티팩트만을 공개하도록 지정하면 충돌을 줄입니다. 변경된 아티팩트만 공개하면 비즈니스 규칙 그룹 내에서 호환 가능하지 않은 변경사항을 가져올 수 있는 비즈니스 규칙 그룹(예: 두 개의 규칙 세트) 내에서 두 사용자가 두 개의 다른 아타팩트의 저장소로 변경사항을 공개하는 결과를 가져올 수 있습니다. 이러한 잠재적인 상황 때문에, 이 옵션은 주의해서 사용해야 합니다.

예외 처리

유효성 검증이 아티팩트에서 호출되거나 아티팩트가 공개되면 예외가 발생할 수 있습니다. 유효성 검증 오류가 발생하면, 문제점 목록과 함께 ValidationException이 발생합니다. 동일한 아티팩트를 공개하는 다른 트랜잭션으로 인해 공개 중 문제점이 발생하면, ChangeConflictException이 발생합니다. 아티팩트 변경 시 또 다른 트랜잭션이 검출되면, ChangeConflictException 예외가 발생합니다.

시스템 특성 이름이 중복된 특성을 변경하려고 시도하면 발생되는 SystemPropertyNotChangeableException이 있습니다. 시스템 특성은 변경될 수 없습니다.

아티팩트 공개 중 설정 조작 시도가 일어나면 발생하는 ChangesNotAllowedException이 있습니다.

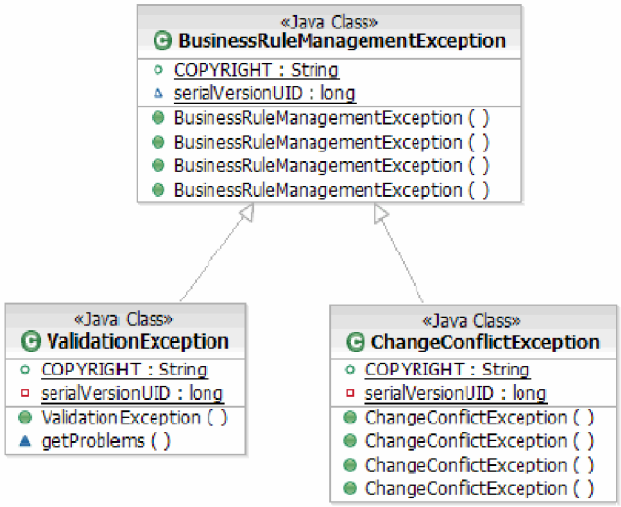


그림 71. BusinessRuleManagementException 및 관련 클래스의 클래스 다이어그램

비즈니스 규칙 그룹 문제점

비즈니스 규칙 그룹이 유효성 검증되거나 비즈니스 규칙 그룹을 공개하려는 시도가 있고 비즈니스 규칙 그룹의 일부가 유효하지 않으면 발생할 수 있는 문제점입니다.

표 23. 비즈니스 규칙 그룹 문제점

예외	설명
ProblemBusRuleNotInAvailTargetList	조작 선택사항 테이블의 기본값 비즈니스 규칙으로 규칙이 지정되었지만 규칙 아티팩트가 해당 조작의 사용 가능한 대상 목록에 없으면 발생하는 문제점. 조작에 사용 가능한 대상 목록의 유효한 비즈니스 규칙이 지정되어야 이 문제점을 피할 수 있습니다.
ProblemDuplicatePropertyName	비즈니스 규칙 그룹의 시스템 또는 사용자 정의 특성이 중복되는 특성을 작성하려고 시도하면 이 문제점이 발생합니다. 이 문제점을 피하려면 고유한 특성 이름을 사용해야 합니다.
ProblemOperationContainsNoTargets	기본값 규칙 대상이나 예정된 규칙 대상 세트가 없는 조작에서 발생하는 문제점. 이 문제점을 피하려면 예정된 시간이나 기본값 중 하나로 최소한 하나의 규칙 대상을 이용하여 조작을 설정해야 합니다.
ProblemOverlappingRanges	조작 선택사항 레코드 시작 날짜 또는 종료 날짜가 또 다른 조작 선택사항 레코드 시작 날짜 및 종료 날짜의 범위와 겹치는 경우 발생하는 문제점. 날짜 범위의 이러한 겹침은 비즈니스 규칙 런타임이 호출할 올바른 규칙 대상을 찾지 못하게 합니다. 조작의 기타 조작 선택사항 레코드의 시작 날짜 또는 종료 날짜를 확인하여 겹침이 없는지 확인하여 이 문제점을 피해야 합니다.
ProblemStartDateAfterEndDate	조작 선택사항 레코드의 시작 날짜가 해당 선택 레코드의 종료 날짜 이후이면 이러한 문제점이 발생합니다. 시작 날짜 또는 종료 날짜가 없는 기본값 레코드를 제외하고 조작 선택사항 레코드에 대해 이러한 문제점이 발생할 수 있습니다. 이 문제점을 피하려면 조작 선택사항 레코드의 종료 날짜 이후의 시작 날짜를 지정하십시오.
ProblemTargetBusRuleNotSet	사용 가능한 대상 규칙 목록에 없는 규칙을 조작 선택사항 레코드가 지정한 경우 발생하는 문제점. 이 문제점을 피하도록 사용 가능한 대상 목록의 규칙을 지정해야 합니다.
ProblemTNSAndNameAlreadyInUse	규칙 세트나 의사결정 테이블에 의해 이미 사용 중인 대상 네임스페이스와 이름으로 새 비즈니스 규칙을 작성하면 발생하는 문제점. 저장소에 저장되는 규칙 아티팩트는 물론 현재 비즈니스 규칙 그룹과 연관된 모든 규칙 세트와 의사결정 테이블에서 검사가 수행됩니다. 이 문제점을 피하려면 다른 대상 네임스페이스나 이름을 사용해야 합니다.
ProblemWrongOperationForOpSelectionRecord	새 조작 선택사항 레코드를 조작 선택사항 레코드 목록에 추가하고 새 레코드의 조작이 목록의 레코드 조작과 일치하지 않으면 발생하는 문제점. 이 문제점을 피하려면 조작 선택사항 레코드 목록 오브젝트에서 `newOperationSelectionRecord` 메소드를 사용하여 새 조작을 작성해야 합니다.

규칙 세트 및 의사결정 테이블 문제점

표 24. 규칙 세트 및 의사결정 테이블 문제점

예외	설명
ProblemInvalidBooleanValue	규칙 세트에 있는 규칙 템플리트의 매개변수나 의사결정 테이블의 조건 값 또는 조치 값이 부울 유형의 매개변수에 대해 "true" 또는 "false"가 아닌 다른 값을 수신했을 때 문제점이 발생합니다. 예를 들어, 잘못된 매개변수 값은 "T" 또는 "F"입니다. 이러한 문제점을 방지하기 위해 부울 유형의 매개변수로 작업할 때 "true" 또는 "false" 값을 사용하십시오.

표 24. 규칙 세트 및 의사결정 테이블 문제점 (계속)

예외	설명
ProblemParmNotDefinedInTemplate	템플리트 매개변수의 값이 지정되고 템플리트의 유효한 매개변수 목록에 매개변수가 정의되지 않으면 발생하는 문제점. 템플리트에서 설정하기 전에 매개변수를 검사해야 합니다. RuleTemplate, TreeActionValueTemplate 또는 TreeConditionValueTemplate 템플리트의 경우 발생할 수 있습니다.
ProblemParmValueListContainsUnexpectedValue	유효한 매개변수가 템플리트를 이용하여 전달되지만, 해당 매개변수에 대해 너무 많은 매개변수가 있을 때 발생하는 문제점. 매개변수 수를 줄여야 합니다. RuleTemplate, TreeActionValueTemplate 또는 TreeConditionValueTemplate 템플리트의 경우 발생할 수 있습니다.
ProblemRuleBlockContainsNoRules	규칙 세트의 규칙 블록에 있는 모든 규칙을 제거하고 규칙 세트의 유효성을 검증하거나 공개하려고 시도할 때 이 문제점이 발생합니다. 규칙 세트의 규칙 블록에는 최소 하나의 규칙이 있어야 합니다.
ProblemTemplateNotAssociatedWithRuleSet	규칙 세트에 규칙을 추가하려고 하며 해당 규칙 세트로 정의되지 않은 템플리트를 이용하여 규칙을 작성한 경우 발생하는 문제점. 새 규칙 작성 시, 규칙 세트에 정의된 템플리트를 사용해야 이 문제점을 피할 수 있습니다.
ProblemRuleNameAlreadyInUse	규칙 세트의 규칙 블록에 규칙을 추가하려고 하며 이 규칙이 해당 규칙 세트의 기존 규칙과 동일한 이름을 갖는 경우 발생하는 문제점. 이 문제점을 피하려면 새 규칙을 추가하기 전에 규칙 이름을 검사해야 합니다.
ProblemTemplateParameterNotSpecified	규칙 세트의 규칙이나 의사결정 테이블의 조치나 조건 값에 해당하는 템플리트를 갱신할 때 매개변수를 포함하지 않을 때 발생하는 문제점. 이 문제점을 피하려면 템플리트의 모든 매개변수를 지정해야 합니다.
ProblemTypeConversionError	템플리트의 매개변수를 적절한 유형으로 변환할 수 없을 때 이 문제점이 발생합니다. 모든 매개변수는 문자열 오브젝트로 취급된 후 매개변수 유형(boolean, byte, short, int, long, float 및 double)으로 변환됩니다. 이 매개변수의 지정된 유형으로 매개변수 값 문자열을 변환할 수 없는 경우, 이 오류가 발생합니다. 이 문제점을 피하려면 매개변수 유형(boolean, byte, short, int, long, float 및 double)으로 변환할 수 있는 문자열을 지정해야 합니다.
ProblemValueViolatesParmConstraints	해당 매개변수의 템플리트 내에서 정의된 값의 범위나 열거 내에 매개변수가 없으면 이 문제점이 발생합니다. 규칙 세트의 규칙 템플리트의 범위나 열거 또는 의사결정 테이블의 조치 값이나 조건 값 템플리트로 제한된 매개변수의 경우 이 문제점이 발생할 수 있습니다. 이 문제점을 피하려면 열거 내에 있는 값을 사용해야 합니다.
ProblemInvalidActionValueTemplate	트리 조치에서 값 정의에 템플리트 인스턴스를 설정하려고 하지만 해당하는 템플리트가 해당 트리 조치에 사용 가능하지 않은 경우 발생하는 문제점. 이 문제점을 피하려면 트리 조치에서 값 정의를 작성하기 위해 올바른 템플리트를 사용하십시오.
ProblemInvalidConditionValueTemplate	Case Edge에서 조건 정의에 템플리트 인스턴스를 설정하려고 하지만 해당하는 템플리트가 해당 Case Edge에 사용 가능하지 않은 경우 발생하는 문제점. 이 문제점을 피하려면 Case Edge에서 조건 정의를 작성하기 위해 올바른 템플리트를 사용하십시오.
ProblemTreeActionIsNull	새 조건 값이 작성되고 조치가 템플리트 인스턴스를 이용하여 설정되지 않으면 이 문제점이 발생합니다. ActionNode에서 템플리트를 사용하여, 새 템플리트 인스턴스를 작성하여 `TreeActions` 목록에 설정하십시오.

권한

클래스는 임의의 권한 레벨을 지원하지 않습니다. 이는 고유한 권한 양식을 추가하기 위해 클래스를 사용하는 클라이언트 응용프로그램에 따라 결정됩니다.

예제

여러 가지 클래스가 비즈니스 규칙 그룹을 검색하고 규칙 세트 및 의사결정 테이블을 수정하는 방법을 보여주는 몇 개의 예제가 제공됩니다. 찾아보고 재사용될 수 있는 WebSphere Integration Developer로 가져올 수 있는 프로젝트 교환 파일(ZIP)에서 예제가 제공됩니다.

프로젝트 교환은 몇 개의 프로젝트를 포함합니다.

- **BRMgmtExamples** – 다양한 예제에서 사용되는 비즈니스 규칙 아티팩트를 이용하는 모듈 프로젝트.
- **BRMgmt** – com.ibm.websphere.sample.brules.mgmt 패키지에 있는 예제를 이용한 Java 프로젝트.
- **BRMgmtDriverWeb** – 샘플 실행을 위해 인터페이스를 이용하는 웹 프로젝트.

WebSphere Process Server로 설치된 이후에 발행될 수 있는 EAR 파일(`BRMgmtExamples.ear`)로도 예제가 제공됩니다. 웹 인터페이스가 예제와 함께 제공됩니다. 아티팩트를 검색하고 수정사항을 작성하고 변경사항을 공개하기 위해 클래스 사용에 초점을 두는 예제의 경우 웹 인터페이스는 단순합니다. 이는 높은 수준의 기능을 수행하는 웹 인터페이스가 됨을 의미하지는 않습니다. 클래스는 튼튼한 인터페이스를 빌드하기 위해 손쉽게 사용되거나 비즈니스 규칙을 수정하는 데 초점을 두는 다른 Java 응용프로그램에서 사용됩니다.

주: WebSphere Process Server V6.1용 비즈니스 규칙 관리 프로그래밍 안내서에서 예제 프로젝트 교환 및 EAR 파일을 다운로드할 수 있습니다.

WebSphere Process Server v6.1에 응용 프로그램을 설치하거나 색인 페이지를 다음에서 액세스할 수 있습니다.

`http://<hostname>:<port>/BRMgmtDriverWeb/`

예제: `http://localhost:9080/BRMgmtDriverWeb/`

예제가 발행되면, 규칙 아티팩트를 변경합니다. 모든 예제가 발행되면, 응용프로그램을 재설치하여 모든 예제의 샘플 결과를 다시 볼 수 있습니다.

웹 브라우저에 표시된 결과를 비롯하여 완료 코드를 사용하여 예제를 자세히 설명합니다.

일반 조작을 수행하기 위해 몇 개의 추가 클래스가 작성되었으며 예제 웹 응용프로그램 내에서 정보 표시를 돕습니다. Formatter 및 RuleArtifactUtility 클래스에 대한 자세한 정보는 부록을 참조하십시오.

이 예제를 완전히 이해하기 위해, WebSphere Integration Developer 내에서 여러 아티팩트를 연구하면 상당히 도움이 됩니다.

예제 1: 모든 비즈니스 규칙 그룹 검색 및 인쇄

이 예제는 모든 비즈니스 규칙 그룹을 검색하며 속성, 특성 및 각 비즈니스 규칙 그룹의 조작을 인쇄합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;
```

비즈니스 규칙 관리 클래스의 경우, com.ibm.wbiserver.brules.mgmt 패키지에서 그러한 클래스를 사용하고 com.ibm.wbiserver.brules 패키지나 다른 패키지에서는 사용하지 마십시오. 이들 다른 패키지는 IBM 내부 클래스용입니다.

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import
com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;

public class Example1 {
static Formatter out = new Formatter();
static public String executeExample1()
{
try
{
out.clear();
```

BusinessRuleManager 클래스는 비즈니스 규칙 그룹을 검색하고 비즈니스 규칙 그룹에 대한 변경사항을 공개하는 기본 클래스입니다. 이것은 규칙 세트와 의사결정 테이블과 같은 모든 규칙 아티팩트를 변경하고 작업하는 것을 포함합니다. 이름과 네임스페이스 및 특성에 의해 특정 비즈니스 규칙 그룹의 검색을 단순화시키는 BusinessRuleManager 클래스에 대한 몇 가지 메소드가 있습니다.

```
// Retrieve all business rule groups
List<BusinessRuleGroup> brgList = BusinessRuleManager
  .getBusinessRuleGroups(0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Iterate through the list of business rule groups
while (iterator.hasNext())
{
```

```
  brg = iterator.next();
  // Output attributes for each business rule group
  out.println1nBold("Business Rule Group");
```

비즈니스 규칙 그룹의 기본 속성을 검색하고 표시할 수 있습니다.

```
out.println("Name: " + brg.getName());
out.println("Namespace: " +
brg.getTargetNameSpace());
out.println("Display Name: " +
brg.getDisplayName());
out.println("Description: " + brg.getDescription());
out.println("Presentation Time zone: "
     + brg.getPresentationTimezone());
out.println("Save Date: " + brg.getSaveDate());
```

비즈니스 규칙 그룹의 특성을 검색하고 수정할 수 있습니다.

```
PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Output property names and values
while (propIterator.hasNext())
{
prop = propIterator.next();
out.println("Property Name: " +
prop.getName());
out.println("Property Value: " +
prop.getValue());
}
```

비즈니스 규칙 그룹의 조작도 사용 가능하며 규칙 세트 및 의사결정 테이블과 같은 비즈니스 규칙 아티팩트를 검색하는 방법입니다.

```
List<Operation> opList = brg.getOperations();

Iteration<Operation> opIterator = opList.iterator();
Operation op = null;
// Output operations for the business rule group
while (opIterator.hasNext())
{
op = opIterator.next();
out.println("Operation: " + op.getName());
}
out.println("");}
} catch (BusinessRuleManagementException e)
{
e.printStackTrace();
out.println(e.getMessage());
}
return out.toString();
}
}
```

예제 1의 웹 브라우저 출력.

예제 **1** 실행

```
비즈니스 규칙 그룹
이름: ApprovalValues
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: ApprovalValues
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: Department
특성 값: Accounting
특성 이름: RuleType
특성 값: regulatory
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: ApprovalValues
특성 이름: IBMSystemDisplayName
특성 값: ApprovalValues
조작: getApprover

비즈니스 규칙 그룹
이름: ConfigurationValues
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: ConfigurationValues
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: Department
특성 값: General
특성 이름: RuleType
특성 값: messages
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: ConfigurationValues
특성 이름: IBMSystemDisplayName
특성 값: ConfigurationValues
조작: getMessages

비즈니스 규칙 그룹
이름: DiscountRules
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: DiscountRules
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: Department
특성 값: Accounting
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: RuleType
특성 값: monetary
특성 이름: IBMSystemTargetNameSpace
```

```
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: DiscountRules
특성 이름: IBMSystemDisplayName
특성 값: DiscountRules
조작: calculateOrderDiscount
조작: calculateShippingDiscount
```

예제 2: 비즈니스 규칙 그룹, 규칙 세트 및 의사결정 테이블 검색 및 인쇄

예제 1의 기능 이외에, 이 예제는 각 조작의 선택사항 테이블을 인쇄한 후 기본값 비즈니스 규칙 대상(규칙 세트나 의사결정 테이블 중 하나)과 조작을 위해 예정된 다른 비즈니스 규칙을 출력합니다. 규칙 세트와 의사결정 테이블 둘 다를 출력합니다.

예제의 대부분은 동일하지만 참조를 위해 제공됩니다.

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
public class Example2
{
 status Formatter out = new Formatter();
 static public String executeExample2()
 {
  try
  {
    out.clear();
```

이 예제의 이름을 사용하여 특정한 비즈니스 규칙 그룹이 검색됩니다.

```
  // Retrieve all business rule groups
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByName("DiscountRules",
    QueryOperator.EQUAL, 0, 0);

  Iterator<BusinessRuleGroup> iterator = brgList.iterator();

  BusinessRuleGroup brg = null;
  // Iterate through the list of business rule groups
  while (iterator.hasNext())
   {
   brg = iterator.next();
   // Output attributes for each business rule group
   out.println1nBold("Business Rule Group");
   out.println("Name: " + brg.getName());
   out.println("Namespace: " +
```

```
 brg.getTargetNameSpace());
out.println("Display Name: " +
 brg.getDisplayName());
out.println("Description: " + brg.getDescription());
out.println("Presentation Time zone: "
 + brg.getPresentationTimezone());
out.println("Save Date: " + brg.getSaveDate());

PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Output property names and values
while (propIterator.hasNext())
{
  prop = propIterator.next();
  out.println("Property Name: " +
  prop.getName());
  out.println("Property Value: " +
  prop.getValue());
}
```

각 조작의 경우, 선택사항 테이블에는 여러 규칙 아티팩트 목록과 이들이 활성화되었을 때의 스케줄이 있습니다. 각 조작에 대해 기본값 비즈니스 규칙이 지정될 수 있습니다. 기본값 비즈니스 규칙이 지정되어야 하거나 예정된 비즈니스 규칙이 있어야 한다는 요구사항은 없지만, 최소한 기본값 비즈니스 규칙이나 하나의 예정된 비즈니스 규칙이 있어야 합니다. 이 지원 때문에, 기본값 비즈니스 규칙을 사용하기 전에 null을 검사하고 OperationSelectionRecordList의 *크기*를 검사하는 것이 가장 좋습니다.

```
List<Operation> opList = brg.getOperations();

Iterator<Operation> opIterator = opList.iterator();
Operation op = null;
out.println("");
out.printlnBold("Operations");
// Output operations for the business rule group
while (opIterator.hasNext())
 {
   op = opIterator.next();
   out.printBold("Operation: ");
   out.println(op.getName());

   // Retrieve the default business rule for the operation
   BusinessRule defaultRule =
   op.getDefaultBusinessRule();
   // If the default rule is found, print out the business rule
   // using the appropriate method for rule type
   if (defaultRule != null)
   {
     out.printlnBold("Default Destination:");
```

기본값 비즈니스 규칙은 RuleSet 또는 DecisionTable 유형이며 규칙 아티팩트를 처리하기 위해 올바른 유형으로 캐스트될 수 있습니다.

```
        if (defaultRule instanceof RuleSet)
           out.println(RuleArtifactUtility.
           intRuleSet(defaultRule));
        else
           out.print(RuleArtifactUtility.
           tDecisionTable(defaultRule));
      }
      OperationSelectionRecordList
      opSelectionRecordList = op
        .getOperationSelectionRecordList()
        ;

      Iterator<OperationSelectionRecord>
      opSelRecordIterator = opSelectionRecordList
        .iterator();
      OperationSelectionRecord record = null;
```

OperationSelectionRecord는 규칙 아티팩트와 규칙 아티팩트가 활성화될 때의 스케줄로 구성됩니다.

```
      while (opSelRecordIterator.hasNext())
      {
        out.println1nBold("Scheduled
        Destination:");
        record = opSelRecordIterator.next();

        out.println("Start Date: " +
        record.getStartDate()
          + " - End Date: " +
          record.getEndDate());
        BusinessRule ruleArtifact = record
          .getBusinessRuleTarget();

        if (ruleArtifact instanceof RuleSet)
          out.println(RuleArtifactUtility.pr
          intRuleSet(ruleArtifact));
        else
          out.print(RuleArtifactUtility.prin
          tDecisionTable(ruleArtifact));
       }
    }
  }
  out.println("");
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
 out.println(e.getMessage());
 return out.toString();
 }
}
```

예제

예제 2의 웹 브라우저 출력.

```
비즈니스 규칙 그룹
이름: DiscountRules
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
```

```
표시 이름: DiscountRules
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: Department
특성 값: Accounting
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: RuleType
특성 값: monetary
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: DiscountRules
특성 이름: IBMSystemDisplayName
특성 값: DiscountRules

조작
조작: calculateOrderDiscount
기본값 대상:
규칙 세트
이름: calculateOrderDiscount
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
규칙: CopyOrder
표시 이름: CopyOrder
설명: null
펼친 사용자 프리젠테이션: null
사용자 프리젠테이션: null
규칙: FreeGiftInitialization
표시 이름: FreeGiftInitialization
설명: null
펼친 사용자 프리젠테이션: Product ID for Free Gift =
5001AE80 Quantity = 1 Cost =
0.0 Description = Free gift for discounted order
사용자 프리젠테이션: Product ID for Free Gift =
{0} Quantity = {1} Cost = {2}
설명 = {3}Parameter Name: param0
매개변수 값: 5001AE80
매개변수 이름: param1
매개변수 값: 1
매개변수 이름: param2
매개변수 값: 0.0
매개변수 이름: param3
매개변수 값: Free gift for discounted order
규칙: Rule1
표시 이름: Rule1
설명: null
펼친 사용자 프리젠테이션: If customer is gold status,
then apply a discount of 20.0
and include a free gift
사용자 프리젠테이션: If customer is {0} status,
then apply a discount of {1} and include a
free gift
매개변수 이름: param0
매개변수 값: gold
매개변수 이름: param1
매개변수 값: 20.0
규칙: Rule2
```

```
표시 이름: Rule2
설명: null
펼친 사용자 프리젠테이션: If customer.status == silver,
then provide a discount of
15.0
사용자 프리젠테이션: If customer.status == {0},
then provide a discount of {1}
매개변수 이름: param0
매개변수 값: silver
매개변수 이름: param1
매개변수 값: 15.0
규칙: Rule3
표시 이름: Rule3
설명: Template for non-gold customers
펼친 사용자 프리젠테이션: If customer.status ==
bronze, then provide a discount of
10.0
사용자 프리젠테이션: If customer.status == {0},
then provide a discount of {1}
매개변수 이름: param0
매개변수 값: bronze
매개변수 이름: param1
매개변수 값: 10.0

조작: calculateShippingDiscount
기본값 대상:
의사결정 테이블
이름: calculateShippingDiscount
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules

Init 규칙: Rule1
표시 이름: Rule1
설명: null
확장된 사용자 프리젠테이션: null
사용자 프리젠테이션: null
```

예제 3: AND를 이용해 다중 특성으로 비즈니스 규칙 그룹 검색

이 예제는 예제 1과 유사하지만, 특성 이름이 Department이고 값이 "accounting", 그리고 특성 이름이 RuleType이고 값이 "regulatory"인 비즈니스 규칙 그룹만을 검색합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example3
```

```
{
static Formatter out = new Formatter();
static public String executeExample3()
{
  try
  {
    out.clear();
```

비즈니스 규칙 그룹의 조회는 트리 구조를 따르는 조회 노드로 구성됩니다. 각 조회 노드에는 왼쪽 항목과 오른쪽 항목 및 조건이 있습니다. 각 항목과 오른쪽 항목은 다른 조회 노드일 수 있습니다. 이 예제에서 비즈니스 규칙 그룹은 두 개의 특성 값의 조합으로 검색됩니다.

```
   // Retrieve business rule groups based on two conditions
   // Create PropertyQueryNodes for each one condition
   PropertyQueryNode propertyNode1 = QueryNodeFactory
     .createPropertyQueryNode("Department",
     QueryOperator.EQUAL,"Accounting");
   PropertyQueryNode propertyNode2 = QueryNodeFactory
     .createPropertyQueryNode("RuleType", QueryOperator.EQUAL,
     "regulatory");
   // Combine the two PropertyQueryNodes with an AND node
   AndNode andNode =
   QueryNodeFactory.createAndNode(propertyNode1, propertyNode2);

   // Use andNode in search for business rule groups
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByProperties(andNode, 0, 0);

   Iterator<BusinessRuleGroup> iterator = brgList.iterator();

   BusinessRuleGroup brg = null;
   // Iterate through the list of business rule groups
   while (iterator.hasNext())
   {
     brg = iterator.next();
     // Output attributes for each business rule group
     out.println1nBold("Business Rule Group");
     out.println("Name: " + brg.getName());
     out.println("Namespace: " +
     brg.getTargetNameSpace());
     out.println("Display Name: " + brg.getDisplayName());
     out.println("Description: " + brg.getDescription());
     out.println("Presentation Time zone: "
     + brg.getPresentationTimezone());
     out.println("Save Date: " + brg.getSaveDate());

     PropertyList propList = brg.getProperties();

     Iterator<Property> propIterator =
     propList.iterator();
     Property prop = null;
     // Output property names and values
     while (propIterator.hasNext())
     {
       prop = propIterator.next();
       out.println("₩t Property Name: " +
```

```
        prop.getName());
        out.println("₩t Property Value: " +
        prop.getValue());
      }
    }
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

예제

예제 3의 웹 브라우저 출력.

example3 실행

```
비즈니스 규칙 그룹
이름: ApprovalValues
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: ApprovalValues
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: Department
특성 값: Accounting
특성 이름: RuleType
특성 값: regulatory
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: ApprovalValues
특성 이름: IBMSystemDisplayName
특성 값: ApprovalValues
```

예제 4: OR을 이용해 다중 특성으로 비즈니스 규칙 그룹 검색

이 예제는 예제 3과 유사하지만; 특성 이름이 Department이고 값이 "accounting", 또는 특성 이름이 RuleType이고 값이 "monetary"인 비즈니스 규칙 그룹만을 검색합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
```

```
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example4
{
static Formatter out = new Formatter();
static public String executeExample4()
{
  try
  {
    out.clear();
```

다른 특성은 조회를 작성하고 다른 비즈니스 규칙 그룹을 리턴합니다.

```
    // Retrieve business rule groups based on two conditions
    // Create PropertyQueryNodes for each one condition
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"Accounting");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
      QueryOperator.EQUAL,"monetary");
    // Combine the two PropertyQueryNodes with an OR node
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1,
      propertyNode2);
    // Use orNode in search for business rule groups
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(orNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Iterate through the list of business rule groups
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Output attributes for each business rule group
      out.println1nBold("Business Rule Group");
      out.println("Name: " + brg.getName());
      out.println("Namespace: " +
      brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());

      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Output property names and values
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
```

```
        out.println("₩t Property Name: " +
        prop.getName());
        out.println("₩t Property Value: " +
        prop.getValue());
      }
      out.println("");
    }
  } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

예제

예제 4의 웹 브라우저 출력.

example4 실행

```
비즈니스 규칙 그룹
이름: ApprovalValues
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: ApprovalValues
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: Department
특성 값: Accounting
특성 이름: RuleType
특성 값: regulatory
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: ApprovalValues
특성 이름: IBMSystemDisplayName
특성 값: ApprovalValues

비즈니스 규칙 그룹
이름: DiscountRules
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: DiscountRules
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: Department
특성 값: Accounting
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: RuleType
특성 값: monetary
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
```

```
특성 이름: IBMSystemName
특성 값: DiscountRules
특성 이름: IBMSystemDisplayName
특성 값: DiscountRules
```

예제 5: 복합 조회를 이용해 비즈니스 규칙 그룹 검색

이 예제는 예제 3 및 4의 조합이며 더 복잡한 조회를 작성하는 방법을 보여줍니다. 이 예제에서 검색은 2개의 조회 조건을 결합시키는 조회를 이용하여 수행됩니다. 첫 번째 조회 조건은 특성 이름이 Department이고 값이 "General"이거나 특성 이름이 MissingProperty이고 값이 "somevalue"인 비즈니스 규칙 그룹을 검색하는 것입니다. 특성 이름이 RuleType이며 값이 "messages"인 조건에 AND를 이용하여 이 조회 조건을 조합합니다.

비즈니스 규칙 그룹 조회의 더 많은 예제가 부록에서 사용 가능합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example5
{
static Formatter out = new Formatter();
static public String executeExample5()
{
  try
  {
    out.clear();

    // Retrieve business rule groups based on three conditions where
    // two of the conditions are combined in an OR node
    // Create PropertyQueryNodes for each condition for the OR node
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"General");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("MissingProperty",
    QueryOperator.EQUAL, "SomeValue");
    // Combine the two PropertyQueryNodes with an OR node
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1, propertyNode2);
```

```
    // Create the third PropertyQueryNode
      PropertyQueryNode propertyNode3 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
    QueryOperator.EQUAL,"messages");
```

왼쪽 조건이 AND 노드를 이용하여 오른쪽과 결합됩니다. AndNode는 조회 트리의 루트입니다.

```
    // Combine OR node with third PropertyQueryNode with
    AndNode AndNode andNode =
    QueryNodeFactory.createAndNode(propertyNode3, orNode);

    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(andNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Iterate through the list of business rule groups
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Output attributes for each business rule group
      out.println1nBold("Business Rule Group");
      out.println("Name: " + brg.getName());
      out.println("Namespace: " +
        brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());
      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Output property names and values
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
        out.println("₩t Property Name: " +
        prop.getName());
        out.println("₩t Property Value: " +
        prop.getValue());
      }
    }
   }   catch (BusinessRuleManagementException e)
  {
e.printStackTrace();
  out.println(e.getMessage());
  }
 return out.toString();
 }
}
```

예제

예제 5의 웹 브라우저 출력.

example5 실행

```
비즈니스 규칙 그룹
이름: ConfigurationValues
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
표시 이름: ConfigurationValues
설명: null
프리젠테이션 시간대: LOCAL
저장 날짜: Sun Jan 06 17:56:51 CST 2008
특성 이름: IBMSystemVersion
특성 값: 6.2.0
특성 이름: Department
특성 값: General
특성 이름: RuleType
특성 값: messages
특성 이름: IBMSystemTargetNameSpace
특성 값: http://BRSamples/com/ibm/websphere/sample/brules
특성 이름: IBMSystemName
특성 값: ConfigurationValues
특성 이름: IBMSystemDisplayName
특성 값: ConfigurationValues
```

예제 6: 비즈니스 규칙 그룹 특성 갱신 및 공개

이 예제에서 비즈니스 규칙 그룹의 특성이 갱신된 후 비즈니스 규칙 그룹이 공개됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example6
{
static Formatter out = new Formatter();

static public String executeExample6()
{
 try
 {
   out.clear();
   out.println1nBold("Business Rule Group before publish:");
   // Retrieve business rule groups by a single property value
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsBySingleProperty("Department",
     QueryOperator.EQUAL,"General", 0, 0);

   if (brgList.size() > 0)
```

```
  {
    // Get the first business rule group from the list
    BusinessRuleGroup brg = brgList.get(0);
    // Retrieve the property from the business rule group
    UserDefinedProperty userDefinedProperty =
    (UserDefinedProperty) brg
      .getProperty("Department");

    out.println("Business Rule Group: " + brg.getName());
    out.println("Department Property value: "
    + brg.getProperty("Department").getValue());
```

getProperty 메소드는 참조에 의해 특성을 리턴하며 특성에 작성한 변경사항은 직접 비즈니스 규칙 그룹에 작성됩니다.

```
    // Modify the property value in the brg
    // This updates the property value directly in the
    brg object
    userDefinedProperty.setValue("GeneralConfig");
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();
    // Add the changed business rule group to the list
    publishList.add(brg);
```

BusinessRuleManager 클래스는 비즈니스 규칙 그룹에 작성한 변경사항을 공개하는 데 사용됩니다. 변경사항을 공개하기 위해, 오직 하나의 항목이 공개되는 경우에도 BusinessRuleManager 공개 메소드로 목록이 전달됩니다.

```
    // Publish the list with the updated business rule group
    BusinessRuleManager.publish(publishList, true);

    out.println("");

    // Retrieve the business rule group again to verify the
    // changes were published
    out.println1nBold("Business Rule Group after publish:");
    brgList = BusinessRuleManager
    .getBRGsBySingleProperty("Department",
    QueryOperator.EQUAL, "GeneralConfig", 0, 0);

    brg = brgList.get(0);

    out.println("Business Rule Group: " + brg.getName());
    // Display the property value to show the change
    out.println("Department Property value: "
    + brg.getProperty("Department").getValue());
  }
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
 out.println(e.getMessage());
 }
return out.toString();
}
}
```

예제

예제 6의 웹 브라우저 출력.

example6 실행

```
공개 이전 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: ConfigurationValues
Department 특성 값: General

공개 이후 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: ConfigurationValues
Department 특성 값: GeneralConfig
```

예제 7: 다중 비즈니스 규칙 그룹에서 특성 갱신 및 공개

이 예제에서, 다중 비즈니스 규칙 그룹의 특성은 공개 이전에 갱신됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example7
{
static Formatter out = new Formatter();

static public String executeExample7()
{
  try
  {
    out.clear();
    out.printlnBold("Business Rule Group before publish:");
    // Retrieve business rule groups by a single property value
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsBySingleProperty("Department",
      QueryOperator.EQUAL, "Accounting", 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;

    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Iterate through all of the business rule groups and
    // modify the property
    while (iterator.hasNext())
    {
      // Retrieve the property from the business rule group
```

```
      brg = iterator.next();

      out.println("Business Rule Group: " + brg.getName());

      // Retrieve the property from the business rule group
      UserDefinedProperty prop = (UserDefinedProperty) brg
        .getProperty("Department");
      out.println("Department Property value: "
      +
      brg.getProperty("Department").getValue())
      ;

      // Modify the property value in the brg
      // This updates the property value directly in the
      brg object
      prop.setValue("Finance");
```

변경된 각 비즈니스 규칙 그룹이 목록에 추가됩니다.

```
      // Add the changed business rule group to the list
      publishList.add(brg);
      }

      // Publish the list with the updated business rule
      group
      BusinessRuleManager.publish(publishList, true);

      out.println("");

      // Retrieve the business rule groups again to verify the
      // changes were published
      out.println1nBold("Business Rule Group after
      publish:");

      brgList = BusinessRuleManager
        .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL,
        "Finance", 0, 0);
      iterator = brgList.iterator();

      while (iterator.hasNext())
      {
        brg = iterator.next();
        out.println("Business Rule Group: " +
        brg.getName());
        out.println("Department Property value: "
         +
         brg.getProperty("Department").getVa
         lue());
      }
} catch (BusinessRuleManagementException e)
{
e.printStackTrace();
  out.println(e.getMessage());
}
return out.toString();
}
}
```

예제

예제 7의 웹 브라우저 출력.

example7 실행

```
공개 이전 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: ApprovalValues
Department 특성 값: Accounting
비즈니스 규칙 그룹: DiscountRules
Department 특성 값: Accounting

공개 이후 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: ApprovalValues
Department 특성 값: Finance
비즈니스 규칙 그룹: DiscountRules
Department 특성 값: Finance
```

예제 8: 비즈니스 규칙 그룹의 기본값 비즈니스 규칙 변경

이 예제에서, 기본값 비즈니스 규칙은 특정 조작의 사용 가능한 대상 목록의 일부인 다른 비즈니스 규칙을 이용하여 변경됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example8
{
static Formatter out = new Formatter();

static public String executeExample8()
{
  try
  {
    out.clear();

  // Retrieve a business rule group by target namespace and name
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

  if (brgList.size() > 0)
  {
    out.printlnBold("Business Rule Group before publish:");
```

```
// Get the first business rule group from the list
// This should be the only business rule group in the list as
// the combination of target namespace and name are unique
BusinessRuleGroup brg = brgList.get(0);

out.print("Business Rule Group: ");
out.println(brg.getName());

// Get the operation of the business rule group that
// will have its default business rule updated
Operation op =
brg.getOperation("calculateShippingDiscount");
```

조작의 사용 가능한 대상 목록의 일부인 다른 규칙을 이용하여 갱신하기 전에 기본값 비즈니스 규칙이 검색됩니다. 규칙 세트 및 의사결정 테이블은 조작에 특정하며 조작을 위한 비즈니스 규칙 아티팩트만이 기본값으로 설정되거나 조작 시 다른 시간으로 예정될 수 있습니다.

```
// Retrieve the default business rule for the operation
BusinessRule defaultRule =
op.getDefaultBusinessRule();
out.print("Default Rule: ");
out.println(defaultRule.getName());

// Get the list of available business rules for this
operation
List<BusinessRule> ruleList =
op.getAvailableTargets();

Iterator<BusinessRule> iterator =
ruleList.iterator();
BusinessRule rule = null;

// Find a business rule that is different from the
current
// default
// business rule
while (iterator.hasNext())
{
  rule = iterator.next();
  if
  (!defaultRule.getName().equals(rule.getName()))
  {
```

기본값 비즈니스 규칙은 조작 오브젝트에서 설정됩니다. 널로 기본값 비즈니스 규칙을 설정하면 조작에서 기본값 비즈니스 규칙을 제거하지만, 모든 조작이 지정된 기본값 비즈니스 규칙을 가지는 것이 좋습니다.

```
    // Set the default business rule to be a
    // different business rule
    // This change is to the operation object
    // directly
    op.setDefaultBusinessRule(rule);
    break;
  }
}
```

```
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();
    // Add the changed business rule group to the list
    publishList.add(brg);
    // Publish the list with the updated business rule
    group
    BusinessRuleManager.publish(publishList, true);

    out.println("");

    // Retrieve the business rule groups again to verify the
    // changes were published

    out.printlnBold("Business Rule Group after publish:");
    brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere/sample/brules",
      QueryOperator.EQUAL, "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    out.println("Business Rule Group: " + brg.getName());
    op = brg.getOperation("calculateShippingDiscount");

    // Retrieve the default business rule for the operation
    defaultRule = op.getDefaultBusinessRule();
    out.print("Default Rule: ");
    out.println(defaultRule.getName());
   }
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

예제

예제 8의 웹 브라우저 출력.

example8 실행

```
공개 이전 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: DiscountRules
기본값 규칙: calculateShippingDiscount

공개 이후 비즈니스 규칙 그룹:
비즈니스 규칙 그룹: DiscountRules
기본값 규칙: calculateShippingDiscountHoliday
```

예제 9: 비즈니스 규칙 그룹에서 조작의 다른 규칙 스케줄

이 예제에서, 비즈니스 규칙은 특정 조작의 공개 시간으로부터 1시간 활성화되도록 예정됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example9 {
static Formatter out = new Formatter();

static public String executeExample9()
{
 try
 {
   out.clear();

   // Retrieve a business rule group by target namespace and name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
     QueryOperator.EQUAL,
     "DiscountRules",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
   {
     out.println("");
     out.printlnBold("Business Rule Group before publish:");
     // Get the first business rule group from the list
     // This should be the only business rule group in the
     list as
     // the combination of target namespace and name are unique
     BusinessRuleGroup brg = brgList.get(0);

     // Get the operation of the business rule group that
     // will have a new business rule scheduled
     Operation op =
     brg.getOperation("calculateShippingDiscount");

     printOperationSelectionRecord(op);
     // Get the list of available business rules for this operation
     List<BusinessRule> ruleList =
```

```
op.getAvailableTargets();

// Get the first rule in the list as this will be scheduled
// for the operation
BusinessRule rule = ruleList.get(0);

// Get the list of scheduled business rules
OperationSelectionRecordList opList = op
  .getOperationSelectionRecordList();
// Create an end date in the future for the business rule
Date future = new Date();
long futureTime = future.getTime() + 3600000;
```

예정된 새 규칙의 경우, 시작 날짜 및 종료 날짜는 규칙과 함께 지정될 수 있습니다. 시작 날짜가 널로 설정되면, 이것은 공개 시 규칙이 활성화됨을 표시합니다. 종료 날짜가 널로 설정되면, 규칙은 종료 날짜를 갖지 않습니다. 스케줄 겹침은 허용되지 않으며 조작에 대한 유효성 검증 메소드를 호출하여 확인될 수 있습니다.

```
// Create the new scheduled business rule with the current
// date which means this rule will become active immediately
// upon
// publish and the future date.
newOperationSelectionRecord(new Date(),
  new Date(futureTime), rule);
// Add the new scheduled business rule to the list of
// scheduled rule
opList.addOperationSelectionRecord(newRecord);
```

겹침이 존재하지 않음을 확인하는 유효성 검증 조작입니다.

```
// Validate the list to insure there isn't an overlap
List<Problem> problems = op.validate();
if (problems.size() == 0)
{
  // Use the original list or create a new list
  // of business rule groups
  List<BusinessRuleGroup> publishList = new
  ArrayList<BusinessRuleGroup>();
  // Add the changed business rule group to the list
  publishList.add(brg);
  // Publish the list with the updated business
  rule group
  BusinessRuleManager.publish(publishList, true);
  out.println("");

  // Retrieve the business rule groups again to
  verify the
  // changes were published
  out.printlnBold("Business Rule Group after
  publish:");

  brgList =
  BusinessRuleManager.getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere
    /sample/brules",
    QueryOperator.EQUAL,
    "DiscountRules",
```

```
          QueryOperator.EQUAL, 0, 0);
          brg = brgList.get(0);

          op =
          brg.getOperation("calculateShippingDiscount");

          printOperationSelectionRecord(op);
        }
        // else handle the validation error
      }
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
}
/*
Method to print the operation selection record for an operation. The
start date and end date are printed as well as the name of the rule
artifact for the scheduled time.
*/
private static void printOperationSelectionRecord(Operation op)
{
 OperationSelectionRecordList opSelectionRecordList = op
   .getOperationSelectionRecordList();
 Iterator<OperationSelectionRecord> opSelRecordIterator =
 opSelectionRecordList
   .iterator();
 OperationSelectionRecord record = null;
 while (opSelRecordIterator.hasNext())
 {
   out.println1nBold("Scheduled Destination:");
   record = opSelRecordIterator.next();
   out.println("Start Date: " + record.getStartDate()
   + " - End Date: " + record.getEndDate());
   BusinessRule ruleArtifact = record.getBusinessRuleTarget();
   out.println("Rule: " + ruleArtifact.getName());
   }
 }
}
```

예제

예제 9의 웹 브라우저 출력.

example9 실행

```
공개 이전 비즈니스 규칙 그룹:
예정된 대상:
시작 날짜: Thu Dec 01 00:00:00 CST 2005 - End Date:
Sun Dec 25 00:00:00 CST 2005
규칙: calculateShippingDiscountHoliday

공개 이후 비즈니스 규칙 그룹:
예정된 대상:
시작 날짜: Thu Dec 01 00:00:00 CST 2005 - End Date:
Sun Dec 25 00:00:00 CST 2005
```

```
규칙: calculateShippingDiscountHoliday
예정된 대상:
시작 날짜: Mon Jan 07 21:08:31 CST 2008 - End Date:
Mon Jan 07 22:08:31 CST 2008
규칙: calculateShippingDiscount
```

예제 10: 규칙 세트에서 템플리트의 매개변수 값 수정

이 예제에서 템플리트로 정의된 인스턴스는 매개변수 값을 변경하여 수정된 후 공개됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;

public class Example10
{
static Formatter out = new Formatter();

static public String executeExample10()
{
  try
  {
    out.clear();

    // Retrieve a business rule group by target namespace and
    name
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "ApprovalValues",
        QueryOperator.EQUAL, 0, 0);
    if (brgList.size() > 0)
    {
      // Get the first business rule group from the list
      // This should be the only business rule group in the
      list as
      // the combination of target namespace and name are
      unique
      BusinessRuleGroup brg = brgList.get(0);
      // Get the operation of the business rule group that
      // has the business rule that will be modified as
      // the business rules are associated with a specific
```

```
// operation
Operation op = brg.getOperation("getApprover");

// Get the business rule on the operation that will
be modified
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
  "getApprover", QueryOperator.EQUAL, 0,
  0);

if (ruleList.size() > 0)
{
  out.println("");
  out.printlnBold("Rule set before publish:");
  // Get the rule to be modified. Rules are
  unique by
  // target namespace and name, but for this
  example
  // there is only one business rule named
  "getApprover"
  RuleSet ruleSet = (RuleSet) ruleList.get(0);
  out.print(RuleArtifactUtility.printRuleSet(rule
  Set));
```

규칙 세트의 모든 규칙은 규칙 블록에 있습니다. 하나의 규칙 블록만이 지원되며 규칙 블록을 검색하려면 getFirstRuleBlock 메소드를 사용해야 합니다.

```
// A rule set has all of the rules defined in a
rule block
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

// Iterate through the rules in the rule block
to find the
// rule instance called "LargeOrderApprover"
while (ruleIterator.hasNext())
{
  RuleSetRule rule = ruleIterator.next();
```

규칙이 규칙 템플리트를 이용하여 정의되지 않은 경우, 검색될 수 있는 웹 프리젠테이션만이 있습니다. 템플리트를 이용하여 정의되지 않은 규칙은 갱신될 수 없습니다. 규칙 이름이 알려진 경우 규칙이 템플리트를 이용하여 정의되었는지 여부를 검사하는 것이 가장 좋습니다.

```
// The rule must have been defined with a
template
// in order for it to be changed. Check
if the current
// rule is even based on a template.
if (rule instanceof
RuleSetTemplateInstanceRule)
{
```

규칙을 작성하려면 TemplateInstance 오브젝트를 사용하십시오.

```
// Get the rule template instance
RuleSetTemplateInstanceRule
templateInstance =
(RuleSetTemplateInstanceRule) rule;

// Check for the rule instance
which matches
// the rule to modify
if
(templateInstance.getName().equals(
  "LargeOrderApprover"))
{
```

템플리트 인스턴스의 경우, 매개변수 값만이 수정될 수 있습니다. ParameterValue를 검색하고 적절한 값으로 설정하여 매개변수를 수정합니다. ParameterValue는 참조에 의해 전달되므로, 규칙, 규칙 세트 및 비즈니스 규칙 그룹을 직접 갱신합니다.

```
    // Get the parameter from the
    rule instance
    ParameterValue parameter =
    templateInstance
      .getParameterValue("par
      am2");

    // Modify the value of the
    parameter
    parameter.setValue("superviso
    r");
    break;
  }
 }
}
// Use the original list or create a new list
// of business rule groups
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Add the changed business rule group to the list
publishList.add(brg);

// Publish the list with the updated business rule
 group
 BusinessRuleManager.publish(publishList, true);

out.println("");
// Retrieve the business rule groups again to verify
the
// changes were published
out.printlnBold("Rule set after publish:");

brgList = BusinessRuleManager
  .getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere/sample/brules",
    QueryOperator.EQUAL, "ApprovalValues",
    QueryOperator.EQUAL, 0, 0);
```

```
    brg = brgList.get(0);
    op = brg.getOperation("getApprover");
    ruleList = op.getBusinessRulesByName(
      "getApprover", QueryOperator.EQUAL, 0,0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(ruleSet));
    }
  }
  } catch (BusinessRuleManagementException e)
  {
e.printStackTrace();
    out.println(e.getMessage());
  }
  return out.toString();
  }
}
```

예제

예제 10의 웹 브라우저 출력.

예제 **10** 실행

```
공개 이전 규칙 세트:
규칙 세트
이름: getApprover
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
규칙: LargeOrderApprover
표시 이름: LargeOrderApprover
설명: null
펼친 사용자 프리젠테이션: 항목 주문이 10개를 초과하고 주문 금액이
$5000을 초과하면 관리자의 승인이 필요함
사용자 프리젠테이션: 항목 주문이 {0}개를 초과하고
주문이 ${1}를 초과하면 {2}의 승인이 필요함
매개변수 이름: param0
매개변수 값: 10
매개변수 이름: param1
매개변수 값: 5000
매개변수 이름: param2
매개변수 값: manager
규칙: DefaultApprover
표시 이름: DefaultApprover
설명: null
펼친 사용자 프리젠테이션: approver = peer
사용자 프리젠테이션: approver = {0}
매개변수 이름: param0
매개변수 값: peer

공개 이후 규칙 세트:
규칙 세트
이름: getApprover
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
규칙: LargeOrderApprover
표시 이름: LargeOrderApprover
설명: null
펼친 사용자 프리젠테이션: 항목 주문이 10개를 초과하고
주문이 $5000를 초과하면 감독자의 승인이 필요함
```

```
사용자 프리젠테이션: 항목 주문이 {0}개를 초과하고
주문이 ${1}를 초과하면 {2}의 승인이 필요함
매개변수 이름: param0
매개변수 값: 10
매개변수 이름: param1
매개변수 값: 5000
매개변수 이름: param2
매개변수 값: supervisor
규칙: DefaultApprover
표시 이름: DefaultApprover
설명: null
펼친 사용자 프리젠테이션: approver = peer
사용자 프리젠테이션: approver = {0}
매개변수 이름: param0
매개변수 값: peer
```

예제 11: 템플리트에서 규칙 세트로 새 규칙 추가

이 예제에서, 새 규칙이 템플리트에서 규칙 세트로 추가됩니다. 새 규칙 인스턴스가 작성되기 전에, 새 규칙 인스턴스에 대한 매개변수가 작성됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example11
{
static Formatter out = new Formatter();

static public String executeExample11()
{
 try
 {
  out.clear();
  // Retrieve a business rule group by target namespace and
  name
  List<BusinessRuleGroup> brgList = BusinessRuleManager
   .getBRGsByTNSAndName(
   "http://BRSamples/com/ibm/websphere
   /sample/brules",
   QueryOperator.EQUAL,
   "ApprovalValues",
```

```
 QueryOperator.EQUAL, 0, 0);

if (brgList.size() > 0)
{
 // Get the first business rule group from the list
 // This should be the only business rule group in the
 list as
 // the combination of target namespace and name are
 unique
 BusinessRuleGroup brg = brgList.get(0);
 // Get the operation of the business rule group that
 // has the business rule that will be modified as
 // the business rules are associated with a specific
 // operation
 Operation op = brg.getOperation("getApprover");

 // Get the business rule on the operation that will
 be modified
 List<BusinessRule> ruleList =
 op.getBusinessRulesByName(
  "getApprover", QueryOperator.EQUAL, 0,0);

 if (ruleList.size() > 0)
 {
  out.println("");
  out.printlnBold("Rule set before publish:");
  // Get the rule to be modified. Rules are unique by
  // target namespace and name, but for this example
  // there is only one business rule named
  "getApprover"
  RuleSet ruleSet = (RuleSet) ruleList.get(0);
  out.print(RuleArtifactUtility.printRuleSet(rule
  Set));
```

새 규칙을 규칙 세트에 추가하려면, 적절한 템플리트가 규칙 세트에 있어야 하며 인스턴스가 템플리트에서 작성되어야 합니다. 이름별로 템플리트의 위치를 정할 수 있습니다.

```
  // Get the list of rule templates
  ListRuleSetRuleTemplate> ruleTemplates =
  ruleSet
   .getRuleTemplates();

  Iterator<RuleSetRuleTemplate> templateIterator
  = ruleTemplates
   .iterator();

  while (templateIterator.hasNext())
  {
   RuleSetRuleTemplate template =
   templateIterator.next();

   // Locate the template to use to create a
   new rule
```

```
if
(template.getName().equals("Template_Larg
eOrder"))
{
```

템플리트 인스턴스의 경우, 매개변수 목록이 작성되어야 합니다.

```
// Create a list for the parameters
for this template
// rule instance
List<ParameterValue> paramList =
new ArrayList<ParameterValue>();

// From the template definition,
get a specific parameter
// and set the value
Parameter param =
template.getParameter("param0");
ParameterValue paramValue = param
 .createParameterValue("
 20");

// Add parameter to the list
paramList.add(paramValue);

// Get the next parameter and set
the value
param = template.getParameter("param1");
paramValue =
param.createParameterValue("7500");

// Add parameter to the list
paramList.add(paramValue);

// Get the next parameter and set
the value
param =
template.getParameter("param2");
 paramValue = param
 .createParameterValue("
 2nd-line manager");

// Add parameter to the list
paramList.add(paramValue);
```

작성된 매개변수를 사용하여, 템플리트 인스턴스를 작성할 수 있습니다.

```
// Create the template rule
instance with the parameter
// list
RuleSetTemplateInstanceRule
 templateInstance = template
 .createRuleFromTemplate
 ("ExtraLargeOrder",
 paramList);
```

```
     // Get the ruleblock for the rule
      set
     RuleBlock ruleBlock =
     ruleSet.getFirstRuleBlock();
```

템플리트 인스턴스가 작성되면, 규칙 블록에 추가할 수 있습니다. 규칙 블록에 템플리트 인스턴스를 추가하면 다른 템플리트 규칙 인스턴스 중간에 정렬될 수 있습니다.

```
     // Add the template rule to the
     ruleblock
     ruleBlock.addRule(templateInstance)
     ;

     break;
     }
    }

    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the
    list
    publishList.add(brg);

    // Publish the list with the updated business
    rule group
    BusinessRuleManager.publish(publishList, true);

    out.println("");

    // Retrieve the business rule groups again to
    verify the
    // changes were published
    out.printlnBold("Rule set after publish:");

    brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "ApprovalValues",
      QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    op = brg.getOperation("getApprover");
    ruleList = op.getBusinessRulesByName(
         "getApprover", QueryOperator.EQUAL,
         0, 0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(rule
    Set));
    }
  }
 } catch (BusinessRuleManagementException e)
 {
```

```
e.printStackTrace();
  out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

예제

예제 11의 웹 브라우저 출력.

example11 실행

```
공개 이전 규칙 세트:
규칙 세트
이름: getApprover
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
규칙: LargeOrderApprover
표시 이름: LargeOrderApprover
설명: null
펼친 사용자 프리젠테이션: 항목 주문이 10개를 초과하고 주문이
$5000를 초과하면 감독자의 승인이 필요함
사용자 프리젠테이션: 항목 주문이 {0}개를 초과하고 주문이
${1}를 초과하면 {2}의 승인이 필요함
매개변수 이름: param0
매개변수 값: 10
매개변수 이름: param1
매개변수 값: 5000
매개변수 이름: param2
매개변수 값: supervisor
규칙: DefaultApprover
표시 이름: DefaultApprover
설명: null
펼친 사용자 프리젠테이션: approver = peer
사용자 프리젠테이션: approver = {0}
매개변수 이름: param0
매개변수 값: peer

공개 이후 규칙 세트:
규칙 세트
이름: getApprover
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
규칙: LargeOrderApprover
표시 이름: LargeOrderApprover
설명: null
펼친 사용자 프리젠테이션: 항목 주문이 10개를 초과하고 주문이
$5000를 초과하면 감독자의 승인이 필요함
사용자 프리젠테이션: 항목 주문이 {0}개를 초과하고 주문이
${1}를 초과하면 {2}의 승인이 필요함
매개변수 이름: param0
매개변수 값: 10
매개변수 이름: param1
매개변수 값: 5000
매개변수 이름: param2
매개변수 값: supervisor
규칙: DefaultApprover
표시 이름: DefaultApprover
설명: null
```

```
펼친 사용자 프리젠테이션: approver = peer
사용자 프리젠테이션: approver = {0}
매개변수 이름: param0
매개변수 값: peer
규칙: ExtraLargeOrder
표시 이름:
설명: null
펼친 사용자 프리젠테이션: 항목 주문이 20개를 초과하고 주문이
$7500를 초과하면 관리자의 승인이 필요함
사용자 프리젠테이션: 항목 주문이 {0}개를 초과하고 주문이
${1}를 초과하면 {2}의 승인이 필요함
매개변수 이름: param0
매개변수 값: 20
매개변수 이름: param1
매개변수 값: 7500
매개변수 이름: param2
매개변수 값: 2nd-line manager
```

예제 12: 매개변수 값을 변경하여 의사결정 테이블에서 템플리트 수정 후 공개

이 예제에서, 조건 및 조치(둘 다 템플리트를 이용하여 정의)는 공개되기 전에 매개변수 값을 변경하여 의사결정 테이블에서 수정됩니다.

의사결정 테이블에서 조건 및 조치를 수정하는 가장 손쉬운 방법은 각 조건 레벨의 템플리트와 각 조치에 대해 고유한 이름을 사용하는 것입니다. 고유한 이름이 검색될 수 있으며 그런 다음, 템플리트를 이용하여 정의된 템플리트 인스턴스를 변경할 수 있습니다. 특정 템플리트의 템플리트 인스턴스를 변경하면, 해당 레벨에서 해당 템플리트를 이용하여 정의된 모든 조건 값이 갱신됩니다. 조치 표현식의 경우, 각 인스턴스는 고유하며 한 인스턴스에 대한 변경이 다른 인스턴스를 변경하지 않습니다.

이 예제에서, 갱신을 위해 특정 Case Edge 찾기, 특정 매개변수 값 찾기 그리고 특정 템플리트를 이용하여 정의된 조치 표현식 찾기를 단순화시키기 위해 작성된 몇 개의 추가 메소드가 있습니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
```

```
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example12 {
static Formatter out = new Formatter();

static public String executeExample12()
{
 try
 {
   out.clear();
   // Retrieve a business rule group by target namespace and
   name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere
    /sample/brules",
    QueryOperator.EQUAL,
    "ConfigurationValues",
    QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
   {
    // Get the first business rule group from the list
    // This should be the only business rule group in the
    list as
    // the combination of target namespace and name are
    unique
    BusinessRuleGroup brg = brgList.get(0);

    // Get the operation of the business rule group that
    // has the business rule that will be modified as
    // the business rules are associated with a specific
    // operation
    Operation op = brg.getOperation("getMessages");

    // Get all business rules available for this
    operation
    List<BusinessRule> ruleList =
    op.getAvailableTargets();

    // For this operation there is only 1 business rule
    and
    // it is the business that we want to update
    DecisionTable decisionTable = (DecisionTable)
    ruleList.get(0);
    out.println("");
    out.printlnBold("Decision table before publish:");
    out
     .print(RuleArtifactUtility
     .printDecisionTable(decisionT
      able));
```

init 규칙과 조건 및 조치가 트리 블록에 포함됩니다. 트리 블록을 이용하여, 루트 노드를 검색할 수 있습니다.

```
// Get the tree block that contains all of the
conditions
// and actions for the decision table
TreeBlock treeBlock = decisionTable.getTreeBlock();
// From the tree block, get the tree node which is
the
// starting point for navigating through the decision
table
TreeNode treeNode = treeBlock.getRootNode();
```

갱신되는 조건이 "Condition Value Template 2.1" 이름으로 템플리트를 이용하여 정의되었습니다. `getCaseEdge` 메소드는 템플리트가 정의된 Case Edge를 찾기 위해 TreeNode에서 적절한 Case Edge로 반복 검색합니다. 메소드는 템플리트가 정의된 레벨이 현재 레벨과 마찬가지로 잘 알려진 상태라고 예상합니다. 동일한 이름이 다중 Case Edge에서 사용되는 경우 특정 이름으로 템플리트를 사용해 Case Edge를 찾기 위해 이 메소드를 사용할 수 있습니다.

```
// Find the case edge at level 1 below the root with
// specific template with a parameter value that has
// a specific name. Since we are starting at the top,
// the current depth is 0
CaseEdge caseEdge = getCaseEdge(treeNode, "param0",
"Condition Value Template 2.1", 1, 0);
```

Case Edge를 찾으면, 조건의 `ConditionValueTemplateInstance`를 검색할 수 있습니다.

```
if (caseEdge != null)
{
 // Case edge was found. Get the value
 definition of the
 // case edge
 TreeConditionValueDefinition condition =
  caseEdge
  .getValueDefinition();
 // Get the condition expression defined with a
  template
  TemplateInstanceExpression conditionExpression
  = condition
   .getConditionValueTemplateInstance(
 );
```

`ConditionValueTemplateInstance`를 이용하여, 적절한 매개변수 값을 검색한 후 `getParameterValue` 메소드를 이용하여 갱신할 수 있습니다.

```
 // Get the template for the expression
 Template conditionTemplate =
 conditionExpression
  .getTemplate();

  // Check that template is correct as it is
  possible to have
```

```
// multiple templates for a condition value,
but only one
// applied
if (conditionTemplate.getName().equals(
 "Condition Value Template 2.1"))
{
 // Get the parameter value
 ParameterValue parameterValue =
 getParameterValue("param0",
  conditionExpression);

 // Set the new parameter value
 parameterValue.setValue("info");
}
```

그런 다음, 갱신될 필요가 있는 템플리트를 이용하여 정의된 여러 조치 표현식을 검색할 수 있습니다. getActionExpressions 메소드는 이름 조치 값 Template 1로 템플리트를 이용하여 정의된 모든 조치를 리턴합니다.

```
ConditionNode conditionNode = (ConditionNode)
treeNode;

// Get the case edges tree node
ListCaseEdge> caseEdges =
conditionNode.getCaseEdges();

// Create a list to hold all of the action
expressions that
// also need to be updated. Because every
action is
// independent of other action even though the
template is
// shared, all must be updated.
List<TemplateInstanceExpression> expressions =
new Vector<TemplateInstanceExpression>();

// Retrieve all of the expressions
for (CaseEdge edge : caseEdges)
{
  getActionExpressions("Action Value
  Template 1", edge,
  expressions);
}
```

조치 표현식 목록을 사용하여, 각 항목을 갱신할 수 있습니다. 템플리트를 이용하여 정의된 조치 표현식의 경우 올바른 매개변수 값을 갱신할 수 있습니다.

```
// Update the correct parameter in each
expression
for (TemplateInstanceExpression expression
expressions)
{
 for (ParameterValue parameterValue :
 expression
  .getParameterValues())
 {
  // Check for correct parameter
```

```
     although there is
     // only one paramater in our
     template
     if
      (parameterValue.getParameter().getN
      ame().equals("param0")) {
       String value =
       parameterValue.getValue();
       parameterValue.setValue("Info
       "
       +
       value.substring(value.
       indexOf(":"),
       value.length()));
    }
  }
}
// With the condition value and actions
updated, the
// business rule group can be published.
// Use the original list or create a new list
// of business rule groups
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Add the changed business rule group to the
list
publishList.add(brg);

// Publish the list with the updated business
rule group
BusinessRuleManager.publish(publishList, true);

out.println("");

// Retrieve the business rule groups again to
verify the
// changes were published
out.printlnBold("Decision table after
publish:");

brgList =
BusinessRuleManager.getBRGsByTNSAndName(
 "http://BRSamples/com/ibm/websphere
 /sample/brules",
 QueryOperator.EQUAL,
 "ConfigurationValues",
 QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
op = brg.getOperation("getMessages");
ruleList = op.getAvailableTargets();

decisionTable = (DecisionTable)
ruleList.get(0);
out.print(RuleArtifactUtility
 .printDecisionTable(decisionTable))
 ;
```

```
    }
   }
 } catch (BusinessRuleManagementException e)
 {
e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
}

/*
Method to recursively navigate through a decision table and locate a
case
edge that has a template with a specific name and contains a specific
parameter to change. This method assumes that the level(depth) in the
decesion table of the value that is to be changed is known and the
current level(currentDepth) is tracked *
*/
static private CaseEdge getCaseEdge(TreeNode node, String pName,
  String templateName, int depth, int currentDepth)
{
 // Check if the current node is an action. This is an indication
 // that this branch of the decision table has been exhausted
 // looking for the case edge
 if (node instanceof ActionNode)
 {
return null;
 }

 // Get the case edges for this node
 List<CaseEdge> caseEdges = ((ConditionNode) node).getCaseEdges();
 for (CaseEdge caseEdge : caseEdges)
 {

 // Check if the correct level has been reached
 if (currentDepth < depth)
  {
   // Move down one level and then call getCaseEdge
   again
   // to process that level
   currentDepth++;
   return getCaseEdge(caseEdge.getChildNode(), pName,
    templateName, depth, currentDepth);
  } else
  {
   // The correct level has been reached. Get the
   condition in
   // order to check the templates on that condition on
   whether
   // they match the template sought
   TreeConditionValueDefinition condition = caseEdge
    .getValueDefinition();

   // Get the expression for the condition which has
   been defined
   // with a template
   TemplateInstanceExpression expression = condition
    .getConditionValueTemplateInstance();
```

```
   // Get the template from the expression
      Template template = expression.getTemplate();

   // Check if this is the template sought
    if (template.getName().equals(templateName))
    {
     // The template is found to match
      return caseEdge;
    } else
     caseEdge = null;
   }
 }
return null;
}

/*
This method will check the different parameter values for an expression
and if the correct one is found, return that parameter value.
*/
private static ParameterValue getParameterValue(String pName,
  TemplateInstanceExpression expression)
{
 // Check that the expression is not null as null would indicate
 // that the expression that was passed in was probably not
 defined
 // with a template and does not have any parameters to check.
 if (expression != null) {
  // Get the parameter values for the expression
  List<ParameterValue> parameterValues = expression
    .getParameterValues();

  for (ParameterValue parameterValue : parameterValues)
  {
   // For the different parameters, check that it
   matches the
   // parameter value sought

   if
    (parameterValue.getParameter().getName().equals(pName
   ))
   {
    // Return the parameter value that matched
      return parameterValue;
   }
  }
 }
return null;
}
/*
This method finds all of the action expressions that are
defined with a specific template. It recursively works through
a case edge and adds action expressions that match to the
expressions parameter.
*/

private static void getActionExpressions(String templateName,
CaseEdge next, List<TemplateInstanceExpression>
expressions)
```

```
{
 ActionNode actionNode = null;
 TreeNode treeNode = next.getChildNode();

 // Check if the current node is at the action node level
 if (treeNode instanceof ConditionNode)
 {
  List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
   .getCaseEdges();

  Iterator<CaseEdge> caseEdgesIterator =
  caseEdges.iterator();

  // Work through all case edges to find the action
  // expressions
  while (caseEdgesIterator.hasNext())
   {
    getActionExpressions(templateName,
    caseEdgesIterator.next(),
     expressions);
   }
        } else {
   // ActionNode found
   actionNode = (ActionNode) treeNode;

   List<TreeAction> treeActions = actionNode.getTreeActions();
   // Check that there is at least one treeAction specified
   for
   // the expression and work through the expressions checking
   // if the expressions have been created with the specific
   // template.
   if (!treeActions.isEmpty())
   {

    Iterator<TreeAction> iterator =
    treeActions.iterator();

   while (iterator.hasNext())
    {
     TreeAction treeAction = iterator.next();
     TemplateInstanceExpression expression =
     treeAction
      .getValueTemplateInstance();

     Template template = expression.getTemplate();

     if (template.getName().equals(templateName))
     {
      // Expression found with matching
      template
      expressions.add(expression);
     }
    }
   }
  }
 }
}
```

예제

예제 12의 웹 브라우저 출력.

```
예제 12 실행

공개 이전 규칙 세트:
의사결정 테이블
이름: getMessages
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules

공개 이후 의사결정 테이블:
의사결정 테이블
이름: getMessages
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
```

예제 13: 조건 값 및 조치를 의사결정 테이블에 추가

이 예제에서, 조건 값 및 조치가 의사결정 테이블에 추가됩니다. 템플리트를 사용하여 의사결정 테이블에 조건 값을 추가할 수 있습니다.

조건 값을 조건 노드에 추가하면 Case Edge를 추가합니다. 새 Case Edge가 Case Edge 목록 끝에 추가됩니다. 조건 값의 경우, 적절한 매개변수 값 세트가 있는 템플리트 인스턴스 표현식을 지정해야 합니다. 템플리트 인스턴스 표현식을 지정하려면 특정 템플리트를 사용해야 합니다. 해당 유형의 조건에 대해 올바른 템플리트를 검색하기 위해 각 조건 노드 레벨에서 템플리트 이름에 고유한 이름을 부여하는 것이 좋습니다. 단일 템플리트 정의를 사용하면, 어떤 레벨에서 조건이 추가되는지 판별하기 어렵습니다.

조건 노드에 조건 값을 설정하면 동일한 템플리트 인스턴스를 지닌 조건 값을 모든 조건 노드에 동일한 레벨로 추가합니다. 이 작업은 의사결정 테이블이 균형된 상태에서 수행됩니다. 또한 새 조건 값을 추가하는 일부로서, 새 조치 노드가 추가됩니다. 이 조치 노드는 사용자 프리젠테이션 및 템플리트 인스턴스 표현식에 지정되는 널값을 갖는 트리 조치를 취합니다. 조건 값은 조치 노드를 하위 노드로 갖지 않는 조건 노드에 추가될 수 있으므로, 조건 노드를 추가하면 조치 노드의 수가 늘어날 수 있습니다. 조치 노드의 수는 조건 노드가 추가된 레벨, 해당 레벨에서의 조건 노드 수 그리고 각 하위 레벨에서 조건 노드의 수에 근거합니다.

작성된 조치 노드를 찾기 위해, 널 사용자 프리젠테이션 및 템플리트 인스턴스 표현식을 갖는 트리 조치가 있는 조치 노드의 검색을 수행할 수 있습니다. TreeActionValueTemplate는 TreeAction으로 설정될 수 있는 표현식을 작성하기 위해 사용될 수 있습니다. 이 패턴은 모든 새 조치 노드에 반복될 필요가 있습니다.

이 예제에서는 새 트리 조치 설정을 돕기 위해 두 가지 메소드가 제공되었습니다. getEmptyActionNode는 현재 조건 노드에서 Empty 조치 노드를 반복적으로 찾으며 getParameterValue는 이름에 의해 지정된 매개변수의 값을 리턴합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
```

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example13
{
 static Formatter out = new Formatter();

 static public String executeExample13()
 {
  try
  {
   out.clear();

   // Retrieve a business rule group by target namespace
   // and name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
   "http://BRSamples/com/ibm/websphere/sample/brules",
     QueryOperator.EQUAL,"ConfigurationValues",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
    {
     // Get the first business rule group from the
     // list. This should be the only business
     // rule group in the list as the combination
     // of target namespace and name are unique
     BusinessRuleGroup brg = brgList.get(0);

     // Get the operation of the business rule
     // group that has the business rule that will
     // be modified as the business rules are
     // associated with a specific operation
     Operation op = brg.getOperation("getMessages");
```

```
// Get all business rules available for
// this operation
List<BusinessRule> ruleList =
 op.getAvailableTargets();

// For this operation there is only 1 business
// rule and it is the business that we want
// to update

DecisionTable decisionTable = (DecisionTable)
 ruleList.get(0);
 out.printlnBold("Decision table before
 publish:");
 out.print(RuleArtifactUtility
 .printDecisionTable(decisionTable));
```

조건 값을 추가할 레벨의 위치를 지정해야 합니다. 이 레벨은 일반적으로 클래스를 사용하는 사용자 인터페이스나 응용프로그램이 조건을 추가할 위치를 파악하고 있으므로 매개변수로 전달됩니다.

```
// Get the tree block that contains all of
// the conditions and actions for the decision
// table
TreeBlock treeBlock =
 decisionTable.getTreeBlock();

// From the tree block, get the tree node which
// is the starting point for navigating through
// the decision table
ConditionNode conditionNode = (ConditionNode)
 treeBlock.getRootNode();

// Get the case edges for this node which is
// the first level of conditions
List<CaseEdge> caseEdges =
 conditionNode.getCaseEdges();

// Get the case edge which will have the new
// condition added
CaseEdge caseEdge = caseEdges.get(0);

// For the case edge get the condition node in
// order to retrieve the templates for the
// condition
conditionNode = (ConditionNode)
    caseEdge.getChildNode();

// Get the templates for the condition
List<TreeConditionValueTemplate>
 treeValueConditionTemplates = conditionNode
 .getAvailableValueTemplates();

Iterator<TreeConditionValueTemplate>
 treeValueConditionTemplateIterator =
```

```
 treeValueConditionTemplates.iterator();

TreeConditionValueTemplate conditionTemplate =
 null;
```

의사결정 테이블에서 각 조건 노드 레벨의 고유한 템플리트 이름을 사용해서 올바른 조건 노드 값에서 조건 값이 추가되는지 좀 더 쉽게 확인할 수 있습니다.

```
// Find the template that should be used
 while
(treeValueConditionTemplateIterator.hasNext())
{
 conditionTemplate =
  treeValueConditionTemplateIterator
   .next();
 if (conditionTemplate.getName().equals(
  "Condition Value Template
  2.1"))
  {
   // Template found
   break;
  }
  conditionTemplate = null;
}
if (conditionTemplate != null)
{
```

올바른 템플리트를 찾으면, 인스턴스를 작성하여 이를 조건 노드에 추가하기 전에 적절한 매개변수 값이 설정되게 할 수 있습니다.

```
// Get the parameter definition from the
// template
Parameter conditionParameter =
conditionTemplate.getParameter("param0");

// Create a parameter value instance to
// be used in a new condition template
// instance
ParameterValue conditionParameterValue =
  conditionParameter
  .createParameterValue("fatal");

List<ParameterValue>
  conditionParameterValues = new
  ArrayList<ParameterValue>();

// Add the parameter value to a list

conditionParameterValues
 .add(conditionParameterValue);

// Create a new condition template
// instance with the parameter value
TemplateInstanceExpression
 newConditionValue =
 conditionTemplate
  .createTemplateInstanceExpression(c
```

```
 onditionParameterValues);
// Add the condition template instance to
// this condition node
conditionNode

.addConditionValueToThisLevel(newConditionValue);
// When a condition node is added there
// are new action nodes that are created
// and empty. These must be filled with
// action template instances. By
// searching for each empty action
// node from the parent level, all of the
// new empty action nodes can be found.
conditionNode = (ConditionNode)
conditionNode.getParentNode();
```

조건 값이 조건 노드에 추가되면, 새 조치 노드의 트리 조치는 TreeActionValueTemplate를 이용하여 설정되어야 합니다. 우선 Case Edge의 Empty 조치 노드를 찾으십시오. 상위 조건 노드를 사용하여 조건 노드를 통해 반복함에 따라 모든 조치 노드를 선택하게 됨을 확인하십시오.

```
// Get the case edges for the parent node
caseEdges = conditionNode.getCaseEdges();

Iterator<CaseEdge> caseEdgesIterator =
caseEdges.iterator();

while (caseEdgesIterator.hasNext())
{
 // For each case edge, retrieve an
 // empty action node if it exists
 ActionNode actionNode =
 getEmptyActionNode(caseEdgesIterator
 .next());

 // Check if all actions are filled
 if (actionNode != null)
 {
```

트리 조치가 빈 상태인 조치 노드를 발견한 경우 TreeActionValueTemplate를 사용하여 트리 조치를 설정해야 합니다. 우선 템플리트를 찾은 다음 템플리트 인스턴스를 작성하기 전에 매개변수를 지정하십시오. 템플리트 인스턴스를 작성하면, 트리 조치를 갱신할 수 있습니다. 이 예제에서는 동일한 조건 노드 아래의 다른 조치 노드에서 다른 트리 조치의 값으로 매개변수가 설정되었습니다. 새 매개변수 값을 작성하는 데 사용되는 값이 없는 다른 트리 조치가 포함된 의사결정 테이블의 경우, 응용프로그램의 매개변수로 값을 전달해야 합니다.

```
  // Get the list of tree
  // actions. These
  // are not the actual
  // actions, but the
  // placeholders for the
  // actions
  List<TreeAction>
```

```
treeActionList = actionNode
 .getTreeActions();

List<TreeActionTermDefinition>
treeActionTermDefinitions =
 treeBlock
.getTreeActionTermDefinitions();

List<TreeActionValueTemplate>
 treeActionValueTemplates =
 treeActionTermDefinitions
 .get(0).getValueTemplates();

TreeActionValueTemplate
 actionTemplate = null;

for (TreeActionValueTemplate
  tempActionTemplate :
  treeActionValueTemplates)
  {

  if
  (tempActionTemplate.get
   Name().equals(
   "Action Value
   Template 1"))
   {
    actionTemplate =
    tempActionTemplate;
    break;
   }
  }

  if (actionTemplate != null)
  {
   // Get another action
   // that is under
   // the parent condition
   // node in order
   // to use the value as
   // the basis for
   // the error message in
   // the new
   // action node. Move up
   // to the
   // parent condition
   // node first
   ConditionNode
   parentNode =
   (ConditionNode)
   actionNode
    .getParentNode();

   // Get the first case
   // edge of the
   // parent node as this
   // action will
   // always be filled in
```

```
// as new actions
// are added to the end
// of the case
// edge list.
CaseEdge caseE =
 parentNode.getCas
 eEdges().get(
 0);

// The child node is an
// action node
// and at the same
// level as the new
// action node.
ActionNode aNode =
(ActionNode) caseE
 .getChildNode();

// Get the list of tree
// actions
TreeAction
 existingTreeAction =
 aNode
 .getTreeActions()
 .get(0);

// Get the template
// instance
// expression for the
// tree action
// from which you can
// retrieve the
// parameter

TemplateInstanceExpression
existingExpression =
 existingTreeAction
 .getValueTemplateInstance();

ParameterValue
existingParameterValue =
getParameterValue(
  "param0",
existingExpression);

String actionValue =
existingParameterValue
 .getValue();

// Create the new
// message from the
// message of the
// existing
// tree action
actionValue = "Fatal"
  +
actionValue.substring(actionValue
 .indexOf(":"), actionValue
```

```
        .length());
        Parameter
        actionParameter =
        actionTemplate
        .getParameter("param0");

        // Get the parameter
        // from the template
        ParameterValue
        actionParameterValue =
         actionParameter
        .createParameterValue(actionValue);

        // Add the parameter to
        // a list of templates
        List<ParameterValue>
        actionParameterValues = new
        ArrayList<ParameterValue>();

      actionParameterValues.add(actionParameterValue);

        // Create a new tree
        // action instance

       TemplateInstanceExpression
        treeAction = actionTemplate
  .createTemplateInstanceExpression(actionParameterValues);

       // Set the tree action
       // in the action node
       // by setting it in the
       // tree action list
```

이때 조치 노드의 트리 조치가 갱신됩니다.

```
       treeActionList.get(0)
       .setValueTemplateInstance(
       treeAction);
       }
      }
    }
  }
  // With the condition value and actions
  // updated, the business rule group can be
  // published.
  // Use the original list or create a new list
  // of business rule groups
   List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

  // Add the changed business rule group to the
  // list
   publishList.add(brg);

  // Publish the list with the updated business
  // rule group

  BusinessRuleManager.publish(publishList, true);
```

```
     brgList =
      BusinessRuleManager.getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere/sample/brules",
      QueryOperator.EQUAL, ConfigurationValues",
      QueryOperator.EQUAL, 0, 0);
      brg = brgList.get(0);
      op = brg.getOperation("getMessages");
      ruleList = op.getAvailableTargets();
      decisionTable = (DecisionTable)
       ruleList.get(0);
      out.printlnBold("Decision table after
       publish:");
      out
       .print(RuleArtifactUtility
        .printDecisionTable(decisionTable));
     }
    } catch (ValidationException e)
    {
     List<Problem> problems = e.getProblems();

     out.println("Problem = " +
     problems.get(0).getErrorType().name());

e.printStackTrace();
     out.println(e.getMessage());
    } catch (BusinessRuleManagementException e)
    {
e.printStackTrace();
     out.println(e.getMessage());
    }
    return out.toString();
   }

   /*
   * This method searches from the current case edge for any
   * action nodes that have empty tree actions. An empty
   * action node is found by looking at the end of the list
   * of case edges and checking if the action node has tree
   * actions that have both a null user presentation and
   * TemplateInstanceExpression.
   */
   private static ActionNode getEmptyActionNode(CaseEdge next)
   {
    ActionNode actionNode = null;
    TreeNode treeNode = next.getChildNode();

   if (treeNode instanceof ConditionNode)
   {
    List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
     .getCaseEdges();

    if (caseEdges.size() > 1)
    {
     // Get right-most case-edge as the new
     // condition and thus empty actions are at the
     // right-end of the case edges
     actionNode = getEmptyActionNode(caseEdges
```

```
   .get(caseEdges.size() - 1));

  if (actionNode != null)
  {
   return actionNode;
  }
}
} else
{
 actionNode = (ActionNode) treeNode;

 List<TreeAction> treeActions =
 actionNode.getTreeActions();

 if (!treeActions.isEmpty())
 {
 if
 ((treeActions.get(0).getValueUserPresentation() == null)
  &&
 (treeActions.get(0).getValueTemplateInstance() == null))
  {
   return actionNode;
  }
 }
  `actionNode = null;
 }
 return actionNode;
}
/*
* This method will check the different parameter values for an
* expression and if the correct one is found, return that
* parameter value.
*/
private static ParameterValue getParameterValue(String pName,
  TemplateInstanceExpression expression)
{
 ParameterValue parameterValue = null;

 // Check that the expression is not null as null would
 // indicate that the expression that was passed in was
 // probably not defined with a template and does not have
 // any parameters to check.
 if (expression != null)
 {
  // Get the parameter vlues for the expression
  List<ParameterValue> parameterValues = expression
   .getParameterValues();
  Iterator<ParameterValue> parameterIterator =
   parameterValues
   .iterator();

  // For the different parameters, check that it
  // matches the parameter value sought
  while (parameterIterator.hasNext())
  {
   parameterValue = parameterIterator.next();

   if
```

```
    (parameterValue.getParameter().getName().equals(pName))
    {
    // Return the parameter value that
    // matched
    return parameterValue;
    }
  }
 }
 return parameterValue;
}
}
```

예제

예제 13의 웹 브라우저 출력.

예제 **13** 실행

```
공개 이전 의사결정 테이블:
의사결정 테이블
이름: getMessages
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules

공개 이후 의사결정 테이블:
의사결정 테이블
이름: getMessages
네임스페이스: http://BRSamples/com/ibm/websphere/sample/brules
```

예제 14: 규칙 세트에서 오류 핸들

이 예제는 규칙 세트에서 문제점을 찾아내는 데 초점을 두고 적절한 메시지가 표시될 수 있는지 또는 상황을 정정하기 위해 취할 수 있는 조치 등과 같이 어떤 문제점이 발생했는지를 찾아냅니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.problem.ValidationError;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;
```

```
public class Example14 {
static Formatter out = new Formatter();

static public String executeExample14() {
 try {
   out.clear();

   // Retrieve a business rule group by target namespace and
   name
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
     QueryOperator.EQUAL,
     "ApprovalValues",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0) {
     // Get the first business rule group from the list
     // This should be the only business rule group in the
     list as
     // the combination of target namespace and name are
     unique
     BusinessRuleGroup brg = brgList.get(0);
     out.println("Business Rule Group retrieved");

     // Get the operation of the business rule group that
     // has the business rule that will be modified as
     // the business rules are associated with a specific
     // operation
     Operation op = brg.getOperation("getApprover");

     // Retrieve specific rule by name
     List<BusinessRule> ruleList =
     op.getBusinessRulesByName(
       "getApprover", QueryOperator.EQUAL, 0,
       0);

     // Get the specific rule
     RuleSet ruleSet = (RuleSet) ruleList.get(0);
     out.println("Rule Set retrieved");

     RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

     Iterator<RuleSetRule> ruleIterator =
     ruleBlock.iterator();

     // Search through the rules to find the rule to
     change
     while (ruleIterator.hasNext()) {
       RuleSetRule rule = ruleIterator.next();

       // Check that the rule was defined with a
       template
       // as it can be changed.
       if (rule instanceof
       RuleSetTemplateInstanceRule) {
```

```
        // Get the template rule instance
        RuleSetTemplateInstanceRule
        templateInstance =
        (RuleSetTemplateInstanceRule) rule;
        // Check for the correct template rule
        instance
        if (templateInstance.getName().equals(
          "LargeOrderApprover")) {
```

문제를 발생시키기 위해 이 예제에서는 표현식에 호환 가능하지 않은 값으로 매개변수를 설정합니다. 이 매개변수는 정수일 것으로 예상하지만, 문자열이 전달됩니다.

```
          // Get the parameter from the
          template instance
          ParameterValue parameter =
          templateInstance
            .getParameterValue("par
            am1");

          // Set an incorrect value for this
          parameter
          // This will cause a validation
          error
          parameter.setValue("$3500");
          out.println("Incorrect parameter
          value set");
          break;
        }
      }
    }
    // This code should never be reached because of the
    error
    // introduced
    // above

    // With the condition value and actions updated, the
    business
    // rule
    // group can be published.
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the list
    publishList.add(brg);

    // Publish the list with the updated business rule
    group
    BusinessRuleManager.publish(publishList, true);
  }
```

ValidationException이 발생할 수 있으며 예외에서 문제점을 검색할 수 있습니다. 각 문제점의 경우, 오류가 발생했는지 판별하기 위해 오류를 검사할 수 있습니다. 메시지를 인쇄하거나 적절한 조치를 취할 수 있습니다.

```
  } catch (ValidationException e) {
  out.println("Validation Error");

  List<Problem> problems = e.getProblems();

  Iterator<Problem> problemIterator = problems.iterator();

  // Check the list of problems for the appropriate error and
  // perform the appropriate action, for example report error
               // or correct error
  while (problemIterator.hasNext()) {
    Problem problem = problemIterator.next();
    ValidationError error = problem.getErrorType();

    // Check for specific error value
    if (error == ValidationError.TYPE_CONVERSION_ERROR) {
      // Handle this error by reporting the problem
      out
        .println("Problem: Incorrect value
        entered for a parameter");
      return out.toString();
    }
    // else if....
    // Checks can be done for other errors and the
    // appropriate error message or action can be
    performed
    // correct the problem
  }
} catch (BusinessRuleManagementException e) {
  out.println("Error occurred.");
e.printStackTrace();
}
return out.toString();
}
}
```

예제

예제 14의 웹 브라우저 출력.

예제 **14** 실행

```
비즈니스 규칙 그룹이 검색되었음
규칙 세트가 검색되었음
유효성 검증 오류
문제점: 매개변수에 입력된 잘못된 값
```

예제 15: 비즈니스 규칙 그룹에서 오류 핸들

비즈니스 규칙 그룹이 공개되었을 때 발생하는 문제점을 처리하는 방법을 보여줄 때 이 예제는 예제 14와 유사합니다. 문제점을 판별할 수 있는 방법을 표시하며 올바른 메시지를 인쇄하거나 조치를 수행하게 합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
```

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example15
{
static Formatter out = new Formatter();

static public String executeExample15()
{
 try
 {
  out.clear();

 // Retrieve a business rule group by target namespace and
 name
 List<BusinessRuleGroup> brgList = BusinessRuleManager
  .getBRGsByTNSAndName(
  "http://BRSamples/com/ibm/websphere
  /sample/brules",
  QueryOperator.EQUAL,
  "ApprovalValues",
  QueryOperator.EQUAL, 0, 0);
  if (brgList.size() > 0)
  {
   // Get the first business rule group from the list
   // This should be the only business rule group in the
   list as
   // the combination of target namespace and name are
   unique
   BusinessRuleGroup brg = brgList.get(0);
   out.println("Business Rule Group retrieved");

   // Get the operation of the business rule group that
   // has the business rule that will be modified as
   // the business rules are associated with a specific
   // operation
   Operation op = brg.getOperation("getApprover");

   // Retrieve specific rule by name
```

```
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
 "getApprover", QueryOperator.EQUAL, 0,
 0);

// Get the specific rule
RuleSet ruleSet = (RuleSet) ruleList.get(0);
 out.println("Rule Set retrieved");

RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
 ruleBlock.iterator();

// Search through the rules to find the rule to
change
while (ruleIterator.hasNext())
{
 RuleSetRule rule = ruleIterator.next();

 // Check that the rule was defined with a
 template
 // as it can be changed.
 if (rule instanceof
 RuleSetTemplateInstanceRule)
 {
  // Get the template rule instance
  RuleSetTemplateInstanceRule
  templateInstance =
  (RuleSetTemplateInstanceRule) rule;

  // Check for the correct template rule
  instance
  if (templateInstance.getName().equals(
    "LargeOrderApprover"))
  {
   // Get the parameter from the
   template instance
   ParameterValue parameter =
   templateInstance
    .getParameterValue("par
    am1");

   // Set the value for this parameter
   // This value is in the correct
   format and will
   // not cause a validation error
   parameter.setValue("4000");
   out.println("Rule set parameter
   value on set correctly");
   break;
   }
  }
}
```

규칙 세트가 올바른지 확인하기 위해 유효성 검증 메소드가 호출될 수 있습니다. 유효성 검증 메소드는 모든 오브젝트에서 사용 가능하며 문제점을 판별하기 위해 검사될 수

있는 문제점 목록을 리턴합니다. 오브젝트에서 유효성 검증 호출 시, 마찬가지로 포함된 모든 오브젝트에서 유효성 검증 메소드가 호출됩니다.

```
// Validate the changes made the rule set
List<Problem> problems = ruleSet.validate();
out.println("Rule set validated");

// No errors should occur for this test case, however,
// check if there are problems and then
// perform the correct action to recover or report
// the error
if (problems != null)
{
 Iterator<Problem> problemIterator =
 problems.iterator();

 while (problemIterator.hasNext())
 {
  Problem problem = problemIterator.next();

  if (problem instanceof
  ProblemStartDateAfterEndDate)
  {
   out
   .println("Incorrect
    value entered for a
     parameter");
   return out.toString();
  }
 }
} else
{
 out.println("No problems found for the rule
 set");
}
// Get the list of available rule targets
List<BusinessRule> ruleList2 =
op.getAvailableTargets();

// Get the first rule that will be scheduled
incorrectly
BusinessRule rule = ruleList2.get(0);

// The error condition will be to set the end time
for a
// scheduled rule to be 1 hour before the start time
// This will cause a validation error
Date future = new Date();
long futureTime = future.getTime() - 360000;

// Get the operation selection list to add the
incorrectly
// scheduled item
OperationSelectionRecordList opList = op
.getOperationSelectionRecordList();

// Create a new scheduled rule instance
// No error is thrown until validated or a publish
```

```
    // occurs as more changes might be made
    OperationSelectionRecord newRecord = opList
     .newOperationSelectionRecord(new Date(),
     new Date(
     futureTime), rule);
```

잘못된 날짜 세트로 레코드가 추가되면, 오류를 야기하지 않습니다. 가능한 겹침이 발생하거나 변경 중인 프로세스에 있는 조작에 대해서는 선택사항 레코드가 설정되지 않습니다. 조작 선택사항 레코드가 있는 비즈니스 규칙 그룹이 공개되면 오류가 발생합니다. 오브젝트를 공개하기 전에 유효성 검증 메소드가 호출되며 오류가 있는 경우 예외가 발생합니다.

```
    // Add the scheduled rule instance to the operation
    // No error here either
    opList.addOperationSelectionRecord(newRecord);
    out.println("New selection record added with
    incorrect schedule");

    // With the condition value and actions updated, the
    business
    // rule
    // group can be published.
    // Use the original list or create a new list
    // of business rule groups
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Add the changed business rule group to the list
    publishList.add(brg);

    // Publish the list with the updated business rule
    group
    BusinessRuleManager.publish(publishList, true);
   }
} catch (ValidationException e) {
out.println("Validation Error");

List<Problem> problems = e.getProblems();

Iterator<Problem> problemIterator = problems.iterator();
// There might be multiple problems
// Go through the problems and handle each one or
// report the problem
while (problemIterator.hasNext())
{
 Problem problem = problemIterator.next();

 // Each problem is a different type that can be
 compared
 if (problem instanceof ProblemStartDateAfterEndDate)
 {
  out
  .println("Rule schedule is
   incorrect. Start date is after end
   date.");
  return out.toString();
```

```
     }
    // else if....
    // Checks can be done for other errors and the
    // appropriate error message or action can be
    performed
    // correct the problem
  }
 }catch (BusinessRuleManagementException e)
 {
  out.println("Error occurred.");
e.printStackTrace();
 }
 return out.toString();
 }
}
```

예제

예제 15의 웹 브라우저 출력.

예제 **15** 실행

```
비즈니스 규칙 그룹이 검색되었음
규칙 세트가 검색되었음
규칙 세트 매개변수 값 제대로 설정
규칙 세트 유효성 검증함
유효성 검증 오류
규칙 스케줄이 잘못되었습니다. 시작 날짜가 종료 날짜 이후입니다.
```

조회 예제 추가

다음 예제는 예제 1-15를 포함하는 응용프로그램에 포함되지 않지만, 비즈니스 규칙 그룹을 검색하기 위해 조회를 작성하는 예제를 더 많이 제공합니다.

이 예제에서 다른 특성 및 와일드카드 값(_,%)은 다른 연산자(AND, OR, LIKE, NOT_LIKE, EQUAL 및 NOT_EQUAL)와 함께 사용됩니다.

예제

예제에서, 4개의 비즈니스 규칙 그룹의 여러 조합 간에 리턴하는 조회가 수행됩니다. 조회에서 사용되는 비즈니스 규칙 그룹의 여러 속성 및 특성을 이해하는 것이 중요합니다.

```
이름: BRG1
Targetnamespace : http://BRG1/com/ibm/br/rulegroup
특성:
organization, 8JAAdepartment, claimsID, 00000567region:
SouthCentralRegion
관리자: Joe Bean

이름: BRG2
Targetnamespace : http://BRG2/com/ibm/br/rulegroup
특성:
organization, 7GAAdepartment, accountingID, 0000047ID_cert45,
ABCregion: NorthRegion
```

```
이름: BRG3
Targetnamespace : http://BRG3/com/ibm/br/rulegroup
특성:
organization, 7FABdepartment, financeID, 0000053ID_app45,
DEFregion: NorthCentralRegion

이름: BRG4
Targetnamespace : http://BRG4/com/ibm/br/rulegroup
특성:
organization, 7HAAdepartment, shippingID, 0000023ID_app45,
GHIregion: SouthRegion
```

단일 특성별 조회:

단일 특성별 조회의 예제입니다.

```
List<BusinessRuleGroup> brgList = null;

brgList = BusinessRuleManager.getBRGsBySingleProperty(
  "department", QueryOperator.EQUAL,
  "accounting", 0, 0);
// Returns BRG2
```

특성별 및 값의 처음과 끝에 있는 와일드카드별(%) 비즈니스 규칙 그룹 조회:

특성별 및 값의 처음과 끝에 있는 와일드카드별(%) 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Query Prop AND Prop
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "region", QueryOperator.LIKE,
  "%Region");

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode(
  "ID", QueryOperator.LIKE,
  "000005%");

QueryNode queryNode =
QueryNodeFactory.createAndNode(leftNode,
  rightNode);

brgList =
BusinessRuleManager.getBRGsByProperties(queryNode, 0, 0);
// Returns BRG1 and BRG3
```

특성별 및 와일드카드별('_') 비즈니스 규칙 그룹 조회:

특성별 및 와일드카드별(%) 비즈니스 규칙 그룹 조회의 예제입니다.

```
brgList = BusinessRuleManager.getBRGsBySingleProperty("ID",
QueryOperator.LIKE, "00000_3", 0, 0);

// Returns BRG3 and BRG4
```

다중 와일드카드('_' 및 '%')와 특성별 비즈니스 규칙 그룹 조회:

다중 와일드카드('_' 및 '%')와 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("region",
QueryOperator.LIKE, "__uth%Region",
  0, 0);

// Returns BRG1 and BRG4
```

NOT_LIKE 연산자 및 와일드카드('_')에 의한 비즈니스 규칙 그룹 조회:

NOT_LIKE 연산자 및 와일드카드('_')에 의한 비즈니스 규칙 그룹 조회의 예제입니다.

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7__A", 0, 0);

// Returns BRG1 and BRG3

brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7%", 0, 0);

// Returns BRG1
```

NOT_EQUAL 연산자에 의한 비즈니스 규칙 그룹 조회:

NOT_EQUAL 연산자에 의한 비즈니스 규칙 그룹 조회의 예제입니다.

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("department",
QueryOperator.NOT_EQUAL,
  "claims", 0, 0);
// Returns BRG1
```

PropertyIsDefined에 의한 비즈니스 규칙 그룹 조회:

PropertyIsDefined에 의한 비즈니스 규칙 그룹 조회의 예제입니다.

```
PropertyIsDefinedQueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

brgList = BusinessRuleManager.getBRGsByProperties(node, 0,
0);

// Returns BRG1
```

NOT PropertyIsDefined에 의한 비즈니스 규칙 그룹 조회:

NOT PropertyIsDefined에 의한 비즈니스 규칙 그룹 조회의 예제입니다.

```
// NOT Prop
QueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

NotNode notNode = QueryNodeFactory.createNotNode(node);

brgList = BusinessRuleManager.getBRGsByProperties(notNode,
0, 0);

// Returns BRG1
```

단일 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회:

단일 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
  QueryOperator.LIKE, "00000%");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
  notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

AND 연산자와 결합된 다중 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회:

AND 연산자와 결합된 다중 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// NOT Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "cla%");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

AndNode andNode = QueryNodeFactory.createAndNode(notNode,
```

```
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG1 and BRG2
```

OR 연산자와 결합된 다중 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회:

OR 연산자와 결합된 다중 NOT 노드와 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// NOT Prop OR NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department", QueryOperator.EQUAL,
  "claims");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

다중 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

다중 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop AND Prop) AND (Prop AND Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode,rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE,"000004_");

QueryNode leftNode2 =
```

```
QueryNodeFactory.createPropertyQueryNode("region",
QueryOperator.EQUAL,
  "NorthRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft,andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

AND 및 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

AND 및 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop AND Prop) OR (Prop AND NOT Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode, rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "8JAA");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE, "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, notNode);

OrNode orNode = QueryNodeFactory.createOrNode(andNodeLeft,
andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG2 and BRG3
```

AND 및 NOT 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

AND 및 NOT 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Prop AND NOT (Prop AND Prop)
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE, "000005%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.EQUAL,
  "8JAA");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG3
```

NOT 및 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

NOT 및 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// NOT (Prop AND Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8_A_");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
rightNode);
```

```
brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Prop AND (Prop AND (Prop AND Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

PropertyIsDefinedQueryNode node2 =
QueryNodeFactory.createPropertyIsDefinedQueryNode("ID_cert4
5");

AndNode andNode = QueryNodeFactory.createAndNode(node2,
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);
// Returns BRG2
```

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop AND (Prop AND Prop)) AND Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
"___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.LIKE,
"7%");

QueryNode leftNode2 =
```

```
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE,
"%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_app45",QueryOp
erator.LIKE, "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

중첩 AND 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 AND 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Prop AND (Prop AND (Prop AND NOT Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,notNode);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "AB_");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
```

```
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 AND 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop AND (Prop AND Prop)) AND Prop - Return empty
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

//Returns no BRGs
```

중첩 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop OR (Prop OR Prop)) OR Prop

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
```

```
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,rightNode2);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG1
```

중첩 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 OR 연산자와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// (Prop OR (Prop OR NOT Prop)) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,notNode);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);

QueryNode leftNode =
```

```
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

중첩 OR 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 OR 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// Prop OR NOT(Prop OR Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,
  rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft = QueryNodeFactory.createOrNode(leftNode,
notNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG3
```

중첩 OR 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회:

중첩 OR 연산자 및 NOT 노드와 결합된 다중 특성별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// NOT(Prop OR Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(notNode,leftNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG2 and BRG4
```

AND 연산자와 결합된 노드 목록별 비즈니스 규칙 그룹 조회:

AND 연산자와 결합된 노드 목록별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// AND list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");
```

```
list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7H%");

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

AND 연산자와 결합된 NOT 노드 및 노드 목록별 비즈니스 규칙 그룹 조회:

AND 연산자와 결합된 NOT 노드 및 노드 목록별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// AND list with a notNode
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

NotNode notNode =

QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);
```

```
brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Return BRG4
```

OR 연산자와 결합된 노드 목록별 비즈니스 규칙 그룹 조회:

OR 연산자와 결합된 노드 목록별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// OR list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG3
```

OR 연산자와 결합된 Not 노드 및 노드 목록별 비즈니스 규칙 그룹 조회:

OR 연산자와 결합된 Not 노드 및 노드 목록별 비즈니스 규칙 그룹 조회의 예제입니다.

```
// OR list with Not node
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
```

```
  QueryOperator.LIKE,
  "8%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

list.add(leftNode2);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

공통 조작 클래스

이 절에는 공통 조작을 단순화하기 위해 예제에서 사용된 추가 클래스가 포함되어 있습니다.

포맷터 클래스

이 클래스는 다른 예제를 표시하는 데 도움이 되는 다양한 메소드를 제공합니다. 출력을 형식화하기 위해 각기 다른 HTML 태그를 추가합니다.

```
package com.ibm.websphere.sample.brules.mgmt;

public class Formatter {

private StringBuffer buffer;

public Formatter()
{
  buffer = new StringBuffer();
}

public void println(Object o)
{
  buffer.append(o);
  buffer.append("<br>₩n");
  }

  public void print(Object o)
```

```
  {
    buffer.append(o);
  }

  public void println Bold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b><brbr>₩n");
  }

  public void printBold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b>");
  }

  public String toString()
  {
    return buffer.toString();
  }

  public void clear()
  {
    buffer = new StringBuffer();
  }
}
```

RuleArtifactUtility 클래스

이 유틸리티 클래스는 공용 메소드입니다. 첫 번째 공용 메소드는 의사결정 테이블을 인쇄하기 위해 사용됩니다. 이 메소드는 의사결정 테이블에 대한 조건 및 조치를 인쇄하기 위한 재귀 호출을 사용하는 개인용 메소드를 사용합니다. 두 번째 공용 메소드는 규칙 세트를 인쇄하기 위해 사용됩니다.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.RuleTemplate;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableRule;
import
com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
```

```
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionTermDefinition;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class RuleArtifactUtility
{
  static Formatter out = new Formatter();

  /*
  Method to print out a decision table with the conditions and
  actions printed out in a HTML tabular format. The conditions
  and actions are printed out with a separate method that
  recursively works through the case edges of the decision
  tables.
  */

  public static String printDecisionTable(BusinessRule
  ruleArtifact)
  {
    out.clear();
    out.printlnBold("Decision Table");
    DecisionTable decisionTable = (DecisionTable)
    ruleArtifact;
    out.println("Name: " +
    decisionTable.getName());
    out.println("Namespace: " +
    decisionTable.getTargetNameSpace());

    // Output the init rule for the decision table
    before
    // working through the table of conditions and
    actions
    DecisionTableRule initRule =
    decisionTable.getInitRule();
    if (initRule != null)
    {
      out.printBold("Init Rule: ");
      out.println(initRule.getName());
      out.println("Display Name: " +
      initRule.getDisplayName());
      out.println("Description: " +
      initRule.getDescription());
      // The expanded user presentation
      will automatically populate the
      // presentation with the parameter
      values and can be used for
      // display if the init rule was
      defined with a template. If no
      // template was defined the
      expanded user presentation
```

```
// is the same as the regular
presentation.
out.println("Extended User
Presentation: "
  +
initRule.getExpandedUse
rPresentation());
// The regular user presentation
will have placeholders in the
// string where the
// parameter can be substituted if
the init rule was defined with a
// template
// If the rule was not defined with
a template, the user
// presentation will only
// be a string without
placeholders. The placeholders are
of a
// format of {n} where
// n is the index (zero-based) of
the parameter in the template. This
// value
// can be used to create an
interface for editing where there
are
// fields with
// the parameter values available
for editing
out.println("User Presentation: " +
initRule.getUserPresentation());
// Init rules might be defined with
or without a template
// Check to make sure a template
was used before trying
// to access the parameters
if (initRule instanceof
DecisionTableTemplateInstanceRule)
{
  DecisionTableTemplateIn
  stanceRule
  templateInstance =
  (DecisionTableTemplateI
  nstanceRule) initRule;

  RuleTemplate template =
  templateInstance.getRul
  eTemplate();

  List<Parameter>
  parameters =
  template.getParameters(
  );
  Iterator<Parameter>
  paramIterator =
  parameters.iterator();

  Parameter parameter =
```

```
        null;

        while
        (paramIterator.hasNext(
        )) {
        parameter =
        paramIterator.next();

         out.println("Parameter
        Name: " +
        parameter.getName());
        out.println("Parameter
        Value: "
        +
        templateInstance.getPar
        ameterValue(parameter
        .getName()));
        }
      }
    }
    // For the rest of the decision table, start at
    the root and
    // recursively work through the different case
    edges and
    // actions
    TreeBlock treeBlock =
    decisionTable.getTreeBlock();
    TreeNode treeNode = treeBlock.getRootNode();

    printDecisionTableConditionsAndActions(treeNode
    , 0);
    out.println("");
    return out.toString();
  }
  /*Method to recursively work through the case edges and print
  out the conditions and actions.
*/
  static private void printDecisionTableConditionsAndActions(
    TreeNode treeNode, int indent)
  {
    out.print("<table border=₩"1₩">");
    if (treeNode instanceof ConditionNode)
    {
      // Get the case edges for the
      current TreeNode
      // and for each case edge print out
      the conditions
      ConditionNode conditionNode =
      (ConditionNode) treeNode;

      List<CaseEdge> caseEdges =
      conditionNode.getCaseEdges();
      Iterator<CaseEdge> caseEdgeIterator
      = caseEdges.iterator();

      CaseEdge caseEdge = null;

      while (caseEdgeIterator.hasNext())
```

```
{
  out.print("<tr>");
  // If this is the start
  of the conditions for the
  // condition node,
  print out the condition term
  if (indent == 0)
  {
  out.print("<td>");

  TreeConditionTermDefinition
  termDefinition =
  conditionNode
  .getTermDefinition();

  out.print(termDefinitio
  n.getUserPresentation()
  );
  out.print("</td>");
  indent++;
  } else {
  // After the condition
  term has been printed
  for a
  // case edge skip for
  the rest of the case
  edges
  out.print("<td></td>");
  }

  caseEdge =
  caseEdgeIterator.next()
  ;

  out.print("<td>");

  // Check if the
  caseEdge is defined by
  a template
  if
  (caseEdge.getValueDefin
  ition() != null)
  {
  TemplateInstanceExpress
  ion templateInstance =
  caseEdge
  .getValueTemplateInstan
  ce();

  out.println(templateIns
  tance.getExpandedUserPr
  esentation());

  TreeConditionValueDefin
  ition valueDef =
  caseEdge
  .getValueDefinition();
```

```
out.println(valueDef.ge
tUserPresentation());

Template template =
templateInstance.getTem
plate();

// Get the parameters
for the template
definition and
// print out the
parameter names and
values
List<Parameter>
parameters =
template.getParameters(
);
Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =
templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
(paramIterator.hasNext(
) &&
paramValues.hasNext())
{
parameter =
paramIterator.next();
parameterValue =
paramValues.next();

out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
  +
parameterValue.getValue
());
}
}

out.print("</td><td>");
// Print the child node
```

```
for the caseEdge
printDecisionTableCondi
tionsAndActions(caseEdg
e.getChildNode(),
0);

out.print("</td></tr>")
;
}


// Add Otherwise condition if it
exists
TreeNode otherwise =
conditionNode.getOtherwiseCase();

if (otherwise != null)
{
  out.print("<tr><td></td>
  <td>Otherwise</td><td>
  ");
  // Print the Otherwise
  ConditionNode
  printDecisionTableCondi
  tionsAndActions(otherwi
  se, 0);
  out.print("</td></td>")
  ;
}
out.print("</table>");
} else {
// ActionNode has been found and
different logic is needed
// to print out the TreeActions
ActionNode actionNode =
(ActionNode) treeNode;
List<TreeAction> treeActions =
actionNode.getTreeActions();

Iterator<TreeAction>
treeActionIterator =
treeActions.iterator();

TreeAction treeAction = null;

// The ActionNode can contain
multiple TreeActions to
// print out
while
(treeActionIterator.hasNext())
{
  out.print("<tr>");
  treeAction =
  treeActionIterator.next
  ();

  TreeActionTermDefinitio
  n treeActionTerm =
```

```
treeAction
.getTermDefinition();

if (indent == 0) {
out.print("<td>");
out.print(treeActionTer
m.getUserPresentation()
);
out.print("</td>");
}
out.print("<td>");
TemplateInstanceExpress
ion templateInstance =
treeAction
.getValueTemplateInstan
ce();

// Check that a
template was specified
for
// the TreeAction
before working with the
// parameter name and
values
if (templateInstance !=
null) {
out.println(templateIns
tance.getExpandedUserPr
esentation());

Template template =
templateInstance.getTem
plate();

List<Parameter>
parameters =
template.getParameters(
);

Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =
templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
```

```
        (paramIterator.hasNext(
        ) &&
        paramValues.hasNext())
        {
        {parameter =
        paramIterator.next();
        parameterValue =
        paramValues.next();

        out.println(" Parameter
        Name: " +
        parameter.getName());
        out.println(" Parameter
        Value: "
        +
        parameterValue.getValue
        ());

        }
        } else
        {
        // If a template was
        not used, the only item
        that is
        // available is the
        UserPresentation if it
        was
        // specified when the
        rule was created
        out.print(treeAction.ge
        tValueUserPresentation(
        ));
        }

        out.print("</td></tr>")
        ;
      }
      out.print("</table>");
    }
  }
  /*
    Method to print out a rule set
  */
public static String printRuleSet(BusinessRule
ruleArtifact)
{
   out.clear();
   out.printlnBold("Rule Set");
   RuleSet ruleSet = (RuleSet) ruleArtifact;
   out.println("Name: " + ruleSet.getName());
   out.println("Namespace: " +
   ruleSet.getTargetNameSpace());

   // The rules in a rule set are contained in a
   rule block
   RuleBlock ruleBlock =
   ruleSet.getFirstRuleBlock();
```

```
Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

RuleSetRule rule = null;

// Iterate through the rules in the rule block.
while (ruleIterator.hasNext())
{
  rule = ruleIterator.next();
  out.printBold("Rule: ");
  out.println(rule.getName());
  out.println("Display Name: " +
  rule.getDisplayName());
  out.println("Description: " +
  rule.getDescription());
  // The expanded user presentation
  will automatically populate the
  // presentation with the parameter
  values and can be used for
  // display if the rule was defined
  with a template. If no
  // template was defined the
  expanded user presentation
  // is the same as the regular
  presentation.
  out.println("Expanded User
  Presentation: "
    +
  rule.getExpandedUserPre
  sentation());
  // The regular user presentation
  will have placeholders in the
  // string where the parameter can
  be substituted if the rule
  // was defined with a template. If
  the rule was not defined with
  // a template, the user
  presentation will only be a string
  // without placeholders. The
  placeholders are of a format of {n}
  // where n is the index (zerobased)
  of the parameter in the
  // template. This value can be used
  to create an interface for
  // editing where there are fields
  with the parameter values
  // available for editing
  out.println("User Presentation: " +
  rule.getUserPresentation());

  // Check if the rule was defined
  with a template
  if (rule instanceof
  RuleSetTemplateInstanceRule) {
    RuleSetTemplateInstance
    Rule templateInstance =
    (RuleSetTemplateInstanc
    eRule) rule;
```

```
            RuleSetRuleTemplate
            template =
            templateInstance
            .getRuleSetRuleTemplate
            ();

            List<Parameter>
            parameters =
            template.getParameters(
            );
            Iterator<Parameter>
            paramIterator =
            parameters.iterator();

            Parameter parameter =
            null;

            // Retrieve all of the
            parameters and output
            the name and value
            while
            (paramIterator.hasNext(
            ))
            {
            parameter =
            paramIterator.next();

            out.println("Parameter
            Name: " +
            parameter.getName());
            out.println("Parameter
            Value: "
              +
            templateInstance.getPar
            ameterValue(
            parameter.getName()).ge
            tValue());
            }
          }
        }
        out.println("");
        return out.toString();
      }
}
```

위젯 프로그래밍

Business Space는 IBM의 다양한 비즈니스 프로세스 관리 제품에 대해 사용 가능한 위젯을 제공합니다. 그러나 사용자 고유의 위젯을 작성하여 IBM의 위젯과 통합할 수도 있습니다. 다음 참조에서는 사용자 위젯을 작성하는 이유와 작성 방법을 설명하고 사용자 고유 위젯 작성 시 사용할 수 있는 API에 대한 참조사항을 제공합니다.

Business Space Information Center에서 다음 주제를 참조하십시오.

- 위젯 개발 개요
- 위젯 프로그래밍 안내

제 4 장 비즈니스 프로세스 및 타스크용 클라이언트 응용프로그램 개발

모델 작성 도구를 사용하여 비즈니스 프로세스 및 타스크를 빌드 및 전개할 수 있습니다. 이들 프로세스와 타스크는 런타임 시 상호작용합니다. 예를 들어, 프로세스가 시작되거나 타스크가 청구되고 완료됩니다. Business Process Choreographer Explorer를 사용하여 프로세스 및 타스크와 상호작용하거나 Business Process Choreographer API를 사용하여 이 상호작용에 대한 사용자 정의된 클라이언트를 개발할 수 있습니다.

이 태스크 정보

이러한 클라이언트는 Business Process Choreographer Explorer JSF(JavaServer Faces) 컴포넌트를 사용하는 웹 클라이언트, 웹 서비스 클라이언트 또는 EJB(Enterprise JavaBeans) 클라이언트가 될 수 있습니다. Business Process Choreographer는 클라이언트를 개발하기 위한 EJB(Enterprise JavaBeans) API 및 웹 서비스용 인터페이스를 제공합니다. EJB API에는 다른 EJB 응용프로그램을 포함한 모든 Java 응용프로그램으로 액세스할 수 있습니다. 웹 서비스용 인터페이스는 Java 환경 또는 Microsoft .Net 환경에서 액세스할 수 있습니다.

비즈니스 프로세스 및 휴먼 타스크와 상호작용에 사용되는 프로그래밍 인터페이스 비교

EJB(Enterprise JavaBean), 웹 서비스 및 JMS(Java Message Service), 그리고 REST(Representational State Transfer Services) 일반 프로그래밍 인터페이스는 비즈니스 프로세스 및 휴먼타스크와 상호작용하는 클라이언트 응용프로그램 빌드에 사용 가능합니다. 이러한 각 인터페이스는 서로 다른 특성을 가집니다.

사용자가 선택하는 프로그래밍 인터페이스는 클라이언트 응용프로그램이 제공해야 하는 기능, 기존의 일반 사용자 클라이언트 인프라가 있는지 여부, 휴먼 워크플로우를 처리할지 여부를 포함하여 여러 요소에 따라 결정됩니다. 사용할 인터페이스를 결정하는 데 도움이 되도록 다음 표는 EJB, 웹 서비스, JMS 및 REST 프로그래밍 인터페이스의 특성을 비교합니다.

	EJB 인터페이스	웹 서비스 인터페이스	**JMS** 메시지 인터페이스	**REST** 인터페이스
기능	이 인터페이스는 비즈니스 프로세스 및 휴먼 타스크 둘 다에 사용 가능합니다. 프로세스 및 타스크와 일반적으로 작업하는 클라이언트를 빌드하려면 이 인터페이스를 사용하십시오.	이 인터페이스는 비즈니스 프로세스 및 휴먼 타스크 둘 다에 사용 가능합니다. 알려진 프로세스 및 타스크 세트에 대해 클라이언트를 빌드하려면 이 인터페이스를 사용하십시오.	이 인터페이스는 비즈니스 프로세스 전용입니다. 알려진 프로세스 세트에 대해 메시징 클라이언트를 빌드하려면 이 인터페이스를 사용하십시오.	이 인터페이스는 비즈니스 프로세스 및 휴먼 타스크 둘 다에 사용 가능합니다. 알려진 프로세스 및 타스크 세트에 대해 웹 2.0 스타일 클라이언트를 빌드하려면 이 인터페이스를 사용하십시오.

	EJB 인터페이스	웹 서비스 인터페이스	JMS 메시지 인터페이스	REST 인터페이스
데이터 처리	비즈니스 오브젝트 메타데이터 액세스에 대한 스키마의 원격 아티팩트 로딩을 지원합니다. EJB 클라이언트 응용프로그램이 연결 대상인 WebSphere Process Server와 동일한 셀에서 실행 중인 경우, 프로세스 및 타스크의 비즈니스 오브젝트에 필요한 스키마가 클라이언트에서 사용 가능할 필요는 없으며, 원격 아티팩트 로더(RAL)를 사용하여 서버에서 로드할 수 있습니다. RAL은 클라이언트 응용프로그램이 전체 WebSphere Process Server 서버 설치에서 실행되는 경우 교차 셀로도 사용될 수 있습니다. 그러나 RAL은 클라이언트 응용프로그램이 WebSphere Process Server 클라이언트 설치에서 실행되는 경우에는 교차 셀 설정에 사용할 수 없습니다.	입력 데이터, 출력 데이터 및 변수에 대한 스키마 아티팩트는 클라이언트에서 해당 형식으로 사용 가능해야 합니다.	입력 데이터, 출력 데이터 및 변수에 대한 스키마 아티팩트는 클라이언트에서 해당 형식으로 사용 가능해야 합니다.	입력 데이터, 출력 데이터 및 변수에 대한 스키마 아티팩트는 클라이언트에서 해당 형식으로 사용 가능해야 합니다.
클라이언트 환경	WebSphere Process Server 설치 또는 WebSphere Process Server 클라이언트 설치.	웹 서비스 호출을 지원하는 모든 런타임 환경(Microsoft .NET 포함).	SCA JMS 가져오기를 사용하는 SCA 모듈을 포함하여 JMS 클라이언트를 지원하는 모든 런타임 환경.	REST 클라이언트를 지원하는 모든 런타임 환경.
보안	Java EE(Java Platform, Enterprise Edition) 보안.	웹 서비스 보안.	WebSphere Process Server 설치에 대한 Java EE(Java Platform, Enterprise Edition) 보안. 예를 들어, WebSphere MQ 보안 메커니즘을 사용하여 JMS 클라이언트 응용프로그램이 API 메시지를 넣는 큐를 보안할 수도 있습니다.	REST 메소드를 호출하는 클라이언트 응용프로그램은 적절한 HTTP 인증 메커니즘을 사용해야 합니다.

다중 프로토콜을 통해 조작을 공개할 수 있습니다. 서로 다른 프로토콜에서 동일한 조작을 사용하는 경우 다음과 같은 일반적인 고려사항을 검토하십시오.

- 웹 서비스 및 REST 인터페이스에서 PIID, AIID 및 TKIID와 같은 모든 오브젝트 ID는 문자열 유형으로 표시됩니다. EJB API 인터페이스에서만 모든 유형의 오브젝트 ID가 표시됩니다.
- 조작 과부하는 EJB 메소드에서만 사용되고 WSDL 조작에서는 사용되지 않습니다. 다중 WSDL 조작이 존재하는 경우도 있고, 생략(`minOccurs="0"`) 또는 널값(`nillable="true"`)을 통해 모든 변형 매개변수를 사용할 수도 있는 하나의 WSDL 조작이 존재하는 경우도 있습니다.
- 일부 EJB 메소드에서는 XML 네임스페이스 및 로컬 이름이 독립 매개변수로 전달됩니다. 대부분의 WSDL 조작에서는 QName XML 스키마 유형을 사용하여 이들 매개변수를 전달합니다.

- 장기 실행 중인 WSDL 요청-응답 조작(예: EJB 인터페이스의 callWithReplyContext 조작 또는 WSDL 인터페이스의 callAsync 조작)과의 비동기 상호작용은 JMS 인터페이스의 call 조작을 통해 표시됩니다.
- EJB 인터페이스는 포함된 필드의 Getter 및 Setter 메소드를 공개하는 API 오브젝트 세트를 리턴합니다. 웹 서비스 및 REST 인터페이스는 클라이언트에게 복합 유형(XML 또는 JSON) 문서를 리턴합니다.
- 휴먼 타스크를 조작하는 일부 휴먼 타스크 관리자 서비스 또한 휴먼 타스크를 호출하는 활동을 조작하는 비즈니스 플로우 관리자 서비스로 사용 가능합니다.

관련 태스크

EJB 클라이언트 응용프로그램 개발
EJB API는 WebSphere Process Server에 설치된 비즈니스 프로세스 및 휴먼 타스크로 작업하는 EJB 클라이언트 응용프로그램을 개발하기 위한 일반 메소드 세트를 제공합니다.

웹 서비스 API 클라이언트 응용프로그램 개발
Business Process Choreographer 웹 서비스 API를 통해 비즈니스 프로세스 응용프로그램 및 휴먼 타스크 응용프로그램에 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다. 클라이언트 응용프로그램 개발 프로세스는 웹 서비스 프록시 생성과 클라이언트 응용프로그램에 대한 보안 및 트랜잭션 정책 추가를 포함하여 많은 필수 및 선택적 단계로 구성됩니다.

JMS 클라이언트 응용프로그램 개발
JMS(Java Messaging Service) API를 통해 비동기적으로 비즈니스 프로세스 응용프로그램에 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

비즈니스 프로세스 및 타스크 데이터에 대한 조회

장기 실행 중 비즈니스 프로세스 및 휴먼 타스크의 인스턴스 데이터는 데이터베이스에 지속적으로 저장되며 조회로 액세스 가능합니다. 또한 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트의 템플리트 데이터는 조회 인터페이스를 통해 액세스될 수 있습니다.

EJB 조회 인터페이스, 조회 API 및 조회 테이블 API를 Business Process Choreographer와 함께 사용할 수 있습니다.

프로세스나 타스크 관련 데이터에 액세스하는 클라이언트에 따라 위 인터페이스 중 하나 이상을 선택할 수 있습니다. 타스크 및 프로세스 목록 데이터를 조회하기 위해 REST 및 웹 서비스 API를 Business Process Choreographer에서 사용할 수 있습니다. 그러나 볼륨이 큰 프로세스 목록 및 타스크 목록 조회의 경우 성능상 이유로 Business Process Choreographer EJB 조회 테이블 API 또는 REST 조회 테이블 API를 사용합니다.

프로세스 및 타스크 데이터 검색을 위한 프로그래밍 인터페이스 비교

Business Process Choreographer에서는 프로세스 및 타스크 데이터 검색을 위해 query table API와 query API를 제공합니다. 이러한 각 인터페이스는 서로 다른 특성을 가집니다.

선택하는 조회 인터페이스는 클라이언트 응용프로그램이 제공해야 하는 기능, 기존 일반 사용자 클라이언트 인프라가 있는지 여부 및 성능 고려사항을 포함한 여러 요소에 따라 다릅니다. 사용할 인터페이스를 결정하는 데 도움이 되도록 다음 표에서는 query table과 query 프로그래밍 인터페이스의 특성을 비교합니다.

특성	**query table API**	**query API**
가용성	비즈니스 플로우 관리자 EJB 인터페이스 및 REST 프로그래밍 인터페이스에서 query table API를 사용할 수 있습니다.	query API는 EJB, 웹 서비스, JMS 및 REST 프로그래밍 인터페이스에 사용할 수 있습니다.
컨텐츠 검색 메소드	API에서는 다음과 같은 메소드를 제공합니다. • queryEntities • queryEntityCount • queryRows • queryRowCount	API에서는 다음과 같은 메소드를 제공합니다. • query • queryAll • queryProcessTemplates • queryTaskTemplates
메타데이터 검색 메소드	API에서는 다음과 같은 메소드를 제공합니다. • getQueryTableMetaData • findQueryTableMetaData	API에서는 다음과 같은 메소드를 제공합니다. • QueryResultSet.getColumnType
조회 테이블 이름	조회 테이블 API가 실행되는 조회 테이블을 지정합니다. 한 번에 하나의 조회 테이블만 조회할 수 있습니다. 예: `queryEntities("CUST.TASKS", ...)`.	SELECT 절은 조회가 실행되는 사전정의된 데이터베이스 뷰 및 열을 지정합니다. 이 스펙은 SQL select 절과 비슷합니다. 예: `query("TASK.TKIID, TASK.STATE, WORK_ITEM.REASON", ...)`.
SELECT절 및 선택된 속성	조회 테이블 API의 필터 옵션을 사용하여 조회가 리턴되는 속성을 지정하십시오. 조회는 하나의 조회 테이블에 대해 실행되기 때문에 속성은 이름으로 고유하게 식별할 수 있습니다.	SELECT절을 사용하여 속성을 지정하십시오. 속성 이름의 구문은 *view_name.attribute_name*입니다. 예를 들어, 타스크 상태를 검색하려면 조회에 TASK.STATE를 지정하십시오.
WHERE절 및 필터	조회 테이블 API에서 queryCondition 특성을 사용하여 조회 결과를 더 자세히 필터링합니다. 조회 테이블 정의에 1차 조회 테이블 필터, 권한 필터 또는 조회 테이블 필터가 지정된 경우 조회 테이블에서 미리 필터링된 컨텐츠가 제공됩니다.	WHERE 절을 사용하여 조회 결과를 필터링하십시오.
WHERE절 및 선택 기준	조회 테이블 API에서는 이 양식의 조회 API WHERE 절이 필요하지 않습니다. 추가 필터링을 수행하려면 조회 테이블 API에서 queryCondition 특성을 사용하십시오. 조회 테이블 정의의 선택 기준은 첨부된 조회 테이블의 특정 특성을 선택합니다. 조회 API에서 WHERE 절을 통해 필터링 외에 이러한 선택이 수행됩니다.	선택 기준은 조회 API에 사용할 수 없습니다. 그러나 선택 기준은 예를 들면, QUERY_PROPERTY, TASK_CPROP 또는 TASK_DESC의 로케일이나 이름을 정의하는 WHERE 절의 일부와 비슷합니다. 예를 들어, `QUERY_PROPERTY.NAME='xyz'`의 WHERE 절은 QUERY_PROPERTY가 첨부된 조회 테이블의 조회 테이블 정의에서 `NAME='xyz'`를 선택 기준으로 지정하는 것과 같습니다.

특성	query table API	query API
작업 항목 및 권한	WORK_ITEM 조회 테이블을 사용하여 작업 항목에 액세스하십시오. AuthorizationOptions 오브젝트 또는 AdminAuthorizationOptions 오브젝트를 사용하여 조회 테이블 API와 조회 테이블 정의(조회 테이블 개발 시)에서 작업 항목의 사용을 사용자 정의할 수 있습니다. 예를 들어, TASK 조회 테이블 조회 시 모두 작업 항목을 제외시키려면 queryCondition 특성을 `WI.EVERYBODY=0`으로 지정하거나 AuthorizationOptions 특성에서 `setUseEverbody(Boolean.FALSE)`를 지정하십시오.	WORK_ITEM 보기를 사용하여 작업 항목에 액세스하십시오. 네 유형의 작업 항목(모든 사용자, 개인, 그룹 및 상속된 작업 항목) 모두가 조회 결과에 고려됩니다. 특정 유형의 작업 항목을 위해 작업 항목을 필터링하려면 WHERE 절을 사용자 정의하십시오. 예를 들어, 모든 사용자 작업 항목을 제외하려면 WHERE 절에 `WORK_ITEM.EVERYBODY=0`을 지정하십시오.
매개변수	복합 조회 테이블의 필터 및 선택 기준에 매개변수를 사용할 수 있습니다.	저장된 조회를 사용하지 않는 한 조회 API에 매개변수를 사용할 수 없습니다.
저장된 조회 및 조회 테이블	저장된 조회와 조회 테이블의 차이점은 저장된 조회는 하나의 특정 조회에 대해 정의되고 조회 테이블은 특정 조회 세트에 대해 정의된다는 점입니다. 예를 들어, 이 정보는 일반적으로 조회가 실행될 때만 사용할 수 있기 때문에 조회 테이블 정의에는 order-by절을 지정할 수 없습니다.	저장된 조회를 사용하여 사전정의된 옵션 세트가 포함된 조회를 실행할 수 있습니다.
구체화된 보기	구체화된 보기는 조회 테이블 API에 사용할 수 없습니다.	구체화된 보기는 데이터베이스 기술을 사용하여 조회에 대한 성능 향상을 제공합니다.
사용자 정의 테이블	추가 조회 테이블에서는 사용자 정의 테이블과 동일한 기능을 제공합니다.	사용자 정의 테이블은 Business Process Choreographer 데이터베이스 스키마 외부의 조회에서 데이터를 포함하는 데 사용됩니다.
queryAll 및 권한 옵션	AdminAuthorizationOptions 오브젝트는 AuthorizationOptions 오브젝트 대신 조회 테이블 API로 전달할 수 있는 queryAll 기능을 제공합니다. 호출자는 BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor 또는 TaskSystemMonitor에 있어야 합니다.	특정 사용자 또는 그룹의 작업 항목에 제한되지 않고 조회 결과의 모든 오브젝트를 리턴하기 위해 BPESystemAdministrator Java EE 역할이 있는 사용자가 사용할 수 있는 queryAll 메소드입니다.
자국어 지원	조회 테이블 속성 및 조회 테이블의 경우, 조회 테이블 사용 시 로컬화된 표시 이름과 설명을 사용할 수 있습니다.	선택한 보기의 열 이름이 데이터베이스에 표시되는 대로 또는 select 절에서 지정된 대로 리턴됩니다.

Business Process Choreographer에서 테이블 조회

조회 테이블은 Business Process Choreographer 데이터베이스 스키마에 포함된 데이터에 대한 타스크 및 프로세스 목록 조회를 지원합니다. 여기에는 Business Process Choreographer가 관리하는 휴먼 타스크 데이터 및 비즈니스 프로세스 데이터와 외부 비즈니스 데이터가 포함됩니다. 조회 테이블에서는 클라이언트 응용프로그램이 사용할 수 있는 Business Process Choreographer의 데이터에 대한 개념을 제공합니다. 이로 인해 클라이언트 응용프로그램은 조회 테이블의 실제 구현과 독립적이게 됩니다. 조회 테이블 정의는 Business Process Choreographer Container에 전개되며 조회 테이블 API를 사용하여 액세스할 수 있습니다.

세 가지 유형의 조회 테이블이 있습니다.

- 사전 정의된 조회 테이블
- 추가 조회 테이블
- 복합 조회 테이블

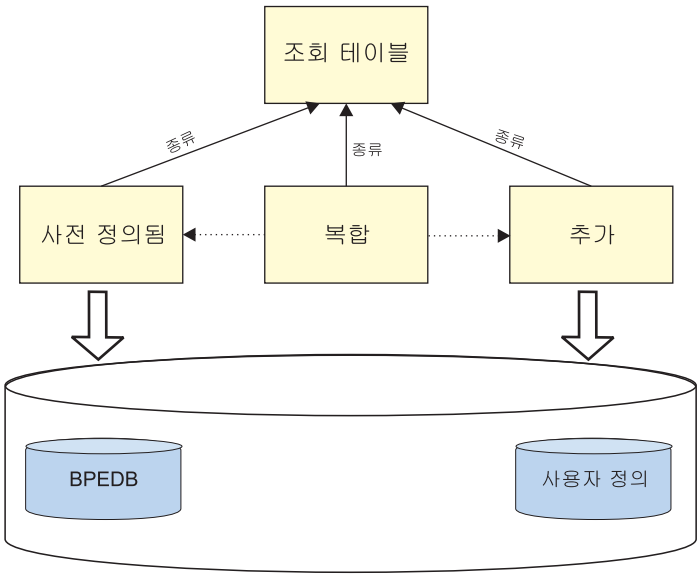


그림 72. Business Process Choreographer에서 테이블 조회

조회 테이블은 조회 테이블 런타임에 비슷한 모델을 사용하여 표시되며 조회 테이블 API를 사용하여 조회할 수 있습니다. 사전 정의된 조회 테이블과 보충 조회 테이블은 데이터베이스에 있는 테이블 또는 보기를 직접 가리키지만 복합 조회 테이블은 이 데이터의 파트를 작성하여 단일 조회 테이블에 나타냅니다.

조회 테이블을 사용하면 Business Process Choreographer의 기존 조회 인터페이스 및 사전 정의된 데이터베이스 보기가 향상됩니다. 조회 테이블의 특징은 다음과 같습니다.

- 성능 최적화 액세스 패턴을 사용하여 프로세스 및 타스크 목록 조회 실행을 위해 최적화됩니다.
- 필요한 정보에 대한 액세스를 단순화하고 통합합니다.
- 권한 및 필터 옵션을 세밀하게 구성할 수 있도록 합니다.

조회 테이블을 사용자 정의할 수 있습니다. 예를 들어, 특정 시나리오와 관련된 타스크 또는 프로세스 인스턴스만 포함되도록 조회 테이블을 구성할 수 있습니다. 큰 볼륨의 프로세스 목록 및 타스크 목록 조회와 같이 성능이 중요한 곳에서는 조회 테이블을 사용하는 것이 바람직합니다.

조회 테이블 빌더는 Eclipse 플러그인으로 제공되어 다음을 수행합니다.

- 복합 및 추가 조회 테이블 개발
- XML 형식으로 조회 테이블 정의 가져오기 및 내보내기

WebSphere Business Process Management SupportPacs 사이트에서 조회 테이블 빌더를 다운로드할 수 있습니다. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

사전정의된 조회 테이블

사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

사전정의된 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다. 조회 테이블 API를 사용하여 테이블에 액세스하면 조회 API를 사용할 때보다 많은 구성 옵션이 제공됩니다.

특성

사전정의된 조회 테이블에는 다음과 같은 특성이 있습니다.

표 25. 사전정의된 조회 테이블의 특성

특성	설명
이름	조회 테이블 이름은 사전 정의된 데이터베이스 보기 중 하나의 이름이며 대문자입니다(예: TASK).
속성	사전 정의된 조회 테이블의 속성은 조회에 사용할 수 있는 정보의 일부를 정의합니다. 이러한 속성은 사전 정의된 데이터베이스 뷰를 통해 지정되는 열의 이름(대문자)입니다. 이름과 유형으로 속성을 정의합니다. 유형은 다음 중 하나입니다. • **부울**: 부울 값 • **10진수**: 부동 소수점 숫자 • **ID**: 오브젝트 ID(예: TASK 조회 테이블 TASK의 TKIID) • **숫자**: 정수(short 또는 long) • **문자열**: 문자열 • **시간소인**: 시간소인

표 25. 사전정의된 조회 테이블의 특성 (계속)

특성	설명
권한	사전 정의된 조회 테이블에서는 인스턴스 기반 또는 역할 기반 권한을 사용합니다. • 인스턴스 데이터가 포함된 사전 정의된 조회 테이블에는 인스턴스 기반 권한이 필요합니다. 이는 조회를 수행하는 사용자의 작업 항목이 포함된 오브젝트만 리턴됨을 의미합니다. 그러나 AdminAuthorizationOptions 오브젝트를 사용하여 임의의 사용자의 작업 항목이 있는지만 확인하도록 확인 범위를 줄일 수 있습니다. 사용자는 비즈니스 플로우 관리자 EJB가 사용되는 경우 해당 조회에 대해 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator Java EE 역할이 있어야 합니다. • 템플리트 데이터가 있는 사전 정의된 조회 테이블에는 역할 기반 권한이 필요합니다. 즉 비즈니스 플로우 관리자 EJB가 사용되는 경우에는 BPESystemAdministrator Java EE 역할이 있는 사용자만 조회 테이블의 컨텐츠에 액세스할 수 있으며 휴먼 타스크 관리자 EJB가 사용되는 경우에는 TaskSystemAdministrator Java EE 역할이 있는 사용자만 액세스할 수 있습니다.

인스턴스 데이터가 있는 사전 정의된 조회 테이블

다음 표에서는 인스턴스 데이터가 포함된 사전정의된 조회 테이블에 대해 설명합니다. 조회 테이블은 다음과 같은 특성을 지닙니다.

- 복합 조회 테이블의 1차 조회로 사용할 수 있습니다.
- 직접 조회한 경우 인스턴스 기반 권한을 사용합니다. 이는 권한 정보를 저장하는 보기(즉, 사전정의된 WORK_ITEM 보기 또는 조회 테이블)와의 결합(SQL-)을 사용하여 수행됩니다.
- 인스턴스 데이터(예: 타스크 인스턴스 또는 프로세스 인스턴스의 데이터)를 포함합니다.

표 26. 인스턴스 데이터가 포함된 사전정의된 조회 테이블

인스턴스 데이터	조회 테이블 이름
프로세스 인스턴스의 활동에 대한 정보.	ACTIVITY
	ACTIVITY_ATTRIBUTE
	ACTIVITY_SERVICE
휴먼 타스크에 속하는 에스컬레이션에 대한 정보.	ESCALATION
	ESCALATION_CPROP
	ESCALATION_DESC
프로세스 인스턴스에 대한 정보.	PROCESS_ATTRIBUTE
	PROCESS_INSTANCE
	QUERY_PROPERTY

표 26. 인스턴스 데이터가 포함된 사전정의된 조회 테이블 (계속)

인스턴스 데이터	조회 테이블 이름
휴먼 타스크에 대한 정보.	TASK
	TASK_CPROP
	TASK_DESC

WORK_ITEM 조회 테이블에는 인스턴스 데이터도 포함되어 있지만 이 조회 테이블은 1차 조회 테이블이나 첨부된 조회 테이블로 사용할 수 없습니다. 인스턴스 기반 권한을 사용하는 조회 테이블 조회 시 작업 항목 정보를 내재적으로 사용할 수 있습니다. 즉, 인스턴스 기반 권한으로 조회 테이블을 조회하는 경우에는 WORK_ITEM 조회 테이블의 속성이 조회 테이블에서 명시적으로 지정되지 않은 경우에도 해당 속성을 사용할 수 있습니다.

템플리트 데이터가 있는 사전 정의된 조회 테이블

템플리트 데이터가 포함된 사전 정의된 조회 테이블에는 역할 기반 권한이 필요합니다. 해당 조회 테이블은 AdminAuthorizationOptions 오브젝트를 사용하여 관리자만 조회할 수 있습니다.

다음 표에서는 템플리트 데이터가 포함된 사전정의된 조회 테이블에 대해 설명합니다. 조회 테이블은 다음과 같은 특성을 지닙니다.

- 복합 조회 테이블의 1차 조회 테이블로 사용할 수 있습니다.
- 직접 조회되는 경우 역할 기반 권한을 사용하십시오. 즉 API 조회 메소드를 사용하는 호출자는 비즈니스 플로우 관리자 EJB가 사용되는 경우 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator Java EE 역할이 있어야 합니다. 또한 AdminAuthorizationOptions가 사용되어야 합니다.
- 템플리트 데이터를 포함하고 있습니다(예: 타스크 템플리트나 프로세스 템플리트의 템플리트 데이터).

표 27. 템플리트 데이터가 포함된 사전정의된 조회 테이블

템플리트 데이터	조회 테이블 이름
응용프로그램 컴포넌트에 대한 정보.	APPLICATION_COMP
에스컬레이션 템플리트에 대한 정보.	ESC_TEMPL
	ESC_TEMPL_CPROP
	ESC_TEMPL_DESC
프로세스 템플리트에 대한 정보.	PROCESS_TEMPLATE
	PROCESS_TEMPL_ATTR
타스크 템플리트에 대한 정보.	TASK_TEMPL
	TASK_TEMPL_CPROP
	TASK_TEMPL_DESC

관련 개념

『추가 조회 테이블』
Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

356 페이지의『복합 조회 테이블』
Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

364 페이지의『조회 테이블 개발』
Business Process Choreographer의 추가 및 복합 조회 테이블은 응용프로그램 개발 중에 조회 테이블 빌더를 사용하여 개발됩니다. 사전정의된 조회 테이블은 개발되거나 전개될 수 없습니다. Business Process Choreographer가 설치될 때 사용 가능하며 Business Process Choreographer 데이터베이스 스키마의 아티팩트에 대한 간단한 보기를 제공합니다.

385 페이지의『조회 테이블 조회』
비즈니스 플로우 관리자 EJB 및 REST API에서 사용 가능한 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회가 실행됩니다.

374 페이지의『조회 테이블에 대한 권한』
조회 테이블에서 조회를 실행할 때 인스턴스 기반 권한 또는 역할 기반 권한을 사용하거나 아무 권한도 사용하지 않을 수 있습니다.

추가 조회 테이블

Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

추가 조회 테이블은 Business Process Choreographer 데이터베이스의 데이터베이스 보기 또는 데이터베이스 테이블과 관련됩니다. 보충 조회 테이블은 고객 응용프로그램이 관리하는 비즈니스 데이터가 포함된 조회 테이블입니다. 추가 조회 테이블에서는 사전정의된 조회 테이블에 포함된 정보와 복합 조회 테이블에 있는 정보를 제공합니다.

추가 조회 테이블에는 다음과 같은 특성이 있습니다.

표 28. 추가 조회 테이블의 특성

특성	설명
이름	조회 테이블 이름은 Business Process Choreographer 설치 내에서 고유해야 합니다. 조회 실행 시 이 이름을 사용하여 조회되는 조회 테이블을 식별합니다. 조회 테이블은 *prefix.name*과 같이 정의되는 이름을 사용하여 고유하게 식별됩니다. *prefix.name*의 최대 길이는 28자입니다. 접두부는 예약된 접두부 `'IBM'`과 달라야 합니다(예: `'COMPANY.BUS_DATA'`). 테이블 이름의 끝에는 숫자를 사용하지 마십시오. 한 조회에서 단일 테이블이 여러 번 사용되는 경우, 테이블의 이름은 0 - 9 범위의 숫자를 사용하여 확장할 수 있습니다(예: CUSTOM_VIEW0, CUSTOM_VIEW1 등). 테이블 이름의 끝에 이미 숫자가 있는 경우에는 Business Process Choreographer가 이 숫자를 제거하므로 QueryUnknownTableException이 발생합니다.
데이터베이스 이름	데이터베이스에 있는 관련 테이블 또는 보기의 이름입니다. 대문자만 사용할 수 있습니다.
데이터베이스 스키마	데이터베이스에 있는 관련 테이블 또는 보기의 스키마입니다. 대문자만 사용할 수 있습니다. 데이터베이스 스키마는 Business Process Choreographer 데이터베이스의 데이터베이스 스키마와 달라야 합니다. 그러나 테이블 또는 보기는 Business Process Choreographer 데이터베이스에 액세스하는 데 사용되는 동일한 JDBC 데이터 소스로 액세스할 수 있어야 합니다.
속성	보충 조회 테이블의 속성은 조회에 사용할 수 있는 정보의 일부를 정의합니다. 이러한 속성은 관련 데이터베이스 테이블 또는 뷰에 있는 열의 관련 이름과 일치해야 합니다. 이름과 유형으로 속성을 정의합니다. 이름은 대문자로 정의됩니다. 유형은 다음 중 하나입니다. • **부울**: 부울 값 • **10진수**: 부동 소수점 숫자 • **ID**: 길이가 16바이트인 오브젝트 ID(예: TASK 조회 테이블의 TKIID) • **숫자**: 정수(short 또는 long) • **문자열**: 문자열 • **시간소인**: 시간소인
결합	결합은 복합 조회 테이블에 첨부된 경우 추가 조회 테이블에 정의되어야 합니다. 결합은 추가 조회 테이블의 정보를 1차 조회 테이블의 정보와 상관시키는 데 사용되는 속성을 정의합니다. 결합이 정의된 경우에는 소스 속성과 대상 속성의 유형이 동일해야 합니다.
권한	보충 조회 테이블에 권한이 지정되지 않아서 인증된 모든 사용자가 컨텐츠를 볼 수 있습니다.

관련 개념

351 페이지의『사전정의된 조회 테이블』
사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

『복합 조회 테이블』
Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

364 페이지의『조회 테이블 개발』
Business Process Choreographer의 추가 및 복합 조회 테이블은 응용프로그램 개발 중에 조회 테이블 빌더를 사용하여 개발됩니다. 사전정의된 조회 테이블은 개발되거나 전개될 수 없습니다. Business Process Choreographer가 설치될 때 사용 가능하며 Business Process Choreographer 데이터베이스 스키마의 아티팩트에 대한 간단한 보기를 제공합니다.

385 페이지의『조회 테이블 조회』
비즈니스 플로우 관리자 EJB 및 REST API에서 사용 가능한 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회가 실행됩니다.

374 페이지의『조회 테이블에 대한 권한』
조회 테이블에서 조회를 실행할 때 인스턴스 기반 권한 또는 역할 기반 권한을 사용하거나 아무 권한도 사용하지 않을 수 있습니다.

복합 조회 테이블

Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

복합 조회 테이블은 클라이언트 개발자가 설계하며, 조회 실행 시 최적화된 데이터 액세스를 위해 필터 및 권한 옵션을 상세하게 구성할 수 있습니다. 복합 조회 테이블은 타스크 및 프로세스 목록 조회를 위해 최적화된 SQL로 구현됩니다.

복합 조회 테이블은 조회의 실제 구현에 대한 추상을 제공하여 조회를 최적화할 수 있으므로 프로덕션 시나리오에서 표준 Business Process Choreographer 조회 API 대신 복합 조회 테이블을 사용할 것을 권장합니다.

또한 런타임에 조회 테이블에 액세스하는 클라이언트를 다시 전개하지 않고 복합 조회 테이블을 변경할 수 있습니다.

다음 그림에서는 복합 조회 테이블의 컨텐츠에 대한 개요를 제공합니다.

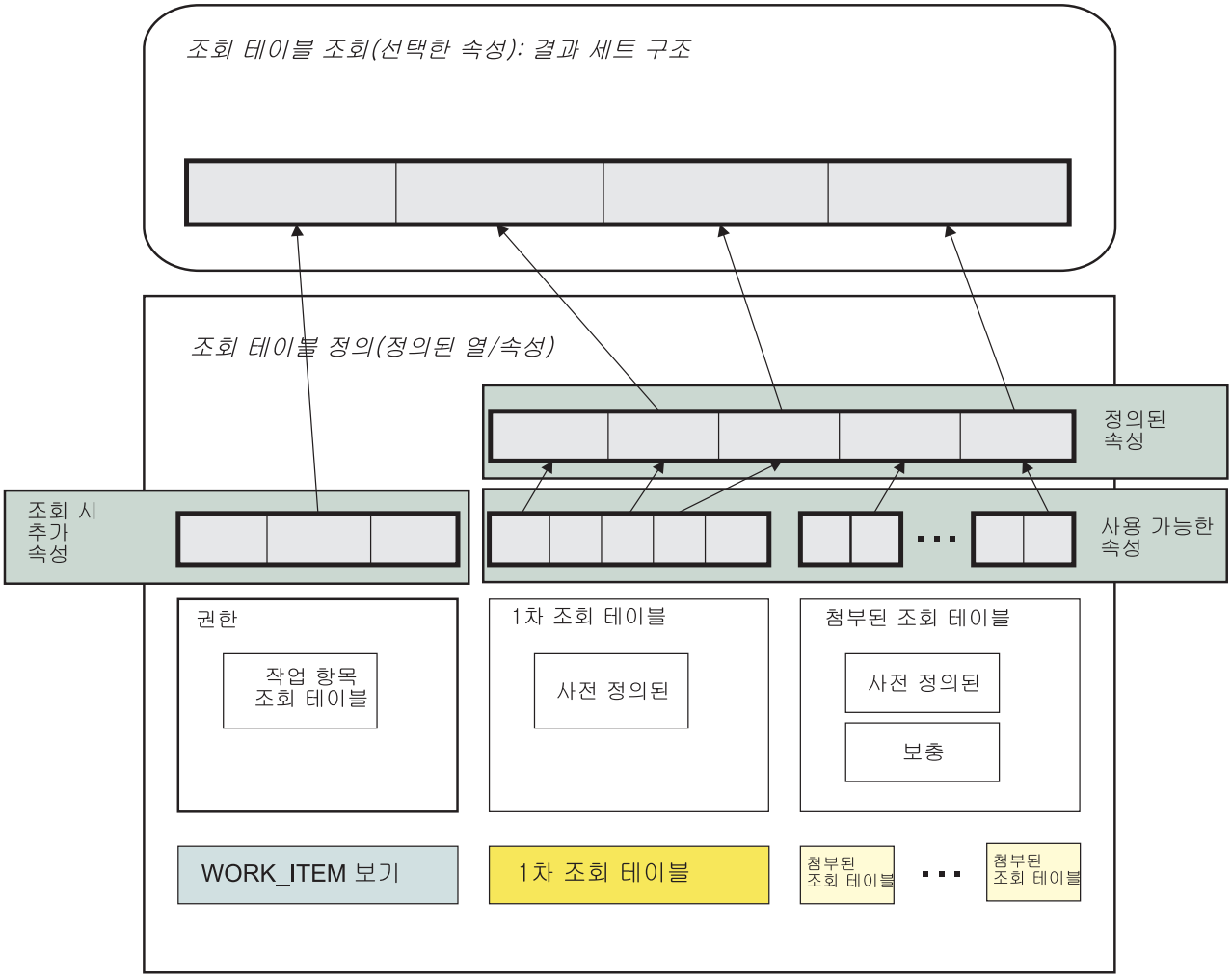


그림 73. 복합 조회 테이블 컨텐츠

복합 조회 테이블은 모두 하나의 1차 조회 테이블과 0개 이상의 첨부된 조회 테이블로 정의됩니다.

1차 조회 테이블:

- 복합 조회 테이블에 포함된 기본 정보를 구성합니다.
- 사전정의된 조회 테이블 중 하나여야 합니다.
- 1차 키로 복합 조회 테이블에서 각 오브젝트를 고유하게 식별합니다. 예를 들어, TASK 사전정의된 조회 테이블의 경우에는 ID TKIID 타스크입니다.
- 인스턴스 기반 권한이 사용되는 경우 WORK_ITEM 조회 테이블에 포함된 작업 항목을 사용하여 조회 테이블의 컨텐츠에 권한을 부여합니다.

- 복합 조회 테이블 조회 시 테이블의 행으로 리턴되는 오브젝트 목록을 판별합니다.

첨부된 조회 테이블:

- 시스템에 이미 전개된 사전정의된 조회 테이블 및 추가 조회 테이블이 될 수 있습니다.
- 1차 조회 테이블에서 제공하는 정보 이외의 정보를 추가로 제공하기 위해 사용할 수 있습니다. 예를 들어, TASK가 1차 조회 테이블인 경우 TASK_DESC 조회 테이블에 제공된 타스크에 대한 설명이 복합 조회 테이블의 컨텐츠에 추가될 수 있습니다.

일반적으로 1차 조회 테이블은 복합 조회 테이블의 용도에 따라 선택됩니다.

- 복합 조회 테이블에서 타스크 목록에 대해 설명하는 경우에는 TASK 조회 테이블이 1차 조회 테이블입니다.
- 복합 조회 테이블에서 프로세스 목록에 대해 설명하는 경우에는 PROCESS_INSTANCE 조회 테이블이 1차 조회 테이블입니다.
- ACTIVITY 1차 조회 테이블을 사용하여 활동 목록을 검색합니다.
- 휴먼 타스크 에스컬레이션 목록은 ESCALATION 1차 조회 테이블을 사용하여 검색합니다.

1차 조회 테이블과 첨부된 조회 테이블 간의 관계

첨부된 조회 테이블과 1차 조회 테이블에는 일대일 또는 일대영 관계가 있어야 합니다. 일대일 또는 일대영 관계가 위반되면 조회 실행 시 런타임 예외가 발생합니다.

1차 조회 테이블과 첨부된 조회 테이블은 첨부된 조회 테이블에 정의된 결합 속성을 사용하여 상관됩니다. 이 결합 속성은 Business Process Choreographer의 다양한 조회 테이블에 있는 데이터 간 관계에 대해 설명하기 때문에 사전정의된 조회 테이블을 위해 변경할 수 없습니다. 결합 속성은 일반적으로 일대일 또는 일대영 관계를 충분히 관리할 수 있습니다. 예를 들어, CONTAINMENT_CTX_ID 속성을 TASK 조회 테이블에 사용하여 PROCESS_INSTANCE 조회 테이블의 PIID 속성으로 식별되는 관련 프로세스 인스턴스 정보를 첨부합니다.

일대다 관계가 있는 경우, 조회 테이블을 정의할 때 조회 테이블 빌더에서 추가 기준(선택 기준)을 지정해야 합니다. 예를 들어, 선택 기준은 `"LOCALE='en_US'`가 될 수 있습니다. 타스크에는 단일 타스크에 대해 여러 로케일을 사용하여 식별되는 여러 설명이 있을 수 있습니다.

예제 1:

다음 그림에서는 첨부된 조회 테이블에 지정되는 선택 기준의 샘플 시각화를 제공합니다.

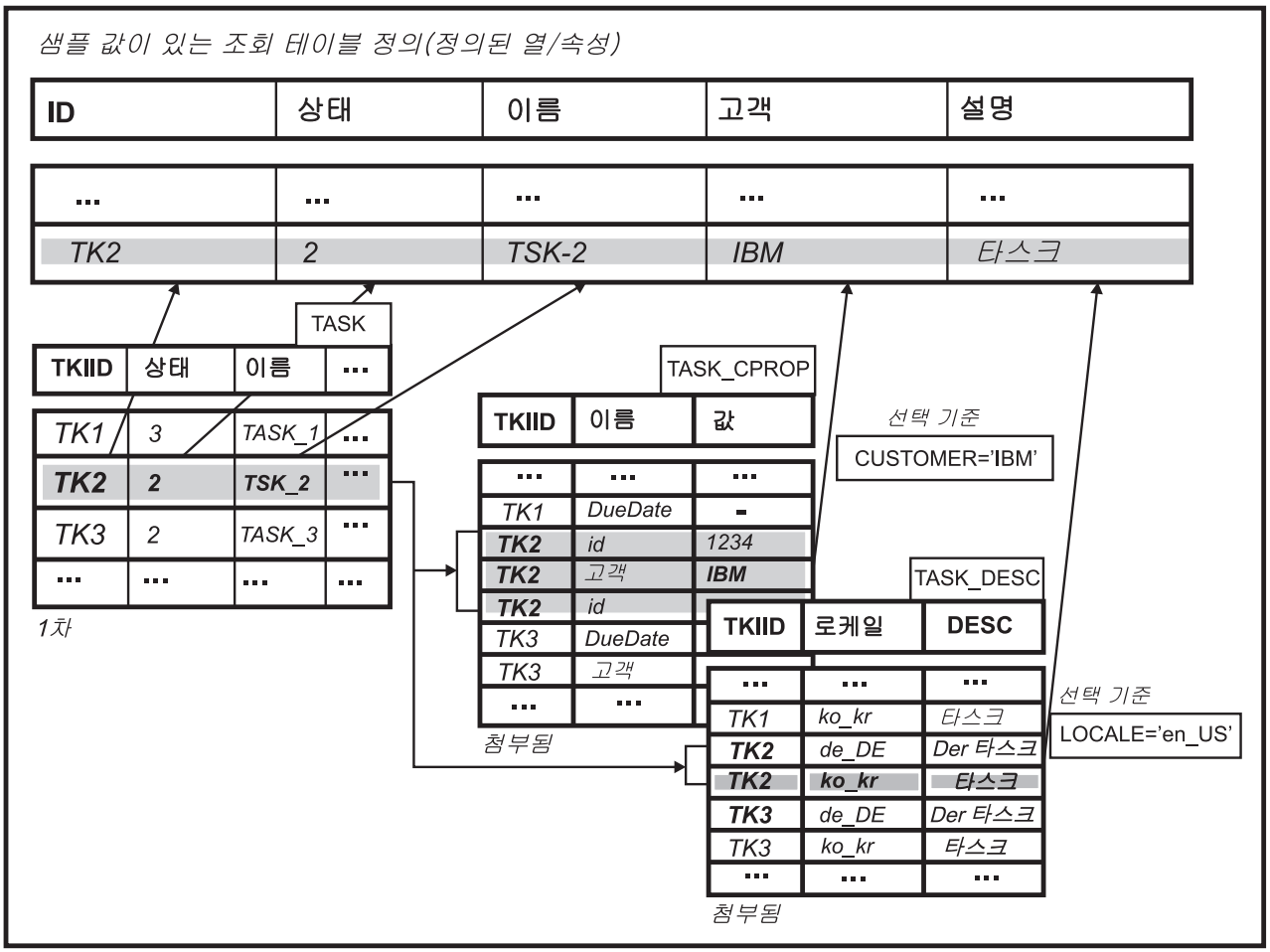


그림 74. 선택 기준이 사용된 복합 조회 테이블

복합 조회 테이블에는 ID, STATE, NAME, CUSTOMER 및 DESCRIPTION 속성이 포함되어 있습니다.

- ID, STATE 및 NAME은 TASK 1차 조회 테이블에서 제공합니다.
- CUSTOMER는 TASK의 사용자 정의 특성입니다. 사용자 정의 특성은 TASK_CPROP 조회 테이블에 저장됩니다. 특정 타스크의 경우 사용자 정의 특성은 이름으로 고유하게 식별됩니다. 이는 선택 기준 `"CUSTOMER='IBM'"`에 반영됩니다.
- DESCRIPTION은 TASK_DESC 조회 테이블에 저장되는, 타스크에 대한 설명입니다. 각 타스크 인스턴스에서 특정 타스크에 대한 타스크 설명은 로케일을 통해 식별됩니다. 이는 선택 기준 `"LOCALE='en_US'"`에 반영됩니다.

예제 2:

이 예제에서는 TASK를 1차 조회 테이블로 사용하고 TASK_DESC를 첨부된 조회 테이블로 사용하여 1차 조회 테이블과 첨부된 조회 테이블 간 관계에 중점을 둡니다. 복합 조회 테이블을 정의하는 경우에는 TASK_DESC 조회 테이블의 LOCALE 속성을 지정하여 1차 조회 테이블과 첨부된 조회 테이블 간에 일대일 또는 일대영 관계가 있

는지 확인해야 합니다. 이 표에서는 TASK_DESC 첨부된 조회 테이블의 유효한 선택 기준이 있는 복합 조회 테이블의 샘플 컨텐츠에 대해 설명합니다.

표 29. 복합 조회 테이블의 유효한 컨텐츠

TASK 1차 조회 테이블 정보	**TASK_DESC 첨부된 조회 테이블 정보**	
이름	로케일	설명
task_one	ko_kr	설명.
task_two	ko_kr	설명.
...	...	...

다음 표에서는 선택 기준이 잘못 설정되어 있는 경우 즉, 일대일 또는 일대영 관계를 위반한 경우 유효하지 않은 컨텐츠(**bold** 유형)를 가정하여 보여줍니다.

표 30. 복합 조회 테이블의 유효하지 않은 컨텐츠

TASK(1차 조회 테이블)의 정보	**TASK_DESC(첨부된 조회 테이블)의 정보**	
이름	로케일	설명
task_one	ko_kr	설명.
task_one	**de_DE**	**Das ist eine Beschreibung.**
...	...	...

특성

복합 조회 테이블에는 다음과 같은 특성이 있습니다.

표 31. 복합 조회 테이블의 특성

특성	설명
이름	조회 테이블 이름은 Business Process Choreographer 설치에서 고유해야 합니다. 조회가 실행되면 이 조회 테이블 이름을 사용하여 조회되는 조회 테이블을 식별합니다. 복합 조회 테이블의 경우 *prefix.name*으로 정의되는 이름을 사용하여 조회 테이블을 식별합니다. *prefix.name*의 최대 길이는 28자입니다. 접두부는 예약된 접두부 'IBM'과 달라야 합니다(예: 'COMPANY.TODO_TASK_LIST'). 테이블 이름의 끝에는 숫자를 사용하지 마십시오. 한 조회에서 단일 테이블이 여러 번 사용되는 경우, 테이블의 이름은 0 - 9 범위의 숫자를 사용하여 확장할 수 있습니다(예: CUSTOM_VIEW0, CUSTOM_VIEW1 등). 테이블 이름의 끝에 이미 숫자가 있는 경우에는 Business Process Choreographer가 이 숫자를 제거하므로 QueryUnknownTableException이 발생합니다.

표 31. 복합 조회 테이블의 특성 (계속)

특성	설명
속성	복합 조회 테이블의 속성은 조회에 사용할 수 있는 정보의 일부를 정의합니다. 대문자로 된 이름으로 속성을 정의합니다. 유형은 참조된 속성에서 상속되며 속성은 다음 중 하나입니다. • **부울**: 부울 값 • **10진수**: 부동 소수점 숫자 • **ID**: 오브젝트 ID(예: 조회 테이블 TASK의 TKIID) • **숫자**: 정수(short 또는 long) • **문자열**: 문자열 • **시간소인**: 시간소인 복합 조회 테이블의 속성은 1차 조회 테이블 또는 첨부된 조회 테이블의 속성에 대한 참조를 사용하여 정의됩니다. 복합 조회 테이블의 속성은 참조된 속성의 유형 및 상수를 상속합니다. 조회 테이블 정의의 파트인 속성 이외에 작업 항목 정보를 런타임 시 조회할 수 있습니다. 1차 조회 테이블에 TASK 또는 PROCESS_INSTANCE와 같은 인스턴스 데이터가 포함되어 있고 복합 조회 테이블에서 인스턴스 기반 권한이 사용되는 경우 조회가 가능합니다. 예를 들어, 사용자가 잠재적 소유자인 휴먼 타스크만 리턴하도록 조회를 정의할 수 있습니다.

표 31. 복합 조회 테이블의 특성 (계속)

특성	설명
권한	각 복합 조회 테이블은 조회 테이블에서 조회가 실행되는 경우 인스턴스 기반이나 역할 기반 권한이 사용되는지 또는 아무 권한도 사용되지 않는지 여부를 정의합니다. 인스턴스 기반 권한이 정의된 경우 조회를 수행하는 사용자의 작업 항목이 포함된 오브젝트만 리턴됩니다. 그러나 AdminAuthorizationOptions를 사용하여 임의의 사용자의 작업 항목이 있는지만 확인하도록 확인 범위를 줄일 수 있습니다. 비즈니스 플로우 관리자 EJB가 사용되는 경우에는 사용자에게 해당 조회에 대해 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우에는 TaskSystemAdministrator Java EE 역할이 있어야 합니다. 또한 AdminAuthorizationOptions를 조회 테이블 API로 전달해야 합니다. 역할 기반 권한이 정의된 경우, 비즈니스 플로우 관리자 EJB가 사용되는 경우에는 사용자에게 해당 조회에 대해 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우에는 TaskSystemAdministrator Java EE 역할이 있어야 합니다. 또한 AdminAuthorizationOptions를 조회 테이블 API로 전달해야 합니다. 권한이 정의되지 않은 경우에는 조회 테이블에 관련 오브젝트의 작업 항목이 있는지 확인하지 않고 조회가 실행됩니다. 인증된 모든 사용자가 조회 테이블의 컨텐츠를 볼 수 있습니다. 1차 조회 테이블에 인스턴스 데이터가 포함되어 있는 경우 인스턴스 기반 권한을 정의할 수 있고 1차 조회 테이블에 템플리트 데이터가 포함되어 있는 경우 역할 기반 권한을 정의할 수 있습니다. 사용되는 1차 조회 테이블 종류에 관계 없이 복합 조회 테이블에 대한 권한을 정의할 수 없습니다.

필터

필터는 복합 조회 테이블에 포함된 행 또는 오브젝트 수를 제한하는 데 사용됩니다.

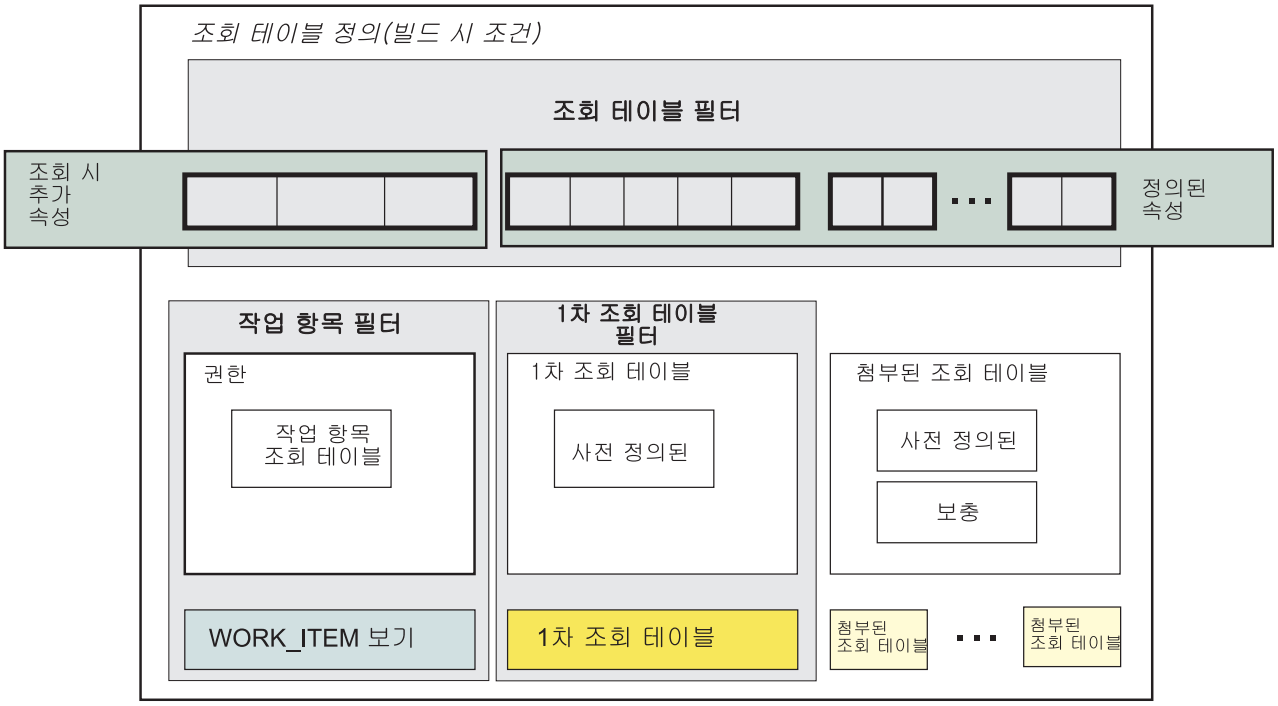


그림 75. 복합 조회 테이블의 필터

복합 조회 테이블의 필터는 다음 조회 테이블에서 개발 중에 정의할 수 있습니다.

- 1차 조회 테이블 필터로서의 1차 조회 테이블
- 1차 조회 테이블에 인스턴스 데이터가 포함되어 있는 경우 권한을 부여하는 내재적으로 사용 가능한 WORK_ITEM 조회 테이블. 이 필터는 권한 필터라고 하며 복합 조회 테이블이 인스턴스 기반 권한을 사용하도록 구성된 경우에만 사용할 수 있습니다.
- 조회 테이블 필터로서의 복합 조회 테이블

필터는 조회 테이블 개발 중에 정의됩니다. 예를 들어, TASK 1차 조회 테이블이 있는 복합 조회 테이블은 준비 상태("STATE=STATE_READY가 1차 조회 테이블 필터임)에 있는 타스크에 대해 필터링할 수 있습니다.

권한

1차 조회 테이블이 있는 복합 조회 테이블의 컨텐츠에 대한 액세스 권한은 1차 조회 테이블에 액세스하는 데 사용되는 권한과 비슷합니다. 차이점은 복합 조회 테이블을 더 제한적으로 구성할 수 있다는 점입니다.

- 인스턴스 기반 권한을 사용하도록 구성된 경우 복합 조회 테이블에 포함된 데이터에서 WORK_ITEM 조회 테이블에 작업 항목이 있는지 확인합니다. 이러한 확인은 1차 조회 테이블에 대해 수행됩니다. 복합 조회 테이블의 구성에 따라 모두, 개별, 그룹 및 상속된 작업 항목이 확인에 사용됩니다. 상속됨 작업 항목이 지정된 경우 참여하는 휴먼 타스크와 같이 구성된 대로 관련된 모든 사용자, 개인 또는 그룹 작업

항목이 있는 상위로서 프로세스 인스턴스가 포함된 오브젝트는 복합 조회 테이블에 포함됩니다. 일반적으로 상속됨 작업 항목은 관리자에게만 유용합니다.

- 템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블은 인스턴스 기반 권한을 사용하도록 설정해서는 안 됩니다. 역할 기반 권한이 사용되는 경우, 비즈니스 플로우 관리자 EJB가 사용되는 경우에는 BPESystemAdministrator Java EE 역할이 있는 사용자만 조회를 실행할 수 있으며 휴먼 타스크 관리자 EJB가 사용되는 경우에는 TaskSystemAdministrator Java EE 역할이 있는 사용자만 실행할 수 있습니다. 또한 AdminAuthorizationOptions 오브젝트가 사용되어야 합니다.

관련 개념

351 페이지의 『사전정의된 조회 테이블』
사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

354 페이지의 『추가 조회 테이블』
Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

조회 테이블 개발

Business Process Choreographer의 추가 및 복합 조회 테이블은 응용프로그램 개발 중에 조회 테이블 빌더를 사용하여 개발됩니다. 사전정의된 조회 테이블은 개발되거나 전개될 수 없습니다. Business Process Choreographer가 설치될 때 사용 가능하며 Business Process Choreographer 데이터베이스 스키마의 아티팩트에 대한 간단한 보기를 제공합니다.

조회 테이블 빌더는 Eclipse 플러그인으로 사용할 수 있으며 WebSphere Business Process Management SupportPacs 사이트에서 다운로드할 수 있습니다. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

조회 테이블은 응용프로그램의 개발 및 전개 방법에 영향을 미칩니다. 다음 단계에서는 조회 테이블을 사용하는 Business Process Choreographer 응용프로그램 설계 및 개발 시 관련된 역할에 대해 설명합니다.

표 32. 조회 테이블 개발 단계

단계	대상	설명
1. 분석	비즈니스 분석자, 클라이언트 개발자	클라이언트 응용프로그램에서 필요한 조회 테이블을 분석합니다. 응답할 질문은 다음과 같습니다. • 사용자에게 제공되는 타스크 또는 프로세스 목록 수는 몇 개입니까? 동일한 조회 테이블을 공유할 수 있는 타스크 또는 프로세스 목록이 있습니까? • 사용되는 권한 유형은 무엇입니까? 인스턴스 기반 권한, 역할 기반 권한과 없음 중 어느 것입니까? • 재사용할 수 있는 기타 조회 테이블이 시스템에 이미 정의되어 있습니까? • 조회 테이블에서 여러 언어로 컨텐츠를 제공해야 합니까? 여러 언어로 컨텐츠를 제공해야 하는 경우 첨부된 조회 테이블의 선택 기준은 LOCALE=$LOCALE이어야 합니다.
2. 조회 테이블 개발	클라이언트 개발자, 비즈니스 분석자	클라이언트 응용프로그램에서 사용되는 조회 테이블을 개발합니다. 조회 테이블 조회로 최적 성능이 달성되도록 조회 테이블의 정의를 지정합니다.
3. 조회 테이블 전개	관리자	조회 테이블을 사용하려면 먼저 조회 테이블을 런타임에 전개해야 합니다. 이 단계는 manageQueryTable.py wsadmin 명령을 사용하여 수행됩니다.
4. 조회 테이블 조회	클라이언트 개발자	조회 테이블에 대한 조회 실행은 조회 테이블 개발의 마지막 단계입니다. 클라이언트 개발자는 조회 테이블의 이름과 해당 속성을 알고 있어야 합니다.

다음은 조회 테이블을 조회하기 위해 조회 테이블 API를 사용하는 샘플 코드입니다. 예제 1 및 2는 간편성의 이유로 사전정의된 조회 테이블 TASK를 조회하기 위해 제공됩니다. 예제 3 및 4는 시스템에서 전개될 것으로 보이는 복합 조회 테이블을 조회합니다. 응용프로그램 개발에서는 사전 정의된 조회 테이블을 직접 조회하지 않고 복합 조회 테이블을 사용해야 합니다.

예제 1

```
// get the naming context and lookup the Business
// Flow Manager Enterprise JavaBeans home; note that the Business Flow
// Manager Enterprise JavaBeans home should be cached for performance
// reasons; also, it is assumed that there's an Enterprise JavaBeans
// reference to the local business flow manager Enterprise JavaBeans
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
 (LocalBusinessFlowManagerHome)
 ctx.lookup("java:comp/env/ejb/BFM");

// if the human task manager Enterprise JavaBeans is used, do:
```

```
// LocalHumanTaskManagerHome home =
//   (LocalHumanTaskManagerHome) ctx.lookup("java:comp/env/ejb/HTM");
// assuming that a EJB reference to the human task manager EJB
// has been defined

// create the business flow manager client-side stub
LocalBusinessFlowManager bfm = home.create();
// if the human task manager EJB is used, do:
// LocalHumanTaskManager htm = home.create();
// note that the human task manager Enterprise JavaBeans provides the
// same methods as the business flow manager Enterprise JavaBeans
// ************************************************
// ****************** example 1 ******************
// ************************************************

// execute a query against the TASK predefined query
// table; this relates to a simple My ToDo's task list
EntityResultSet ers = null;
ers = bfm.queryEntities("TASK", null, null, null);

// print the result to STDOUT
EntityInfo entityInfo = ers.getEntityInfo();
List attList = entityInfo.getAttributeInfo();
int attSize = attList.size();

Iterator iter = ers.getEntities().iterator();
while( iter.hasNext() ) {
 System.out.print("Entity: ");
 Entity entity = (Entity) iter.next();
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    entity.getAttributeValue(ai.getName()));
 }
 System.out.println();
}
```

예제 2

```
// ************************************************
// ****************** example 2 ******************
// ************************************************

// same example as example 1, but using the row-based
// query approach
RowResultSet rrs = null;
rrs = bfm.queryRows("TASK", null, null, null);

attList = rrs.getAttributeInfo();
attSize = attList.size();

// print the result to STDOUT
while (rrs.next()) {
 System.out.print("Row: ");
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    rrs.getAttributeValue(ai.getName()));
```

```
    }
    System.out.println();
  }
```

예제 3

```
  // ************************************************
  // ****************** example 3 *******************
  // ************************************************

  // execute a query against a composite query table
  // that has been deployed on the system before;
  // the name is assumed to be COMPANY.TASK_LIST
  ers = bfm.queryEntities(
    "COMPANY.TASK_LIST", null, null, null);
^
  //  print the result to STDOUT ...
```

예제 4

```
  // ************************************************
  // ****************** example 4 *******************
  // ************************************************

  // query against the same query table as in example 3,
  // but with customized options
  FilterOptions fo = new FilterOptions();

  // return only objects which are in state ready
  fo.setQueryCondition("STATE=STATE_READY");

  // sort by the id of the object
  fo.setSortAttributes("ID");

  // limit the number of entities to 50
  fo.setThreshold(50);

  // only get a sub-set of the defined attributes
  // on the query table
  fo.setSelectedAttributes("ID, STATE, DESCRIPTION");

  AuthorizationOptions ao = new AuthorizationOptions();

  // do not return objects that everybody is allowed
  // to see
  ao.setEverybodyUsed(Boolean.FALSE);

  ers = bfm.queryEntities(
    "COMPANY.TASK_LIST", fo, ao, null);

  //  print the result to STDOUT ...
```

관련 개념

385 페이지의 『조회 테이블 조회』
비즈니스 플로우 관리자 EJB 및 REST API에서 사용 가능한 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회가 실행됩니다.

『조회 테이블의 필터 및 선택 기준』
필터 및 선택 기준은 SQL WHERE절과 비슷한 구문을 사용하는 조회 테이블 빌더를 사용하여 조회 테이블 개발 중에 정의됩니다. 명확하게 정의된 이러한 필터 및 선택 기준을 사용하여 조회 테이블의 속성을 기반으로 하는 조건을 지정하십시오.

351 페이지의 『사전정의된 조회 테이블』
사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

354 페이지의 『추가 조회 테이블』
Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

356 페이지의 『복합 조회 테이블』
Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

조회 테이블의 필터 및 선택 기준

필터 및 선택 기준은 SQL WHERE절과 비슷한 구문을 사용하는 조회 테이블 빌더를 사용하여 조회 테이블 개발 중에 정의됩니다. 명확하게 정의된 이러한 필터 및 선택 기준을 사용하여 조회 테이블의 속성을 기반으로 하는 조건을 지정하십시오.

조회 테이블 빌더 설치에 대한 정보는 WebSphere Business Process Management SupportPacs 사이트를 참조하십시오. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

속성

표현식에 사용된 속성은 조회 테이블 속성을 참조합니다. 표현식의 위치에 따라 여러 속성을 사용할 수 있습니다. 클라이언트 개발자의 경우 조회 테이블 API에 전달된 조회 필터가 표현식을 사용할 수 있는 유일한 위치입니다. 복합 조회 테이블 개발자의 경우 기타 여러 위치에 표현식을 사용할 수 있습니다. 다음 표에서는 여러 위치에서 사용할 수 있는 속성에 대해 설명합니다.

표 33. 조회 테이블 표현식의 속성

테이블 유형	표현식	사용 가능한 속성
조회 테이블 API	조회 필터	• 조회 테이블에 정의된 모든 속성. • 인스턴스 기반 권한이 사용되는 경우에는 접두부가 'WI.'인 WORK_ITEM 조회 테이블에 정의된 모든 속성. 예제: • STATE=STATE_READY(조회 테이블에 STATE 속성이 포함되어 있는 경우 및 STATE_READY 상수가 이 속성에 정의된 경우) • STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER(조회 테이블에 STATE 속성이 포함되어 있으며 조회 테이블에서 인스턴스 기반 권한을 사용하는 경우)
복합 조회 테이블	조회 테이블 필터	
	1차 조회 테이블 필터	• 1차 조회 테이블에 대해 정의된 모든 속성. 예제: • STATE=STATE_READY(조회 테이블에 STATE 속성이 포함되어 있으며 STATE_READY 상수가 이 속성에 대해 정의된 경우)
	권한 필터	• 접두부가 'WI.'인 WORK_ITEM 사전정의된 조회 테이블에 정의된 모든 속성. 예제: • WI.REASON=REASON_POTENTIAL_OWNER
	선택 기준	• 관련 첨부된 조회 테이블에 정의된 모든 속성. 예제: • LOCALE='en_US'(TASK_DESC 조회 테이블과 같이 첨부된 조회 테이블에 LOCALE 속성이 포함되어 있는 경우)

다음 그림에는 표현식에 있는 필터 및 선택 기준의 다양한 위치가 표시되며 예제가 포함되어 있습니다.

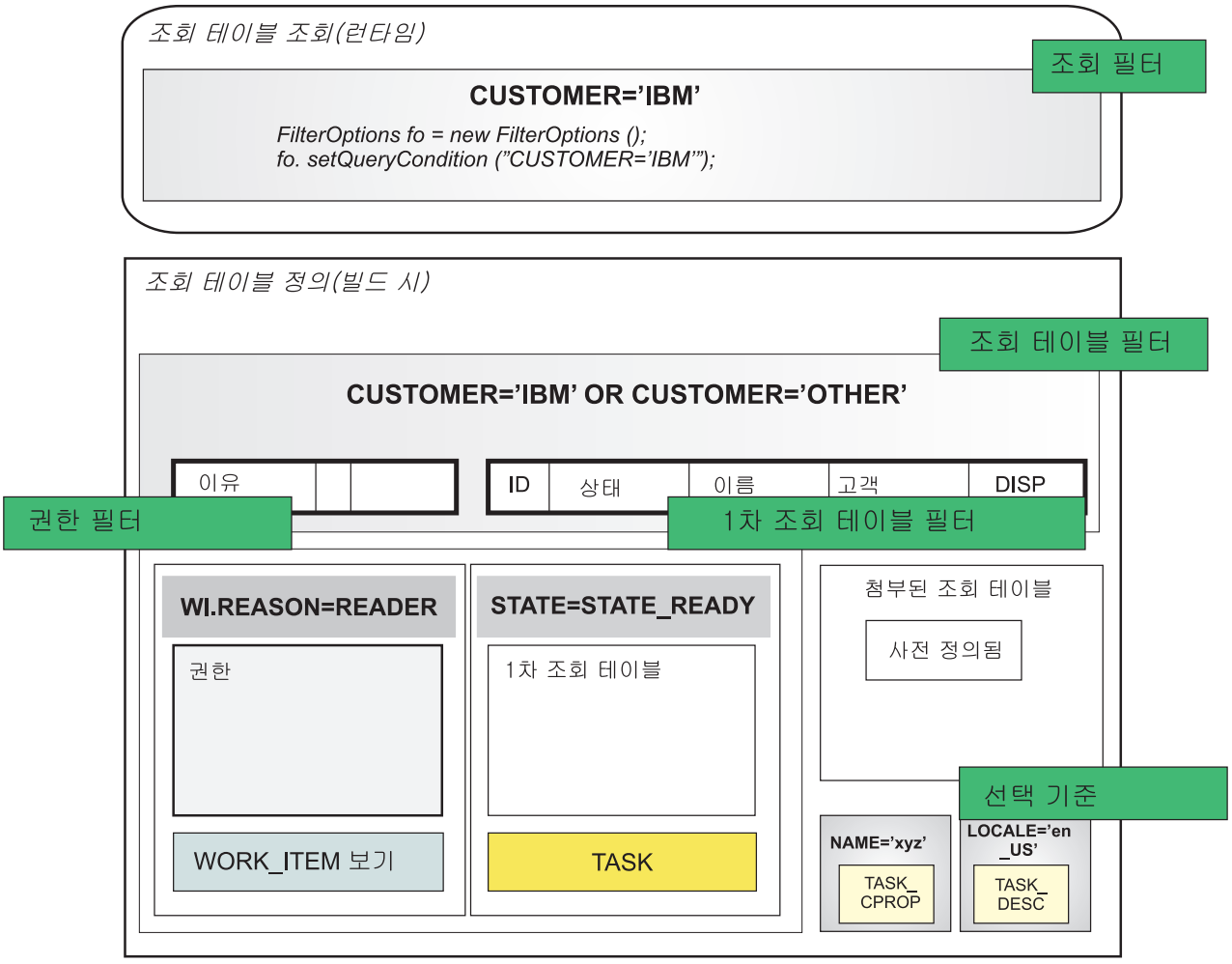


그림 76. 표현식의 필터 및 선택 기준

표현식

표현식의 구문은 다음과 같습니다.

```
expression  ::=  attribute binary_op value |
                 attribute unary_op |
                 attribute list_op list |
                 (expression) |
                 expression AND expression |
                 expression> OR expression
```

다음 규칙이 적용됩니다.

- AND가 OR보다 우선순위가 높습니다. 부속표현식은 AND 및 OR을 사용하여 연결됩니다.
- 대괄호를 사용하여 표현식을 그룹화할 수 있으며 대괄호는 쌍이 맞아야 합니다.

예제:

- STATE = STATE_READY

- NAME IS NOT NULL
- STATE IN (2, 5, STATE_FINISHED)
- ((PRIORITY=1) OR (WI.REASON=2)) AND (STATE=2)

표현식에 유효한 속성을 판별하는 특정 범위에서 표현식이 실행됩니다. 선택 기준(또는 조회 필터)은 조회가 실행되는 조회 테이블의 범위에서 실행됩니다.

다음 예제는 사전정의된 TASK 조회 테이블에서 실행되는 조회를 위한 것입니다.

```
'(STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER)
OR (WI.REASON=REASON_OWNER)'
```

2항 연산자

다음 2항 연산자를 사용할 수 있습니다.

binary_op ::= = | < | > | <> | <= | >= | LIKE | NOT LIKE

다음 규칙이 적용됩니다.

- 2항 연산자의 왼쪽 피연산자는 조회 테이블의 속성을 참조해야 합니다.
- 2항 연산자의 오른쪽 피연산자는 리터럴 값, 상수 값 또는 매개변수여야 합니다.
- LIKE 및 NOT LIKE 연산자는 STRING 속성 유형의 속성에 대해서만 유효합니다.
- 왼쪽 피연산자와 오른쪽 피연산자는 호환 가능한 속성 유형으로 되어 있어야 합니다.
- 사용자 매개변수는 왼쪽 속성의 속성 유형과 호환 가능해야 합니다.

예제:

- STATE > 2
- NAME LIKE 'start%'
- STATE <> PARAM(theState)

단항 연산자

다음 1진 연산자를 사용할 수 있습니다.

unary_op ::= IS NULL | IS NOT NULL

다음 규칙이 적용됩니다.

- 단항 연산자의 왼쪽 피연산자는 조회 테이블의 속성을 참조해야 합니다. 유효한 속성은 필터 또는 선택 기준의 위치에 따라 다릅니다.
- 널값이 있는지 모든 속성을 확인할 수 있습니다(예: CUSTOMER IS NOT NULL).

예제:

```
DESCRIPTION IS NOT NULL
```

목록 연산자

다음 목록 연산자를 사용할 수 있습니다.

list_op ::= IN | NOT IN

다음 규칙이 적용됩니다.

- List 연산자의 오른쪽은 사용자 매개변수로 바꿔서는 안 됩니다.
- 사용자 매개변수를 오른쪽 피연산자의 목록 내에서 사용할 수 있습니다.

예제:

```
STATE IN (STATE_READY, STATE_RUNNING, PARAM(st), 1)
```

목록은 다음과 같이 표시됩니다.

list ::= *value* [, *list*]

다음 규칙이 적용됩니다.

- List 연산자의 오른쪽은 사용자 매개변수로 바꿔서는 안 됩니다.
- 사용자 매개변수를 오른쪽 피연산자의 목록 내에서 사용할 수 있습니다.

예제:

- (2, 5, 8)
- (STATE_READY, STATE_CLAIMED)

값

표현식에서 값은 다음 중 하나입니다.

- **상수**: 사전정의된 조회 테이블의 속성에 대해 정의된 상수 값입니다. 예를 들어, STATE_READY가 TASK 조회 테이블의 STATE 속성에 대해 정의됩니다.
- **리터럴**: 하드코딩된 값입니다.
- **매개변수**: 조회가 실행될 때 특정 값으로 매개변수가 바뀝니다.

상수는 사전정의된 조회 테이블의 일부 속성에 사용할 수 있습니다. 사전 정의된 조회 테이블의 속성에 사용할 수 있는 상수에 대한 정보는 사전 정의된 보기에 대한 정보를 참조하십시오. 정수 값을 정의하는 상수만 조회 테이블과 함께 표시됩니다. 또한 상수 대신 관련 리터럴 값 또는 매개변수를 사용할 수 있습니다.

예제:

- 필터에서 TASK 조회 테이블의 STATE 속성에 STATE_READY를 사용하여 타스크가 준비 상태에 있는지 확인할 수 있습니다.

- 필터에서 WORK_ITEM 조회 테이블의 REASON 속성에 REASON_POTENTIAL_OWNER를 사용하여 조회 테이블에 대해 조회를 실행하는 사용자가 잠재적 소유자인지 확인할 수 있습니다.
- TASK 조회 테이블에서 조회가 실행되는 경우 조회 필터 STATE=STATE_READY는 STATE=2와 동일합니다.

리터럴은 표현식에도 사용할 수 있습니다. 특수 구문은 시간소인 및 ID에 사용해야 합니다.

예제:

- `STATE=1`
- `NAME='theName'`
- `CREATED > TS ('2008-11-26 T12:00:00')`
- `TKTID=ID('_TKT:801a011e.9d57c52.ab886df6.1fcc0000')`

표현식의 **매개변수**는 복합 조회 테이블의 동적 특성을 허용합니다.다음과 같은 사용자 매개변수 및 시스템 매개변수가 있습니다.

- 사용자 매개변수는 PARAM(*name*)을 사용하여 지정됩니다. 이러한 매개변수는 조회 실행 시 제공되어야 합니다. 이 매개변수는 com.ibm.bpe.api.Parameter 클래스의 인스턴스로 조회 테이블 API에 전달됩니다.
- 시스템 매개변수는 조회 실행 시 지정되지 않고 조회 테이블 런타임 시 제공되는 매개변수입니다. 시스템 매개변수 $USER 및 $LOCALE을 사용할 수 있습니다.
  - $USER는 문자열로, 조회를 실행한 사용자 값이 포함되어 있습니다.
  - $LOCALE은 문자열로, 조회를 실행할 때 사용된 로케일 값이 포함되어 있습니다. $LOCALE의 값 예제는 'en_US'입니다.

특정 로케일에서 선택하는 첨부된 조회 테이블의 선택 기준에 매개변수를 지정할 수 있습니다. 예를 들어, 1차 조회 테이블이 복합 조회 테이블의 TASK인 경우 첨부된 조회 테이블은 TASK_DESC입니다. 매개변수의 예제는 다음과 같습니다.

- `STATE=PARAM(theState)`
- `LOCALE=$LOCALE`
- `OWNER=$USER`

관련 개념

364 페이지의『조회 테이블 개발』

Business Process Choreographer의 추가 및 복합 조회 테이블은 응용프로그램 개발 중에 조회 테이블 빌더를 사용하여 개발됩니다. 사전정의된 조회 테이블은 개발되거나 전개될 수 없습니다. Business Process Choreographer가 설치될 때 사용 가능하며 Business Process Choreographer 데이터베이스 스키마의 아티팩트에 대한 간단한 보기를 제공합니다.

385 페이지의『조회 테이블 조회』

비즈니스 플로우 관리자 EJB 및 REST API에서 사용 가능한 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회가 실행됩니다.

관련 태스크

406 페이지의『Business Process Choreographer Explorer에 대한 조회 테이블 작성』

EJB query API 대신 조회 테이블을 사용하면 Business Process Choreographer Explorer의 성능이 향상됩니다. 조회 테이블을 작성하려면 조회 테이블 빌더를 사용하십시오.

조회 테이블에 대한 권한

조회 테이블에서 조회를 실행할 때 인스턴스 기반 권한 또는 역할 기반 권한을 사용하거나 아무 권한도 사용하지 않을 수 있습니다.

권한 유형은 조회 테이블에 정의됩니다.

- 인스턴스 기반 권한은 작업 항목을 사용하여 조회 테이블의 오브젝트에 권한을 부여함을 나타냅니다. 이 작업은 적절한 작업 항목이 존재하는지 확인하는 과정을 통해 수행됩니다.
- 역할 기반 권한은 Java EE 역할을 기반으로 합니다. 이는 조회 테이블의 컨텐츠를 보려면 API 조회 메소드를 사용하는 호출자에게 비즈니스 플로우 관리자 EJB가 사용되는 경우 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator Java EE 역할이 있어야 함을 나타냅니다. 이 권한은 템플리트 데이터가 있는 사전 정의된 조회 테이블 및 템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블에서만 사용 가능합니다. 해당 조회 테이블에 있는 오브젝트에는 관련 작업 항목이 없습니다.
- 권한이 지정되지 않은 경우에는 필터가 적용된 이후에, 인증된 모든 사용자가 조회 테이블의 모든 컨텐츠를 볼 수 있습니다.

다음 표에는 사전 정의된 조회 테이블의 권한 유형과 복합 및 보충 조회 테이블에서 구성할 수 있는 권한 유형에 대해 간략히 설명되어 있습니다.

표 34. 조회 테이블의 권한 유형

조회 테이블	인스턴스 기반 권한	역할 기반 권한	권한 없음
사전 정의됨	인스턴스 데이터가 포함된 사전 정의된 조회 테이블에 필요합니다.	템플리트 데이터가 포함된 사전 정의된 조회 테이블에 필요합니다.	해당되지 않음
복합	설정을 해제할 수 있습니다. 설정 해제는 권한이 사용되지 않으며 보안 제한이 대체됨을 의미합니다. 즉, 인증된 사용자는 모두 해당 오브젝트에 대한 권한이 있는지에 관계없이 조회 테이블을 사용하여 데이터를 검색할 수 있습니다. 템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블은 인스턴스 기반 권한을 사용하도록 설정해서는 안 됩니다.	예를 들어, 템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블의 경우 설정 해제할 수 있습니다. 이는 권한이 사용되지 않으며 보안 제한이 대체됨을 의미합니다. 즉, 인증된 사용자는 모두 해당 오브젝트에 대한 권한이 있는지에 관계없이 조회 테이블을 사용하여 데이터를 검색할 수 있습니다. 인스턴스 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블은 역할 기반 권한을 사용하지 않도록 설정되어야 합니다.	필터가 적용된 이후에 인증된 모든 사용자가 조회 테이블의 모든 컨텐츠를 볼 수 있습니다.
추가	보충 조회 테이블은 Business Process Choreographer를 통해 관리되지 않기 때문에 인스턴스 기반 권한을 사용하도록 설정해서는 안 되며, 따라서 보충 조회 테이블의 컨텐츠에 적합한 권한 정보가 없습니다.	보충 조회 테이블은 역할 기반 권한을 사용하지 않도록 설정되어야 합니다.	필터가 적용된 이후에 인증된 모든 사용자가 조회 테이블의 모든 컨텐츠를 볼 수 있습니다.

다음 그림은 조회 테이블의 유형에 따라 권한 유형에서 사용 가능한 옵션에 대해 간략하게 설명합니다. 또한 이 그림에서는 여러 동작과 조회 테이블 API 및 그 권한 옵션에 대해 간략하게 설명합니다.

그림 77. 조회 테이블의 인스턴스 기반 권한

*) onBehalfUser가 설정된 경우 (A)가 적용됨

작업 항목을 사용하는 조회 결과의 오브젝트에 대한 인스턴스 기반 권한은 조회 테이블 API에 전달되는 권한 매개변수와 조회 테이블의 인스턴스 기반 권한 플래그의 설정에 따라 다릅니다.

- (A) AuthorizationOptions 오브젝트를 사용한 복합 또는 사전정의된 조회 테이블에 대한 조회에서는 이 특정 사용자의 관련 작업 항목과 상관되는 엔티티를 리턴합니다. AdminAuthorizationOptions 오브젝트가 사용되고 onBehalfUser가 설정된 경우에도

마찬가지입니다. 사용자에게 타스크 또는 프로세스 목록을 제공하는 표준 클라이언트에서는 일반적으로 조회 테이블과 조회 테이블 API 매개변수의 이러한 조합을 사용합니다.

- (B) 조회 테이블의 전체 컨텐츠는 조회 테이블의 인스턴스 기반 권한으로 구성된 것과 같이 관련 작업 항목이 있는 엔티티로 구성됩니다. 인스턴스 기반 권한에서는 네 가지 유형의 작업 항목(모든 사용자, 개인, 그룹 및 상속됨)을 고려합니다. API 조회 메소드를 사용하는 호출자는 비즈니스 플로우 관리자 EJB가 사용되는 경우 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator Java EE 역할이 있어야 합니다. 조회 테이블과 조회 테이블 API 매개변수의 이러한 조합은 사용 가능한 타스크 또는 프로세스의 전체 목록이 표시되거나 검색되어야 하는 관리 시나리오에서 사용하기 위한 것입니다.
- (C) AdminAuthorizationOptions 또는 AuthorizationOptions가 조회 테이블 API로 전달되는 경우 인스턴스 기반 또는 역할 기반 권한을 사용하지 않는 조회 테이블에서 실행된 조회는 동일한 결과를 리턴합니다. 이는 보충 및 복합 조회 테이블에서 사용 가능합니다. 작업 항목 또는 Java EE 역할을 검사하지 않으므로 인증된 모든 사용자가 전체 컨텐츠를 볼 수 있습니다. 클라이언트가 Business Process Choreographer에서 제공하는 인스턴스 기반 또는 역할 기반 권한 제한조건을 적용하여 오브젝트 가시성을 제한하지 않으려면 조회 테이블 정의 개발 시 권한 검사 설정을 해제할 수 있습니다. 그러나 청구 및 완료 사용 시에는 사용자에게 관련 작업 항목이 있어야 합니다.
- (D) 역할 기반 권한을 사용하는 경우에만 역할 기반 권한이 구성되어 있는 복합 조회 테이블 또는 사전 정의된 조회 테이블의 템플리트 데이터에 액세스할 수 있습니다. 이를 위해 API 조회 메소드를 사용하는 호출자는 비즈니스 플로우 관리자 EJB가 사용되는 경우 BPESystemAdministrator Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator Java EE 역할이 있어야 합니다. 조회 API 대신 조회 테이블 API를 사용하여 템플리트 정보에 액세스할 수 있습니다.

작업 항목 및 인스턴스 기반 권한

Business Process Choreographer에서 제공하는 인스턴스 기반 권한은 작업 항목을 기반으로 합니다. 각 작업 항목은 누가 어떤 오브젝트에 대해 어떤 권한을 갖고 있는지 설명합니다. 인스턴스 기반 권한이 사용되는 경우에는 WORK_ITEM 조회 테이블을 사용하여 이 정보에 액세스할 수 있습니다.

이 표에서는 조회 테이블에 대해 조회가 샐행될 때 인스턴스 기반 권한이 사용되는 경우 고려되는 서로 다른 유형의 작업 항목에 대해 설명합니다.

표 35. 작업 항목 유형

작업 항목 유형	설명
모든 사용자	모든 사용자가 특정 오브젝트(예: 타스크 또는 프로세스 인스턴스)에 액세스할 수 있습니다. 이 경우에는 관련 작업 항목의 EVERYBODY 속성이 TRUE로 설정됩니다.
개인	특정 사용자를 위해 작성되는 작업 항목입니다. 관련 작업 항목의 OWNER_ID 속성은 특정 사용자로 설정됩니다. 타스크와 같은 하나의 오브젝트에 대해 OWNER_ID 속성이 다른 여러 작업 항목이 있을 수 있습니다.
그룹	특정 그룹의 사용자를 위해 작성되는 작업 항목입니다. 관련 작업 항목의 GROUP_NAME 속성은 특정 그룹으로 설정됩니다.
상속됨	프로세스 인스턴스의 독자 및 관리자도 에스컬레이션을 포함하여 이러한 프로세스 인스턴스에 속하는 휴먼 타스크에 대한 액세스를 상속할 수 있습니다. 런타임 시 복합 SQL 결합으로 타스크 조회의 상속됨 작업 항목에 대한 검사가 수행되며 이 검사는 성능에 영향을 미칩니다.

작업 항목은 여러 상황에서 Business Process Choreographer에 의해 작성됩니다. 예를 들어, 타스크 작성 시 관련 사용자 지정 기준이 지정된 경우 여러 역할(예: 독자 및 잠재적 소유자)에 대해 작업 항목이 작성됩니다.

다음 표에서는 조회 테이블에 대해 조회가 실행될 때 인스턴스 기반 권한이 사용되는 경우 정의된 사용자 지정 기준에 따라 작성되는 작업 항목의 유형에 대해 설명합니다. 상속됨 작업 항목은 프로세스 응용프로그램 개발 중에 명시적으로 모델링되지 않은 관계를 반영하므로 표에 나타나지 않습니다.

표 36. 작업 항목 및 사용자 지정 기준

작업 항목 유형	관련 사용자 지정 기준
모든 사용자	모든 사용자
개인	*Nobody*, *Everybody* 및 *Group* verb를 제외한 모든 사용자 지정 기준
그룹	그룹

복합 조회 테이블의 권한 필터

복합 조회 테이블에서는 인스턴스 기반 권한이 사용되는 경우 권한 필터를 지정할 수 있습니다. 이 필터는 작업 항목의 특정 속성에 따라 권한에 사용되는 작업 항목을 제한합니다. 예를 들어, TASK 1차 조회 테이블이 포함된 복합 조회 테이블에서 권한 필터 `"WI.REASON=REASON_POTENTIAL_OWNER`는 사용자가 조회를 실행할 때 리턴되는 타스크를 제한합니다. 결과에는 해당 사용자가 수행할 작업을 표시하는 타스크만 포함되어 있으며 결과는 사용자가 청구할 권한이 있는 타스크로 제한됩니다. 이 필터는 조회 테이블 필터 또는 조회 필터로 지정할 수도 있지만 조회 성능 향상을 위해 이러한 필터를 권한 필터로 지정하는 것이 좋습니다.

관련 개념

351 페이지의 『사전정의된 조회 테이블』
사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

354 페이지의 『추가 조회 테이블』
Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

356 페이지의 『복합 조회 테이블』
Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

391 페이지의 『조회 테이블 API의 권한 옵션』
Business Process Choreographer에서 조회 테이블에 대해 조회를 실행하는 경우 권한 옵션을 입력 매개변수로 조회 테이블 API의 메소드에 전달할 수 있습니다.

관련 태스크

406 페이지의 『Business Process Choreographer Explorer에 대한 조회 테이블 작성』
EJB query API 대신 조회 테이블을 사용하면 Business Process Choreographer Explorer의 성능이 향상됩니다. 조회 테이블을 작성하려면 조회 테이블 빌더를 사용하십시오.

조회 테이블의 속성 유형

조회 테이블이 정의되는 경우, 리터럴 값이 조회에 사용되는 경우 및 조회 결과 값에 액세스하는 경우 Business Process Choreographer에 속성 유형이 필요합니다. 각 속성 유형에 규칙 및 맵핑을 사용할 수 있습니다.

Java 프로그래밍 언어 및 데이터베이스에서 사용 가능한 유형의 서브세트를 사용하여 조회 테이블의 속성 유형을 정의합니다. 속성 유형은 구체적인 Java 유형 또는 데이터베이스 유형의 추상입니다. 추가 조회 테이블의 경우에는 유효한 데이터베이스 유형 대 속성 유형 맵핑을 사용해야 합니다.

다음 표에서는 속성 유형에 대해 설명합니다.

표 37. 속성 유형

속성 유형	설명
ID	휴먼 타스크(TKIID), 프로세스 인스턴스(PIID) 또는 기타 오브젝트를 식별하는 데 사용되는 ID입니다. 예를 들어, ID는 지정된 TKIID로 식별되는 특정 휴먼 타스크를 완료하거나 청구하는 데 사용됩니다.
STRING	타스크 설명 또는 조회 특성은 문자열로 표시될 수 있습니다.
NUMBER	타스크의 우선순위와 같은 속성에 숫자가 사용됩니다.
TIMESTAMP	시간소인은 휴먼 타스크가 작성된 시간이나 프로세스 인스턴스가 완료된 시간과 같은 특정 시점을 표시합니다.
DECIMAL	10진수를 조회 특성의 유형으로 사용할 수 있습니다(예: XSD 유형 실수의 변수로 조회 특성 정의 시).
BOOLEAN	부울의 값은 true와 false 중 하나가 될 수 있습니다. 예를 들어, 휴먼 타스크에서는 단일 사용자가 이 타스크의 잠재적 소유자로 존재하는 경우 타스크가 자동으로 청구되는지를 식별하는 속성(autoClaim)을 제공합니다.

데이터베이스 유형 대 속성 유형 맵핑:

Business Process Choreographer에서 조회 테이블을 정의하는 경우, 조회 테이블에서 조회를 실행하는 경우 및 조회 결과의 값에 액세스하려는 경우 속성 유형을 사용하십시오.

다음 표에서는 데이터베이스 유형 및 속성 유형에 대한 해당 유형의 맵핑에 대해 설명합니다.

표 38. 데이터베이스 유형 대 속성 유형 맵핑

데이터베이스 유형	속성 유형
16바이트의 2진 유형. Business Process Choreographer 테이블의 TASK에 대한 TKIID와 같은 ID에 사용된 유형입니다.	ID
문자 기반 유형. 길이는 조회 테이블의 속성에서 참조하는 데이터베이스 테이블에 있는 열에 따라 다릅니다.	STRING
정수 데이터베이스 유형(예: integer, short 또는 long).	NUMBER
시간소인 데이터베이스 유형.	TIMESTAMP
10진수 유형(예: float 또는 double).	DECIMAL
부울 값으로 변환 가능한 유형(예: 숫자). 1은 *true*로 해석되고 다른 숫자는 모두 *false*로 해석됩니다.	BOOLEAN

추가 조회 테이블에서는 일반적으로 기존 데이터베이스 테이블 및 보기를 참조하므로 테이블 또는 보기를 작성할 필요가 없습니다.

예제

Business Process Choreographer에서 추가 조회 테이블로 표시되는 DB2 환경의 테이블(CUSTOM.ADDITIONAL_INFO)이 있다고 간주하십시오. 다음 SQL문에서는 데이터베이스 테이블을 작성합니다.

```
CREATE TABLE CUSTOM.ADDITIONAL_INFO
(
  PIID       CHAR(16) FOR BIT DATA,
  INFO       VARCHAR(220),
  COUNT      INTEGER
);
```

다음과 같은 데이터베이스 열 유형 대 조회 테이블 속성 유형 맵핑이 CUSTOM.ADDITIONAL_INFO 테이블의 추가 조회 테이블에 사용됩니다.

표 39. 데이터베이스 유형 대 속성 유형 맵핑 예제

데이터베이스 열 및 유형	조회 테이블 속성 및 유형
PIID CHAR(16) FOR BIT DATA	PIID(ID)
INFO VARCHAR(220)	INFO(STRING)
COUNT INTEGER	COUNT(NUMBER)

속성 유형 대 리터럴 표시 맵핑:

Business Process Choreographer에서 조회 테이블이 정의되는 경우, 조회 테이블에서 조회가 실행되는 경우 및 조회 결과의 값에 액세스하는 경우 속성 유형이 사용됩니다. 이 주제에서 속성 유형 대 리터럴 표시 맵핑에 대한 정보를 확인하십시오.

리터럴 값은 필터 및 선택 기준을 정의하는 표현식(예: 복합 조회 테이블의 필터 및 조회 테이블 API에 전달되는 필터)에 사용할 수 있습니다.

다음 표에서는 속성 유형과 리터럴 값에 대한 속성 유형의 맵핑에 대해 설명합니다. 플레이스홀더는 *기울임꼴*로 표시됩니다. 조회 테이블 API에 전달할 수 있는 ID 및 TIMESTAMP 속성 유형에서는 조회 API에서도 사용할 수 있는 특수 구문을 사용하는 점에 유의하십시오.

표 40. 속성 유형 대 리터럴 값 맵핑

속성 유형	구문 및 표현식에서 리터럴 값으로 사용하는 방법
ID	`ID ('string representation of an ID' )`
	클라이언트 응용프로그램 개발 시 ID는 com.ibm.bpe.api.OID 인터페이스의 인스턴스나 문자열로 표시됩니다. 문자열 표시는 toString 메소드를 사용하여 com.ibm.bpe.api.OID 인터페이스의 인스턴스에서 얻을 수 있습니다. 문자열은 작은따옴표로 묶어야 합니다.
STRING	`'the string'`
	문자열은 따옴표로 묶어야 합니다.
NUMBER	`number`
	따옴표가 없는 텍스트로서의 숫자. 사전 정의된 조회 테이블의 일부 숫자 속성에 상수를 정의하고 이를 사용할 수 있습니다.

표 40. 속성 유형 대 리터럴 값 맵핑 (계속)

속성 유형	구문 및 표현식에서 리터럴 값으로 사용하는 방법
TIMESTAMP	`TS ('YYYY-MM-DDThh:mm:ss')`
	시간소인은 다음과 같이 지정해야 합니다. • `YYYY`는 4자리 연도임 • `MM`은 2자리 월임 • `DD`는 2자리 일임 • `hh`는 2자리 시임(24시간) • `mm`은 2자리 분임 • `ss`는 2자리 초임. 시간소인은 사용자 시간대에 정의된 대로 해석됩니다.
DECIMAL	`number.fraction`
	따옴표가 없는 텍스트로서의 10진수 숫자. `.fraction` 부분은 선택적입니다.
BOOLEAN	`true, false`
	텍스트로서의 부울 값.

예제

- `filterOptions.setQueryCondition(STATE=2);`
- `filterOptions.setQueryCondition(STATE=STATE_READY);`
- `a selection criterion on an attached query table TASK_DESC: "LOCALE='en_US'`
- `filterOptions.setQueryCondition(PTID=ID ('_PT:8001011e.1dee8e51.247d6df6.29a60000'));`

속성 유형 대 매개변수 맵핑:

Business Process Choreographer에서 조회 테이블을 정의하는 경우, 조회 테이블에서 조회를 실행하는 경우 및 조회 결과의 값에 액세스하려는 경우 속성 유형을 사용하십시오.

다음 표에서는 필터 및 선택 기준을 정의하는 표현식(예: 복합 조회 테이블의 필터 및 조회 테이블 API에 전달된 필터)에 사용할 수 있는 속성 유형 및 매개변수 값에 대한 속성 유형의 맵핑에 대해 설명합니다.

표 41. 속성 유형 대 사용자 매개변수 값 맵핑

속성 유형	표현식에서 매개변수 값으로 사용하는 방법
ID	PARAM*(name)* 클라이언트 응용프로그램 개발 시 ID는 com.ibm.bpe.api.OID 인터페이스의 인스턴스나 문자열로 표시됩니다. 매개변수로서는 두 표시 모두 유효합니다. 유효한 OID를 반영하는 바이트 배열도 사용할 수 있습니다(바이트).
STRING	PARAM*(name)* toString 메소드에 의해 런타임 시 조회 테이블 API에 전달되는 오브젝트의 문자열 표시입니다.
NUMBER	PARAM*(name)* 숫자의 java.lang.Long, java.lang.Integer, java.lang.Short 또는 java.lang.String 표시가 조회 테이블 API에 전달됩니다. 사전 정의된 조회 테이블의 일부 속성에 정의된 상수 이름도 전달할 수 있습니다.
TIMESTAMP	PARAM*(name)* 다음 표시가 유효합니다. • 시간소인의 java.lang.String 표시 • com.ibm.bpe.api.UTCDate의 인스턴스 • java.util.Calendar의 인스턴스
DECIMAL	PARAM*(name)* 조회 테이블 API에 전달되는 10진수의 java.lang.Long, java.lang.Integer, java.lang.Short, java.lang.Double, java.lang.Float 또는 java.lang.String 표시입니다.
BOOLEAN	PARAM*(name)* 유효값은 다음과 같습니다. • 부울의 java.lang.String 표시 • 적절한 값을 가진 java.lang.Short, java.lang.Integer, java.lang.Long(0(false인 경우) 또는 1(true인 경우)) • java.lang.Boolean 오브젝트

예제

```
...
// this example shows a query against a composite query table
// COMP.TASKS with a parameter "customer"
java.util.List params = new java.util.ArrayList();

list.add(new com.ibm.bpe.api.Parameter("customer", "IBM");
// the business flow manager Enterprise JavaBeans or the
// human task manager Enterprise JavaBeans can be used to access query tables
service.bfm.queryEntities("COMP.TASKS", null, null, params);
...
```

속성 유형 대 Java 오브젝트 유형 맵핑:

Business Process Choreographer에서 조회 테이블이 정의되는 경우, 조회 테이블에서 조회가 실행되는 경우 및 조회 결과의 값에 액세스하는 경우 속성 유형이 사용됩니다. 이 주제에서 속성 유형 대 Java 오브젝트 유형 맵핑에 대한 정보를 확인하십시오.

다음 표에서는 조회 결과 세트에 있는 속성 유형과 Java 오브젝트 유형에 대한 속성 유형의 맵핑에 대해 설명합니다.

표 42. 속성 유형 대 Java 오브젝트 유형 맵핑

속성 유형	관련 Java 오브젝트 유형
ID	com.ibm.bpe.api.OID
STRING	java.lang.String
NUMBER	java.lang.Long
TIMESTAMP	java.util.Calendar
DECIMAL	java.lang.Double
BOOLEAN	java.lang.Boolean

예제

```
...
// the following example shows a query against a composite query table
// COMP.TA; attribute "STATE" is of attribute type NUMBER
...
// run the query
// the business flow manager Enterprise JavaBeans or the
// human task manager Enterprise JavaBeans can be used to access query tables
EntityResultSet rs = bfm.queryEntities("COMP.TA",null,null,params);

// get the entities and iterate over it
List entities = rs.getEntities();
for (int i = 0 ; i < entities.size(); i++) {

      // work on a particular entity
      Entity en = (Entity) entities.get(i);

      // note that the following code could be written
      // more generalized using the attribute info objects
      // contained in ei.getAttributeInfo()

      // get attribute STATE
      Long state = (Long) en.getAttributeValue("STATE");
      ...
}
...
```

속성 유형 호환성:

Business Process Choreographer에서 조회 테이블을 정의하는 경우, 조회 테이블에서 조회를 실행하는 경우 및 조회 결과의 값에 액세스하려는 경우 속성 유형을 사용하십시오.

다음 표는 조회 테이블에서 필터 및 선택 기준을 정의하는 데 사용할 수 있는 속성 유형 및 호환 가능한 속성 유형을 나타냅니다. 호환 가능한 속성 유형은 **X**로 표시됩니다.

표 43. 속성 유형 호환성

속성 유형	ID	STRING	NUMBER	TIMESTAMP	DECIMAL	BOOLEAN
ID	X					
STRING		X				
NUMBER			X		X	
TIMESTAMP				X		
DECIMAL			X		X	
BOOLEAN						X

필터 및 조건 기준을 지정하는 조회 테이블 표현식에서 속성 유형 또는 비교되는 값은 반드시 호환 가능해야 합니다. 예를 들어, `WI.OWNER_ID=1`은 왼쪽의 피연산자 유형이 STRING이고 오른쪽 피연산자 유형은 NUMBER이기 때문에 유효하지 않은 필터입니다.

조회 테이블 조회

비즈니스 플로우 관리자 EJB 및 REST API에서 사용 가능한 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회가 실행됩니다.

조회는 하나의 조회 테이블에서만 실행됩니다. 엔티티 기반 API 메소드 및 행 기반 API 메소드를 사용하여 조회 테이블에서 컨텐츠를 검색합니다. 입력 매개변수가 조회 테이블 API의 메소드에 전달됩니다.

관련 개념

364 페이지의『조회 테이블 개발』
Business Process Choreographer의 추가 및 복합 조회 테이블은 응용프로그램 개발 중에 조회 테이블 빌더를 사용하여 개발됩니다. 사전정의된 조회 테이블은 개발되거나 전개될 수 없습니다. Business Process Choreographer가 설치될 때 사용 가능하며 Business Process Choreographer 데이터베이스 스키마의 아티팩트에 대한 간단한 보기를 제공합니다.

351 페이지의『사전정의된 조회 테이블』
사전정의된 조회 테이블에서는 Business Process Choreographer 데이터베이스에 있는 데이터에 대한 액세스를 제공합니다. 이러한 테이블은 TASK 보기 또는 PROCESS_INSTANCE 보기와 같은 해당 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. 이 사전 정의된 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화되어 있으므로 사전 정의된 데이터베이스 보기의 기능 및 성능이 향상됩니다.

354 페이지의『추가 조회 테이블』
Business Process Choreographer의 추가 조회 테이블은 Business Process Choreographer가 관리하지 않는 조회 테이블 API 비즈니스 데이터에 표시됩니다. 추가 조회 테이블을 사용하면 비즈니스 프로세스 인스턴스 정보 또는 휴먼 타스크 정보 검색 시 사전정의된 조회 테이블의 데이터와 함께 이 외부 데이터를 사용할 수 있습니다.

356 페이지의『복합 조회 테이블』
Business Process Choreographer의 복합 조회 테이블에는 데이터베이스에 데이터의 특정 표시가 없습니다. 복합 조회 테이블은 사전 정의된 관련 조회 테이블 및 보충 조회 테이블의 데이터 조합을 구성합니다. 복합 조회 테이블을 사용하여 내가 수행할 작업과 같은 프로세스 인스턴스 목록 또는 타스크 목록에 대한 정보를 검색하십시오.

368 페이지의『조회 테이블의 필터 및 선택 기준』
필터 및 선택 기준은 SQL WHERE절과 비슷한 구문을 사용하는 조회 테이블 빌더를 사용하여 조회 테이블 개발 중에 정의됩니다. 명확하게 정의된 이러한 필터 및 선택 기준을 사용하여 조회 테이블의 속성을 기반으로 하는 조건을 지정하십시오.

조회 테이블 API 메소드:

조회는 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에 대해 실행됩니다. 엔티티 기반 API 메소드 및 행 기반 API 메소드를 사용하여 조회 테이블에서 컨텐츠를 검색합니다.

조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행하기 위해 다음 엔티티 기반 메소드 및 행 기반 메소드가 제공됩니다.

표 44. 조회 테이블에 대해 실행되는 조회의 메소드

목적	메소드
조회 컨텐츠	• queryEntities • queryRows 두 메소드 모두 조회 테이블의 컨텐츠를 리턴합니다. queryEntities 메소드는 엔티티를 기반으로 컨텐츠를 리턴하고 queryRows는 행을 기반으로 컨텐츠를 리턴합니다.
오브젝트 수 조회	• queryEntityCount • queryRowCount 두 메소드 모두 조회 테이블에 있는 오브젝트 수를 리턴하지만 실제 수는 엔티티 기반 접근 방법과 행 기반 접근 방법 중 어느 것을 사용하는지에 따라 다를 수 있습니다.

queryEntities 메소드와 queryEntityCount 메소드를 사용하는 엔티티 기반 조회에서는 조회 테이블에 1차 조회 테이블에서 1차 키로 정의한 대로 고유하게 식별 가능한 엔티티가 포함되어 있다고 가정합니다.

queryRows 메소드와 queryRowCount 메소드를 사용하는 열 기반 조회에서는 행을 기반으로 하며 내부에서 탐색하는 데 필요한 first 및 next 메소드를 제공하는 JDBC와 같은 결과 세트를 리턴합니다. 조회 테이블 API를 사용하여 조회 테이블에서 조회를 실행하는 경우 리턴되는 결과 세트를 조회 API를 통해 리턴되는 QueryResultSet와 비교할 수 있습니다. 일반적으로 행 수는 조회 테이블에 포함된 엔티티의 수보다 많습니다. 예를 들어, 해당 타스크 ID로 식별되는 휴먼 타스크와 같은 동일한 엔티티(예: TKIID)가 행 결과 세트에서 여러 번 발생할 수 있습니다.

사전정의된 조회 테이블에 포함된 특정 인스턴스는 Business Process Choreographer 환경에서 한 번만 존재합니다. 인스턴스의 예로는 휴먼 타스크와 비즈니스 프로세스가 있습니다. 이러한 인스턴스는 ID나 ID 세트를 사용하여 고유하게 식별됩니다. 휴먼 타스크 인스턴스의 경우에는 TKIID이고 프로세스 인스턴스의 경우에는 PIID입니다.

복합 조회 테이블은 하나의 1차 조회 테이블과 0개 이상의 첨부된 조회 테이블로 구성됩니다. 복합 조회 테이블에 포함된 오브젝트는 1차 조회 테이블에 포함된 오브젝트의 고유 ID로 고유하게 식별됩니다. 복합 조회 테이블의 1차 조회 테이블은 해당 엔티티 유형을 판별합니다. 예를 들어, TASK 1차 조회 테이블이 있는 복합 조회 테이블에는 TASK 유형의 엔티티가 포함되어 있습니다. 1차 조회 테이블과 첨부된 조회 테이블 간 일대일 또는 일대영 관계에서는 첨부된 조회 테이블의 엔티티가 중복되지 않습니다.

엔티티 기반 조회에서는 1차 조회 테이블에서 1차 키를 통해 정의된, 고유하게 식별할 수 있는 조회 테이블의 엔티티를 사용합니다. 사용자 인터페이스를 개발하는 클라이언트 응용프로그램 프로그래머는 일반적으로 중복되지 않는 고유한 인스턴스에 관심을 갖

습니다. 예를 들어, 사용자 인터페이스에서 휴먼 타스크를 한 번만 표시합니다. 엔티티 기반 조회 테이블 API가 사용되는 경우 고유 인스턴스가 리턴됩니다.

인스턴스 기반 인증이 사용되는 경우 행 기반 조회는 1차 조회 테이블의 중복 행을 리턴할 수 있습니다.

- WORK_ITEM 조회 테이블의 정보는 조회를 통해 검색됩니다. 예를 들어, 조회 테이블에서 정의된 속성과 WI.REASON 속성이 검색되면 결과에 여러 행이 규정됩니다. 이는 사용자가 여러 가지 이유로 타스크 또는 프로세스 인스턴스와 같은 엔티티에 액세스할 수 있기 때문입니다.
- 인스턴스 기반 인증이 사용되고 distinct가 지정되지 않았습니다. 작업 항목 정보가 검색되지 않더라도 인스턴스 기반 인증이 사용되면 여러 행이 리턴될 수 있습니다.

엔티티 기반 조회 테이블 API가 사용되는 경우:

- 엔티티 기반 조회는 항상 SQL Distinct 연산자와 함께 실행됩니다.
- 엔티티 기반 조회는 작업 항목 관련 정보에 대해 배열 값을 허용하는 결과를 리턴합니다.

조회 테이블 API 매개변수:

Business Process Choreographer에서 조회 테이블에 대해 조회를 실행하는 경우 조회 테이블 API 메소드를 사용하여 컨텐츠를 검색합니다.

다음 입력 매개변수가 조회 테이블 API의 메소드에 전달됩니다.

표 45. 조회 테이블 API의 매개변수

매개변수	선택사항	유형 및 설명
조회 테이블 이름	없음	java.lang.String
		조회 테이블의 고유 이름입니다.
필터 옵션	있음	비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.bpe.api.FilterOptions 또는 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.task.api.FilterOptions.
		조회를 정의하는 데 사용할 수 있는 옵션입니다. 예를 들어, 조회 임계값이 이 매개변수에 설정되어 리턴되는 결과 수를 제한합니다.
권한 옵션	있음	비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.bpe.api.AuthorizationOptions 또는 com.ibm.bpe.api.AdminAuthorizationOptions. 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.task.api.AuthorizationOptions 또는 com.ibm.task.api.AdminAuthorizationOptions.
		인스턴스 기반 권한이 사용되는 경우 권한을 더 자세히 제한할 수 있습니다. 역할 기반 권한이 필요한 조회 테이블의 경우 AdminAuthorizationOptions의 인스턴스를 전달해야 합니다.

표 45. 조회 테이블 API의 매개변수 (계속)

매개변수	선택사항	유형 및 설명
매개변수	있음	비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.bpe.api.Parameter의 java.util.List 또는 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.task.api.Parameter.
		이 매개변수는 복합 조회 테이블의 필터 또는 선택 기준에 지정된 사용자 매개변수를 전달하는 데 사용됩니다.

조회는 하나의 특정 조회 테이블에서만 실행됩니다. 여러 조회 테이블 간 관계는 복합 조회 테이블로 정의됩니다. 조회 테이블 API와 구별되는 조회 API의 경우 이는 데이터베이스 보기에 해당합니다.

조회 테이블 빌더를 사용하여 조회 테이블을 개발할 때 표현식에 필터 및 선택 기준을 지정합니다. 자세한 정보는 Information Center에서 복합 조회 테이블에 대한 주제와 조회 테이블의 필터 및 검색 기준에 대한 주제를 참조하십시오. 조회 테이블 빌더에 대한 정보는 WebSphere Business Process Management SupportPacs 사이트를 참조하십시오. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

조회 테이블 이름:

Business Process Choreographer에서 조회 테이블에 대한 조회를 실행하면 조회 테이블 이름이 입력 매개변수로 조회 테이블 API의 메소드에 전달됩니다.

조회 테이블 이름은 조회가 실행되는 조회 테이블의 이름입니다.

- 사전정의된 조회 테이블의 경우, 이것은 사전정의된 조회 테이블의 이름입니다.
- 복합 및 추가 조회 테이블의 경우 이 이름은 조회 테이블 모델링 중에 지정된 해당 조회 테이블의 이름입니다. 복합 또는 추가 조회 테이블의 이름은 *prefix.name* 이름 지정 규칙을 따르며 *prefix*는 'IBM'이 되어서는 안 됩니다.

조회 테이블 이름과 접두부는 모두 대문자여야 합니다. 조회 테이블 이름의 최대 길이는 28자입니다.

조회 테이블에 대한 필터 옵션:

Business Process Choreographer에서 조회 테이블에 대해 조회를 실행하는 경우 필터 옵션을 입력 매개변수로 조회 테이블 API의 메소드에 전달할 수 있습니다.

비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.bpe.api.FilterOptions 클래스의 인스턴스 또는 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.task.api.FilterOptions 클래스의 인스턴스가 조회 테이블 API로 전달될 수 있습니다. 필터 옵션을 사용하면 다음을 사용하여 조회를 구성할 수 있습니다.

- 임계값 및 오프셋(skipCount)
- 정렬 속성(SQL 조회의 ORDER BY절과 비슷함)
- 사용자 제공 조회 필터
- 리턴된 속성 세트(작업 항목 정보 포함)
- 기타

조회 테이블에서 얻을 수 있는 결과 세트는 조회 테이블의 정의에 의해 지정됩니다. 그러나 조회가 실행될 때 추가 옵션을 지정할 수 있습니다. 다음 표는 FilterOptions 오브젝트를 사용하여 필터 옵션으로 지정할 수 있는 옵션에 대해 설명합니다.

표 46. 조회 테이블 API 매개변수: 필터 옵션

옵션	유형	설명
선택된 속성	java.lang.String	• 결과 세트에 리턴되어야 하는 쉼표로 구분된 조회 테이블 속성 목록입니다. • 인스턴스 기반 권한이 사용되는 경우 접두부가 `WI.`인 WORK_ITEM 조회 테이블의 속성을 지정하여 작업 항목 정보를 검색할 수 있습니다(예: `WI.REASON`). • 널이 지정된 경우 작업 항목 정보 없이 조회 테이블의 모든 속성이 리턴됩니다.
정렬 속성	java.lang.String	쉼표로 구분된 조회 테이블 속성 목록입니다(선택적으로 뒤에 ASC(오름차순) 또는 DESC(내림차순)가 붙음). 이 목록은 SQL ORDER BY절과 비슷합니다. *sortAttributes* ::= *attribute* [ASC|DESC] [, *sortAttributes*]. ASC 또는 DESC가 지정되지 않은 경우에는 ASC가 지정된 것으로 가정합니다. 정렬은 정렬 속성의 순서대로 수행됩니다. 다음 예제는 TASK 조회 테이블의 타스크를 상태를 기준으로 내림차순으로 정렬하고 동일한 STATE의 그룹 내에서 NAME을 기준으로 오름차순으로 정렬합니다. `"STATE DESC, NAME ASC.`
임계값	java.lang.Integer	다음의 최대값을 정의합니다. • queryRows가 사용되는 경우 리턴되는 행 수. • queryEntities가 사용되는 경우 리턴되는 엔티티 수. 엔티티 결과 세트에 임계값 만큼의 엔티티가 포함되어 있지 않더라도 개별 조회 테이블에서 사용 가능한 실제 엔티티 수는 임계값을 초과할 수 있습니다. 이 문제는 작업 항목 정보를 선택한 경우 기술적인 이유 때문에 발생합니다. • queryRowCount 또는 queryEntityCount가 사용되는 경우 리턴되는 계수입니다. 기본값은 임계값이 설정되어 있지 않음을 의미하는 널입니다.
건너뛰기 수	java.lang.Integer	건너뛰는 행 수(행 기반 조회) 또는 엔티티 수(엔티티 기반 조회)를 정의합니다. 임계값 매개변수와 같이 엔티티 기반 조회에서는 skipCount가 정확하지 않을 수 있습니다. 건너뛰기 수는 큰 결과 세트에 대한 페이징을 허용하는 데 사용됩니다. 기본값은 skipCount가 설정되어 있지 않음을 의미하는 널입니다.

표 46. 조회 테이블 API 매개변수: 필터 옵션 (계속)

옵션	유형	설명
시간대	java.util.TimeZone	시간소인 변환 시 사용되는 시간대입니다. 예제로는 사전정의된 조회 테이블 TASK의 CREATED가 있습니다. 지정되지 않은 경우(널)에는 서버의 시간대가 사용됩니다.
로케일	java.util.Locale	$LOCALE 시스템 매개변수의 값을 계산하는 데 사용되는 로케일입니다. 선택 기준에서 $LOCALE을 사용하는 예는 다음과 같습니다. 'LOCALE=$LOCALE'.
별개의 행	java.lang.Boolean	행 기반 조회에만 사용됩니다. True로 설정된 경우 행 기반 조회에서는 별개의 행을 리턴합니다. 이 옵션은 작업 항목 정보의 다양성으로 인해 고유한 행이 리턴되는 것을 의미하지는 않습니다.
조회 조건	java.lang.String	이 옵션은 결과 세트에 대해 추가 필터링을 수행합니다. 조회 테이블에 정의된 모든 속성을 참조할 수 있습니다. 조회 테이블에 대한 권한이 필요한 경우, WI 접두부를 사용하여 WORK_ITEM 조회 테이블에 정의된 열을 참조할 수도 있습니다(예: WI.REASON=REASON_POTENTIAL_OWNER).

조회 테이블 API의 권한 옵션:

Business Process Choreographer에서 조회 테이블에 대해 조회를 실행하는 경우 권한 옵션을 입력 매개변수로 조회 테이블 API의 메소드에 전달할 수 있습니다.

조회가 실행될 때 비즈니스 플로우 관리자 EJB가 사용되는 경우 com.ibm.bpe.api.AuthorizationOptions 클래스 또는 com.ibm.bpe.api.AdminAuthorizationOptions 클래스의 인스턴스를 사용하고 휴먼 타스크 관리자 EJB가 사용되는 경우에는 com.ibm.task.api.AuthorizationOptions 클래스 또는 com.ibm.task.api.AdminAuthorizationOptions 클래스를 사용하여 추가 권한 옵션을 지정하십시오.

인스턴스 기반 권한이 사용되는 경우 AuthorizationOptions 클래스의 인스턴스를 사용하여 조회에서 리턴된 적합한 인스턴스를 식별하는 데 사용할 작업 항목 유형을 구체화할 수 있습니다.

인스턴스 데이터가 포함된 사전 정의된 조회 테이블에 대해 조회를 실행하는 경우 AuthorizationOptions 클래스의 인스턴스를 조회 테이블 API로 전달할 수 있습니다. 해당 인스턴스는 인스턴스 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블에서 조회가 실행되고 인스턴스 기반 권한이 사용되도록 구성되어 있는 경우에도 전달될 수 있습니다. 템플리트 데이터가 있는 사전 정의된 조회 테이블 또는 역할 기반 권한이 구성된 복합 조회 테이블에서 조회를 실행하는 경우, 비즈니스 플로우 관리자 EJB가 사용되는 경우에는 com.ibm.bpe.api.EngineNotAuthorizedException 예외가 처리되고 휴먼 타스크 관리자 EJB가 사용되는 경우에는 com.ibm.task.api.NotAuthorizedException 예외가 처리됩니다. 다른 경우에는 모두 조회 테이블 API에 전달된 권한 옵션이 무시됩니다.

복합 조회 테이블은 포함된 오브젝트(또는 엔티티) 식별 시 고려되는 작업 항목의 유형을 제한할 수 있습니다. 예를 들어, 조회 테이블 API에 전달되는 권한 옵션이 모든 사

용자 작업 항목을 사용하도록 구성되어 있으면 모든 사용자 작업 항목이 복합 조회 테이블의 정의에 사용되도록 정의되어 있는 경우에만 이 사항이 고려됩니다. 단순한 예로서 조회 테이블 정의에서 고려되도록 지정되지 않은 작업 항목 유형은 조회 테이블 API에서 고려하기 위해 겹쳐쓸 수 없지만 조회 테이블 정의에서 고려되도록 지정된 작업 항목 유형은 사용되지 않도록 겹쳐쓸 수 있습니다. 또한 복합 또는 사전 정의된 조회 테이블의 권한 유형을 조회 테이블 API로 겹쳐쓸 수 없습니다.

조회되는 조회 테이블 유형에 따라 권한 오브젝트가 지정되지 않았거나 관련 속성(모든 사용자, 개인, 그룹 또는 상속됨)이 널로 설정되어 있는 경우(기본값임) 다른 권한 옵션 기본값이 적용됩니다.

다음 표에서는 사용된 조회 테이블 유형 및 작업 항목 유형에 대한 인스턴스 기반 권한의 권한 옵션 기본값에 대해 설명합니다.

표 47. 조회 테이블 API 매개변수: 인스턴스 기반 권한의 권한 옵션 기본값

조회 테이블 유형	모든 사용자 작업 항목	개인 작업 항목	그룹 작업 항목	상속됨 작업 항목
인스턴스 데이터가 포함된 사전정의된 조회 테이블	TRUE	TRUE	TRUE	FALSE
템플리트 데이터가 포함된 사전정의된 조회 테이블	해당 없음	해당 없음	해당 없음	해당 없음
인스턴스 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블	TRUE	TRUE	TRUE	TRUE
템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블	해당 없음	해당 없음	해당 없음	해당 없음
추가	해당 없음	해당 없음	해당 없음	해당 없음

해당 없음은 인스턴스 기반 권한이 사용되지 않음을 의미하므로 작업 항목에 관한 권한 오브젝트의 모든 설정이 무시됩니다.

TRUE가 지정되면 조회 테이블이 특정 작업 항목 유형을 사용하도록 정의된 경우 결과 조회에서는 이 유형의 작업 항목만 고려합니다. 이는 인스턴스 데이터가 포함된 사전정의된 조회 테이블 모두에 적용되지만 복합 조회 테이블에는 적용되지 않습니다. 그룹 작업 항목의 경우에는 휴먼 타스크 컨테이너에 대해 후자도 사용 가능으로 설정해야 합니다. TRUE로 설정된 상속됨 작업 항목의 예는 프로세스 인스턴스의 관리자가 해당 프로세스 인스턴스에 대해 작성된 참여하는 휴먼 타스크 인스턴스를 확인할 수 있는 것입니다.

다음과 같은 경우 AuthorizationOptions 클래스의 인스턴스 대신 AdminAuthorizationOptions 클래스의 인스턴스를 지정하십시오.

- 역할 기반 권한이 있는 조회 테이블에서 조회가 실행됩니다. 템플리트 조회가 포함된 사전 정의된 조회 테이블에는 역할 기반 권한이 필요하며 템플리트 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블을 역할 기반 권한을 사용하도록 구성할 수 있습니다.
- 인스턴스 데이터가 포함된 조회 테이블이나 인스턴스 데이터가 포함된 1차 조회 테이블이 있는 복합 조회 테이블에서 조회가 실행되는 경우. 조회에서는 특정 사용자의 권한으로 인한 제한에 관계없이 해당 조회 테이블의 컨텐츠를 리턴합니다. 이 동작은 조회 테이블 API와 구별되는 조회 API에서 queryAll 메소드를 사용하는 것과 동등합니다.
- 다른 사용자를 대신하여 조회를 실행해야 하는 경우.

다음 표에서는 위의 다양한 동작을 완료하는 방법에 대해 설명합니다.

표 48. 조회 테이블 API 매개변수: AdminAuthorizationOptions

상황	설명
onBehalfUser가 널로 설정됨	• 역할 기반 권한이 있는 조회 테이블에서 조회가 실행되는 경우 해당 조회 테이블의 모든 컨텐츠가 리턴됩니다. • 인스턴스 기반 권한을 사용하는 조회 테이블에서 조회가 실행되는 경우 특정 사용자의 작업 항목이 있는지 해당 조회 테이블에 포함된 특정 오브젝트를 확인하지 않습니다. 조회 테이블에 포함된 모든 오브젝트가 리턴됩니다.
onBehalfUser가 특정 사용자로 설정됨	인스턴스 기반 권한이 사용되는 경우, 지정된 사용자의 권한으로 조회가 실행되고 해당 사용자의 작업 항목이 있는지 조회 테이블의 오브젝트를 확인합니다.

AdminAuthorizationOptions를 지정하는 경우, 호출자는 비즈니스 플로우 관리자 EJB가 사용되는 경우 BPESystemAdministrator 또는 BPESystemMonitor Java EE 역할이 있어야 하며 휴먼 타스크 관리자 EJB가 사용되는 경우 TaskSystemAdministrator 또는 TaskSystemMonitor Java EE 역할이 있어야 합니다.

관련 개념

374 페이지의『조회 테이블에 대한 권한』

조회 테이블에서 조회를 실행할 때 인스턴스 기반 권한 또는 역할 기반 권한을 사용하거나 아무 권한도 사용하지 않을 수 있습니다.

매개변수:

Business Process Choreographer에서 조회 테이블에 대한 조회를 실행할 때 사용자 매개변수를 입력 매개변수로 조회 테이블 API의 메소드에 전달할 수 있습니다. 조회 테

이블 정의에서는 1차 조회 테이블, 권한 및 조회 테이블에서 필터의 매개변수를 지정할 수 있습니다. 첨부된 조회 테이블의 선택 기준에서도 매개변수를 지정할 수 있습니다.

시스템 매개변수 $USER 및 $LOCALE은 런타임 시 필터와 선택 기준에서 바뀌며 조회 테이블 API로 이들 매개변수를 전달할 필요가 없습니다. 필터 옵션에서 로케일을 설정하여 $LOCALE 시스템 매개변수 계산에 필요한 입력 값을 제공합니다.

조회가 실행되면 사용자 매개변수를 조회 테이블 API에 전달해야 합니다. 이는 비즈니스 플로우 관리자 EJB JavaBeans가 사용되는 경우 com.ibm.bpe.api.Parameter 클래스 인스턴스 목록 또는 휴먼 타스크 관리자 EJB JavaBeans가 사용되는 경우 com.ibm.task.api.Parameter 클래스 인스턴스를 전달하여 수행됩니다.

매개변수 오브젝트에 다음 특성을 지정해야 합니다.

표 49. 조회 테이블 API의 사용자 매개변수

특성	설명
이름	조회 테이블 정의에 사용된 매개변수의 이름입니다. 이 이름은 대소문자를 구분합니다.
값	매개변수의 값입니다. 매개변수 유형은 해당 매개변수가 사용되는 모든 필터 및 선택 기준의 왼쪽 피연산자 유형과 호환 가능해야 합니다. 사전 정의된 조회 테이블의 일부 속성에 정의된 상수를 문자열로 전달할 수 있습니다(예: STATE_READY).

예제

```
// execute a query against a composite query
// table CUST.CPM with the primary query table filter
// set to 'STATE=PARAM(theState)'
EntityResultSet ers = null;
List parameterList = new ArrayList();
parameterList.add(new Parameter
("theState", new Integer(2)));

// run the query;
// the business flow manager EJB or the
// human task manager EJB can be used to access query tables
ers = bfm.queryEntities
("CUST.CPM", null, null, parameterList);

// work on the result set
// ...
```

조회 테이블 조회 결과:

Business Process Choreographer에서 조회 테이블에 대한 조회를 실행하는 경우 조회 테이블 API 메소드를 사용합니다. queryEntityCount 메소드 또는 queryRowCount 메소드 조회의 결과는 숫자입니다. queryEntities 및 queryRows 메소드는 결과 세트를 리턴합니다.

EntityResultSet

비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 queryEntities 메소드는 com.ibm.bpe.api.EntityResultSet 클래스의 인스턴스를 리턴합니다. 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 메소드는 com.ibm.task.api.EntityResultSet 클래스의 인스턴스를 리턴합니다. 엔티티 결과 세트에는 다음과 같은 특성이 있습니다.

표 50. 조회 테이블 API 엔티티의 엔티티 결과 세트 특성

특성	설명
queryTableName	조회가 실행된 조회 테이블의 이름입니다.
entityTypeName	• 복합 조회 테이블에 대해 조회가 실행된 경우 이 이름은 1차 조회 테이블의 이름입니다. • 사전정의된 조회 테이블이나 추가 조회 테이블에 대해 조회가 실행된 경우 이 이름은 조회 테이블의 이름이며 *queryTableName* 특성과 값이 동일합니다.
EntityInfo	이 특성에는 엔티티 결과 세트에 포함된 엔티티의 메타 정보가 포함되어 있습니다. 비즈니스 플로우 관리자 EJB JavaBeans가 사용되는 경우 이 오브젝트에서 com.ibm.bpe.api.AttributeInfo 오브젝트의 java.util.List 목록 또는 휴먼 타스크 관리자 EJB JavaBeans가 사용되는 경우 com.ibm.task.api.AttributeInfo 오브젝트의 목록을 검색할 수 있습니다. 이 목록에는 이 결과 세트의 엔티티에 포함된 정보의 속성 이름 및 속성 유형이 포함되어 있습니다. 이러한 엔티티의 키를 구성하는 속성에 대한 메타 정보도 포함되어 있습니다.
entities	비즈니스 플로우 관리자 EJB가 사용되는 경우 com.ibm.bpe.api.Entity 오브젝트의 java.util.List 목록 또는 휴먼 타스크 관리자가 사용되는 경우 com.ibm.task.api.Entity 오브젝트의 목록입니다.
locale	$LOCALE 시스템 매개변수에 사용하도록 계산되는 로케일입니다.

Entity 클래스의 인스턴스에는 조회 테이블 조회에서 검색되는 정보가 포함되어 있습니다. 엔티티는 타스크, 프로세스 인스턴스, 활동 또는 에스컬레이션과 같은 고유하게 식별 가능한 오브젝트를 표시합니다. 다음 특성을 엔티티에 사용할 수 있습니다.

표 51. 조회 테이블 API 엔티티의 엔티티 특성

특성	설명
EntityInfo	엔티티 결과 세트에도 포함된 EntityInfo 오브젝트입니다. 비즈니스 플로우 관리자 EJB JavaBeans가 사용되는 경우 이 오브젝트에서 com.ibm.bpe.api.AttributeInfo 오브젝트의 java.util.List 목록 또는 휴먼 타스크 관리자 EJB JavaBeans가 사용되는 경우 com.ibm.task.api.AttributeInfo 오브젝트의 목록을 검색할 수 있습니다. 이 목록에는 이 결과 세트의 엔티티에 포함된 정보의 속성 이름 및 속성 유형이 포함되어 있습니다. 이러한 엔티티의 키를 구성하는 속성에 대한 메타 정보도 포함되어 있습니다.
attributeValue(*attributeName*)	이 엔티티에 적용하기 위해 검색되는 지정된 속성 값입니다. 유형은 이 속성의 관련 AttributeInfo 오브젝트에 포함되어 있습니다.

표 51. 조회 테이블 API 엔티티의 엔티티 특성 (계속)

특성	설명
attributeValuesOfArray (*attributeName*)	값의 배열입니다. 속성 정보 특성 *array*가 true로 설정되어 있는 경우 이 특성을 사용하십시오. 해당 속성이 작업 항목 정보를 참조하는 경우에만 true로 설정됩니다.

엔티티 결과 세트에 있는 엔티티 수는 엔티티 목록에서 size 메소드를 사용하여 검색합니다.

예제: 엔티티 기반 조회 테이블 API:

```
...
// the following example shows a query against
// predefined query table TASK, using the entity-based API

...
// run the query
// service is a (Local)BusinessFlowManager object or a
// (Local)HumanTaskManager object
EntityResultSet rs = service.queryEntities("TASK", null, null, null);

// get the entities meta information
EntityInfo ei = rs.getEntityInfo();
List atts = ei.getAttributeInfo();

// get the entities and iterate over it
Iterator entitiesIter = rs.getEntities().iterator();
while (entitiesIter.hasNext()) {

      // work on a particular entity
      Entity en = (Entity) entitiesIter.next();

      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = en.getAttributeValue(ai.getName()) ;

            // process...
      }
}
...
```

RowResultSet

비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 queryRows 메소드는 com.ibm.bpe.api.RowResultSet 클래스의 인스턴스를 리턴합니다. 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 queryRows 메소드는 com.ibm.task.api.RowResultSet 클래스의 인스턴스를 리턴합니다. 이 유형의 결과 세트는 JDBC 결과 세트와 비슷합니다. 행 결과 세트에는 다음과 같은 특성이 있습니다.

표 52. 조회 테이블 API 행의 행 결과 세트 특성

특성	설명
primaryQueryTableName	• 복합 조회 테이블에 대해 조회가 실행된 경우 이 이름은 1차 조회 테이블의 이름입니다. • 사전정의된 조회 테이블이나 추가 조회 테이블에 대해 조회가 실행된 경우 이 이름은 조회 테이블의 이름이며 *queryTableName* 특성과 값이 동일합니다.
attributeInfo	이 특성에는 비즈니스 플로우 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.bpe.api.AttributeInfo 오브젝트의 목록 또는 휴먼 타스크 관리자 Enterprise JavaBeans가 사용되는 경우 com.ibm.task.api.AttributeInfo 오브젝트의 목록이 포함되어 있습니다. 이는 이 결과 세트에 대한 메타 정보를 설명합니다. AttributeInfo 오브젝트에는 정보의 속성 이름 및 속성 유형이 포함되어 있습니다. 행 결과 세트에 키가 없기 때문에 키에 대한 메타데이터는 포함되지 않습니다.
attributeValue	이 행에 대해 검색된 지정된 속성의 값입니다. 유형은 이 속성의 관련 AttributeInfo 오브젝트에 포함되어 있습니다.
next, first, last, previous	이러한 메소드를 사용하여 행 결과 세트를 탐색합니다. 행 결과 세트의 사용법을 반복기, 열거 또는 JDBC 결과 세트와 비교하십시오.

행 결과 세트의 행 수는 행 목록에서 size 메소드를 사용하여 검색합니다.

예제: 행 기반 조회 테이블 API

```
...
// the following example shows a query against
// predefined query table TASK, using the entity-based API
...
// run the query
// service is a (Local)BusinessFlowManager object or a
// (Local)HumanTaskManager object
RowResultSet rs = service.queryRows("TASK", null, null, null);

// get the entities meta information
List atts = rs.getAttributeInfo();

// get the entities and iterate over it
while (rs.next()) {

      // work on a particular row
      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value =

     rs.getAttributeValue(ai.getName()) ;

            // process...
      }
}
...
```

메타데이터 검색을 위한 조회 테이블 조회

조회는 조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에 대해 실행됩니다. 조회 테이블에서 메타데이터를 검색하기 위한 메소드가 있습니다.

조회 테이블 API를 사용하여 Business Process Choreographer의 조회 테이블에서 조회를 실행하는 경우 메타데이터를 검색하기 위해 다음 메소드가 제공됩니다.

표 53. 조회 테이블에서 메타데이터 검색에 사용할 메소드

목적	메소드
특정 조회 테이블의 메타데이터 리턴	getQueryTableMetaData
특정 특성을 가진 조회 테이블 메타데이터의 목록 리턴	findQueryTableMetaData
엔티티를 기본으로 하여 조회 테이블의 컨텐츠와 선택된 속성에 해당되는 메타데이터의 서브세트 리턴	queryEntities
행을 기본으로 하여 조회 테이블의 컨텐츠와 선택된 속성에 해당되는 메타데이터의 서브세트 리턴	queryRows

조회 테이블의 메타데이터는 구조와 관련된 데이터 및 자국어 지원과 관련된 데이터로 구성됩니다.

다음 표에는 조회 테이블 구조와 관련된 메타데이터가 표시되어 있습니다.

표 54. 조회 테이블 구조와 관련된 메타데이터

메타데이터	설명	**getQuery-TableMetaData**를 통해 리턴됨	**findQuery-TableMetaData**를 통해 리턴됨	**queryEntities**를 통해 리턴됨	**queryRows**를 통해 리턴됨
조회 테이블 이름	조회 테이블의 이름	예	예	예	예
1차 조회 테이블 이름	보충 및 사전 정의된 조회 테이블의 경우 조회 테이블의 이름, 복합 조회 테이블의 경우에는 1차 조회 테이블의 이름	예	예	예	예
유형	조회 테이블의 유형(사전 정의, 복합 또는 보충)	예	예	아니오	아니오
권한	조회 테이블에 정의된 권한 • 작업 항목 사용 • 인스턴스 기반, 역할 기반 또는 권한 없음	예	예	아니오	아니오
정의된 속성	조회 테이블에 정의된 속성의 메타데이터	예	예	아니오. 선택된 속성의 메타데이터가 리턴됩니다.	아니오. 선택된 속성의 메타데이터가 리턴됩니다.
키 속성	조회 테이블의 키 속성	예	예	예	아니오. 행 기반 조회에 적용 불가능합니다.

다음 표에는 조회 테이블 자국어 지원과 관련된 메타데이터가 표시되어 있습니다.

표 55. 조회 테이블 자국어 지원과 관련된 메타데이터

메타데이터	설명	getQuery-TableMetaData를 통해 리턴됨	findQuery-TableMetaData를 통해 리턴됨	queryEntities를 통해 리턴됨	queryRows를 통해 리턴됨
locales[]	조회 테이블의 표시 이름 및 설명이 정의된 로케일입니다.	예	예	아니오	아니오
로케일	API에 전달된 로케일로 인해 결정되는 $LOCALE 시스템 매개변수 값입니다.	예	예	예	예
조회 테이블의 표시 이름 및 설명	정의된 모든 로케일에 제공되는 조회 테이블의 표시 이름 및 설명입니다.	예	예	아니오	아니오
속성의 표시 이름 및 설명	정의된 모든 로케일에 제공되는 속성의 표시 이름 및 설명입니다.	예	예	아니오	아니오

조회 테이블 메타데이터를 리턴하는 모든 EJB 조회 테이블 API 메소드에서 FilterOptions.setLocale 및 MetaDataOptions.setLocale과 같은 로케일 매개변수를 사용할 수 있습니다. 이 매개변수는 클라이언트가 사용자에게 정보를 제공하기 위해 사용하는 Java 로케일로 설정되어야 합니다. 이 로케일 매개변수는 필터 및 선택 기준에서 사용할 수 있는 $LOCALE 시스템 매개변수 값을 계산하는 데 사용됩니다. 리턴되는 로케일에는 $LOCALE에 사용되는 실제 Java 로케일이 포함됩니다.

특정 조회 테이블의 표시 이름 및 설명이 검색된 경우 getLocale을 관련 메소드에 전달하여 타스크의 설명과 같은 로케일의 표시 이름 및 설명을 가져오십시오. 예를 들어, 'LOCALE=$LOCALE'의 선택 기준을 사용하여 다음 설명이 첨부됩니다.

예제

```
// the following example shows how meta data for a particular
// composite query table can be retrieved

...
// run the query
MetaDataOptions mdo = new MetaDataOptions("TASK", null, false
, new Locale("en_US"));
List list = bfm.findQueryTableMetaData(mdo);

// to get the meta data of a specific query table
// use bfm.getQueryTableMetaData(...)

// iterate through the list of query tables that have
TASK as primary query table
// => at least one query table is returned:
the predefined query table TASK

Iterator iter = list.iterator();
while( iter.hasNext() ) {
 QueryTableMetaData md = (QueryTableMetaData) iter.next();
 Locale effectiveLocale = md.getLocale();
 String queryTableDisplayName = md.getDisplayName(effectiveLocale);
 System.out.println("found query table: " + queryTableDisplayName);
 List attributesList = md.getAttributeMetaData();
```

```
 Iterator attrIter = attributesList.iterator();
 while (attrIter.hasNext()) {
  AttributeMetaData amd = (AttributeMetaData) attrIter.next();
  String attributeDisplayName = amd.getDisplayName(effectiveLocale);
  System.out.println("₩tattribute:" + attributeDisplayName);
 }
}
```

최적으로 일치하는 로케일

첨부된 조회 테이블에 조건을 지정할 때 $LOCALE 값을 사용하면 지정된 로케일이 메타데이터와 정확하게 일치하지 않는 경우 예상치 못한 결과를 리턴할 수 있습니다. 예를 들어, 언어를 en으로 지정하는 메타데이터가 있는 조회 테이블에 대해 로케일 en_US의 조회를 전달하는 경우 리턴되는 결과는 null이 됩니다.

이러한 상황이 발생하지 않도록 하려면 최적으로 일치하는 알고리즘을 적용하여 조회에 사용된 실제 로케일을 계산하는 LOCALE=$LOCALE_BEST_MATCH를 사용할 수 있습니다. 예를 들어, 언어가 en인 조회 테이블에 대해 로케일 en_US의 조회를 전달하는 경우 en을 사용하여 수행됩니다.

LOCALE=$LOCALE_BEST_MATCH 조건에는 다른 논리 또는 비교 연산자를 지정할 수 없습니다. 첨부된 조회 테이블에서 최적으로 일치하는 로케일 조건만 사용할 수 있습니다. 이를 다른 조회에 조건으로 지정하는 경우 오류가 발생합니다.

조회 테이블 메타데이터 자국어 지원

조회 테이블 메타데이터와 관련하여 자국어 지원이 지원됩니다.

여러 로케일로 복합 조회 테이블에 표시 이름 및 설명을 제공할 수 있습니다. 예를 들어, 복합 조회 테이블에서 en_US 로케일, de 로케일 및 기본 로케일로 조회 테이블의 표시 이름을 정의할 수 있습니다. 조회 테이블 빌더를 사용하여 조회 테이블을 개발할 때 이러한 정의가 수행됩니다. 로컬화된 표시 이름 및 설명이 포함된 조회 테이블을 전개하려면 Business Process Choreographer Container에 조회 테이블을 전개할 때 -deploy jarFile 옵션을 사용해야 합니다.

로케일 처리와 관련해서는 조회 테이블 API 메소드의 동작 queryEntities 및 queryRows와 조회 테이블 API의 메타데이터 메소드 getQueryTableMetaData 및 findQueryTableMetaData가 Java 자원 번들에서 제공되는 것과 비슷합니다.

조회 테이블 메타데이터의 표시 이름 및 설명을 조회 테이블의 컨텐츠와 일치시키기 위해 $LOCALE 시스템 매개변수는 조회 테이블에서 표시 이름 및 설명이 지정된 로케일에 따라 다릅니다.

예제

타스크 목록 또는 프로세스 목록을 표시하고 요청을 작성하여 조회 테이블을 조회하는, 클라이언트의 다음 시나리오를 검토합니다.

- 클라이언트가 사용자에게 정보를 제공하기 위해 사용하는 로케일을 지정하지 않았습니다. 응용프로그램에서 다른 언어를 사용할 수 있도록 설정되어 있지 않을 가능성이 높습니다.
  - 표시 이름 및 설명에 사용할 기본 로케일이 조회 테이블에 지정됩니다. 현재 버전의 조회 테이블 빌더를 사용하여 빌드된 모든 복합 및 보충 조회 테이블에도 기본 로케일이 지정됩니다. 따라서 $LOCALE의 값은 default로 설정됩니다.
  - 조회 테이블에서는 기본 로케일을 사용할 조회 테이블의 표시 이름 또는 설명을 지정하지 않습니다. -deploy qtdFile 옵션을 사용하여 전개된 모든 조회 테이블 및 모든 사전 정의된 조회 테이블에서도 표시 이름 및 설명이 지정되지 않습니다. $LOCALE 값은 Java 자원 번들 메소드에 따라 결정됩니다.
- 클라이언트가 사용자에게 정보를 제공하기 위해 사용하는 로케일을 지정했습니다. 예를 들면, 조회 테이블의 REST API를 사용하는 경우에 해당됩니다.
  - 표시 이름 및 설명이 조회 테이블에 지정됩니다. 클라이언트가 전달한 로케일에 따라서 Java 자원 번들 메소드를 사용하여 $LOCALE 값을 계산합니다.
  - 표시 이름 및 설명이 조회 테이블에 지정되지 않습니다. $LOCALE 값은 클라이언트가 전달한 값으로 설정됩니다.

최적으로 일치하는 로케일

첨부된 조회 테이블에 조건을 지정할 때 $LOCALE 값을 사용하면 지정된 로케일이 메타데이터와 정확하게 일치하지 않는 경우 예상치 못한 결과를 리턴할 수 있습니다. 예를 들어, 언어를 en으로 지정하는 메타데이터가 있는 조회 테이블에 대해 로케일 en_US의 조회를 전달하는 경우 리턴되는 결과는 null이 됩니다.

이러한 상황이 발생하지 않도록 하려면 최적으로 일치하는 알고리즘을 적용하여 조회에 사용된 실제 로케일을 계산하는 LOCALE=$LOCALE_BEST_MATCH를 사용할 수 있습니다. 예를 들어, 언어가 en인 조회 테이블에 대해 로케일 en_US의 조회를 전달하는 경우 en을 사용하여 수행됩니다.

LOCALE=$LOCALE_BEST_MATCH 조건에는 다른 논리 또는 비교 연산자를 지정할 수 없습니다. 첨부된 조회 테이블에서 최적으로 일치하는 로케일 조건만 사용할 수 있습니다. 이를 다른 조회에 조건으로 지정하는 경우 오류가 발생합니다.

관련 태스크

406 페이지의『Business Process Choreographer Explorer에 대한 조회 테이블 작성』

EJB query API 대신 조회 테이블을 사용하면 Business Process Choreographer Explorer의 성능이 향상됩니다. 조회 테이블을 작성하려면 조회 테이블 빌더를 사용하십시오.

조회 테이블 및 조회 성능

조회 테이블은 Business Process Choreographer에서 휴먼 타스크 및 비즈니스 프로세스 목록을 검색하는 클라이언트 응용프로그램을 개발하는 데 필요한 정리 프로그래밍 모델을 소개합니다. 조회 테이블을 사용하면 성능이 향상됩니다. 조회 테이블 API 매개변수 및 성능에 영향을 미치는 기타 요소에 대한 정보가 제공됩니다.

조회 테이블의 조회 응답 시간은 사용되는 권한 옵션, 필터 및 선택 기준에 따라 주로 결정됩니다. 고려할 일반적인 성능 팁은 다음과 같습니다.

- 권한 옵션은 성능에 상당한 영향을 미칩니다. 개인 및 그룹 작업 항목과 같이 가능한 한 적은 옵션을 사용하여 권한을 사용하십시오. 상속된 작업 항목 사용을 피하십시오. 권한 옵션은 조회가 실행될 때 더 제한될 수 있습니다. 또한 불필요한 경우에는 작업 항목을 사용하는 권한이 필수 권한이 아님을 지정하십시오.
- 작업 항목을 사용하는 권한이 필수이면, 권한 필터를 지정하십시오. 예를 들어, 잠재적 소유자 작업 항목이 있는 조회 테이블의 오브젝트만을 허용하려면 WI.REASON=REASON_POTENTIAL_OWNER를 사용하십시오.
- 1차 조회 테이블에서의 필터링은 효과적입니다. 예를 들어, TASK가 1차 조회 테이블인 조회 테이블에서 준비 상태의 타스크만을 허용하는 경우가 있습니다.
- 조회 필터는 물론, 조회가 실행될 때 전달되는 필터인 조회 테이블의 필터는 성능 측면에서 기본 필터보다 덜 효과적입니다.
- 가능하다면, 필터 및 선택사항 기준에서 매개변수 사용을 피하십시오.
- 필터 및 선택사항 기준에서 `LIKE` 연산자 사용을 피하십시오.

복합 조회 테이블 정의

다음 표에서는 복합 조회 테이블에 정의된 옵션이 조회 성능에 미치는 영향에 대한 정보를 제공합니다. 이 표에서는 복합 조회 테이블 정의와 관련된 기타 주제도 제공합니다. 성능 영향 열에 미친 영향은 평균 성능 영향이며 실제 영향 관찰 결과는 다를 수 있습니다.

표 56. 복합 조회 테이블 옵션의 조회 성능 영향

오브젝트 또는 주제	성능 영향	설명
조회 테이블 필터	부정적	조회 테이블에 대한 필터는 조회 성능에 가장 부정적인 영향을 미치는 필터입니다. 이러한 필터는 일반적으로 데이터베이스에 정의된 색인을 사용할 수 없습니다.
1차 조회 테이블 필터	긍정적	1차 조회 테이블에 대한 필터는 조회 결과 세트 계산의 초기 단계에서 높은 성능 필터링을 제공합니다. 1차 조회 테이블 필터를 사용하여 조회 테이블의 컨텐츠를 제한하는 것이 좋습니다.
권한 필터	긍정적	권한에 대한 필터는 1차 조회 테이블 필터가 조회의 성능을 향상시키는 방법과 같은 조회의 성능을 향상시킬 수 있습니다. 가능한 경우에는 권한 필터를 적용해야 합니다. 예를 들어, 독자 작업 항목을 고려하지 말아야 하는 경우 `WI.REASON=REASON_READER`를 지정하십시오.
선택 기준	없음	일대일 또는 일대영 관계를 충족하기 위해 일부 1차 조회 테이블과 첨부된 조회 테이블의 관계에는 선택 기준을 정의해야 합니다. 선택 기준은 적은 수의 행에 대해서만 평가되므로 일반적으로 성능에 미치는 영향이 적습니다.
매개변수	없음	현재는 조회 테이블에서 매개변수를 사용해도 성능에 부정적인 영향을 미치지 않습니다. 그러나 매개변수는 필요한 경우에만 사용해야 합니다.
인스턴스 기반 권한	부정적	인스턴스 기반 권한이 사용되는 경우 작업 항목이 존재하는지 조회 테이블의 각 오브젝트를 확인해야 합니다. 작업 항목은 WORK_ITEM 조회 테이블에서는 항목으로 표시됩니다. 이 확인 작업은 성능에 영향을 줍니다.
인스턴스 기반 권한 • 모든 사용자 • 개인 • 그룹 • 상속됨	부정적	조회 테이블에 사용하기 위해 지정된 작업 항목의 각 유형은 성능에 영향을 미칩니다. 조회 볼륨이 큰 응용프로그램에서는 개인 및 그룹 작업 항목만 사용하거나 둘 중 하나만 사용해야 합니다. 상속됨 작업 항목은 일반적으로 필요하지 않으며 특히 수행할 작업을 표시하는 휴먼 타스크를 리턴하는 타스크 목록 정의 시에는 필요하지 않습니다. 예를 들어, 엔클로징 비즈니스 프로세스에 대한 권한에 따라 사용자에게 읽기 액세스 권한이 있는 비즈니스 프로세스에 속하는 타스크 목록을 리턴하기 위한 경우와 같이 상속됨 작업 항목은 명백히 필요한 경우에만 사용해야 합니다.
역할 기반 권한 또는 권한 없음	없음	역할 기반 권한이 사용되거나 사용되는 권한이 없는 경우 작업 항목에 대한 확인이 수행되지 않습니다.
정의된 속성 수	현재는 없음	현재는 조회 테이블에 포함된 속성 수가 성능에 영향을 미치지 않습니다. 그러나 필요한 속성만 조회 테이블의 파트가 되어야 합니다.

조회 테이블 API

다음 표에서는 조회 테이블 API에 지정된 옵션의 조회 성능 영향에 대한 정보를 제공합니다. 성능 영향 열에 미친 영향은 평균 성능 영향이며 실제 영향 관찰 결과는 다를 수 있습니다.

표 57. 조회 테이블 API 옵션의 조회 성능 영향

옵션	성능 영향	설명
선택된 속성	부정적(영향이 거의 없는 것이 좋음)	조회 테이블에 대해 조회가 실행될 때 선택되는 속성 수는 데이터베이스와 Business Process Choreographer 조회 테이블 런타임 둘 다에서 처리해야 하는 수에 영향을 미칩니다. 또한 복합 조회 테이블의 경우에는 첨부된 조회 테이블이 선택된 속성에 의해 지정되거나 조회 테이블 필터 또는 조회 필터에 의해 참조되는 경우에만 해당 조회 테이블의 정보를 검색해야 합니다.
조회 필터	부정적	지정된 경우 현재 조회 필터는 조회 테이블 필터와 동일하게 성능에 영향을 미칩니다. 그러나 필터가 조회 테이블 API에 전달되는 대신 조회 테이블에 지정된 경우에는 조회 필터를 사용하는 것이 좋습니다.
정렬 속성	부정적	조회 결과 세트를 정렬하는 데는 자원이 많이 소비되며 정렬이 사용되는 경우에는 데이터베이스 최적화가 제한됩니다. 필요하지 않은 경우에는 정렬을 사용해서는 안 됩니다. 그러나 대부분의 응용프로그램에서는 정렬을 수행해야 합니다.
임계값	긍정적	임계값을 지정하면 조회 성능이 상당히 향상될 수 있습니다. 항상 임계값을 지정하는 것이 가장 좋습니다.
건너뛰기 수	부정적	조회 결과 세트에서 특정 수의 오브젝트를 건너뛰는 데는 자원이 많이 소비되므로 건너뛰기는 필요한 경우에만 수행해야 합니다(예: 조회 결과에 대한 페이징 시).
시간대	없음	시간대 설정은 성능에 영향을 미치지 않습니다.
로케일	없음	로케일 설정은 성능에 영향을 미치지 않습니다.
별개의 행	부정적	조회에서 별개의 행을 사용하면 성능에 일부 영향을 미치지만 중복되지 않은 행을 검색하려면 별개의 행을 사용해야 합니다. 이 옵션은 행 기반 조회에만 영향을 미치며 다른 경우에는 무시됩니다.
계수 조회	긍정적	특정 조회의 행 수 또는 총 엔티티 수가 필요한 경우(조회 테이블의 일부 항목에는 컨텐츠가 필요하지 않은 경우)에는 queryEntityCount 또는 queryRowCount 메소드를 사용해야 합니다. Business Process Choreographer 런타임은 계수 조회에 대해서만 유효한 최적화를 적용할 수 있습니다.

기타 고려사항

성능과 관련하여 고려해야 할 다른 요소는 다음과 같습니다.

표 58. 조회 테이블 성능: 기타 고려사항

항목	설명
시스템의 조회 테이블 수	Business Process Choreographer Container에 전개되는 조회 테이블 수는 조회 테이블 조회의 성능에 영향을 주지 않습니다. 또한 현재는 해당 조회 테이블 수가 비즈니스 프로세스 인스턴스 탐색에 영향을 주지 않으며 휴먼 타스크에 대한 청구 또는 완료 조작에도 영향을 미치지 않습니다. 유지보수를 위해 조회 테이블 수를 적절한 수준으로 유지하십시오. 일반적으로 하나의 조회 테이블은 사용자 인터페이스에 표시되는 하나의 타스크 목록 또는 프로세스 목록을 표시합니다.

표 58. 조회 테이블 성능: 기타 고려사항 (계속)

항목	설명
데이터베이스 조정	최적화된 SQL을 사용하여 조회 테이블의 컨텐츠에 액세스하더라도 Business Process Choreographer 데이터베이스에서 데이터베이스 조정을 구현해야 합니다. • 데이터베이스 메모리는 하드웨어 제한조건과 데이터베이스 서버에서 실행 중인 기타 프로세스를 고려하여 최대값으로 설정해야 합니다. • 데이터베이스에 대한 통계는 최신 상태여야 하며 주기적으로 갱신해야 합니다. 일반적으로 큰 토폴로지에는 이러한 프로시저가 이미 구현되어 있습니다. 예를 들어, 데이터베이스에서 데이터의 변경사항을 반영하려면 매주 한 번 최적화 프로그램을 위한 데이터베이스 통계를 수집하십시오. • 데이터베이스 시스템에서는 데이터 컨테이너 재구성(또는 조각 모음) 도구를 제공합니다. 데이터베이스에서 데이터의 실제 레이아웃도 조회 성능과 조회의 액세스 경로에 영향을 줄 수 있습니다. • 최적 색인은 양호한 조회 성능을 위해 중요합니다. Business Process Choreographer는 일반적인 시나리오의 조회 성능과 프로세스 탐색 둘 다에 대해 최적화된 사전정의된 색인과 함께 제공됩니다. 사용자 정의된 환경에서는 큰 볼륨의 타스크 또는 프로세스 목록 조회를 지원하기 위해 추가 색인이 필요할 수 있습니다. 조회 테이블에서 실행되는 조회를 지원하려면 데이터베이스에서 제공하는 도구를 사용하십시오.

Business Space에 대한 조회 테이블 작성

조회 테이블 빌더에서 사전 정의된 특성이 있는 복합 조회 테이블 정의를 사용하여 Business Space에 대한 조회 테이블을 작성할 수 있습니다.

시작하기 전에

조회 테이블 빌더는 Eclipse 플러그인으로 사용할 수 있으며 WebSphere Business Process Management SupportPacs 사이트에서 다운로드할 수 있습니다. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

프로시저

1. 조회 테이블 빌더에서 프로젝트를 오른쪽 마우스 단추로 클릭한 다음 **새로 작성 → Business Space에 대한 복합 조회 정의**를 선택하십시오. 마법사의 지시사항에 따라 조회 테이블 정의를 작성하십시오. 새 조회 테이블 정의는 사전 정의된 특성으로 구성됩니다. 필요한 경우 조회 테이블 정의에 특성을 더 추가하고 조회 테이블 정의 파일을 WebSphere Process Server에 전개하십시오.

   주: 조회 테이블 빌더에서 특성에 지정한 이름이 Choreographer에 대한 Business Space에서 타스크 특성의 이름으로 사용됩니다.

2. 조회 테이블 정의 파일을 작성하고 전개한 후 Business Space에서 구성할 수 있습니다. 예를 들어, 조회 테이블 정의 파일을 전개한 후 타스크 목록 위젯에 대해 다음을 수행하십시오.
   a. 위젯 메뉴를 열고 **구성**을 선택한 다음 **컨텐츠** 탭을 클릭하십시오.
   b. **컨텐츠** 탭에서 **표시할 타스크 목록 선택** 드롭 다운 목록을 열어서 위젯 사용자가 사용할 수 있도록 설정할 수 있는 목록을 표시하십시오. **타스크 목록 추가**를 선택하십시오. 전개한 조회 테이블 정의를 이 목록에서 선택할 수 있어야 합니다.

   조회 테이블 정의를 사용할 수 없는 경우에는 조회 테이블 빌더로 돌아가서 정의 파일이 올바르게 정의되고 전개되었는지 확인하십시오.

Business Process Choreographer Explorer에 대한 조회 테이블 작성

EJB query API 대신 조회 테이블을 사용하면 Business Process Choreographer Explorer의 성능이 향상됩니다. 조회 테이블을 작성하려면 조회 테이블 빌더를 사용하십시오.

시작하기 전에

조회 테이블 빌더는 Eclipse 플러그인으로 사용할 수 있으며 WebSphere Business Process Management SupportPacs 사이트에서 다운로드할 수 있습니다. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크에 액세스하려면, 이 주제의 관련된 참조 절을 참조하십시오.

프로시저

1. 조회 테이블 빌더에서 프로젝트를 오른쪽 마우스 단추로 클릭한 다음 **새로 작성 → Business Space에 대한 복합 조회 정의**를 선택하십시오. 이 옵션은 Business Process Choreographer Explorer에 필요한 모든 열이 미리 선택되었는지 확인합니다.
2. 마법사의 지시사항에 따라 조회 테이블 정의를 작성하십시오. 필요한 경우 조회 테이블 정의에 특성을 더 추가하십시오. 조회 테이블을 정의하는 경우 다음과 같은 측면을 고려하십시오.

   필터 기준

   Business Process Choreographer Explorer에서 조회 테이블을 기반으로 보기를 작성하는 경우에는 검색 기준에 추가 필터 또는 변수를 지정할 수 없습니다. 조회 테이블을 작성할 때 이러한 필터 기준과 변수의 매개변수를 지정해야 합니다.

   Business Process Choreographer Explorer에서 조회 테이블 정의에 매개변수를 사용하여 둘 이상의 보기에 대해 조회 테이블을 사용할 수 있습니

다. 보다 융통성 있게 사용하려면 사용자 정의 보기에 대한 조회가 실행될 때 매개변수 기본값을 겹쳐쓸 수 있는지 여부도 지정할 수 있습니다.

권한 Business Process Choreographer Explorer에서 조회 테이블을 기반으로 보기를 작성하는 경우에는 사용자 역할에 따라 검색 기준을 필터링할 수 없습니다. 조회 테이블을 정의할 때 사용자 역할에 대한 필터 기준을 설정해야 합니다. 템플리트 정보를 기반으로 하는 1차 조회 테이블의 경우, 역할 기반 권한이 아니라 인스턴스 기반 권한을 권한 유형으로 사용하십시오. 인스턴스 정보를 기반으로 하는 1차 조회 테이블의 경우에는 적절한 인스턴스 기반 권한 필터를 지정하십시오.

자국어 지원

조회 테이블 빌더에서 특성을 정의하는 경우, 이 특성의 이름 및 설명을 다른 언어로 지정할 수도 있습니다. 사용자 정의된 보기에 대한 조회가 실행되는 경우 Business Process Choreographer Explorer는 브라우저의 언어 설정에 적합한 번역을 사용합니다.

조회 테이블 정의의 이름 및 설명 표시

조회 테이블 빌더에서는 보기가 지원하는 모든 언어로 이름 및 설명 표시를 제공할 수 있습니다.

열의 이름 및 설명 표시

런타임에 Business Process Choreographer Explorer는 결과 목록에 표시되는 알맞은 국제화된 열을 검색합니다. 1차 조회 테이블에서 가져온 열(예: PIID)의 경우, Business Process Choreographer Explorer는 지원되는 모든 언어에 대해 이미 사용 가능한 번역을 사용합니다.

첨부된 조회 테이블에서 가져온 열(예: QUERY_PROPERTY)의 경우에는 조회 테이블 빌더에서 사용자의 비즈니스가 지원하는 모든 언어로 표시 이름 및 설명을 제공해야 합니다.

타스크 이름 및 설명

WebSphere Integration Developer에 국제화된 타스크 이름 및 설명이 있는 경우 브라우저의 언어 및 국가 설정에 따라 Business Process Choreographer Explorer에 해당 이름 및 설명이 표시됩니다. 브라우저 설정이 프로세스 모델에 정의된 설정과 일치하지 않는 경우에는 기본 언어의 번역이 사용됩니다.

정렬 기준

조회 테이블 정의에 대한 정렬 기준을 정의하는 경우, 여러 가지 특성(예: 프로세스 상태)이 정수 값으로 저장되지만 Business Process Choreographer Explorer는 결과 목록에 번역된 문자열로 표시합니다. 따라서 일부 언어에서는 예상치 못한 정렬 결과가 생성될 수 있습니다.

새 조회 테이블 정의는 사전 정의된 특성 및 사용자가 추가한 모든 특성으로 구성됩니다.

다음에 수행할 작업

조회 테이블 빌더에서 Application Server에 조회 정의를 전개하고 테스트하십시오. Business Process Choreographer Explorer가 이 서버에 연결된 경우, 이제 Business Process Choreographer Explorer를 사용자 자신의 용도에 맞게 또는 다른 사용자 그룹을 위해 사용자 정의하여 조회 테이블을 사용할 수 있습니다. Business Process Choreographer Explorer가 다른 서버에 연결된 경우에는 적절한 서버에 조회 테이블을 전개해야만 이를 사용하여 사용자 정의된 보기를 작성할 수 있습니다.

관련 개념

374 페이지의 『조회 테이블에 대한 권한』
조회 테이블에서 조회를 실행할 때 인스턴스 기반 권한 또는 역할 기반 권한을 사용하거나 아무 권한도 사용하지 않을 수 있습니다.

368 페이지의 『조회 테이블의 필터 및 선택 기준』
필터 및 선택 기준은 SQL WHERE절과 비슷한 구문을 사용하는 조회 테이블 빌더를 사용하여 조회 테이블 개발 중에 정의됩니다. 명확하게 정의된 이러한 필터 및 선택 기준을 사용하여 조회 테이블의 속성을 기반으로 하는 조건을 지정하십시오.

400 페이지의 『조회 테이블 메타데이터 자국어 지원』
조회 테이블 메타데이터와 관련하여 자국어 지원이 지원됩니다.

Business Process Choreographer EJB 조회 API

서비스 API의 query 메소드 또는 queryAll 메소드를 사용하여 비즈니스 프로세스 및 타스크에 대한 저장된 정보를 검색합니다.

query 메소드는 모든 사용자가 호출할 수 있으며 작업 항목이 존재하는 오브젝트의 특성을 리턴합니다. queryAll 메소드는 Java EE 역할(BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor 또는 TaskSystemMonitor) 중 하나가 있는 사용자만 호출할 수 있습니다. 이 메소드는 데이터베이스에 저장된 모든 오브젝트의 특성을 리턴합니다.

모든 API 조회는 SQL 조회에 맵핑됩니다. 결과적인 SQL 조회의 양식은 다음 측면에 따라 다릅니다.

- Java EE 역할 중 하나가 있는 사용자가 조회를 호출했는지 여부
- 조회한 오브젝트. 오브젝트 특성을 조회할 수 있도록 사전 정의된 데이터베이스 보기가 제공됩니다.
- from 절, 조인 조건 및 액세스 제어에 대한 사용자 특정 조건의 삽입

사용자 정의 특성 및 변수 특성을 모두 조회에 포함시킬 수 있습니다. 조회에 몇 개의 사용자 정의 특성 또는 변수 특성을 포함시킬 경우, 해당하는 데이터베이스 테이블에서 자체 결합이 발생합니다. 데이터베이스 시스템에 따라 이러한 query() 호출이 성능에 영향을 미칠 수도 있습니다.

또한 createStoredQuery 메소드를 사용하여 Business Process Choreographer 데이터베이스에 조회를 저장할 수 있습니다. 저장된 조회를 정의하는 경우 조회 기준을 제공해야 합니다. 저장된 조회를 실행할 때 즉, 런타임 시 데이터가 어셈블될 때 조회 기준이 동적으로 적용됩니다. 저장된 조회에 매개변수가 포함된 경우에는 조회 실행 시 매개변수를 분석합니다.

Business Process Choreographer API에 대한 자세한 정보는 프로세스 관련 메소드의 경우 com.ibm.bpe.api 패키지에서, 타스크 관련 메소드의 경우 com.ibm.task.api 패키지에서 Javadoc을 참조하십시오.

API query 메소드의 구문

Business Process Choreographer API 조회의 구문은 SQL 조회와 유사합니다. Select 절, where 절, order-by 절, skip-tuple 매개변수, 임계값 매개변수 및 시간대 매개변수가 조회에 포함될 수 있습니다.

조회 구문은 오브젝트 유형에 따라 다릅니다. 다음 표는 각각의 서로 다른 오브젝트 유형의 구문을 표시합니다.

표 59. 여러 오브젝트 유형의 조회 구문

오브젝트	구문
프로세스 템플리트	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
타스크 템플리트	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
비즈니스-프로세스 및 타스크-관련 데이터	`QueryResultSet query (java.lang.String selectClause, java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer skipTuples java.lang.Integer threshold, java.util.TimeZone timezone);`

Select 절:

조회 함수의 Select 절은 조회에서 리턴할 오브젝트 특성을 식별합니다.

Select 절은 결과 조회를 설명합니다. 리턴할 오브젝트 특성(결과 열)를 식별하는 이름 목록을 지정합니다. 구문은 SQL SELECT 절의 구문과 유사해서 쉼표를 사용하여 절

의 각 부분을 구분합니다. 절의 각 부분은 사전 정의된 보기 중 하나에서 열을 지정해야 합니다. 보기 이름 및 열 이름으로 열을 완전히 지정해야 합니다. QueryResultSet 오브젝트의 리턴된 열은 Select 절에 지정된 열과 동일한 순서로 표시됩니다.

Select 절은 AVG(), SUM(), MIN() 또는 MAX()와 같은 SQL 집계 함수를 지원하지 않습니다.

조회할 수 있는 사용자 정의 특성 및 특성 값과 같은 여러 이름-값 쌍의 특성을 선택하려면 보기 이름에 한 자리수의 카운터를 추가하십시오. 이 카운터는 1에서 9 사이의 값을 취할 수 있습니다.

Select 절의 예제

- "WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON"

  작업 항목에 대한 연관된 오브젝트의 유형 및 지정 이유를 가져옵니다.
- "DISTINCT WORK_ITEM.OBJECT_ID"

  호출자가 작업 항목으로 보유하는 모든 오브젝트 ID를 중복되지 않게 가져옵니다.
- "ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON"

  호출자가 작업 항목으로 보유하는 활동의 이름 및 지정 이유를 가져옵니다.
- "ACTIVITY.STATE, PROCESS_INSTANCE.STARTER"

  연관된 프로세스 인스턴스의 활동 및 시작자의 상태를 가져옵니다.
- "DISTINCT TASK.TKIID, TASK.NAME"

  호출자가 작업 항목으로 보유하는 타스크의 모든 ID 및 이름을 중복되지 않게 가져옵니다.
- "TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE"

  추가로 where 절에 지정되는 사용자 정의 특성의 값을 가져옵니다.
- "QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE

  조회할 수 있는 변수의 특성 값을 가져옵니다. 이러한 파트는 추가로 where 절에 지정됩니다.
- "COUNT(DISTINCT TASK.TKIID)"

  where 절을 만족시키는 고유 타스크의 작업 항목 수를 계수합니다.

Where 절:

조회 함수의 where 절은 조회 도메인에 적용할 필터 기준을 설명합니다.

where 절의 구문은 SQL WHERE 절의 구문에 유사합니다. API where 절에 SQL from 절 또는 join 술부를 명시적으로 추가할 필요가 없으며 해당 구성은 조회가 실행될 때 자동으로 추가됩니다. 필터 기준을 적용하지 않으려는 where 절에 대해 null을 지정해야 합니다.

Where 절 구문에서는 다음 사항을 지원합니다.

- 키워드: AND, OR, NOT
- 비교 연산자: =, <=, <, <>, >,>=, LIKE

  LIKE 조작은 조회되는 데이터베이스에 정의된 와일드카드 문자를 지원합니다.
- 연산 설정: IN

다음 규칙도 적용됩니다.

- 오브젝트 ID 상수를 ID('string-rep-of-oid')로 지정하십시오.
- 2진 상수를 BIN('UTF-8 string')으로 지정하십시오.
- 정수 열거 대신 상징적 상수를 사용하십시오. 예를 들어, 활동 상태 표현식 ACTIVITY.STATE=2를 지정하는 대신 ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY를 지정하십시오.
- 비교 명령문에 있는 특성의 값에 작은따옴표 표시(')가 들어 있는 경우 따옴표를 중복 표시하십시오(예: "TASK_CPROP.STRING_VALUE='d''automatisation'").
- 한 자리의 접미부를 보기 이름에 추가하여 사용자 정의 특성과 같은 여러 이름-값 쌍의 특성을 참조하십시오(예: "TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'").
- 시간 소인 상수를 TS('yyyy-mm-ddThh:mm:ss')로 지정하십시오. 현재 날짜를 참조하려면 CURRENT_DATE를 시간 소인으로 지정하십시오.

  시간 소인에서 날짜 또는 시간 값을 최소한으로 지정해야 합니다.
  - 날짜만 지정하면 시간 값이 0으로 설정됩니다.
  - 시간만 지정하면 날짜는 현재 날짜로 설정됩니다.
  - 날짜를 지정하면 연도는 네 자리 숫자로 구성되어야 하고 월 및 일 값은 선택적입니다. 누락 월 및 일 값은 01로 설정됩니다. 예를 들어, TS('2003')는 TS('2003-01-01T00:00:00')와 동일합니다.
  - 시간을 지정하는 경우 이 값은 24시간 시스템으로 표시됩니다. 예를 들어, 현재 날짜가 2003년 1월 1일인 경우 TS('T16:04') 또는 TS('16:04')는 TS('2003-01-01T16:04:00')와 동일합니다.

Where 절의 예제

- 오브젝트 ID와 기존 ID 비교

```
"WORK_ITEM.WIID = ID('_WI:800c00ed.df8d7e7c.feffff80.38')"
```

이 where 절 유형은 보통 이전 호출에서 기존 오브젝트 ID로 동적으로 작성됩니다. 이 오브젝트 ID가 *wiid1* 변수에 저장되는 경우 해당 절을 다음과 같이 구성할 수 있습니다.

```
"WORK_ITEM.WIID = ID('" + wiid1.toString() + "')"
```

- 시간 소인 사용

```
"ACTIVITY.STARTED >= TS('2002-06-1T16.00.00')"
```

- 상징적 상수 사용

```
"WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER"
```

- 부울 값 true 및 false 사용

```
"ACTIVITY.BUSINESS_RELEVANCE = TRUE"
```

- 사용자 정의 특성 사용

```
"TASK_CPROP1.NAME = 'prop1' AND " TASK_CPROP1.STRING_VALUE = 'v1' AND
 TASK_CPROP2.NAME = 'prop2' AND " TASK_CPROP2.STRING_VALUE = 'v2'"
```

Order-by 절:

조회 함수의 order-by 절은 결과 조회 세트에 대한 정렬 기준을 지정합니다.

결과가 정렬되는 보기에서 열 목록을 지정할 수 있습니다. 이 열은 보기 및 열의 완전한 이름이어야 합니다.

Order-by 절 구문은 SQL order-by 절의 구문과 동일해서 쉼표를 사용하여 절의 각 부분을 구분합니다. ASC를 지정하여 열을 오름차순으로 정렬하고 DESC를 지정하여 열을 내림차순으로 정렬할 수도 있습니다. 결과 조회 세트를 정렬하지 않으려면 order-by 절에 null을 지정해야 합니다.

정렬 기준은 서버에 적용됩니다. 즉, 서버의 로케일이 정렬에 사용됩니다. 둘 이상의 열을 지정하는 경우 결과 조회 세트는 첫 번째 열의 값으로 정렬된 다음 두 번째 열의 값으로 정렬되는 식으로 진행됩니다. SQL 조회에 대해 지정할 수 있는 위치에서 order-by 절에 열을 지정할 수는 없습니다.

order-by 절의 예제

- "PROCESS_TEMPLATE.NAME"

  프로세스 템플리트 이름별로 알파벳 순으로 조회 결과를 정렬합니다.

- "PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC"

  조회 결과를 작성 날짜로 정렬하고 특정 날짜에 대해서는 프로세스 인스턴스 이름별로 알파벳 역순으로 결과를 정렬합니다.

- "ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE"

조회 결과를 활동 소유자별로 정렬한 후 활동 템플리트 이름별로 정렬한 다음 활동 상태별로 정렬합니다.

Skip-tuples 매개변수:

skip-tuples 매개변수는 결과 조회 세트가 시작될 때부터, 무시되어 결과 조회 세트의 호출자에게 리턴되지 않을 query-result-set 튜플의 수를 지정합니다.

임계값 매개변수가 지정된 이 매개변수를 사용하여 클라이언트 응용프로그램에서 페이징을 구현하십시오. (예를 들어, 처음 20개의 항목을 검색하고 그 다음 20개 항목씩 계속 검색하는 방법으로 검색하십시오.)

매개변수가 null로 설정되고 임계값 매개변수를 설정하지 않은 경우 모든 규정화된 튜플이 리턴됩니다.

skip-tuples 매개변수의 예제

- new Integer(5)

  처음 다섯개의 규정화된 튜플이 리턴되지 않도록 지정합니다.

Threshold 매개변수:

조회 함수의 threshold 매개변수는 서버에서 결과 조회 세트의 클라이언트로 리턴되는 오브젝트의 수를 제한합니다.

프로덕션 시나리오의 결과 조회 세트에는 수천 개 또는 수백만 개의 항목이 있을 수 있으므로 임계값 매개변수의 값을 지정하십시오. 임계값 매개변수를 적절하게 설정하면 데이터베이스 조회 속도가 빨라지고 서버에서 클라이언트로 전송해야 하는 데이터 수가 적어집니다. 예를 들어, 한 번에 소수의 항목만 표시해야 하는 그래픽 사용자 인터페이스에서는 임계값 매개변수가 유용합니다.

이 매개변수가 null로 설정되고 skip-tuples 매개변수를 설정하지 않는 경우 모든 규정화된 오브젝트가 리턴됩니다.

임계값 매개변수의 예제

- new Integer(50)

  50개의 규정화된 튜플이 리턴되도록 지정합니다.

Timezone 매개변수:

조회 함수의 time-zone 매개변수는 조회의 time-stamp 상수의 시간대를 정의합니다.

조회를 시작하는 클라이언트와 조회를 실행하는 서버의 시간대는 다를 수 있습니다. time-zone 매개변수를 사용하여 로컬 시간을 지정하는 것처럼 where 절에서 time-stamp 상수의 시간대를 지정합니다. 결과 조회 세트에서 리턴된 날짜는 조회에서 지정된 시간대와 동일하게 됩니다.

매개변수가 null로 설정되는 경우 시간 소인 상수가 UTC(Coordinated Universal Time) 시간으로 가정됩니다.

time-zone 매개변수의 예제

- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null,
                 (Integer)null,
                 java.util.TimeZone.getDefault() );
  ```

  2005년 1월 1일 로컬 시간 17:40 이후에 시작된 활동에 대한 오브젝트 ID를 리턴합니다.
- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (String)null, (Integer)null, (TimeZone)null);
  ```

  2005년 1월 1일 17:40 UTC 이후에 시작된 활동에 대한 오브젝트 ID를 리턴합니다. 예를 들어, 이 스펙은 동부 표준시로 6시간 전입니다.

조회에 변수를 사용하여 데이터 필터링:

조회 결과에서 조회 기준에 일치하는 오브젝트를 리턴합니다. 변수 값에 따라 결과를 필터링하고자 할 수도 있습니다.

이 태스크 정보

런타임 시 프로세스에 의해 사용되는 변수를 프로세스 모델에 정의할 수 있습니다. 해당 변수에 대해 조회할 수 있는 변수를 선언합니다.

예를 들어, John Smith는 자신의 보험 회사의 서비스 센터에 전화를 걸어 자신의 사고 차량의 보험 청구 진행 상태를 알아보고자 합니다. 청구 관리자는 고객 ID를 사용하여 청구서를 찾습니다.

프로시저

1. 옵션: 조회할 수 있는 프로세스의 변수 특성을 표시하십시오.

   프로세스 템플리트 ID를 사용하여 프로세스를 식별하십시오. 조회할 수 있는 변수를 알 경우 이 단계를 생략할 수 있습니다.

   ```
   List variableProperties = process.getQueryProperties(ptid);
   for (int i = 0; i < variableProperties.size(); i++)
   {
      QueryProperty queryData = (QueryProperty)variableProperties.get(i);
   ```

```
    String variableName = queryData.getVariableName();
    String name         = queryData.getName();
    int mappedType      = queryData.getMappedType();
    ...
}
```

2. 필터 기준에 일치하는 변수를 갖는 프로세스 인스턴스를 표시하십시오.

   이 프로세스에서 고객 ID는 조회할 수 있는 customerClaim 변수의 일부로 모델화됩니다. 따라서 고객 ID를 사용하여 청구서를 찾을 수 있습니다.

```
QueryResultSet result = process.query
    ("PROCESS_INSTANCE.NAME, QUERY_PROPERTY.STRING_VALUE",
     "QUERY_PROPERTY.VARIABLE_NAME = 'customerClaim' AND " +
     "QUERY_PROPERTY.NAME = 'customerID' AND " +
     "QUERY_PROPERTY.STRING_VALUE like 'Smith%'",
      (String)null, (Integer)null,
      (Integer)null, (TimeZone)null );
```

   이 조치는 프로세스 인스턴스 이름 및 Smith로 시작하는 ID를 가진 고객의 고객 ID 값을 포함하는 조회 결과 세트를 리턴합니다.

결과 조회:

조회 결과 세트에는 Business Process Choreographer API 조회의 결과가 포함됩니다.

결과 세트의 요소는 호출자가 제공한 where 절을 충족시키며 호출자가 볼 수 있게 권한이 부여된 오브젝트의 특성입니다. API next 메소드를 사용하여 상대적 형식으로 또는 first 및 last 메소드를 사용하여 절대적 형식으로 요소를 읽을 수 있습니다. 결과 조회 세트의 내부 커서는 첫 번째 요소 앞에 지정되므로 요소를 읽기 전에 첫 번째 또는 다음 메소드를 호출해야 합니다. *크기* 메소드를 사용하여 세트에 있는 요소 수를 판별할 수 있습니다.

결과 조회 세트의 요소는 활동 인스턴스 및 프로세스 인스턴스와 같은 작업 항목 및 연관된 참조 오브젝트의 선택된 속성으로 구성됩니다. QueryResultSet 요소의 첫 번째 속성(열)은 조회 요청의 Select 절에 지정된 첫 번째 속성 값을 지정합니다. QueryResultSet 요소의 두 번째 속성(열)은 조회 요청의 Select 절에 지정된 두 번째 속성 값을 지정합니다.

속성 유형과 호환되는 메소드를 호출하고 해당 열 색인을 지정하여 속성의 값을 검색할 수 있습니다. 열 색인의 번호는 1로 시작합니다.

속성 유형	메소드
String	getString
OID	getOID
Timestamp	getTimestamp getString getTimestampAsLong

속성 유형	메소드
Integer	getInteger getShort getLong getString getBoolean
Boolean	getBoolean getShort getInteger getLong getString
byte[]	getBinary

예제:

다음 조회가 실행됩니다.

```
QueryResultSet resultSet = process.query("ACTIVITY.STARTED,
                                         ACTIVITY.TEMPLATE_NAME AS NAME,
                                         WORK_ITEM.WIID, WORK_ITEM.REASON",
                                         (String)null, (String)null,
                                         (Integer)null, (TimeZone)null);
```

리턴된 결과 조회 세트에는 네 개의 열이 있습니다.

- 열 1: 시간 소인
- 열 2: 문자열
- 열 3: 오브젝트 ID
- 열 4: 정수

다음 메소드를 사용하여 속성 값을 검색할 수 있습니다.

```
while (resultSet.next())
{
 java.util.Calendar activityStarted = resultSet.getTimestamp(1);
 String templateName = resultSet.getString(2);
 WIID wiid = (WIID) resultSet.getOID(3);
 Integer reason = resultSet.getInteger(4);
}
```

결과 세트의 표시 이름을 인쇄된 테이블의 표제로 사용할 수 있습니다. 다음 이름은 보기의 열 이름 또는 조회의 AS 절로 정의된 이름입니다. 다음 메소드를 사용하여 예제의 표시 이름을 검색할 수 있습니다.

```
resultSet.getColumnDisplayName(1) returns "STARTED"
resultSet.getColumnDisplayName(2) returns "NAME"
resultSet.getColumnDisplayName(3) returns "WIID"
resultSet.getColumnDisplayName(4) returns "REASON"
```

사용자 특정 액세스 조건

사용자 특정 액세스 조건은 SQL SELECT 문이 API 조회에서 생성될 때 추가됩니다. 이 조건은 호출자가 지정한 조건을 충족시키며 호출자에게 권한이 부여된 오브젝트만이 호출자에게 리턴되도록 합니다.

추가되는 액세스 조건은 사용자가 시스템 관리자인지 여부에 따라 다릅니다.

시스템 관리자가 아닌 사용자가 호출한 조회

생성된 SQL WHERE 절은 API where 절을 사용자에게 특정한 액세스 제어 조건과 결합합니다. 조회는 사용자에게 액세스 권한이 부여된 오브젝트 즉, 사용자에게 작업 항목이 있는 오브젝트만을 검색합니다. 작업 항목은 타스크 또는 프로세스와 같은 비즈니스 오브젝트의 권한 역할에 사용자나 사용자 그룹을 지정하는 것을 나타냅니다. 예를 들어, John Smith가 제공된 타스크의 잠재적 소유자 역할 구성원이면 이 관계를 나타내는 작업 항목 오브젝트가 존재합니다.

시스템 관리자가 아닌 사용자가 타스크를 조회하는 경우 그룹 작업 항목이 사용 가능하지 않으면 다음 액세스 조건이 WHERE 절에 추가됩니다.

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'user'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

따라서 John Smith가 잠재적 소유자인 타스크의 목록을 가져오려는 경우 API where 절은 다음과 유사할 수 있습니다.

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

이 API 절로 인한 SQL 문의 액세스 조건은 다음과 같습니다.

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JohnSmith'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

이는 John Smith가 자신이 프로세스 독자이거나 프로세스 관리자이며 작업 항목이 없는 타스크 및 활동을 보려면, PROCESS_INSTANCE 보기로부터 특성을 조회의 select, where 또는 order-by 절에 추가해야 함을 의미합니다(예: PROCESS_INSTANCE.PIID).

그룹 작업 항목이 사용 가능하면 그룹에 액세스 권한이 있는 오브젝트에 사용자가 액세스할 수 있도록 추가 액세스 조건이 WHERE 절에 추가됩니다.

시스템 관리자가 호출한 조회

시스템 관리자는 query 메소드를 호출하여 작업 항목과 연관된 오브젝트를 검색할 수 있습니다. 이 경우 WORK_ITEM 보기와의 결합이 생성된 SQL 조회에 추가되지만 WORK_ITEM.OWNER_ID에 대한 액세스 제어 조건은 없습니다.

이러한 상황에서는 타스크에 대한 SQL 조회에 다음이 포함됩니다.

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
```

queryAll 조회

이 유형의 조회는 시스템 관리자나 시스템 모니터만이 호출할 수 있습니다. 액세스 제어 조건과 WORK_ITEM 보기와의 결합이 모두 추가되지 않습니다. 이 유형의 조회는 전체 오브젝트에 대한 데이터를 모두 리턴합니다.

query 및 queryAll 메소드의 예제

이 예제는 다양한 일반 API 조회 및 조회가 처리될 때 생성되는 연관된 SQL 문의 구문을 보여줍니다.

예제: 준비 상태의 타스크 조회:

이 예제는 query 메소드를 사용하여 로그온한 사용자가 작업할 수 있는 타스크를 검색하는 방법을 보여줍니다.

John Smith는 그에게 지정된 타스크의 목록을 확인하려고 합니다. 사용자가 타스크에 대해 작업하려면 타스크가 준비 상태에 있어야 합니다. 로그온한 사용자에게도 타스크에 대한 잠재적 소유자 작업 항목이 있어야 합니다. 다음 코드 스니펫은 이 조회에 대한 query 메소드 호출을 보여줍니다.

```
query( "DISTINCT TASK.TKIID",
           "TASK.KIND IN ( TASK.KIND.KIND_HUMAN,

    TASK.KIND.KIND_PARTICIPATING )
             AND " +
           "TASK.STATE = TASK.STATE.STATE_READY AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
            (String)null, (String)null, (Integer)null, (TimeZone)null )
```

다음 조치는 SQL SELECT 문이 생성될 때 수행됩니다.

- 액세스 제어의 조건이 where 절에 추가됩니다. 이 예제에서는 그룹 작업 항목이 사용 가능하지 않다고 가정합니다.
- TASK.STATE.STATE_READY와 같은 상수가 숫자 값으로 교체됩니다.
- FROM 절 및 조인 조건이 추가됩니다.

다음 코드 스니펫은 API 조회에서 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND    WI.REASON = 1
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID

   = null AND  WI.EVERYBODY = true )
```

API 조회를 sampleProcess와 같은 특정 프로세스에 대한 타스크로 제한하려는 경우 조회가 다음과 같이 됩니다.

```
query( "DISTINCT TASK.TKIID",
           "PROCESS_TEMPLATE.NAME = 'sampleProcess' AND "+
           "TASK.KIND IN ( TASK.KIND.KIND_HUMAN,

    TASK.KIND.KIND_PARTICIPATING )
             AND " +
           "TASK.STATE = TASK.STATE.STATE_READY AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
            (String)null, (String)null, (Integer)null, (TimeZone)null )
```

예제: 청구됨 상태의 타스크 조회:

이 예제는 query 메소드를 사용하여 로그온한 사용자가 청구한 타스크를 검색하는 방법을 보여줍니다.

사용자, John Smith는 자신이 청구했으며 여전히 청구됨 상태에 있는 타스크를 검색하려 합니다. "John Smith가 청구함"을 지정하는 조건은 `TASK.OWNER = 'JohnSmith'` 입니다. 다음 코드 스니펫은 조회에 대한 query 메소드 호출을 보여줍니다.

```
query( "DISTINCT TASK.TKIID",
           "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
           "TASK.OWNER = 'JohnSmith'",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

다음 코드 스니펫은 API 조회에서 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   TA.OWNER = 'JohnSmith'
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID

   = null AND WI.EVERYBODY = true )
```

타스크가 청구되면 타스크의 소유자에 대한 작업 항목이 작성됩니다. 따라서 John Smith가 청구한 타스크에 대한 조회를 구성하는 방식은 `TASK.OWNER = 'JohnSmith'`를 사용하는 대신 다음 조건을 조회에 추가하는 것입니다.

```
WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER
```

그 결과 조회가 다음 코드 스니펫과 같이 표시됩니다.

```
query( "DISTINCT TASK.TKIID",
           "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

다음 조치는 SQL SELECT 문이 생성될 때 수행됩니다.

- 액세스 제어의 조건이 where 절에 추가됩니다. 이 예제에서는 그룹 작업 항목이 사용 가능하지 않다고 가정합니다.
- TASK.STATE.STATE_READY와 같은 상수가 숫자 값으로 교체됩니다.
- FROM 절 및 조인 조건이 추가됩니다.

다음 코드 스니펫은 API 조회에서 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   AND    WI.REASON = 4
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID

   = null AND WI.EVERYBODY = true )
```

John은 휴가를 떠날 예정이므로 그의 팀장인 Anne Grant는 그의 현재 워크로드를 확인하고자 합니다. Anne에게는 시스템 관리자 권한이 있습니다. 그녀가 호출하는 조회는 John이 호출한 조회와 동일합니다. 그러나 Anne은 관리자이기 때문에 생성되는 SQL 문은 차이가 있습니다. 다음 코드 스니펫은 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  TA.TKIID = WI.OBJECT_ID =
   AND    TA.STATE = 8
   AND   TA.OWNER = 'JohnSmith')
```

Anne이 관리자이기 때문에 액세스 제어 조건이 WHERE 절에 추가되지 않습니다.

예제: 에스컬레이션 조회:

이 예제는 query 메소드를 사용하여 로그온한 사용자에 대한 에스컬레이션을 검색하는 방법을 보여줍니다.

타스크가 에스컬레이트되면 에스컬레이션 수신자 작업 항목이 작성됩니다. 사용자, Mary Jones는 그녀에게 에스컬레이트된 타스크의 목록을 확인하려고 합니다. 다음 코드 스니펫은 조회에 대한 query 메소드 호출을 보여줍니다.

```
query( "DISTINCT ESCALATION.ESIID, ESCALATION.TKIID",
         "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON

  _ESCALATION_RECEIVER",
           (String)null, (String)null, (Integer)null

    , (TimeZone)null )
```

다음 조치는 SQL SELECT 문이 생성될 때 수행됩니다.

- 액세스 제어의 조건이 where 절에 추가됩니다. 이 예제에서는 그룹 작업 항목이 사용 가능하지 않다고 가정합니다.
- TASK.STATE.STATE_READY와 같은 상수가 숫자 값으로 교체됩니다.
- FROM 절 및 조인 조건이 추가됩니다.

다음 코드 스니펫은 API 조회에서 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT ESCALATION.ESIID, ESCALATION.TKIID
  FROM   ESCALATION ESC, WORK_ITEM WI
  WHERE  ESC.ESIID = WI.OBJECT_ID
  AND    WI.REASON  = 10
  AND
 ( WI.OWNER_ID = 'MaryJones' OR WI.OWNER_ID

 = null AND  WI.EVERYBODY = true )
```

예제: queryAll 메소드 사용:

이 예제는 queryAll 메소드를 사용하여 프로세스 템플리트에 속한 모든 활동을 검색하는 방법을 보여줍니다.

queryAll 메소드는 시스템 관리자 또는 시스템 모니터 권한이 있는 사용자만이 사용할 수 있습니다. 다음 코드 스니펫은 프로세스 템플리트, sampleProcess에 속한 모든 활동을 검색하여, 조회에 대한 queryAll 메소드를 보여줍니다.

```
queryAll( "DISTINCT ACTIVITY.AIID",
       "PROCESS_TEMPLATE.NAME = 'sampleProcess'",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

다음 코드 스니펫은 API 조회에서 생성되는 SQL 조회를 보여줍니다.

```
SELECT DISTINCT ACTIVITY.AIID
  FROM   ACTIVITY AI, PROCESS_TEMPLATE PT
  WHERE  AI.PTID = PT.PTID
  AND    PT.NAME = 'sampleProcess'
```

호출은 관리자가 수행하므로 생성된 SQL 문에는 액세스 제어 조건이 추가되지 않습니다. WORK_ITEM 보기와의 결합도 추가되지 않습니다. 이는 조회가 작업 항목이 없는 활동을 포함하여 프로세스 템플리트에 대한 모든 활동을 검색함을 의미합니다.

예제: 조회에 조회 특성 포함:

이 예제는 query 메소드를 사용하여 비즈니스 프로세스에 속한 타스크를 검색하는 방법을 보여줍니다. 프로세스에는 프로세스에 대해 정의된, 검색에 포함시키려는 조회 특성이 있습니다.

예를 들어, 비즈니스 프로세스에 속하며 준비 상태에 있는 모든 휴먼 타스크를 검색하려 합니다. 프로세스에는 값이 CID_12345인 조회 특성, **customerID** 및 네임스페이스가 있습니다. 다음 코드 스니펫은 조회에 대한 query 메소드 호출을 보여줍니다.

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                  PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY.NAME = 'customerID' AND " +
          " QUERY_PROPERTY.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY.NAMESPACE =
             'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )

      AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

이제 제공된 네임스페이스와 함께 두 번째 조회 특성(예: **Priority**)을 조회에 추가하려는 경우 조회에 대한 query 메소드 호출은 다음과 같습니다.

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                  PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY1.NAME = 'customerID' AND " +
          " QUERY_PROPERTY1.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY1.NAMESPACE =
                 'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/'

    AND " +
          " QUERY_PROPERTY2.NAME = 'Priority' AND " +
          " QUERY_PROPERTY2.NAMESPACE =
                 'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/'

    AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

둘 이상의 조회 특성을 조회에 추가하는 경우에는 코드 스니펫에 표시된대로 각 특성에 번호를 지정해야 합니다. 그러나 사용자 정의 특성을 조회하면 성능에 영향이 미칩니다. 즉, 조회의 사용자 정의 특성 수와 함께 성능이 저하됩니다.

예제: 조회에 사용자 정의 특성 포함:

이 예제는 query 메소드를 사용하여 사용자 정의 특성이 있는 타스크를 검색하는 방법을 보여줍니다.

예를 들어, 값이 CID_12345인 사용자 정의 특성, **customerID**가 있는 준비 상태의 모든 휴먼 타스크를 검색하려 합니다. 다음 코드 스니펫은 조회에 대한 query 메소드 호출을 보여줍니다.

```
query ( " DISTINCT TASK.TKIID ",
          " TASK_CPROP.NAME = 'customerID' AND " +
          " TASK_CPROP.STRING_VALUE = 'CID_12345' AND " +
```

```
            " TASK.KIND IN
               ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )

       AND " +
            " TASK.STATE = TASK.STATE.STATE_READY ",
            (String)null, (String)null, (Integer)null, (TimeZone)null );
```

이제 타스크 및 사용자 정의 특성을 검색하려는 경우 조회에 대한 query 메소드 호출은 다음과 같습니다.

```
query ( " DISTINCT TASK.TKIID, TASK_CPROP.NAME, TASK_CPROP.STRING_VALUE",
            " TASK.KIND IN
               ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )

       AND " +
            " TASK.STATE = TASK.STATE.STATE_READY ",
            (String)null, (String)null, (Integer)null, (TimeZone)null );
```

다음 코드 스니펫은 API 조회에서 생성되는 SQL 문을 보여줍니다.

```
SELECT DISTINCT TA.TKIID , TACP.NAME , TACP.STRING_VALUE
   FROM  TASK TA LEFT JOIN TASK_CPROP TACP ON (TA.TKIID = TACP.TKIID),
         WORK_ITEM WI
   WHERE WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND  (WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID IS

   NULL AND WI.EVERYBODY = 1 )
```

이 SQL 문은 TASK 보기와 TASK_CPROP 보기 사이의 외부 결합을 포함합니다. 이는 사용자 정의 특성이 없는 경우에도 WHERE 절을 충족시키는 타스크가 검색됨을 의미합니다.

저장된 조회 관리

저장된 조회를 사용하여 자주 실행되는 조회를 저장할 수 있습니다. 저장된 조회는 모든 사용자가 사용할 수 있는 조회(공용 조회) 또는 특정 사용자에게 속하는 조회(개인용 조회)가 될 수 있습니다.

이 태스크 정보

저장된 조회는 데이터베이스에 저장되고 이름으로 식별되는 조회입니다. 개인용 및 공용 저장된 조회는 동일한 이름을 사용할 수 있습니다. 서로 다른 소유자의 개인용 저장된 조회 또한 동일한 이름을 사용할 수 있습니다.

비즈니스 프로세스 오브젝트, 타스크 오브젝트 또는 두 오브젝트 유형의 조합에 대한 조회를 저장할 수 있습니다.

관련 개념

『저장된 조회의 매개변수』
저장된 조회는 데이터베이스에 저장되고 이름으로 식별되는 조회입니다. 규정화된 튜플은 조회 실행 시 동적으로 어셈블됩니다. 저장된 조회를 재사용하려면 런타임 시 분석되는 조회 정의에 매개변수를 사용하면 됩니다.

저장된 조회의 매개변수:

저장된 조회는 데이터베이스에 저장되고 이름으로 식별되는 조회입니다. 규정화된 튜플은 조회 실행 시 동적으로 어셈블됩니다. 저장된 조회를 재사용하려면 런타임 시 분석되는 조회 정의에 매개변수를 사용하면 됩니다.

예를 들어, 사용자 정의 이름을 저장할 사용자 정의 특성을 정의했습니다. 특정 고객 ACME Co.와 연관된 타스크를 리턴할 조회를 정의할 수 있습니다. 이 정보를 조회하기 위한 조회의 where 절은 다음 예제와 유사합니다.

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE

  = 'ACME Co.'";
```

BCME Ltd 고객을 또한 검색할 수 있도록 이 조회를 재사용하려면, 사용자 정의 특성의 값에 매개변수를 사용할 수 있습니다. 타스크 조회에 매개변수를 추가할 경우, 다음 예제와 유사할 수도 있습니다.

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE

  = '@param1'";
```

`@param1` 매개변수는 query 메소드에 전달되는 매개변수 목록에서 런타임시 해석됩니다. 다음 규칙은 조회의 매개변수 사용에 적용됩니다.

- 매개변수는 where 절에서만 사용할 수 있습니다.
- 매개변수는 문자열입니다.
- 매개변수는 문자열 대체를 사용하여 런타임 시 대체됩니다. 특수 문자가 필요할 경우, where절에 지정하거나 또는 매개변수의 일부로 런타임 시 전달해야 합니다.
- 매개변수 이름은 `@param` 문자열과 정수의 결합으로 구성됩니다. 가장 작은 숫자는 1이며, 런타임 시 조회에 전달되는 매개변수 목록의 첫 번째 항목을 가리킵니다.
- 매개변수는 where 절 내에서 여러 번 사용할 수 있습니다. 매개변수를 사용한 전체 횟수는 그와 동일한 값으로 대체됩니다.

관련 태스크

423 페이지의『저장된 조회 관리』
저장된 조회를 사용하여 자주 실행되는 조회를 저장할 수 있습니다. 저장된 조회는 모든 사용자가 사용할 수 있는 조회(공용 조회) 또는 특정 사용자에게 속하는 조회(개인용 조회)가 될 수 있습니다.

공용 저장된 조회 관리:

공용 저장된 조회는 시스템 관리자가 작성합니다. 해당 조회는 모든 사용자에게 사용 가능합니다.

이 태스크 정보

시스템 관리자는 공용 저장된 조회를 작성, 보기 및 삭제할 수 있습니다. API 호출에서 사용자 ID를 지정하지 않으면 저장된 조회는 공용 저장된 조회로 간주됩니다.

프로시저

1. 공용 저장된 조회를 작성하십시오.

   예를 들어, 다음 코드 스니펫은 프로세스 인스턴스에 대한 저장된 조회를 작성하고 이를 CustomerOrdersStartingWithA 이름으로 저장합니다.

   ```
   process.createStoredQuery("CustomerOrdersStartingWithA",
               "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
               "PROCESS_INSTANCE.NAME LIKE 'A%'",
               "PROCESS_INSTANCE.NAME",
                (Integer)null, (TimeZone)null);
   ```

   저장된 조회의 결과는 문자 A로 시작하는 모든 프로세스 인스턴스 이름 및 관련 프로세스 인스턴스 ID(PIID)의 저장된 목록입니다.
2. 저장된 조회에서 정의한 조회를 실행하십시오.

   ```
   QueryResultSet result = process.query("CustomerOrdersStartingWithA",
                 new Integer(0), null);
   ```

   이 조치를 실행하면 기준에 맞는 오브젝트가 리턴됩니다. 이 경우 A로 시작하는 고객의 모든 주문이 리턴됩니다.
3. 사용 가능한 공용 저장된 조회의 이름을 표시하십시오.

   다음 코드 스니펫은 리턴되는 조회의 목록을 공용 조회로 제한하는 방법을 표시합니다.

   ```
   String[] storedQuery = process.getStoredQueryNames(StoredQueryData.KIND_PUBLIC);
   ```
4. 옵션: 특정 저장된 조회에서 정의한 조회를 확인하십시오.

   저장된 개인용 조회는 저장된 공용 조회와 동일한 이름을 사용할 수 있습니다. 해당 이름이 동일할 경우, 개인용 저장된 조회가 리턴됩니다. 다음 코드 스니펫은 지

정된 이름을 가진 공용 조회만을 리턴하는 방법을 표시합니다. 저장된 조회를 검색하기 위해 휴먼 타스크 관리자 API를 사용하는 경우, StoredQueryData 대신 리턴되는 오브젝트에 대해 StoredQuery를 사용하십시오.

```
StoredQueryData storedQuery = process.getStoredQuery
   (StoredQueryData.KIND_PUBLIC, "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();String owner = storedQuery.getOwner();
```

5. 공용 저장된 조회를 삭제하십시오.

   다음 코드 스니펫은 1단계에서 작성한 저장된 조회를 삭제하는 방법을 표시합니다.

   ```
   process.deleteStoredQuery("CustomerOrdersStartingWithA");
   ```

기타 사용자에 대한 개인용 저장된 조회 관리:

모든 사용자가 개인용 조회를 작성할 수 있습니다. 해당 조회는 조회 소유자 및 시스템 관리자에게만 사용 가능합니다.

이 태스크 정보

시스템 관리자의 경우 특정 사용자에게 속하는 개인용 저장된 조회를 관리할 수 있습니다.

프로시저

1. 사용자 ID Smith에 대한 개인용 저장된 조회를 작성하십시오.

   예를 들어, 다음 코드 스니펫은 프로세스 인스턴스에 대한 저장된 조회를 작성하고 이를 CustomerOrdersStartingWithA로 사용자 ID Smith에 저장합니다.

   ```
   process.createStoredQuery("Smith", "CustomerOrdersStartingWithA",
                "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
                "PROCESS_INSTANCE.NAME LIKE 'A%'",
                "PROCESS_INSTANCE.NAME",
                 (Integer)null, (TimeZone)null,
                 (List)null, (String)null);
   ```

   저장된 조회의 결과는 문자 A로 시작하는 모든 프로세스 인스턴스 이름 및 관련 프로세스 인스턴스 ID(PIID)의 저장된 목록입니다.

2. 저장된 조회에서 정의한 조회를 실행하십시오.

   ```
   QueryResultSet result = process.query
                         ("Smith", "CustomerOrdersStartingWithA",
                           (Integer)null, (Integer)null, (List)null);
   new Integer(0));
   ```

   이 조치를 실행하면 기준에 맞는 오브젝트가 리턴됩니다. 이 경우 A로 시작하는 고객의 모든 주문이 리턴됩니다.

3. 특정 사용자에게 속하는 개인용 조회의 이름 목록을 가져오십시오.

예를 들어, 다음 코드 스니펫은 사용자 Smith에게 속한 개인용 조회 목록을 가져오는 방법을 표시합니다.

```
String[] storedQuery = process.getStoredQueryNames("Smith");
```

4. 특정 조회의 세부사항을 보십시오.

   다음 코드 스니펫은 사용자 Smith가 소유한 CustomerOrdersStartingWithA 조회의 세부사항을 보는 방법을 표시합니다.

```
StoredQueryData storedQuery = process.getStoredQuery
   ("Smith", "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();String owner = storedQuery.getOwner();
```

   저장된 조회를 검색하기 위해 휴먼 타스크 관리자 API를 사용하는 경우, StoredQueryData 대신 리턴되는 오브젝트에 대해 StoredQuery를 사용하십시오.

5. 개인용 저장된 조회를 삭제하십시오.

   다음 코드 스니펫은 사용자 Smith가 소유한 개인용 조회를 삭제하는 방법을 표시합니다.

```
process.deleteStoredQuery("Smith", "CustomerOrdersStartingWithA");
```

개인용 저장된 조회에 대한 작업:

시스템 관리자가 아닐 경우, 자신의 개인용 저장된 조회를 작성, 실행 및 삭제할 수 있습니다. 또한 시스템 관리자가 작성한 공용 저장된 조회를 사용할 수 있습니다.

프로시저

1. 개인용 저장된 조회를 작성하십시오.

   예를 들어, 다음 코드 스니펫은 프로세스 인스턴스에 대한 저장된 조회를 작성하고 이를 특정 이름으로 저장합니다. 사용자 ID를 지정하지 않으면 저장된 조회는 로그온 사용자에 대한 개인용 저장된 조회로 간주됩니다.

```
process.createStoredQuery("CustomerOrdersStartingWithA",
             "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
             "PROCESS_INSTANCE.NAME LIKE 'A%'",
             "PROCESS_INSTANCE.NAME",
              (Integer)null, (TimeZone)null);
```

   이 조회는 문자 A 및 연관된 프로세스 인스턴스 ID(PIID)로 시작하는 모든 프로세스 인스턴스 이름의 정렬된 목록을 리턴합니다.

2. 저장된 조회에서 정의한 조회를 실행하십시오.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",
               new Integer(0));
```

이 조치를 실행하면 기준에 맞는 오브젝트가 리턴됩니다. 이 경우 A로 시작하는 고객의 모든 주문이 리턴됩니다.

3. 로그온한 사용자가 액세스할 수 있는 저장된 조회의 이름 목록을 가져오십시오.

   다음 코드 스니펫은 사용자가 액세스할 수 있는 공용 및 개인용 저장된 조회를 모두 가져오는 방법을 표시합니다.

   ```
   String[] storedQuery = process.getStoredQueryNames();
   ```

4. 특정 조회의 세부사항을 보십시오.

   다음 코드 스니펫은 사용자 Smith가 소유한 CustomerOrdersStartingWithA 조회의 세부사항을 보는 방법을 표시합니다.

   ```
   StoredQueryData storedQuery = process.getStoredQuery
      ("CustomerOrdersStartingWithA");
   String selectClause = storedQuery.getSelectClause();
   String whereClause = storedQuery.getWhereClause();
   String orderByClause = storedQuery.getOrderByClause();
   Integer threshold = storedQuery.getThreshold();String owner = storedQuery.getOwner();
   ```

   저장된 조회를 검색하기 위해 휴먼 타스크 관리자 API를 사용하는 경우, StoredQueryData 대신 리턴되는 오브젝트에 대해 StoredQuery를 사용하십시오.

5. 개인용 저장된 조회를 삭제하십시오.

   다음 코드 스니펫은 개인용 저장된 조회를 삭제하는 방법을 보여줍니다.

   ```
   process.deleteStoredQuery("CustomerOrdersStartingWithA");
   ```

비즈니스 프로세스 및 휴먼 타스크용 EJB 클라이언트 응용프로그램 개발

EJB API는 WebSphere Process Server에 설치된 비즈니스 프로세스 및 휴먼 타스크로 작업하는 EJB 클라이언트 응용프로그램을 개발하기 위한 일반 메소드 세트를 제공합니다.

이 태스크 정보

EJB(Enterprise JavaBeans) API로 다음을 수행할 클라이언트 응용프로그램을 작성할 수 있습니다.

- 프로세스 및 타스크의 시작 및 완료 시 삭제에 이르기까지 라이프사이클 관리
- 활동 및 프로세스 복구
- 작업 그룹 구성원을 통한 워크로드 관리 및 분배

EJB API는 다음과 같은 두 가지의 Stateless 세션 Enterprise Bean으로 제공됩니다.

- BusinessFlowManagerService 인터페이스는 비즈니스 프로세스 응용프로그램용 메소드를 제공합니다.
- HumanTaskManagerService 인터페이스는 타스크 기반 응용프로그램용 메소드를 제공합니다.

EJB API에 대한 자세한 정보는 com.ibm.bpe.api 패키지 및 com.ibm.bpe.task 패키지에서 Javadoc을 참조하십시오.

다음 단계는 EJB 클라이언트 응용프로그램을 개발하는 데 필요한 조치에 대한 개요를 제공합니다.

프로시저

1. 응용프로그램이 제공할 기능을 결정하십시오.
2. 사용할 세션 Bean을 결정하십시오.

   응용프로그램으로 구현하려는 시나리오에 따라 세션 Bean 중 하나 또는 모두를 사용할 수 있습니다.
3. 응용프로그램 사용자에게 필요한 권한을 결정하십시오.

   응용프로그램에 포함된 메소드를 호출하고 이들 메소드가 리턴하는 오브젝트 및 오브젝트 속성을 볼 수 있는 올바른 권한 역할을 응용프로그램 사용자에게 지정해야 합니다. 해당 세션 Bean의 인스턴스를 작성할 때 WebSphere Application Server는 컨텍스트를 인스턴스와 연관시킵니다. 컨텍스트에는 호출자의 프린시펄 ID, 그룹 멤버십 목록 및 역할에 대한 정보가 들어 있습니다. 이 정보는 각 호출에 대해 호출자의 권한을 확인하는 데 사용됩니다.

   Javadoc은 각 메소드에 대한 승인 정보를 포함합니다.
4. 응용프로그램을 표현하는 방법을 결정하십시오.

   EJB API는 로컬 또는 원격으로 호출됩니다.
5. 응용프로그램을 개발하십시오.
   a. EJB API에 액세스하십시오.
   b. EJB API를 사용하여 프로세스 또는 타스크와 상호작용하십시오.
      - 데이터를 조회하십시오.
      - 데이터에 대해 작업하십시오.

관련 개념

관리 권한 대체 모드

관련 참조

463 페이지의『BusinessFlowManagerService 인터페이스』
BusinessFlowManagerService 인터페이스는 클라이언트 응용프로그램에 피호출되는 비즈니스 프로세스 기능을 표시합니다.

485 페이지의『HumanTaskManagerService 인터페이스』
HumanTaskManagerService 인터페이스는 로컬 또는 원격 클라이언트가 호출할 수 있는 타스크 관련 기능을 표시합니다.

EJB API에 액세스

EJB(Enterprise JavaBeans) API는 두 가지 Stateless 세션 엔터프라이즈 Bean으로 제공됩니다. 비즈니스 프로세스 응용프로그램 및 타스크 응용프로그램은 Bean의 홈 인터페이스를 통해 해당하는 세션 엔터프라이즈 Bean에 액세스합니다.

이 태스크 정보

BusinessFlowManagerService 인터페이스는 비즈니스 프로세스 응용프로그램용 메소드를 제공하고 HumanTaskManagerService 인터페이스는 타스크 기반 응용프로그램용 메소드를 제공합니다. 응용프로그램은 임의의 Java 응용프로그램(다른 EJB(Enterprise JavaBeans) 응용프로그램 포함)이 될 수 있습니다.

세션 Bean의 원격 인터페이스 액세스

비즈니스 프로세스 또는 휴먼 타스크의 EJB 클라이언트 응용프로그램은 Bean의 원격 홈 인터페이스를 통해 세션 Bean의 원격 인터페이스에 액세스합니다.

이 태스크 정보

세션 Bean은 프로세스 응용프로그램에 대해서는 BusinessFlowManager 세션 Bean 또는 타스크 응용프로그램에 대해서는 HumanTaskManager 세션 Bean일 수 있습니다.

프로시저

1. 세션 Bean의 원격 인터페이스에 대한 참조를 응용프로그램 전개 설명자에 추가하십시오. 다음 파일 중 하나에 참조를 추가하십시오.
   - Java EE(Java Platform, Enterprise Edition) 클라이언트 응용프로그램의 경우 `application-client.xml` 파일
   - 웹 응용프로그램의 경우, `web.xml` 파일
   - EJB(Enterprise JavaBeans) 응용프로그램의 경우, `ejb-jar.xml` 파일

   프로세스 응용프로그램의 원격 홈 인터페이스에 대한 참조가 다음 예에 표시됩니다.

```
<ejb-ref>
 <ejb-ref-name>ejb/BusinessFlowManagerHome</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
 <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
```

타스크 응용프로그램의 원격 홈 인터페이스에 대한 참조가 다음 예에 표시됩니다.

```
<ejb-ref>
 <ejb-ref-name>ejb/HumanTaskManagerHome</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>com.ibm.task.api.HumanTaskManagerHome</home>
 <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
```

WebSphere Integration Developer를 사용하여 EJB 참조를 전개 설명자에 추가하는 경우 EJB 참조의 바인딩은 응용프로그램이 전개될 때 자동으로 작성됩니다. EJB 참조 추가에 대한 자세한 내용은 WebSphere Integration Developer 문서를 참조하십시오.

2. 비즈니스 오브젝트의 정의를 제공하려는 방법을 결정하십시오.

원격 클라이언트 응용프로그램에서 비즈니스 오브젝트에 대해 작업하려면 프로세스 또는 타스크와 상호작용하는 데 사용되는 비즈니스 오브젝트의 해당 스키마(XSD 또는 WSDL 파일)에 대한 액세스가 있어야 합니다. 이러한 파일에 대한 액세스는 다음 방법 중 하나로 제공될 수 있습니다.

- 클라이언트 응용프로그램이 Java EE 관리 환경에서 실행되지 않는 경우, 클라이언트 응용프로그램의 EAR 파일로 파일을 패키징하십시오.
- 클라이언트 응용프로그램이 관리 Java EE 환경의 웹 응용프로그램 또는 EJB 클라이언트인 경우, 클라이언트 응용프로그램의 EAR 파일로 파일을 패키징하거나 원격 아티팩트 로딩을 활용하십시오.
  a. Business Process Choreographer EJB API createMessage 및 ClientObjectWrapper.getObject 메소드를 사용하여 서버의 해당 응용프로그램에서 원격 비즈니스 오브젝트 정의를 확실하게 로드하십시오.
  b. 서비스 데이터 오브젝트 프로그래밍 API를 사용하여 비즈니스 오브젝트를 이미 인스턴스화된 비즈니스 오브젝트의 일부로 작성하거나 읽으십시오. DataObject 인스턴스에 대해 commonj.sdo.DataObject.createDataObject 또는 getDataObject 메소드를 사용하여 이를 수행하십시오.
  c. XML 스키마 any 또는 anyType을 사용하여 유형이 지정된 비즈니스 오브젝트의 특성에 대한 값으로서 비즈니스 오브젝트를 작성하려는 경우, 비즈니스 오브젝트 서비스를 사용하여 비즈니스 오브젝트를 작성하거나 읽으십시오. 이를 수행하려면 스키마가 로드될 응용프로그램을 지시하도록 원격 아티팩트 로더 컨텍스트를 설정해야 합니다. 그런 다음, 해당 비즈니스 오브젝트 서비스를 사용할 수 있습니다.

예를 들어, 비즈니스 오브젝트를 작성하십시오. 여기서, "ApplicationName"은 비즈니스 오브젝트 정의를 포함하는 응용프로그램의 이름입니다.

```
BOFactory bofactory = (BOFactory) new
   ServiceManager().locateService

   ("com/ibm/websphere/bo/BOFactory");

com.ibm.wsspi.al.ALContext.setContext
   ("RALTemplateName", "ApplicationName");
try {
     DataObject dataObject = bofactory.create("uriName",

     "typeName" );
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

예를 들어, XML 입력을 읽으십시오. 이때, "ApplicationName"은 비즈니스 오브젝트 정의가 포함된 응용프로그램의 이름입니다.

```
BOXMLSerializer serializerService =
  (BOXMLSerializer) new ServiceManager().locateService
    ("com/ibm/websphere/bo/BOXMLSerializer");
ByteArrayInputStream input = new ByteArrayInputStream(<?xml?>..);

com.ibm.wsspi.al.ALContext.setContext
          ("RALTemplateName", "ApplicationName");
try {
     BOXMLDocument document = serializerService

     .readXMLDocument(input);
     DataObject dataObject = document.getDataObject();
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

3. JNDI(Java Naming and Directory Interface)를 통해 세션 Bean의 원격 홈 인터페이스를 찾으십시오.

다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the BusinessFlowManager bean
Object result =
       initialContext.lookup

       ("java:comp/env/ejb/BusinessFlowManagerHome");

// Convert the lookup result to the proper type
   BusinessFlowManagerHome processHome =
           (BusinessFlowManagerHome)javax.rmi

    .PortableRemoteObject.narrow
           (result,BusinessFlowManagerHome.class);
```

세션 Bean의 원격 홈 인터페이스는 EJB 오브젝트에 대한 작성 메소드를 포함합니다. 이 메소드는 세션 Bean의 원격 인터페이스를 리턴합니다.

4. 세션 Bean의 원격 인터페이스에 액세스하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   BusinessFlowManager process = processHome.create();
   ```

   세션 Bean에 액세스할 수 있다고 해서 호출자가 프로세스에 대한 모든 조치를 수행할 수 있는 것입니다. 호출자는 또한 해당 조치를 수행할 수 있는 권한을 가지고 있어야 합니다. 세션 Bean의 인스턴스를 작성할 때 컨텍스트는 세션 Bean의 인스턴스와 연관됩니다. 컨텍스트에는 호출자의 프린시펄 ID와 그룹 멤버십 목록이 포함되며, 호출자에게 Business Process Choreographer Java EE 역할 중 하나가 있는지 여부가 표시됩니다. 관리 보안을 설정하지 않은 경우라도 컨텍스트를 사용하여 각 호출에 대한 호출자의 권한을 확인할 수 있습니다. 관리 보안을 설정하지 않은 경우, 호출자의 프린시펄 ID는 UNAUTHENTICATED 값을 가집니다.

5. 서비스 인터페이스에서 표시한 비즈니스 함수를 호출하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   process.initiate("MyProcessModel",input);
   ```

   응용프로그램의 호출이 트랜잭션으로 실행됩니다. 트랜잭션은 다음 방법 중 하나로 설정 및 종료됩니다.

   - WebSphere Application Server를 사용하여 자동으로 설정 및 종료(전개 설명자가 TX_REQUIRED 지정)
   - 응용프로그램을 사용하여 명시적으로 설정 및 종료. 응용프로그램 호출을 하나의 트랜잭션으로 번들화할 수 있습니다.

     ```
     // Obtain user transaction interface
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup

            ("java:comp/UserTransaction");

     // Begin a transaction
     transaction.begin();

     // Applications calls ...

     // On successful return, commit the transaction
     transaction.commit();
     ```

   팁: 데이터베이스 잠금 충돌을 차단하려면 다음과 유사한 명령문이 동시에 실행되지 않도록 하십시오.

   ```
   // Obtain user transaction interface
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup
   ```

```
        ("java:comp/UserTransaction");

transaction.begin();

//read the activity instance
process.getActivityInstance(aiid);
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

getActivityInstance 메소드 및 기타 읽기 조작은 읽기 잠금을 설정합니다. 이 예제에서는 활동 인스턴스의 읽기 잠금이 활동 인스턴스의 갱신 잠금으로 업그레이드됩니다. 이로 인해 이 트랜잭션이 동시에 실행될 때 데이터베이스 교착 상태가 초래될 수 있습니다.

예제

다음 예제에서는 3단계에서 5단계까지를 통해 타스크 응용프로그램을 찾는 방법을 보여줍니다.

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the HumanTaskManager bean
Object result =
         initialContext.lookup

  ("java:comp/env/ejb/HumanTaskManagerHome");

// Convert the lookup result to the proper type
HumanTaskManagerHome taskHome =
           (HumanTaskManagerHome)javax.rmi

    .PortableRemoteObject.narrow
           (result,HumanTaskManagerHome.class);

...
//Access the remote interface of the session bean.
HumanTaskManager task = taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

세션 Bean의 로컬 인터페이스 액세스

비즈니스 프로세스 또는 휴먼 타스크의 EJB 클라이언트 응용프로그램은 Bean의 로컬 홈 인터페이스를 통해 세션 Bean의 로컬 인터페이스에 액세스합니다.

이 태스크 정보

세션 Bean은 프로세스 응용프로그램에 대해서는 BusinessFlowManager 세션 Bean 또는 휴먼 타스크 응용프로그램에 대해서는 HumanTaskManager 세션 Bean일 수 있습니다.

프로시저

1. 세션 Bean의 로컬 인터페이스에 대한 참조를 응용프로그램 전개 설명자에 추가하십시오. 다음 파일 중 하나에 참조를 추가하십시오.
   - Java EE(Java Platform, Enterprise Edition) 클라이언트 응용프로그램의 경우 `application-client.xml` 파일
   - 웹 응용프로그램의 경우, `web.xml` 파일
   - EJB(Enterprise JavaBeans) 응용프로그램의 경우, `ejb-jar.xml` 파일

   프로세스 응용프로그램의 로컬 홈 인터페이스에 대한 참조가 다음 예에 표시됩니다.

   ```
   <ejb-local-ref>
    <ejb-ref-name>ejb/LocalBusinessFlowManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
    <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
   </ejb-local-ref>
   ```

   타스크 응용프로그램의 로컬 홈 인터페이스에 대한 참조가 다음 예에 표시됩니다.

   ```
   <ejb-local-ref>
    <ejb-ref-name>ejb/LocalHumanTaskManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
    <local>com.ibm.task.api.LocalHumanTaskManager</local>
   </ejb-local-ref>
   ```

   WebSphere Integration Developer를 사용하여 EJB 참조를 전개 설명자에 추가하는 경우 EJB 참조의 바인딩은 응용프로그램이 전개될 때 자동으로 작성됩니다. EJB 참조 추가에 대한 자세한 내용은 WebSphere Integration Developer 문서를 참조하십시오.

2. JNDI(Java Naming and Directory Interface)를 통해 세션 Bean의 로컬 홈 인터페이스를 찾으십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   // Obtain the default initial JNDI context
   InitialContext initialContext = new InitialContext();

   // Lookup the local home interface of the BusinessFlowManager bean

    LocalBusinessFlowManagerHome processHome =
         (LocalBusinessFlowManagerHome)initialContext.lookup
         ("java:comp/env/ejb/LocalBusinessFlowManagerHome");
   ```

세션 Bean의 로컬 홈 인터페이스는 EJB 오브젝트에 대한 작성 메소드를 포함합니다. 이 메소드는 세션 Bean의 로컬 인터페이스를 리턴합니다.

3. 세션 Bean의 로컬 인터페이스에 액세스하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   LocalBusinessFlowManager process = processHome.create();
   ```

   세션 Bean에 액세스할 수 있다고 해서 호출자가 프로세스에 대한 모든 조치를 수행할 수 있는 것입니다. 호출자는 또한 해당 조치를 수행할 수 있는 권한을 가지고 있어야 합니다. 세션 Bean의 인스턴스를 작성할 때 컨텍스트는 세션 Bean의 인스턴스와 연관됩니다. 컨텍스트에는 호출자의 프린시펄 ID와 그룹 멤버십 목록이 포함되며, 호출자에게 Business Process Choreographer Java EE 역할 중 하나가 있는지 여부가 표시됩니다. 관리 보안을 설정하지 않은 경우라도 컨텍스트를 사용하여 각 호출에 대한 호출자의 권한을 확인할 수 있습니다. 관리 보안을 설정하지 않은 경우, 호출자의 프린시펄 ID는 UNAUTHENTICATED 값을 가집니다.

4. 서비스 인터페이스에서 표시한 비즈니스 함수를 호출하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   process.initiate("MyProcessModel",input);
   ```

   응용프로그램의 호출이 트랜잭션으로 실행됩니다. 트랜잭션은 다음 방법 중 하나로 설정 및 종료됩니다.

   - WebSphere Application Server를 사용하여 자동으로 설정 및 종료(전개 설명자가 TX_REQUIRED 지정)
   - 응용프로그램을 사용하여 명시적으로 설정 및 종료. 응용프로그램 호출을 하나의 트랜잭션으로 번들화할 수 있습니다.

     ```
     // Obtain user transaction interface
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

     // Begin a transaction
     transaction.begin();

     // Applications calls ...

     // On successful return, commit the transaction
     transaction.commit();
     ```

   팁: 데이터베이스 교착 상태를 차단하려면 다음과 유사한 명령문이 동시에 실행되지 않도록 하십시오.

   ```
   // Obtain user transaction interface
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

   transaction.begin();
   ```

```
//read the activity instance
process.getActivityInstance(aiid);
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

getActivityInstance 메소드 및 기타 읽기 조작은 읽기 잠금을 설정합니다. 이 예제에서는 활동 인스턴스의 읽기 잠금이 활동 인스턴스의 갱신 잠금으로 업그레이드됩니다. 이로 인해 이 트랜잭션이 동시에 실행될 때 데이터베이스 교착 상태가 초래될 수 있습니다.

예제

다음 예제에서는 2단계에서 4단계까지를 통해 타스크 응용프로그램을 찾는 방법을 보여줍니다.

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the local home interface of the HumanTaskManager bean
LocalHumanTaskManagerHome taskHome =
        (LocalHumanTaskManagerHome)initialContext.lookup
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Access the local interface of the session bean
LocalHumanTaskManager task = taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

비즈니스 프로세스용 응용프로그램 개발

비즈니스 프로세스는 비즈니스 목표를 달성하기 위해 특정 순서로 호출되는 비즈니스 관련 활동 세트입니다. 예제에서는 프로세스에 대한 일반적인 조치를 수행하는 응용프로그램의 개발 방법을 설명합니다.

이 태스크 정보

비즈니스 프로세스는 마이크로플로우 또는 장기 실행 프로세스가 될 수 있습니다.

- 마이크로플로우는 동기적으로 실행되는 단기 실행 비즈니스 프로세스입니다. 짧은 시간 후 결과가 호출자에게 리턴됩니다.
- 장기 실행, 가로채기 가능 프로세스가 함께 체인으로 연결된 일련의 활동으로 실행됩니다. 프로세스에 특정 구성을 사용하면 프로세스 플로우에 인터럽트가 발생합니다(예: 휴먼 타스크 호출, 동기화 바인딩을 사용한 서비스 호출 또는 타이머 구동 활동 사용).

일반적으로 프로세스의 병렬 분기는 비동기적으로 탐색합니다. 즉, 병렬 분기의 활동은 동시에 실행됩니다. 활동 유형 및 트랜잭션 설정에 따라 활동은 자체 고유 트랜잭션으로 실행될 수 있습니다.

프로세스 인스턴스의 조치에 필요한 역할

LocalBusinessFlowManager 인터페이스에 액세스할 수 있다고 해서 호출자가 프로세스에 대한 모든 조치를 수행할 수 있는 것은 아닙니다. 호출자는 조치를 수행할 수 있는 권한이 있는 역할로 클라이언트 응용프로그램을 로그온해야 합니다.

다음 표에는 특정 역할이 프로세스 인스턴스에 대해 수행할 수 있는 조치가 표시됩니다.

조치	호출자의 프린시펄 역할		
	독자	시작자	관리자
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspend			x
transferWorkItem			x

주: 프로세스 관리를 시스템 관리자로 제한하면 인스턴스 기반 관리를 사용할 수 없습니다. 즉, 프로세스, 범위 및 활동에 대한 관리 조치는 BPESystemAdministrator 역할의 사용자로 제한됩니다. 또한 프로세스 인스턴스 또는 그 일부에 대한 읽기, 보기 및 모니터링은 BPESystemAdministrator 또는 BPESystemMonitor 역할의 사용자만 수행할 수 있습니다. 이 관리 모드에 대한 자세한 정보는 doc/bpc/cnot_instance_based_admin.dita의 내용을 참조하십시오.

비즈니스 프로세스 활동의 조치에 필요한 역할

LocalBusinessFlowManager 인터페이스에 액세스할 수 있다고 해서 호출자가 활동에 대한 모든 조치를 수행할 수 있는 것은 아닙니다. 호출자는 조치를 수행할 수 있는 권한이 있는 역할로 클라이언트 응용프로그램을 로그온해야 합니다.

다음 표에는 특정 역할이 활동 인스턴스에 대해 수행할 수 있는 조치가 표시됩니다.

조치	호출자의 프린시펄 역할				
	독자	편집자	잠재적 소유자	소유자	관리자
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x
getCustomProperties	x	x	x	x	x
getCustomProperty	x	x	x	x	x
getCustomPropertyNames	x	x	x	x	x
getFaultMessage	x	x	x	x	x
getFaultNames	x	x	x	x	x
getInputMessage	x	x	x	x	x
getOutputMessage	x	x	x	x	x
getVariable	x	x	x	x	x
getVariableNames	x	x	x	x	x
getInputVariableNames	x	x	x	x	x
getOutputVariableNames	x	x	x	x	x
getWorkItems	x	x	x	x	x
setCustomProperty		x		x	x
setFaultMessage		x		x	x
setOutputMessage		x		x	x

조치	호출자의 프린시펄 역할				
	독자	편집자	잠재적 소유자	소유자	관리자
setVariable					x
transferWorkItem				x (잠재적 소유자 또는 관리자 전용)	x

주: 프로세스 관리를 시스템 관리자로 제한하면 인스턴스 기반 관리를 사용할 수 없습니다. 즉, 프로세스, 범위 및 활동에 대한 관리 조치는 BPESystemAdministrator 역할의 사용자로 제한됩니다. 또한 프로세스 인스턴스 또는 그 일부에 대한 읽기, 보기 및 모니터링은 BPESystemAdministrator 또는 BPESystemMonitor 역할의 사용자만 수행할 수 있습니다. 이 관리 모드에 대한 자세한 정보는 doc/bpc/cnot_instance_based_admin.dita의 내용을 참조하십시오.

비즈니스 프로세스 라이프사이클 관리

프로세스 인스턴스는 프로세스를 시작할 수 있는 Business Process Choreographer API 메소드가 호출된 경우에 생성됩니다. 프로세스 인스턴스의 탐색은 모든 활동이 종료 상태가 될 때까지 계속됩니다. 다양한 조치를 수행하여 프로세스 인스턴스의 라이프사이클을 관리할 수 있습니다.

이 태스크 정보

예제에서는 프로세스에 대한 다음 일반적인 라이프사이클 조치를 수행하는 응용프로그램의 개발 방법을 설명합니다.

비즈니스 프로세스 시작:

비즈니스 프로세스가 시작되는 방법은 프로세스가 마이크로플로우인지 또는 장기 실행 프로세스인지에 따라 달라집니다. 프로세스를 시작하는 서비스는 프로세스 시작 방법에 중요하게 작용합니다. 프로세스는 고유한 시작 서비스를 포함하거나 여러 시작 서비스를 포함할 수 있습니다.

이 태스크 정보

예제에서는 시작 마이크로플로우 및 장기 실행 프로세스의 일반적인 시나리오에 대한 응용프로그램을 개발하는 방법을 보여줍니다.

고유 시작 서비스를 포함하는 마이크로플로우 실행:

마이크로플로우는 수신 활동 또는 선택 활동으로 시작됩니다. 마이크로플로우가 수신 활동으로 시작되거나 선택 활동에 하나의 onMessage 정의만 있는 경우 시작 서비스는 고유합니다.

이 태스크 정보

마이크로 플로우가 요청-응답 조작을 구현하는 경우, 즉, 프로세스에 응답이 포함된 경우 호출 메소드를 사용하여 프로세스 템플리트 이름을 셀의 매개변수로서 전달하는 프로세스를 실행할 수 있습니다.

마이크로플로우가 단방향 조작이면 sendMessage 메소드를 사용하여 프로세스를 실행하십시오. 이 예에서는 이 메소드를 다루지 않습니다.

프로시저

1. 옵션: 프로세스 템플리트를 표시하여 실행하려는 프로세스 이름을 찾으십시오.

   프로세스의 이름을 이미 아는 경우 이 단계는 선택적입니다.

   ```
   ProcessTemplateData[] processTemplates
   = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   결과는 이름별로 정렬됩니다. 조회는 호출 메소드로 시작할 수 있는 처음 50개의 정렬된 템플리트를 포함하는 배열을 리턴합니다.

2. 해당 유형의 입력 메시지로 프로세스를 시작하십시오.

   메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다.

   ```
   ProcessTemplateData template = processTemplates[0];
   //create a message for the single starting receive activity
   ClientObjectWrapper input = process.createMessage
                              (template.getID(),
                               template.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject()
   instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }

   //run the process
   ClientObjectWrapper output = process.call(template.getName()
   , input);
   DataObject myOutput = null;
   if ( output.getObject() != null && output.getObject()
   instanceof DataObject )
   {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
   }
   ```

이 조치를 통해 프로세스 템플리트의 인스턴스인 CustomerTemplate가 작성되고 일부 고객 데이터가 전달됩니다. 프로세스가 완료된 경우에만 조작이 리턴됩니다. 프로세스 OrderNo의 결과가 호출자에게 리턴됩니다.

비고유 시작 서비스를 포함하는 마이크로플로우 실행:

마이크로플로우는 수신 활동 또는 선택 활동으로 시작됩니다. 마이크로플로우에서 여러 onMessage 정의를 갖는 선택 활동으로 시작할 경우 시작 서비스는 고유하지 않습니다.

이 태스크 정보

마이크로플로우가 요청-응답 조작을 구현하는 경우, 즉, 프로세스에 응답이 포함된 경우, 호출 메소드를 사용하여 호출에서 시작 서비스의 ID를 전달하는 프로세스를 실행할 수 있습니다.

마이크로플로우가 단방향 조작이면 sendMessage 메소드를 사용하여 프로세스를 실행하십시오. 이 예에서는 이 메소드를 다루지 않습니다.

프로시저

1. 옵션: 프로세스 템플리트를 표시하여 실행하려는 프로세스 이름을 찾으십시오.

   프로세스의 이름을 이미 아는 경우 이 단계는 선택적입니다.

   ```
   ProcessTemplateData[] processTemplates
   = process.queryProcessTemplates
   ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
    "PROCESS_TEMPLATE.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   결과는 이름별로 정렬됩니다. 조회는 마이크로플로우로 시작할 수 있는 처음 50개의 정렬된 템플리트를 포함하는 배열을 리턴합니다.

2. 호출할 시작 서비스를 판별하십시오.

   이 예에서는 첫 번째 발견된 템플리트를 사용합니다.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
           process.getStartActivities(template.getID());
   ```

3. 해당 유형의 입력 메시지로 프로세스를 시작하십시오.

   메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다.

   ```
   ActivityServiceTemplateData activity
   = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input =
         process.createMessage(activity.getServiceTemplateID(),
   ```

```
                              activity.getActivityTemplateID(),
                              activity.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
//run the process
ClientObjectWrapper output = process.call(activity
.getServiceTemplateID(),
    activity.getActivityTemplateID(),
                input);
//check the output of the process, for example,
an order number
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject()
instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

이 조치를 통해 프로세스 템플리트의 인스턴스인 CustomerTemplate가 작성되고 일부 고객 데이터가 전달됩니다. 프로세스가 완료된 경우에만 조작이 리턴됩니다. 프로세스 OrderNo의 결과가 호출자에게 리턴됩니다.

고유 시작 서비스를 포함하는 장기 실행 프로세스 시작:

시작 서비스가 고유한 경우 시작 메소드를 사용하고 프로세스 템플리트 이름을 매개변수로 전달할 수 있습니다. 장기 실행 프로세스가 단일 수신 또는 선택 활동으로 시작하고 단일 선택 활동이 하나의 onMessage 정의만 갖는 경우입니다.

프로시저

1. 옵션: 프로세스 템플리트를 표시하여 시작할 프로세스의 이름을 찾으십시오.

   프로세스의 이름을 이미 아는 경우 이 단계는 선택적입니다.

   ```
   ProcessTemplateData[] processTemplates
   = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION

          _MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

   결과는 이름별로 정렬됩니다. 조회는 시작 메소드로 시작할 수 있는 처음 50개의 정렬된 템플리트를 포함하는 배열을 리턴합니다.

2. 해당 유형의 입력 메시지로 프로세스를 시작하십시오.

메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다. 프로세스-인스턴스 이름을 지정하는 경우 밑줄로 시작하지 않아야 합니다. 프로세스-인스턴스 이름이 지정되지 않은 경우 문자열 형식의 프로세스 인스턴스 ID(PIID)가 이름으로 사용됩니다.

```
ProcessTemplateData template
= processTemplates[0];
//create a message for the single starting receive activity
ClientObjectWrapper input = process.createMessage
                            (template.getID(),
                            template.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null &&
input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example,

  a customer name
  myMessage.setString("CustomerName", "Smith");
}
//start the process
PIID piid = process.initiate(template.getName(),
"CustomerOrder", input);
```

이 조치를 통해 인스턴스 CustomerOrder가 작성되고 일부 고객 데이터가 전달됩니다. 프로세스가 시작되면 조작을 통해 새 프로세스 인스턴스의 오브젝트 ID가 호출자에게 리턴됩니다.

프로세스 인스턴스의 시작자는 요청의 호출자로 설정됩니다. 해당 사용자는 프로세스 인스턴스의 작업 항목을 수신합니다. 프로세스 인스턴스의 프로세스 관리자, 독자 및 편집자가 결정되어 프로세스 인스턴스에 대한 작업 항목을 수신합니다. 후속 활동 인스턴스가 결정됩니다. 이들은 자동으로 시작되거나 휴먼 타스크, 수신 또는 선택 활동인 경우 잠재적 소유자에 대한 작업 항목이 작성됩니다.

비고유 시작 서비스를 포함하는 장기 실행 프로세스 시작:

장기 실행 프로세스는 여러 시작 수신 또는 선택 활동을 통해 시작할 수 있습니다. 시작 메소드를 사용하여 프로세스를 시작할 수 있습니다. 시작 서비스가 고유하지 않은 경우 예를 들어, 프로세스가 여러 개의 수신 또는 선택 활동으로 시작하거나 여러 onMessage 정의를 갖는 선택 활동으로 시작하면 호출할 서비스를 식별해야 합니다.

프로시저

1. 옵션: 프로세스 템플리트를 표시하여 시작할 프로세스의 이름을 찾으십시오.

   프로세스의 이름을 이미 아는 경우 이 단계는 선택적입니다.

   ```
   ProcessTemplateData[] processTemplates
   = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
   ```

```
    PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_LONG_RUNNING",
  "PROCESS_TEMPLATE.NAME",
   new Integer(50),
   (TimeZone)null);
```

결과는 이름별로 정렬됩니다. 조회는 장기 실행 프로세스로 시작할 수 있는 처음 50 개의 정렬된 템플리트를 포함하는 배열을 리턴합니다.

2. 호출할 시작 서비스를 판별하십시오.

```
ProcessTemplateData template = processTemplates[0];
ActivityServiceTemplateData[] startActivities =
       process.getStartActivities(template.getID());
```

3. 해당 유형의 입력 메시지로 프로세스를 시작하십시오.

메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다. 프로세스-인스턴스 이름을 지정하는 경우 밑줄로 시작하지 않아야 합니다. 프로세스-인스턴스 이름이 지정되지 않은 경우 문자열 형식의 프로세스 인스턴스 ID(PIID)가 이름으로 사용됩니다.

```
ActivityServiceTemplateData activity
= startActivities[0];
//create a message for the service to be called
ClientObjectWrapper input = process.createMessage
                           (activity.getServiceTemplateID(),
                            activity.getActivityTemplateID(),
                            activity.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example,

  a customer name
  myMessage.setString("CustomerName", "Smith");
}
//start the process
PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                            activity.getActivityTemplateID(),
                            input);
```

이 조치는 인스턴스를 작성하고 일부 고객 데이터를 전달합니다. 프로세스가 시작되면 조작을 통해 새 프로세스 인스턴스의 오브젝트 ID가 호출자에게 리턴됩니다.

프로세스 인스턴스의 시작자는 요청 호출자로 설정되고 프로세스 인스턴스에 대한 작업 항목을 수신합니다. 프로세스 인스턴스의 프로세스 관리자, 독자 및 편집자가 결정되어 프로세스 인스턴스에 대한 작업 항목을 수신합니다. 후속 활동 인스턴스가 결정됩니다. 이들은 자동으로 시작되거나 휴먼 타스크, 수신 또는 선택 활동인 경우 잠재적 소유자에 대한 작업 항목이 작성됩니다.

비즈니스 프로세스 일시중단 및 재개:

장기 실행, 맨 위 레벨 프로세스 인스턴스가 실행 중인 경우 일시중단할 수 있으며 이를 다시 재개하여 완료하십시오.

시작하기 전에

이 태스크 정보

이후에 프로세스에서 사용되는 백엔드 시스템으로 액세스를 구성하는 경우와 같은 상황에 프로세스 인스턴스를 일시중단할 수 있습니다. 프로세스의 전제조건이 충족되면 프로세스 인스턴스를 재개할 수 있습니다. 프로세스 인스턴스의 실패 원인이 된 문제점을 해결하기 위해 프로세스를 일시중단했거나 문제점을 해결한 후 프로세스를 재개할 수도 있습니다.

프로세스 인스턴스를 일시중단하려면 실행 또는 실패 상태여야 합니다. 호출자는 프로세스 관리자 또는 시스템 관리자여야 합니다. 그러나 비즈니스 플로우 관리자가 프로세스 관리를 시스템 관리자로 제한하는 대체 프로세스 관리 권한 모드를 사용하는 경우에는 BPESystemAdministrator 역할의 호출자만 이 조치를 수행할 수 있습니다.

프로시저

1. 일시중단하려는 실행 중인 프로세스인 CustomerOrder를 가져오십시오.

   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```

2. 프로세스 인스턴스를 일시중단하십시오.

   ```
   PIID piid = processInstance.getID();
   process.suspend( piid );
   ```

   이 조치를 실행하면 지정된 맨 위 레벨 프로세스 인스턴스가 일시중단됩니다. 프로세스 인스턴스는 일시중단 상태가 됩니다. 이 상태에서, 시작된 활동은 여전히 완료할 수 있지만 새 활동은 활성화되지 않습니다. 또한 자율 속성이 하위로 설정된 서브프로세스는 실행 중, 실패 중, 종료 중 또는 보상 중 상태에 있는 경우 일시중단됩니다. 이 프로세스 인스턴스와 연관된 인라인 타스크 및 독립형 타스크는 일시중단되지 않습니다.

3. 프로세스 인스턴스를 재개하십시오.

   ```
   process.resume( piid );
   ```

   이 조치를 실행하면 프로세스 인스턴스 및 서브프로세스가 일시중단되기 이전 상태로 돌아갑니다.

비즈니스 프로세스 다시 시작:

완료됨, 중단됨, 실패함 또는 보상됨 상태의 프로세스 인스턴스를 다시 시작할 수 있습니다.

이 태스크 정보

프로세스 인스턴스를 다시 시작하는 것은 프로세스 인스턴스를 처음으로 시작하는 것과 비슷합니다. 그러나 프로세스 인스턴스가 다시 시작되면 프로세스 인스턴스 ID가 인식되고 인스턴스의 입력 메시지가 사용 가능해집니다.

프로세스에 프로세스 인스턴스를 작성할 수 있는 둘 이상의 수신 활동 또는 선택 활동(선택 사항 수신 활동이라고도 함)이 있는 경우, 이 활동에 속한 모든 메시지를 사용하여 프로세스 인스턴스를 다시 시작합니다. 이 활동이 요청-응답 조작을 구현할 경우 연관된 응답 활동을 탐색하면 응답이 다시 전송됩니다.

호출자는 프로세스 관리자 또는 시스템 관리자여야 합니다. 그러나 비즈니스 플로우 관리자가 프로세스 관리를 시스템 관리자로 제한하는 대체 프로세스 관리 권한 모드를 사용하는 경우에는 BPESystemAdministrator 역할의 호출자만 이 조치를 수행할 수 있습니다.

프로시저

1. 다시 시작하려는 프로세스를 가져오십시오.

```
ProcessInstanceData processInstance =
                    process.getProcessInstance("CustomerOrder");
```

2. 프로세스 인스턴스를 다시 시작하십시오.

```
PIID piid = processInstance.getID();
process.restart( piid );
```

   이 조치를 실행하면 지정된 프로세스 인스턴스가 다시 시작됩니다.

프로세스 인스턴스 종료:

복구 불가능한 상태의 최상위 레벨 인스턴스를 종료해야 합니다.

이 태스크 정보

이 조치를 수행하려면 호출자가 프로세스 관리자 또는 시스템 관리자여야 합니다. 그러나 비즈니스 플로우 관리자가 프로세스 관리를 시스템 관리자로 제한하는 대체 프로세스 관리 권한 모드를 사용하는 경우에는 BPESystemAdministrator 역할의 호출자만 이 조치를 수행할 수 있습니다.

프로세스 인스턴스는 처리되지 않은 서브프로세스 또는 활동의 완료를 대기하지 않고 즉시 종료되기 때문에 예외적인 상황에서만 이 조치를 수행해야 합니다.

프로시저

1. 종료할 프로세스 인스턴스를 검색하십시오.

```
ProcessInstanceData processInstance =
       process.getProcessInstance("CustomerOrder");
```

2. 프로세스 인스턴스를 종료하십시오.

   프로세스 인스턴스를 종료하는 경우 보상 여부에 관계없이 프로세스를 종료할 수 있습니다.

   보상이 있는 상태로 프로세스 인스턴스를 종료하려면 다음을 수행하십시오.

   ```
   PIID piid = processInstance.getID();
   process.forceTerminate(piid, CompensationBehaviour.INVOKE_COMPENSATION);
   ```

   보상 없이 프로세스 인스턴스를 종료하려면 다음을 수행하십시오.

   ```
   PIID piid = processInstance.getID();
   process.forceTerminate(piid);
   ```

   보상이 있는 상태로 프로세스 인스턴스를 종료하면 최상위 레벨 범위에 결함이 발생한 것처럼 프로세스의 보상이 실행됩니다. 보상 없이 프로세스 인스턴스를 종료하는 경우 프로세스 인스턴스는 활동, 수행할 타스크 또는 인라인 호출 타스크가 정상적으로 종료될 때까지 기다리지 않고 즉시 종료됩니다.

   프로세스와 관련된 프로세스 및 독립형 타스크로 시작된 응용프로그램은 강제 종료 요청으로 종료되지 않습니다. 이 응용프로그램이 종료되는 경우, 프로세스가 시작한 응용프로그램을 명시적으로 종료하는 명령문을 프로세스 응용프로그램에 추가해야 합니다.

프로세스 인스턴스 삭제:

완료된 프로세스 인스턴스는 해당 특성이 프로세스 모델의 프로세스 템플리트용으로 설정된 경우 자동으로 Business Process Choreographer 데이터베이스에서 삭제됩니다. 예를 들어, 데이터베이스에 프로세스 인스턴스를 보관하여 감사 로그에 작성되지 않은 프로세스 인스턴스에서 데이터를 조회하고자 할 수 있습니다. 그러나 저장된 프로세스 인스턴스 데이터는 디스크 공간 및 성능에 영향을 미칠 뿐만 아니라 동일한 상관 세트 값을 사용하는 프로세스 인스턴스를 작성할 수 없도록 합니다. 그러므로 정기적으로 데이터베이스에서 프로세스 인스턴스 데이터를 삭제해야 합니다.

이 태스크 정보

프로세스 인스턴스를 삭제하려면 프로세스 관리자 권한이 있어야 하며 이 프로세스 인스턴스는 맨 위 레벨 프로세스 인스턴스여야 합니다.

다음 예에서는 완료된 모든 프로세스 인스턴스를 삭제하는 방법을 보여줍니다.

프로시저

1. 완료된 프로세스 인스턴스를 표시하십시오.

```
QueryResultSet result =
     process.query("DISTINCT PROCESS_INSTANCE.PIID",
                   "PROCESS_INSTANCE.STATE =
                           PROCESS_INSTANCE.STATE.STATE_FINISHED",
                    (String)null, (Integer)null, (TimeZone)null);
```

이 조치를 실행하면 완료된 프로세스 인스턴스 목록이 결과 조회 세트로 리턴됩니다.

2. 완료된 프로세스 인스턴스를 삭제하십시오.

```
while (result.next() )
{
   PIID piid = (PIID) result.getOID(1);
   process.delete(piid);
}
```

이 조치는 선택한 프로세스 인스턴스 및 인라인 타스크를 데이터베이스에서 삭제합니다.

휴먼 타스크 활동 처리

비즈니스 프로세스의 휴먼 타스크 활동은 작업 항목을 통해 조직의 여러 사용자에게 지정됩니다. 프로세스가 시작하면 작업 항목이 잠재적 사용자에 대해 작성됩니다.

이 태스크 정보

휴먼 타스크 활동이 활성화되면 활동 인스턴스 및 연관된 수행할 타스크가 모두 작성됩니다. 휴먼 타스크 활동의 핸들 및 작업 항목 관리는 휴먼 타스크 관리자에게 위임됩니다. 활동 인스턴스의 상태 변경은 타스크 인스턴스에 반영되며 반대의 경우도 가능합니다.

잠재적 소유자가 활동을 청구합니다. 이 사용자는 관련 정보를 제공하고 활동을 완료해야 합니다.

프로시저

1. 작업이 가능한 로그온 상태의 사용자가 가지고 있는 활동을 표시하십시오.

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
                    ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
                    WORK_ITEM.REASON =
                         WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                    (String)null, (Integer)null, (TimeZone)null);
```

이 조치는 로그온 상태의 사용자가 작업할 수 있는 활동을 포함하는 결과 조회 세트를 리턴합니다.

2. 작업할 활동을 청구하십시오.

```
if (result.size() > 0)
{
 result.first();
   AIID aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject()

  instanceof DataObject )
  {
    activityInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

활동이 청구될 때 활동의 입력 메시지가 리턴됩니다.

3. 활동의 작업이 완료되면 활동을 완료하십시오. 활동은 성공적으로 완료되거나 결함 메시지와 함께 완료될 수 있습니다. 활동이 성공적인 경우 출력 메시지가 전달됩니다. 활동이 성공적이지 않은 경우 활동은 실패 상태 또는 중지 상태가 되며 결함 메시지가 전달됩니다. 이 조치에 해당하는 메시지를 작성해야 합니다. 메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다.

   a. 활동을 완료하려면 출력 메시지를 작성하십시오.

   ```
   ActivityInstanceData activity = process.getActivityInstance(aiid);
   ClientObjectWrapper output =
         process.createMessage(aiid, activity

         .getOutputMessageTypeName());
   DataObject myMessage = null ;
   if ( output.getObject()!= null && output.getObject()
   instanceof DataObject )
   {
     myMessage = (DataObject)output.getObject();
     //set the parts in your message, for example, an order number
     myMessage.setInt("OrderNo", 4711);
   }

   //complete the activity
   process.complete(aiid, output);
   ```

   이 조치는 순번을 포함하는 출력 메시지를 설정합니다.

   b. 결함이 발생할 때 활동을 완료하려면 결함 메시지를 작성하십시오.

   ```
   //retrieve the faults modeled for the human task activity
   List faultNames = process.getFaultNames(aiid);

   //create a message of the appropriate type
   ClientObjectWrapper myFault =
         process.createMessage(aiid, faultNames.get(0));

   // set the parts in your fault message, for example,
   an error number
   DataObject myMessage = null ;
   ```

```
if ( myFault.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

process.complete(aiid, myFault,(String)faultNames.get(0) );
```

이 조치는 활동을 실패 또는 중지 상태로 설정합니다. 프로세스 모델의 활동에 대한 **continueOnError** 매개변수가 참으로 설정되는 경우, 활동은 실패 상태로 되며 탐색이 계속됩니다. **continueOnError** 매개변수가 false로 설정되고 주변 범위에서 결함이 발견되지 않으면 활동은 중지 상태가 됩니다. 이 상태에서는 완료 강제 실행 또는 재시도 강제 실행을 통해 활동을 복구할 수 있습니다.

관련 개념


활동 및 비즈니스 프로세스의 오류 시 계속 동작

비즈니스 프로세스를 정의할 때, 예상치 못한 결함이 발생하고 해당 결함에 대한 결함 핸들러가 정의되지 않은 경우 발생하는 일을 지정할 수 있습니다. 프로세스를 정의할 때 **오류 발생 시 계속 진행** 설정을 사용하여 결함이 발생하는 위치에서 중지하도록 지정할 수 있습니다.

단일 사용자 워크플로우 처리

일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예: 온라인 서점에서 책 주문). 이러한 유형의 워크플로우에는 병렬 경로가 없습니다. initiateAndClaimFirst 및 completeAndClaimSuccessor API는 이 워크플로우 유형의 처리를 지원합니다. 다음 예제는 클라이언트측 페이지 플로우를 사용하는 단일 개인 워크플로우의 구현을 나타냅니다.

이 태스크 정보

단일 사용자 워크플로우를 *페이지 플로우* 또는 *화면 플로우*라고도 합니다. 두 종류의 페이지 플로우가 있습니다.

- 클라이언트측 페이지 플로우: 여기에서는 클라이언트측 기술을 사용하여 여러 페이지 간의 탐색을 실현합니다(예: 다중 페이지 Lotus® Forms 양식).
- 서버측 페이지 플로우: 모델링된 비즈니스 프로세스 및 휴먼 타스크 세트를 사용하여 실현되므로 후속타스크가 동일한 사용자에게 지정됩니다.

서버측 페이지 플로우는 클라이언트측 페이지 플로우보다 강력하지만 처리하는 데 더 많은 서버 자원을 이용합니다. 그러므로 이 유형의 워크플로우는 기본적으로 다음과 같은 상황에서 사용할 것을 고려하십시오.

- 사용자 인터페이스에서 수행되는 단계 사이에 서비스를 호출해야 합니다(예: 데이터 검색 또는 갱신).

- 사용자 인터페이스 상호작용을 완료한 후 CEI 이벤트를 작성하도록 요구하는 감사 요구사항이 있습니다.

단일 사용자 워크플로우의 일반적인 예는 온라인 서점의 주문 프로세스입니다. 여기에서 구매자는 책을 주문하는 조치 순서를 완료합니다. 이러한 일련의 조치는 일련의 휴먼 타스크 활동(수행할 타스크)으로 구현될 수 있습니다. 구매자가 여러 권의 책을 주문하기로 결정하는 경우에는 단일 주문 프로세스를 시작하고 다음 휴먼 타스크 활동을 청구하는 것과 같습니다.

initiateAndClaimFirst API는 페이지 플로우를 시작합니다. 즉 지정된 프로세스를 시작하고 활동 순서에서 첫 번째 휴먼 타스크 활동을 청구합니다. 청구한 활동에 대한 정보(작업할 입력 메시지 포함)가 리턴됩니다.

그런 다음 completeAndClaimSuccessor API는 휴먼 타스크 활동을 완료하고 로그온한 사용자에 대해 동일한 프로세스 인스턴스 내의 다음 활동을 청구합니다. 작업할 입력 메시지를 포함하여 다음 청구 활동에 대한 정보가 리턴됩니다. 다음 활동은 완료된 활동의 동일 트랜잭션 내에서만 사용 가능하기 때문에 프로세스 모델에서 모든 휴먼 타스크 활동의 트랜잭션 동작을 participates로 설정해야 합니다.

비즈니스 플로우 관리자 API와 휴먼 타스크 관리자 API를 모두 사용하는 예제와 이 예제를 비교하십시오.

프로시저

1. 책 주문 프로세스를 시작하고 활동 순서의 첫 번째 활동을 청구하십시오. 해당 유형의 입력 메시지로 프로세스를 시작하십시오.메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다. 프로세스 인스턴스 이름을 지정하는 경우에는 밑줄로 시작하지 않아야 합니다. 프로세스 인스턴스 이름이 지정되지 않으면 문자열 형식의 프로세스 인스턴스 ID(PIID)가 이름으로 사용됩니다.
   a. 프로세스 템플리트를 검색하여 적절한 유형의 입력 메시지를 작성하십시오.

```
ProcessTemplateData template =
process.getProcessTemplate("CustomerOrder");
ClientObjectWrapper input = process.createMessage(template.getID(),
                            template.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

   b. 프로세스를 시작하고 첫 번째 휴먼 타스크 활동을 청구하십시오.

```
InitiateAndClaimFirstResult result =
  process.initiateAndClaimFirst("CustomerOrder",
```

```
    "MyOrderProcess", input);
  AIID aiid = result.getAIID();
  ClientObjectWrapper input = result.getInputMessage();
```

첫 번째 활동이 청구될 때 청구된 활동의 입력 메시지와 ID가 리턴됩니다.

2. 활동의 작업이 완료되면 활동을 완료하고 다음 활동을 청구하십시오.

   활동을 완료하려면 출력 메시지를 전달됩니다. 출력 메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다.

```
ActivityInstanceData activity =
process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity

      .getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the activity and claim the next one
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   이 조치는 주문 번호가 포함된 출력 메시지를 설정하고 순서에 있는 다음 활동을 청구합니다. 후속 활동으로 AutoClaim을 설정하고 추적할 수 있는 여러 경로가 있으면 모든 후속 활동이 청구되며 다음 활동으로 임의 활동이 리턴됩니다. 이 사용자에게 지정할 수 있는 후속 활동이 없으면 Null이 리턴됩니다.

   프로세스에 후속 병렬 경로가 있고 이들 경로에 로그온 사용자가 둘 이상의 활동의 잠재적 소유자인 휴먼 타스크 활동이 있으면 임의 활동이 자동으로 청구되어 그 다음 활동으로 리턴됩니다.

3. 다음 활동을 작업하십시오.

```
String name = successor.getActivityName();

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject()

        instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

4. 2단계를 계속하여 활동을 완료하십시오.

관련 태스크

491 페이지의『휴먼 타스크를 포함하는 단일 사용자 워크플로우 처리』
일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예: 온라인 서점에서 책 주문). 이 예제는 서버측 페이지 플로우를 사용하여 단일 사용자 워크플로우를 구현하는 방법을 보여줍니다. 비즈니스 플로우 관리자와 휴먼 타스크 관리자 API는 모두 워크플로우를 처리하는 데 사용됩니다.

대기 중 활동에 메시지 전송

인바운드 메시지 활동(수신 활동, 선택 활동의 onMessage, 이벤트 핸들러의 onEvent)을 사용하여 실행 중인 프로세스를 "외부"의 이벤트와 동기화할 수 있습니다. 예를 들어, 정보 요청에 대해 고객이 응답으로 보낸 전자 우편을 수신하는 것이 해당 이벤트가 될 수 있습니다.

이 태스크 정보

시작 타스크를 사용하여 활동에 메시지를 전송할 수 있습니다.

프로시저

1. 특정 프로세스 인스턴스 ID를 가진 프로세스 인스턴스에서 로그온한 사용자로부터 메시지를 대기 중인 활동 서비스 템플리트를 표시하십시오.

   ```
   ActivityServiceTemplateData[] services = process.getWaitingActivities(piid);
   ```

2. 대기 중인 첫 번째 서비스로 메시지를 전송하십시오.

   첫 번째 서비스는 사용자가 사용할 서비스로 간주됩니다. 호출자는 메시지를 수신하는 활동의 잠재적 시작자 또는 프로세스 인스턴스 관리자여야 합니다.

   ```
   VTID vtid = services[0].getServiceTemplateID();
   ATID atid = services[0].getActivityTemplateID();
   String inputType = services[0].getInputMessageTypeName();

   // create a message for the service to be called
     ClientObjectWrapper message =
           process.createMessage(vtid,atid,inputMessageTypeName);
     DataObject myMessage = null;
     if ( message.getObject()!= null && message

     .getObject() instanceof DataObject )
     {
       myMessage = (DataObject)message.getObject();
       //set the strings in the message, for example,

       chocolate is to be ordered
       myMessage.setString("Order", "chocolate");
     }
   ```

```
  // send the message to the waiting activity
  process.sendMessage(vtid, atid, message);
}
```

이 조치는 지정된 메시지를 대기 중 활동 서비스로 전송하고 일부 순서 데이터를 전달합니다.

프로세스 인스턴스 ID를 지정하여 메시지가 지정된 프로세스 인스턴스에 전송되는지 확인할 수도 있습니다. 프로세스 인스턴스 ID가 지정되지 않는 경우 메시지가 메시지의 상관 값에서 식별한 프로세스 인스턴스 및 활동 서비스로 전송됩니다. 프로세스 인스턴스 ID가 지정된 경우 상관 값을 사용해 찾은 프로세스 인스턴스를 확인하여 지정된 프로세스 인스턴스 ID를 갖는지 확인하십시오.

이벤트 핸들

전체 비즈니스 프로세스 및 각 영역은 연관된 이벤트가 발생되는 경우 호출되는 이벤트 핸들러와 연관될 수 있습니다. 프로세스가 이벤트 핸들러를 사용하여 웹 서비스 조작을 제공한다는 점에서 이벤트 핸들러는 수신 활동 또는 선택 활동과 유사합니다.

이 태스크 정보

해당 범위가 실행 중인 한 이벤트 핸들러를 계속 호출할 수 있습니다. 또한 이벤트 핸들러의 다중 인스턴스는 동시에 활성화될 수 있습니다.

다음 코드 스니펫은 주어진 프로세스 인스턴스용 활성 이벤트 핸들러를 가져오는 방법과 입력 메시지를 전송하는 방법을 보여줍니다.

프로시저

1. 프로세스 인스턴스 ID의 데이터를 판별하고 프로세스의 활성 이벤트 핸들러를 표시하십시오.

```
ProcessInstanceData processInstance =
       process.getProcessInstance( "CustomerOrder2711");
EventHandlerTemplateData[] events = process.getActiveEventHandlers(
                                    processInstance.getID() );
```

2. 입력 메시지를 전송하십시오.

   이 예에서는 첫 번째 발견된 이벤트 핸들러를 사용합니다.

```
EventHandlerTemplateData event = null;
if ( events.length > 0 )
{
    event = events[0];

    // create a message for the service to be called
    ClientObjectWrapper input = process.createMessage(
    event.getID(), event.getInputMessageTypeName());

    if (input.getObject() != null && input.getObject()
```

```
        instanceof DataObject )
        {
           DataObject inputMessage = (DataObject)input.getObject();
           // set content of the message, for example,

           a customer name, order number
           inputMessage.setString("CustomerName", "Smith");
           inputMessage.setString("OrderNo", "2711");

            // send the message
           process.sendMessage( event.getProcessTemplateName(),
                                event.getPortTypeNamespace(),
                                event.getPortTypeName(),
                                event.getOperationName(),
         input );
         }
    }
```

이 조치를 실행하면 지정된 메시지가 프로세스의 활성 이벤트 핸들러에 전송됩니다.

프로세스 결과 분석

프로세스는 WSDL(Web Services Description Language) 단방향 또는 요청-응답 조작으로 모델화된 웹 서비스 조작을 노출할 수 있습니다. 단방향 인터페이스가 있는 장기 실행 프로세스의 결과는 프로세스에 출력이 없기 때문에 getOutputMessage 메소드를 사용하여 검색할 수 없습니다. 그러나 대신에 변수의 컨텐츠를 조회할 수 있습니다.

이 태스크 정보

프로세스 인스턴스가 파생된 프로세스 템플리트가 파생된 프로세스 인스턴스의 자동 삭제를 지정하지 않는 경우에만 프로세스의 결과가 데이터베이스에 저장됩니다.

프로시저

프로세스 결과를 분석하십시오. 예를 들어, 순번을 확인하십시오.

```
QueryResultSet result = process.query
                       ("PROCESS_INSTANCE.PIID",
                        "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
                         PROCESS_INSTANCE.STATE =
                                 PROCESS_INSTANCE.STATE.STATE_FINISHED",
                         (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  PIID piid = (PIID) result.getOID(1);
  ClientObjectWrapper output = process.getOutputMessage(piid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject()

  instanceof DataObject )
  {
```

```
    myOutput  = (DataObject)output.getObject();
    int order = myOutput.getInt("OrderNo");
  }
}
```

활동 복구

장기 실행 프로세스는 역시 장기 실행하는 활동을 포함할 수 있습니다. 이 활동은 미발견 오류를 발견하여 중지 상태로 될 수 있습니다. 실행 중인 상태의 활동이 응답하지 않는 것으로 나타날 수도 있습니다. 두 가지 경우 모두 프로세스 관리자가 프로세스 탐색을 계속할 수 있도록 여러 가지 방법으로 활동에 조치를 취할 수 있습니다.

이 태스크 정보

Business Process Choreographer API는 활동을 복구하기 위해 forceRetry 및 forceComplete 메소드를 제공합니다. 예제에서는 사용자 응용프로그램에 활동에 대한 복구 조치를 추가하는 방법을 보여줍니다.

활동 강제 완료:

장기 실행 프로세스의 활동에 결함이 발생할 수 있습니다. 결함 핸들러가 엔클로징 범위에서 이러한 결함을 발견하지 못했으며 연관된 활동 템플리트가 오류 발생 시 활동이 중지되도록 지정할 경우, 활동이 중지 상태로 되어 복구될 수 있도록 합니다. 이 상태에서 활동을 강제로 완료할 수 있습니다.

이 태스크 정보

예를 들어, 활동이 응답하지 않는 경우에도 실행 상태의 활동을 강제로 완료할 수도 있습니다.

특정 활동 유형에 대한 추가 요구사항이 존재합니다.

휴먼 타스크 활동

force-complete 호출에서 전송되었어야 하는 메시지 또는 발생했어야 하는 결함과 같은 매개변수를 전달할 수 있습니다.

스크립트 활동

force-complete 호출에서 매개변수를 전달할 수 없습니다. 그러나 복구해야 할 변수를 설정해야 합니다.

호출 활동

활동이 실행 상태인 경우 서브프로세스가 아닌 비동기 서비스를 호출하는 호출 활동을 강제로 완료할 수도 있습니다. 비동기 서비스가 호출되고 응답되지 않는 경우 다음을 수행할 수 있습니다.

프로시저

1. 중지 상태에 있는 중지된 활동을 표시하십시오.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

이 조치는 CustomerOrder 프로세스 인스턴스에 대한 중지된 활동을 리턴합니다.

2. 활동(예: 중지된 휴먼 타스크 활동)을 완료하십시오.

   이 예에서는 출력 메시지가 전달됩니다.

```
if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);
   ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper output =
        process.createMessage(aiid, activity.

 getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( output.getObject()!= null && output.

  getObject() instanceof DataObject )
   {
     myMessage = (DataObject)output.getObject();
     //set the parts in your message, for example, an order number
     myMessage.setInt("OrderNo", 4711);
   }

   boolean continueOnError = true;
   process.forceComplete(aiid, output, continueOnError);
}
```

   이 조치를 실행하여 활동을 완료합니다. 오류가 발생하면 **continueOnError** 매개변수는 forceComplete 요청에서 발생한 결함에 대해 수행할 조치를 판별합니다.

   이 예에서 **continueOnError**는 true입니다. 즉, 결함이 있는 경우 활동은 실패 상태가 됩니다. 이 결함은 처리되거나 프로세스 범위에 도달할 때까지 활동의 앤클로징 범위로 전달됩니다. 프로세스는 실패 중 상태가 되고 결국 실패됨 상태가 됩니다.

관련 개념

활동 및 비즈니스 프로세스의 오류 시 계속 동작

비즈니스 프로세스를 정의할 때, 예상치 못한 결함이 발생하고 해당 결함에 대한 결함 핸들러가 정의되지 않은 경우 발생하는 일을 지정할 수 있습니다. 프로세스를 정의할 때 **오류 발생 시 계속 진행** 설정을 사용하여 결함이 발생하는 위치에서 중지하도록 지정할 수 있습니다.

중지된 활동 재실행:

장기 실행 프로세스의 활동이 엔클로징 범위에서 미발견 결함을 발견하고 연관 활동 템플리트에서 오류가 발생할 때 활동이 중지되도록 지정하는 경우, 활동이 중지 상태가 되기 때문에 복구할 수 있습니다. 활동을 재실행할 수 있습니다.

이 태스크 정보

활동에서 사용한 변수를 설정할 수 있습니다. 스크립트 활동을 실행할 때 force-retry 호출에서 활동을 통해 예상하는 메시지와 같은 매개변수를 전달할 수도 있습니다.

프로시저

1. 중지된 활동을 표시하십시오.

```
QueryResultSet result =
      process.query("DISTINCT ACTIVITY.AIID",
                    "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                     PROCESS_INSTANCE.NAME='CustomerOrder'",
                     (String)null, (Integer)null, (TimeZone)null);
```

   이 조치는 CustomerOrder 프로세스 인스턴스에 대한 중지된 활동을 리턴합니다.

2. 활동(예: 중지된 휴먼 타스크 활동)의 실행을 재시도하십시오.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper input =
        process.createMessage(aiid, activity

 .getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( input.getObject()!= null && input.getObject()

  instanceof DataObject )
    {
      myMessage = (DataObject)input.getObject();
      //set the strings in your message, for example,

      chocolate is to be ordered
      myMessage.setString("OrderNo", "chocolate");
    }

   boolean continueOnError = true;
   process.forceRetry(aiid, input, continueOnError);
}
```

   이 조치를 통해 활동이 재시도됩니다. 오류가 발생하면 **continueOnError** 매개변수는 forceRetry 요청 처리 중 오류가 발생할 경우 해당 조치를 취할 것인지 판별합니다.

   이 예에서 **continueOnError**는 true입니다. 즉, forceRetry 요청 처리 중 오류가 발생한 경우 활동이 실패 상태가 됨을 의미합니다. 이 결함은 처리되거나 프로세스

범위에 도달할 때까지 활동의 엔클로징 범위로 전달됩니다. 그러면 프로세스는 실패 상태가 되고 프로세스 상태가 실패 상태로 종료되기 전에 프로세스 레벨의 결함 핸들러가 실행됩니다.

관련 개념


활동 및 비즈니스 프로세스의 오류 시 계속 동작
비즈니스 프로세스를 정의할 때, 예상치 못한 결함이 발생하고 해당 결함에 대한 결함 핸들러가 정의되지 않은 경우 발생하는 일을 지정할 수 있습니다. 프로세스를 정의할 때 **오류 발생 시 계속 진행** 설정을 사용하여 결함이 발생하는 위치에서 중지하도록 지정할 수 있습니다.

결합, 루프 또는 카운터 평가에 실패했기 때문에 중지된 활동 복구:

조인 조건 또는 루프 조건에서 예외가 발생하거나 forEach 카운터 값이 평가된 경우 활동이 중지될 수 있습니다. 관리자는 활동 실행을 재시도하지 않기로 결정합니다(예를 들어, 평가가 다시 실패할 수 있으므로). 이러한 경우에 Business Process Choreographer EJB API를 사용하여 프로세스 탐색을 계속할 수 있도록 표현식에 올바른 값을 제공할 수 있습니다.

이 태스크 정보

모든 유형의 활동에 대한 조인 조건의 값 및 while 또는 repeat-until 활동의 루프 조건 값을 설정할 수 있습니다. 또한 시작 및 종료 카운터 값과 forEach 활동에 대해 완료된 분기의 최대수도 설정할 수 있습니다. 완료된 분기에 설정하는 값은 프로세스 모델에서 forEach 활동의 정의에 따라 다릅니다. 모델에 빠른 종료 조건이 지정된 경우에는 완료된 분기의 최대수에 값을 설정하십시오. 빠른 종료 조건이 지정되지 않은 경우에는 완료된 분기의 최대수 값을 널로 설정하십시오.

다음 샘플은 루프 조건의 값을 설정하는 방법을 보여줍니다.

프로시저

1. 루프 조건의 평가가 실패하여 중지된 활동을 나열하십시오.

```
QueryResultSet result = process.query(
  "DISTINCT ACTIVITY.AIID",
  "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
   ACTIVITY.STOP_REASON = ACTIVITY.STOP_REASON.STOP_REASON_IMPLEMENTATION_FAILED AND
   (ACTIVITY.KIND = ACTIVITY.KIND.KIND_WHILE OR
      ACTIVITY.KIND = ACTIVITY.KIND.KIND_REPEAT_UNTIL) AND
   PROCESS_INSTANCE.NAME='CustomerOrder'",
   (String)null, (Integer)null, (TimeZone)null);
```

   마찬가지로 조인 조건의 평가 또는 forEach 카운터가 실패하여 중지된 활동을 나열할 수 있습니다.

   - 실패한 조인 조건에 대해 다음 표현식을 사용하십시오.

     ```
     ACTIVITY.STOP_REASON.STOP_REASON_ACTIVATION_FAILED
     ```

- 실패한 forEach 카운터에 대해 다음 표현식을 사용하십시오.

```
ACTIVITY.STOP_REASON.STOP_REASON_IMPLEMENTATION_FAILED AND
(ACTIVITY.KIND = ACTIVITY.KIND.KIND_FOR_EACH_SERIAL OR
   ACTIVITY.KIND = ACTIVITY.KIND.KIND_FOR_EACH_PARALLEL)
```

이 조치는 루프 조건의 평가가 실패하여 중지된 CustomerOrder 프로세스 인스턴스의 활동을 리턴합니다.

2. 루프 조건의 값(예: true)을 제공하십시오.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);

  process.forceLoopCondition(aiid, true);
}
```

이 조치는 활동에 대한 루프 조건의 값을 true로 설정하고 프로세스 인스턴스 탐색을 계속합니다.

마찬가지로 조인 조건의 값(`process.forceJoinCondition(aiid, true);`) 또는 forEach 활동 카운터의 값(`process.forceForEachCounterValues(aiid, 1, 5, new Integer(2));`)을 설정할 수 있습니다.

중지된 활동과 연관된 상관 세트 갱신:

협업 세트는 웹 서비스 간의 Stateful 협업을 지원하는 데 사용됩니다. 이러한 경우에 Business Process Choreographer EJB API를 사용하여 프로세스 탐색을 계속할 수 있도록 표현식에 올바른 값을 제공할 수 있습니다.

이 태스크 정보

중지됨 상태의 활동은 다음 이유 중 하나 때문에 연관된 상관 세트를 갱신해야 할 수 있습니다.

- 상관 세트가 평가될 때 예외가 발생했습니다. 상관 세트를 초기화해야 하지만 이미 초기화되었습니다.
- 상관 세트가 평가될 때 예외가 발생했습니다. 상관 세트를 초기화하지 않아야 하지만 값이 설정되지 않았습니다. 예를 들어, 활동 초기화를 건너뛴 경우에 발생할 수 있습니다.
- 활동을 재시도해야 합니다. 활동이 상관 세트를 초기화한 경우에는 forceRetry 메소드를 호출하기 전에 상관 세트를 초기화 취소하거나 변경할 수 있습니다.
- 활동을 완료해야 합니다. 활동이 상관 세트를 초기화한 경우에는 forceComplete 메소드를 호출하기 전에 상관 세트를 초기화 취소하거나 변경할 수 있습니다.

프로세스의 상관 세트 인스턴스 또는 활동 인스턴스를 검색할 수 있습니다. 다음 예제는 상관 세트 인스턴스를 초기화 또는 초기화 취소하는 방법을 나타냅니다.

프로시저

1. 중지 상태에 있는 중지된 활동을 표시하십시오.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

   이 조치는 CustomerOrder 프로세스 인스턴스에 대한 중지된 활동을 리턴합니다.

2. 활동에 대해 정의된 상관 세트 인스턴스를 검색하십시오.

```
AIID aiid = null;

List correlationSet = null;

if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);

   ActivityInstanceData activity = process.getActivityInstance(aiid);

   correlationSet = process.getCorrelationSetInstances
                    (aiid, activity.getInputMessageTypeName());
}
```

3. 상관 세트(예: MyCorrelationSet)를 초기화 취소하십시오.

```
for ( int i=0; i<correlationSet.size(); i++ )
{
  CorrelationSetInstanceData correlationSetInstance =
            (CorrelationSetInstanceData)correlationSet.get(i);

  if ( correlationSetInstance.isInitialized() &&
       correlationSetInstance.getCorrelationSetName().equals("MyCorrelationSet") )
  {
    process.uninitializeCorrelationSet
       ( activity.getProcessInstanceID(), correlSetInstance.getCorrelationSetName() );
  }
}
```

   이 조치는 상관 세트 MyCorrelationSet를 초기화 취소합니다.

4. 상관 세트(예: MyCorrelationSet)를 초기화하십시오. 이 예제에서는 상관 세트의 문자열 값 특성이 설정됩니다.

```
for ( int i=0; i<correlationSet.size(); i++ )
{
  CorrelationSetInstanceData correlationSetInstance =
            (CorrelationSetInstanceData)correlationSet.get(i);

  if ( correlationSetInstance.getCorrelationSetName().equals("MyCorrelationSet") )
  {
    List correlationSetProperties =
                    correlationSetInstance.getCorrelationSetProperties();
    for ( int j=0; j<correlationSetProperties.size(); j++ )
    {
      CorrelationPropertyInstanceData property =
                (CorrelationPropertyInstanceData)correlationSetProperties.get(j);

      if ( property.getPropertyName().equals("MyProperty") )
      {
        property.setValue("NewValue");

        process.initializeCorrelationSet
           ( activity.getProcessInstanceID(), correlationSetInstance );
```

```
        }
      }
    }
  }
```

이 조치는 상관 세트 MyCorrelationSet의 문자열 값 특성 MyProperty를 초기화합니다.

BusinessFlowManagerService 인터페이스

BusinessFlowManagerService 인터페이스는 클라이언트 응용프로그램에 피호출되는 비즈니스 프로세스 기능을 표시합니다.

BusinessFlowManagerService 인터페이스에서 호출할 수 있는 메소드는 프로세스 상태 또는 해당 메소드를 포함하는 응용프로그램을 사용하는 사용자의 활동 및 권한에 따라 다릅니다. 비즈니스 프로세스 오브젝트를 조작하는 기본 메소드는 다음과 같습니다. 이 메소드 및 BusinessFlowManagerService 인터페이스에서 사용할 수 있는 기타 메소드에 대한 자세한 정보는 com.ibm.bpe.api 패키지의 Javadoc을 참조하십시오.

프로세스 템플리트

프로세스 템플리트는 비즈니스 프로세스의 스펙을 포함하는 버전이 지정되고 전개되며 설치된 프로세스 모델입니다. 알맞은 요청(예: sendMessage())을 발행하여 프로세스 템플리트를 인스턴스화하고 시작할 수 있습니다. 프로세스 인스턴스는 서버에서 자동으로 실행됩니다.

표 60. 프로세스 템플리트의 API 메소드

메소드	설명
getProcessTemplate	지정된 프로세스 템플리트를 검색합니다.
queryProcessTemplates	데이터베이스에 저장되는 프로세스 템플리트를 검색합니다.

프로세스 인스턴스

다음 API 메소드는 프로세스 인스턴스 시작과 관련이 있습니다.

표 61. 프로세스 인스턴스 시작과 관련된 API 메소드

메소드	설명
call	마이크로플로우를 작성하고 실행합니다.
callWithReplyContext	고유 시작 서비스가 있는 마이크로플로우 또는 지정된 프로세스 템플리트의 고유 시작 서비스가 있는 장기 실행 프로세스를 작성하고 실행합니다. 이 호출은 비동기적으로 결과를 기다립니다.
callWithUISettings	마이크로플로우를 작성 및 실행하고 출력 메시지 및 클라이언트 사용자 인터페이스(UI) 설정을 리턴합니다.

표 61. 프로세스 인스턴스 시작과 관련된 API 메소드 (계속)

메소드	설명
initiate	프로세스 인스턴스를 작성하고 프로세스 인스턴스의 처리를 시작합니다. 장기 실행 프로세스에 이 메소드를 사용하십시오. 또한 해제하거나 잊어버리려는 마이크로플로우에 대해서도 이 메소드를 사용할 수 있습니다.
initiateAndSuspend	프로세스 인스턴스를 작성하지만 프로세스 인스턴스의 처리를 즉시 일시중단합니다.
initiateAndClaimFirst	프로세스 인스턴스를 작성하고 첫 번째 인라인 휴먼 타스크를 청구합니다.
sendMessage	지정된 메시지를 지정된 활동 서비스 및 프로세스 인스턴스에 전송합니다. 동일한 상관 세트 값을 가진 프로세스 인스턴스가 없는 경우에는 새로 작성됩니다. 프로세스에는 고유하거나 고유하지 않은 시작 서비스가 있을 수 있습니다.
getStartActivities	지정된 프로세스 템플리트에서 프로세스 인스턴스를 시작할 수 있는 활동 정보를 리턴합니다.
getActivityServiceTemplate	지정된 활동 서비스 템플리트를 검색합니다.

표 62. 프로세스 인스턴스의 라이프사이클을 제어하는 API 메소드

메소드	설명
suspend	실행 중 또는 실패 상태의 장기 실행, 맨 위 레벨 프로세스 인스턴스의 실행을 일시중단합니다.
resume	일시중단 상태의 장기 실행, 맨 위 레벨 프로세스 인스턴스의 실행을 재개합니다.
restart	완료, 실패 또는 종료 상태의 장기 실행, 맨 위 레벨 프로세스 인스턴스를 다시 시작합니다.
forceTerminate	지정된 맨 위 레벨 프로세스 인스턴스, 하위 자율성이 있는 서브프로세스 및 실행 중이거나 청구되거나 대기 중인 활동을 종료합니다.
delete	지정된 맨 위 레벨 프로세스 인스턴스 및 하위 자율성이 있는 서브프로세스를 삭제합니다.
query	검색 기준에 일치하는 데이터베이스의 특성을 검색합니다.
queryEntities	조회 테이블을 사용하여 데이터베이스에서 검색 기준에 일치하는 특성을 검색합니다.
getWaitingActivities	활동을 계속 처리할 수 있도록 메시지를 대기 중인 활동에 대한 정보를 리턴합니다.
migrate	프로세스 인스턴스를 프로세스 모델의 지정된 새 버전으로 이주합니다.

활동

호출 활동의 경우 프로세스 모델에서 이 활동이 오류 상태에서 계속되도록 지정할 수 있습니다. continueOnError 플래그가 false로 설정되어 처리되지 않은 오류가 발생한 경

우 활동이 중지 상태가 됩니다. 그런 다음, 프로세스 관리자가 활동을 복구할 수 있습니다. 예를 들어, continueOnError 플래그 및 연관된 복구 기능은 호출 활동이 실패하기도 하는 장기 실행 프로세스에 사용될 수 있지만 보상 모델화 및 결함 처리에 너무 많은 노력이 필요합니다.

활동에 대한 작업 및 복구에 다음 메소드를 사용할 수 있습니다.

표 63. 활동 인스턴스의 라이프사이클 제어용 API 메소드

메소드	설명
claim	사용자가 활동 작업을 할 수 있도록 준비 활동 인스턴스를 청구합니다.
cancelClaim	활동 인스턴스의 청구를 취소합니다.
complete	활동 인스턴스를 완료합니다.
completeAndClaimSuccessor	활동 인스턴스를 완료하고 로그온한 사용자의 동일한 프로세스 인스턴스에 있는 다음 활동 인스턴스를 청구합니다.
forceComplete	다음을 강제로 완료합니다. • 실행 또는 중지 상태의 활동 인스턴스. • 준비 또는 청구됨 상태의 휴먼 타스크 활동. • 대기 중 상태의 Wait 활동.
forceRetry	다음을 강제로 반복합니다. • 실행 또는 중지 상태의 활동 인스턴스. • 준비 또는 청구됨 상태의 휴먼 타스크 활동.
forceNavigate, forceForEach, forceLoop, forceJoin	이 메소드는 중지된 활동의 탐색을 강제 실행합니다.
skip	활동 처리를 건너뜁니다.
jump	한 활동에서 다른 활동으로 점프합니다.
query	검색 기준에 일치하는 데이터베이스의 특성을 검색합니다.
queryEntities	조회 테이블을 사용하여 데이터베이스에서 검색 기준에 일치하는 특성을 검색합니다.

변수 및 사용자 정의 특성

인터페이스는 변수값 검색 및 설정에 사용되는 get 및 set 메소드를 제공합니다. 이름 지정된 특성을 프로세스 및 활동 인스턴스와 연관시키고 프로세스 및 활동 인스턴스에서 이름 지정된 특성을 검색할 수 있습니다. 사용자 정의 특성 이름 및 값은 java.lang.String 유형이어야 합니다.

표 64. 변수 및 사용자 정의 특성에 대한 API 메소드

메소드	설명
getVariable	지정된 변수를 검색합니다.
setVariable	지정된 변수를 설정합니다.
getCustomProperty	지정된 활동 또는 프로세스 인스턴스의 이름 지정된 사용자 정의 특성을 검색합니다.

표 64. 변수 및 사용자 정의 특성에 대한 API 메소드 (계속)

메소드	설명
getCustomProperties	지정된 활동 또는 프로세스 인스턴스의 사용자 정의 특성을 검색합니다.
getCustomPropertyNames	지정된 활동 또는 프로세스 인스턴스의 사용자 정의 특성 이름을 검색합니다.
setCustomProperty	지정된 활동 또는 프로세스 인스턴스의 사용자 정의 특정 값을 저장합니다.

휴먼 타스크용 응용프로그램 개발

타스크는 컴포넌트에서 휴먼을 서비스로 호출하거나 휴먼이 서비스를 호출하는 방법입니다. 휴먼 타스크에 대한 일반 응용프로그램의 예가 제공됩니다.

이 태스크 정보

휴먼 타스크 관리자 API에 대한 자세한 내용은 com.ibm.task.api 패키지의 Javadoc을 참조하십시오.

동기 인터페이스를 호출하는 호출 타스크 시작

호출 타스크는 SCA(Service Component Architecture) 컴포넌트와 연관됩니다. 타스크는 시작될 때 SCA 컴포넌트를 호출합니다. 연관된 SCA 컴포넌트를 동기식으로 호출할 수 있는 경우에만 호출 타스크를 동기식으로 시작하십시오.

이 태스크 정보

예를 들어, 이러한 SCA 컴포넌트는 마이크로플로우 또는 간단한 Java 클래스로 구현할 수 있습니다.

이 시나리오는 타스크 템플리트의 인스턴스를 작성하고 몇가지 고객 데이터를 전달합니다. 타스크는 양방향 조작이 리턴할 때까지 실행 상태로 남아 있습니다. 타스크 OrderNo의 결과가 호출자에게 리턴됩니다.

프로시저

1. 옵션: 타스크 템플리트를 표시하여 실행하려는 호출 타스크의 이름을 찾으십시오.

   타스크의 이름을 이미 아는 경우 이 단계는 선택적입니다.

   ```
   TaskTemplate[] taskTemplates = task.queryTaskTemplates
   ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
    "TASK_TEMPL.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   결과는 이름별로 정렬됩니다. 조회에서 처음 50개의 정렬된 원래 템플리트를 포함하는 배열이 리턴됩니다.

2. 해당 유형의 입력 메시지를 작성하십시오.

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage
( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null &&
input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. 타스크를 작성한 후 타스크를 동기적으로 실행하십시오.

   동기적으로 실행할 타스크는 반드시 양방향 조작이어야 합니다. 이 예에서는 createAndCallTask 메소드를 사용하여 타스크를 작성하고 실행합니다.

```
ClientObjectWrapper output =
task.createAndCallTask( template.getName(),
                    template.getNamespace(),
                  input);
```

4. 타스크의 결과를 분석하십시오.

```
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject()
instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

비동기 인터페이스를 호출하는 호출 타스크 시작

호출 타스크는 SCA(Service Component Architecture) 컴포넌트와 연관됩니다. 타스크는 시작될 때 SCA 컴포넌트를 호출합니다. 연관된 SCA 컴포넌트를 비동기로 호출할 수 있는 경우에만 호출 타스크를 비동기로 시작하십시오.

이 태스크 정보

예를 들어, 이러한 SCA 컴포넌트는 장기 실행 프로세스 또는 단방향 조작으로 구현할 수 있습니다.

이 시나리오는 타스크 템플리트의 인스턴스를 작성하고 몇가지 고객 데이터를 전달합니다.

프로시저

1. 옵션: 타스크 템플리트를 표시하여 실행하려는 호출 타스크의 이름을 찾으십시오.

   타스크의 이름을 이미 아는 경우 이 단계는 선택적입니다.

```
TaskTemplate[] taskTemplates
= task.queryTaskTemplates
  ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
   "TASK_TEMPL.NAME",
    new Integer(50),
    (TimeZone)null);
```

결과는 이름별로 정렬됩니다. 조회에서 처음 50개의 정렬된 원래 템플리트를 포함하는 배열이 리턴됩니다.

2. 해당 유형의 입력 메시지를 작성하십시오.

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage(
template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. 타스크를 작성하고 비동기적으로 실행하십시오.

이 예에서는 createAndStartTask 메소드를 사용하여 타스크를 작성하고 실행합니다.

```
task.createAndStartTask( template.getName(),
                         template.getNamespace(),
                         input,
                         (ReplyHandlerWrapper)null);
```

타스크 인스턴스 작성 및 시작

이 시나리오는 공동 작업 타스크(API에서 *휴먼 타스크*라고도 함)를 정의하는 타스크 템플리트의 인스턴스를 작성하고 타스크 인스턴스를 시작하는 방법을 보여줍니다.

프로시저

1. 옵션: 타스크 템플리트를 나열하여 실행할 협업 타스크의 타스크 템플리트 ID(TKTID)를 찾으십시오.

타스크 템플리트 ID를 이미 아는 경우 이 단계는 선택적입니다.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_HUMAN",
 "TASK_TEMPL.NAME",
  new Integer(50),
  (TimeZone)null);
```

결과는 이름별로 정렬됩니다. 조회는 처음 50개의 정렬된 타스크 템플리트를 포함하는 배열을 리턴합니다.

2. 해당 유형의 입력 메시지를 작성하십시오.

```
TaskTemplate template = taskTemplates[0];

// create a message for the selected task
ClientObjectWrapper input = task.createInputMessage
( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
```

3. 공동 작업 타스크를 작성하고 시작하십시오. 이 예제에는 응답 핸들러가 지정되어 있지 않습니다.

   이 예에서는 createAndStartTask 메소드를 사용하여 타스크를 작성하고 시작합니다.

```
TKIID tkiid = task.createAndStartTask( template.getName(),
                          template.getNamespace(),
                            input,
                       (ReplyHandlerWrapper)null);
```

   작업 항목이 타스크 인스턴스를 인식하는 사용자에 대해 작성됩니다. 예를 들어, 잠재적 사용자는 새 타스크 인스턴스를 청구할 수 있습니다.

4. 타스크 인스턴스를 청구하십시오.

```
ClientObjectWrapper input2 = task.claim(tkiid);
DataObject taskInput = null ;
if ( input2.getObject()!= null &&
input2.getObject() instanceof DataObject )
{
  taskInput = (DataObject)input2.getObject();
  // read the values
  ...
}
```

   타스크 인스턴스가 청구될 때 타스크의 입력 메시지가 리턴됩니다.

수행할 타스크 또는 공동 작업 타스크 처리

수행할 타스크(API에서 *참여 중인 타스크*라고도 함) 또는 공동 작업 타스크(API에서 *휴먼 타스크*라고도 함)는 작업 항목을 통해 조직의 다양한 개인에게 지정됩니다. 수행할 타스크 및 연관된 작업 항목은 예를 들어, 프로세스가 휴먼 타스크 활동을 탐색할 때 작성됩니다.

이 태스크 정보

잠재적 소유자 중 하나가 작업 항목과 연관된 타스크를 청구합니다. 이 사용자는 관련 정보를 제공하고 타스크를 완료해야 합니다.

프로시저

1. 작업이 가능한 로그온된 사용자에 속한 타스크를 표시하십시오.

```
QueryResultSet result =
     task.query("TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_READY AND
                (TASK.KIND = TASK.KIND.KIND_PARTICIPATING OR
                 TASK.KIND = TASK.KIND.KIND_HUMAN)AND
                 WORK_ITEM.REASON =
                   WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                 (String)null, (Integer)null, (TimeZone)null);
```

   이 조치는 로그온 사용자가 작업할 수 있는 타스크를 포함하는 결과 조회 세트를 리턴합니다.

2. 작업할 타스크를 청구하십시오.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper input = task.claim(tkiid);
  DataObject taskInput = null ;
  if ( input.getObject()!= null &&

  input.getObject() instanceof DataObject )
  {
    taskInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

   타스크가 청구될 때 타스크의 입력 메시지가 리턴됩니다.

3. 타스크의 작업이 완료되면 타스크를 완료하십시오.

   타스크는 성공적으로 완료되거나 결함 메시지와 함께 완료될 수 있습니다. 타스크가 성공적인 경우 출력 메시지가 전달됩니다. 타스크가 성공하지 못한 경우 결함 메시지가 전달됩니다. 이 조치에 해당하는 메시지를 작성해야 합니다.

   a. 타스크를 완료하려면 출력 메시지를 작성하십시오.

```
ClientObjectWrapper output =
      task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the task
task.complete(tkiid, output);
```

이 조치는 순번을 포함하는 출력 메시지를 설정합니다. 타스크는 완료 상태가 됩니다.

b. 결함이 발생할 때 타스크를 완료하려면 결함 메시지를 작성하십시오.

```
//retrieve the faults modeled for the task
List faultNames = task.getFaultNames(tkiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      task.createFaultMessage(tkiid, (String)faultNames.get(0));

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

task.complete(tkiid, (String)faultNames.get(0), myFault);
```

이 조치를 실행하면 오류 코드를 포함하는 결함 메시지가 설정됩니다. 타스크는 실패 상태가 됩니다.

관련 개념


협업 타스크에 대한 상태 전이 다이어그램

협업 타스크는 다른 사용자를 위한 작업을 수행하는 사용자를 지원합니다. 협업 타스크의 라이프사이클에서 특정 상호작용은 특정 타스크 상태에서만 가능하며 이러한 상호작용은 타스크 상태에도 영향을 미칩니다.

타스크 인스턴스의 일시중단 및 재개

공동 작업 타스크 인스턴스 (API에서 *휴먼 타스크*라고도 함) 또는 수행할 타스크 인스턴스(API에서 *참여 중인 타스크*라고도 함)를 일시중단할 수 있습니다.

시작하기 전에

타스크 인스턴스는 준비 상태이거나 청구 상태일 수 있습니다. 또한 에스컬레이션 상태일 수 있습니다. 호출자는 타스크 인스턴스의 소유자, 작성자 또는 관리자여야 합니다.

이 태스크 정보

타스크 인스턴스가 실행 중일 때, 이것을 일시중단할 수 있습니다. 예를 들어, 타스크를 완료하는 데 필요한 정보를 모을 필요가 있을 때, 일시중단할 수 있습니다. 정보가 사용 가능한 경우, 타스크 인스턴스를 재개할 수 있습니다.

프로시저

1. 로그온 상태의 사용자가 청구한 타스크 목록을 가져오십시오.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                   "TASK.STATE

       = TASK.STATE.STATE_CLAIMED",
                                   (String)null,
                                   (Integer)null,
                                   (TimeZone)null);
```

   이 조치는 로그온 상태의 사용자가 청구한 타스크 목록을 포함하는 조회 결과 세트를 리턴합니다.

2. 타스크 인스턴스를 일시중단하십시오.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.suspend(tkiid);
}
```

   이 조치는 특정한 타스크 인스턴스를 일시중단합니다. 타스크 인스턴스는 일단중단 상태가 됩니다.

3. 프로세스 인스턴스를 재개하십시오.

```
task.resume( tkiid );
```

   이 조치는 타스크 인스턴스를 일시중단되기 이전의 상태로 돌려놓습니다.

타스크 결과 분석

수행할 타스크(API에서 *참여 중인* 타스크라고도 함) 또는 공동 작업 타스크(API에서 *휴먼 타스크*라고도 함)는 비동기로 실행합니다. 타스크가 시작될 때 응답 핸들러가 지정되었으면 타스크 완료 시 출력 메시지가 자동으로 리턴됩니다. 응답 핸들러가 지정되지 않았으면 메시지는 명시적으로 검색되어야 합니다.

이 태스크 정보

타스크 인스턴스를 파생시킨 타크스 템플리트가 파생된 타스크 인스턴스의 자동 삭제를 지정하지 않는 경우에만 타스크의 결과가 데이터베이스에 저장됩니다.

프로시저

타스크 결과를 분석하십시오.
이 예는 완료된 타스크의 순서 번호를 확인하는 방법을 보여줍니다.

```
QueryResultSet result =
task.query("DISTINCT TASK.TKIID",
            "TASK.NAME = 'CustomerOrder' AND
            TASK.STATE = TASK.STATE.STATE_FINISHED",
           (String)null, (Integer)null, (TimeZone)null);
```

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper output = task.getOutputMessage(tkiid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject()

  instanceof DataObject)
  {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
  }
}
```

타스크 인스턴스 종료

타스크 인스턴스 종료는 복구 불가능 상태로 알려진 타스크 인스턴스를 종료할 수 있는 관리자 권한이 있는 사용자에게 필요합니다. 타스크 인스턴스는 즉시 종료되므로 예외 상황에서만 타스크 인스턴스를 종료해야 합니다.

프로시저

1. 종료할 타스크 인스턴스를 검색하십시오.

   ```
   Task taskInstance = task.getTask(tkiid);
   ```

2. 타스크 인스턴스를 종료하십시오.

   ```
   TKIID tkiid = taskInstance.getID();
   task.terminate(tkiid);
   ```

   타스크 인스턴스는 미결 서브프로세스 또는 활동을 대기하지 않고 즉시 종료됩니다.

타스크 인스턴스 삭제

인스턴스가 파생된 연관 타스크 템플리트에 지정된 경우에는 타스크 인스턴스가 완료될 때 자동으로 삭제됩니다. 다음 예에서는 완료되고 자동으로 삭제되지 않는 모든 타스크 인스턴스를 삭제하는 방법을 보여줍니다.

프로시저

1. 완료된 타스크 인스턴스를 표시하십시오.

   ```
   QueryResultSet result =
        task.query("DISTINCT TASK.TKIID",
                   "TASK.STATE = TASK.STATE.STATE_FINISHED",
                    (String)null, (Integer)null, (TimeZone)null);
   ```

   이 조치를 실행하면 완료된 타스크 인스턴스를 표시하는 결과 조회 세트가 리턴됩니다.

2. 완료된 타스크 인스턴스를 삭제하십시오.

```
while (result.next() )
{
 TKIID tkiid = (TKIID) result.getOID(1);
 task.delete(tkiid);
}
```

청구된 타스크 릴리스

잠재적 소유자가 타스크를 청구할 때 이 사용자는 타스크를 완료해야 합니다. 그러나 또다른 잠재적 소유자가 청구할 수 있도록 청구된 타스크를 해제해야할 수도 있습니다.

이 태스크 정보

관리자 권한이 있는 사용자는 상황에 따라 청구한 타스크를 해제할 필요가 있습니다. 이러한 상황은 예를 들면, 타스크가 완료되어야 하지만 타스크 소유자가 부재 중인 경우에 발생할 수 있습니다. 타스크 소유자도 청구된 타스크를 해제할 수 있습니다.

프로시저

1. 예를 들어, Smith와 같은 특정 사용자가 소유한 청구된 타스크를 목록으로 표시합니다.

   ```
   QueryResultSet result =
        task.query("DISTINCT TASK.TKIID",
                   "TASK.STATE = TASK.STATE.STATE_CLAIMED AND
                    TASK.OWNER = 'Smith'",
                    (String)null, (Integer)null, (TimeZone)null);
   ```

   이 조치는 지정된 사용자 Smith가 청구한 타스크를 표시하는 결과 조회 세트를 리턴합니다.

2. 청구된 타스크를 해제하십시오.

   ```
   if (result.size() > 0)
   {
     result.first();
     TKIID tkiid = (TKIID) result.getOID(1);
     task.cancelClaim(tkiid, true);
   }
   ```

   이 조치는 다른 잠재적 소유자 중 하나가 청구할 수 있도록 타스크를 준비 상태로 리턴합니다. 원래 소유자가 설정한 출력 또는 결함 데이터는 보존됩니다.

작업 항목 관리

활동 인스턴스 또는 타스크 인스턴스의 지속 시간동안 오브젝트에 연관된 사용자 설정은 사용자가 휴가이거나 새로운 직원이 고용되거나 워크로드가 다르게 분배되어야 하는 경우와 같은 상황에서는 변경될 수 있습니다. 이 변경사항을 허용하려면 작업 항목을 작성, 삭제 또는 전송할 응용프로그램을 개발할 수 있습니다.

이 태스크 정보

작업 항목은 특정 이유로 사용자 또는 사용자 그룹에 오브젝트를 지정하는 것입니다. 오브젝트는 일반적으로 휴먼 타스크 활동 인스턴스, 프로세스 인스턴스 또는 타스크 인스턴스입니다. 이유는 오브젝트에 대한 사용자의 역할에서 파생됩니다. 사용자는 오브젝트에 연관되어 여러 다른 역할을 가질 수 있고 이러한 각 역할에 대해 작업 항목이 작성되기 때문에 오브젝트에 여러 작업 항목이 있을 수 있습니다. 예를 들어, 수행할 타스크 인스턴스에는 관리자, 독자, 편집자 및 소유자 작업 항목이 동시에 있을 수 있습니다.

작업 항목을 관리하기 위해 수행할 수 있는 조치는 사용자가 가지고 있는 역할에 따라 다릅니다. 예를 들어, 관리자는 작업 항목을 작성, 삭제 및 전송할 수 있지만 타스크 소유자는 작업 항목을 전송할 수만 있습니다.

프로시저

- 작업 항목을 작성하십시오.

```
// query the task instance for which an additional
// administrator is to be specified
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // create the work item
  task.createWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_ADMINISTRATOR,"Smith");
}
```

  이 조치를 통해 관리자 역할을 보유하고 있는 사용자 Smith에 대해 작업 항목이 작성됩니다.

- 작업 항목을 삭제하십시오.

```
// query the task instance for which a work item is to be deleted
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // delete the work item
  task.deleteWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_READER,"Smith");
}
```

  이 조치를 통해 독자 역할을 보유하고 있는 사용자 Smith에 대한 작업 항목이 삭제됩니다.

- 작업 항목을 전송하십시오.

```
// query the task that is to be rescheduled
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.NAME='CustomerOrder' AND
                 TASK.STATE=TASK.STATE.STATE_READY AND
                 WORK_ITEM.REASON=WORK_ITEM.REASON.REASON

   _POTENTIAL_OWNER AND
                 WORK_ITEM.OWNER_ID='Miller'",
                 (String)null, (Integer)null, (TimeZone)null);
if ( result.size() > 0 )
{
  result.first();
  // transfer the work item from user Miller to user Smith
  // so that Smith can work on the task
  task.transferWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_POTENTIAL_OWNER

      ,"Miller","Smith");
}
```

이 조치를 통해 사용자 Smith에게 작업 항목을 전송하여 작업할 수 있도록 합니다.

런타임 시 타스크 템플리트 및 타스크 인스턴스 작성

보통 WebSphere Integration Developer와 같은 모델링 도구를 사용하여 타스크 템플리트를 빌드할 수 있습니다. 그런 다음, 타스크 템플리트를 WebSphere Process Server에 설치하고 이 템플리트로부터(예: Business Process Choreographer Explorer 사용) 인스턴스를 작성합니다. 그러나 또한 런타임 시 휴먼 또는 참여 중인 타스크 인스턴스나 템플리트를 작성할 수도 있습니다.

이 태스크 정보

예를 들어, 응용프로그램을 전개할 때 타스크 정의를 사용할 수 없는 경우, 워크플로우에 포함된 타스크가 아직 알려지지 않은 경우 또는 사용자 그룹 간의 일부 임시 협업을 다루는 타스크가 필요한 경우에 이를 수행할 수 있습니다.

com.ibm.task.api.TaskModel 클래스의 인스턴스를 작성해서 임시 수행할 작업 또는 협업 타스크를 모델화하고 이들을 사용하여 재사용 가능한 타스크 템플리트를 작성하거나 일회 실행 타스크 인스턴스를 직접 작성할 수 있습니다. TaskModel 클래스의 인스턴스를 작성하려면 팩토리 메소드 세트가 com.ibm.task.api.ClientTaskFactory 팩토리 클래스에서 사용 가능해야 합니다. 런타임 시 휴먼 타스크 모델화는 EMF(Eclipse Modeling Framework)에 기초합니다.

프로시저

1. createResourceSet 팩토리 메소드를 사용하여 org.eclipse.emf.ecore.resource.ResourceSet을 작성하십시오.

2. 옵션: 복잡한 메시지 유형을 사용하려는 경우 팩토리 메소드 getXSDFactory()를 사용하여 얻거나 loadXSDSchema 팩토리 메소드를 사용하여 기존 XML 스키마를 직접 가져올 수 있는 org.eclipse.xsd.XSDFactory를 사용하여 정의할 수 있습니다.

   WebSphere Process Server에서 복잡한 유형을 사용하려면 엔터프라이즈 응용프로그램의 일부로 유형을 전개하십시오.

3. javax.wsdl.Definition 유형의 WSDL(Web Services Definition Language) 정의를 작성하거나 가져오십시오.

   createWSDLDefinition 메소드를 사용하여 새 WSDL 정의를 작성할 수 있습니다. 그런 다음, 포트 유형 및 조작에 이를 추가할 수 있습니다. loadWSDLDefinition 팩토리 메소드를 사용하여 기존 WSDL 정의를 직접 가져올 수도 있습니다.

4. createTTask 팩토리 메소드를 사용하여 타스크 정의를 작성하십시오.

   보다 복잡한 타스크 요소를 추가하거나 조작하려는 경우 getTaskFactory 팩토리 메소드를 통해 검색할 수 있는 com.ibm.wbit.tel.TaskFactory 클래스를 사용할 수 있습니다.

5. createTaskModel 팩토리 메소드를 사용하여 타스크 모델을 작성하고 작성한 다른 모든 아티팩트를 집계하며 1단계에서 작성한 자원 번들에 이를 전달하십시오.
6. 옵션: TaskModel validate 메소드를 사용하여 모델의 유효성을 검증하십시오.

결과

TaskModel 매개변수가 있는 휴먼 타스크 관리자 EJB API create 메소드 중 하나를 사용하여 재사용 가능한 타스크 템플리트 또는 일회 실행 타스크 인스턴스를 작성하십시오.

단순 Java 유형을 사용하는 런타임 타스크 작성:

이 예는 해당 인터페이스에서 단순 Java 유형(예: String 오브젝트)만 사용하는 런타임 타스크를 작성합니다.

이 태스크 정보

이 예는 자원이 로드된 호출 엔터프라이즈 응용프로그램의 컨텍스트 내부에서만 실행됩니다.

프로시저

1. ClientTaskFactory에 액세스하여 새 타스크 모델의 정의를 포함시킬 자원 세트를 작성하십시오.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. WSDL 정의를 작성하고 조작 설명을 추가하십시오.

```
// create the WSDL interface
Definition definition = factory.createWSDLDefinition
         ( resourceSet, new QName(

  "http://www.ibm.com/task/test/", "test" ) );

// create a port type
PortType portType = factory.createPortType(
definition, "doItPT" );

// create an operation; the input and output
messages are of type String:
// a fault message is not specified
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://www.w3.org/2001/XMLSchema",

   "string" ),
          new QName( "http://www.w3.org/2001/XMLSchema",

   "string" ),
          (Map)null );
```

3. 새 휴먼 타스크의 EMF 모델을 작성하십시오.

   타스크 인스턴스를 작성 중이면 유효 시작 날짜(UTCDate)가 필요하지 않습니다.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate(

           "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   이 단계는 타스크 모델의 특성을 기본값으로 초기화합니다.

4. 휴먼 타스크 모델의 특성을 수정하십시오.

```
// use the methods from the com.ibm.wbit.tel
package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify
composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);
```

```
// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 모든 자원 정의가 들어 있는 타스크 모델을 작성하십시오.

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

6. 타스크 모델의 유효성을 검증한 후 발견된 모든 유효성 검증 문제점을 정정하십시오.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

7. 런타임 타스크 인스턴스 또는 템플리트를 작성하십시오.

   HumanTaskManagerService 인터페이스를 사용하여 타스크 인스턴스 또는 타스크 템플리트를 작성하십시오. 응용프로그램이 단순 Java 유형만 사용하므로 응용프로그램 이름을 지정할 필요가 없습니다.

   - 다음 스니펫은 타스크 인스턴스를 작성합니다.

     ```
     atask.createTask( taskModel, (String)null, "HTM" );
     ```

   - 다음 스니펫은 타스크 템플리트를 작성합니다.

     ```
     task.createTaskTemplate( taskModel, (String)null );
     ```

결과

런타임 인스턴스가 작성되면 시작할 수 있습니다. 런타임 타스크 템플리트가 작성되면 템플리트로부터 타스크 인스턴스를 작성할 수 있습니다.

복합 유형을 사용하는 런타임 타스크 작성:

이 예는 해당 인스턴스에서 복합 유형을 사용하는 런타임 타스크를 작성합니다. 복합 유형은 이미 정의되어 있습니다. 즉, 클라이언트의 로컬 파일 시스템에 복합 유형의 설명이 있는 XSD 파일이 들어 있습니다.

이 태스크 정보

이 예는 자원이 로드된 호출 엔터프라이즈 응용프로그램의 컨텍스트 내부에서만 실행됩니다.

프로시저

1. ClientTaskFactory에 액세스하여 새 타스크 모델의 정의를 포함시킬 자원 세트를 작성하십시오.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. 복합 유형의 XSD 정의를 자원 세트에 추가하여 조작을 정의할 때 이를 사용할 수 있도록 하십시오.

   파일은 코드가 실행되는 위치에 따라 위치합니다.

```
factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
```

3. WSDL 정의를 작성하고 조작 설명을 추가하십시오.

```
// create the WSDL interface
Definition definition = factory.createWSDLDefinition
         ( resourceSet, new QName(

  "http://www.ibm.com/task/test/", "test" ) );

// create a port type
PortType portType = factory.createPortType
( definition, "doItPT" );

// create an operation; the input message is an
InputBO and
// the output message an OutputBO;
// a fault message is not specified
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://Input", "InputBO" ),
          new QName( "http://Output", "OutputBO" ),
          (Map)null );
```

4. 새 휴먼 타스크의 EMF 모델을 작성하십시오.

   타스크 인스턴스를 작성 중이면 유효 시작 날짜(UTCDate)가 필요하지 않습니다.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate(

           "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   이 단계는 타스크 모델의 특성을 기본값으로 초기화합니다.

5. 휴먼 타스크 모델의 특성을 수정하십시오.

```
// use the methods from the
com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify
composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);
```

```
// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. 모든 자원 정의가 들어 있는 타스크 모델을 작성하십시오.

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

7. 타스크 모델의 유효성을 검증한 후 발견된 모든 유효성 검증 문제점을 정정하십시오.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

8. 런타임 타스크 인스턴스 또는 템플리트를 작성하십시오.

   HumanTaskManagerService 인터페이스를 사용하여 타스크 인스턴스 또는 타스크 템플리트를 작성하십시오. 데이터 유형 정의가 들어 있는 응용프로그램 이름을 제공하여 액세스할 수 있도록 해야 합니다. 응용프로그램에는 더미 타스크 또는 프로세스가 들어 있어 Business Process Choreographer가 이 응용프로그램을 로드하도록 해야 합니다.

   - 다음 스니펫은 타스크 인스턴스를 작성합니다.

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - 다음 스니펫은 타스크 템플리트를 작성합니다.

     ```
     task.createTaskTemplate( taskModel, "BOapplication" );
     ```

결과

런타임 인스턴스가 작성되면 시작할 수 있습니다. 런타임 타스크 템플리트가 작성되면 템플리트로부터 타스크 인스턴스를 작성할 수 있습니다.

기존 인터페이스를 사용하는 런타임 타스크 작성:

이 예는 이미 정의된 인터페이스를 사용하는 런타임 타스크를 작성합니다. 즉, 클라이언트의 로컬 파일 시스템에 인터페이스 설명이 있는 파일이 들어 있습니다.

이 태스크 정보

이 예는 자원이 로드된 호출 엔터프라이즈 응용프로그램의 컨텍스트 내부에서만 실행됩니다.

프로시저

1. ClientTaskFactory에 액세스하여 새 타스크 모델의 정의를 포함시킬 자원 세트를 작성하십시오.

   ```
   ClientTaskFactory factory =
   ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. WSDL 정의 및 사용자 조작의 설명에 액세스하십시오.

인터페이스 설명은 코드가 실행되는 위치에 따라 위치합니다.

```
Definition definition = factory.loadWSDLDefinition(
                          resourceSet, "interface.wsdl"  );
PortType portType = definition.getPortType(
                     new QName( definition.getTargetNamespace(),

       "doItPT" ) );
Operation operation = portType.getOperation
                               ("doIt", (String)null, (String)null);
```

3. 새 휴먼 타스크의 EMF 모델을 작성하십시오.

   타스크 인스턴스를 작성 중이면 유효 시작 날짜(UTCDate)가 필요하지 않습니다.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate(

           "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   이 단계는 타스크 모델의 특성을 기본값으로 초기화합니다.

4. 휴먼 타스크 모델의 특성을 수정하십시오.

```
// use the methods from the com.ibm.wbit.tel
package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 모든 자원 정의가 들어 있는 타스크 모델을 작성하십시오.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. 타스크 모델의 유효성을 검증한 후 발견된 모든 유효성 검증 문제점을 정정하십시오.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. 런타임 타스크 인스턴스 또는 템플리트를 작성하십시오.

HumanTaskManagerService 인터페이스를 사용하여 타스크 인스턴스 또는 타스크 템플리트를 작성하십시오. 데이터 유형 정의가 들어 있는 응용프로그램 이름을 제공하여 액세스할 수 있도록 해야 합니다. 응용프로그램에는 더미 타스크 또는 프로세스가 들어 있어 Business Process Choreographer가 이 응용프로그램을 로드하도록 해야 합니다.

- 다음 스니펫은 타스크 인스턴스를 작성합니다.

```
task.createTask( taskModel, "BOapplication", "HTM" );
```

- 다음 스니펫은 타스크 템플리트를 작성합니다.

```
task.createTaskTemplate( taskModel, "BOapplication" );
```

결과

런타임 인스턴스가 작성되면 시작할 수 있습니다. 런타임 타스크 템플리트가 작성되면 템플리트로부터 타스크 인스턴스를 작성할 수 있습니다.

호출 응용프로그램의 인터페이스를 사용하는 런타임 타스크 작성:

이 예는 호출 응용프로그램에 포함된 인터페이스를 사용하는 런타임 타스크를 작성합니다. 예를 들어, 런타임 타스크는 비즈니스 프로세스의 Java 스니펫에 작성되며 프로세스 응용프로그램의 인터페이스를 사용합니다.

이 태스크 정보

이 예는 자원이 로드된 호출 엔터프라이즈 응용프로그램의 컨텍스트 내부에서만 실행됩니다.

프로시저

1. ClientTaskFactory에 액세스하여 새 타스크 모델의 정의를 포함시킬 자원 세트를 작성하십시오.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();

// specify the context class loader so that
following resources are found
ResourceSet resourceSet = factory.createResourceSet
              ( Thread.currentThread().getContextClassLoader() );
```

2. WSDL 정의 및 사용자 조작의 설명에 액세스하십시오.

   포함하고 있는 패키지 JAR 파일에 경로를 지정하십시오.

```
Definition definition = factory.loadWSDLDefinition( resourceSet,
                            "com/ibm/workflow/metaflow/interface.wsdl" );
                        PortType portType = definition.getPortType(
                        new QName( definition.getTargetNamespace(),

        "doItPT" ) );
Operation operation = portType.getOperation
                                    ("doIt", (String)null, (String)null);
```

3. 새 휴먼 타스크의 EMF 모델을 작성하십시오.

   타스크 인스턴스를 작성 중이면 유효 시작 날짜(UTCDate)가 필요하지 않습니다.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate(

          "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   이 단계는 타스크 모델의 특성을 기본값으로 초기화합니다.

4. 휴먼 타스크 모델의 특성을 수정하십시오.

```
// use the methods from the com.ibm.wbit.tel package, for example,
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// retrieve the task factory to create or modify
composite task elements
TaskFactory taskFactory = factory.getTaskFactory();

// specify escalation settings
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// create escalationReceiver and add verb
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// create escalation and add escalation receiver
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. 모든 자원 정의가 들어 있는 타스크 모델을 작성하십시오.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. 타스크 모델의 유효성을 검증한 후 발견된 모든 유효성 검증 문제점을 정정하십시오.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. 런타임 타스크 인스턴스 또는 템플리트를 작성하십시오.

   HumanTaskManagerService 인터페이스를 사용하여 타스크 인스턴스 또는 타스크 템플리트를 작성하십시오. 데이터 유형 정의가 들어 있는 응용프로그램 이름을 제공하여 액세스할 수 있도록 해야 합니다.

   - 다음 스니펫은 타스크 인스턴스를 작성합니다.

```
task.createTask( taskModel, "WorkflowApplication", "HTM" );
```

   - 다음 스니펫은 타스크 템플리트를 작성합니다.

```
task.createTaskTemplate( taskModel, "WorkflowApplication" );
```

결과

런타임 인스턴스가 작성되면 시작할 수 있습니다. 런타임 타스크 템플리트가 작성되면 템플리트로부터 타스크 인스턴스를 작성할 수 있습니다.

HumanTaskManagerService 인터페이스

HumanTaskManagerService 인터페이스는 로컬 또는 원격 클라이언트가 호출할 수 있는 타스크 관련 기능을 표시합니다.

호출할 수 있는 메소드는 타스크 상태와 이 메소드를 포함하는 응용프로그램을 사용하는 사용자의 권한에 따라 다릅니다. 타스크 오브젝트를 조정하는 기본 메소드가 여기에 표시됩니다. 이 메소드와 HumanTaskManagerService 인터페이스에서 사용할 수 있는 기타 메소드에 대한 자세한 정보는 com.ibm.task.api 패키지의 Javadoc을 참조하십시오.

타스크 템플리트

타스크 템플리트에 대해 작업할 경우 다음 메소드를 사용할 수 있습니다.

표 65. 타스크 템플리트의 API 메소드

메소드	설명
getTaskTemplate	지정된 타스크 템플리트를 검색합니다.
createTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성합니다.
createAndCallTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성 및 실행하고 그 결과를 동기적으로 대기합니다.
createAndStartTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성 및 시작합니다.
createAndStartTaskAsSubtask	타스크 인스턴스를 지정된 타스크의 서브타스크로 작성하고 시작합니다.
createInputMessage	지정된 타스크 템플리트에 대한 입력 메시지를 작성합니다. 예를 들어, 타스크 시작에 사용되는 메시지를 작성합니다.
queryTaskTemplates	데이터베이스에 저장되는 타스크 템플리트를 검색합니다.

타스크 인스턴스

타스크 인스턴스에 대해 작업할 경우 다음 메소드를 사용할 수 있습니다.

표 66. 타스크 인스턴스의 API 메소드

메소드	설명
getTask	타스크 인스턴스를 검색합니다. 타스크 인스턴스는 모든 상태가 될 수 있습니다.

표 66. 타스크 인스턴스의 API 메소드 (계속)

메소드	설명
callTask	호출 타스크를 동기식으로 시작합니다.
startTask	이미 작성된 타스크를 시작합니다.
startTaskAsSubtask	타스크를 타스크 인스턴스의 서브타스크로 시작합니다.
suspend	공동 작업 또는 수행할 타스크를 일시중단합니다.
resume	공동 작업 또는 수행할 타스크를 재개합니다.
restart	타스크 인스턴스를 다시 시작합니다.
terminate	지정된 타스크 인스턴스를 종료합니다. 호출 타스크가 종료되면 이 조치는 호출된 서비스에 영향을 주지 않습니다.
delete	지정된 타스크 인스턴스를 삭제합니다.
claim	처리에 대한 타스크를 청구합니다.
update	타스크 인스턴스를 갱신합니다.
complete	타스크 인스턴스를 완료합니다.
completeWithFollowOnTask	타스크 인스턴스를 완료하고 후속타스크를 시작합니다.
cancelClaim	다른 잠재적 소유자가 해당 타스크 인스턴스를 사용할 수 있도록 청구된 타스크 인스턴스를 해제합니다.
createWorkItem	타스크 인스턴스의 작업 항목을 작성합니다.
transferWorkItem	작업 항목을 특정 소유자에게 전송합니다.
deleteWorkItem	작업 항목을 삭제합니다.

에스컬레이션

다음 메소드를 에스컬레이션 작업에 사용할 수 있습니다.

표 67. 에스컬레이션 작업에 사용되는 API 메소드

메소드	설명
getEscalation	지정된 에스컬레이션 인스턴스를 검색합니다.
triggerEscalation	수동으로 에스컬레이션을 트리거합니다.

사용자 정의 특성

타스크, 타스크 템플리트 및 에스컬레이션 모두 사용자 정의 특성을 가질 수 있습니다. 인터페이스는 get 및 set 메소드를 제공하여 사용자 정의 특성 값을 검색하고 설정합니다. 또한 이름 지정된 특성을 타스크 인스턴스와 연관시키고 타스크 인스턴스에서 이름 지정된 특성을 검색할 수 있습니다. 사용자 정의 특성 이름 및 값은 java.lang.String 유형이어야 합니다. 다음 메소드는 타스크, 타스크 템플리트 및 에스컬레이션에 사용할 수 있습니다.

표 68. 변수 및 사용자 정의 특성에 대한 API 메소드

메소드	설명
getCustomProperty	지정된 타스크 인스턴스의 이름 지정된 사용자 정의 특성을 검색합니다.
getCustomProperties	지정된 타스크 인스턴스의 사용자 정의 특성을 검색합니다.
getCustomPropertyNames	타스크 인스턴스의 사용자 정의 특성 이름을 검색합니다.
setCustomProperty	지정된 타스크 인스턴스의 사용자 정의 고유값을 저장합니다.

비즈니스 프로세스 및 휴먼 타스크용 응용프로그램 개발

사용자가 대부분의 비즈니스 프로세스 시나리오에 관련됩니다. 예를 들어, 비즈니스 프로세스는 프로세스가 시작되거나 관리될 때 또는 휴먼 타스크 활동이 수행될 때 사용자 상호작용이 필요합니다. 이 시나리오를 지원하려면 비즈니스 플로우 관리자 API 및 휴먼 타스크 관리자 API를 모두 사용해야 합니다.

이 태스크 정보

비즈니스 프로세스 시나리오의 사용자를 호출하려면 다음 타스크 종류를 비즈니스 프로세스에 포함할 수 있습니다.

- 인라인 호출 타스크(API에서 *오리지네이팅 타스크*라고도 함)

  모든 수신 활동, 선택 활동의 각 onMessage 요소 및 이벤트 핸들러의 각 onEvent 요소에 호출 타스크를 제공할 수 있습니다. 그러면 이 타스크는 프로세스를 시작하고 실행 중인 프로세스 인스턴스와 통신할 권한이 있는 사용자를 제어합니다.
- 관리 타스크

  관리 타스크를 제공하여 프로세스를 관리하거나 프로세스의 실패한 활동에 대한 관리 조작을 수행할 권한이 있는 사용자를 지정할 수 있습니다.
- 수행할 타스크(API에서 *참여 중인 타스크*라고도 함)

  수행할 타스크는 휴먼 타스크 활동을 구현합니다. 이 유형의 활동은 프로세스에 사용자를 포함시킵니다.

비즈니스 프로세스의 휴먼 타스크 활동은 개인이 비즈니스 프로세스 시나리오에서 수행하는 수행할 타스크를 나타냅니다. 비즈니스 플로우 관리자 API 및 휴먼 타스크 관리자 API를 모두 사용하여 이 시나리오를 실현할 수 있습니다.

- 비즈니스 프로세스는 수행할 타스크로 표시되는 휴먼 타스크 활동을 포함하여 프로세스에 속한 모든 활동의 컨테이너입니다. 프로세스 인스턴스가 작성되면 고유 오브젝트 ID(PIID)가 지정됩니다.

- 프로세스 인스턴스의 실행 중 휴먼 타스크 활동이 활성화되면 고유 오브젝트 ID(AIID)로 식별되는 활동 인스턴스가 작성됩니다. 동시에 오브젝트 ID(TKIID)로 식별되는 인라인 수행할 타스크 인스턴스도 작성됩니다. 타스크 인스턴스에 대한 휴먼 타스크 활동의 관계는 오브젝트 ID를 사용하여 설정됩니다.
  - 활동 인스턴스의 수행할 타스크 ID는 연관된 수행할 타스크의 TKIID로 설정됩니다.
  - 타스크 인스턴스의 포함 컨텍스트 ID는 연관된 활동 인스턴스를 포함하는 프로세스 인스턴스의 PIID로 설정됩니다.
  - 타스크 인스턴스의 상위 컨텍스트 ID는 연관된 활동 인스턴스의 AIID로 설정됩니다.
- 모든 인라인 수행할 타스크 인스턴스의 라이프사이클은 프로세스 인스턴스에서 관리합니다. 프로세스 인스턴스가 삭제되면 타스크 인스턴스도 삭제됩니다. 예를 들어, 포함 컨텍스트 ID가 프로세스 인스턴스의 PIID로 설정된 모든 타스크가 자동으로 삭제됩니다.

시작할 수 있는 프로세스 템플리트 또는 활동 판별

비즈니스 프로세스는 비즈니스 플로우 관리자 API의 call, initiate 또는 sendMessage 메소드를 호출해서 시작할 수 있습니다. 프로세스에 시작 활동이 하나만 있으면 프로세스 템플리트 이름이 매개변수로 필요한 메소드 서명을 사용할 수 있습니다. 프로세스에 둘 이상의 시작 활동이 있는 경우에는 시작 활동을 명시적으로 식별해야 합니다.

이 태스크 정보

비즈니스 프로세스가 모델화되면 모델러는 사용자의 서브세트만이 프로세스 템플리트로부터 프로세스 인스턴스를 작성할 수 있다고 결정할 수 있습니다. 인라인 호출 타스크를 프로세스의 시작 활동에 연관시키고 이 타스크에 대한 권한 제한사항을 지정해서 이를 수행할 수 있습니다. 타스크의 관리자 또는 잠재적 시작자인 사용자만이 타스크의 인스턴스와 프로세스 템플리트의 인스턴스를 작성하도록 허용됩니다.

인라인 호출 타스크가 시작 활동과 연관되지 않은 경우 또는 타스크에 대한 권한 제한사항이 지정되지 않은 경우에는 모든 사용자가 시작 활동을 사용하여 프로세스 인스턴스를 작성할 수 있습니다.

프로세스에는 둘 이상의 시작 활동이 있으며 각 활동의 잠재적 시작자 또는 관리자에 대한 사용자 조회는 서로 다릅니다. 이는 활동 B는 사용하지 않고 활동 A를 사용해서 프로세스를 시작하도록 사용자에게 권한을 부여할 수 있습니다.

프로시저

1. 비즈니스 플로우 관리자 API를 사용하여 시작 상태에 있는 프로세스 템플리트의 현재 버전 목록을 작성하십시오.

팁: queryProcessTemplates 메소드는 아직 시작하지 않은 응용프로그램의 일부인 프로세스 템플리트만을 제외합니다. 따라서 결과를 필터링하지 않고 이 메소드를 사용하면 메소드가 상태와 무관하게 프로세스 템플리트의 모든 버전을 리턴합니다.

```
// current timestamp in UTC format, converted to yyyy-mm-ddThh:mm:ss
String now = (new UTCDate()).toXsdString();
String whereClause = "PROCESS_TEMPLATE.STATE =
                      PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                      PROCESS_TEMPLATE.VALID_FROM =
                      (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                       WHERE NAME=PROCESS_TEMPLATE.NAME AND
                       VALID_FROM <= TS('" + now + "'))";

ProcessTemplateData[] processTemplates = process.queryProcessTemplates
                                        ( whereClause,
                                          "PROCESS_TEMPLATE.NAME",
                                          (Integer)null,

       (TimeZone)null);
```

프로세스 템플리트 이름순으로 결과가 정렬됩니다.

2. 사용자에게 권한이 부여된 시작 활동의 목록 및 프로세스 템플리트의 목록을 작성하십시오.

   프로세스 템플리트의 목록은 단일 시작 활동이 있는 프로세스 템플리트를 포함합니다. 이 활동은 모두 보안되지 않거나 로그온한 사용자가 이 활동을 시작할 수 있습니다. 또는 최소 하나의 시작 활동으로 시작할 수 있는 프로세스 템플리트를 집적하려 할 수 있습니다.

   팁: 프로세스 관리자도 프로세스 인스턴스를 시작할 수 있습니다. 그러나 비즈니스 플로우 관리자가 프로세스 관리를 시스템 관리자로 제한하는 대체 프로세스 관리 권한 모드를 사용하는 경우에는 BPESystemAdministrator 역할의 사용자만 이 조치를 수행할 수 있습니다. 그러므로 템플리트의 전체 목록을 얻으려면 로그온된 사용자가 관리자인지 여부도 확인해야 합니다.

   ```
   List authorizedProcessTemplates = new ArrayList();
   List authorizedActivityServiceTemplates = new ArrayList();
   ```

3. 각 프로세스 템플리트에 대한 시작 활동을 판별하십시오.

   ```
   for( int i=0; i<processTemplates.length; i++ )
   {
     ProcessTemplateData template = processTemplates[i];
     ActivityServiceTemplateData[] startActivities =
                    process.getStartActivities(template.getID());
   ```

4. 각 시작 활동에 대해 연관된 인라인 호출 타스크 템플리트의 ID를 검색하십시오.

   ```
   for( int j=0; j<startActivities.length; j++ )
   {
     ActivityServiceTemplateData activity = startActivities[j];
     TKTID tktid = activity.getTaskTemplateID();
   ```

a. 호출 타스크 템플리트가 존재하지 않는 경우 이 시작 활동으로 프로세스 템플리트가 보안되지 않습니다.

   이 경우 모든 사용자가 이 시작 활동을 사용하여 프로세스 인스턴스를 작성할 수 있습니다.

```
boolean isAuthorized = false;
       if ( tktid == null )
    {
      isAuthorized = true;
      authorizedActivityServiceTemplates.add(activity);
    }
```

b. 호출 타스크 템플리트가 존재하는 경우에는 휴먼 타스크 관리자 API를 사용하여 로그온한 사용자의 권한을 확인하십시오.

   예제에서 로그온한 사용자는 Smith입니다. 로그온한 사용자는 호출 타스크의 잠재적 시작자 또는 관리자여야 합니다.

```
if ( tktid != null )
    {
      isAuthorized =
        task.isUserInRole
            (tkid, "Smith", WorkItem.REASON_POTENTIAL_STARTER) ||
        task.isUserInRole(tktid, "Smith",

 WorkItem.REASON_ADMINISTRATOR);

      if ( isAuthorized )
      {
        authorizedActivityServiceTemplates.add(activity);
      }
    }
```

   사용자에게 지정된 역할이 있는 경우 또는 역할에 대한 사용자 지정 기준이 지정되지 않은 경우 isUserInRole 메소드는 true 값을 리턴합니다.

5. 프로세스 템플리트 이름만을 사용해서 프로세스를 시작할 수 있는지 여부를 확인하십시오.

```
if ( isAuthorized && startActivities.length == 1 )
    {
      authorizedProcessTemplates.add(template);
    }
```

6. 루프를 종료하십시오.

```
  } // end of loop for each activity service template
}   // end of loop for each process template
```

휴먼 타스크를 포함하는 단일 사용자 워크플로우 처리

일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예: 온라인 서점에서 책 주문). 이 예제는 서버측 페이지 플로우를 사용하여 단일 사용자 워크플로우를 구현하는 방법을 보여줍니다. 비즈니스 플로우 관리자와 휴먼 타스크 관리자 API는 모두 워크플로우를 처리하는 데 사용됩니다.

이 태스크 정보

단일 사용자 워크플로우를 *페이지 플로우* 또는 *화면 플로우*라고도 합니다. 두 종류의 페이지 플로우가 있습니다.

- 클라이언트측 페이지 플로우: 여기에서는 클라이언트측 기술을 사용하여 여러 페이지 간의 탐색을 실현합니다(예: 다중 페이지 Lotus Forms 양식).
- 서버측 페이지 플로우: 모델링된 비즈니스 프로세스 및 휴먼 타스크 세트를 사용하여 실현되므로 *후속타스크*가 동일한 사용자에게 지정됩니다.

서버측 페이지 플로우는 클라이언트측 페이지 플로우보다 강력하지만 처리하는 데 더 많은 서버 자원을 이용합니다. 그러므로 이 유형의 워크플로우는 기본적으로 다음과 같은 상황에서 사용할 것을 고려하십시오.

- 사용자 인터페이스에서 수행되는 단계 사이에 서비스를 호출해야 합니다(예: 데이터 검색 또는 갱신).
- 사용자 인터페이스 상호작용을 완료한 후 CEI 이벤트를 작성하도록 요구하는 감사 요구사항이 있습니다.

온라인 서점에서 구매자는 서적을 주문하기 위해 일련의 조치를 완료합니다. 이러한 일련의 조치는 일련의 휴먼 타스크 활동(수행할 타스크)으로 구현될 수 있습니다. 구매자가 몇 권의 책을 주문하려고 결정하면 이는 다음 휴먼 타스크 활동을 청구하는 것과 동등합니다. 일련의 타스크에 대한 정보는 휴먼 타스크 관리자가 타스크 자체를 유지보수하는 동안 비즈니스 플로우 관리자가 유지보수합니다.

비즈니스 플로우 관리자 API만을 사용하는 예제와 이 예제를 비교하십시오.

프로시저

1. 비즈니스 플로우 관리자 API를 사용하여 작업하려는 프로세스 인스턴스를 가져오십시오.

   이 예제에서는 CustomerOrder 프로세스의 인스턴스입니다.

   ```
   ProcessInstanceData processInstance =
      process.getProcessInstance("CustomerOrder");String piid =

      processInstance.getID().toString();
   ```

2. 휴먼 타스크 관리자 API를 사용하여 지정된 프로세스 인스턴스의 일부인 준비된 수행할 타스크(일종의 참여)를 조회하십시오.

타스크의 포함 컨텍스트 ID를 사용하여 포함하는 프로세스 인스턴스를 지정하십시오. 단일 사용자 워크플로우의 경우 조회는 일련의 휴먼 타스크 활동에서 첫 번째 휴먼 타스크 활동과 연관된 수행할 타스크를 리턴합니다.

```
//
 // Query the list of to-do tasks that can be claimed

 by the logged-on user
 // for the specified process instance
//
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.CONTAINMENT_CTX_ID = ID('" + piid + "') AND
                 TASK.STATE = TASK.STATE.STATE_READY AND
                 TASK.KIND = TASK.KIND.KIND_PARTICIPATING AND
                 WORK_ITEM.REASON = WORK_ITEM.REASON

   .REASON_POTENTIAL_OWNER",
                (String)null, (Integer)null, (TimeZone)null);
```

3. 리턴되는 수행할 타스크를 청구하십시오.

```
if (result.size() > 0)
{
 result.first();
 TKIID tkiid = (TKIID) result.getOID(1);
ClientObjectWrapper input = task.claim(tkiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject()

  instanceof DataObject )
  {
    taskInput = (DataObject)input.getObject();
    // read the values
    ...
  }
}
```

타스크가 청구될 때 타스크의 입력 메시지가 리턴됩니다.

4. 수행할 타스크와 연관된 휴먼 타스크 활동을 판별하십시오.

다음 메소드 중 하나를 사용하여 활동을 타스크와 상관시킬 수 있습니다.

- task.getActivityID 메소드:

```
AIID aiid = task.getActivityID(tkiid);
```

- 이 타스크 오브젝트의 일부인 상위 컨텍스트 ID:

```
AIID aiid = null;
Task taskInstance = task.getTask(tkiid);
OID oid = taskInstance.getParentContextID();
if ( oid != null and oid instanceof AIID )
{
  aiid = (AIID)oid;
}
```

5. 타스크에 대한 작업이 완료되면 비즈니스 플로우 관리자 API를 사용하여 타스크 및 연관된 휴먼 타스크 활동을 완료하고 프로세스 인스턴스의 다음 휴먼 타스크 활동을 청구하십시오.

   휴먼 타스크 활동을 완료하기 위해 출력 메시지가 전달됩니다. 출력 메시지를 작성하는 경우 메시지 유형 이름을 지정하여 메시지 정의가 포함되도록 해야 합니다.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the human task activity and its associated to-do task,
// and claim the next human task activity
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   이 조치는 주문 번호가 포함된 출력 메시지를 설정하고 순서에 있는 다음 휴먼 타스크 활동을 청구합니다. 후속 활동으로 AutoClaim을 설정하고 추적할 수 있는 여러 경로가 있으면 모든 후속 활동이 청구되며 다음 활동으로 임의 활동이 리턴됩니다. 이 사용자에게 지정할 수 있는 후속 활동이 없으면 Null이 리턴됩니다.

   프로세스에 후속 병렬 경로가 있고 이들 경로에 로그온 사용자가 둘 이상의 활동의 잠재적 소유자인 휴먼 타스크 활동이 있으면 임의 활동이 자동으로 청구되어 그 다음 활동으로 리턴됩니다.

6. 다음 휴먼 타스크 활동에 대해 작업하십시오.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

7. 5단계를 계속해서 휴먼 타스크 활동을 완료하고 다음 휴먼 타스크 활동을 검색하십시오.

관련 태스크

451 페이지의『단일 사용자 워크플로우 처리』
일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예: 온라인 서점에서 책 주문). 이러한 유형의 워크플로우에는 병렬 경로가 없습니다. initiateAndClaimFirst 및 completeAndClaimSuccessor API는 이 워크플로우 유형의 처리를 지원합니다. 다음 예제는 클라이언트측 페이지 플로우를 사용하는 단일 개인 워크플로우의 구현을 나타냅니다.

예외 및 결함 처리

BPEL 프로세스의 여러 지점에서 결함이 발생할 수 있습니다.

이 태스크 정보

BPEL(Business Process Execution Language) 결함은 다음에서 발생할 수 있습니다.

- 웹 서비스 호출(WSDL(Web Services Description Language) 결함)
- Throw 활동
- Business Process Choreographer에서 인식하는 BPEL 표준 결함

이 결함을 처리하기 위한 메커니즘이 존재합니다. 다음 메커니즘 중 하나를 사용하여 프로세스 인스턴스에서 생성한 결함을 처리하십시오.

- 해당 결함 핸들러로 제어 전달
- 프로세스의 이전 작업 보상
- 프로세스 중지 및 상황을 복구하도록 함(force-retry, force-complete)

BPEL 프로세스는 프로세스에서 제공한 조작 호출자에게 결함을 리턴할 수도 있습니다. 결함 이름 및 결함 데이터가 포함된 응답 활동으로 프로세스의 결함을 모델화할 수 있습니다. 이러한 결함은 API 호출자에게 확인된 예외로 리턴됩니다.

BPEL 프로세스가 BPEL 결함을 처리하지 않거나 API 예외가 발생한 경우, 런타임 예외가 API 호출자에게 리턴됩니다. API 예외의 예는 인스턴스를 작성할 프로세스 모델이 존재하지 않는 경우입니다.

결함 및 예외 처리에 대해서는 다음 타스크에서 설명합니다.

Business Process Choreographer EJB API 예외 처리

BusinessFlowManagerService 인터페이스 또는 HumanTaskManagerService 인터페이스의 메소드가 성공적으로 완료되지 않는 경우 오류의 원인을 선언하는 예외가 처리됩니다. 이 예외를 처리하여 호출자에게 안내를 제공할 수 있습니다.

이 태스크 정보

그러나 예외의 서브세트만 처리하고 다른 잠재적 예외에 대해서는 일반적인 안내를 제공하는 것이 일반적입니다. 특정한 예외는 모두 일반적인 ProcessException 또는 TaskException에서 상속됩니다. 최종 `catch(ProcessException)` 또는 `catch(TaskException)`문으로 일반 예외를 발견하십시오. 이 명령문은 발생할 수 있는 다른 모든 예외를 고려하기 때문에 응용프로그램의 상위 호환성을 보증합니다.

휴먼 타스크 활동에 대해 설정된 결함 확인

휴먼 타스크 활동이 처리될 때 성공적으로 완료할 수 있습니다. 이 경우 출력 메시지를 전달할 수 있습니다. 휴먼 타스크 활동이 완료되지 않는 경우 결함 메시지를 전달할 수 있습니다.

이 태스크 정보

결함 메시지를 읽고 오류 원인을 판별할 수 있습니다.

프로시저

1. 실패 또는 중지 상태의 task 활동을 표시하십시오.

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "(ACTIVITY.STATE =

     ACTIVITY.STATE.STATE_FAILED OR
                    ACTIVITY.STATE =

       ACTIVITY.STATE.STATE_STOPPED) AND
                    ACTIVITY.KIND=ACTIVITY.KIND.KIND_STAFF",
                    (String)null, (Integer)null, (TimeZone)null);
```

   이 조치를 실행하면 실패 또는 중지 상태의 활동을 포함하는 결과 조회 세트가 리턴됩니다.

2. 결함 이름을 읽으십시오.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ClientObjectWrapper faultMessage = process.getFaultMessage(aiid);
  DataObject fault = null ;
  if ( faultMessage.getObject() != null &&

  faultMessage.getObject()
       instanceof DataObject )
  {
    fault = (DataObject)faultMessage.getObject();
    Type type = fault.getType();
```

```
    String name = type.getName();
    String uri = type.getURI();
  }
}
```

결함 이름이 리턴됩니다. 결함 이름을 검색하는 대신 중지된 활동에 대해 처리되지 않은 예외를 분석할 수도 있습니다.

중지된 호출 활동에서 발생한 결함 확인

올바르게 설계된 프로세스에서는 일반적으로 결함 핸들러로 예외와 결함을 처리합니다. 활동 인스턴스에서 호출 활동에 대해 발생한 예외나 결함에 관한 정보를 검색할 수 있습니다.

이 태스크 정보

활동에 결함이 발생하는 경우 결함 유형으로 활동 복구에 필요한 조치가 결정됩니다.

프로시저

1. 중지 상태에 있는 휴먼 타스크 활동을 표시하십시오.

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    ACTIVITY.KIND=ACTIVITY.KIND.KIND_INVOKE",
                    (String)null, (Integer)null, (TimeZone)null);
```

   이 조치는 중지된 호출 활동을 포함하는 결과 조회 세트를 리턴합니다.

2. 결함 이름을 읽으십시오.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);

  ProcessException excp = activity.getUnhandledException();
  if ( excp instanceof ApplicationFaultException )
  {
   ApplicationFaultException fault = (ApplicationFaultException)excp;
   String faultName = fault.getFaultName();
  }
}
```

실패한 프로세스 인스턴스에 대해 발생한 결함 또는 처리되지 않은 예외 확인

올바르게 설계된 프로세스에서는 일반적으로 결함 핸들러로 예외와 결함을 처리합니다. 프로세스가 양방향 조작을 구현하는 경우 프로세스 인스턴스 오브젝트의 결함 이름 특성에서 결함이나 처리된 예외에 관한 정보를 검색할 수 있습니다. 결함의 경우 getFaultMessage API를 사용하여 대응하는 결함 메시지를 검색할 수도 있습니다.

이 태스크 정보

결함 핸들러가 처리하지 않는 예외로 인해 프로세스 인스턴스가 실패하는 경우 프로세스 인스턴스 오브젝트에서 처리되지 않은 예외에 대한 정보를 검색할 수 있습니다. 반대로, 결함 핸들러가 결함을 발견하는 경우 결함에 대한 정보가 제공되지 않습니다. 그러나 결함 이름과 메시지를 검색하고 FaultReplyException 예외를 사용하여 호출자로 리턴할 수 있습니다.

프로시저

1. 실패 상태의 프로세스 인스턴스를 표시하십시오.

```
QueryResultSet result =
     process.query("PROCESS_INSTANCE.PIID",
                   "PROCESS_INSTANCE.STATE =
                       PROCESS_INSTANCE.STATE.STATE_FAILED",
                    (String)null, (Integer)null, (TimeZone)null);
```

   이 조치는 실패한 프로세스 인스턴스가 포함된 조회 결과 세트를 리턴합니다.

2. 처리되지 않은 예외에 대한 정보를 읽으십시오.

```
if (result.size() > 0)
{
  result.first();
  PIID piid = (PIID) result.getOID(1);
  ProcessInstanceData pInstance = process.getProcessInstance(piid);

  ProcessException excp = pInstance.getUnhandledException();
  if ( excp instanceof RuntimeFaultException )
  {
   RuntimeFaultException xcp = (RuntimeFaultException)excp;
   Throwable cause = xcp.getRootCause();
  }
  else if ( excp instanceof StandardFaultException )
  {
   StandardFaultException xcp = (StandardFaultException)excp;
   String faultName = xcp.getFaultName();
  }
  else if ( excp instanceof ApplicationFaultException )
  {
   ApplicationFaultException xcp = (ApplicationFaultException)excp;
   String faultName = xcp.getFaultName();
  }
}
```

결과

이 정보를 사용하여 결함 이름 또는 문제점의 근본 원인을 찾으십시오.

비즈니스 프로세스 및 휴먼 타스크용 웹 서비스 API 클라이언트 응용프로그램 개발

Business Process Choreographer 웹 서비스 API를 통해 비즈니스 프로세스 응용프로그램 및 휴먼 타스크 응용프로그램에 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다. 클라이언트 응용프로그램 개발 프로세스는 웹 서비스 프록시 생성과 클라이언트 응용프로그램에 대한 보안 및 트랜잭션 정책 추가를 포함하여 많은 필수 및 선택적 단계로 구성됩니다.

이 태스크 정보

버전 7부터는 JAX-WS 기반 웹 서비스 API가 버전 6의 JAX-RPC 기반 Business Process Choreographer 웹 서비스 API(릴리스 6.0.2에서 처음 공개됨)를 대체합니다. JAX-RPC 기반 Business Process Choreographer 웹 서비스 API는 더 이상 사용되지 않으므로 JAX-WS 기반 API를 사용하여 새 웹 서비스 클라이언트 응용프로그램을 구현해야 합니다.

주: Business Process Choreographer JMS(Java Message Service) API는 여전히 버전 6의 WSDL 및 XML 스키마 정의를 사용합니다.

모든 웹 서비스 클라이언트 환경에서 클라이언트 응용프로그램을 개발할 수 있습니다. 다음 단계는 이러한 응용프로그램을 개발하는 데 필요한 조치의 개요를 제공합니다.

프로시저

1. 클라이언트 응용프로그램에서 사용할 웹 서비스 API(비즈니스 플로우 관리자 API, 휴먼 타스크 관리자 API 또는 둘 다)를 결정하십시오.
2. WebSphere Process Server 환경에서 필요한 파일을 내보내십시오.
3. 클라이언트 응용프로그램 개발 환경에서 내보낸 아티팩트를 사용하여 *웹 서비스 프록시*를 생성하십시오.
4. 사용자 클라이언트 응용프로그램을 개발하십시오.
5. 필수 보안 또는 트랜잭션 정책을 클라이언트 응용프로그램에 추가하십시오.

웹 서비스 컴포넌트 및 제어 순서

웹 서비스 응용프로그램에서는 많은 클라이언트 측 및 서버측 컴포넌트가 웹 서비스 요청 및 응답을 나타내는 제어 순서와 관련이 있습니다.

제어의 일반 순서는 다음과 같습니다.

1. 클라이언트측:
   a. 클라이언트 응용프로그램(사용자가 제공)이 웹 서비스에 대한 요청을 발행합니다.

b. 웹 서비스 프록시(사용자가 제공하기도 하지만 클라이언트 측 유틸리티를 사용하여 자동으로 생성할 수 있음)가 SOAP 요청 엔벨로프(envelope)에 서비스 요청을 랩핑하여 엔드포인트로 정의된 URL에 요청을 전달합니다.

2. 네트워크에서 HTTP 또는 HTTPS를 사용하여 웹 서비스 엔드포인트에 요청을 전송합니다.
3. 서버측:
   a. 일반 웹 서비스 API는 요청을 수신하여 디코드합니다.
   b. 일반 비즈니스 플로우 관리자 또는 휴먼 타스크 관리자 컴포넌트로 요청을 직접 처리하거나 지정된 비즈니스 프로세스 또는 휴먼 타스크로 요청을 전달합니다.
   c. 리턴된 데이터가 SOAP 응답 엔벨로프에 랩핑됩니다.
4. 네트워크에서 HTTP 또는 HTTPS를 사용하여 클라이언트측 환경으로 응답을 전송합니다.
5. 다시 클라이언트측:
   a. 클라이언트측 개발 인프라에서 SOAP 응답 엔벨로프를 랩핑 해제합니다.
   b. 웹 서비스 프록시가 SOAP 응답에서 데이터를 추출하여 클라이언트 응용프로그램으로 전달합니다.
   c. 클라이언트 응용프로그램이 필요에 따라 리턴된 데이터를 처리합니다.

예제

다음은 휴먼 타스크 관리자 웹 서비스 API에 액세스하여 수행할 작업 타스크를 처리하는 클라이언트 응용프로그램에 대한 개략적인 설명입니다.

1. 클라이언트 응용프로그램은 사용자가 작업할 수행할 작업 타스크의 목록을 요청하는 WebSphere Process Server에 대한 query 웹 서비스 호출을 발행합니다.
2. WebSphere Process Server는 수행할 작업 타스크 목록을 리턴합니다.
3. 그러면 클라이언트 응용프로그램은 수행할 작업 타스크 중 하나를 청구하는 claim 웹 서비스 호출을 발행합니다.
4. WebSphere Process Server는 타스크의 입력 메시지를 리턴합니다.
5. 클라이언트 응용프로그램은 출력 또는 결함 메시지와 함께 타스크를 완료하기 위한 complete 웹 서비스 호출을 발행합니다.

비즈니스 프로세스 및 휴먼 타스크에 대한 웹 서비스 API 요구사항

WebSphere Integration Developer로 개발되어 Business Process Choreographer에서 실행되는 비즈니스 프로세스 및 휴먼 타스크는 웹 서비스 API를 통해 액세스할 수 있는 특정 규칙을 준수해야 합니다.

요구사항은 다음과 같습니다.

- 비즈니스 프로세스 및 휴먼 타스크의 인터페이스는 JAX-WS 2.0(Java API for XML-based Web Services) 스펙에 정의된 "문서/리터럴 랩핑" 스타일을 사용하여 정의해야 합니다. 이는 WebSphere Integration Developer로 개발한 모든 비즈니스 프로세스 및 휴먼 타스크의 기본 스타일입니다.
- 조작의 매개변수 요소에 maxOccurs 속성을 사용하지 않거나 이 속성의 값이 기본값 `maxOccurs="1"`로 설정되었는지 확인하십시오.
- 웹 서비스 조작의 비즈니스 프로세스 및 휴먼 타스크에 표시되는 결함 메시지는 XML 스키마 요소로 정의된 단일 WSDL 메시지로 구성되어야 합니다. 예를 들어, 다음과 같습니다.

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

관련 정보

JAX-WS 2.0(Java API for XML-based Web Services) 다운로드 페이지

사용해야 하는 WSDL 유형

JAX-WS 기반 Business Process Choreographer 웹 서비스 API

버전 7부터는 JAX-WS 기반 웹 서비스 API가 버전 6의 JAX-RPC 기반 Business Process Choreographer 웹 서비스 API(릴리스 6.0.2에서 처음 공개됨)를 대체합니다. 비즈니스 프로세스용 하나와 휴먼 타스크용 하나의 두 가지 Business Process Choreographer 웹 서비스 인터페이스가 제공되며 각각에는 자체 파일 아티팩트 및 XML 정의 네임스페이스가 있습니다.

다음 표는 JAX-WS 기반 웹 서비스의 파일 아티팩트 및 XML 정의 네임스페이스 개요를 제공합니다.

표 69. JAX-WS 기반 웹 서비스의 파일 아티팩트 및 XML 정의 네임스페이스

Business Process Choreographer 웹 서비스 인터페이스	**JAX-WS 웹 서비스 파일 아티팩트**	**JAX-WS 웹 서비스 XML 네임스페이스**
비즈니스 플로우 관리자 웹 서비스	BFMJAXWSService.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0/Binding
비즈니스 플로우 관리자 웹 서비스 인터페이스	BFMJAXWSInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0
비즈니스 플로우 관리자 웹 서비스 데이터 유형	BFMDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/business-process/types/7.0
비즈니스 플로우 관리자 콜백 웹 서비스	BFMJAXWSCallbackService.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/callback-services/7.0/Binding
비즈니스 플로우 관리자 콜백 웹 서비스 인터페이스	BFMJAXWSCallbackInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/callback-services/7.0
휴먼 타스크 관리자 웹 서비스	HTMJAXWSService.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/services/7.0/Binding
휴먼 타스크 관리자 웹 서비스 인터페이스	HTMJAXWSInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/services/7.0
휴먼 타스크 관리자 웹 서비스 데이터 유형	HTMDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/human-task/types/7.0
휴먼 타스크 관리자 콜백 웹 서비스	HTMJAXWSCallbackService.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/callback-services/7.0/Binding
휴먼 타스크 관리자 콜백 웹 서비스 인터페이스	HTMJAXWSCallbackInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/callback-services/7.0
공통 Business Process Choreographer 데이터 유형	BPCDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/7.0

Business Process Choreographer 웹 서비스 API: 표준

다음 링크를 사용하여 웹 응용프로그램에 적용되는 표준에 대한 관련 보충 정보를 찾을 수 있습니다. 정보는 비 IBM 인터넷 사이트에 있으며 이 사이트의 스폰서가 정보의 기술적 정확성을 제어합니다.

편의를 위해 다음 링크가 제공됩니다. 정보가 IBM WebSphere Process Server에 맞지 않는 경우가 있지만 일반적으로 웹 서비스를 이해하는 데 유용합니다.

- JAX-WS 2.0(Java API for XML-based Web Services) (JSR-224; Java Community Process)
- JAXB(Java Architecture for XML Binding) 2.0 (JSR-222; Java Community Process)
- WSDL(Web Services Description Language) 1.1 (W3C)
- XML 스키마 파트 0: Primer Second Edition (W3C)
- XML 스키마 파트 1: Structures Second Edition (W3C)
- XML 스키마 파트 2: Datatypes Second Edition (W3C)
- SOAP(Simple Object Access Protocol) 1.1 (W3C)
- WS-Policy(Web Services Policy Framework) 1.5 (W3C)
- WS-Security 1.1 (OASIS)
- WS-Security UserName Token Profile 1.1 (OASIS)
- WS-AtomicTransaction 1.2 (OASIS)
- WS-Interoperability Basic Profile 1.1 (WS-Interoperability Organization)

서버 환경에서 웹 서비스 클라이언트 응용프로그램을 위한 아티팩트 공개 및 내보내기

Business Process Choreographer 웹 서비스 API에 액세스하는 클라이언트 응용프로그램을 개발하려면 우선 WebSphere 서버 환경에서 여러 아티팩트를 공개하고 내보내야 합니다.

이 태스크 정보

내보내는 아티팩트는 다음과 같습니다.

- Business Process Choreographer 웹 서비스 API를 구성하는 웹 서비스 엔드포인트, 포트 유형 및 조작을 설명하는 WSDL(Web Service Definition Language) 파일(웹 서비스 프록시 생성에 항상 필요)
- Business Process Choreographer WSDL 파일의 서비스에서 참조되는 데이터 유형 정의가 들어 있는 XSD(XML Schema Definition) 파일(웹 서비스 프록시 생성에 항상 필요)

- WebSphere 서버에서 실행 중인 비즈니스 프로세스 또는 휴먼 타스크의 인터페이스 및 데이터 유형을 설명하는 사용자 자신의 WSDL 및 XSD 파일. 이러한 추가 파일은 클라이언트 응용프로그램이 웹 서비스 API를 통해 비즈니스 프로세스 또는 휴먼 타스크와 직접 상호작용해야 하는 경우에만 필요합니다. 클라이언트 응용프로그램이 사용자의 프로세스 또는 타스크 인스턴스와 직접 상호작용하지 않고 Business Process Choreographer로 충족되는 조작(예: 조회 발행)만 호출하는 경우에는 필요하지 않습니다.
- 웹 서비스 API에 대한 서비스 속성의 품질을 설명하는 WS-Policy(Web Service Policy) 파일. 클라이언트측 웹 서비스 정책을 작성하기 위한 기초로 사용하기 위해 내보낼 수 있습니다.

  WS-Security

  요청 메시지에는 사용자 이름 토큰 또는 LPTA 토큰이 있어야 합니다.

  WS-Transaction

  요청 메시지에는 WS-AtomicTransaction 컨텍스트가 포함될 수 있습니다. 이 컨텍스트가 있는 경우, 요청은 호출자의 트랜잭션 범위에서 처리됩니다.

해당 아티팩트를 공개한 후 클라이언트 프로그래밍 환경으로 이를 복사해야 합니다. 클라이언트 프로그래밍 환경에서는 이러한 아티팩트를 사용하여 웹 서비스 프록시 및 헬퍼 클래스를 생성합니다.

Business Process Choreographer WSDL 파일 공개

WSDL(Web Service Definition Language) 파일은 웹 서비스 API에서 사용 가능한 모든 조작에 대한 자세한 설명을 포함합니다. 비즈니스 플로우 관리자 및 휴먼 타스크 관리자 웹 서비스 API에 대한 별도의 WSDL 파일을 사용할 수 있습니다. 이 파일은 사용자 응용프로그램에 대해 웹 서비스 프록시를 생성하는 데 사용됩니다.

시작하기 전에

WSDL 파일을 공개하기 전에 올바른 웹 서비스 엔드포인트 주소를 지정했는지 확인하십시오. 이 주소는 클라이언트 응용프로그램이 웹 서비스 API를 액세스하는 데 사용하는 URL입니다.

이 태스크 정보

이 WSDL 파일과 WSDL 파일에서 참조되는 모든 XSD 파일을 공개해야 합니다. 그런 다음 WebSphere 환경에서 사용자의 개발 환경으로 복사할 수 있으며, 개발 환경에서 웹 서비스 프록시 및 헬퍼 클래스를 생성하는 데 사용됩니다. Business Process Choreographer WSDL 파일은 한 번만 공개하면 됩니다.

웹 서비스 응용프로그램에 대한 비즈니스 프로세스 WSDL 파일 공개:

WSDL 파일을 공개하려면 관리 콘솔을 사용하십시오.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.
2. **응용프로그램 → SCA 모듈**을 클릭하십시오.

   주: 응용프로그램 → 응용프로그램 유형 → WebSphere 엔터프라이즈 응용프로그램을 클릭하여 사용 가능한 모든 엔터프라이즈 응용프로그램 목록을 표시할 수도 있습니다.
3. SCA 모듈 또는 응용프로그램 목록에서 **BPEContainer** 응용프로그램을 선택하십시오.
4. **추가 특성** 목록에서 **WSDL 파일 공개**를 선택하십시오.
5. 목록에서 .zip 파일을 클릭하십시오.
6. 표시된 파일 다운로드 창에서 **저장**을 클릭하십시오.
7. 로컬 폴더를 찾아보고 **저장**을 클릭하십시오.

결과

내보낸 .zip 파일의 이름은 `BPEContainer_nodename_servername_WSDLFiles.zip`입니다. .zip 파일에는 웹 서비스를 설명하는 WSDL 파일과 WSDL 파일에서 참조되는 모든 XSD 파일이 들어 있습니다.

주: 내보낸 .zip 파일에는 버전 7에서 소개된 JAX-WS 웹 서비스와 버전 6에서 사용된 JAX-RPC 웹 서비스 둘 다의 WSDL 및 XSD 아티팩트가 포함되어 있습니다. wsimport 도구를 사용하여 웹 서비스 프록시를 생성하는 경우에는 JAX-WS 웹 서비스 아티팩트를 선택하십시오. JAX-RPC 아티팩트는 무시됩니다.

웹 서비스 응용프로그램에 대한 휴먼 타스크 WSDL 파일 공개:

WSDL 파일을 공개하려면 관리 콘솔을 사용하십시오.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.
2. **응용프로그램 → SCA 모듈**을 클릭하십시오.

   주: 응용프로그램 → 응용프로그램 유형 → WebSphere 엔터프라이즈 응용프로그램을 클릭하여 사용 가능한 모든 엔터프라이즈 응용프로그램 목록을 표시할 수도 있습니다.
3. SCA 모듈 또는 응용프로그램 목록에서 **TaskContainer** 응용프로그램을 선택하십시오.
4. **추가 특성** 목록에서 **WSDL 파일 공개**를 선택하십시오.
5. 목록에서 .zip 파일을 클릭하십시오.
6. 표시된 파일 다운로드 창에서 **저장**을 클릭하십시오.

7. 로컬 폴더를 찾아보고 **저장**을 클릭하십시오.

결과

내보낸 .zip 파일의 이름은 `TaskContainer_nodename_servername_WSDLFiles.zip` 입니다. .zip 파일에는 웹 서비스를 설명하는 WSDL 파일과 WSDL 파일에서 참조되는 모든 XSD 파일이 들어 있습니다.

주: 내보낸 .zip 파일에는 버전 7에서 소개된 JAX-WS 웹 서비스와 버전 6에서 사용된 JAX-RPC 웹 서비스 둘 다의 WSDL 및 XSD 아티팩트가 포함되어 있습니다. wsimport 도구를 사용하여 웹 서비스 프록시를 생성하는 경우에는 JAX-WS 웹 서비스 아티팩트를 선택하십시오. JAX-RPC 아티팩트는 무시됩니다.

비즈니스 프로세스 및 휴먼 타스크 웹 서비스 응용프로그램에 대해 WSDL 및 XSD 파일 내보내기

비즈니스 프로세스 및 휴먼 타스크 인터페이스에는 이들을 웹 서비스로 외부에서 액세스할 수 있게 해주는 잘 정의된 인터페이스가 있습니다. WSDL 인터페이스 정의 및 XML 스키마 데이터 유형 정의를 클라이언트 프로그래밍 환경으로 내보내야 합니다.

이 태스크 정보

이 프로시저는 클라이언트 응용프로그램이 상호작용해야 하는 각 비즈니스 프로세스 및 휴먼 타스크에 대해 반복되어야 합니다.

예를 들어, 휴먼 타스크를 작성하고 시작하려면, 다음 정보 항목을 타스크 인터페이스에 전달해야 합니다.

- 타스크 템플리트 이름
- 타스크 템플리트 네임스페이스
- 형식화된 비즈니스 데이터를 포함하는 입력 메시지
- 응답 메시지를 리턴하는 응답 랩퍼
- 결함 및 예외를 리턴하기 위한 결함 메시지

해당 항목은 단일 비즈니스 오브젝트 내에 캡슐화됩니다. 웹 서비스 인터페이스의 모든 조작은 문서/리터럴 랩핑 조작으로 모델화됩니다. 이들 조작의 입력 및 출력 매개변수는 랩퍼 문서에서 캡슐화됩니다. 기타 비즈니스 오브젝트는 대응하는 응답 및 결함 메시지 형식을 정의합니다.

웹 서비스를 통해 비즈니스 프로세스 또는 휴먼 타스크를 작성하고 시작하려면 클라이언트측의 클라이언트 응용프로그램에서 랩퍼 오브젝트를 사용할 수 있어야 합니다.

이는 WebSphere 환경에서 비즈니스 오브젝트를 WSDL(Web Service Definition Language) 및 XSD(XML Schema Definition) 파일로 내보내고 데이터 유형 정의를 클라이언트 프로그래밍 환경으로 가져옴으로써 달성됩니다.

프로시저

1. WebSphere Integration Developer 작업공간이 아직 실행 중이지 않으면 실행하십시오.
2. 내보낼 비즈니스 오브젝트를 포함하는 라이브러리 모듈을 선택하십시오. 라이브러리 모듈은 필요한 비즈니스 오브젝트가 있는 압축 파일입니다.
3. 라이브러리 모듈을 내보내십시오.
4. 내보낸 파일을 클라이언트 응용프로그램 개발 환경으로 복사하십시오.

예제

비즈니스 프로세스가 다음과 같은 웹 서비스 조작을 구현한다고 가정하십시오.

```
<wsdl:operation name="updateCustomer">
    <wsdl:input message="tns:updateCustomerRequestMsg"
                name="updateCustomerRequest"/>
    <wsdl:output message="tns:updateCustomerResponseMsg"
                 name="updateCustomerResponse"/>
    <wsdl:fault message="tns:updateCustomerFaultMsg"
                name="updateCustomerFault"/>
  </wsdl:operation>
```

WSDL 메시지는 다음과 같이 정의됩니다.

```
<wsdl:message name="updateCustomerRequestMsg">
    <wsdl:part element="types:updateCustomer"
               name="updateCustomerParameters"/>
  </wsdl:message>
  <wsdl:message name="updateCustomerResponseMsg">
    <wsdl:part element="types:updateCustomerResponse"
               name="updateCustomerResult"/>
  </wsdl:message>
  <wsdl:message name="updateCustomerFaultMsg">
    <wsdl:part element="types:updateCustomerFault"
               name="updateCustomerFault"/>
  </wsdl:message>
```

구체적 사용자 정의 요소 types:updateCustomer, types:updateCustomerResponse 및 types:updateCustomerFault는 클라이언트 응용프로그램에서 수행되는 모든 *일반* 조작(call, sendMessage 등)에서 UserData 매개변수를 통해 웹 서비스 API로 전달하고 웹 서비스 API에서 수신해야 합니다.

클라이언트 응용프로그램측에서 내보낸 XSD 파일을 사용하여 생성한 클래스를 사용하여 사용자 정의 요소가 작성되고 직렬화되고 직렬화 해제됩니다. 이러한 클래스의 생성은 내보낸 WSDL 및 XSD 파일이 포함된 웹 서비스 프록시 생성의 일부입니다.

웹 서비스 인터페이스의 일반 조작은 문서 랩퍼 요소를 비즈니스 프로세스 또는 휴먼 타스크로 구현한 조작과 주고 받습니다. 이전 예제의 샘플 조작에 대해 웹 서비스 SOAP 메시지는 다음과 같습니다.

```
<soapenv:Envelope xmlns:soapenv="..." ...>
  <soapenv:Header>
    ...
  </soapenv:Header>
  <soapenv:Body>
    <bfm:sendMessage
         xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0">
      <processTemplateName>customerProcessTemplate</processTemplateName>
      <portType xmlns:cns="http://example.com/customerProcess">cns

      :customerProcessPortType</portType>
      <operation>updateCustomer</operation>
      <input>
        <cns:updateCustomer xmlns:cns="http://example.com/customerProcess">
          <street>1600 Pennsylvania Avenue Northwest</street>
          <city>Washington, DC 20006</city>
        </cns:updateCustomer>
      </input>
    </bfm:sendMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Java 웹 서비스 환경에서 클라이언트 응용프로그램 개발

Java 웹 서비스와 호환 가능한 Java 기반 개발 환경을 사용하여 Business Process Choreographer 웹 서비스 API에 대한 클라이언트 응용프로그램을 개발할 수 있습니다.

웹 서비스 프록시 생성(Java 웹 서비스)

Java 웹 서비스 클라이언트 응용프로그램은 *웹 서비스 프록시*를 사용하여 Business Process Choreographer 웹 서비스 API와 상호작용합니다.

이 태스크 정보

Java 웹 서비스에 대한 웹 서비스 프록시에는 클라이언트 응용프로그램이 웹 서비스 요청을 수행하기 위해 호출하는 많은 JavaBeans 클라이언트가 있습니다. 웹 서비스 프록시는 서비스 매개변수의 어셈블리를 SOAP 메시지로 처리하고 HTTP를 통해 웹 서비스로 SOAP 메시지를 전송하고 웹 서비스에서 응답을 수신하며 리턴된 데이터를 클라이언트 응용프로그램으로 전달합니다.

그러므로 기본적으로 웹 서비스 프록시를 사용하여 클라이언트 응용프로그램은 웹 서비스를 로컬 함수처럼 호출할 수 있습니다.

주: 웹 서비스 프록시는 한 번만 생성하면 됩니다. 그러면 동일한 웹 서비스 API에 액세스하는 모든 클라이언트 응용프로그램은 동일한 웹 서비스 프록시를 사용할 수 있습니다.

IBM 웹 서비스 환경에서는 다음 방법 중 하나로 웹 서비스 프록시를 생성할 수 있습니다.

- Rational Application Developer 또는 WebSphere Integration Developer 통합 개발 환경 사용
- wsimport 명령행 도구 사용

기타 Java 웹 서비스 개발 환경에는 일반적으로 wsimport 도구 또는 독점 클라이언트 응용프로그램 생성 기능이 포함되어 있습니다.

Rational Application Developer를 사용하여 웹 서비스 응용프로그램에 대한 웹 서비스 프록시 생성:

Rational Application Developer 통합 개발 환경을 사용하여 웹 서비스 클라이언트 응용프로그램에 대한 웹 서비스 프록시를 생성할 수 있습니다. 다음 단계 순서는 Rational Application Developer 버전 7.5.3에 적용됩니다.

시작하기 전에

웹 서비스 프록시를 생성하기 전에 우선 비즈니스 프로세스 또는 휴먼 타스크 웹 서비스 인터페이스를 설명하는 WSDL 및 XSD 파일을 WebSphere 환경에서 내보낸 후 사용자의 클라이언트 프로그래밍 환경에 복사해야 합니다.

프로시저

1. 해당하는 WSDL 파일을 프로젝트에 추가하십시오.
   - 비즈니스 프로세스의 경우:
     a. 내보낸 `BPEContainer_nodename_servername_WSDLFiles.zip` 파일의 압축을 임시 디렉토리에 푸십시오. 이 디렉토리의 컨텐츠는 변경하지 마십시오. 비즈니스 프로세스와 상호작용하기 위한 웹 서비스 프록시 생성에는 다음 WSDL 및 XSD 파일만 사용됩니다.
        - BFMJAXWSService.wsdl
        - BFMJAXWSInterface.wsdl
        - BFMJAXWSCallbackService.wsdl
        - BFMJAXWSCallbackInterface.wsdl
        - BFMDataTypes.xsd
        - BPCDataTypes.xsd
        - wsa.xsd

     b. 압축을 푼 디렉토리 `BPEContainer_`*`nodename_servername`*`.ear/bfmjaxws.jar`에서 서브디렉토리 `META-INF`를 가져오십시오.
   - 휴먼 타스크의 경우:

a. 내보낸 TaskContainer_nodename_servername_WSDLFiles.zip 파일의 압축을 임시 디렉토리에 푸십시오. 이 디렉토리의 컨텐츠는 변경하지 마십시오. 휴먼 타스크와 상호작용하기 위한 웹 서비스 프록시 생성에는 다음 WSDL 및 XSD 파일만 사용됩니다.
   - HTMJAXWSService.wsdl
   - HTMJAXWSInterface.wsdl
   - HTMJAXWSCallbackService.wsdl
   - HTMJAXWSCallbackInterface.wsdl
   - HTMDataTypes.xsd
   - BPCDataTypes.xsd
   - wsa.xsd

b. 압축을 푼 디렉토리 TaskContainer_*nodename_servername*.ear/htmjaxws.jar에서 서브디렉토리 META-INF를 가져오십시오.

새 wsdl 디렉토리 및 서브디렉토리 구조가 프로젝트에 작성됩니다.

2. 새로 작성한 wsdl 디렉토리에 있는 BFMJAXWSService.wsdl 파일을 선택하십시오.
3. 마우스 오른쪽 단추를 클릭한 후 **웹 서비스 → 클라이언트 생성**을 선택하십시오.

   나머지 단계를 계속하기 전에 서버가 시작되었는지 확인하십시오.
4. 웹 서비스 창에서 **다음**을 클릭하여 모든 기본값을 승인하십시오.
5. 웹 서비스 JAX-WS 웹 서비스 클라이언트 구성 창에서 생성할 JAX-WS 코드의 버전을 2.0으로 변경하고 **완료**를 클릭하여 다른 모든 기본값을 승인하십시오.
6. HTMJAXWSService.wsdl에 대해 이 프로시저의 2 - 5단계를 다시 수행하고 프롬프트되는 모든 파일을 겹쳐쓰십시오.

결과

많은 프록시, 위치 지정자 및 JAXB 클래스로 구성되는 웹 서비스 프록시가 생성되어 프로젝트에 추가됩니다.

wsimport 명령행 도구를 사용하여 웹 서비스 응용프로그램에 대한 웹 서비스 프록시 생성:

wsimport 명령행 도구를 사용하여 웹 서비스 응용프로그램에 대한 웹 서비스 프록시를 생성할 수 있습니다.

시작하기 전에

웹 서비스 프록시를 생성하기 전에 우선 비즈니스 프로세스 또는 휴먼 타스크 웹 서비스 API를 설명하는 WSDL 파일을 WebSphere 환경에서 내보낸 후 사용자의 클라이언트 프로그래밍 환경에 복사해야 합니다.

프로시저

1. Business Process Choreographer 웹 서비스 API에 대한 웹 서비스 프록시를 생성하십시오.

   주: JAX-WS 응용프로그램에 대한 wsimport 명령행 도구의 자세한 설명은 WebSphere Application Server wsimport 명령행 도구 문서를 참조하십시오.

   ```
   wsimport.bat BFMJAXWSService.wsdl myService1.wsdl myService2.wsdl
    –d proxy-bfm
    -wsdllocation <bfm_location>

   wsimport.bat HTMJAXWSService.wsdl myService1.wsdl myService2.wsdl
    –d proxy-htm
    -wsdllocation <htm_location>
   ```

   이 예제에서 `myService1.wsdl` 및 `myService2.wsdl`에는 사용자 정의 비즈니스 프로세스, 휴먼 타스크 또는 둘 다의 인터페이스 정의가 있습니다. 또한 <*bfm_location*> 및 <*htm_location*>은 각각 `BFMJAXWSService.wsdl` 및 `HTMJAXWSService.wsdl`의 WSDL <*port*> 요소에서 얻을 수 있습니다.

   두 프록시 모두를 단일 공통 디렉토리(예: proxy-bpc)에 병합하고 프롬프트되는 경우 기존 파일을 겹쳐쓸 수 있습니다.
2. 프로젝트에 생성된 클래스 파일을 포함시키십시오.

관련 태스크

『비즈니스 프로세스 및 휴먼 타스크용 클라이언트 응용프로그램 작성(Java 웹 서비스)』

클라이언트 응용프로그램은 Business Process Choreographer 웹 서비스 API에 요청을 전송하거나 웹 서비스 API에서 응답을 수신합니다. 웹 서비스 프록시를 사용하여 통신을 관리하고 헬퍼 클래스로 복잡한 데이터 유형을 형식화하면 클라이언트 응용프로그램에서 웹 서비스 메소드를 로컬 함수처럼 호출할 수 있습니다.

비즈니스 프로세스 및 휴먼 타스크용 클라이언트 응용프로그램 작성(Java 웹 서비스)

클라이언트 응용프로그램은 Business Process Choreographer 웹 서비스 API에 요청을 전송하거나 웹 서비스 API에서 응답을 수신합니다. 웹 서비스 프록시를 사용하여 통신을 관리하고 헬퍼 클래스로 복잡한 데이터 유형을 형식화하면 클라이언트 응용프로그램에서 웹 서비스 메소드를 로컬 함수처럼 호출할 수 있습니다.

시작하기 전에

클라이언트 응용프로그램 작성을 시작하기 전에 웹 서비스 프록시를 생성하십시오.

이 태스크 정보

웹 서비스 호환 개발 도구(예: IBM Rational Application Developer)를 사용하여 클라이언트 응용프로그램을 개발할 수 있습니다. 웹 서비스 API를 호출하는 모든 종류의 웹 서비스 응용프로그램을 빌드할 수 있습니다.

프로시저

1. 새 클라이언트 응용프로그램 프로젝트를 작성하십시오.
2. 웹 서비스 프록시를 생성하십시오.
3. 클라이언트 응용프로그램을 코딩하십시오.
4. 프로젝트를 빌드하십시오.
5. 클라이언트 응용프로그램을 실행하십시오.

예

다음 예는 비즈니스 플로우 관리자 웹 서비스 API를 사용하는 방법을 보여줍니다.

```
try {
   // create bfm proxy
   BFMJAXWSPortType bfm = new BFMJAXWSService().getBFMJAXWSPort();

   // call getProcessTemplate
   ProcessTemplateType ptt =
     bfm.getProcessTemplate("MY_PROCESS_TEMPLATE_NAME");

   // handle return value
   System.out.println("Process template '" + ptt.getName() +
       "' found, details following:");
   System.out.println("Execution mode: " +
       ptt.getExecutionMode());
   System.out.println("Schema version: " +
       ptt.getSchemaVersion());
 } catch (Exception e) {
   if ( e instanceof ProcessFaultMsg )
   {
     ProcessFaultMsg pfm = (ProcessFaultMsg) e;
     List<FaultStackType> list =
         ( pfm.getFaultInfo() ).getFaultStack();
     FaultStackType fault = list.get( 0 );
     System.out.println( "ProcessFaultMessage: " +
         fault.getMessage() );
   }
           else
   {
     e.printStackTrace( System.out );
   }
 }
```

관련 태스크

508 페이지의『wsimport 명령행 도구를 사용하여 웹 서비스 응용프로그램에 대한 웹 서비스 프록시 생성』

wsimport 명령행 도구를 사용하여 웹 서비스 응용프로그램에 대한 웹 서비스 프록시를 생성할 수 있습니다.

506 페이지의『웹 서비스 프록시 생성(Java 웹 서비스)』

Java 웹 서비스 클라이언트 응용프로그램은 *웹 서비스 프록시*를 사용하여 Business Process Choreographer 웹 서비스 API와 상호작용합니다.

보안 추가

Business Process Choreographer 웹 서비스를 사용하려면 인증 메커니즘에 대해 클라이언트 응용프로그램을 구성해야 합니다.

이 태스크 정보

기본적으로 Business Process Choreographer는 다음 인증 메커니즘을 지원합니다.

사용자 이름 토큰

웹 서비스 처리자는 "사용자 이름"으로 요청자를 식별하고 선택적으로 암호를 사용하여 웹 서비스 프로바이더에 대한 ID를 인증하는 수단으로 사용자 이름 토큰을 제공합니다.

2진 보안 토큰 - LTPA(Lightweight Third-Party Authentication) 토큰

웹 서비스 처리자는 웹 서비스 프로바이더에 대한 요청자를 인증하는 수단으로 LTPA 토큰을 제공합니다.

Business Process Choreographer 웹 서비스 보안 정책을 대체 인증 메커니즘으로 바꿀 수 있습니다. 그러나 인증되지 않은 사용자로 Business Process Choreographer 웹 서비스를 호출할 수는 없으므로 하나의 인증 메커니즘이 항상 필요합니다.

트랜잭션 지원 추가

서비스 요청의 일부로 클라이언트 응용프로그램 컨텍스트를 전달하여 서버측 요청 처리가 클라이언트의 트랜잭션에 참여할 수 있도록 웹 서비스 클라이언트 응용프로그램을 구성할 수 있습니다. 이러한 원자적 트랜잭션 지원은 WS-AT(Web Services-Atomic Transaction) 스펙에 정의됩니다.

이 태스크 정보

Business Process Choreographer는 각 웹 서비스 조작 요청을 독립 글로벌 트랜잭션으로 실행합니다. 다음 방법 중 하나로 트랜잭션 지원을 사용하도록 클라이언트 응용프로그램을 구성할 수 있습니다.

- 클라이언트의 트랜잭션 컨텍스트를 전파합니다. 서버측 요청 처리가 클라이언트 응용프로그램 트랜잭션 컨텍스트에서 수행되므로 클라이언트의 트랜잭션과 함께 커미트(또

는 백아웃)됩니다. 반대로, 서버에서 웹 서비스 조작 실행 중에 문제가 발생하여 롤백을 요청하는 경우에는 클라이언트 응용프로그램의 트랜잭션도 롤백됩니다.

- 트랜잭션 지원을 사용하지 않도록 구성합니다. Business Process Choreographer는 요청을 실행할 새 글로벌 트랜잭션을 작성하지만 서버측 요청 처리는 클라이언트 응용프로그램 트랜잭션 컨텍스트로 수행되지 않습니다.

Business Process Choreographer 웹 서비스에 첨부된 웹 서비스 정책을 사용하면 위에 설명한 것처럼 각 요청 메시지에 WS-AT 트랜잭션 컨텍스트를 포함할 수 있습니다. 클라이언트 트랜잭션 컨텍스트를 전달하지 않고 웹 서비스 조작을 호출하려는 경우에는 프로바이더측 트랜잭션 정책을 무시하고 트랜잭션 정책 없이 웹 서비스 클라이언트를 구성하는 것이 안전합니다.

Business Process Choreographer JMS API를 사용한 클라이언트 응용프로그램 개발

JMS(Java Messaging Service) API를 통해 비동기적으로 비즈니스 프로세스 응용프로그램에 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

이 태스크 정보

JMS 클라이언트 응용프로그램은 요청 및 응답 메시지를 JMS API와 교환합니다. 요청 메시지를 작성하기 위해 클라이언트 응용프로그램은 해당 조작의 문서/리터럴 랩퍼를 나타내는 XML 요소로 JMS TextMessage 메시지 본문을 채웁니다.

비즈니스 프로세스의 요구사항

WebSphere Integration Developer로 개발되어 Business Process Choreographer에서 실행되는 비즈니스 프로세스는 JMS API를 통해 액세스하는 특정 규칙을 준수해야 합니다.

요구사항은 다음과 같습니다.

1. 비즈니스 프로세스의 인터페이스는 XML 기반 RPC용 Java API(JAX-RPC 1.1) 스펙에 정의된 "문서/리터럴 랩핑" 스타일을 사용하여 정의해야 합니다. 이는 WebSphere Integration Developer로 개발된 모든 비즈니스 프로세스 및 휴먼 타스크의 기본 스타일입니다.
2. 웹 서비스 조작의 비즈니스 프로세스 및 휴먼 타스크에서 노출된 결함 메시지는 XML 스키마 요소로 정의된 단일 WSDL 메시지로 구성되어야 합니다. 예를 들어, 다음과 같습니다.

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

관련 정보

XML 기반 RPC용 Java API(JAX-RPC) 다운로드 페이지

사용해야 하는 WSDL 유형

JMS 렌더링 권한

JMS 인터페이스의 사용 권한을 부여하려면 WebSphere Application Server에서 보안 설정을 사용해야 합니다.

비즈니스 프로세스 컨테이너가 설치되면 **JMSAPIUser** 역할이 사용자 ID에 맵핑됩니다. 이 사용자 ID는 모든 JMS API 요청을 발행하는 데 사용됩니다. 예를 들어, **JMSAPIUser**가 "사용자 A"에 맵핑된 경우 모든 JMS API 요청이 "사용자 A"에서 시작될 프로세스 엔진에 표시됩니다.

JMSAPIUser 역할에 다음 권한을 지정해야 합니다.

요청	필수 권한
forceTerminate	프로세스 관리자
sendEvent	잠재적 활동 소유자 또는 프로세스 관리자

주: 기타 모든 요청에는 특수 권한이 필요하지 않습니다.

특수 권한은 비즈니스 프로세스 관리자 역할의 사용자에게 부여됩니다. 비즈니스 프로세스 관리자는 특수 역할로서 프로세스 인스턴스의 프로세스 관리자와 차이가 있습니다. 비즈니스 프로세스 관리자에게는 모든 특권이 있습니다.

프로세스 인스턴스가 존재하는 동안에는 사용자 레지스트리에서 프로세스 시작자의 사용자 ID를 삭제할 수 없습니다. 이 사용자 ID를 삭제하면 이 프로세스를 계속 탐색할 수 없습니다. 시스템 로그 파일에서 다음 예외를 수신합니다.

```
no unique ID for: <user ID>
```

JMS 인터페이스 액세스

JMS 인터페이스를 통해 메시지를 전송하고 수신하려면 응용프로그램이 먼저 BPC.cellname.Bus에 대한 연결을 작성하고, 세션을 작성한 다음 메시지 생성자 및 처리자를 생성해야 합니다.

이 태스크 정보

프로세스 서버는 지점간 패러다임을 따르는 JMS(Java Message Service) 메시지를 승인합니다. JMS 메시지를 전송하거나 수신하는 응용프로그램은 다음 조치를 수행해야 합니다.

다음 예제에서는 JMS 클라이언트가 관리 환경(EJB, 응용프로그램 클라이언트 또는 웹 클라이언트 컨테이너)에서 실행된다고 가정합니다.

프로시저

1. BPC.*cellname*.Bus에 대한 연결을 작성하십시오. 클라이언트 응용프로그램의 요청에 대해 사전 구성된 연결 팩토리가 존재하지 않습니다. 클라이언트 응용프로그램은 JMS API의 ReplyConnectionFactory를 사용하거나 자체 연결 팩토리를 작성할 수 있습니다. 이 경우 JNDI(Java Naming and Directory Interface) 검색을 사용하여 연결 팩토리를 검색할 수 있습니다. JNDI 검색 이름은 Business Process Choreographer의 외부 요청 큐를 구성할 때 지정된 이름과 동일해야 합니다. 다음 예제에서는 클라이언트 응용프로그램이 "jms/clientCF"라는 자체 연결 팩토리를 작성한다고 가정합니다.

```
//Obtain the default initial JNDI context.
Context initialContext = new InitialContext();

// Look up the connection factory.
// Create a connection factory that connects to the BPC bus.
// Call it, for example, "jms/clientCF".
// Also configure an appropriate authentication alias.
ConnectionFactory connectionFactory =
            (ConnectionFactory)initialcontext.lookup("jms/clientCF");

// Create the connection.
Connection connection = connectionFactory.createConnection();
```

2. 메시지 생성자 및 처리자를 작성할 수 있도록 세션을 작성하십시오.

```
// Create a transaction session using auto-acknowledgment.
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. 메시지를 전송할 메시지 생성자를 작성하십시오. JNDI 검색 이름은 Business Process Choreographer의 외부 요청 큐를 구성할 때 지정된 이름과 동일해야 합니다.

```
// Look up the destination of the Business Process Choreographer
input queue to
// send messages to.
Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");

// Create a message producer.
MessageProducer producer = session.createProducer(sendQueue);
```

4. 응답을 수신할 메시지 처리자를 작성하십시오. 응답 대상의 JNDI 검색 이름은 사용자 정의 대상을 지정할 수 있지만 기본(Business Process Choreographer에서 정의한) 응답 대상 jms/BFMJMSReplyQueue를 지정할 수도 있습니다. 두 경우 모두 응답 대상이 BPC.<cellname>.Bus에 있어야 합니다.

```
// Look up the destination of the reply queue.
Queue  replyQueue = (Queue) initialcontext.lookup
("jms/BFMJMSReplyQueue");

// Create a message consumer.
MessageConsumer consumer = session.createConsumer(replyQueue);
```

5. 메시지를 전송하십시오.

```
// Start the connection.
connection.start();

// Create a message - see the task descriptions for examples
- and send it.
// This method is defined elsewhere ...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Set mandatory JMS header.
// targetFunctionName is the operation name of JMS API
// (for example, getProcessTemplate, sendMessage)
requestMessage.setStringProperty("TargetFunctionName",
targetFunctionName);

// Set the reply queue; this is mandatory if the replyQueue
// is not the default queue (as it is in this example).
requestMessage.setJMSReplyTo(replyQueue);

// Send the message.
producer.send(requestMessage);

// Get the message ID.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. 응답을 수신하십시오.

```
// Receive the reply message and analyse the reply.
TextMessage  replyMessage = (TextMessage) consumer.receive();

// Get the payload.
String payload = replyMessage.getText();

session.commit();
```

7. 연결을 닫고 자원을 해제하십시오.

```
 // Final housekeeping; free the resources.
session.close();
connection.close();
```

주: 각 트랜잭션 후에는 연결을 닫을 필요가 없습니다. 연결이 시작되어 연결을 닫기 전까지는 많은 요청 및 응답 메시지를 교환할 수 있습니다. 예제는 단일 비즈니스 메소드 내에 단일 호출이 있는 간단한 경우를 보여줍니다.

Business Process Choreographer JMS 메시지의 구조

각 JMS 메시지의 헤더와 본문에는 사전 정의된 구조가 있어야 합니다.

JMS(Java Message Service) 메시지는 다음으로 이루어져 있습니다.

- 메시지 식별 및 라우팅 정보에 대한 메시지 헤더
- 컨텐츠를 보유하는 메시지의 본문(페이로드)

Business Process Choreographer는 텍스트 메시지 형식만을 지원합니다.

메시지 헤더

JMS는 클라이언트가 여러 메시지 헤더 필드에 액세스할 수 있게 합니다.

Business Process Choreographer JMS 클라이언트에서 다음 헤더 필드를 설정할 수 있습니다.

JMSReplyTo
응답을 요청에 전송할 대상입니다. 요청 메시지에서 이 필드를 지정하지 않으면 응답이 Export 인터페이스의 기본 응답 대상에 전송됩니다(Export는 비즈니스 프로세스 컴포넌트의 클라이언트 인터페이스 렌더링임). 이 대상은 `initialContext.lookup("jms/BFMJMSReplyQueue");`을 사용하여 확보할 수 있습니다.

TargetFunctionName
WSDL 조작의 이름입니다(예: "queryProcessTemplates"). 이 필드는 항상 설정해야 합니다. TargetFunctionName은 여기에 설명된 대로 일반 JMS 메시지 인터페이스의 조작을 지정함에 유의하십시오. call 또는 sendMessage 조작을 사용하는 것처럼 간접적으로 호출할 수 있는 구체적 프로세스나 타스크로 제공되는 조작과 혼동해선 안됩니다.

Business Process Choreographer 클라이언트는 다음 헤더 필드도 액세스할 수 있습니다.

JMSMessageID
메시지를 고유하게 식별합니다. 메시지가 전송될 때 JMS 프로바이더로 설정하십시오. 메시지를 전송하기 전에 클라이언트가 JMSMessageID를 설정하면 이를 JMS 프로바이더로 겹쳐씁니다. 메시지 ID가 인증 용도로 필요한 경우 클라이언트는 메시지를 전송한 후 JMSMessageID를 검색할 수 있습니다.

JMSCorrelationID
링크 메시지입니다. 이 필드는 설정하지 마십시오. Business Process Choreographer 응답 메시지는 요청 메시지의 JMSMessageID를 포함합니다.

각 응답 메시지는 다음 JMS 헤더 필드를 포함합니다.

- **IsBusinessException**

  WSDL 출력 메시지의 경우 "False" 또는 WSDL 결함 메시지의 경우는 "true"입니다.

ServiceRuntimeExceptions는 비동기 클라이언트 응용프로그램에 리턴되지 않습니다. JMS 요청 메시지 처리 중 심각한 예외가 발생하면 런타임 장애가 발생하여 이 요청 메시지를 처리 중인 트랜잭션이 롤백됩니다. 그러면 JMS 요청 메시지가 다시 전달됩니다. SCA

내보내기의 일부로 메시지를 처리하는 중(예: 메시지 직렬화를 해제하는 중) 장애가 일찍 발생하면 SCA 내보내기의 수신 대상에 지정된 실패한 전달의 최대 수까지 재시도됩니다. 실패한 전달의 최대 수에 도달하면 Business Process Choreographer 버스의 시스템 예외 대상에 요청 메시지가 추가됩니다. 그러나 비즈니스 플로우 관리자의 SCA 컴포넌트에 의한 요청을 실제로 처리하는 중 장애가 발생하면 WebSphere Process Server의 실패 이벤트 관리 인프라에서 실패한 요청 메시지가 핸들됩니다. 즉, 재시도가 예외 상황을 해결하지 못하면 실패 이벤트 관리 데이터베이스에서 메시지가 끝날 수 있습니다.

메시지 본문

비즈니스 프로세스 또는 휴먼 타스크에 표시되는 조작은 문서/리터럴 랩퍼 스타일을 충족해야 합니다. JMS 메시지 본문은 조작의 문서/리터럴 랩퍼 요소를 나타내는 XML 문서가 포함된 문자열입니다. JMS 메시지 인터페이스의 일반 조작은 비즈니스 프로세스 또는 휴먼 타스크로 구현된 조작과 문서-랩퍼 요소를 주고 받습니다.

다음 예제는 간단한 유효 요청 메시지 본문을 나타냅니다.

```
<bfm:queryProcessTemplates
     xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0">
  <whereClause>PROCESS_TEMPLATE.STATE IN (1)</whereClause>
</bfm:queryProcessTemplates>
```

다음 예제는 더 복잡한 유효 요청 메시지 본문을 나타냅니다. 클라이언트 응용프로그램에는 특정 프로세스에 메시지를 제출하기 위한 sendMessage API 조작이 있습니다. 프로세스 입력 메시지는 API 매개변수 중 하나입니다. 이 메시지는 고객 프로세스가 표시하는 비즈니스 조작의 입력 메시지입니다. 프로세스에는 메시지를 이용하는 수신 활동이 있습니다.

`bfm:sendMessage` 요소는 JMS API 조작의 문서 랩퍼 요소입니다. 여기에는 프로세스가 구현하는 조작에 대한 문서 랩퍼 요소인 `cns:updateCustomer` 요소가 포함됩니다. 예를 들어, 이 프로세스에는 `cns:customerProcessPortType` WSDL 포트 유형 및 `updateCustomer` WSDL 조작을 참조하는 `bpel:receive` activity 활동이 있습니다.

```
<bfm:sendMessage
     xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0">
  <processTemplateName>customerProcessTemplate</processTemplateName>
  <portType xmlns:cns="http://example.com/customerProcess">cns:customerProcessPortType</portType>
  <operation>updateCustomer</operation>
  <cns:updateCustomer xmlns:cns="http://example.com/customerProcess">
    <street>1600 Pennsylvania Avenue Northwest</street>
    <city>Washington, DC 20006</city>
  </cns:updateCustomer>
</bfm:sendMessage>
```

관련 태스크

519 페이지의『비즈니스 예외의 응답 메시지 점검』
JMS 클라이언트 응용프로그램은 비즈니스 예외에 대한 모든 응답 메시지의 메시지 헤더를 확인해야 합니다.

JMS 클라이언트 응용프로그램의 아티팩트 복사

WebSphere 프로세스 서버 환경에서 여러 아티팩트를 복사해서 JMS 클라이언트 응용프로그램을 보다 쉽게 작성할 수 있습니다.

이 태스크 정보

JMS 메시지 본문을 작성하기 위해 BOXMLSerializer를 사용하는 경우에만 이 아티팩트가 필수입니다. JMS API의 경우 이 아티팩트는 다음과 같습니다.

```
BFMIF.wsdl
BFMIF.xsd
BPCGen.xsd
wsa.xsd
```

WebSphere Process Server 환경에서 개발 환경으로 해당 파일을 공개하고 내보내야 합니다.

JMS 응용프로그램에 대한 비즈니스 프로세스 WSDL 파일 공개

WSDL 파일을 공개하려면 관리 콘솔을 사용하십시오.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.
2. **응용프로그램 → SCA 모듈**을 클릭하십시오.

   주: 응용프로그램 → 응용프로그램 유형 → WebSphere 엔터프라이즈 응용프로그램을 클릭하여 사용 가능한 모든 엔터프라이즈 응용프로그램 목록을 표시할 수도 있습니다.
3. SCA 모듈 또는 응용프로그램 목록에서 **BPEContainer** 응용프로그램을 선택하십시오.
4. **추가 특성** 목록에서 **WSDL 파일 공개**를 선택하십시오.
5. 목록에서 .zip 파일을 클릭하십시오.
6. 표시된 파일 다운로드 창에서 **저장**을 클릭하십시오.
7. 로컬 폴더를 찾아보고 **저장**을 클릭하십시오.

결과

내보낸 .zip 파일의 이름은 BPEContainer_WSDLFiles.zip입니다. .zip 파일에는 WSDL 파일과 WSDL 파일에서 참조되는 모든 XSD 파일이 들어 있습니다.

비즈니스 예외의 응답 메시지 점검

JMS 클라이언트 응용프로그램은 비즈니스 예외에 대한 모든 응답 메시지의 메시지 헤더를 확인해야 합니다.

이 태스크 정보

JMS 클라이언트 응용프로그램은 먼저 응답 메시지 헤더의 **IsBusinessException** 특성을 확인해야 합니다.

예를 들어, 다음과 같습니다.

예

```
// receive response message
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject()
.receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse is a bussiness fault
  // any api can end w/a processFaultMsg
  // the call api also w/a businessFaultMsg
 }
else {
  // strResponse is the output message
}
```

관련 개념

515 페이지의『Business Process Choreographer JMS 메시지의 구조』
각 JMS 메시지의 헤더와 본문에는 사전 정의된 구조가 있어야 합니다.

예제: Business Process Choreographer JMS API를 사용하여 장기 실행 중 프로세스 실행

이 예제에서는 장기 실행 중인 프로세스를 이용하여 작업하기 위해 JMS API를 사용하는 일반 클라이언트 응용프로그램을 작성하는 방법을 표시합니다.

프로시저

1. 513 페이지의『JMS 인터페이스 액세스』에 설명된 대로 JMS 환경을 설정하십시오.
2. 설치된 프로세스 정의 목록을 확보하십시오.
   - queryProcessTemplates를 전송하십시오.
   - 그 결과 ProcessTemplate 오브젝트 목록이 리턴됩니다.

3. 시작 활동 목록을 확보하십시오(createInstance="yes"로 선택하거나 수신).
   - getStartActivities를 전송하십시오.
   - 그 결과 InboundOperationTemplate 오브젝트 목록이 리턴됩니다.
4. 입력 메시지를 작성하십시오. 이는 환경에 특정하며 사전에 전개된 프로세스 특정 아티팩트를 사용해야 할 수도 있습니다.
5. 프로세스 인스턴스를 작성하십시오.
   - sendMessage를 발행하십시오.

   JMS API의 경우 call 조작을 사용하여 비즈니스 프로세스가 제공하는 장기 실행 요청-응답 조작과 상호작용할 수도 있습니다. 이 조작은 오랜 기간 후에도 조작 결과나 결함을 지정된 응답 대상에 리턴합니다. 따라서 call 조작을 사용하면 프로세스 출력 또는 결함 메시지를 얻기 위해 query 및 getOutputMessage 조작을 사용할 필요가 없습니다.
6. 옵션: 다음 단계를 반복하여 프로세스 인스턴스로부터 출력 메시지를 확보하십시오.
   a. query를 발행하여 프로세스 인스턴스의 완료된 상태를 확보하십시오.
   b. getOutputMessage를 발행하십시오.
7. 옵션: 프로세스에 노출된 추가 조작에 대해 작업하십시오.
   a. InboundOperationTemplate 오브젝트 목록을 얻으려면 getWaitingActivities 또는 getActiveEventHandlers를 발행하십시오.
   b. 입력 메시지를 작성하십시오.
   c. sendMessage로 메시지를 전송하십시오.
8. 옵션: getCustomProperties 및 setCustomProperties를 사용하여 포함된 활동 또는 프로세스에 정의된 사용자 정의 특성을 확보하고 설정하십시오.
9. 프로세스 인스턴스를 이용한 작업을 완료하십시오.
   a. delete 및 terminate를 전송하여 장기 실행 프로세스에 대한 작업을 완료하십시오.

JSP 컴포넌트를 사용하여 비즈니스 프로세스 및 휴먼 타스크용 웹 응용프로그램 개발

Business Process Choreographer는 여러 가지 JSF(JavaServer Faces) 컴포넌트를 제공합니다. 이들 컴포넌트를 확장 및 통합하여 웹 응용프로그램에 비즈니스 프로세스 및 휴먼 타스크를 추가할 수 있습니다.

이 태스크 정보

WebSphere Integration Developer를 사용하여 웹 응용프로그램을 빌드할 수 있습니다. 휴먼 타스크를 포함하는 응용프로그램의 경우, JSF 사용자 정의 클라이언트를 생성

할 수 있습니다. JSF 클라이언트 생성에 대한 자세한 정보는 WebSphere Integration Developer의 Information Center를 참조하십시오.

Business Process Choreographer가 제공하는 JSF 컴포넌트를 사용하여 웹 클라이언트를 개발할 수 있습니다.

프로시저

1. 동적 프로젝트를 작성한 후 JSF 기본 컴포넌트를 포함하도록 웹 프로젝트 기능 특성을 변경하십시오.

   웹 프로젝트 작성에 대한 자세한 정보는 WebSphere Integration Developer의 Information Center를 참조하십시오.

2. 전제조건인 Business Process Choreographer Explorer JAR(Java archive)을 추가하십시오.

   프로젝트의 `WEB-INF/lib` 디렉토리에 다음 파일을 추가하십시오.

   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar
   - bpcjsfcomponents.jar

   WebSphere Process Server에서 이 파일은 모두 다음 디렉토리에 있습니다.

   - Windows Windows 플랫폼: *install_root*₩ProcessChoreographer₩client
   - Linux UNIX Linux® 및 UNIX® 플랫폼: *install_root*/ProcessChoreographer/client

3. 웹 응용프로그램 전개 설명자 `web.xml` 파일에 필요한 EJB 참조를 추가하십시오.

```
<ejb-ref id="EjbRef_1">
  <ejb-ref-name>ejb/BusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
  <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
<ejb-ref id="EjbRef_2">
  <ejb-ref-name>ejb/HumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.task.api.HumanTaskManagerHome</home>
  <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
<ejb-local-ref id="EjbLocalRef_1">
  <ejb-ref-name>ejb/LocalBusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
  <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
</ejb-local-ref>
<ejb-local-ref id="EjbLocalRef_2">
  <ejb-ref-name>ejb/LocalHumanTaskManagerEJB</ejb-ref-name>
```

```
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
    <local>com.ibm.task.api.LocalHumanTaskManager</local>
  </ejb-local-ref>
```

4. JSF 응용프로그램에 Business Process Choreographer Explorer JSF 컴포넌트를 추가하십시오.

   a. 응용프로그램에 필요한 태그 라이브러리 참조를 JSP(JavaServer Pages) 파일에 추가하십시오. 일반적으로 JSF 및 HTML 태그 라이브러리 및 JSF 컴포넌트에 필요한 태그 라이브러리가 필요합니다.

      - `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>`
      - `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`
      - `<%@ taglib uri="http://com.ibm.bpe.jsf/taglib" prefix="bpe" %>`

   b. JSP 페이지의 본문에 `<f:view>` 태그를 추가하고 `<f:view>` 태그에 `<h:form>` 태그를 추가하십시오.

   c. JSP 파일에 JSF 컴포넌트를 추가하십시오.

      응용프로그램에 따라 List 컴포넌트, Details 컴포넌트, CommandBar 컴포넌트 또는 Message 컴포넌트를 JSP 파일에 추가하십시오. 각 컴포넌트의 다중 인스턴스를 추가할 수 있습니다.

   d. JSF 구성 파일에 관리 Bean을 구성하십시오.

      기본 구성 파일은 `faces-config.xml` 파일입니다. 이 파일은 웹 응용프로그램의 `WEB-INF` 디렉토리에 있습니다.

      JSP 파일에 추가하는 컴포넌트에 따라 조회 및 기타 랩퍼 오브젝트에 대한 참조도 JSF 구성 파일에 추가해야 합니다. 올바른 오류 처리를 위해서는 JSF 구성 파일의 오류 페이지에 대한 탐색 대상 및 오류 Bean을 모두 정의해야 합니다. 오류 Bean의 이름에 대해 `BPCError`를 사용하고 오류 페이지의 탐색 대상 이름에 대해 `error`를 사용하는지 확인하십시오.

```
<faces-config>
...
<managed-bean>
  <managed-bean-name>BPCError</managed-bean-name>
  <managed-bean-class>com.ibm.bpc.clientcore.util.ErrorBeanImpl
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

...
<navigation-rule>
...
<navigation-case>
<description>
The general error page.
</description>
```

```
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

오류 페이지를 트리거하는 오류 상황에서 예외는 오류 Bean에 설정됩니다.

e. JSF 컴포넌트를 지원하는 데 필요한 사용자 정의 코드를 구현하십시오.

5. 응용프로그램을 전개하십시오.

Network Deployment 환경에서 응용프로그램을 전개 중인 경우에는 대상 자원 JNDI(Java Naming and Directory Interface) 이름을 셀에서 비즈니스 플로우 관리자 및 휴먼 타스크 관리자 API를 찾을 수 있는 값으로 변경하십시오.

- 비즈니스 프로세스 컨테이너가 동일한 관리 셀의 다른 서버에 구성된 경우에는 이름의 구조가 다음과 같습니다.

  cell/*nodes*/*nodename*/*servers*/*servername*
  /com/ibm/bpe/api/BusinessManagerHome
  cell/*nodes*/*nodename*/*servers*/*servername*
  /com/ibm/task/api/HumanTaskManagerHome

- 비즈니스 프로세스 컨테이너가 동일한 셀의 클러스터에 구성된 경우 이름의 구조는 다음과 같습니다.

  cell/*clusters*/*clustername*/com/ibm/bpe/api/BusinessFlowManagerHome
  cell/*clusters*/*clustername*/com/ibm/task/api/HumanTaskManagerHome

EJB 참조를 JNDI 이름으로 맵핑하거나 ibm-web-bnd.xmi 파일에 참조를 수동으로 추가하십시오.

다음 표는 참조 바인딩 및 해당 기본 맵핑을 표시합니다.

표 70. JNDI 이름에 대한 참조 바인딩 맵핑

참조 바인딩	**JNDI 이름**	주석
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	원격 세션 Bean
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	로컬 세션 Bean
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	원격 세션 Bean
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	로컬 세션 Bean

결과

전개된 웹 응용프로그램에는 Business Process Choreographer Explorer 컴포넌트가 제공하는 기능이 포함되어 있습니다.

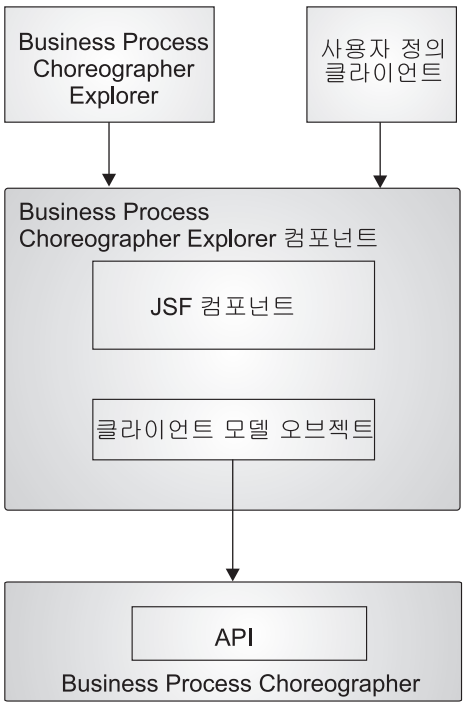
다음에 수행할 작업

프로세스 및 타스크 메시지에 사용자 정의 JSP를 사용 중인 경우 JSP를 전개하는 데 사용된 웹 모듈을 사용자 정의 JSF 클라이언트가 맵핑된 동일한 서버로 맵핑해야 합니다.

Business Process Choreographer Explorer 컴포넌트

Business Process Choreographer Explorer 컴포넌트는 JSF(JavaServer Faces) 테크놀러지를 기초로 하는 구성 가능하며 재사용할 수 있는 요소 세트입니다.이 요소를 웹 응용프로그램에 임베디드할 수 있습니다. 그런 다음, 웹 응용프로그램은 설치된 프로세스 및 휴먼 타스크 응용프로그램에 액세스할 수 있습니다.

이 컴포넌트는 JSF 컴포넌트 세트와 클라이언트 모델 오브젝트 세트로 구성됩니다. Business Process Choreographer, Business Process Choreographer Explorer 및 기타 사용자 정의 클라이언트에 대한 컴포넌트의 관계는 다음 그림으로 표시됩니다.



JSF 컴포넌트

Business Process Choreographer Explorer 컴포넌트는 다음과 같은 JSF 컴포넌트를 포함합니다. 비즈니스 프로세스 및 휴먼 타스크에 대해 작업하기 위한 웹 응용프로그램을 빌드할 때 이들 JSF 컴포넌트를 JSP(JavaServer Pages) 파일에 임베드합니다.

- List 컴포넌트

List 컴포넌트는 응용프로그램 오브젝트의 목록을 테이블에 표시합니다(예: 타스크, 활동, 프로세스 인스턴스, 프로세스 템플리트, 작업 항목 또는 에스컬레이션). 이 컴포넌트에는 연관된 목록 핸들러가 포함되어 있습니다.

- Details 컴포넌트

  Details 컴포넌트는 타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시합니다. 이 컴포넌트에는 연관된 세부사항 핸들러가 포함되어 있습니다.

- CommandBar 컴포넌트

  CommandBar 컴포넌트는 단추가 있는 표시줄을 표시합니다. 이 단추는 자세히 보기에 있는 오브젝트 또는 목록에서 선택된 오브젝트에 대해 조작되는 명령을 나타냅니다. 이 오브젝트는 목록 핸들러나 세부사항 핸들러에서 제공됩니다.

- Message 컴포넌트

  Message 컴포넌트는 서비스 데이터 오브젝트(SDO) 또는 단순 유형을 포함할 수 있는 메시지를 표시합니다.

클라이언트 모델 오브젝트

클라이언트 모델 오브젝트는 JSF 컴포넌트와 함께 사용됩니다. 이 오브젝트는 기본 Business Process Choreographer API의 일부 인터페이스를 구현하며 원래 오브젝트를 랩핑합니다. 클라이언트 모델 오브젝트는 일부 특성에 대한 레이블 및 변환기에 대한 자국어 지원을 제공합니다.

JSF 컴포넌트의 오류 처리

JSF(JavaServer Faces) 컴포넌트는 사전 정의된 관리 Bean, BPCError를 오류 처리에 사용합니다. 오류 페이지를 트리거하는 오류 상황에서 예외는 오류 Bean에 설정됩니다.

이 Bean은 com.ibm.bpc.clientcore.util.ErrorBean 인터페이스를 구현합니다. 오류 페이지는 다음 상황에서 표시됩니다.

- 목록 핸들러에 대해 정의된 조회를 실행하는 중에 오류가 생성되고, 이 오류가 명령의 execute 메소드에 의한 ClientException 오류인 경우
- 명령의 execute 메소드에 의해 ClientException 오류가 생성되고 이 오류가 ErrorsInCommandException 오류가 아니거나 CommandBarMessage 인터페이스를 구현하는 오류가 아닌 경우
- 오류 메시지가 컴포넌트에 표시되고 메시지의 하이퍼링크를 따라가는 경우

com.ibm.bpc.clientcore.util.ErrorBeanImpl 인터페이스의 기본 구현이 사용 가능합니다.

이 인터페이스는 다음과 같이 정의됩니다.

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * This setter method call allows a locale and
     * the exception to be passed. This allows the
     * getExceptionMessage methods to return localized Strings
     *
     */
    public void setException(Exception ex, Locale locale);

    public Exception getException();
    public String getStack();
    public String getNestedExceptionMessage();
    public String getNestedExceptionStack();
    public String getRootExceptionMessage();
    public String getRootExceptionStack();

    /*
     * This method returns the exception message
     * concatenated recursively with the messages of all
     * the nested exceptions.
     */
    public String getAllExceptionMessages();

    /*
     * This method is returns the exception stack
     * concatenated recursively with the stacks of all
     * the nested exceptions.
     */
    public String getAllExceptionStacks();
}
```

관련 개념

532 페이지의『List 컴포넌트의 목록 처리』

List 컴포넌트를 사용하면 JSF 응용프로그램의 목록을 표시하면 com.ibm.bpe.jsf.handler.BPCListHandler 클래스가 제공하는 오류 처리 기능을 이용할 수 있습니다.

클라이언트 모델 오브젝트의 기본 변환기 및 레이블

클라이언트 모델 오브젝트는 Business Process Choreographer API의 해당 인터페이스를 구현합니다.

List 컴포넌트 및 Details 컴포넌트는 모든 Bean에서 작동합니다. Bean의 모든 특성을 표시할 수 있습니다. 그러나 Bean의 특성에 사용되는 변환 및 레이블을 설정하려면 List 컴포넌트의 `column` 태그나 Details 컴포넌트의 `property` 태그를 사용해야 합니다. 변환 및 레이블을 설정하는 대신 다음 정적 메소드를 정의해서 특성의 기본 변환기와 레이블을 정의할 수 있습니다. 다음 정적 메소드를 정의할 수 있습니다.

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

다음 표는 해당 비즈니스 플로우 관리자 및 휴먼 타스크 관리자 API 클래스를 구현하고 특성의 기본 레이블 및 변환기를 제공하는 클라이언트 모델 오브젝트를 보여줍니다. 이러한 인터페이스 랩핑은 로케일 구분 레이블 및 특성 세트에 대한 변환기를 제공합니다. 다음 표는 Business Process Choreographer 인터페이스의 해당 클라이언트 모델 오브젝트에 대한 맵핑을 보여줍니다.

표 71. Business Process Choreographer가 클라이언트 모델 오브젝트로 맵핑되는 방법

Business Process Choreographer 인터페이스	**클라이언트 모델 오브젝트 클래스**
com.ibm.bpe.api.ActivityInstanceData	com.ibm.bpe.clientmodel.bean.ActivityInstanceBean
com.ibm.bpe.api.ActivityServiceTemplateData	com.ibm.bpe.clientmodel.bean.ActivityServiceTemplateBean
com.ibm.bpe.api.ProcessInstanceData	com.ibm.bpe.clientmodel.bean.ProcessInstanceBean
com.ibm.bpe.api.ProcessTemplateData	com.ibm.bpe.clientmodel.bean.ProcessTemplateBean
com.ibm.task.api.Escalation	com.ibm.task.clientmodel.bean.EscalationBean
com.ibm.task.api.Task	com.ibm.task.clientmodel.bean.TaskInstanceBean
com.ibm.task.api.TaskTemplate	com.ibm.task.clientmodel.bean.TaskTemplateBean

JSF 응용프로그램에 List 컴포넌트 추가

클라이언트 모듈 오브젝트의 목록(예: 비즈니스 프로세스 인스턴스 또는 타스크 인스턴스)을 표시하려면 Business Process Choreographer Explorer List 컴포넌트를 사용하십시오.

프로시저

1. JSP(JavaServer Pages) 파일에 List 컴포넌트를 추가하십시오.

   `h:form` 태그에 `bpe:list` 태그를 추가하십시오. `bpe:list` 태그는 모델 속성을 포함해야 합니다. 목록의 각 행에 표시될 오브젝트의 특성을 추가하려면 `bpe:list` 태그에 `bpe:column` 태그를 추가하십시오.

   다음 예는 타스크 인스턴스를 표시하기 위해 List 컴포넌트를 추가하는 방법을 보여줍니다.

   ```
   <h:form>

      <bpe:list model="#{TaskPool}">
         <bpe:column name="name" action="taskInstanceDetails" />
         <bpe:column name="state" />
         <bpe:column name="kind" />
         <bpe:column name="owner" />
         <bpe:column name="originator" />
      </bpe:list>

   </h:form>
   ```

   이 모델 속성은 관리 Bean인 TaskPool을 나타냅니다. 관리 Bean은 목록이 반복하여 개별 행에 표시하는 Java 오브젝트의 목록을 제공합니다.

2. `bpe:list` 태그에 참조된 관리 Bean을 구성하십시오.

List 컴포넌트의 경우, 이 관리 Bean은
com.ibm.bpe.jsf.handler.BPCListHandler 클래스의 인스턴스여야 합니다.

다음 예는 TaskPool 관리 Bean을 구성 파일에 추가하는 방법을 보여줍니다.

```
<managed-bean>
<managed-bean-name>TaskPool</managed-bean-name>
<managed-bean-class>com.ibm.bpe.jsf.handler.BPCListHandler
</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>query</property-name>
      <value>#{TaskPoolQuery}</value>
   </managed-property>
   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>TaskPoolQuery</managed-bean-name>
<managed-bean-class>sample.TaskPoolQuery</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
   <managed-property>
     <property-name>type</property-name>
     <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>htmConnection</managed-bean-name>
<managed-bean-class>com.ibm.task.clientmodel.HTMConnection<
/managed-bean-class>
<managed-bean-scope>application</managed-bean-scope>
   <managed-property>
      <property-name>jndiName</property-name>
      <value>java:comp/env/ejb/LocalHumanTaskManagerEJB</value>
   </managed-property>
</managed-bean>
```

이 예는 TaskPool에 두 개의 구성 가능한 특성(query 및 type)가 있는 것을 보여줍니다. 조회 특성의 값은 다른 관리 Bean인 TaskPoolQuery를 나타냅니다. 유형 특성의 값은 Bean 클래스를 지정하며, 표시된 목록의 열에 해당 특성이 표시됩니다. 연관된 조회 인스턴스는 특성 유형도 가질 수 있습니다. 특성 유형이 지정된 경우, 이는 목록 핸들러에 지정된 유형과 동일해야 합니다.

조회 결과를 강력한 유형의 Bean으로 표현할 수 있으면 모든 유형의 조회 논리를 JSF 응용프로그램에 추가할 수 있습니다. 예를 들어, TaskPoolQuery는 com.ibm.task.clientmodel.bean.TaskInstanceBean 오브젝트의 목록을 사용하여 구현됩니다.

3. 목록 핸들러가 참조하는 관리 Bean의 사용자 정의 코드를 추가하십시오.

다음 예는 TaskPool 관리 bean에 대한 사용자 정의 코드를 추가하는 방법을 보여 줍니다.

```
public class TaskPoolQuery implements Query {

  public List execute throws ClientException {

    // Examine the faces-config file for a managed bean "htmConnection".
    //
    FacesContext ctx = FacesContext.getCurrentInstance();
    Application  app = ctx.getApplication();
    ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
    htmConnection = (HTMConnection) htmVb.getValue(ctx);
    HumanTaskManagerService taskService =
         htmConnection.getHumanTaskManagerService();

    // Then call the actual query method on the Human

    Task Manager service.
    //
    // Add the database columns for all of the properties

    you want to show
    // in your list to the select statement
    //
    QueryResultSet queryResult = taskService.query(
     "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND,

 TASK.STATE, TASK.TYPE,"
      + "TASK.STARTER, TASK.OWNER, TASK.STARTED,

      TASK.ACTIVATED, TASK.DUE,
         TASK.EXPIRES, TASK.PRIORITY",
      "TASK.KIND IN(101,102,105) AND TASK.STATE IN(2)
        AND WORK_ITEM.REASON IN (1)",
      (String)null,
      (Integer)null,
      (TimeZone)null);
    List applicationObjects = transformToTaskList ( queryResult );
    return applicationObjects ;
  }


  private List transformToTaskList(QueryResultSet result) {

ArrayList array = null;
int entries = result.size();
array = new ArrayList( entries );

// Transforms each row in the QueryResultSet to a task instance beans.
 for (int i = 0; i < entries; i++) {
   result.next();
   array.add( new TaskInstanceBean( result, connection ));
 }
 return array ;
  }
}
```

TaskPoolQuery bean은 Java 오브젝트의 특성을 조회합니다. 이 Bean은 com.ibm.bpc.clientcore.Query 인터페이스를 구현해야 합니다. 목록 핸들러는 내용을 새로 고칠 때 조회의 execute 메소드를 호출합니다. 이 호출은 Java 오브젝트 목록을 리턴합니다. getType 메소드는 리턴된 Java 오브젝트의 클래스 이름을 리턴해야 합니다.

결과

JSF 응용프로그램은 이제 요청된 오브젝트 목록의 특성(예: 상태, 종류, 소유자 및 사용 가능한 타스크 인스턴스의 작성자)를 표시하는 JavaServer 페이지를 포함합니다.

목록 처리 방법

List 컴포넌트의 모든 인스턴스가 com.ibm.bpe.jsf.handler.BPCListHandler 클래스의 인스턴스와 연관됩니다.

이 목록 핸들러는 연관된 목록에서 선택된 항목을 추적하고 공고 메커니즘을 제공해서 목록 항목을 다른 종류의 항목에 대한 세부사항 페이지와 연관시킵니다. 목록 핸들러는 `bpe:list` 태그의 **model** 속성을 통해 List 컴포넌트에 바인드됩니다.

목록 핸들러의 공고 메커니즘은 com.ibm.bpe.jsf.handler.ItemListener 핸들러를 사용하여 구현됩니다. JSF(JavaServer Faces) 응용프로그램의 구성 파일에서 이 인터페이스의 구현을 등록할 수 있습니다.

공고는 목록의 링크를 클릭하면 트리거됩니다. 링크는 **action** 속성이 설정된 모든 열에 대해 렌더링됩니다. **action** 속성 값은 JSF 탐색 대상을 리턴하는 JSF 조치 메소드 또는 JSF 탐색 대상입니다.

또한 BPCListHandler 클래스도 refreshList 메소드를 제공합니다. JSF 메소드 바인딩에서 이 메소드를 사용하여 조회를 다시 실행하기 위한 사용자 인터페이스 제어를 구현할 수 있습니다.

조회 구현

목록 핸들러를 사용하여 모든 종류의 오브젝트와 특성을 표시할 수 있습니다. 목록 컨텐츠는 목록 핸들러에 대해 구성되는 com.ibm.bpc.clientcore.Query 인터페이스의 구현이 리턴하는 오브젝트 목록에 따라 다르게 표시됩니다. BPCListHandler 클래스의 setQuery 메소드를 사용하여 조회를 프로그래밍하여 설정하거나 응용프로그램의 JSF 구성 파일에서 쿼리를 구성할 수 있습니다.

Business Process Choreographer API를 비롯하여, 컨텐츠 관리 시스템이나 데이터베이스와 같은 응용프로그램에서 액세스할 수 있는 다른 소스 정보에 대해서도 조회를 실행할 수 있습니다. 유일한 요구사항은 조회의 결과가 execute 메소드에 의한 오브젝트의 `java.util.List`로 리턴되어야 한다는 것입니다.

리턴된 오브젝트의 유형은 조회가 정의된 목록의 열에 표시되는 모든 특성에 대해 적절한 Getter 메소드를 사용할 수 있다는 것을 보장해야 합니다. faces 구성 파일에 정의된 BPCListHandler 인스턴스의 유형 특성 값을 리턴된 오브젝트의 완전한 클래스 이름으로 설정하여 리턴된 오브젝트의 유형이 목록 정의에 맞는지를 확인할 수 있습니다. 조회 구현의 getType 호출에서 이 이름을 리턴할 수 있습니다. 런타임 시 목록 핸들러는 오브젝트 유형이 정의에 맞는지 확인합니다.

오류 메시지를 목록의 특정 항목으로 맵핑하기 위해 조회에서 리턴된 오브젝트가 서명 `public Object getID()`을 가지고 메소드를 구현해야 합니다.

기본 변환기 및 레이블

조회에서 리턴된 항목은 Bean이여야 하며 클래스는 BPCListHandler 클래스 또는 com.ibm.bpc.clientcore.Query 인터페이스의 정의 유형에 지정된 클래스와 일치해야 합니다. 또한 List 컴포넌트는 항목 클래스 또는 수퍼클래스가 다음 메소드를 구현하는지 확인합니다.

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

이 메소드가 Bean에 대해 정의되면 List 컴포넌트는 레이블을 목록의 기본 레이블로 사용하고 SimpleConverter를 특성에 대한 기본 변환기로 사용합니다. `bpe:list` 태그의 **label** 및 **converterID** 속성으로 이 설정을 겹쳐쓸 수 있습니다. 자세한 정보는 SimpleConverter 인터페이스 및 ColumnTag class에 대한 Javadoc을 참조하십시오.

사용자 특정 시간대 정보

JSF(JavaServer Faces) 컴포넌트는 List 컴포넌트의 사용자 특정 시간대 정보를 처리할 유틸리티를 제공합니다.

BPCListHandler 클래스는 com.ibm.bpc.clientcore.util.User 인터페이스를 사용하여 각 사용자의 시간대와 로케일 정보를 가져옵니다. List 컴포넌트는 JSF(JavaServer Faces) 구성 파일에서 관리 Bean 이름인 **user**로 구성된 인터페이스의 구현을 예상합니다. 이 항목이 구성 파일에서 누락되면 WebSphere Process Server가 실행 중인 시간대가 리턴됩니다.

com.ibm.bpc.clientcore.util.User 인터페이스가 다음과 같이 정의됩니다.

```
public interface User {

    /**
     * The locale used by the client of the user.
     * @return Locale.
     */
    public Locale getLocale();
   /**
    * The time zone used by the client of the user.
    * @return TimeZone.
```

```
     */
     public TimeZone getTimeZone();

    /**
     * The name of the user.
     * @return name of the user.
     */
     public String getName();
}
```

List 컴포넌트의 목록 처리

List 컴포넌트를 사용하면 JSF 응용프로그램의 목록을 표시하면 com.ibm.bpe.jsf.handler.BPCListHandler 클래스가 제공하는 오류 처리 기능을 이용할 수 있습니다.

조회나 명령을 실행하는 중 오류가 발생한 경우

조회를 실행하는 중에 오류가 발생하는 경우, BPCListHandler 클래스는 불충분한 액세스 권한 때문에 발생한 오류인지 아니면 다른 예외 때문에 발생한 오류인지를 구분합니다. 불충분한 액세스 권한 때문에 발생한 오류를 발견하려면 조회의 execute 메소드에서 발생한 ClientException의 **rootCause** 매개변수가 com.ibm.bpe.api.EngineNotAuthorizedException 또는 com.ibm.task.api.NotAuthorizedException 예외여야 합니다. List 컴포넌트는 조회 결과 대신에 오류 메시지를 표시합니다.

불충분한 액세스 권한 때문에 발생한 오류가 아닌 경우, BPCListHandler 클래스는 예외 오브젝트를 JSF 응용프로그램 구성 파일의 BPCError 키가 정의한 com.ibm.bpc.clientcore.util.ErrorBean 인터페이스의 구현에 전달합니다. 예외가 설정되면 오류 탐색 대상이 호출됩니다.

목록에 표시된 항목을 사용하여 작업하는 중 오류가 발생한 경우

BPCListHandler 클래스는 com.ibm.bpe.jsf.handler.ErrorHandler 인터페이스를 구현합니다. setErrors 메소드의 java.util.Map 유형의 맵 매개변수를 가진 오류에 대한 정보를 제공합니다. 이 맵은 ID(키로 사용)와 예외(값으로 사용)를 포함합니다. 이 ID는 오류를 일으킨 오브젝트의 getID 메소드에서 리턴된 값이어야 합니다. 맵이 설정되고 모든 ID가 목록에 표시된 모든 항목과 일치하는 경우, 목록 핸들러는 오류 메시지를 포함한 열을 목록에 자동으로 추가합니다.

목록에 날짜가 지난 오류 메시지를 추가하지 않으려면 오류 맵을 재설정하십시오. 다음 상황에서는 맵이 자동으로 재설정됩니다.

- BPCListHandler 클래스의 refreshList 메소드가 호출되는 경우
- BPCListHandler 클래스에 새 조회가 설정되는 경우

- 목록의 항목에 조치를 트리거하는 데 CommandBar 컴포넌트를 사용하는 경우. CommandBar 컴포넌트는 이 메커니즘을 오류 처리의 한 방법으로 사용합니다.

관련 개념

525 페이지의『JSF 컴포넌트의 오류 처리』

JSF(JavaServer Faces) 컴포넌트는 사전 정의된 관리 Bean, BPCError를 오류 처리에 사용합니다. 오류 페이지를 트리거하는 오류 상황에서 예외는 오류 Bean에 설정됩니다.

List 컴포넌트: 태그 정의

Business Process Choreographer Explorer의 List 컴포넌트는 테이블에 있는 오브젝트의 목록을 표시합니다(예: 타스크, 활동, 프로세스 인스턴스, 프로세스 템플리트, 작업 항목 및 에스컬레이션).

List 컴포넌트는 JSF 컴포넌트 태그 `bpe:list` 및 `bpe:column`으로 구성됩니다. `bpe:column` 태그는 `bpe:list` 태그의 하위 요소입니다.

컴포넌트 클래스

com.ibm.bpe.jsf.component.ListComponent

예제 구문

```
<bpe:list model="#{ProcessTemplateList}">
          rows="20"
          styleClass="list"
          headerStyleClass="listHeader"
          rowClasses="normal">

    <bpe:column name="name" action="processTemplateDetails"/>
    <bpe:column name="validFromTime"/>
    <bpe:column name="executionMode" label="Execution mode"/>
    <bpe:column name="state" converterID="my.state.converter"/>
    <bpe:column name="autoDelete"/>
    <bpe:column name="description"/>

</bpe:list>
```

태그 속성

`bpe:list` 태그의 본문에는 `bpe:column` 태그만 들어갈 수 있습니다. 테이블이 렌더링될 때 List 컴포넌트는 응용프로그램 오브젝트의 목록을 반복하고 각 오브젝트에 대한 모든 열을 렌더링합니다.

표 72. bpe:list 속성

속성	필수 여부	설명
buttonStyleClass	아니오	바닥글 영역의 단추를 렌더링하는 데 사용되는 계단식 스타일시트(CSS) 스타일 클래스

표 72. bpe:list 속성 (계속)

속성	필수 여부	설명
cellStyleClass	아니오	개별 테이블 셀을 렌더링하기 위한 CSS 스타일 클래스
checkbox	아니오	다중 항목을 선택하기 위한 선택란이 표현되는지 여부를 판별합니다. 이 속성은 `true` 또는 `false` 값을 가집니다. 값을 `true`로 설정하면 선택란 열이 렌더링됩니다.
headerStyleClass	아니오	테이블 헤더를 렌더링하기 위한 CSS 스타일 클래스
model	예	com.ibm.bpe.jsf.handler.BPCListHandler 클래스의 관리 Bean에 대한 값 바인딩
rows	아니오	페이지에 표시된 행 수입니다. 항목 수가 행 수를 초과할 경우 페이징 단추가 테이블 끝에 표시됩니다. 이 속성에 대한 값 표현식은 지원되지 않습니다.
rowClasses	아니오	테이블의 행을 렌더링하기 위한 CSS 스타일 클래스
selectAll	아니오	이 속성을 `true`로 설정하면 목록의 모든 항목이 기본적으로 선택됩니다.
styleClass	아니오	제목, 행 및 페이징 단추를 포함하여 전체 테이블을 렌더링하는 데 사용되는 CSS 스타일 클래스

표 73. bpe:column 속성

속성	필수 여부	설명
action	아니오	이 속성을 지정하면 링크가 열에 렌더링됩니다. 이 링크를 클릭하면 JavaServer Faces 조치 메소드 또는 Faces 탐색 대상이 트리거됩니다. JavaServer Faces 조치 메소드의 구조는 String method()와 같습니다.
converterID	아니오	특성 값을 변환하는 데 사용되는 Faces 변환기 ID입니다. 이 속성을 설정하지 않으면 모델이 이 특성에 제공하는 Faces 변환기 ID가 사용됩니다.
label	아니오	테이블 헤더 행의 셀 또는 열의 헤더의 레이블로 사용되는 리터럴 또는 값 바인딩 표현식입니다. 이 속성을 설정하지 않으면 모델이 이 특성에 제공하는 레이블이 사용됩니다.
name	예	이 열에 표시되는 특성의 이름

JSF 응용프로그램에 Details 컴포넌트 추가

타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시하려면 Business Process Choreographer Explorer의 Details 컴포넌트를 사용하십시오.

프로시저

1. JSP(JavaServer Pages) 파일에 Details 컴포넌트를 추가하십시오.

<h:form> 태그에 bpe:details 태그를 추가하십시오. bpe:details 태그는 **model** 속성을 포함해야 합니다. bpe:property 태그를 사용하여 Details 컴포넌트에 특성을 추가할 수 있습니다.

다음 예는 타스크 인스턴스에 대한 일부 특성을 표시하기 위해 Details 컴포넌트를 추가하는 방법을 보여줍니다.

```
<h:form>

   <bpe:details model="#{TaskInstanceDetails}">
      <bpe:property name="displayName" />
      <bpe:property name="owner" />
      <bpe:property name="kind" />
      <bpe:property name="state" />
      <bpe:property name="escalated" />
      <bpe:property name="suspended" />
      <bpe:property name="originator" />
      <bpe:property name="activationTime" />
      <bpe:property name="expirationTime" />
   </bpe:details>

</h:form>
```

model 속성은 관리 Bean, TaskInstanceDetails를 나타냅니다. 이 Bean은 Java 오브젝트의 특성을 제공합니다.

2. bpe:details 태그에 참조된 관리 Bean을 구성하십시오.

   Details 컴포넌트의 경우, 이 관리 Bean은 com.ibm.bpe.jsf.handler.BPCDetailsHandler 클래스의 인스턴스여야 합니다. 이 핸들러 클래스는 Java 오브젝트를 랩핑하여 해당 공용 특성을 세부사항 컴포넌트에 알려줍니다.

   다음 예는 TaskInstanceDetails 관리 Bean을 구성 파일에 추가하는 방법을 보여줍니다.

```
<managed-bean>
   <managed-bean-name>TaskInstanceDetails</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.jsf.handler.BPCDetailsHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>
```

   이 예는 TaskInstanceDetails에 구성할 수 있는 type 특성이 있음을 보여줍니다. 이 유형 특성의 값은 Bean 클래스 (com.ibm.task.clientmodel.bean.TaskInstanceBean)를 지정하며 표시된 세부사항 행에 해당 특성이 표시됩니다. Bean 클래스는 JavaBeans 클래스일 수 있습니다. Bean이 기본 변환기 및 특성 레이블을 제공하면 변환기 및 레이블을 사용하여 List 컴포넌트와 동일한 방식으로 렌더링합니다.

결과

JSF 응용프로그램은 이제 지정된 오브젝트의 세부사항(예: 타스크 인스턴스의 세부사항)을 표시하는 JavaServer 페이지를 포함합니다.

Details 컴포넌트: 태그 정의

Business Process Choreographer Explorer의 Details 컴포넌트는 타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시합니다.

Details 컴포넌트는 JSF 컴포넌트 태그인 `bpe:details` 및 `bpe:property`로 구성됩니다. `bpe:property` 태그는 `bpe:details` 태그의 하위 요소입니다.

컴포넌트 클래스

com.ibm.bpe.jsf.component.DetailsComponent

예제 구문

```
<bpe:details model=#{MyActivityDetails}>
    <bpe:property name=name/>
    <bpe:property name=owner/>
    <bpe:property name=activated/>
</bpe:details>
```

```
<bpe:details model=#{MyActivityDetails} style=style styleClass=cssStyle>
             style=style
             styleClass=cssStyle
</bpe:details>
```

태그 속성

표시된 속성의 서브세트 및 이 속성이 표시되는 순서를 모두 지정하려면 `bpe:property` 태그를 사용하십시오. details 태그에 속성 태그가 들어 있지 않은 경우, 이 details 태그는 모델 오브젝트의 모든 사용 가능한 속성을 표현합니다.

표 74. bpe:details 속성

속성	필수 여부	설명
columnClasses	아니오	열을 렌더링하는 데 사용되며 쉼표로 구분되는 계단식 스타일시트(CSS) 스타일 클래스 목록
id	아니오	컴포넌트의 JavaServer Faces ID
model	예	com.ibm.bpe.jsf.handler.BPCDetailsHandler 클래스의 관리 Bean에 대한 값 바인딩
rowClasses	아니오	행을 렌더링하는 데 사용되며 쉼표로 구분되는 CSS 스타일 클래스의 목록
styleClass	아니오	HTML 요소를 렌더링하는 데 사용되는 CSS 클래스

표 75. bpe:property 속성

속성	필수 여부	설명
converterID	아니오	JSP(JavaServer Faces) 구성 파일에 변환기를 등록하는 데 사용되는 ID
label	아니오	특성의 레이블입니다. 이 속성을 설정하지 않으면 클라이언트 모델 클래스가 기본 레이블을 제공합니다.
name	예	표시될 특성의 이름입니다. 이 이름은 해당 클라이언트 모델 클래스에 정의된 이름 지정된 특성에 해당되어야 합니다.

JSF 응용프로그램에 CommandBar 컴포넌트 추가

단추가 있는 표시줄을 표시하려면 Business Process Choreographer Explorer CommandBar 컴포넌트를 사용하십시오. 이 단추는 목록에서 선택한 오브젝트 또는 오브젝트의 자세히 보기에서 조작되는 명령을 나타냅니다.

이 태스크 정보

사용자가 사용자 인터페이스에서 단추를 누르면 해당 명령이 선택된 오브젝트에 대해 실행됩니다. JSF(JavaServer Faces) 응용프로그램에서 CommandBar 컴포넌트를 추가하고 확장할 수 있습니다.

프로시저

1. JSP(JavaServer Pages) 파일에 CommandBar 컴포넌트를 추가하십시오.

   `<h:form>` 태그에 `bpe:commandbar` 태그를 추가하십시오. `bpe:commandbar` 태그는 모델 속성을 포함해야 합니다.

   다음 예는 타스크 인스턴스 목록에 새로 고치기 및 청구 명령을 제공하는 CommandBar 컴포넌트를 추가하는 방법을 보여줍니다.

```
<h:form>

   <bpe:commandbar model="#{TaskInstanceList}">
      <bpe:command commandID="Refresh" >
                   action="#{TaskInstanceList.refreshList}"
                   label="Refresh"/>

      <bpe:command commandID="MyClaimCommand" >
                   label="Claim" >
                   commandClass="<customcode>"/>
   </bpe:commandbar>

</h:form>
```

   이 **model** 속성은 관리 Bean을 나타냅니다. 이 Bean은 ItemProvider 인터페이스를 구현해야 하며 선택된 Java 오브젝트를 제공합니다. CommandBar 컴포넌트는 일반적으로 동일한 JSP 파일에 있는 Details 컴포넌트 또는 List 컴포넌트와 함께

사용됩니다. 일반적으로 이 태그에 지정된 모델은 동일한 페이지의 Details 컴포넌트 또는 List 컴포넌트에 지정된 모델과 동일합니다. 예를 들어, List 컴포넌트의 경우, 명령은 목록에서 선택한 항목에 대해 수행됩니다.

이 예에서 **model** 속성은 TaskInstanceList 관리 Bean을 나타냅니다. 이 Bean은 타스크 인스턴스 목록에 선택한 오브젝트를 제공합니다. 해당 Bean은 ItemProvider 인터페이스를 구현해야 합니다. 이 인터페이스는 BPCListHandler 클래스 및 BPCDetailsHandler 클래스에 의해 구현됩니다.

2. 옵션: bpe:commandbar 태그에 참조된 관리 Bean을 구성하십시오.

   CommandBar **model** 속성이 이미 구성된 관리 Bean(예: 목록 또는 세부사항 핸들러)을 나타낼 경우 추가 구성은 필요하지 않습니다. 모델에 대해 BPCListHandler 클래스나 BPCDetailsHandler 클래스 중 어느 것도 사용하지 않는 경우, ItemProvider 인터페이스를 구현하는 클래스가 있는 다른 오브젝트를 참조해야 합니다.

3. 사용자 정의 명령을 구현하는 코드를 JSF 응용프로그램에 추가하십시오.

   다음 코드 스니펫은 Command 인터페이스를 구현하는 명령 클래스를 쓰는 방법을 보여줍니다. 이 명령 클래스(MyClaimCommand)는 JSP 파일에 있는 bpe:command 태그에 의해 참조됩니다.

```
public class MyClaimCommand implements Command {

  public String execute(List selectedObjects) throws ClientException {
     if( selectedObjects != null && selectedObjects.size() > 0 ) {
        try {
         // Determine HumanTaskManagerService from an HTMConnection bean.
         // Configure the bean in the faces-config.xml for easy access
         // in the JSF application.
         FacesContext ctx = FacesContext.getCurrentInstance();
         ValueBinding vb =
           ctx.getApplication().createValueBinding("{htmConnection}");
         HTMConnection htmConnection = (HTMConnection)

htmVB.getValue(ctx);
         HumanTaskManagerService htm =
            htmConnection.getHumanTaskManagerService();

         Iterator iter = selectedObjects.iterator() ;
         while( iter.hasNext() ) {
           try {
                TaskInstanceBean task = (TaskInstanceBean) iter.next() ;
                TKIID tiid = task.getID() ;

                htm.claim( tiid ) ;
                task.setState( new Integer(TaskInstanceBean

.STATE_CLAIMED ) ) ;

           }
           catch( Exception e ) {
```

```
                ;   // Error while iterating or claiming

        task instance.
                    // Ignore for better understanding of the sample.
             }
           }
         }
         catch( Exception e ) {
           ;  // Configuration or communication error.
              // Ignore for better understanding of the sample
         }
       }
       return null;
    }

   // Default implementations
   public boolean isMultiSelectEnabled() { return false; }
   public boolean[] isApplicable(List itemsOnList) {return null; }
   public void setContext(Object targetModel) {; // Not used here }
}
```

명령은 다음과 같은 방법으로 처리됩니다.

a. 사용자가 명령 표시줄에서 해당 단추를 누르면 명령이 호출됩니다. CommandBar 컴포넌트는 **model** 속성에서 지정된 항목 프로바이더에서 선택된 항목을 검색한 후 선택된 오브젝트 목록을 commandClass 인스턴스의 execute 메소드로 전달합니다.

b. 옵션: **commandClass** 속성은 명령 인터페이스를 구현하는 사용자 정의 명령 구현을 나타냅니다. 즉, 해당 명령은 public String execute(List selectedObjects) throws ClientException 메소드를 구현해야 합니다. 이 명령은 JSF 응용프로그램의 다음 탐색 규칙을 판별하는 데 사용되는 결과를 리턴합니다.

c. 옵션: 명령이 완료되면 CommandBar 컴포넌트가 **action** 속성을 평가합니다. **action** 속성은 `public String Method()` 서명이 있는 JSF 조치 메소드에 대한 메소드 바인딩 또는 static 문자열이 될 수 있습니다. 명령 클래스의 결과를 대체하거나 탐색 규칙에 대한 결과를 명시적으로 지정하려면 이 **action** 속성을 사용하십시오. 명령이 ErrorsInCommandException 예외 이외의 다른 예외를 생성하는 경우 이 **action** 속성은 처리되지 않습니다.

d. **commandClass** 속성에 명령 클래스가 지정되지 않은 경우에는 즉시 조치가 호출됩니다. 예를 들어, 예제의 새로 고치기 명령의 경우 JSF 값 표현식 `#{TaskInstanceList.refreshList}`가 명령 대신에 호출됩니다.

결과

이제 JSF 응용프로그램은 사용자 정의 명령 표시줄을 구현하는 JavaServer 페이지를 포함합니다.

명령 처리 방법

CommandBar 컴포넌트를 사용하여 응용프로그램에 조치 단추를 추가할 수 있습니다. 컴포넌트는 사용자 인터페이스에서 조치에 대한 단추를 작성하고 단추를 누를 때 작성되는 이벤트를 처리합니다.

이 단추는 BPCListHandler 클래스 또는 BPCDetailsHandler 클래스와 같은 com.ibm.bpe.jsf.handler.ItemProvider 인터페이스가 리턴하는 오브젝트에서 실행되는 함수들을 트리거합니다. CommandBar 컴포넌트는 `bpe:commandbar` 태그의 **model** 속성 값으로 정의된 항목 프로바이더를 사용합니다.

응용프로그램의 사용자 인터페이스의 명령 막대 섹션에서 단추가 눌러지면, 다음과 같이 CommandBar 컴포넌트가 연관된 이벤트를 처리합니다.

1. CommandBar 컴포넌트는 이벤트를 생성하는 단추에 지정된 com.ibm.bpc.clientcore.Command 인터페이스의 구현을 정의합니다.
2. CommandBar 컴포넌트와 연관된 모델이 com.ibm.bpe.jsf.handler.ErrorHandler 인터페이스를 구현하는 경우, 이전 이벤트에서 오류 메시지를 제거하기 위해 clearErrorMap 메소드가 호출됩니다.
3. ItemProvider 인터페이스의 getSelectedItems가 호출됩니다. 리턴된 항목 목록은 명령의 execute 메소드로 전달되며 이 명령이 호출되게 됩니다.
4. CommandBar 컴포넌트는 JSP(JavaServer Faces) 탐색 대상을 결정합니다. **action** 속성이 `bpe:commandbar` 태그에 지정되지 않은 경우, execute 메소드의 리턴값은 탐색 대상을 지정합니다. **action** 속성이 JSF 메소드 바인딩으로 설정되면 이 메소드에서 리턴된 문자열이 탐색 대상으로 해석됩니다. 또한 **action** 속성은 명확한 탐색 대상을 지정할 수도 있습니다.

CommandBar 컴포넌트: 태그 정의

Business Process Choreographer Explorer CommandBar 컴포넌트는 단추가 있는 표시줄을 표시합니다. 이 단추는 자세히 보기에 있는 오브젝트 또는 목록에서 선택된 오브젝트에 대해 조작됩니다.

CommandBar 컴포넌트는 JSF 컴포넌트 태그 `bpe:commandbar` 및 `bpe:command`로 구성됩니다. `bpe:command` 태그는 `bpe:commandbar` 태그의 하위 요소입니다.

컴포넌트 클래스

com.ibm.bpe.jsf.component.CommandBarComponent

예제 구문

```
<bpe:commandbar model="#{TaskInstanceList}">

    <bpe:command
       commandID="Work on"
       label="Work on..."
```

```
        commandClass="com.ibm.bpc.explorer.command.WorkOnTaskCommand"
        context="#{TaskInstanceDetailsBean}"/>

    <bpe:command
        commandID="Cancel"
        label="Cancel"
        commandClass="com.ibm.task.clientmodel.command

        .CancelClaimTaskCommand"
        context="#{TaskInstanceList}"/>

</bpe:commandbar>
```

태그 속성

표 76. bpe:commandbar 속성

속성	필수 여부	설명
buttonStyleClass	아니오	명령 표시줄의 단추를 렌더링하는 데 사용되는 계단식 스타일시트(CSS) 스타일 클래스
id	아니오	컴포넌트의 JavaServer Faces ID
model	예	ItemProvider 인터페이스를 구현하는 관리 Bean에 대한 값 바인딩 표현식입니다. 이 관리 Bean은 보통 같은 JSP(JavaServer Pages) 파일에 있는 List 컴포넌트 또는 Details 컴포넌트가 CommandBar 컴포넌트로서 사용하는 com.ibm.bpe.jsf.handler.BPCListHandler 클래스 또는 com.ibm.bpe.jsf.handler.BPCDetailsHandler 클래스입니다.
styleClass	아니오	명령 표시줄을 렌더링하는 데 사용되는 CSS 스타일 클래스

표 77. bpe:command 속성

속성	필수 여부	설명
action	아니오	명령 단추로 트리거될 JavaServer Faces 조치 메소드 또는 Faces 탐색 대상입니다. 조치에서 리턴되는 탐색 대상은 다른 모든 탐색 규칙을 겹쳐씁니다. 명령에서 ErrorsInCommandException 예외가 발생하거나 예외가 발생하지 않을 때 조치가 호출됩니다.
commandClass	아니오	명령 클래스의 이름입니다. 클래스의 인스턴스는 CommandBar 컴포넌트로 작성되며 명령 단추가 선택되면 실행됩니다.
commandID	예	명령 ID
context	아니오	**commandClass** 속성을 사용하여 지정되는 명령에 컨텍스트를 제공하는 오브젝트입니다. 컨텍스트 오브젝트는 명령 표시줄에 최초로 액세스할 때 검색됩니다.
immediate	아니오	명령이 트리거될 때 지정됩니다. 이 속성의 값이 `true`이면 페이지의 입력이 처리될 때 명령이 트리거됩니다. 기본값은 `false`입니다.
label	예	명령 표시줄에 표현되는 단추의 레이블

표 77. *bpe:command 속성 (계속)*

속성	필수 여부	설명
rendered	아니오	버튼이 렌더링되는지 여부를 판별합니다. 이 속성의 가능한 값은 Boolean 값 또는 값 표현식입니다.
styleClass	아니오	버튼을 렌더링하는 데 사용되는 CSS 스타일 클래스입니다. 이 스타일은 명령 표시줄에 대해 정의된 단추 스타일을 대체합니다.

JSF 응용프로그램에 Message 컴포넌트 추가

JSF(JavaServer Faces) 응용프로그램에서 데이터 오브젝트 및 기본 유형을 표현하려면 Business Process Choreographer Explorer Message 컴포넌트를 사용하십시오.

이 태스크 정보

메시지 유형이 기본 유형일 경우 레이블 및 입력 필드가 표현됩니다. 메시지 유형이 데이터 오브젝트일 경우 컴포넌트는 이 오브젝트를 트래버스하여 오브젝트 내에 요소를 표현합니다.

프로시저

1. JSP(JavaServer Pages) 파일에 Message 컴포넌트를 추가하십시오.

   `<h:form>` 태그에 `bpe:form` 태그를 추가하십시오. `bpe:form` 태그는 model 속성을 포함해야 합니다.

   다음 예는 Message 컴포넌트를 추가하는 방법을 보여줍니다.

   ```
   <h:form>

      <h:outputText value="Input Message" />
      <bpe:form model="#{MyHandler.inputMessage}" readOnly="true" />

      <h:outputText value="Output Message" />
      <bpe:form model="#{MyHandler.outputMessage}" />

   </h:form>
   ```

   Message 컴포넌트의 **model** 속성은 com.ibm.bpc.clientcore.MessageWrapper 오브젝트를 나타냅니다. 이 랩퍼 오브젝트는 서비스 데이터 오브젝트(SDO) 오브젝트 또는 Java 기본 유형(예: `int` 또는 `boolean`)을 랩핑합니다. 이 예에서 메시지는 MyHandler 관리 Bean의 특성에 의해 제공됩니다.

2. `bpe:form` 태그에 참조된 관리 Bean을 구성하십시오.

   다음 예는 MyHandler 관리 Bean을 구성 파일에 추가하는 방법을 보여줍니다.

   ```
   <managed-bean>
   <managed-bean-name>MyHandler</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.sample.jsf.MyHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   ```

```
    <managed-property>
       <property-name>type</property-name>
       <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
    </managed-property>

</managed-bean>
```

3. JSF 응용프로그램에 사용자 정의 코드를 추가하십시오.

   다음 예는 입력 및 출력 메시지를 구현하는 방법을 보여줍니다.

```
public class MyHandler implements ItemListener {

  private TaskInstanceBean taskBean;
  private MessageWrapper inputMessage, outputMessage

  /* Listener method, e.g. when a task instance was selected

  in a list handler.
   * Ensure that the handler is registered in the faces-config.xml

   or manually.
   */
  public void itemChanged(Object item) {
    if( item instanceof TaskInstanceBean ) {
        taskBean = (TaskInstanceBean) item ;
    }
  }

  /* Get the input message wrapper
   */
  public MessageWrapper getInputMessage() {
    try{
        inputMessage = taskBean.getInputMessageWrapper() ;
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return inputMessage;
  }

  /* Get the output message wrapper
   */
  public MessageWrapper getOutputMessage() {
    // Retrieve the message from the bean. If there is

    no message, create
    // one if the task has been claimed by the user. Ensure that only
    // potential owners or owners can manipulate the output message.
    try{
        outputMessage = taskBean.getOutputMessageWrapper();
        if( outputMessage == null
         && taskBean.getState() == TaskInstanceBean.STATE_CLAIMED ) {
            HumanTaskManagerService htm =

     getHumanTaskManagerService();
            outputMessage = new MessageWrapperImpl();
            outputMessage.setMessage(
```

```
                htm.createOutputMessage( taskBean.getID() ).getObject()
            );
        }
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return outputMessage
  }
}
```

MyHandler 관리 Bean은 com.ibm.jsf.handler.ItemListener 인터페이스를 구현하여 이 인터페이스가 목록 핸들러에 항목 리스너로서 등록될 수 있도록 합니다. 사용자가 목록에서 항목을 누르면 이 선택된 항목에 대한 통지가 MyHandler Bean의 itemChanged(Object item) 메소드로 전달됩니다. 핸들러는 항목 유형을 확인한 후 연관된 TaskInstanceBean 오브젝트에 대한 참조를 저장합니다. 이 인터페이스를 사용하려면 `faces-config.xml` 파일에 있는 해당 목록 핸들러의 itemListener 목록에 항목을 추가하십시오.

MyHandler Bean은 getInputMessage 및 getOutputMessage 메소드를 제공합니다. 이들 메소드는 둘 다 MessageWrapper 오브젝트를 리턴합니다. 이들 메소드는 참조된 타스크 인스턴스 Bean으로 호출을 위임합니다. 타스크 인스턴스 Bean이 널을 리턴할 경우(예: 메시지가 설정되지 않았기 때문에) 핸들러는 비어 있는 새 메시지를 작성하여 저장합니다. Message 컴포넌트는 MyHandler Bean이 제공하는 메시지를 표시합니다.

결과

이제 JSF 응용프로그램은 데이터 오브젝트 및 기본 유형을 표현할 수 있는 JavaServer 페이지를 포함할 수 있습니다.

Message 컴포넌트: 태그 정의

Business Process Choreographer Explorer의 Message 컴포넌트는 JSF(JavaServer Faces) 응용프로그램에 `commonj.sdo.DataObject` 오브젝트 및 기본 유형(예: 정수 및 문자열)을 표현합니다.

Message 컴포넌트는 JSF 컴포넌트 태그 `bpe:form`으로 구성됩니다.

컴포넌트 클래스

com.ibm.bpe.jsf.component.MessageComponent

예제 구문

```
<bpe:form model=
"#{TaskInstanceDetailsBean.inputMessageWrapper}"
          simplification="true" readOnly="true"
          styleClass4table="messageData"
          styleClass4output="messageDataOutput">
</bpe:form>
```

태그 속성

표 78. bpe:form 속성

속성	필수	설명
id	아니오	컴포넌트의 JavaServer Faces ID
model	예	`commonj.sdo.DataObject` 오브젝트 또는 `com.ibm.bpc.clientcore.MessageWrapper` 오브젝트를 참조하는 값 바인딩 표현식
readOnly	아니오	이 속성을 `true`로 설정하면 읽기 전용 양식이 표현됩니다. 기본적으로 이 속성은 `false`로 설정됩니다.
simplification	아니오	이 속성을 `true`로 설정하면 단순 유형을 포함하고 0 이상의 카디널리티가 있는 특성이 표시됩니다. 기본적으로 이 속성은 `true`로 설정됩니다.
style4validinput	아니오	유효한 입력을 렌더링하기 위한 캐스케이딩 스타일 시트(CSS) 스타일
style4invalidinput	아니오	유효하지 않은 입력을 렌더링하기 위한 CSS 스타일
styleClass4invalidInput	아니오	유효하지 않은 입력을 렌더링하기 위한 CSS 스타일 클래스 이름
styleClass4output	아니오	출력 요소를 렌더링하기 위한 CSS 스타일 클래스 이름
styleClass4table	아니오	메시지 컴포넌트가 표현하는 테이블을 렌더링하기 위한 CSS 테이블 스타일의 클래스 이름
styleClass4validInput	아니오	유효한 입력을 렌더링하기 위한 CSS 스타일 클래스 이름

타스크 및 프로세스 메시지에 대한 JSP 페이지 개발

Business Process Choreographer Explorer 인터페이스는 비즈니스 데이터 표시 및 입력에 기본 입력 및 출력 양식을 제공합니다. JSP 페이지를 사용하여 사용자 정의한 입력 및 출력 양식을 제공할 수 있습니다.

이 태스크 정보

웹 클라이언트에 사용자 정의 JSP(JavaServer Pages) 페이지를 포함하려면 WebSphere Integration Developer에서 휴먼 타스크를 모델화할 때 이 페이지를 지정해야 합니다. 예를 들어, 특정 타스크와 입출력 메시지 및 특정 사용자 역할 또는 모든 사용자 역할

에 JSP 페이지를 제공할 수 있습니다. 런타임 시 사용자 인터페이스에 사용자 정의 JSP 페이지가 포함되어 출력 데이터를 표시하고 입력 데이터를 수집합니다.

사용자 정의된 양식은 자체적으로 포함된 웹 페이지가 아니며 Business Process Choreographer Explorer가 HTML 양식으로 임베디드한 HTML 단편입니다(예: 메시지의 모든 입력 필드 및 레이블의 단편).

사용자 정의된 양식이 있는 페이지에서 단추를 클릭하면 입력이 제출되고 Business Process Choreographer Explorer에서 유효성이 검증됩니다. 유효성 검증은 제공된 특성의 유형 및 브라우저에서 사용되는 로케일을 기반으로 합니다. 입력의 유효성을 검증할 수 없는 경우, 같은 페이지가 다시 나타나며 messageValidationErrors 요청 속성에 유효성 검증 오류에 대한 정보가 제공됩니다. 정보는 발생한 유효성 검증 예외에 유효하지 않은 특성의 XML 경로 표현식(XPath)을 맵핑하는 맵으로 제공됩니다.

사용자 정의된 양식을 Business Process Choreographer Explorer에 추가하려면 WebSphere Integration Developer를 사용하여 다음 단계를 완료하십시오.

프로시저

1. 사용자 정의된 양식을 작성하십시오.

   웹 인터페이스에서 사용되는 입출력 양식의 사용자 정의 JSP 페이지는 메시지 데이터에 액세스할 필요가 있습니다. JSP의 Java 스니펫 또는 JSP 실행 언어를 사용하여 메시지 데이터에 액세스하십시오. 양식의 데이터는 요청 컨텍스트를 통해 사용 가능합니다.
2. 타스크에 JSP 페이지를 지정하십시오.

   휴먼 타스크 편집기에서 휴먼 타스크를 여십시오. 클라이언트 설정에서 사용자 정의 JSP 페이지의 위치 및 사용자 정의된 양식을 적용할 역할(예: 관리자)을 지정하십시오. Business Process Choreographer Explorer의 클라이언트 설정이 타스크 템플리트에 저장됩니다. 런타임 시 타스크 템플리트로 이들 설정을 검색합니다.
3. 웹 아카이브(WAR 파일)에 사용자 정의 JSP 페이지를 패키지하십시오.

   타스크를 포함하는 모듈과 함께 엔터프라이즈 아카이브에 WAR 파일을 포함시키거나 WAR 파일을 별도로 배치할 수 있습니다. JSP가 별도로 전개된 경우에는 Business Process Choreographer Explorer 또는 사용자 정의 클라이언트가 전개되는 서버에서 JSP가 사용 가능해야 합니다.

   프로세스 및 타스크 메시지에 사용자 정의 JSP를 사용 중인 경우 JSP를 전개하는 데 사용된 웹 모듈을 사용자 정의 JSF 클라이언트가 맵핑된 동일한 서버로 맵핑해야 합니다.

결과

런타임 시 사용자 정의된 양식이 Business Process Choreographer Explorer에 표현됩니다.

사용자 정의 JSP 단편

사용자 정의 JSP(JavaServer Pages) 단편은 HTML 양식 태그에 임베디드됩니다. 런타임 시, Business Process Choreographer Explorer는 표현된 페이지에 이러한 단편을 포함합니다.

입력 메시지의 사용자 정의 JSP 단편은 출력 메시지의 JSP 단편 전에 임베디드됩니다.

```
<html....>
   ...
   <form...>
     Input JSP (display task input message)

     Output JSP (display task output message)

   </form>
   ...
</html>
```

사용자 정의 JSP 단편이 HTML 양식 태그에 임베디드되기 때문에 입력 요소를 추가할 수 있습니다. 입력 요소의 이름은 데이터 요소의 XPath(XML Path Language) 표현식과 일치해야 합니다. 입력 요소 이름의 접두부를 제공된 접두부 값으로 지정해야 합니다.

```
<input id="address"
       type="text"
       name="${prefix}/selectPromotionalGiftResponse/address"
       value="${messageMap['/selectPromotionalGiftResponse/address"]}
       size="60"
       align="left" />
```

접두부 값이 request 속성으로 제공됩니다. 이 속성으로 인해 입력 이름이 엔클로징 양식에서 고유합니다. 접두부는 Business Process Choreographer Explorer에서 생성되며 변경할 수 없습니다.

```
String prefix = (String)request.getAttribute("prefix");
```

메시지를 제공된 컨텍스트에서 편집할 수 있는 경우에만 접두부 요소가 설정됩니다. 휴먼 타스크의 상태에 따라 출력 데이터를 여러 가지 방법으로 표시할 수 있습니다. 예를 들어, 타스크가 청구됨 상태인 경우, 출력 데이터를 수정할 수 있습니다. 그러나 타스크가 완료됨 상태인 경우 단지 데이터를 볼 수만 있습니다. JSP 단편에서 접두부 요소의 존재 여부를 테스트하고 그에 따라 메시지를 표현할 수 있습니다. 다음 JSTL문은 접두부 요소가 설정되었는지 여부를 테스트할 수 있는 방법을 보여줍니다.

```
...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<c:choose>
  <c:when test="${not empty prefix}">
    <!--Read/write mode-->
  </c:when>
  <c:otherwise>
    <!--Read-only mode-->
  </c:otherwise>
</c:choose>
```

휴먼 타스크 기능을 사용자 정의하는 플러그인 작성

Business Process Choreographer는 휴먼 타스크 처리 도중 발생하는 이벤트에 대한 이벤트 처리 인프라를 제공합니다. 필요에 맞게 기능을 조정할 수 있도록 플러그인 포인트도 제공합니다. SPI(Service Provider Interface)를 사용하여 이벤트 처리 및 사용자 조회 결과의 게시를 처리하는 사용자 정의된 플러그인을 작성할 수 있습니다.

이 태스크 정보

휴먼 타스크 API 이벤트 및 에스컬레이션 공고 이벤트를 위한 플러그인을 작성할 수 있습니다. 개인 분석에서 리턴된 결과를 처리하는 플러그인을 작성할 수도 있습니다. 예를 들어, 절정 기간에 워크로드 밸런싱을 돕기 위해 결과 목록에 사용자를 추가하고자 할 수도 있습니다.

플러그인을 사용하려면 이를 설치하고 등록해야 합니다. 사용자 조회 결과 게시를 처리하는 플러그인을 TaskContainer 응용프로그램에 등록할 수 있습니다. 그러면 플러그인이 모든 타스크에 사용 가능합니다.

Business Process Choreographer에 대한 API 이벤트 핸들러 작성

API 이벤트는 API 메소드가 휴먼 타스크를 조작하는 경우에 발생합니다. API 이벤트 핸들러 플러그인 SPI(Service Provider Interface)를 사용하여 API에서 전송한 타스크 이벤트 또는 동일한 API 이벤트를 갖는 내부 이벤트를 처리할 플러그인을 작성하십시오.

이 태스크 정보

다음 단계를 완료하여 API 이벤트 핸들러를 작성하십시오.

프로시저

1. APIEventHandlerPlugin5 인터페이스를 구현하거나 APIEventHandler 구현 클래스를 확장하십시오. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.

- APIEventHandlerPlugin5 인터페이스를 사용하는 경우, APIEventHandlerPlugin5 인터페이스 및 APIEventHandlerPlugin 인터페이스의 모든 메소드를 구현해야 합니다.
- APIEventHandler 구현 클래스를 확장할 경우, 필요한 메소드를 대체하십시오.

이 클래스는 Java EE(Java Platform, Enterprise Edition) 엔터프라이즈 응용프로그램의 컨텍스트에서 실행됩니다. 이 클래스 및 클래스의 헬퍼 클래스는 반드시 EJB 스펙을 준수해야 합니다.

주: 해당 클래스에서 HumanTaskManagerService 인터페이스를 호출하려는 경우, 이벤트를 생성한 타스크를 갱신하는 메소드를 호출하지 마십시오. 이 조치로 데이터베이스에서 타스크 데이터가 일치하지 않을 수 있습니다.

2. 플러그인 클래스 및 헬퍼 클래스를 JAR 파일에 어셈블하십시오.

   다음 방법 중 하나를 수행하면 JAR 파일을 사용 가능하도록 설정할 수 있습니다.
   - 응용프로그램 EAR 파일의 유틸리티 JAR 파일.
   - 응용프로그램 EAR 파일과 함께 설치되는 공유 라이브러리.
   - TaskContainer 응용프로그램과 함께 설치되는 공유 라이브러리. 이와 같은 경우, 플러그인이 모든 타스크에 사용 가능합니다.
3. JAR 파일의 `META-INF/services/` 디렉토리에 플러그인에 대한 서비스 프로바이더 구성 파일을 작성하십시오.

   구성 파일은 플러그인을 식별하고 로드하기 위한 메커니즘을 제공합니다. 이 파일은 Java EE 서비스 프로바이더 인터페이스 스펙을 충족합니다.

   a. `com.ibm.task.spi.`*`plug-in_name`*`APIEventHandlerPlugin`이라는 이름으로 파일을 작성하십시오. 여기서, *plug-in_name*은 플러그인의 이름입니다.

      예를 들어, 플러그인의 이름이 `Customer`이고 com.ibm.task.spi.APIEventHandlerPlugin5 인터페이스를 구현하는 경우, 구성 파일의 이름은 `com.ibm.task.spi.`**`Customer`**`APIEventHandlerPlugin`입니다.

   b. 주석 행(숫자 부호(#)로 시작하는 행)이나 빈 행이 아닌 이 파일의 첫 번째 행에서는 1단계에서 작성된 플러그인 클래스의 완전한 이름을 지정하십시오.

      예를 들어, 플러그인 클래스 이름이 `MyAPIEventHandler`이고 해당 클래스가 `com.customer.plugins` 패키지에 있는 경우, 구성 파일의 첫 번째 행에는 `com.customer.plugins.MyAPIEventHandler` 항목이 있어야 합니다.

결과

플러그인 로딩에 사용할 수 있는 서비스 프로바이더 구성 파일 및 API 이벤트를 처리하는 플러그인이 포함된 설치 가능 JAR 파일이 생성됩니다.

참고: API 이벤트 핸들러 및 공고 이벤트 핸들러를 모두 등록하는 경우 eventHandlerName 특성만을 사용할 수 있습니다. API 이벤트 핸들러와 공고 이벤트 핸들러를 모두 사용하려는 경우 SPI 구현에 대한 이벤트 핸들러 이름과 동일한 이름(예: `Customer`)이 플러그인 구현에 있어야 합니다.

단일 클래스나 두 개의 개별 클래스를 사용하여 두 가지 플러그인을 모두 구현할 수 있습니다. 두 경우 모두 JAR 파일의 `META-INF/services/` 디렉토리에 두 파일을 작성해야 합니다(예: com.ibm.task.spi.CustomerNotificationEventHandlerPlugin 및 com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

플러그인 구현 및 헬퍼 클래스를 단일 JAR 파일에 패키지하십시오.

구현 적용을 변경하려면 공유 라이브러리의 JAR 파일을 바꾸고, 연관된 EAR 파일을 다시 전개한 후 서버를 다시 시작하십시오.

다음에 수행할 작업

이제 플러그인을 설치하고 등록하여 런타임 시 휴먼 타스크 컨테이너에서 사용할 수 있습니다. 타스크 인스턴스, 타스크 템플리트 또는 응용프로그램 컴포넌트에 API 이벤트 핸들러를 등록할 수 있습니다.

API 이벤트 핸들러

API 이벤트는 휴먼 타스크를 수정하거나 상태를 변경할 때 발생합니다. 해당 API 이벤트를 처리하기 위해, 타스크 수정 직전(pre-event 메소드)과 API 호출 리턴 직전(post-event 메소드) 이벤트 핸들러가 호출됩니다.

pre-event 메소드에서 ApplicationVetoException 예외가 생성된 경우 API 조치가 수행되지 않으며 예외가 API 호출자에게 리턴되고 이벤트와 연관된 트랜잭션이 롤백됩니다. pre-event 메소드를 내부 이벤트에서 트리거하여 ApplicationVetoException 예외가 생성된 경우, 자동 청구와 같은 내부 이벤트가 수행되지 않지만 클라이언트 응용프로그램에 예외가 리턴되지 않습니다. 이 경우 정보 메시지가 SystemOut.log 파일에 기록됩니다. API 메소드가 처리 도중 예외를 생성하면, 예외가 발견되고 post-event 메소드에 전달됩니다. 예외는 post-event 메소드 리턴 후 호출자에게 다시 전달됩니다.

다음 규칙이 pre-event 메소드에 적용됩니다.

- pre-event 메소드는 연관된 API 메소드 또는 내부 이벤트 매개변수를 수신합니다.
- pre-event 메소드는 처리가 계속되지 않도록 ApplicationVetoException 예외를 생성할 수 있습니다.

다음 규칙이 post-event 메소드에 적용됩니다.

- post-event 메소드는 API 호출에 제공된 매개변수 및 리턴값을 수신합니다. API 메소드 구현에서 예외가 발생하면 post-event 메소드도 예외를 수신합니다.
- post-event 메소드는 리턴값을 수정할 수 없습니다.
- post-event 메소드는 예외를 생성할 수 없습니다. 런타임 예외는 로그되지만 계속 처리되지는 않습니다.

API 이벤트 핸들러를 구현하려면, APIEventHandlerPlugin 인터페이스를 확장하는 APIEventHandlerPlugin3 인터페이스를 구현하거나 기본 com.ibm.task.spi.APIEventHandler SPI 구현 클래스를 확장할 수 있습니다. 이벤트 핸들러는 기본 구현 클래스에서 상속된 경우 항상 SPI의 최신 버전을 구현합니다. Business Process Choreographer의 새 버전으로 업그레이드하면 새 SPI 메소드를 사용할 때 변경이 거의 필요하지 않습니다.

공고 이벤트 핸들러 및 API 이벤트 핸들러가 모두 있는 경우, 하나의 이벤트 핸들러 이름만을 등록할 수 있으므로 두 핸들러의 이름을 동일하게 해야 합니다.

Business Process Choreographer에 대한 공고 이벤트 핸들러 작성

휴먼 타스크가 에스컬레이트될 때 공고 이벤트가 생성됩니다. Business Process Choreographer는 에스컬레이션 작업 항목 작성, 전자 우편 전송 등 에스컬레이션을 처리하는 기능을 제공합니다. 공고 이벤트 핸들러를 작성하여 에스컬레이션을 처리하는 방법을 사용자 정의할 수 있습니다.

이 태스크 정보

공고 이벤트 핸들러를 구현하려면, NotificationEventHandlerPlugin 인터페이스를 구현하거나 기본 com.ibm.task.spi.NotificationEventHandler SPI(Service Provider Interface) 구현 클래스를 확장할 수 있습니다.

다음 단계를 완료하여 공고 이벤트 핸들러를 작성하십시오.

프로시저

1. NotificationEventHandlerPlugin 인터페이스를 구현하거나 NotificationEventHandler 구현 클래스를 확장하는 클래스를 작성하십시오. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.

   NotificationEventHandlerPlugin 인터페이스를 사용할 경우, 모든 인터페이스 메소드를 구현해야 합니다. SPI 구현 클래스를 확장할 경우, 필요한 메소드를 대체하십시오.

이 클래스는 Java EE(Java Platform, Enterprise Edition) 엔터프라이즈 응용프로그램의 컨텍스트에서 실행됩니다. 이 클래스 및 클래스의 헬퍼 클래스는 반드시 EJB 스펙을 준수해야 합니다.

플러그인은 EscalationUser 역할의 권한으로 호출됩니다. 이 역할은 휴먼 타스크 컨테이너 구성 시 정의됩니다.

주: 해당 클래스에서 HumanTaskManagerService 인터페이스를 호출하려는 경우, 이벤트를 생성한 타스크를 갱신하는 메소드를 호출하지 마십시오. 이 조치로 데이터베이스에서 타스크 데이터가 일치하지 않을 수 있습니다.

2. 플러그인 클래스 및 헬퍼 클래스를 JAR 파일에 어셈블하십시오.

   다음 방법 중 하나를 수행하면 JAR 파일을 사용 가능하도록 설정할 수 있습니다.

   - 응용프로그램 EAR 파일의 유틸리티 JAR 파일.
   - 응용프로그램 EAR 파일과 함께 설치되는 공유 라이브러리.
   - TaskContainer 응용프로그램과 함께 설치되는 공유 라이브러리. 이와 같은 경우, 플러그인이 모든 타스크에 사용 가능합니다.

3. 플러그인 클래스 및 헬퍼 클래스를 JAR 파일에 어셈블하십시오.

   헬퍼 클래스가 여러 Java EE 응용프로그램에서 사용되는 경우, 이러한 클래스를 독립 JAR 파일에 패키지하여 공유 라이브러리로 등록할 수 있습니다.

4. JAR 파일의 `META-INF/services/` 디렉토리에 플러그인에 대한 서비스 프로바이더 구성 파일을 작성하십시오.

   구성 파일은 플러그인을 식별하고 로드하기 위한 메커니즘을 제공합니다. 이 파일은 Java EE 서비스 프로바이더 인터페이스 스펙을 충족합니다.

   a. `com.ibm.task.spi.`*`plug-in_name`*`NotificationEventHandlerPlugin`이라는 이름으로 파일을 작성하십시오. 여기서, *plug-in_name*은 플러그인의 이름입니다.

      예를 들어, 플러그인의 이름이 `HelpDeskRequest`(이벤트 핸들러 이름)이고 com.ibm.task.spi.NotificationEventHandlerPlugin 인터페이스를 구현할 경우, 구성 파일의 이름은 `com.ibm.task.spi.`**`HelpDeskRequest`**`NotificationEventHandlerPlugin`입니다.

   b. 주석 행(숫자 부호(#)로 시작하는 행)이나 빈 행이 아닌 이 파일의 첫 번째 행에서는 1단계에서 작성된 플러그인 클래스의 완전한 이름을 지정하십시오.

예를 들어, 플러그인 클래스의 이름이 MyEventAPIHandler이고 com.customer.plugins 패키지에 위치할 경우, 구성 파일의 첫 번째 행에 com.customer.plugins.MyAPIEventHandler와 같은 항목이 포함되어야 합니다.

결과

플러그인 로딩에 사용할 수 있는 서비스 프로바이더 구성 파일 및 공고 이벤트를 처리하는 플러그인이 포함된 설치 가능 JAR 파일이 생성됩니다. 타스크 인스턴스, 타스크 템플리트 또는 응용프로그램 컴포넌트에 API 이벤트 핸들러를 등록할 수 있습니다.

참고: API 이벤트 핸들러 및 공고 이벤트 핸들러를 모두 등록하는 경우 eventHandlerName 특성만을 사용할 수 있습니다. API 이벤트 핸들러와 공고 이벤트 핸들러를 모두 사용하려는 경우 SPI 구현에 대한 이벤트 핸들러 이름과 동일한 이름(예: Customer)이 플러그인 구현에 있어야 합니다.

단일 클래스나 두 개의 개별 클래스를 사용하여 두 가지 플러그인을 모두 구현할 수 있습니다. 두 경우 모두 JAR 파일의 META-INF/services/ 디렉토리에 두 파일을 작성해야 합니다(예: com.ibm.task.spi.CustomerNotificationEventHandlerPlugin 및 com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

플러그인 구현 및 헬퍼 클래스를 단일 JAR 파일에 패키지하십시오.

구현 적용을 변경하려면 공유 라이브러리의 JAR 파일을 바꾸고, 연관된 EAR 파일을 다시 전개한 후 서버를 다시 시작하십시오.

다음에 수행할 작업

이제 플러그인을 설치하고 등록하여 런타임 시 휴먼 타스크 컨테이너에서 사용할 수 있습니다. 타스크 인스턴스, 타스크 템플리트 또는 응용프로그램 컴포넌트에 공고 이벤트 핸들러를 등록할 수 있습니다.

휴먼 타스크의 API 이벤트 핸들러 및 공고 이벤트 핸들러 플러그인 설치

API 이벤트 핸들러 또는 공고 이벤트 핸들러 플러그인을 사용하려면 휴먼 타스크 컨테이너에서 액세스할 수 있도록 해당 플러그인을 설치해야 합니다.

이 태스크 정보

플러그인을 설치하는 방법은 해당 플러그인을 단일 Java EE(Java Platform, Enterprise Edition) 응용프로그램에서 사용하는지 여러 응용프로그램에서 사용하는지에 따라 다릅니다.

플러그인을 설치하려면 다음 단계 중 하나를 완료하십시오.

프로시저

- 단일 Java EE 응용프로그램에서 사용할 플러그인을 설치하십시오.

  플러그인 JAR 파일을 응용프로그램 EAR 파일에 추가하십시오. WebSphere Integration Developer의 전개 설명자 편집기에서 플러그인의 JAR 파일을 기본 EJB(Enterprise JavaBeans) 모듈의 Java EE 응용프로그램에 대한 프로젝트 유틸리티 JAR 파일로 설치하십시오.
- 여러 Java EE 응용프로그램에서 사용할 플러그인을 설치하십시오.

  WebSphere Application Server 공유 라이브러리에 JAR 파일을 저장하고 라이브러리를 플러그인에 액세스하는 데 필요한 응용프로그램과 연관시키십시오. Network Deployment 환경에서 JAR 파일을 사용하려면 사용자의 응용프로그램이 전개된 서버 또는 클러스터 멤버를 호스트하는 각 노드에서 JAR 파일을 수동으로 분배하십시오. 응용프로그램이 전개된 서버 또는 클러스터인 응용프로그램의 전개 대상 범위 또는 셀 범위를 사용할 수 있습니다. 그런 다음, 선택된 전개 범위에 걸쳐 플러그인 클래스가 표시될 수 있다는 점에 유의하십시오.

다음에 수행할 작업

이제 플러그인을 등록할 수 있습니다.

타스크 템플리트, 타스크 모델 및 타스크에 API 이벤트 핸들러 및 공고 이벤트 핸들러 플러그인 등록

임시 타스크 작성, 기존 타스크 갱신, 임시 타스크 모델 작성 또는 타스크 템플리트 정의 시와 같은 다양한 경우에 타스크, 타스크 템플리트 및 타스크 모델에 API 이벤트 핸들러 및 공고 이벤트 핸들러의 플러그인을 등록할 수 있습니다.

이 태스크 정보

다음 레벨에서 타스크에 API 이벤트 핸들러 및 공고 이벤트 핸들러의 플러그인을 등록할 수 있습니다.

타스크 템플리트
: 템플리트를 사용하여 작성된 동일한 핸들러를 사용하는 모든 타스크

임시 타스크 모델
: 모델을 사용하여 작성된 동일한 핸들러를 사용하는 타스크

임시 타스크
: 지정된 핸들러를 사용하는 작성된 타스크

기존 타스크
: 지정된 핸들러를 사용하는 타스크

다음 방법 중 한 가지로 플러그인을 등록할 수 있습니다.

프로시저

- WebSphere Integration Developer에서 모델화된 타스크 템플리트의 경우 타스크 모델의 플러그인을 지정하십시오.
- 임시 타스크 또는 임시 타스크 모델의 경우, 타스크나 타스크 모델을 작성할 때 플러그인을 지정하십시오.

  TTask 클래스의 setEventHandlerName 메소드를 사용하여 이벤트 핸들러 이름을 등록하십시오.
- 런타임 시 타스크 인스턴스의 이벤트 핸들러를 변경하십시오.

  update(Task task) 메소드를 사용하여 런타임 시 타스크 인스턴스에 다른 이벤트 핸들러를 사용하십시오. 호출자에게는 이 특성을 갱신할 수 있는 타스크 관리자 권한이 있어야 합니다.

사후 처리 사용자 조회 결과에 플러그인 사용

Business Process Choreographer의 사용자 분석은 특정 역할(예: 타스크의 잠재적 소유자)에 지정된 사용자 목록을 리턴합니다. 사용자 분석에서 리턴되는 사용자 조회 결과를 변경하는 플러그인을 작성할 수 있습니다. 예를 들어, 워크로드 밸런싱을 개선하려면 조회 결과에서 이미 워크로드가 높은 사용자를 제거할 수 있습니다.

이 태스크 정보

사용자 지정 및 사용자 대체에서 리턴되는 결과를 수정하려면 플러그인 인터페이스를 구현하는 클래스를 작성하고 플러그인의 JAR 파일을 어셈블한 다음 이를 설치하고 활성화해야 합니다.

다음 단계를 완료하여 개인 조회 결과를 사후 처리할 플러그인을 작성하십시오.

프로시저

1. 사용자 조회 결과 사후 처리 플러그인을 구현하십시오.
   StaffQueryResultPostProcessorPlugin 인터페이스 또는 StaffQueryResultPostProcessorPlugin2 인터페이스를 구현하는 클래스를 작성하십시오.
2. 설치 가능한 Jar 파일을 작성하십시오.
   a. 플러그인 클래스 및 헬퍼 클래스를 JAR 파일로 어셈블하십시오.
   b. JAR 파일의 `META-INF/services/` 디렉토리에 플러그인에 대한 서비스 프로바이더 구성 파일을 작성하십시오. 구성 파일은 플러그인을 식별하고 로드하기 위한 메커니즘을 제공합니다. 이 파일은 Java EE 프로바이더 인터페이스 스펙을 충족해야 합니다.
      1) 문서 편집기에서 `com.ibm.task.spi.`*`plug-in_name`*`StaffQueryResultPostProcessorPlugin`(여기서 *plug-in_name*

은 플러그인의 이름)이라는 이름의 서비스 프로바이더 구성 파일을 작성하십시오. 구성 파일의 이름은 구현한 인터페이스 이름에 종속되지 않습니다. 예를 들어, 플러그인 이름이 `MyHandler`이고 com.ibm.task.spi.StaffQueryResultPostProcessorPlugin2 인터페이스를 구현하는 경우, 구성 파일의 이름은 `com.ibm.task.spi.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`입니다.

2) 파일에서 주석 행(숫자 부호(#)로 시작하는 행)도 아니고 공백 행도 아닌 첫 번째 행은 1단계에서 작성한 플러그인 클래스의 완전한 이름을 지정합니다. 예를 들어, 플러그인 클래스가 `StaffPostProcessor`이고 `com.customer.plugins` 패키지에 있는 경우, 구성 파일의 첫 번째 행에는 `com.customer.plugins.StaffPostProcessor`가 포함되어야 합니다.

사용자 조회 결과를 사후 처리하는 플러그인 및 플러그인을 로드하는 데 사용할 수 있는 서비스 프로바이더 구성 파일이 포함된 설치 가능한 JAR 파일이 있습니다.

3. Application Server의 공유 라이브러리에 JAR 파일을 설치하고 휴먼 타스크 관리자 응용프로그램과 연관시키십시오.
   a. Business Process Choreographer가 구성된 서버 또는 클러스터 범위의 플러그인에 대해 WebSphere Application Server 공유 라이브러리를 정의하십시오.
   b. 공유 라이브러리를 `TaskContainer` 응용프로그램과 연관시키십시오.
   c. 서버 또는 클러스터 멤버를 호스트하는 영향 받는 각각의 WebSphere Process Server 설치에 대해 플러그인 JAR 파일을 사용 가능하도록 설정하십시오.
4. 휴먼 타스크 관리자가 플러그인을 사용하도록 구성하십시오.
   a. 관리 콘솔에서 휴먼 타스크 관리자의 사용자 정의 특성 페이지로 이동하십시오.

      서버 → **클러스터** → **WebSphere Application Server 클러스터** → ***cluster_name*** 또는 **서버** → **서버 유형** → **WebSphere Application Server** → ***server_name*** 을 선택하고 **구성** 탭의 **비즈니스 통합** 섹션에서 **Business Process Choreographer**를 펼친 다음 **휴먼 타스크 관리자**를 클릭하십시오. **추가 특성** 아래에서 **사용자 정의 특성**을 선택하십시오.
   b. **Staff.PostProcessorPlugin**이라는 사용자 정의 특성 및 플러그인에 지정한 이름의 값(예: `MyHandler`)을 추가하십시오.

   이제 사용자 조회 결과 사후 처리에 플러그인을 사용할 수 있습니다.
5. 서버를 다시 시작하여 플러그인을 활성화하십시오. 사용자 지정 및 사용자 대체 둘 다가 실행된 이후 사후 처리 플러그인이 호출됩니다.

   주: 플러그인을 수정하는 경우에는 공유 라이브러리에서 JAR 파일을 바꾸고 서버를 다시 시작하십시오.

제 2 부 응용프로그램 전개

제 5 장 모듈 준비 및 설치 개요

모듈 설치(전개라고도 함)는 테스트 환경 또는 프로덕션 환경에서 모듈을 활성화합니다. 이 개요에서는 테스트 및 프로덕션 환경과 모듈 설치와 관련된 일부 단계를 간략히 설명합니다.

주: 프로덕션 환경에 응용프로그램을 설치하는 프로세스는 WebSphere Application Server Network Deployment Information Center의 『응용프로그램 개발 및 전개』에 설명된 프로세스와 유사합니다. 해당 주제에 대해 잘 모를 경우, 이를 먼저 검토하십시오.

프로덕션 환경에 모듈을 설치하기 전에 항상 테스트 환경에서 변경사항을 확인하십시오. 테스트 환경에 모듈을 설치하려면, WebSphere Integration Developer를 사용하십시오 (자세한 정보는 WebSphere Integration Developer Information Center 참조). 프로덕션 환경에 모듈을 설치하려면, WebSphere Process Server를 사용하십시오.

이 주제에서는 프로덕션 환경에 모듈을 준비하고 설치하는 데 필요한 개념과 타스크를 설명합니다. 기타 주제에서는 모듈이 사용하는 오브젝트를 보유하고 있으며 테스트 환경에서 프로덕션 환경으로 모듈을 이동하는 데 사용할 수 있는 파일에 대해 설명합니다. 이 파일과 파일에 포함된 내용을 이해하여 모듈을 올바로 설치하는 것이 중요합니다.

라이브러리 및 Jar 파일 개요

모듈은 종종 라이브러리에 있는 아티팩트를 사용합니다. 해당 아티팩트는 공유 자원을 저장하기 위해 사용된 WebSphere Integration Developer의 특수 프로젝트입니다. 전개 시 WebSphere Integration Developer 라이브러리가 유틸리티 JAR 파일로 변환되고 실행할 응용프로그램에 패키지 됩니다.

모듈을 개발하는 동안 기타 모듈에서 사용할 수 있는 특정 자원 또는 컴포넌트를 식별합니다. 이러한 아티팩트는 라이브러리를 사용해서 공유될 수 있습니다.

라이브러리 개념

라이브러리는 일반적으로 모듈 간에 공유되는 해당 자원과 같이 공유 자원의 조직, 버전 관리 및 개발에 사용되는 WebSphere Integration Developer의 특수 프로젝트입니다. 아티팩트 유형의 서브세트만 작성될 수 있고 라이브러리에 저장될 수 있습니다. 예는 다음과 같습니다.

- 확장자가 .wsdl인 인터페이스 또는 웹 서비스 설명자 파일
- 확장자가 .xsd인 비즈니스 오브젝트 XSD(XML Schema Definition) 파일

- 확장자가 .map인 비즈니스 오브젝트 맵 파일
- 확장자가 .rel 및 .rol인 관계 및 역할 정의 파일

개발 시 이러한 WebSphere Integration Developer 라이브러리가 실행할 응용프로그램의 유틸리티 JAR 파일로 전환됩니다.

모듈에 아티팩트가 필요한 경우 메모리에 로드되지 않았으면 서버는 아티팩트를 EAR 클래스 경로에서 찾아 로드합니다. 그림 78에서는 응용프로그램이 컴포넌트 및 연관 라이브러리를 포함하는 방법을 보여줍니다.

그림 78. 모듈, 컴포넌트 및 라이브러리 간의 관계

JAR, RAR 및 WAR 파일 개념

모듈의 컴포넌트를 포함할 수 있는 많은 파일이 있습니다. 이러한 파일은 J2EE(Java Platform, Enterprise Edition) 스펙에 자세히 설명되어 있습니다. Jar 파일에 대한 세부사항은 JAR 스펙에서 볼 수 있습니다.

WebSphere Process Server에서, JAR 파일은 모듈에서 사용되는 기타 서비스 컴포넌트에 대해 지원되는 모든 참조 및 인터페이스가 있는 모듈의 어셈블된 버전인 응용프로그램을 포함합니다. 응용프로그램을 완전히 설치하려면 이 JAR 파일, 다른 독립 JAR, WAR(Web services archive), RAR(resource archive), 스테이징 라이브러리(EJB - Enterprise Java Beans) JAR 파일 및 다른 아카이브가 필요합니다. 그런 다음 serviceDeploy 명령을 사용하여 설치 가능한 EAR 파일을 작성하십시오.

스테이징 모듈 이름 지정 규칙

라이브러리에서 스테이징 모듈 이름에 대한 요구사항이 있습니다. 이 이름은 특정 모듈에 고유합니다. 응용프로그램 전개에 필요한 다른 모듈의 이름을 지정하여 스테이징 모듈 이름과 충돌하지 않도록 하십시오. *myService*로 이름 지정된 모듈의 경우 스테이징 모듈 이름은 다음과 같습니다.

- *myService*App
- *myService*Web

주: *myService*EJB 및 *myService*EJBClient 스테이징 모듈은 더 이상 serviceDeploy에 의해 작성되지 않습니다. 그러나 파일 이름이 serviceDeploy 명령을 사용하여 삭제될 수 있기 때문에 해당 파일 이름을 사용하지 말아야 합니다.

라이브러리 사용 고려사항

라이브러리를 사용하면 각 호출 모듈이 특정 컴포넌트에 대한 자체 사본을 포함하기 때문에 비즈니스 오브젝트의 일관성 및 모듈간 처리의 일관성을 유지할 수 있습니다. 불일치 및 장애를 피하기 위해 호출 모듈에서 사용되는 컴포넌트 및 비즈니스 오브젝트의 변경은 모든 호출 모듈에서도 일치해야 합니다. 다음과 같이 호출 모듈을 갱신하십시오.

1. 모듈 및 라이브러리의 최신 사본을 프로덕션 서버로 복사하십시오.
2. serviceDeploy 명령을 사용하여 설치 가능한 EAR 파일을 다시 빌드하십시오.
3. 호출 모듈을 포함하고 이를 다시 설치하는 실행 중인 응용프로그램 중지
4. 호출 모듈을 포함하는 응용프로그램을 다시 시작하십시오.

EAR 파일 개요

EAR 파일은 서비스 응용프로그램을 프로덕션 서버에 전개하는 중요한 파일입니다.

EAR(Enterprise Archive) 파일은 응용프로그램 전개에 필요한 라이브러리, 엔터프라이즈 Bean 및 Jar 파일을 포함하는 압축 파일입니다.

WebSphere Integration Developer에서 응용프로그램 모듈을 내보낼 때 JAR 파일을 작성합니다. 이 Jar 파일 및 기타 아티팩트 라이브러리 또는 오브젝트를 설치 프로세스

에 대한 입력으로 사용합니다. serviceDeploy 명령은 응용프로그램을 구성하는 Java 코드 및 컴포넌트 설명을 포함하는 입력 파일에서 EAR 파일을 작성합니다.

서버에 전개 준비

모듈을 개발하고 테스트한 후에는 테스트 시스템에서 모듈을 내보내고 이를 전개의 프로덕션 환경으로 가져와야 합니다. 응용프로그램을 설치하려면 모듈을 내보낼 때 필요한 경로와 모듈에 필요한 라이브러리에 대해 알아야 합니다.

시작하기 전에

이 타스크를 시작하기 전에 테스트 서버에서 모듈을 개발 및 테스트하고 문제점 및 성능 문제를 해결해야 합니다.

중요사항: 전개 환경에서 이미 실행 중인 응용프로그램 또는 모듈을 바꾸지 않도록 하려면 이미 설치된 환경에서 모듈 또는 응용프로그램 이름이 고유해야 합니다.

이 태스크 정보

이 타스크에서는 모든 필수 응용프로그램을 사용할 수 있고 올바른 파일에 패키지되어 있으므로 프로덕션 서버로 가져올 수 있는지 확인합니다.

주: WebSphere Integration Developer에서 EAR(Enterprise Archive) 파일을 내보내 이 파일을 직접 WebSphere Process Server에 설치할 수도 있습니다.

중요사항: 컴포넌트의 서비스가 데이터베이스를 사용하는 경우 서버의 응용프로그램을 데이터베이스에 바로 연결되도록 설치하십시오.

프로시저

1. 전개할 모듈의 컴포넌트를 포함하는 폴더를 찾으십시오.

   컴포넌트 폴더 이름은 기본 모듈 *module*.module 파일이 포함되는 *module-name* 이름으로 지정해야 합니다.
2. 모듈에 포함되는 모든 컴포넌트가 모듈 폴더의 컴포넌트 서브폴더에 있는지 확인하십시오.

   쉽게 사용하기 위해 서브폴더 이름을 *module*/*component*와 비슷한 유형으로 지정하십시오.
3. 각 컴포넌트를 구성하는 모든 파일이 해당 컴포넌트 서브폴더에 있으며 *component-file-name*.component와 비슷한 유형의 이름으로 되어있는지 확인하십시오.

   컴포넌트 파일에는 모듈 내의 개별 컴포넌트에 대한 정의가 포함되어 있습니다.
4. 다른 모든 컴포넌트 및 아티팩트가 필요한 컴포넌트의 서브폴더에 있는지 확인하십시오.

이 단계에서는 컴포넌트가 필요로 하는 아티팩트에 대한 참조가 사용 가능한지 확인합니다. 컴포넌트의 이름은 serviceDeploy 명령이 스테이징 모듈에 사용하는 이름과 충돌하지 않아야 합니다. 스테이징 모듈에 대한 이름 지정 규칙을 참조하십시오.

5. 참조 파일 *module*.references가 562 페이지의 1단계의 모듈 폴더에 있는지 확인하십시오.

   참조 파일은 모듈에 있는 참조 및 인터페이스를 정의합니다.

6. 연결 파일, *module*.wires가 컴포넌트 폴더에 있는지 확인하십시오.

   연결 파일은 모듈에서 참조와 인터페이스의 연결을 완료합니다.

7. Manifest 파일, *module*.manifest가 컴포넌트 폴더에 있는지 확인하십시오.

   Manifest는 모듈을 구성하는 모듈 및 모든 컴포넌트를 표시합니다. serviceDeploy 명령에서 모듈에 필요한 기타 모듈을 찾을 수 있도록 클래스 경로 명령문도 포함합니다.

8. 프로덕션 서버에 모듈을 설치하는 데 사용되는 serviceDeploy 명령의 입력으로 모듈의 압축 파일 또는 Jar 파일을 작성하십시오.

전개 전 MyValue 모듈의 폴더 구조 예

다음 예는 MyValue, CustomerInfo 및 StockQuote 컴포넌트로 구성된 모듈 MyValueModule의 디렉토리 구조를 설명합니다.

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
   StockQuote.java
   StockQuoteAsynch.java
   StockQuoteCallback.java
   StockQuoteImpl.java
```

다음에 수행할 작업

프로덕션 서버에서 모듈 설치 중에서 설명한 대로 모듈을 프로덕션 시스템에 설치하십시오.

클러스터에 서비스 응용프로그램 설치 시 고려사항

클러스터에 서비스 응용프로그램을 설치할 경우 추가 요구사항이 제시됩니다. 클러스터에 서비스 응용프로그램을 설치할 때 다음 고려사항을 유념해야 합니다.

클러스터는 서버에서 요청 워크로드의 균형을 맞추는 데 유용한 경제적 규모를 제공하여 처리 환경에 많은 이점을 제공하고 응용프로그램의 클라이언트에 한가지 레벨의 가용성을 제공할 수 있습니다. 클러스터에 서비스를 포함하고 있는 응용프로그램을 설치하기 전에 다음을 고려하십시오.

- 응용프로그램 사용자가 클러스터링으로 제공되는 처리 능력과 가용성을 필요로 합니까?

  그런 경우, 클러스터링이 올바른 솔루션입니다. 클러스터링은 응용프로그램의 가용성과 용량을 증가시켜 줍니다.

- 클러스터가 서비스 응용프로그램에 대해 올바로 준비되었습니까?

  서비스를 포함하는 첫 번째 응용프로그램을 설치하여 시작하기 전에 클러스터를 올바로 구성해야 합니다. 클러스터를 올바로 구성하지 못하면 응용프로그램이 요청을 올바로 처리하지 못하게 합니다.

- 클러스터가 백업되었습니까?

  백업 클러스터에도 응용프로그램을 설치해야 합니다.

제 6 장 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치

비즈니스 프로세스나 휴먼 타스크 또는 둘 모두를 포함하는 SCA(Service Component Architecture) 모듈을 전개 대상에 분배할 수 있습니다. 전개 대상은 서버 또는 클러스터입니다.

시작하기 전에

응용프로그램을 설치하려는 각 Application Server 또는 클러스터에 대해 비즈니스 플로우 관리자, 휴먼 타스크 관리자가 설치 및 구성되어 있는지 확인하십시오.

이 태스크 정보

관리 콘솔을 사용하거나 명령행을 사용하거나 관리 스크립트를 실행하여 비즈니스 프로세스 및 타스크 응용프로그램을 설치할 수 있습니다.

결과

비즈니스 프로세스 또는 휴먼 타스크 응용프로그램이 설치되면 모든 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트가 시작 상태에 놓입니다. 이 템플리트에서 프로세스 인스턴스 및 타스크 인스턴스를 작성할 수 있습니다.

다음에 수행할 작업

프로세스 인스턴스 또는 타스크 인스턴스를 작성하기 전에 응용프로그램을 시작해야 합니다.

비즈니스 프로세스 및 휴먼 타스크 응용프로그램을 Network Deployment 환경에 설치하는 방법

프로세스 템플리트 또는 휴먼 타스크 템플리트를 Network Deployment 환경에 설치하면 응용프로그램 설치 시 다음 조치가 자동으로 수행됩니다.

응용프로그램은 스테이지에서 설치됩니다. 다음 스테이지를 시작하려면 각 스테이지를 완료해야 합니다.

1. Deployment Manager에서 응용프로그램 설치가 시작됩니다.

   이 스테이지 동안 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트는 WebSphere 구성 저장소에 구성됩니다. 응용프로그램의 유효성도 검사합니다. 오류가 발생하면 Deployment Manager의 FFDC 항목으로 또는 System.out 파일이나 System.err 파일에 오류가 보고됩니다.

2. Node Agent에서 응용프로그램 설치가 계속됩니다.

   이 스테이지 중에는 한 Application Server 인스턴스의 응용프로그램 설치가 트리거됩니다. 이 Application Server 인스턴스는 전개 대상이거나 대상의 일부입니다. 전개 대상이 여러 클러스터 멤버가 있는 클러스터인 경우 이 클러스터의 클러스터 멤버에서 서버 인스턴스가 임의로 선택됩니다. 이 스테이지 중 오류가 발생하면 Node Agent의 FFDC 항목으로 또는 SystemOut.log 파일이나 SystemErr.log 파일에 보고됩니다.

3. 응용프로그램이 서버 인스턴스에서 실행됩니다.

   이 스테이지에서는 프로세스 템플리트 및 휴먼 템플리트가 전개 대상의 Business Process Choreographer 데이터베이스로 전개됩니다. 오류가 발생하면 SystemErr.log 파일의 System.out 파일에 오류가 보고되거나 이 서버 인스턴스의 FFDC 항목으로 보고됩니다.

비즈니스 프로세스 및 휴먼 타스크 전개

WebSphere Integration Developer 또는 serviceDeploy를 사용하여 프로세스 컴포넌트를 엔터프라이즈 응용프로그램(EAR) 파일에 패키지화할 수 있습니다. 전개할 모델의 각 새 버전은 새 엔터프라이즈 응용프로그램으로 패키지해야 합니다.

비즈니스 프로세스 또는 휴먼 타스크를 포함하는 엔터프라이즈 응용프로그램을 설치 시 Business Process Choreographer 데이터베이스에 비즈니스 프로세스 템플리트 또는 휴먼 타스크 템플리트로 적절하게 저장됩니다. 기본값으로 최근에 설치된 템플리트는 시작됨 상태입니다. 그러나 최근에 설치된 엔터프라이즈 응용프로그램은 중지됨 상태입니다. 설치된 각 엔터프라이즈 응용프로그램은 개별적으로 시작하거나 중지할 수 있습니다.

프로세스 템플리트 또는 타스크 템플리트의 여러 다양한 버전을 서로 다른 엔터프라이즈 응용프로그램에 각각 전개할 수 있습니다. 버전은 유효 시작 날짜에 따라 다릅니다. 새 엔터프라이즈 응용프로그램을 설치하면 설치되는 템플리트의 버전이 다음과 같이 판별됩니다.

- 템플리트의 이름 및 대상 네임스페이스가 아직 존재하지 않는 경우 새 템플리트가 설치됩니다.
- 템플리트 이름 및 대상 네임스페이스가 기존 템플리트와 동일하지만 유효 시작 날짜가 다르면 기존 템플리트의 새 버전이 설치됩니다.

주: 템플리트의 이름은 비즈니스 프로세스나 휴먼 타스크가 아닌 컴포넌트의 이름에서 나옵니다.

유효 시작 날짜를 지정하지 않으면 날짜는 다음과 같이 지정됩니다.

- WebSphere Integration Developer를 사용하는 경우 유효 시작 날짜는 휴먼 타스크 또는 비즈니스 프로세스가 모델화된 날짜입니다.
- 서비스 전개를 사용하면 유효 시작 날짜는 serviceDeploy 명령이 실행된 날짜입니다. 공동 작업 타스크만이 응용프로그램이 설치된 날짜를 유효 시작 날짜로 사용합니다.

비즈니스 프로세스 및 휴먼 타스크 응용프로그램의 대화식 설치

wsadmin 도구 및 installInteractive 스크립트를 사용하여 런타임 시 대화식으로 응용프로그램을 설치할 수 있습니다. 이 스크립트를 사용하면 관리 콘솔을 사용하여 응용프로그램을 설치한 경우 변경하지 못하는 설정을 변경할 수 있습니다.

이 태스크 정보

비즈니스 프로세스 응용프로그램을 대화식으로 설치하려면 다음 단계를 수행하십시오.

프로시저

1. wsadmin 도구를 시작하십시오.

   profile_root/bin 디렉토리에 wsadmin을 입력하십시오.

2. 응용프로그램을 설치하십시오.

   wsadmin 명령행 프롬프트에서 다음 명령을 입력하십시오.

   ```
   $AdminApp installInteractive application.ear
   ```

   여기서, *application.ear*은 프로세스 응용프로그램이 있는 엔터프라이즈 아카이브 파일의 규정된 이름입니다. 일련의 타스크 과정에서 프롬프트되어 응용프로그램 값을 변경할 수 있습니다.

3. 구성 변경사항을 저장하십시오.

   wsadmin 명령행 프롬프트에서 다음 명령을 입력하십시오.

   ```
   $AdminConfig save
   ```

   마스터 구성 저장소에 갱신사항을 전송하려면 변경사항을 저장해야 합니다. 스크립트 프로세스를 종료하고 변경사항을 저장하지 않으면 변경사항은 버려집니다.

프로세스 응용프로그램 데이터 소스 및 세트 참조 설정 구성

특정 데이터베이스 인프라에 대한 SQL 문을 실행하는 프로세스 응용프로그램을 구성해야 할 수 있습니다. 이러한 SQL 문은 정보 서비스 활동에서 생성될 수도 있고 프로세스 설치 또는 인스턴스 시작 시 실행되는 명령문이 될 수도 있습니다.

이 태스크 정보

응용프로그램 설치 시, 다음 데이터 소스 유형을 지정할 수 있습니다.

- 프로세스 설치 시 SQL 문을 실행하는 데이터 소스
- 프로세스 인스턴스 시작 시 SQL 문을 실행하는 데이터 소스
- SQL 스니펫 활동을 실행하는 데이터 소스

SQL 스니펫 활동을 실행하는 데 필요한 데이터 소스는 tDataSource 유형의 BPEL 변수에서 정의됩니다. SQL 스니펫 활동에 필요한 데이터베이스 스키마 및 테이블 이름은 tSetReference 유형의 BPEL 변수에서 정의됩니다. 이들 두 변수 모두의 초기값을 구성할 수 있습니다.

wsadmin 도구를 사용하여 데이터 소스를 지정할 수 있습니다.

프로시저

1. wsadmin 도구를 사용하여 프로세스 응용프로그램을 대화식으로 설치하십시오.
2. 데이터 소스 및 세트 참조를 갱신하는 타스크가 나올 때까지 단계에 따라 타스크를 수행하십시오.

   환경에 맞게 설정을 구성하십시오. 다음 예는 이러한 각 타스크에서 변경할 수 있는 설정을 설명합니다.
3. 변경사항을 저장하십시오.

예: wsadmin 도구를 사용하여 데이터 소스 및 세트 참조 갱신

데이터 소스 갱신 타스크에서는 프로세스 설치 또는 시작 시 사용한 명령문 및 초기 변수값에 대한 데이터 소스 값을 변경할 수 있습니다. **세트 참조 갱신** 타스크에서는 데이터베이스 스키마 및 테이블 이름과 관련된 설정을 구성할 수 있습니다.

```
Task [24]: Updating data sources

//Change data source values for initial variable values at process start

Process name:  Test
// Name of the process template
Process start or installation time:  Process start
// Indicates whether the specified value is evaluated
//at process startup or process installation
Statement or variable:  Variable
// Indicates that a data source variable is to be changed
Data source name:  MyDataSource
// Name of the variable
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name to jdbc/newName
Task [25]: Updating set references

// Change set reference values that are used as initial values
for BPEL variables
```

```
Process name:  Test
// Name of the process template
Variable:  SetRef
// The BPEL variable name
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name of the data source of the
set reference to jdbc/newName
Schema name: [IISAMPLE]
// The name of the database schema
Schema prefix: []:
// The schema name prefix.
// This setting applies only if the schema name is generated.
Table name: [SETREFTAB]: NEWTABLE
// Sets the name of the database table to NEWTABLE
Table prefix: []:
// The table name prefix.
// This setting applies only if the table name is generated.
```

관리 콘솔을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거

관리 콘솔을 사용하여 비즈니스 프로세스 또는 휴먼 타스크를 포함하는 응용프로그램을 설치 제거할 수 있습니다.

시작하기 전에

비즈니스 프로세스 또는 휴먼 타스크가 포함된 응용프로그램을 설치 제거하려면 다음 조건을 적용해야 합니다.

- 응용프로그램이 독립형 서버에 설치된 경우 서버가 실행 중이고 Business Process Choreographer 데이터베이스에 액세스해야 합니다.
- 응용프로그램이 클러스터에 설치된 경우 Deployment Manager 및 최소 하나의 클러스터 멤버가 실행 중이어야 합니다. 클러스터 멤버가 Business Process Choreographer 데이터베이스에 대한 액세스 권한을 가져야 합니다.
- 응용프로그램이 관리 서버에 설치된 경우 Deployment Manager 및 관리 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.
- 상태에 관계 없이 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없거나 개발 모드로 실행 중인 독립형 서버가 있습니다.
- 프로세스 인스턴스가 새 프로세스 버전으로 이주되었지만 서비스 호출이 응답하기를 대기 중인 경우, 응답이 수신될 때까지 이전 버전을 포함하는 응용프로그램을 설치 제거할 수 없습니다. 그밖의 모든 경우에는 이주된 인스턴스가 새 버전의 인스턴스로 간주되며 프로세스의 이전 버전을 포함하는 응용프로그램을 설치 제거할 수 있습니다.

이 태스크 정보

비즈니스 프로세스 또는 휴먼 타스크를 포함하는 엔터프라이즈 응용프로그램을 설치 제거하려면 다음 조치를 수행하십시오.

프로시저

1. 관리 콘솔에서 **응용프로그램 → 응용프로그램 유형 → WebSphere 엔터프라이즈 응용프로그램**을 클릭하십시오.
2. 설치 제거하려는 응용프로그램을 선택하고 **중지**를 클릭하십시오.

   프로세스 인스턴스 또는 타스크 인스턴스가 여전히 응용프로그램에 있으면 이 단계가 실패합니다. Business Process Choreographer Explorer를 사용하여 인스턴스를 삭제하거나 bpcTamplates.jacl 관리 스크립트의 **-force** 옵션을 사용하여 인스턴스를 중지하고 삭제한 후 응용프로그램을 설치 제거할 수 있습니다.
3. 설치 제거할 응용프로그램을 선택하고 **설치 제거**를 클릭하십시오.
4. 변경사항을 저장하려면 **저장**을 클릭하십시오.

결과

응용프로그램이 설치 제거됩니다.

관련 태스크

『관리 명령을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거』
bpcTemplates.jacl 스크립트는 관리 콘솔을 대신하여 비즈니스 프로세스 또는 휴먼 타스크가 포함된 응용프로그램을 설치 제거합니다.

관리 명령을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거

bpcTemplates.jacl 스크립트는 관리 콘솔을 대신하여 비즈니스 프로세스 또는 휴먼 타스크가 포함된 응용프로그램을 설치 제거합니다.

시작하기 전에

비즈니스 프로세스 또는 휴먼 타스크가 포함된 응용프로그램을 설치 제거하려면 다음 조건을 적용해야 합니다.

- 응용프로그램이 독립형 서버에 설치된 경우 서버가 실행 중이고 Business Process Choreographer 데이터베이스에 액세스해야 합니다.
- 응용프로그램이 클러스터에 설치된 경우 Deployment Manager 및 최소 하나의 클러스터 멤버가 실행 중이어야 합니다. 클러스터 멤버가 Business Process Choreographer 데이터베이스에 대한 액세스 권한을 가져야 합니다.

- 응용프로그램이 관리 서버에 설치된 경우 Deployment Manager 및 관리 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.
- 관리 클라이언트가 연결되는 서버 프로세스가 실행 중인지 확인하십시오. 관리 클라이언트가 자동으로 서버 프로세스에 연결되는지를 확인하려면 명령 옵션으로 `-conntype NONE` 옵션을 사용하지 마십시오.
- WebSphere 관리 보안이 사용되고 사용자 ID에 운영자 또는 관리자 권한이 없는 경우에는 wsadmin `-user` 및 `-password` 옵션을 포함시켜서 운영자 또는 관리자 권한이 있는 사용자 ID를 지정하십시오. `-uninstall` 옵션을 사용하려면 운영자 권한이 필요하며 `-force` 옵션을 사용하려면 관리자 권한이 필요합니다.
- 다음 중 하나 이상이 참입니다.
  - 어떤 상태로든 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없습니다.
  - **-force** 옵션을 사용하려 합니다.
  - 개발 모드로 실행 중인 독립형 서버가 있습니다.
- 프로세스 인스턴스가 새 프로세스 버전으로 이주되었지만 서비스 호출이 응답하기를 대기 중인 경우, 응답이 수신될 때까지 이전 버전을 포함하는 응용프로그램을 설치 제거할 수 없습니다. 그밖의 모든 경우에는 이주된 인스턴스가 새 버전의 인스턴스로 간주되며 프로세스의 이전 버전을 포함하는 응용프로그램을 설치 제거할 수 있습니다.

이 태스크 정보

다음 단계에서는 bpcTemplates.jacl 스크립트를 사용하여 비즈니스 프로세스 템플리트 또는 휴먼 타스크 템플리트를 포함하는 응용프로그램을 설치 제거하는 방법에 대해 설명합니다.

프로시저

1. 설치 제거할 응용프로그램에 템플리트와 연관된 프로세스 인스턴스 또는 타스크 인스턴스가 여전히 있는 경우에는 다음 중 하나 또는 둘 다를 수행하십시오.
   - Business Process Choreographer Explorer를 사용하여 인스턴스를 삭제하십시오.
   - 설치 제거하려는 응용프로그램에 정의된 프로세스 템플리트에 종속되는 비즈니스 프로세스가 더 이상 없다고 확신하는 경우에는 **-force** 옵션을 사용할 수 있습니다.

주의:

이 옵션과 함께 스크립트를 사용하는 경우에는 스크립트가 한 단계에서 템플리트와 연관된 인스턴스 및 실행 중인 인스턴스와 연관된 모든 데이터를 삭제하고 템플리트를 중지한 후 응용프로그램을 설치 제거합니다. 이 옵션을 사용할 때는 매우 주의해야 합니다.

2. 관리 스크립트가 있는 Business Process Choreographer 서브디렉토리로 변경하십시오. 다음 명령을 입력하십시오.

```
cd install_root/ProcessChoreographer/admin
```

Linux UNIX Linux 및 UNIX 플랫폼에서 다음 명령을 입력하십시오.

```
cd install_root/ProcessChoreographer/admin
```

i5/OS® 플랫폼에서 다음 명령을 입력하십시오.

```
cd install_root/ProcessChoreographer/admin
```

Windows Windows 플랫폼에서 다음 명령을 입력하십시오.

```
cd install_root₩ProcessChoreographer₩admin
```

3. 템플리트를 중지하고 해당 응용프로그램을 설치 제거하십시오.

Windows Windows 플랫폼에서는 다음 명령을 입력하십시오.

```
install_root₩bin₩wsadmin -f bpcTemplates.jacl
                          -uninstall application_name
                         [-force]
```

Linux UNIX Linux 및 UNIX 플랫폼에서는 다음 명령을 입력하십시오.

```
install_root/bin/wsadmin -f bpcTemplates.jacl
                          -uninstall application_name
                         [-force]
```

-uninstall *application_name*

설치 제거할 응용프로그램의 이름을 지정합니다.

-force

이 옵션은 응용프로그램을 설치 제거하기 전에 실행 중인 모든 인스턴스를 중지하고 삭제합니다. 이 옵션 또한 실행 중인 인스턴스와 연관된 모든 데이터를 삭제하므로 주의해서 사용하십시오.

결과

응용프로그램이 설치 제거됩니다.

관련 태스크

569 페이지의『관리 콘솔을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거』

관리 콘솔을 사용하여 비즈니스 프로세스 또는 휴먼 타스크를 포함하는 응용프로그램을 설치 제거할 수 있습니다.

제 7 장 어댑터 및 설치

어댑터를 사용하여 사용자 응용프로그램과 엔터프라이즈 정보 시스템의 다른 컴포넌트 간 통신이 가능합니다.

어댑터 설치 프로세스에 대한 설명은 WebSphere Integration Developer Information Center의 어댑터 구성 및 사용에 있습니다.

제 8 장 실패한 전개 문제점 해결

이 주제에서는 응용프로그램을 전개할 때 문제점의 원인을 판별하기 위해 수행하는 단계에 대해 설명합니다. 또한 몇 가지 가능한 해결책을 제시합니다.

시작하기 전에

이 주제는 다음 사항을 가정합니다.
- 사용자가 모듈 디버깅에 대한 기본적인 내용을 이해합니다.
- 모듈이 전개되는 중에 로깅 및 추적이 활성화되어 있습니다.

이 태스크 정보

전개 문제점 해결 타스크는 오류 공고를 수신한 후에 시작됩니다. 조치를 수행하기 전에 검사해야 하는 실패한 전개의 증상은 여러 가지가 있습니다.

프로시저

1. 응용프로그램 설치가 실패했는지 확인하십시오.

   장애의 원인을 지정한 메시지를 보려면 `SystemOut.log` 파일을 확인하십시오. 응용프로그램이 설치될 수 없는 이유에는 다음이 포함됩니다.
   - 동일한 Network Deployment 셀의 다중 서버에 응용프로그램을 설치하려고 시도하고 있습니다.
   - 응용프로그램을 설치하려는 Network Deployment 셀의 기본 모듈의 이름과 응용프로그램 이름이 같습니다.
   - EAR 파일 내의 Java EE 모듈을 다른 대상 서버에 전개하려고 하는 중입니다.

   중요사항: 설치에 실패하고 응용프로그램에 서비스가 포함되는 경우 응용프로그램을 재설치하려고 하기 전에 장애 이전에 작성된 JCA 활성화 스펙 또는 SIBus 대상을 제거해야 합니다. 이러한 아티팩트를 가장 쉽게 제거하는 방법은 장애가 발생한 후에 **저장 > 모두 버리기**를 클릭하는 것입니다. 실수로 변경사항을 저장한 경우에는 SIBus 대상과 JCA 활성화 스펙을 수동으로 제거해야 합니다. (관리 절에서 SIBus 대상 삭제 및 JCA 활성화 스펙 삭제를 참조하십시오.)
2. 응용프로그램이 올바르게 설치된 경우 성공적으로 시작했는지 확인하십시오.

   응용프로그램이 성공적으로 시작되지 않은 경우 서버가 응용프로그램에 대한 자원을 시작하려고 시도할 때 장애가 발생합니다.

   a. 계속하는 방법을 지시하는 메시지를 보려면 `system.out` 파일을 확인하십시오.

b. 응용프로그램에 필요한 자원이 사용 가능하고 성공적으로 시작되었는지 판별하십시오.

   시작되지 않은 자원이 있으면 응용프로그램이 실행되지 않습니다. 정보 유실에 대비하여 보호됩니다. 자원이 시작되지 않는 이유는 다음과 같습니다.
   - 바인딩이 잘못 지정되었습니다.
   - 자원이 올바르게 구성되지 않았습니다.
   - 자원이 자원 아카이브(RAR) 파일에 포함되지 않았습니다.
   - 웹 자원이 웹 서비스 아카이브(WAR) 파일에 포함되지 않았습니다.

c. 누락된 컴포넌트가 있는지 판별하십시오.

   컴포넌트 누락 이유는 잘못 빌드된 엔터프라이즈 아카이브(EAR) 파일입니다. 모듈에 필요한 모든 컴포넌트가 Java 아카이브(JAR) 파일을 빌드한 테스트 시스템의 올바른 폴더에 있는지 확인하십시오. 자세한 정보는 『서버에 전개 준비』를 참조하십시오.

3. 응용프로그램을 통해 플로우되는 정보가 있는지 응용프로그램을 조사하십시오.

   실행 중인 응용프로그램도 정보 처리에 실패할 수 있습니다. 이유는 2b 단계에서 언급한 이유와 비슷합니다.

   a. 응용프로그램에서 다른 응용프로그램에 포함된 서비스를 사용하는지 확인하십시오. 기타 응용프로그램이 설치되어 성공적으로 시작되었는지 확인하십시오.

   b. 실패한 응용프로그램에서 사용하는 다른 응용프로그램에 포함된 장치의 가져오기 및 내보내기 바인딩이 올바르게 구성되었는지 확인하십시오. 관리 콘솔을 사용하여 바인딩을 확인하고 정정하십시오.

4. 문제점을 정정하고 응용프로그램을 다시 시작하십시오.

JCA 활성화 스펙 삭제

서비스가 포함된 응용프로그램을 설치할 때 시스템이 JCA 응용프로그램 스펙을 빌드합니다. 응용프로그램을 다시 설치하려면 먼저 이 스펙을 삭제해야 하는 경우가 있습니다.

시작하기 전에

응용프로그램 설치에 실패하여 스펙을 삭제하려는 경우 JNDI(Java Naming and Directory Interface) 이름의 모듈이 설치에 실패한 모듈의 이름과 일치하는지 확인하십시오. JNDI 이름의 두 번째 부분은 대상을 구현한 모듈의 이름입니다. 예를 들어, `sca/SimpleBOCrsmA/ActivationSpec`에서 **SimpleBOCrsmA**는 모듈 이름입니다.

이 타스크에 대한 필수 보안 역할: 보안 및 역할 기반 권한이 사용 가능한 경우 이 타스크를 수행하려면 관리자 또는 구성자로 로그인해야 합니다.

이 태스크 정보

서비스가 포함된 응용프로그램을 설치한 후 실수로 구성을 저장했거나 스펙이 필요 없는 경우 JCA 활성화 스펙을 삭제하십시오.

프로시저

1. 삭제할 활성화 스펙을 찾으십시오.

   스펙은 자원 어댑터 패널에 포함되어 있습니다. **자원 > 자원 어댑터**를 클릭하여 이 패널을 탐색하십시오.

   a. **플랫폼 메시징 컴포넌트 SPI 자원 어댑터**를 찾으십시오.

      이 어댑터를 찾으려면 사용자가 독립형 서버의 **노드** 범위에 있거나 전개 환경의 **서버** 범위에 있어야 합니다.

2. 플랫폼 메시징 컴포넌트 SPI 자원 어댑터와 연관된 JCA 활성화 스펙을 표시하십시오.

   자원 어댑터 이름을 클릭하면 다음 패널에 연관된 스펙이 표시됩니다.

3. 삭제하려는 모듈 이름과 일치하는 **JNDI 이름**이 있는 모든 스펙을 삭제하십시오.

   a. 해당 스펙 옆에 있는 선택란을 클릭하십시오.

   b. **삭제**를 클릭하십시오.

결과

시스템이 화면에서 선택한 스펙을 제거합니다.

다음에 수행할 작업

변경사항을 저장하십시오.

SIBus 대상 삭제

서비스 통합 버스(SIBus) 대상은 SCA 모듈이 처리 중인 메시지를 보유하는 데 사용됩니다. 문제점이 발생하면 버스 대상을 제거하여 문제점을 해결해야 합니다.

시작하기 전에

응용프로그램 설치에 실패하여 대상을 삭제하려는 경우 대상 이름의 모듈이 설치에 실패한 모듈의 이름과 일치하는지 확인하십시오. 대상의 두 번째 부분은 대상을 구현한 모듈의 이름입니다. 예를 들어, `sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer`에서 **SimpleBOCrsmA**는 모듈 이름입니다.

이 타스크에 대한 필수 보안 역할: 보안 및 역할 기반 권한이 사용 가능한 경우 이 타스크를 수행하려면 관리자 또는 구성자로 로그인해야 합니다.

이 태스크 정보

서비스가 들어 있는 응용프로그램을 설치한 후 실수로 구성을 저장했거나 대상이 더 이상 필요 없는 경우 SIBus 대상을 삭제하십시오.

주: 이 타스크는 SCA 시스템 버스에서만 대상을 삭제합니다. 또한 서비스가 포함된 응용프로그램을 다시 설치하려면 먼저 응용프로그램 버스에서 항목을 제거해야 합니다(해당 Information Center의 관리 절에 있는 JCA 활성화 스펙 삭제 참조).

프로시저

1. 관리 콘솔에 로그인하십시오.
2. SCA 시스템 버스에서 대상을 표시하십시오.
   a. 탐색 분할창에서 **서비스 통합 → 버스**를 클릭하십시오.
   b. 컨텐츠 분할창에서 **SCA.SYSTEM.*cell_name*.Bus**를 클릭하십시오.
   c. 대상 자원 아래에서 **대상**을 클릭하십시오.
3. 제거하는 모듈과 일치하는 모듈 이름을 가진 각 대상 옆의 선택란을 체크하십시오.
4. **삭제**를 클릭하십시오.

결과

패널에 남아 있는 대상만 표시됩니다.

다음에 수행할 작업

해당 대상을 작성한 모듈과 관련된 JCA 활성화 스펙을 삭제하십시오.