WebSphere® 멀티플랫폼용 Process Server

IBM

버전 6.2.0

모듈 개발 및 전개

WebSphere® 멀티플랫폼용 Process Server

버전 6.2.0

모듈 개발 및 전개

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 이 문서의 맨 마지막의 주의사항 섹션의 일반 정보를 읽으십시오.

2008년 12월 12일

이 개정판은 새 개정판에 별도로 명시하지 않는 한, 멀티플랫폼용 WebSphere Process Server의 버전 6, 릴리스 2, 수정 2(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.

PDF 서적 및 Information Center

PDF 서적은 인쇄 및 오프라인 사용을 위한 목적으로 제공됩니다. 최신 정보는 온라인 Information Center를 참조하십시오.

PDF 서적은 Information Center와 동일한 내용을 제공합니다.

PDF 문서는 버전 6.0 또는 버전 6.1과 같이 주요 Information Center 릴리스 이후 분기 내에 사용 가능합니다.

PDF 문서는 Information Center보다 갱신되는 빈도는 적지만 Redbooks®보다 자주 갱신됩니다. 일반적으로 PDF 서적은 변경사항이 많을 경우에 갱신됩니다.

PDF 서적 외부에 있는 주제에 대한 링크는 Information Center 웹 사이트로 연결합니다. PDF 서적 외부의 대상에 대한 링크는 대상이 PDF 서적 또는 웹 페이지인지를 나타내는 아이콘으로 표시됩니다.

표 1. 이 서적 외부에 있는 주제에 대한 링크 접두부가 붙은 아이콘

아이콘	설명

	Information Center의 페이지를 포함한 웹 페이지에 대한 링크
	
	Information Center에 대한 링크는 간접 라우팅 서비스를 통해 연결되기 때문에 대상 주제가 새 위치로 이동해도 지속적으로 작동합니다.
	
	로컬 Information Center에서 링크된 페이지를 찾으려는 경우 링크 제목을 검색할 수 있습니다. 또는 주제 ID를 검색할 수 있습니다. 다른 제품에 대해 여러 개의 검색 결과를 찾은 경우에는, **그룹별** 제어를 사용하여 확인하려는 주제 인스턴스를 식별할 수 있습니다. 예를 들어 다음과 같습니다.
	1. 링크 URL 복사는 링크를 마우스 오른쪽 단추로 클릭한 후 링크 위치 복사를 선택하십시오. 예: `http://www14.software.ibm.com/webapp/wsbroker/redirect?version=wbpm620&product=wesb-dist&topic=tins_apply_service`
	2. `&topic=` 뒤에 주제 ID를 복사하십시오. 예: `tins_apply_service`
	3. 로컬 Information Center의 검색 필드에 주제 ID를 붙여넣으십시오. 로컬로 설치되어 있는 문서 기능이 있으면 검색 결과에 주제가 표시됩니다. 예를 들어 다음과 같습니다.
	`1개의 결과를 찾음`
	`그룹 분류: 없음 | 플랫폼 | 버전 | 제품`
	`요약 표시`
	`설치 갱신 프로그램으로 픽스팩 및 갱신 팩 설치`
	4. 검색 결과에서 링크를 클릭하여 주제를 표시하십시오.

	PDF 서적의 링크

목차

PDF 서적 및 Information Center iii

그림 ix

표. xi

제 1 부 응용프로그램 개발 1

제 1 장 비즈니스 통합 솔루션 개발 3
비즈니스 통합 프로그래밍 모델 5
비즈니스 통합 아키텍처 및 패턴 6
비즈니스 통합 시나리오. 7
역할, 제품 및 기술 인증 확인 8
비즈니스 오브젝트 프레임워크 10
서비스 컴포넌트 아키텍처. 11
비즈니스 프로세스 16
휴먼 타스크 17
비즈니스 통합 응용프로그램 빌드 17

제 2 장 서비스 모듈 개발 19
모듈 개발 개요 19
서비스 컴포넌트 개발 21
컴포넌트 호출 23
동적으로 컴포넌트 호출 26
분리 모듈 및 대상 개요 27
HTTP 바인딩 31

제 3 장 프로그래밍 안내서 및 기술 33

제 4 장 생성된 서비스 컴포넌트 아키텍처 구현 대체 35

제 5 장 서비스 데이터 오브젝트와 Java 간 변환 대체 37

제 6 장 비SCA 내보내기 바인딩에서 프로토콜 헤더 전파. 39

제 7 장 Java에서 사용하는 런타임 규칙을 서비스 데이터 오브젝트로 변환 41

제 8 장 비즈니스 오브젝트: 스키마 향상 및 산업 스키마 지원 45
동일하게 이름 지정된 요소 구별 45
모델 그룹 지원(모두, 선택사항, 순서 및 그룹 참조) 46
동일하게 이름 지정된 특성 구별 49
마침표를 포함하는 특성 이름 해석. 50
xsi:type을 사용하여 유니온 직렬화 및 직렬화 해제 50
순서 오브젝트를 사용하여 데이터 순서 설정 . . . 51
내 DataObject에 순서가 있는지 확인하는 방법 53
DataObject에 순서가 있는지 알아야 하는 이유 53
혼합 컨텐츠에 대한 작업 방법 54
모델 그룹 배열에 대한 작업 방법 55
단순 유형에 대해 AnySimpleType 사용. 56
복합 유형에 대해 AnyType 사용 58
Any를 사용하여 복합 유형에 대한 글로벌 요소 설정 61
내 DataObject에 태그가 있는지 확인하는 방법 61
Any 값 가져오기/설정 방법 62
Any에 있는 데이터에 대해 유효한 맵핑. . . . 64
AnyAttribute를 사용하여 복합 유형에 대한 글로벌 속성 설정 64
내 DataObject에 anyAttribute 태그가 있는지 확인하는 방법 64
anyAttribute 값 가져오기/설정 방법 65
anyAttribute에 있는 데이터에 대해 유효한 맵핑 66

제 9 장 비즈니스 오브젝트의 배열 67

제 10 장 중첩 비즈니스 오브젝트 작성 69
중첩 비즈니스 오브젝트의 단일 인스턴스. 69
중첩 비즈니스 오브젝트의 다중 인스턴스 작성. . . 70
와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용 72
모델 그룹의 비즈니스 오브젝트 사용 74

제 11 장 XML 문서 유효성 검증 77

제 12 장 비즈니스 규칙 관리 81
프로그래밍 모델 82
비즈니스 규칙 그룹. 82
비즈니스 규칙 그룹 특성 84
조작. 86
비즈니스 규칙 89
규칙 세트 91
의사결정 테이블 93
템플리트 및 매개변수. 102
유효성 검증 104
변경사항 추적 105

BusinessRuleManager 105
예외 처리 109
권한 112
예 112
예제 1: 모든 비즈니스 규칙 그룹 검색 및 인쇄 113
예제 2: 비즈니스 규칙 그룹, 규칙 세트 및 결정 테이블 검색 및 인쇄 116
예제 3: AND를 이용해 다중 특성으로 비즈니스 규칙 그룹 검색. 120
예제 4: OR을 이용해 다중 특성으로 비즈니스 규칙 그룹 검색. 122
예제 5: 복합 조회를 이용해 비즈니스 규칙 그룹 검색 124
예제 6: 비즈니스 규칙 그룹 특성 갱신 및 공개 126
예제 7: 다중 비즈니스 규칙 그룹에서 특성 갱신 및 공개 128
예제 8: 비즈니스 규칙 그룹의 기본값 비즈니스 규칙 변경 130
예제 9: 비즈니스 규칙 그룹에서 조작의 다른 규칙 스케줄 133
예제 10: 규칙 세트에서 템플리트의 매개변수 값 수정 136
예제 11: 템플리트에서 규칙 세트로 새 규칙 추가 140
예제 12: 매개변수 값을 변경하여 결정 테이블에서 템플리트 수정 후 공개 144
예제 13: 조건 값 및 조치를 결정 테이블에 추가 152
예제 14: 규칙 세트에서 오류 핸들 163
예제 15: 비즈니스 규칙 그룹에서 오류 핸들 166
부록 171
포맷터 클래스 171
RuleArtifactUtility 클래스. 172
조회 예제 추가. 182

제 13 장 비즈니스 프로세스 및 타스크용 클라이언트 응용프로그램 개발 199
비즈니스 프로세스 및 휴먼 타스크와 상호작용에 사용되는 프로그래밍 인터페이스 비교 199
비즈니스 프로세스 및 타스크 데이터에 대한 조회 201
Business Process Choreographer에서 테이블 조회 201
Business Process Choreographer EJB 조회 API 217
비즈니스 프로세스 및 휴먼 타스크용 EJB 클라이언트 응용프로그램 개발 231
EJB API에 액세스 233
비즈니스 프로세스 및 타스크 관련 오브젝트 조회 240
비즈니스 프로세스용 응용프로그램 개발 . . . 245
휴먼 타스크용 응용프로그램 개발. 267
비즈니스 프로세스 및 휴먼 타스크용 응용프로그램 개발 287
예외 및 결함 처리. 293
웹 서비스 API 클라이언트 응용프로그램 개발 . . 296
웹 서비스 컴포넌트 및 제어 순서. 297
웹 서비스 API 개요 298
비즈니스 프로세스 및 휴먼 타스크 요구사항 299
클라이언트 응용프로그램 개발. 299
아티팩트 복사 300
Java 웹 서비스 환경에서 클라이언트 응용프로그램 개발 309
.NET 환경에서 클라이언트 응용프로그램 개발 320
비즈니스 프로세스 및 타스크 관련 오브젝트 조회 325
Business Process Choreographer JMS API를 사용한 클라이언트 응용프로그램 개발 328
비즈니스 프로세스의 요구사항. 329
JMS 렌더링 권한 329
JMS 인터페이스 액세스. 330
JMS 클라이언트 응용프로그램의 아티팩트 복사 334
비즈니스 예외의 응답 메시지 점검 334
예제: Business Process Choreographer JMS API를 사용하여 장기 실행 중 프로세스 실행. . 335
JSP 컴포넌트를 사용하여 비즈니스 프로세스 및 휴먼 타스크용 웹 응용프로그램 개발 336
Business Process Choreographer 탐색기 컴포넌트 339
JSF 컴포넌트의 오류 핸들링 341
클라이언트 모델 오브젝트의 기본 변환기 및 레이블 342
JSF 응용프로그램에 목록 컴포넌트 추가 . . . 342
JSF 응용프로그램에 세부사항 컴포넌트 추가 349
JSF 응용프로그램에 CommandBar 컴포넌트 추가 352
JSF 응용프로그램에 메시지 컴포넌트 추가 . . 357
타스크 및 프로세스 메시지에 대한 JSP 페이지 개발 360
사용자 정의 JSP 단편 361
휴먼 타스크 기능을 사용자 정의하는 플러그인 작성 362
API 이벤트 핸들러 작성 363
공고 이벤트 핸들러 작성 366

API 이벤트 핸들러 및 공고 이벤트 플러그인 설치 368
타스크 템플리트, 타스크 모델 및 타스크에 API 이벤트 핸들러 및 공고 이벤트 핸들러 플러그인 등록 369
사용자 조회 결과를 사후 처리하는 플러그인 작성, 설치 및 실행 369

제 2 부 응용프로그램 전개 375

제 14 장 모듈 준비 및 설치 개요 377
라이브러리 및 Jar 파일 개요 377
EAR 파일 개요 379
서버에 전개 준비 380
클러스터에 서비스 응용프로그램 설치에 대한 고려사항 382

제 15 장 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 383
비즈니스 프로세스 및 휴먼 타스크 응용프로그램을 Network Deployment 환경에 설치하는 방법. . . 383
비즈니스 프로세스 및 휴먼 타스크 전개 384
비즈니스 프로세스 및 휴먼 타스크 응용프로그램의 대화식 설치 385
프로세스 응용프로그램 데이터 소스 및 세트 참조 설정 구성 386
관리 콘솔을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거 387
관리 명령을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거 388

제 16 장 어댑터 및 설치 391

제 17 장 실패한 전개 문제점 해결 393
J2C 활성화 스펙 삭제 394
SIBus 대상 삭제 395

제 3 부 부록 397

주의사항 399

그림

1. 전체 BPM 라이프 사이클에서 사용자 프로세스를 모델링, 어셈블링, 전개 및 관리하는 IBM 도구 5
2. WebSphere Process Server 컴포넌트 기반 프레임워크 12
3. WebSphere Process Server에서의 SCA 13
4. 어셈블리 다이어그램 14
5. 단순 호출 모델 28
6. 여러 개의 응용프로그램이 하나의 서비스를 호출. 29
7. UpdateCalculateFinal을 호출하는 분리 호출 모델 30
8. UpdatedCalculateFinal을 호출하는 분리 호출 모델 31
9. 프로토콜 헤더를 포함하는 컨텍스트 전파 40
10. BusinessRuleGroup 및 관련 클래스의 클래스 다이어그램 84
11. 특성 및 관련 클래스의 클래스 다이어그램 86
12. Operation 및 관련 클래스의 클래스 다이어그램 89
13. BusinessRule 및 관련 클래스의 클래스 다이어그램 90
14. BusinessRule 및 관련 클래스의 클래스 다이어그램 93
15. DecisionTable 및 관련 클래스의 클래스 다이어그램 94
16. TreeNode 및 관련 클래스의 클래스 다이어그램 97
17. TreeAction 및 관련 클래스의 클래스 다이어그램 101
18. DecisionTableRule 및 관련 클래스의 클래스 다이어그램 102
19. 템플리트 및 매개변수와 관련 클래스의 클래스 다이어그램 104
20. BusinessRuleManager 및 패키지의 클래스 다이어그램 105
21. QueryNodeFactory 및 관련 클래스의 클래스 다이어그램 108
22. BusinessRuleManagementException 및 관련 클래스의 클래스 다이어그램. 109
23. 모듈, 컴포넌트 및 라이브러리의 관계 378

표

1. 이 서적 외부에 있는 주제에 대한 링크 접두부가 붙은 아이콘 iii
2. 데이터 추상 및 해당 구현. 10
3. WSDL 유형을 Java 클래스로 변환 43
4. 비즈니스 규칙 그룹 문제점 110
5. 규칙 세트 및 결정 테이블 문제점. 110
6. 218
7. 프로세스 템플리트의 API 메소드 264
8. 프로세스 인스턴스 시작과 관련이 있는 API 메소드. 265
9. 프로세스 인스턴스의 라이프 사이클을 제어하는 API 메소드 265
10. 활동 인스턴스의 라이프 사이클 제어용 API 메소드. 266
11. 변수 및 사용자 정의 특성에 대한 API 메소드 267
12. 타스크 템플리트의 API 메소드. 284
13. 타스크 인스턴스의 API 메소드. 285
14. 에스컬레이션 작업에 사용되는 API 메소드 285
15. 변수 및 사용자 정의 특성에 대한 API 메소드 286
16. JNDI 이름에 대한 참조 바인딩 맵핑 339
17. Business Process Choreographer가 클라이언트 모델 오브젝트로 맵핑되는 방법 342
18. bpe:list 속성. 349
19. bpe:column 속성 349
20. bpe:details 속성 351
21. bpe:property 속성 352
22. bpe:commandbar 속성 356
23. bpe:command 속성 356
24. bpe:form 속성 359

제 1 부 응용프로그램 개발

제 1 장 비즈니스 통합 솔루션 개발

이 섹션에서는 비즈니스 통합 프로그래밍 모델의 기본사항에 대해 설명합니다. SCA(Service Component Architecture)를 소개하며 비즈니스 통합과 관련된 패턴을 설명합니다.

비즈니스 통합은 회사가 비즈니스 프로세스를 식별, 통합 및 최적화시킬 수 있는 부문입니다. 목적은 생산성을 개선시키고 조직의 효율성을 개선시키는 것입니다. 비즈니스 통합에서의 이해 관계는 회사 병합 및 합병과 같이 더 민감해지며, 다른 정보 자산의 레거시를 증대시킴에 따라 더 그러합니다. 이러한 자산은 일관성 및 통합 기능이 적으므로"정보의 섬(islands of information: 지리자료가 수치적으로 변환, 저장, 분석, 디스플레이 되는 독립시스템)"문제가 발생합니다.

비즈니스 통합은 BPM(Business Process Management) 및 SOA(Service-Oriented Architecture)에 대한 강한 링크를 갖습니다. 회사의 네이처와 통합 필요 범위에 따라, 비즈니스 통합은 IT 부서의 다른 요구사항을 내포합니다. 일부 프로젝트는 오직 몇 개의 측면만을 처리하는 반면, 일부 더 큰 프로젝트는 이들 요구사항 중 많은 것을 포함합니다. 여기에는 비즈니스 통합 프로젝트의 가장 일반적인 측면 중 일부가 있습니다.

- **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 복합한 상황까지 다양합니다. 복합 응용프로그램 통합은 자주 변환 및 맵핑은 물론 복합 작업 단위 관리를 필요로 합니다.
- **프로세스 자동화**는 개인 또는 조직에 의해 조직적으로 수행되는 활동이 다른 곳에서의 필연적인 활동을 트리거함을 확인하는 또 다른 키의 양상입니다. 이것은 전체 비즈니스 프로세스의 성공적인 완료를 보장합니다. 예를 들어, 회사에서 직원을 고용할 때, 급여 정보가 갱신되어야 하며, 보안 부서에서의 적절한 조치가 필요하며, 직원에게 필요한 도구를 제공할 필요가 있다는 것 등입니다. 프로세스에서의 일부 조치는 인간의 입력과 상호작용을 캡처할 수 있지만, 다른 조치는 백엔드 시스템에서 스크립트를 호출하고 해당 환경에서 다른 서비스를 호출할 수 있습니다.
- **연결성**은 회사 및 비즈니스 파트너 측면 양쪽에서 추상적이나 중요합니다. 연결성에 의해, 조직이나 회사 간의 정보 플로우, 그리고 분산된 IT 서비스를 액세스하는 기능 둘 다를 의미합니다.

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

- 다른 데이터 형식을 처리하므로 효율적인 데이터 변환을 수행할 수 없음
- 매우 다른 기술을 사용하여 발전해온 IT 서비스를 액세스하기 위해 다른 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, 비즈니스 통합은 SOA에 공통인 많은 요소와 주제를 포괄합니다. 비즈니스 통합에 대한 IBM의 비전은 SOA에서와 동일한 여러 기본적인 개념을 토대로 합니다. 이러한 비전의 즉각적인 결과 중 하나는 비즈니스 통합 솔루션의 실현을 위해 다양한 제품을 필요로 할 수 있다는 것입니다. IBM®에서는 모든 다양한 단계와 조작적인 측면을 지원하는 도구 및 런타임 플랫폼의 포트폴리오를 제공합니다.

비즈니스 통합의 IBM 비전을 바꾸어 말하면, SOA IT 하부 구조에서 실행하는 응용프로그램을 사용하여 회사가 비즈니스 프로세스를 정의, 작성, 병합, 결합 및 능률화시킬 수 있어야 합니다. 비즈니스 통합 작업은 사실 역할 기반입니다. 매크로 레벨에서, 프로세스 응용프로그램을 모델링하고, 개발하고, 거버넌스(Governance)하고, 관리하며 모니터링하는 것과 관련됩니다. 적절한 도구 및 프로시저의 도움으로, 엔터프라이즈 내부 및 외부 양쪽으로 사람 및 이기종 시스템과 관련하여 비즈니스 프로세스를 자동화할 수 있게 합니다. 비즈니스 통합의 핵심 측면 중 하나는 사용자 비즈니스 조작을 최적화하는 기능으로 효율적이며, 확장 가능하며, 신뢰성 있고 유연하여 충분히 변경 처리할 수 있게 하는 것입니다.

비즈니스 통합은 개발 도구, 런타임 서버, 모니터링 도구, 서비스 저장소, 툴킷 및 프로세스 템플리트를 필요로 합니다. 비즈니스 통합에는 너무 많은 측면이 있기 때문에, 솔루션을 개발하기 위해서는 둘 이상의 개발 도구를 이용해야 함을 압니다. 이들 도구는 통합 개발자가 복합 비즈니스 솔루션을 어셈블할 수 있게 합니다. 서버는 복합 응용프로그램을 실행하는 고성능 비즈니스 엔진 또는 서비스 컨테이너입니다. 관리는 조직에서 누가 무엇을 수행하고 있는가를 아는 것이며 모니터링 도구가 활동하기 시작하는 곳입니다. 엔터프라이즈가 이들 비즈니스 프로세스나 서비스를 작성함에 따라, 이들 서비스의 거버넌스(Governance), 분류 및 저장영역이 중요해집니다. 해당 기능은 서비스 저장소에서 제공합니다. 시스템을 레거시하기 위한 어댑터나 커넥터와 같이, 특수화된 솔루션 부분을 작성하는 특정 툴킷이 자주 필요합니다.

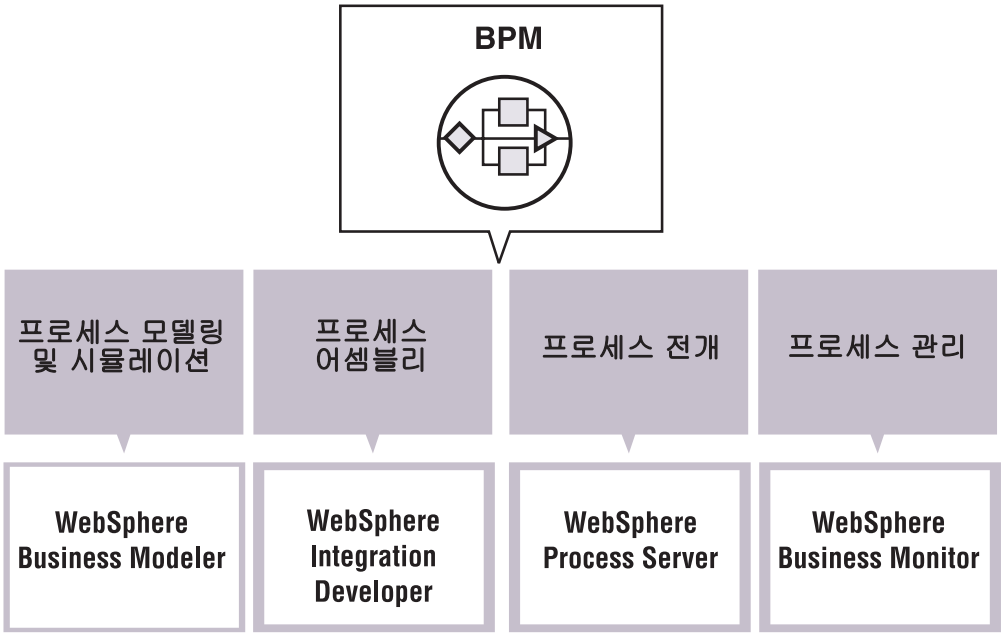


그림 1. 전체 BPM 라이프 사이클에서 사용자 프로세스를 모델링, 어셈블링, 전개 및 관리하는 IBM 도구

비즈니스 통합은 단일 제품을 대상으로 하지 않습니다. 조직 내에서 그리고 조직 전체에서 거의 모든 사용자와 비즈니스 측면 전체와 관련됩니다. 비즈니스 통합은 SOA 참조 아키텍처에서 여러 서비스와 요소를 포함합니다.

프로그래밍 예제와 함께 이들 개념에 대한 자세한 사항은 다음을 참조하십시오.

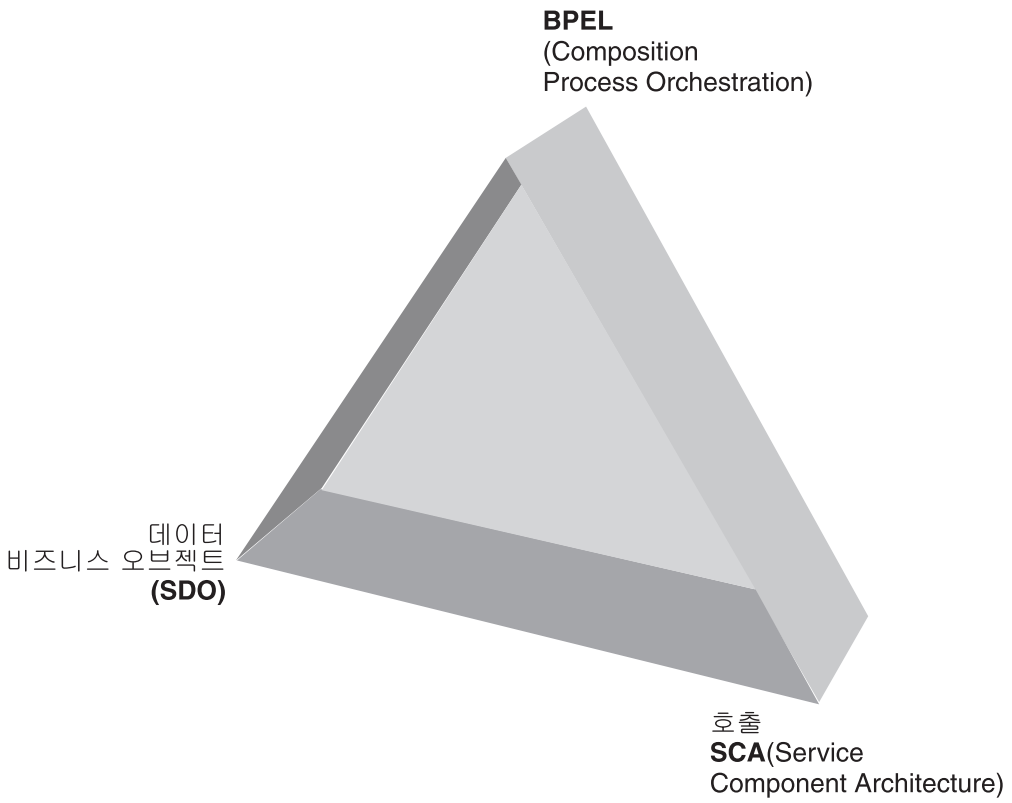
- *WebSphere® Business Integration Primer: Process Server, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1: Development*, IBM Redbooks, SG24-7608-00, June 2008.

비즈니스 통합 프로그래밍 모델

비즈니스 통합은 손쉬운 타스크가 아닙니다. 통합 성취가 손쉬운 타스크가 아닌 데이터를 표시하거나 상호작용하는 여러 가지 방법과 많은 기술이 있습니다. 데이터, 호출 및 작성, 이 프로그래밍 모델의 세 가지 측면을 얻고 서비스 기반 접근의 새 패러다임 일부를 적용하는 경우, SQA의 새 프로그래밍 모델이 나타나기 시작합니다.

우선, XML(Extensible Markup Language)에 의해 표시되며 SDO(Service Data Object)를 이용해서 또는 XPath나 XSLT(Extensible Stylesheet Language Transformation)와 같은 기본 XML 기능을 통해 프로그래밍됩니다. 두 번째로, 서비스 호출은 SCA(Service Component Architecture)로 맵핑합니다. 마지막으로,

BPEL(Business Process Execution Language)을 사용하여 프로세스 통합으로 구체화됩니다. 그림에서는 이 새로운 프로그래밍 모델의 세 가지 측면을 표시합니다.



서비스 컴포넌트 아키텍처

서비스 호출의 일관된 구문 및 메커니즘을 제공하는 이외에, SCA는 재사용 가능한 컴포넌트에서 개발자가 서비스 구현을 캡슐화하는 방법을 제공하는 호출 프레임워크입니다. 개발자가 기술 불가지론적인 방법으로 인터페이스, 구현 및 참조를 정의하게 하며, 사용자가 선택한 기술에 상관없이 요소를 바인드하는 기회를 제공합니다. SCA는 하부 구조에서 비즈니스 로직을 분리하여 응용프로그램 프로그래머가 비즈니스 문제점을 해결하는 데 초점을 둘 수 있게 합니다.

비즈니스 통합 아키텍처 및 패턴

일반 비즈니스 통합 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. 비즈니스 통합 솔루션을 개발하기 위해 사용되는 프로그래밍 모델에 의해 가장 잘 지정됩니다.

이 섹션에서는 SCA(Service Component Architecture)를 소개하며 비즈니스 통합과 관련된 패턴을 설명합니다. 패턴이 우리 삶에 퍼져 있는 듯 합니다. 바느질 패턴, 아이들

이 생각하고 배우는 패턴, 주택 건설 패턴, 목각 패턴, 비행 패턴, 바람 패턴, 의학에서의 실습 패턴, 고객 구매 패턴, 워크플로우 패턴, 전산학에서의 디자인 패턴 및 다른 많은 패턴이 있습니다.

패턴이 솔루션 디자이너 및 개발자를 돕는 데 성공적이었음이 증명되어 왔습니다. 그러므로, 이제 비즈니스 통합 패턴 및 엔터프라이즈 통합 패턴을 갖는 것이 놀라운 일이 아닙니다. 요청과 응답 라우팅, 채널 패턴(예: 공개/서명) 등을 포함하여 비즈니스 통합에 적용 가능한 광범위한 배열 패턴이 있습니다. 추상 패턴은 문제점의 특정 범주를 해석하는 템플리트를 제공하는 반면, 구체적 패턴은 특정 솔루션을 구현하는 방법에 대한 특정 표시를 더 많이 제공합니다. 이 섹션에서는 WebSphere 비즈니스 통합을 위한 IBM 소프트웨어 전략의 프로그래밍 모델 토대가 되며, 데이터와 서비스를 처리하는 패턴에 초점을 둡니다.

비즈니스 통합 시나리오

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

가장 일반적인 비즈니스 프로세스 통합 시나리오는 다음과 같습니다.

- **통합 브로커**: 이 시나리오에서, 비즈니스 통합 솔루션은 다양한 ″백엔드″ 응용프로그램 간의 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 서비스 간에 아교로서 역할하며 그렇지 않으면 관련이 없습니다. 예를 들어, 회사에서 사원을 채용할 때, 다음과 같은 일련의 조치가 일어날 필요가 있습니다.
  - 사원 정보가 급여 시스템에 추가됩니다.
  - 기능에 대한 물리적 액세스를 사원에게 부여할 필요가 있으며, 명찰을 제공해야 합니다.
  - 회사에서는 물리적 자원 파일 세트를 사원(오피스 영역, 컴퓨터 등)에게 지정할 필요가 있습니다.
  - IT 부서에서는 사원의 사용자 프로파일을 작성하고 일련의 응용프로그램에 대한 액세스를 부여할 필요가 있습니다.

    이 프로세스를 자동화하는 것도 비즈니스 통합 시나리오에서의 일반 유스 케이스입니다. 이 시나리오에서, 솔루션은 급여 시스템에 사원을 추가함으로써 트리거되는 자동화된 플로우를 구현합니다. 차후에, 조치를 취하는 데 관여하는 사람들의 작업 항목을 작성하거나 해당 서비스를 호출하여 플로우에서 다른 단계를 트리거합니다.

양쪽 시나리오에서, 통합 솔루션은 다음을 수행할 필요가 있습니다.

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 도구, 프로그래밍 기술 및 도구 스위트를 블렌드하는 것에 의해 좌우됩니다.

비즈니스 통합 프로젝트는 몇 가지 기본 구성 요소를 요구합니다.

- 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 섹션에서는 이러한 각 구성 요소를 정교하게 만듭니다.

명확한 역할 분리

비즈니스 통합 프로젝트는 네 가지가 협조적이나 명확히 구분된 역할의 사람을 요구합니다.

- **비즈니스 분석자**: 비즈니스 분석자는 프로세스의 비즈니스 측면을 캡처하고 프로세스 자체를 적절하게 나타내는 프로세스 모델을 작성하는 데 관련됩니다. 이들의 초점은 프로세스의 재정 성능을 최적화하는 것입니다. 비즈니스 분석자는 프로세스를 구현하는 기술적인 측면과 관련되지 않습니다.
- **컴포넌트 개발자**: 컴포넌트 개발자는 각 서비스와 컴포넌트를 구현하는 데 관여합니다. 이들의 초점은 구현에 사용되는 특정 기술입니다. 이 역할에서는 강한 프로그래밍 백그라운드를 요구합니다.
- **통합 전문가**: 상대적으로 새로운 이 역할은 기본 컴포넌트 세트를 더 큰 비즈니스 통합 솔루션으로 어셈블하는 데 관여하는 사람을 설명합니다. 통합 개발자는 그들이 재사용하고 함께 연결하는 각 컴포넌트와 서비스의 기술적인 세부사항을 알 필요가 없습니다. 이론상으로, 통합 개발자는 어셈블링하는 서비스의 인터페이스를 이해하는 것에만 관련됩니다. 통합 개발자는 어셈블리 프로세스의 통합 도구에 의지해야 합니다.
- **솔루션 전개자**: 솔루션 개발자와 관리자는 일반 사용자에게 사용 가능한 비즈니스 통합 솔루션과 관련됩니다. 이론상으로, 솔루션 전개자는 솔루션이 기능하도록 물리적

자원을 준비시키도록 솔루션과 바인딩하는 데 관여하며(데이터베이스, 대기열 관리자 등) 솔루션의 내부를 완전히 이해하는 것과 관련되지 않습니다. 솔루션의 전개자 초점은 서비스의 품질(QoS)입니다.

공통 비즈니스 오브젝트 모델

설명한 바와 같이, 비즈니스 통합 프로젝트의 핵심 측면은 여러 컴포넌트의 호출을 조정하고 컴포넌트 간의 데이터 교환을 핸들하는 기능을 포함합니다. 특히, 여러 컴포넌트는 여러 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 엔터프라이즈 Java™ Beans(EJB)을 사용하는 Java 응용프로그램을 통합해야 할 수 있습니다. 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 이루어집니다.

비즈니스 오브젝트 프레임워크는 개발자가 XML 스키마를 사용하여 Java 코드를 통해 비즈니스 데이터의 구조를 정의하고 이들 데이터 구조(비즈니스 오브젝트)의 인스턴스를 액세스하고 조작하게 합니다. 비즈니스 오브젝트 프레임워크는 서비스 데이터 오브젝트(SDO) 표준에 기반을 둡니다.

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. 사용자 기술 불가지론적인 방법으로 인터페이스, 구현 및 참조를 정의하게 하며, 사용자가 선택한 어떤 기술이든 요소를 바인드하는 기회를 제공합니다. 이러한 컴포넌트의 호출을 사용 가능하게 하는 SCA 클라이언트 프로그래밍 모델도 있습니다. 특히, 런타임 하부 구조가 Java에 근거하여 비Java 런타임과 상호작용하게 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 도구가 있습니다. 일반적으로 이들 솔루션은 WebSphere Process Server로 전개되며, 어떤 경우에는 WebSphere Enterprise Service Bus로 전개됩니다.

비즈니스 오브젝트 프레임워크

컴퓨터 소프트웨어 산업은 개발자가 비즈니스 오브젝트(BO) 정보를 캡슐화할 수 있는 여러 프로그래밍 모델과 프레임워크를 개발해 왔습니다. 일반적으로, BO 프레임워크는 데이터베이스 독립성을 제공해야 하며, 엔터프라이즈 정보 시스템의 데이터 구조로 또는 데이터베이스 테이블로 사용자 정의 비즈니스 오브젝트를 명백히 맵핑시키며, 비즈니스 오브젝트를 사용자 인터페이스로 바인드합니다. XML 스키마는 최근에 비즈니스 오브젝트의 구조를 표현하는 가장 많이 활용되며 인정 받는 방법입니다.

도구 Perspective에서, WebSphere Integration Developer는 여러 도메인에서 여러 종류의 엔티티를 표현할 수 있도록 개발자에게 일반 BO 모델을 제공합니다. 개발 시, WebSphere Integration Developer는 XML 스키마로서 비즈니스 오브젝트를 표시합니다. 그러나 런타임 시 이러한 동일 비즈니스 오브젝트가 SDO의 Java 인스턴스에 의해 메모리에 표시됩니다. SDO는 IBM 및 BEA System이 공동으로 개발하고 동의한 표준 스펙입니다. IBM은 비즈니스 오브젝트 내에서 데이터 처리를 용이하게 하는 일부 추가적인 서비스를 포함하여 SDO 스펙을 확장했습니다.

BO 프레임워크에 대해 설명하기 전에 처리된 데이터의 기본 유형을 자세히 살펴보면 다음과 같습니다.

- **인스턴스 데이터**는 스칼라 특성을 가진 단순한 기본적인 오브젝트에서부터 오브젝트의 복잡한 계층 구조까지 실제 데이터 및 데이터 구조입니다. 또한 이것은 기본 속성 유형, 복합 유형 정보, 카디널리티(cardinality) 및 기본값의 설명과 같은 데이터 정의를 포함합니다.
- **인스턴스 메타데이터**는 인스턴스 고유의 데이터입니다. 증분 정보는 변경 트래킹(변경 요약으로도 알려짐), 오브젝트와 데이터가 작성된 방식과 연관된 컨텍스트 정보, 그리고 메시지 헤더 및 바닥글과 같은 기본 데이터에 추가됩니다.
- **유형 메타데이터**는 보통 대상 엔터프라이즈 정보 시스템(EIS) 데이터 열(예를 들어, BO 필드 이름을 SAP 테이블 열 이름으로 맵핑)로의 속성-레벨 맵핑과 같이 보통 응용프로그램 고유의 정보입니다.
- **서비스**는 기본적으로 데이터를 얻거나, 데이터를 설정하거나, 요약을 변경하거나 데이터 정의 유형 액세스를 제공하는 헬퍼 서비스입니다.

다음 표에서는 WebSphere 플랫폼에서 기본 데이터 유형의 구현 방법을 표시합니다.

표 2. 데이터 추상 및 해당 구현

데이터 추상	구현
인스턴스 데이터	비즈니스 오브젝트(SDO)
인스턴스 메타데이터	비즈니스 그래프
유형 메타데이터	엔터프라이즈 메타데이터, 비즈니스 오브젝트 유형 메타데이터
서비스	비즈니스 오브젝트 서비스

IBM 비즈니스 오브젝트 프레임워크에 대한 작업

언급한 대로, WebSphere Process Server BO 프레임워크는 SDO 표준의 확장입니다. 따라서, WebSphere Process Server 컴포넌트 간에 교환되는 비즈니스 오브젝트는 `commonj.sdo.DataObject` 클래스의 인스턴스입니다. 그러나, WebSphere Process Server BO 프레임워크는 기본 DataObject 기능성을 간소화하고 강화하는 여러 서비스 및 기능을 추가합니다.

비즈니스 오브젝트의 작성 및 처리를 용이하게 하기 위해, WebSphere BO 프레임워크는 Java 서비스 세트를 제공하여 SDO 스펙을 확장합니다. 이들 서비스는 `com.ibm.websphere.bo`라는 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″ 시멘틱).
- **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**: 이들 서비스는 `commonj.sdo`.Type의 인스턴스를 구체화하며 연관된 메타데이터를 처리하게 합니다. 그런 다음 비즈니스 오브젝트 ″by type″를 작성하기 위해 유형 인스턴스를 사용할 수 있습니다.
- **BOInstanceValidator** : 비즈니스 오브젝트의 데이터가 XSD에 따르는지 여부를 알기 위해 유효성 검증합니다.

서비스 컴포넌트 아키텍처

SCA는 여러 가지 방식으로 구현할 수 있는 추상적 개념입니다. 특정 기술, 프로그래밍 언어, 호출 프로토콜 또는 전송 메커니즘을 요구하지 않습니다. SCA 컴포넌트는 XML 기반 언어인 SCDL(Service Component Definition Language)을 사용하여 설명됩니다.

SCA 컴포넌트에는 다음과 같은 특성이 있습니다.

- 컴포넌트가 실행할 수 있는 로직을 포함하는 구현 아티팩트를 랩핑합니다.

- 하나 이상의 인터페이스를 나타냅니다.
- 다른 컴포넌트에 대해 하나 이상의 참조를 나타낼 수 있습니다. 구현 로직은 컴포넌트가 참조를 나타내는지 여부를 결정합니다. 구현에서 다른 서비스 호출을 필요로 하는 경우, SCA 컴포넌트는 참조를 나타낼 필요가 있습니다.

이 정보는 WebSphere Process Server가 제공하는 SCA 구현과 SCA 컴포넌트를 작성하고 결합시킬 수 있는 WebSphere Integration Developer 도구에 중점을 두고 있습니다. WebSphere Process Server 및 WebSphere Integration Developer는 다음과 같은 구현 아티팩트를 지원합니다.

- 일반 Java 오브젝트
- 비즈니스 프로세스
- 비즈니스 상태 머신
- 휴먼 타스크
- 비즈니스 규칙
- 중개 플로우

SCA는 하부 구조와 비즈니스 로직을 분리하므로 응용프로그램 프로그래머가 비즈니스 문제점을 해결하는 데 집중할 수 있습니다. IBM WebSphere Process Server는 이와 동일한 전제에 기초합니다. 그림 2에서는 WebSphere Process Server의 구조적 모델을 보여줍니다.

단일 컴포넌트 기반 프레임워크로 모든 종류의 통합을 표시합니다.

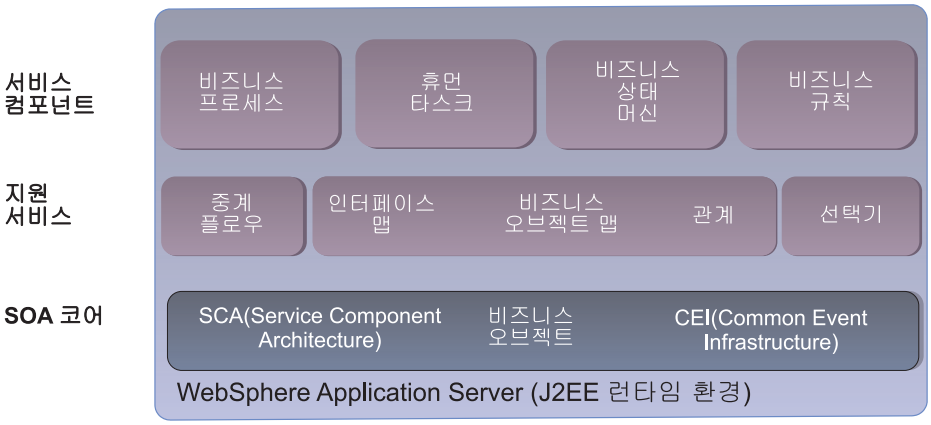


그림 2. WebSphere Process Server 컴포넌트 기반 프레임워크

WebSphere 환경에서, SCA 프레임워크는 WebSphere Application Server의 Java 2 Platform, Enterprise Edition (J2EE) 런타임 환경을 토대로 합니다. 전체 WebSphere Process Server 프레임워크는 SOA Core, 지원 서비스 및 서비스 컴포넌트로 구성됩

니다. 비즈니스 통합에서의 응용프로그램 통합 요구사항과 연결성에 보다 중점을 둔 이 전체 기능의 서브세트가 있는 동일한 프레임워크를 WebSphere Enterprise Service Bus에서 사용 가능합니다.

그림 3에서 명시한 대로, SCA 컴포넌트의 인터페이스는 다음 중 하나로 표시될 수 있습니다.

- Java 인터페이스
- WSDL 포트 유형(WSDL 2.0에서, 포트 유형을 인터페이스라고 함)

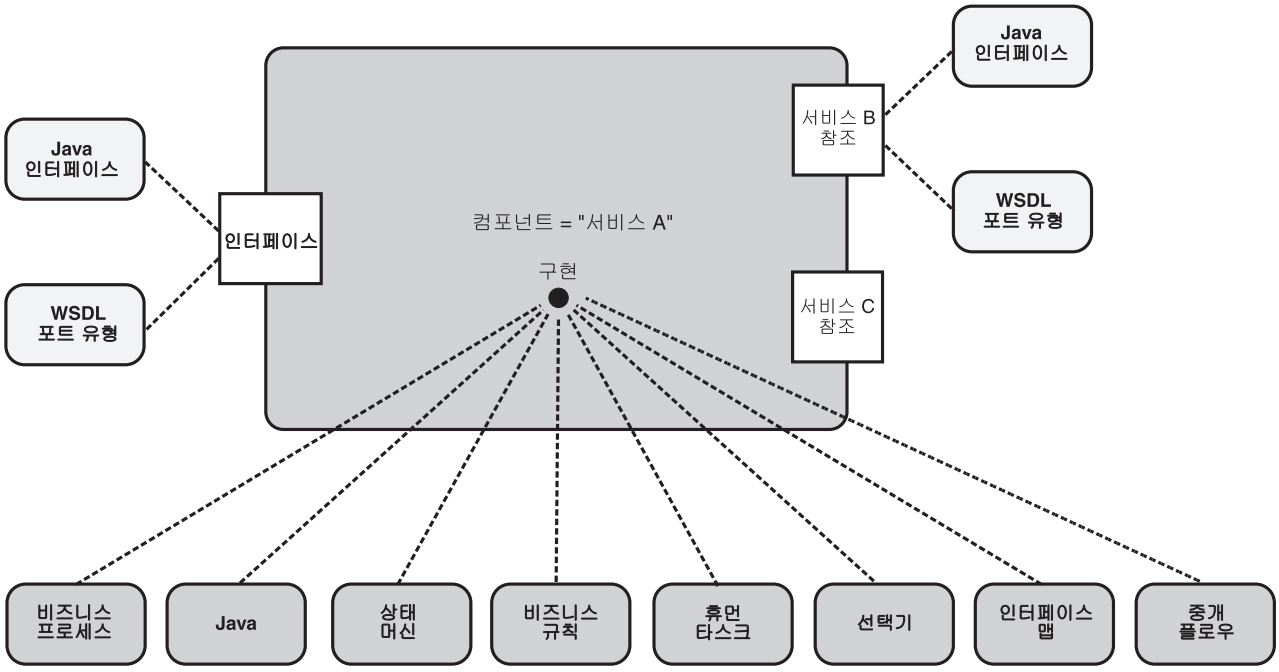


그림 3. WebSphere Process Server에서의 SCA

SCA 모듈은 참조 및 구현을 직접 링크하여 함께 연결되는 컴포넌트 그룹입니다. WebSphere Integration Developer에서, 각 SCA 모듈은 SCA 컴포넌트와 연관된 어셈블리 다이어그램을 지니며 SCA 컴포넌트와 이 컴포넌트에 대한 연결로 구성되는 통합된 비즈니스 응용프로그램을 표시합니다. 통합 개발자의 기본 책임 중 하나는 솔루션을 형성하는 컴포넌트를 연결하여 어셈블리 다이어그램을 작성하는 것입니다. WebSphere Integration Developer는 그래픽 어셈블리 편집자를 제공하여 이 타스크를 지원합니다. 어셈블리 다이어그램 작성 시, 통합 개발자는 다음 방법 중 하나로 작성을 수행할 수 있습니다.

- **하향식** 접근법은 구현을 작성하기 전에 컴포넌트, 인터페이스 및 상호작용을 정의합니다. 통합 개발자는 프로세스의 구조를 정의하고, 필요한 컴포넌트 및 해당 구현 유형을 식별한 후 구현 스켈레톤을 생성합니다.
- **상향식** 접근법은 기존 컴포넌트를 결합합니다. 이 경우, 통합 개발자는 단순히 기존 구현을 어셈블리 다이어그램으로 끌어서 놓기만 하면 됩니다.

고객이 재사용하고 결합하려는 기존의 서비스를 가진 경우 상향식 접근이 더 일반적으로 사용됩니다. 처음부터 새 비즈니스 오브젝트를 작성해야 하는 경우, 일반적으로 하향식 접근을 채택합니다.

SCA 프로그래밍 모델 : 기본

소프트웨어 *컴포넌트*의 개념은 SCA 프로그래밍 모델의 기초를 형성합니다. 언급한 대로, 컴포넌트는 로직을 구현하는 단위로 인터페이스를 통해 다른 컴포넌트에 로직을 사용할 수 있습니다. 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. 그림 4에 표시된 어셈블리 다이어그램에는 세 가지 컴포넌트 C1, C2 및 C3가 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.

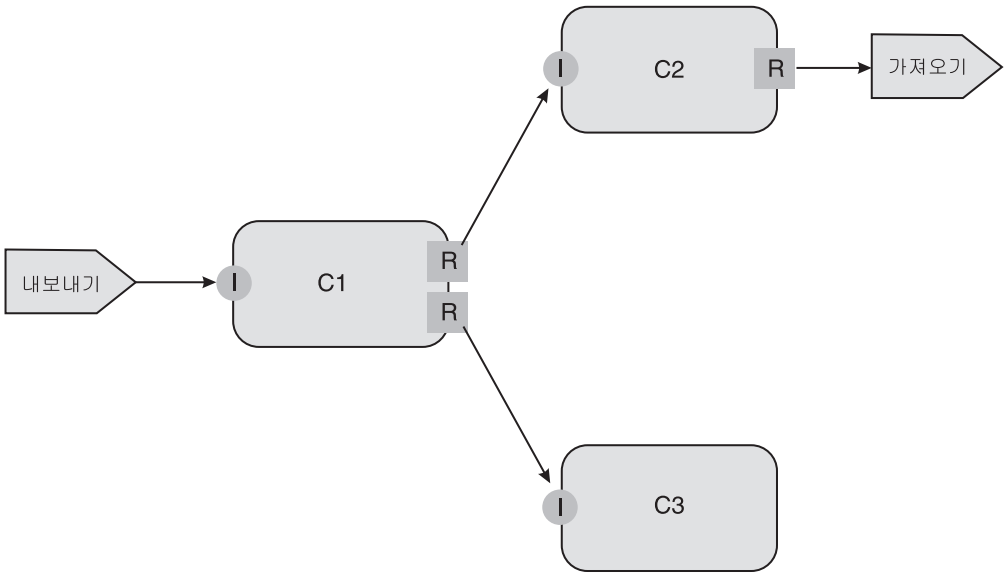


그림 4. 어셈블리 다이어그램

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 찾아보기를 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.sca입니다.

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. 이것이 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 호출과 유사하지만, 응답을 리턴하는 책임을 수신자에게 위임합니다. 응답이 준비되었을 때 수신자가 호출할 수 있는 특정 조작(콜백 조작)을 호출자가 나타낼 필요가 있습니다.

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

가져오기와 유사하게, 외부 응용프로그램에서 사용자 컴포넌트에 액세스해야 할 경우가 자주 발생하는데 이때, 액세스가 가능하도록 설정해야 합니다. 이는 사용자 로직을 ″외부″에 나타내는 특정 컴포넌트를 사용하여 수행됩니다. 이러한 컴포넌트를 *내보내기*라고 합니다. 이것은 동기 또는 비동기적으로 호출될 수 있습니다.

독립형 참조

WebSphere Process Server에서, SCA 서비스 모듈은 여러 다른 J2EE 서브모듈을 포함하는 J2EE EAR 파일로서 패키지됩니다. WAR 파일과 같은 J2EE 요소는 SCA 모

듈과 함께 패키지될 수 있습니다. JSP와 같은 비SCA 아티팩트는 SCA 서비스 모듈과 함께 패키지될 수도 있습니다. 이것은 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. 이것은 WebSphere Integration Developer에서 수행됩니다. 보통 솔루션 전개를 고려할 준비가 되면 QoS 규정자를 적용합니다. 자세한 정보는 서비스 규정자 참조의 품질을 참조하십시오.

비즈니스 프로세스

비즈니스 프로세스, 특히 BPEL 기반 비즈니스 프로세스는 SCA에서 서비스 컴포넌트의 토대를 형성합니다.

단순한 순서 승인 또는 복합 제조 프로세스에 관계없이, 엔터프라이즈는 항상 비즈니스 프로세스를 수행합니다. *비즈니스 프로세스*는 비즈니스 목적을 달성하기 위해 특정 순서에서 호출되는 비즈니스와 관련된 활동 세트입니다. 비즈니스 통합에서, 비즈니스 프로세스는 마크업 언어 종류를 사용하여 정의됩니다.

이들 비즈니스 프로세스는 비즈니스 상태 머신, 휴먼 타스크, 비즈니스 규칙 또는 데이터 맵과 같은 다른 서비스 컴포넌트나 기타 지원 서비스를 호출할 수 있습니다. 그리고 전개 시, 이러한 프로세스는 빠르게 완료되거나 장시간 실행될 수 있습니다. 경우에 따라서는 수년간 실행될 수도 있습니다.

J2EE에서 대부분의 컴포넌트와 같이, 비즈니스 프로세스는 컨테이너에서 실행합니다. IBM WebSphere 플랫폼에서, 이 특정 컨테이너를 Business Process Choreographer라고 합니다. WebSphere Process Server에서, Business Process Choreographer는 실행하는 비즈니스 프로세스와 휴먼 타스크에 관여합니다.

휴먼 타스크

휴먼 타스크는 사용자와 서비스 간의 상호작용을 돕는 컴포넌트입니다.

일부 휴먼 타스크는 사용자를 위한 작업을 나타냅니다. 이들 타스크는 사용자나 자동화된 서비스 중 하나로 시작될 수 있습니다. 휴먼 타스크는 수동 예외 처리 및 승인과 같이 사용자의 상호작용이 필요한 비즈니스 프로세스에서 활동을 구현하기 위해 사용될 수 있습니다. 다른 휴먼 타스크는 서비스를 호출하거나, 사용자 간의 협업을 조정하기 위해 사용될 수 있습니다. 그러나 타스크가 시작되는 방식에 관계없이 타스크가 지정된 그룹에 속한 사용자가 타스크와 연관된 작업을 수행합니다.

사용자의 디렉토리를 사용하여 런타임 시 결정된 역할 또는 그룹과 같은 기준을 지정하거나, 통계적으로 휴먼 타스크에 사용자가 지정됩니다. 또는 휴먼 타스크의 입력 데이터나 비즈니스 프로세스의 데이터가 사용되어 타스크에서 작업을 수행할 적합한 인력을 찾습니다.

비즈니스 통합 응용프로그램 빌드

*비즈니스 통합*은 비즈니스 내에서 또는 엔터프라이즈 세트 간에 응용프로그램, 데이터 및 프로세스를 통합함을 의미합니다. 또한 프로세스 통합을 위해 어셈블되는 일련의 응용프로그램에 대한 로직이 있으므로 통합은 프로세스 개발을 의미하기도 합니다. WebSphere Integration Developer는 비즈니스 통합 응용프로그램을 작성하기 위해 사용됩니다.

이 섹션에서는 비즈니스 통합 모듈의 개발 프로세스의 개요 정보를 제공합니다.

모듈 및 중개 모듈의 일반 개발 플로우는 다음과 같습니다.

1. WebSphere Integration Developer를 시작하고 작업공간을 여십시오.
2. 개발을 위해 비즈니스 통합 Perspective로 전환하십시오.
3. 다중 모듈 간에 공유되는 비즈니스 오브젝트와 인터페이스와 같은 아티팩트를 저장하려면 라이브러리를 작성하십시오.
4. 새 모듈이나 중개 모듈을 작성하십시오.

5. 예를 들어, 고객이나 주문 데이터와 같은 응용프로그램 데이터를 포함하기 위한 비즈니스 오브젝트를 작성하십시오.
6. 인터페이스를 작성하고 각 컴포넌트에 대한 인터페이스 조작을 정의하십시오. 인터페이스는 한 컴포넌트에서 다른 컴포넌트로 전달할 수 있는 데이터를 판별합니다.
7. 서비스 컴포넌트를 작성하고 구현하십시오.
8. 서비스 컴포넌트, 가져오기 및 내보내기를 어셈블리 다이어그램에 추가하여 모듈 어셈블리를 빌드하십시오. 컴포넌트를 함께 연결하십시오. 가져오기 및 내보내기를 프로토콜에 바인드하십시오.
9. 통합된 테스트 환경에서 모듈을 테스트하십시오.
10. 모듈을 WebSphere Process Server에 전개하십시오.
11. 테스트된 모듈을 저장소에 넣고 팀의 다른 사용자와 공유하십시오.

제 2 장 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 컴포넌트 및 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 섹션에서는 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 컴포넌트를 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. 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 섹션에 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 확약 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 확약 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 6 - 8단계의 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』섹션에 연결 구성 방법이 설명되어 있습니다.

컴포넌트 개발 예제

이 예에서는 단일 메소드인 CustomerInfo를 구현하는 동기 서비스 컴포넌트를 보여줍니다. 첫 번째 섹션은 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 섹션은 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 응용프로그램 서버를 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 =
      (CustomerInfo)serviceManager.locateService("customerInfo");
         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);
```

분리 모듈 및 대상 개요

모듈을 개발할 때, 여러 개의 모듈이 사용할 수 있는 서비스를 구별할 수 있습니다. 이러한 방법으로 서비스를 활용하면 개발 주기와 비용을 최소화할 수 있습니다. 여러 모듈에 의해서 사용되는 서비스가 있다면, 호출 모듈을 대상에서 분리하십시오. 그럴 경우에 대상이 업그레이드되면, 새로운 서비스로의 전환이 호출 모듈에 대해서 투명하도록 합니다. 이 주제는 단순 호출 모델과 분리 호출 모델을 대조하여 설명하며 분리를 유용하게 사용할 수 있는 예제를 제공합니다. 특정 예제를 설명하는 경우, 이 예제가 대상에서 모듈을 분리하는 유일한 방법은 아닙니다.

단순 호출 모델

모듈을 개발할 때, 다른 모듈에 있는 서비스를 사용하는 경우도 있습니다. 이렇게 하려면 해당 서비스를 모듈로 가져온 다음 서비스를 호출하면 됩니다. 가져온 서비스는 관리 콘솔의 서비스를 바인딩하거나 WebSphere Integration Developer에서 다른 모듈이 내보낸 서비스로 『연결』됩니다. 단순 호출 모델은 이 모델을 설명합니다.

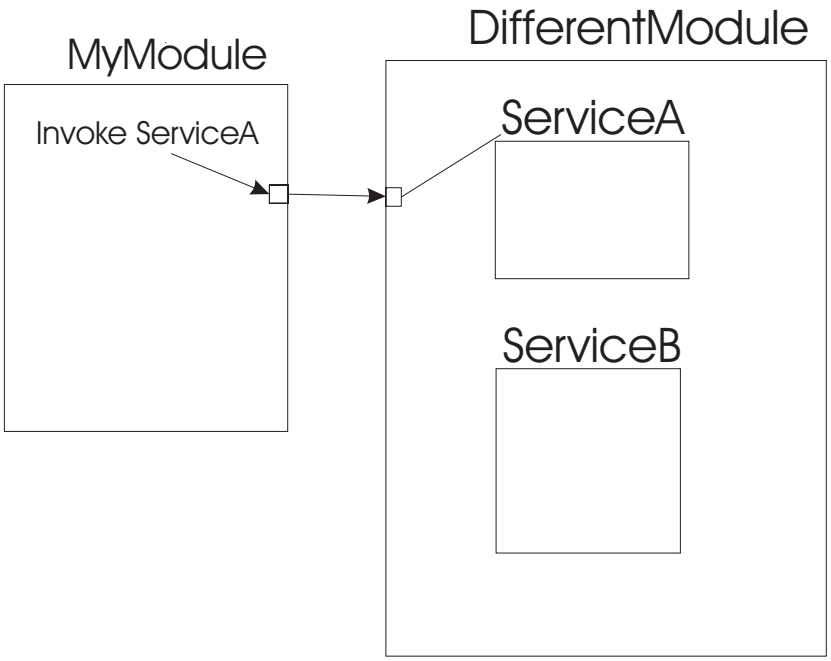


그림 5. 단순 호출 모델

분리 호출 모델

호출 모듈을 호출 대상으로부터 분리하여 모듈 호출을 중지하지 않고 호출의 대상을 변경할 수 있습니다. 이렇게 하면 모듈을 변경하지는 않지만 다운스트림 대상을 변경하기 때문에 대상을 변경하는 경우, 모듈이 처리를 계속할 수 있도록 합니다. 응용프로그램 분리 예는 모듈 호출의 상태에 영향을 받지 않고 분리가 어떻게 대상을 변경하는지에 대한 예를 보여줍니다.

응용프로그램 분리 예

단순 호출 모델을 사용하면 같은 서비스를 호출하는 여러 개의 모듈은 여러 개의 응용프로그램이 하나의 서비스를 호출하는 경우와 비슷하게 됩니다. ModA, ModB 및 ModC는 모두 CalculateFinalCost를 호출합니다.

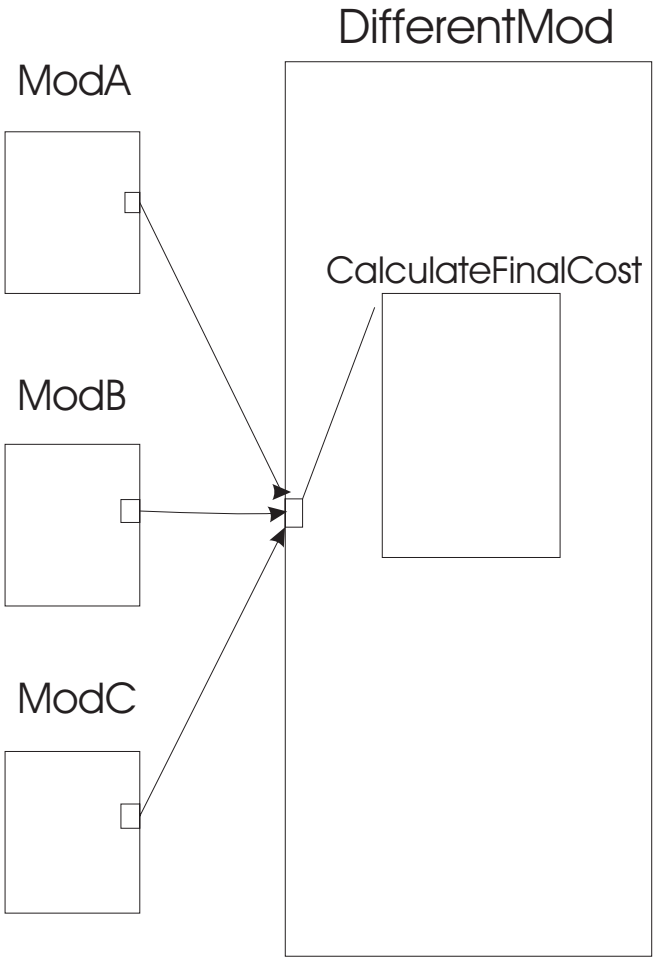


그림 6. 여러 개의 응용프로그램이 하나의 서비스를 호출

CalculateFinalCost에서 제공되는 서비스는 이 서비스를 사용하는 모든 모듈에서 새로운 비용이 반영되도록 갱신되어야 합니다. 개발 팀은 이 변경사항을 통합하기 위해 새로운 서비스인 UpdatedCalculateFinal을 테스트하고 빌드합니다. 이제 새로운 서비스를 프로덕션으로 가져올 준비가 되었습니다. 분리를 사용하지 않는 경우, UpdateCalculateFinal을 호출하려면 CalculateFinalCost를 호출하는 모든 모듈을 갱신해야 합니다. 분리를 사용하는 경우에는 버퍼 모듈을 대상으로 연결하는 바인딩을 변경하기만 하면 됩니다.

주: 이렇게 서비스를 변경하면 필요할 수 있는 다른 모듈에 원래의 서비스를 제공하는 것을 계속할 수 있습니다.

분리를 사용하는 경우, 응용프로그램과 대상(UpdateCalculateFinal을 호출하는 분리 호출 모델 참조) 사이에 버퍼 모듈을 작성합니다.

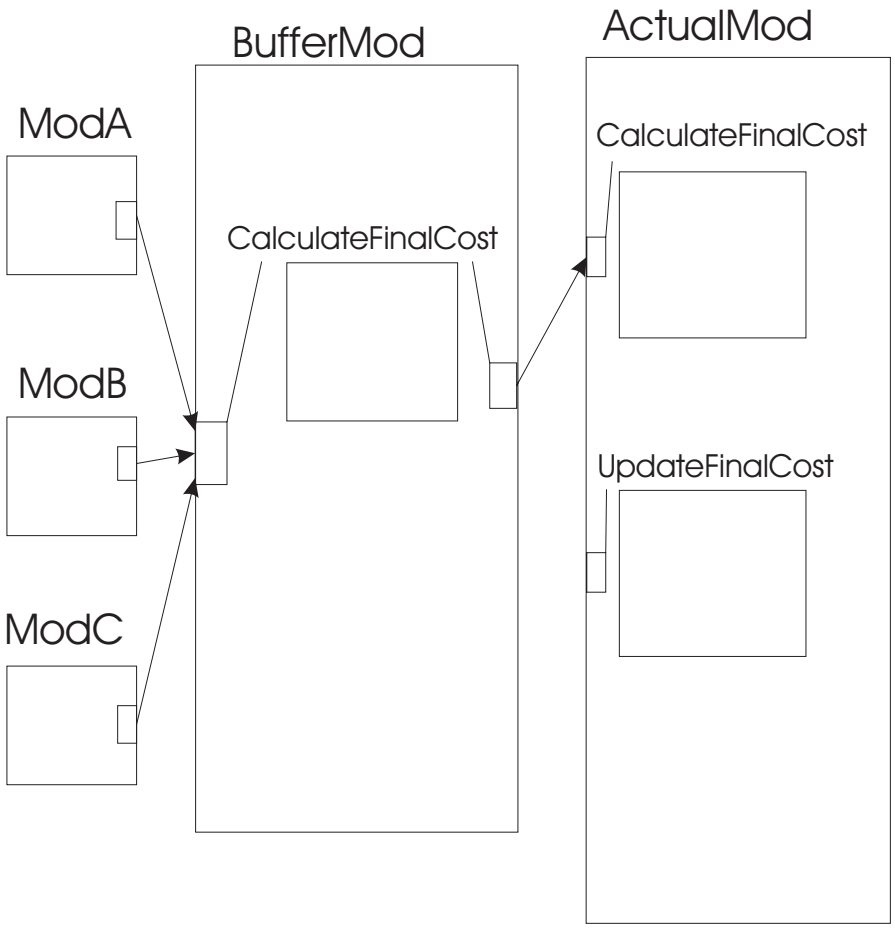


그림 7. UpdateCalculateFinal을 호출하는 분리 호출 모델

호출 모듈이 변경되지 않는 이러한 모델의 경우, 가져온 버퍼 모듈에서 대상 (UpdatedCalculateFinal을 호출하는 분리 호출 모델 참조)으로 변경해야 합니다.

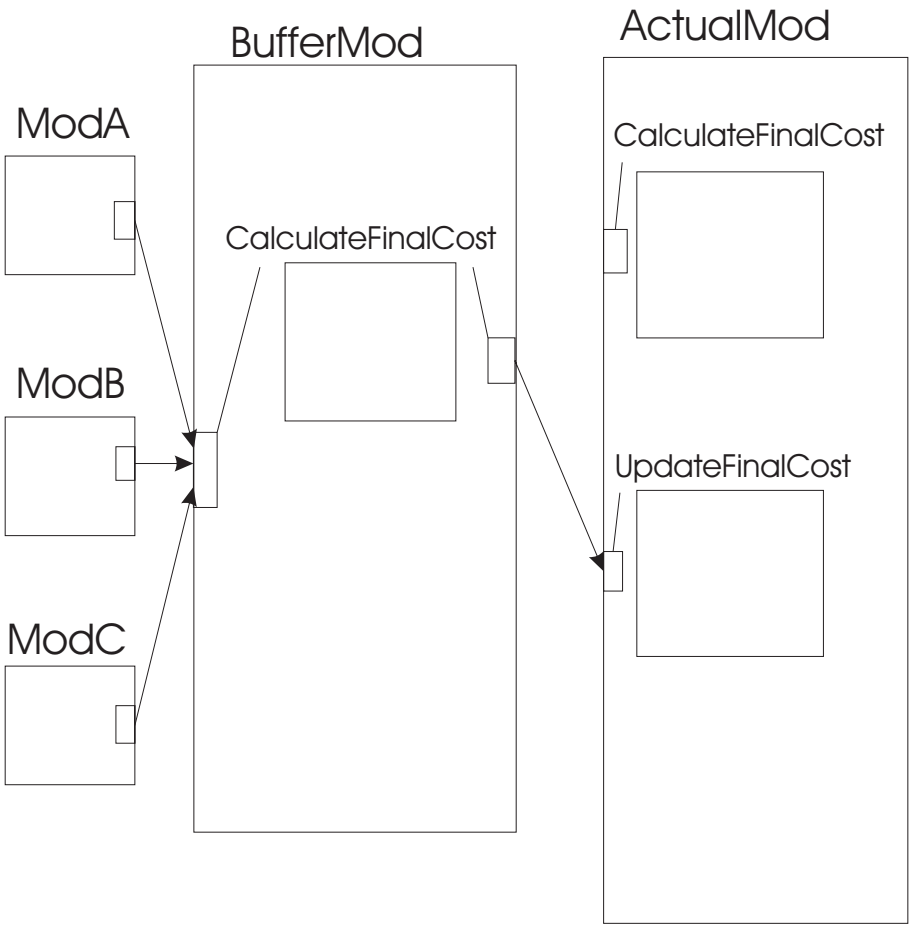


그림 8. UpdatedCalculateFinal을 호출하는 분리 호출 모델

버퍼 모듈이 동시에 대상을 호출하는 경우, 버퍼 모듈을 다시 시작할 때(중개 모듈 또는 비즈니스 모듈의 서비스) 새 대상으로부터 원래 응용프로그램으로 그 결과가 리턴됩니다. 버퍼 모듈이 대상을 비동기적으로 호출하는 경우, 다음 호출의 새 대상으로부터 원래 응용프로그램으로 그 결과가 리턴됩니다.

HTTP 바인딩

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

또한 SCA 런타임 환경의 네트워크가 기존 HTTP 하부 구조와 통신할 수 있습니다.

HTTP 바인딩에서는 여러 가지 HTTP 기능이 공개되어 있습니다.

- 메시지는 HTTP 형식 및 메시지 헤더 정보를 보존하는 방식으로 중개 컴포넌트에 표시됩니다. 이러한 방식은 HTTP 응용프로그램 프로그래머, 사용자 및 관리자에게 더 익숙한 보기를 제공합니다.

- 기존 데이터 바인딩 프레임워크가 HTTP 규칙에 부합하도록 확장되며 SCA 메시지와 HTTP 메시지 헤더 및 본문 간에 맵핑을 제공합니다.
- 여러 공통 HTTP 기능을 지원하도록 가져오기 및 내보내기를 구성할 수 있습니다.
- HTTP 가져오기 또는 내보내기가 들어 있는 SCA 모듈을 설치하는 경우 HTTP에 연결할 수 있도록 런타임 환경이 자동으로 적절히 구성됩니다.

HTTP 가져오기 및 내보내기 작성에 대한 자세한 설명은 Information Center의 **WebSphere Integration Developer > 통합 응용프로그램 개발 > HTTP 데이터 바인딩**에 있습니다.

관련 태스크


HTTP 바인딩 표시

응용프로그램을 전개한 후 HTTP 바인딩이 올바른지 확인하기 위해 이를 검사할 수 있습니다.


HTTP 내보내기 바인딩 변경

관리 콘솔을 사용하면 원래 소스를 변경한 다음 응용프로그램을 다시 전개하지 않아도 HTTP 내보내기 바인딩의 구성을 변경할 수 있습니다.


HTTP 가져오기 바인딩 변경

관리 콘솔을 사용하면 원래 소스를 변경한 다음 응용프로그램을 다시 전개하지 않아도 HTTP 가져오기 바인딩의 구성을 변경할 수 있습니다.

제 3 장 프로그래밍 안내서 및 기술

이 섹션에는 프로그래밍 안내서 및 예제가 있습니다.

다음 링크에는 다양한 컴포넌트, 응용프로그램 및 비즈니스 통합 솔루션을 프로그래밍하는 정보 안내서가 들어 있습니다.

- 비즈니스 오브젝트 프로그래밍: 스키마 향상 및 산업 스키마 지원
- 비즈니스 규칙 관리 프로그래밍 안내서
- XMS 응용프로그램 개발
- 비즈니스 프로세스 및 타스크용 클라이언트 응용프로그램 개발

중요사항: WebSphere Process Server 및 WebSphere Enterprise Service Bus에서 지원하는 API(application programming interface) 및 SPI(system programming interfaces)의 세부사항에 대해서는 InfoCenter의 참조 섹션을 참조하십시오.

제 4 장 생성된 서비스 컴포넌트 아키텍처 구현 대체

때로 Java 코드와 서비스 데이터 오브젝트(SDO) 간에 시스템이 작성하는 변환이 사용자의 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 사용하면 기본 SCA(Service Component Architecture) 클래스 구현을 사용자 고유의 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. 사용자만의 Java 클래스를 정의한 경우 사용자만의 맵을 사용하는 방법을 고려하십시오. 이 프로시저를 사용하여 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 바꿀 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.

관련 개념

41 페이지의 제 7 장『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.

관련 참조


Java 대 XML 변환

시스템은 사전 정의된 규칙을 사용하여 Java 유형에 기반한 XML을 생성합니다.


genMapper 명령행 유틸리티

genMapper 명령을 사용하여 Java 인터페이스에 대한 SCA(Service Component Architecture) 참조로 브릿지하는 컴포넌트를 생성하십시오.

제 5 장 서비스 데이터 오브젝트와 Java 간 변환 대체

때로 서비스 데이터 오브젝트(SDO)와 Java 유형 오브젝트 간에 시스템이 작성하는 변환이 사용자 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 사용하면 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. 사용자만의 Java 클래스를 정의한 경우 사용자만의 맵을 사용하는 방법을 고려하십시오. 이 프로시저를 사용하여 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 바꿀 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.

관련 개념

41 페이지의 제 7 장『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.

관련 참조


Java 대 XML 변환

시스템은 사전 정의된 규칙을 사용하여 Java 유형에 기반한 XML을 생성합니다.


genMapper 명령행 유틸리티

genMapper 명령을 사용하여 Java 인터페이스에 대한 SCA(Service Component Architecture) 참조로 브릿지하는 컴포넌트를 생성하십시오.

제 6 장 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 또 다른 컴포넌트로 호출 실행 시, 응답은 다른 컨텍스트를 포함합니다. 서비스 컴포넌트는 수신 컨텍스트를 가지지만, 다른 서비스 호출 시 다른 서비스가 원래의 전송 컨텍스트를 겹쳐씁니다. 응답 컨텍스트는 새 컨텍스트가 됩니다.

컨텍스트 서비스 전파가 단방향이면, 원래의 컨텍스트는 동일하게 남습니다.

컨텍스트 서비스의 라이프 사이클은 호출과 연관됩니다. 요청은 연관된 컨텍스트와 해당 특정 요청의 처리에 컨텍스트가 속한 라이프 사이클을 갖습니다. 해당 요청 처리가 완료되면, 해당 컨텍스트의 라이프 사이클이 끝납니다.

단기 실행 BPEL(Business Process Execution Language) 프로세스의 경우, 응답 컨텍스트가 요청 컨텍스트를 겹쳐씁니다. 첫째 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.

예

예를 들어, 프로토콜 헤더를 포함하는 컨텍스트는 SOAP 웹 서비스에서 BPEL로 들어오는 요청으로 시작하는 호출 경로를 통해 전파됩니다. BPEL이 요청을 처리하고, BPEL로부터 순차적으로 웹 서비스 바인딩 아웃바운드로 호출이 일어난 후, 또 다른 웹 서비스 바인딩 아웃바운드로 호출합니다. SOAP 웹 서비스의 요청은 컨텍스트 서비스를 사용하여 프로토콜 헤더로 전달합니다. 인바운드 요청에서 컨텍스트 서비스를 얻어 프로토콜 헤더 아웃바운드를 푸시합니다.

이 예제에서 BPEL 대신 또 다른 SCA 컴포넌트를 이용하여 유사한 동작을 볼 수 있습니다.

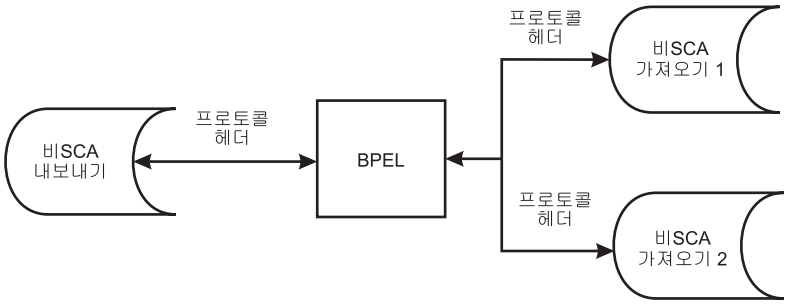


그림 9. 프로토콜 헤더를 포함하는 컨텍스트 전파

이것은 코드 예제입니다.

```
//Import the necessary classes;
import com.ibm.bpm.context.ContextService;
import com.ibm.websphere.sca.ServiceManager;
import com.ibm.bpm.context.cobo.ContextObject;
import com.ibm.bpm.context.cobo.ContextObjectFactory;
import com.ibm.bpm.context.cobo.HeaderInfoType;
import com.ibm.bpm.context.cobo.UserDefinedContextType;

//Locate ContextService;
ContextService contextService =
(ContextService)ServiceManager.INSTANCE.locateService("com/ibm/bpm/context/ContextService");

// Get header info
HeaderInfo headerInfo = contextService.getHeaderInfo();
// Get user defined context in current execution context
UserDefinedContextType userDefinedContext = contextService.getUserDefinedContext();
if(userDefinedContext == null){ // create a new context if context is null
userDefinedContext = ContextObjectFactory.eINSTANCE.createUserDefinedContextType()
}

// Do some modification to header info and userDefinedContext

// Set user defined context back to the current execution context.
contextService.setUserDefinedContext(userDefinedContext);

// Set header info back to the current execution context.
contextService.setHeaderInfo(headerInfo);
```

주: 중개 플로우 컴포넌트에서, `ContextService` API는 사용되지 않아야 합니다. 컨텍스트를 액세스하려면 SMO 프로그래밍 모델을 사용하십시오.

컨텍스트 서비스에는 바인딩 동작을 지시하는 테이블과 구성 가능한 규칙이 있습니다. 자세한 정보는 참조 섹션에서 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.

제 7 장 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 3. WSDL 유형을 Java 클래스로 변환

인터페이스	기본 구체적(**concrete**) 클래스
Collection	HashSet
Map	HashMap
List	ArrayList
Set	HashSet

관련 태스크

35 페이지의 제 4 장『생성된 서비스 컴포넌트 아키텍처 구현 대체』
때로 Java 코드와 서비스 데이터 오브젝트(SDO) 간에 시스템이 작성하는 변환이 사용자의 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 사용하면 기본 SCA(Service Component Architecture) 클래스 구현을 사용자 고유의 구현으로 바꿀 수 있습니다.

37 페이지의 제 5 장『서비스 데이터 오브젝트와 Java 간 변환 대체』
때로 서비스 데이터 오브젝트(SDO)와 Java 유형 오브젝트 간에 시스템이 작성하는 변환이 사용자 요구와 일치하지 않을 수도 있습니다. 이 프로시저를 사용하면 기본 구현을 사용자만의 구현으로 바꿀 수 있습니다.

관련 참조

Java 대 XML 변환
시스템은 사전 정의된 규칙을 사용하여 Java 유형에 기반한 XML을 생성합니다.

genMapper 명령행 유틸리티
genMapper 명령을 사용하여 Java 인터페이스에 대한 SCA(Service Component Architecture) 참조로 브릿지하는 컴포넌트를 생성하십시오.

제 8 장 비즈니스 오브젝트: 스키마 향상 및 산업 스키마 지원

서비스 데이터 오브젝트(SDO) 프레임워크는 WebSphere Process Server가 사용하는 비즈니스 오브젝트 데이터를 위한 기초를 제공합니다.

이 안내서는 일부 기능에 대한 스키마 구성 처리에서 문제점 영역에 대한 정보를 제공합니다. 비즈니스 오브젝트가 정의되는 방법, 비즈니스 오브젝트 개발 지침 및 비즈니스 오브젝트 프로그래밍 API 사용 방법에 대해서는 아래에 있는 "관련 정보" 섹션의 기사를 참조하십시오.

관련 정보

Web Services Description Language(WSDL) 1.1

서비스 데이터 오브젝트 소개

WebSphere Process Server의 비즈니스 오브젝트 점검

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. 이러한 ″단일화″는 해당 구조의 모든 하위 구조를 동일한 레벨에 놓습니다. 이로 인해 단일화된 데이터에서 구조를 추출한 SDO에 중복 네이밍 문제가 발생할 수 있습니다. XSD에서 그룹을 단일화하지 않는 경우 여전히 서로 다른 상위 구조에 중복되는 항목이 각각 들어 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. 다음 XSD를 예로 들 수 있습니다.

```
<?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. 발견된 모든 중복 특성 이름에는 EMF 이름 지정 규칙에 따라 사용되지 않은 다음 번 숫자가 추가됩니다. 다음과 같은 XSD를 예로 들 수 있습니다.

```
<?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 특성 이름으로 **high**, **high1**, **high2**와 마찬가지로 첫 번째 width에서 시작하여 XSD에서 아래 방향으로 순서대로 이름이 지정됩니다.

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"));
```

관련 개념

67 페이지의 제 9 장『비즈니스 오브젝트의 배열』

요소가 데이터 인스턴스를 두 개 이상 포함할 수 있도록 비즈니스 오브젝트에서 요소의 배열을 정의할 수 있습니다.

관련 태스크

74 페이지의『모델 그룹의 비즈니스 오브젝트 사용』

모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

동일하게 이름 지정된 특성 구별

동일한 네임 스페이스를 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 기타 구조의 하위 구조로 중복 글로벌 구조를 여전히 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.

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 순서가 생성되게 합니다.

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 섹션의 나머지 부분에 표시된 일반 정보에서는 기타 순서 조건에서 작업하는 방법에 대해 설명하며, 이는 여전히 <any/>에도 적용됩니다.

관련 개념

67 페이지의 제 9 장『비즈니스 오브젝트의 배열』
요소가 데이터 인스턴스를 두 개 이상 포함할 수 있도록 비즈니스 오브젝트에서 요소의 배열을 정의할 수 있습니다.

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 알아야 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 카디널리티 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 카디널리티가 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
```

관련 태스크

74 페이지의『모델 그룹의 비즈니스 오브젝트 사용』

모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

단순 유형에 대해 AnySimpleType 사용

AnySimpleType은 SDO API에서 기타 다른 단순 유형(문자열, 정수, 부울 등)과 유사하게 처리됩니다.

anySimpleType 과 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 설정되어 있는 원래 데이터 유형 사용을 완화시키지 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 중에도 xsi:type에 의해 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 anySimpleType

에 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 디자인 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 있

습니다. 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 디자인 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.

관련 태스크

72 페이지의『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』
상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

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"
```

관련 태스크

72 페이지의『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』

상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

Any에 있는 데이터에 대해 유효한 맵핑

<any/> 태그는 이름/값 쌍 세트입니다. 디자인 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.

관련 태스크

72 페이지의『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』

상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

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
```

관련 태스크

72 페이지의『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』

상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

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 가져

오기를 수행할 수 있으며 가져오기가 해결됩니다. 이름을 알 수 없는 경우 위의 코드를 사용하여 값을 반복시키고 차례로 값에 액세스할 수 있습니다. 다음 코드 예제는 이 작업의 수행 방법을 보여 줍니다.

```
<?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"
```

관련 태스크

72 페이지의『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』

상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

anyAttribute에 있는 데이터에 대해 유효한 맵핑

AnyAttribute 태그는 any 태그와 비슷하며 이름/값 쌍 세트로 구성됩니다. 따라서 anyAttribute에 유효한 맵핑은 다른 anyAttribute뿐입니다.

개별적으로, anyAttribute 데이터에 포함된 값은 모든 단순 유형 맵핑 규칙을 따르는 기본 단순 유형입니다.

제 9 장 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 분리된 데이터 또는 고정 너비를 사용하여 목록 색인에 배열의 데이터 항목을 채울 수 있습니다. 또한 올바르게 형식화된 데이터를 포함하는 일반 텍스트 파일을 사용하여 이 타스크를 수행할 수 있습니다.

관련 개념

51 페이지의『순서 오브젝트를 사용하여 데이터 순서 설정』
일부 XSD를 정의하는 방법으로 인해 XML에서 발생하는 데이터 순서에 특수한 중요성이 부여됩니다.

46 페이지의『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.

제 10 장 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
- 와일드 카드 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 나타납니다.

```
<?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

종래의 "상의하달" 방법을 사용하여 비즈니스 오브젝트 데이터를 설정한 경우, 하위의 하위 오브젝트에 데이터를 설정하기 전에 하위 오브젝트 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.

관련 태스크

『중첩 비즈니스 오브젝트의 다중 인스턴스 작성』
setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 다중 인스턴스를 작성하십시오.

72 페이지의 『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』
상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

74 페이지의 『모델 그룹의 비즈니스 오브젝트 사용』
모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

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 나타내는 경우, 해당 요소가 널이 아니면(데이터를 포함함) 사용됩니다. 널이면 요소가 작성되고 사용됩니다. 목록의 색인이 목록의 *크기*보다 더 큰 값이면 새 값이 작성되고 추가됩니다. 다음 예제 코드는 *크기*가 2인 목록에서 어떤 일이 일어나는지를 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 크기가 4가 아니거나 maxOccurs에 지정된 크기보다 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");
```

관련 태스크

69 페이지의『중첩 비즈니스 오브젝트의 단일 인스턴스』
setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 단일 인스턴스를 작성하십시오.

『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』
상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

74 페이지의『모델 그룹의 비즈니스 오브젝트 사용』
모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

와일드 카드에 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 와일드 카드 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.

관련 개념

64 페이지의 『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.

64 페이지의 『AnyAttribute를 사용하여 복합 유형에 대한 글로벌 속성 설정』
<anyAttribute/> 태그를 사용하면 임의의 수의 글로벌 속성을 복합 유형에 설정할 수 있습니다.

62 페이지의 『Any 값 가져오기/설정 방법』
Any 필드에 설정된 데이터 이름을 알고 있는 경우 기타 요소 값과 동일한 방법으로 데이터 가져오기를 수행할 수 있습니다.

65 페이지의 『anyAttribute 값 가져오기/설정 방법』
<anyAttribute/> 값 설정은 <any/>를 설정하는 것과 동일한 방법으로 수행되지만 글로벌 요소 대신 글로벌 속성을 사용합니다.

61 페이지의 『Any를 사용하여 복합 유형에 대한 글로벌 요소 설정』
<any/> 태그를 사용하여 복합 유형에 글로벌 요소를 설정할 수 있습니다.

관련 태스크

69 페이지의 『중첩 비즈니스 오브젝트의 단일 인스턴스』
`setWithCreate` 기능을 사용하여 중첩 비즈니스 오브젝트의 단일 인스턴스를 작성하십시오.

70 페이지의 『중첩 비즈니스 오브젝트의 다중 인스턴스 작성』
`setWithCreate` 기능을 사용하여 중첩 비즈니스 오브젝트의 다중 인스턴스를 작성하십시오.

74 페이지의 『모델 그룹의 비즈니스 오브젝트 사용』
모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

모델 그룹의 비즈니스 오브젝트 사용

모델 그룹의 일부인 중첩 비즈니스 오브젝트에 대한 작업 시 그룹 경로 패턴을 모델링해야 합니다.

이 태스크 정보

모델 그룹은 상위 비즈니스 오브젝트에서 비즈니스 오브젝트를 작성하기 위해 사용할 수 있는 xsd:choice 태그를 사용합니다. 그러나 Eclipse Modeling Framework(EMF)는 예외를 생성할 수 있는 네이밍 충돌을 야기할 수 있습니다. 다음 예제 코드는 이 충돌이 어떻게 일어나는지에 대해 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>
```

″child1″ 및 ″child2″라는 요소의 다중 인스턴스가 있을 수 있다는 점에 유의하십시오.

모델 그룹에 대한 서비스 데이터 오브젝트(SDO) 경로 패턴을 사용하여 이 충돌을 해결해야 합니다.

결과

아래 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");
```

관련 개념

46 페이지의 『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.

55 페이지의 『모델 그룹 배열에 대한 작업 방법』
모델 그룹에 maxOccurs > 1인 값이 있는 경우 모델 그룹 배열이 작성됩니다.

관련 태스크

69 페이지의 『중첩 비즈니스 오브젝트의 단일 인스턴스』
setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 단일 인스턴스를 작성하십시오.

70 페이지의 『중첩 비즈니스 오브젝트의 다중 인스턴스 작성』
setWithCreate 기능을 사용하여 중첩 비즈니스 오브젝트의 다중 인스턴스를 작성하십시오.

72 페이지의 『와일드 카드에 정의된 중첩 비즈니스 오브젝트 사용』
상위 오브젝트에서 유형 xsd:any를 지정하면 하위 오브젝트가 이미 있는 경우에만 하위 오브젝트를 지정할 수 있습니다.

제 11 장 XML 문서 유효성 검증

XML 문서 및 비즈니스 오브젝트는 유효성 검증 서비스를 사용하여 유효성이 검증될 수 있습니다.

게다가 기타 서비스는 명백한 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:
```

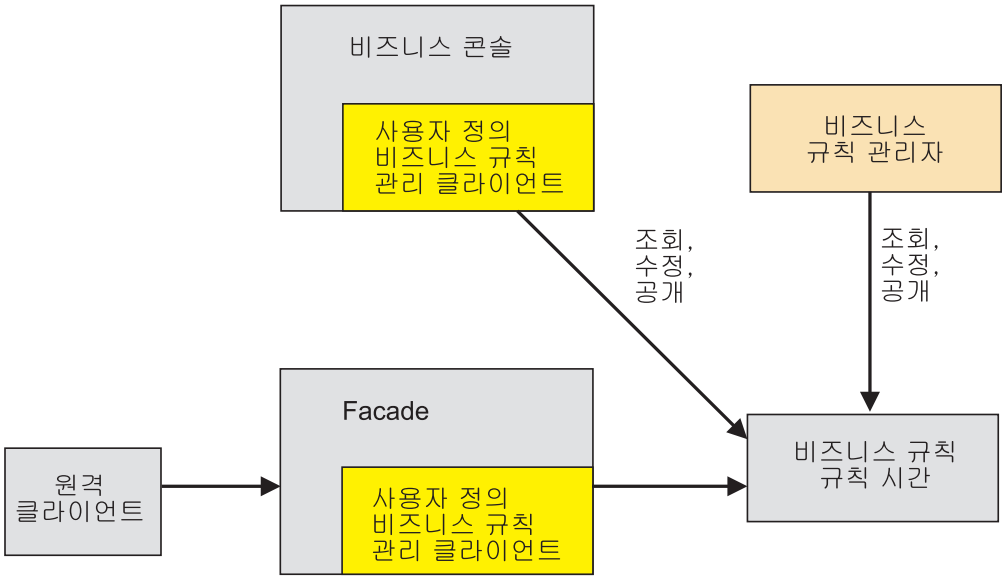
유효성 검증 서비스에 대한 자세한 정보는 참조 섹션에 생성된 API 및 SPI 문서에서 BOInstanceValidator 인터페이스를 참조하십시오.

제 12 장 비즈니스 규칙 관리

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 섹션과 부록으로 구성됩니다. 첫째 섹션에서는 프로그래밍 모델과 여러 클래스 사용 방법을 설명합니다. 클래스 간의 관계를 표시하기 위해 클래스 다이어그램을 제공합니다. 두 번째 섹션에서는 비즈니스 규칙 그룹 검색, 새 규칙 대상 스케줄 및 규칙 세트나 결정 테이블 수정과 같은 조치를 수행하기 위해 클래스 사용에 대한 예제를 제공합니다. 부록에는 일반 조작을 단순화시키기 위해 예제에서 사용된 추가 클래스가 있으며, 와일드 카드를 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 들어 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 장래 유지보수의 용이성을 위해서만이 아니라 일반 사용자의 일관된 프로그래밍 모델을 위해서도 중요하게 고려되었습니다. 이 모델 공유는 데스크탑 도구화 및 런타임 실행 및 작성의 필요성 간의 절충을 필요로 했습니다 -- 모두 각 환경을 충족시키는 명확한 요구사항 세트를 가지며 이들 요구사항은 가끔 서로 충돌했습니다. 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 결정 테이블의 템플리트를 이용하여 정의된 항목만으로 제한됩니다. 새 규칙 세트 및 결정 테이블의 작성은 기존 규칙 세트나 결정 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(예를 들어, 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 결정 테이블을 포함하는 루트 오브젝트로 고려될 수 있습니다.

규칙 세트 및 결정 테이블은 연관된 비즈니스 규칙 그룹을 통해서만 접근될 수 있습니다. 비즈니스 규칙 그룹에 대한 정보를 검색하고 규칙 세트 및 결정 테이블에 접근하기 위해 클래스에 대한 메소드가 제공됩니다. 메소드를 통해 다음 정보가 검색될 수 있습니다.

- 대상 네임 스페이스
- 비즈니스 규칙 그룹의 이름
- 표시 이름

- 이름/표시 이름 동기화
- 설명
- 날짜를 UTC 형식으로 표시할 지, 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 결정 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 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 초래합니다. 그리고 isShell 메소드 사용 시, 지원되는 버전인지 여부를 결정하기 위해 BusinessRuleGroup의 유형이 BusinessRuleGroupShell의 인스턴스인지 여부를 검사합니다.

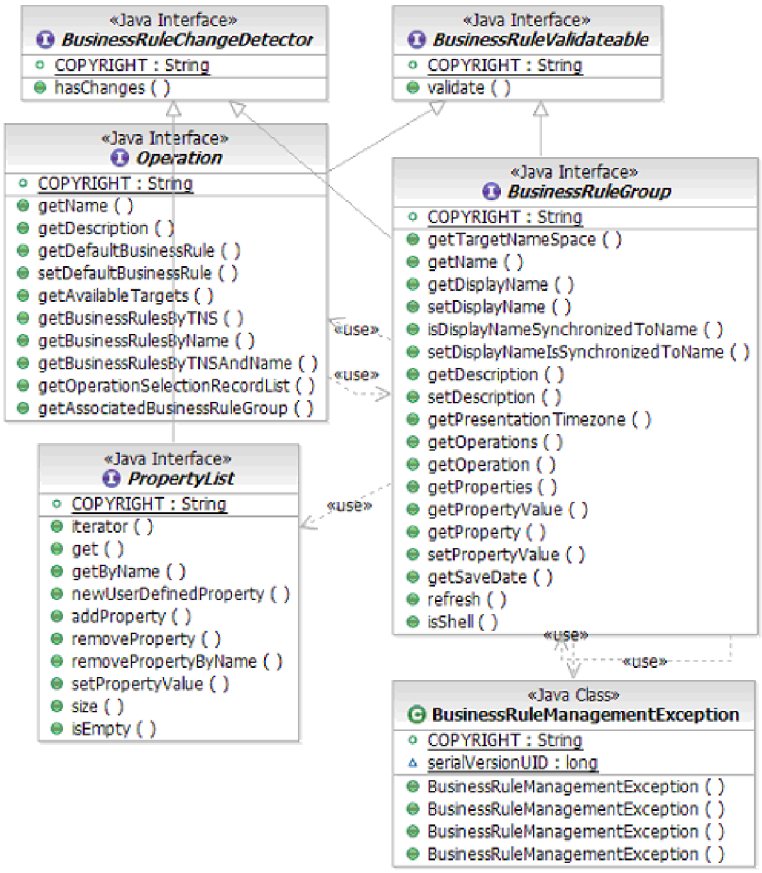


그림 10. BusinessRuleGroup 및 관련 클래스의 클래스 다이어그램

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 런타임에 규칙 세트나 결정 테이블에서도 액세스될 수 있습니다. 이것은 비즈니스 규칙 그룹에서 단일 값을 설정하여 비즈니스 규칙 그룹의 결정 테이블이나 다중 규칙 세트 내에서 사용될 수 있게 합니다. 비즈니스 규칙 그룹에 정의된 특성만이 둘러싼 규칙 세트와 결정 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, 네임 스페이스 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.

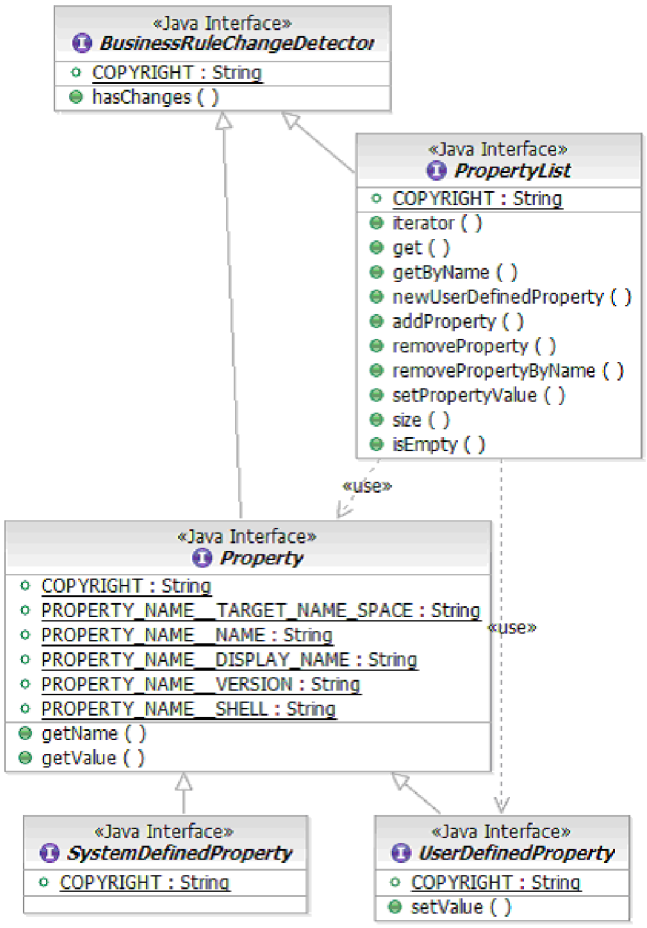


그림 11. 특성 및 관련 클래스의 클래스 다이어그램

조작

조작은 수정하려는 각 규칙 세트 및 결정 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 결정 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 예정되어야 할 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 결정 테이블은 물론 기본값 비즈니스 규칙 대상과 예정된 비즈니스 규칙 대상을 포함합니다. 예정되지 않은 규칙 세트나 결정 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 결정 테이블은 다른 조작의 규칙 세트와 결정 테이블과 달리 고유합니다.

사용 가능한 대상 목록에서 다른 규칙 세트와 결정 테이블 모두가 OperationSelectionRecord의 작성을 통해 활성화되도록 예정될 수 있습니다. 사용 가능한 대상 목록의 특정 규칙 세트나 결정 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. 우수 사례로서, 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 결정 테이블이나 규칙 세트를 검색하는 것 이외에, Operation 클래스는 이름이나 대상 네임 스페이스 특성 값으로 비즈니스 규칙 대상을 검색하도록 허용합니다. Operation 클래스의 메소드를 통해, 해당 조작의 사용 가능한 대상에 표시된 규칙 세트와 결정 테이블이 조회될 수 있습니다. 일치하는 이름과 대상 네임 스페이스를 가질 수 있는 규칙 세트와 결정 테이블은 다른 조작이 사용 가능한 대상 목록의 일부이지만 결과 세트에 포함되지 않습니다. 편의상, getBusinessRulesByName, getBusinessRulesByTNS 및 getBusinessRulesByTNSAndName 메소드는 특정 규칙 세트와 결정 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 네임 스페이스에 의해 사용 가능한 모든 대상 목록에서 규칙 세트나 결정 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

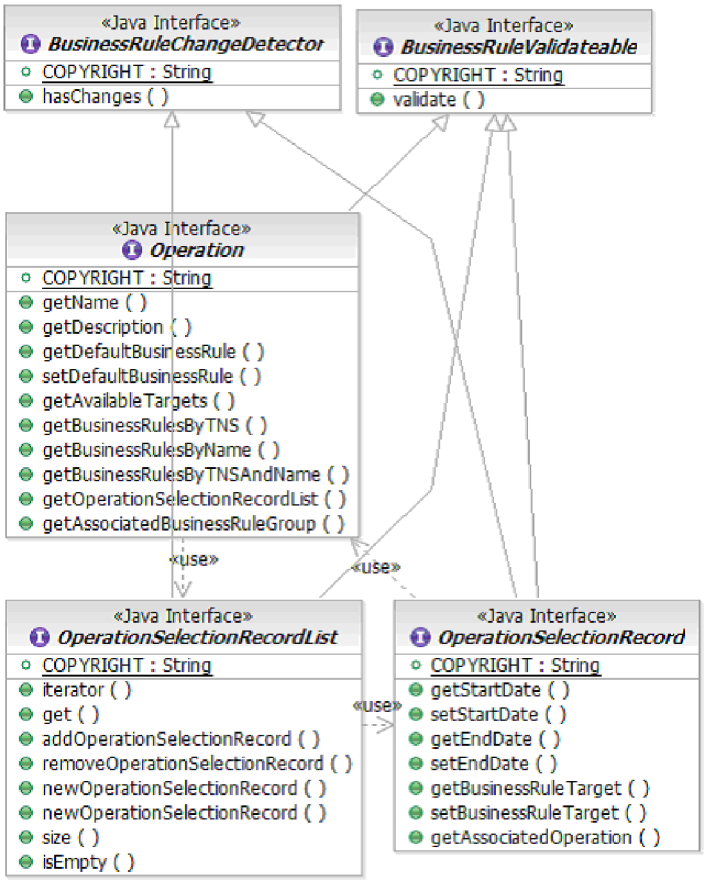


그림 12. Operation 및 관련 클래스의 클래스 다이어그램

비즈니스 규칙

RuleSet 및 DecisionTable 클래스는 규칙 세트 및 결정 테이블 둘 다에서 사용 가능한 정보를 제공하는 메소드를 사용하여 일반 BusinessRule 클래스를 기반으로 합니다.

비즈니스 규칙 그룹 아티팩트와 유사하게, 규칙 세트 및 결정 테이블에는 이름과 대상 네임 스페이스가 있습니다. 다른 규칙 세트와 결정 테이블과 비교할 때 이들 값의 조합은 고유해야 합니다. 예를 들어, 두 개의 규칙 세트는 동일한 대상 네임 스페이스 값을 공유할 수 있지만 다른 이름이나 규칙 세트를 가져야 하며 결정 테이블은 동일한 이름을 가질 수 있지만 다른 대상 네임 스페이스 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 네임 스페이스이거나 이름입니다. 이름 및 네임 스페이스의 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 네임 스페이스 검색
- 규칙 세트 또는 결정 테이블 이름 검색
- 규칙 세트 또는 결정 테이블의 표시 이름 검색 및 설정
- 비즈니스 규칙 유형(규칙 세트 또는 결정 테이블) 검색
- 비즈니스 규칙의 설명 검색 및 설정
- 비즈니스 규칙이 연관된 조작 검색
- 여러 이름 및/또는 대상 네임 스페이스를 사용하여 비즈니스 규칙 사본 작성

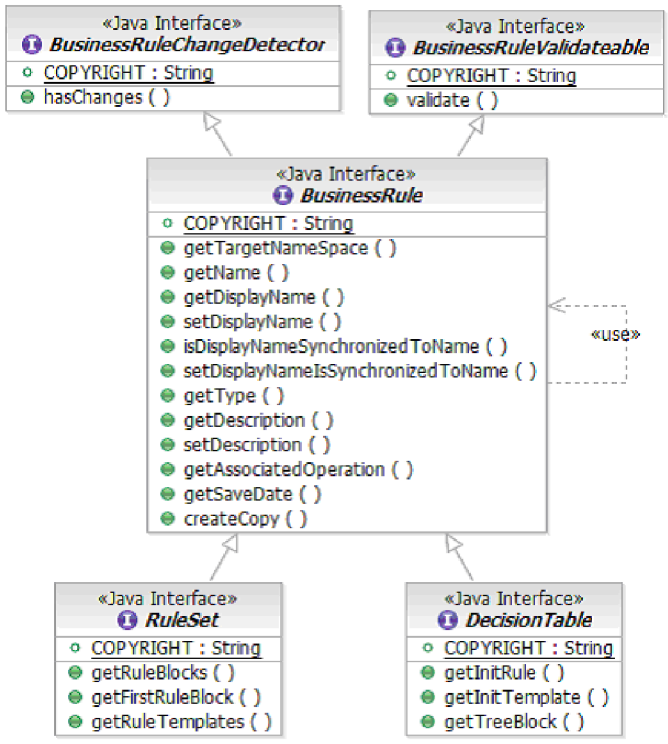


그림 13. BusinessRule 및 관련 클래스의 클래스 다이어그램

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 들어 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 들어 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 디자인되었으며 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

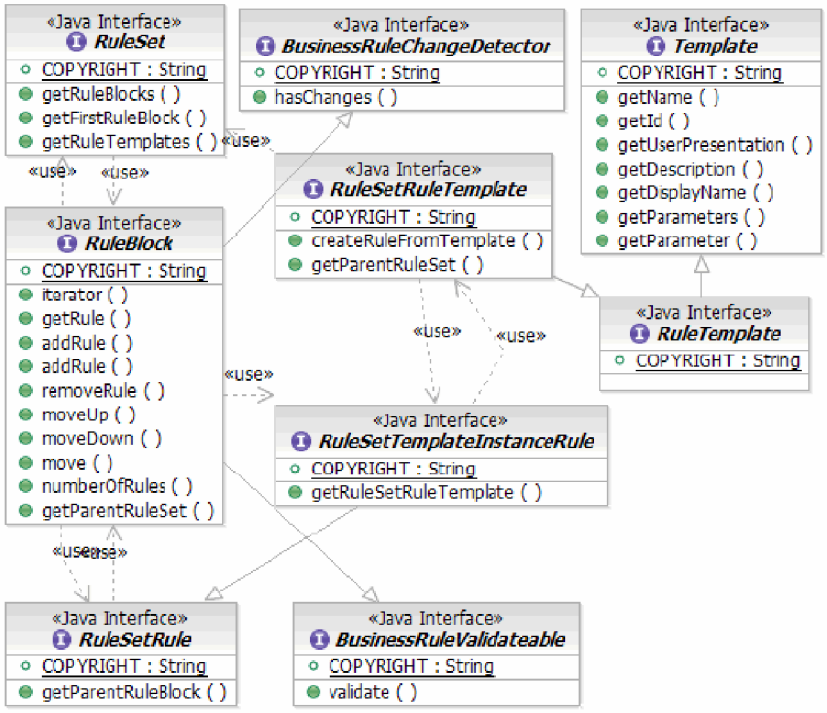


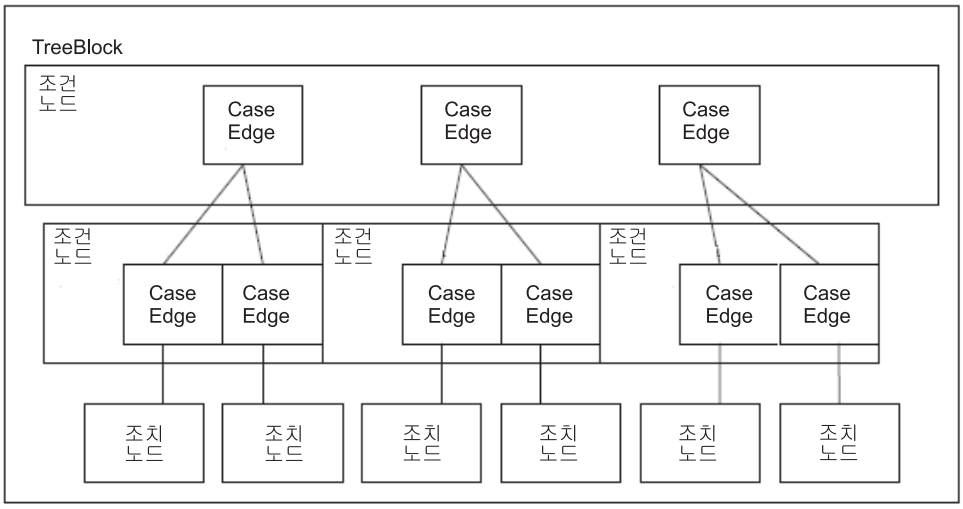
그림 14. BusinessRule 및 관련 클래스의 클래스 다이어그램

의사결정 테이블

결정 테이블은 관리되고 수정될 수 있는 비즈니스 규칙의 다른 유형입니다. 평가되어야 하는 일관된 수의 조건과 조건 충족 시 발행되는 특정 조치 세트가 있으면 일반적으로 결정 테이블이 사용됩니다.

결정 테이블은 결정 트리와 유사하지만 균형이 유지됩니다. 결정 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.

결정 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 결정 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

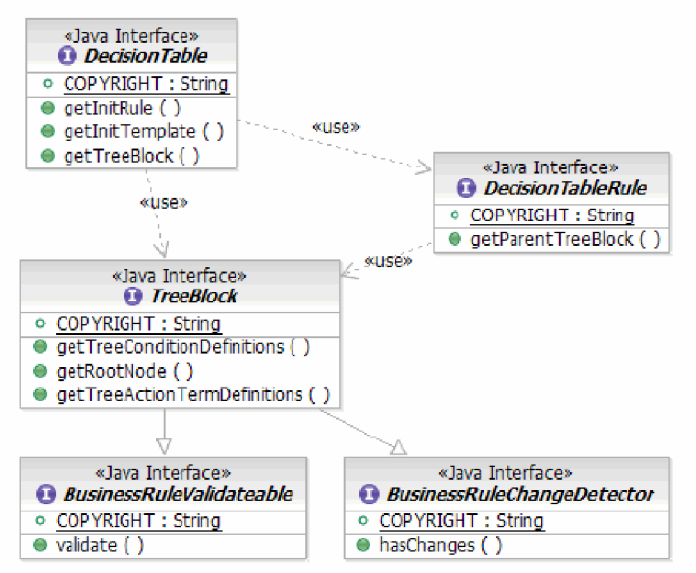
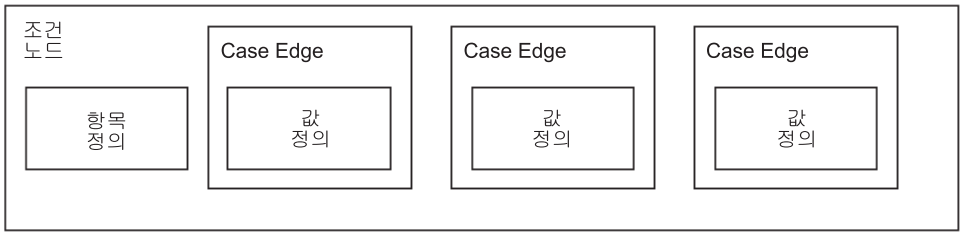


그림 15. DecisionTable 및 관련 클래스의 클래스 다이어그램

결정 테이블의 TreeBlock에는 여러 조건 및 조치 노드가 들어 있습니다. 각 조건 노드(ConditionNode)에는 항목 정의(TreeConditionTermDefinition)가 있으며 임의

수의 Case Edge(CaseEdge)가 있습니다. 항목 정의에는 조건 표현식의 왼쪽 피연산자가 들어 있습니다. Case Edge에는 조건 표현식에서 사용되는 다른 오른쪽 피연산자가 있는 값 정의가 있습니다. 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 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 피연산자의 표시만이며(예제에서 status) 플레이스홀더를 포함하지 않습니다. Case Edge의 경우, 값 정의를 위해 템플리트를 사용할 수 있습니다(TreeConditionValueTemplate). 템플리트 값 정의 인스턴스(TemplateInstanceExpression)는 실제로 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 결정 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(결정 테이블에 있는 Case Edge의 경우에서 `TreeConditionValueTemplate`) 검색

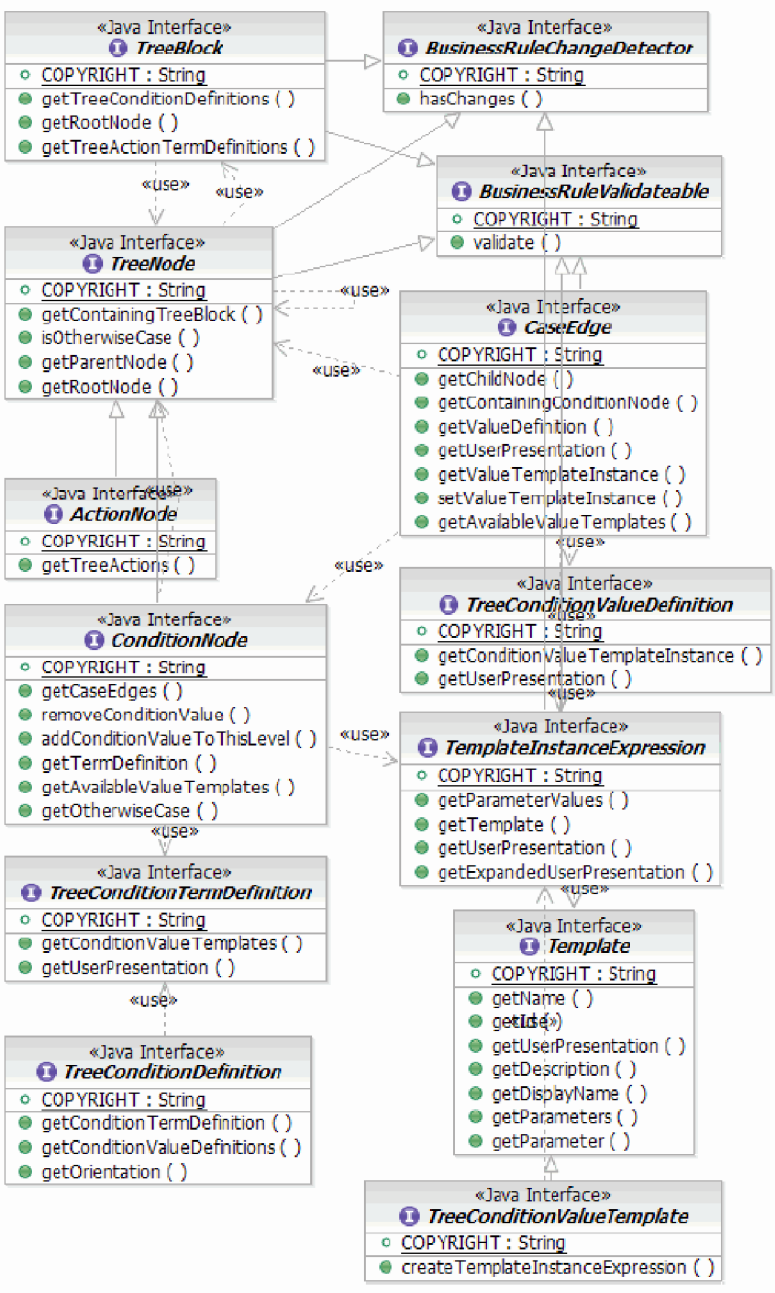


그림 16. TreeNode 및 관련 클래스의 클래스 다이어그램

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 설정할 필요가 있습니다.

새 Case Edge가 추가되면, 자동으로 하위 조건 노드를 추가합니다. 이 하위 조건 노드에는 동일한 레벨에서 조건 노드에 정의된 Case Edge 정의에 근거한 Case Edge가 들어 있습니다. 템플리트나 하드 코딩된 값이 Case Edge에 사용되면, 마찬가지로 하위 조건 노드의 Case Edge에서 사용됩니다. 자동으로 추가된 하위 조건 노드는 자동으로 작성된 고유한 하위 조건 노드를 갖습니다. 이 하위 조건 노드에는 하위 조건 노드가 포함되며 모든 레벨의 조건 노드가 재작성될 때까지 이러한 포함 관계가 유지됩니다.

조건 노드 이외에, 결정 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 실제로 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.

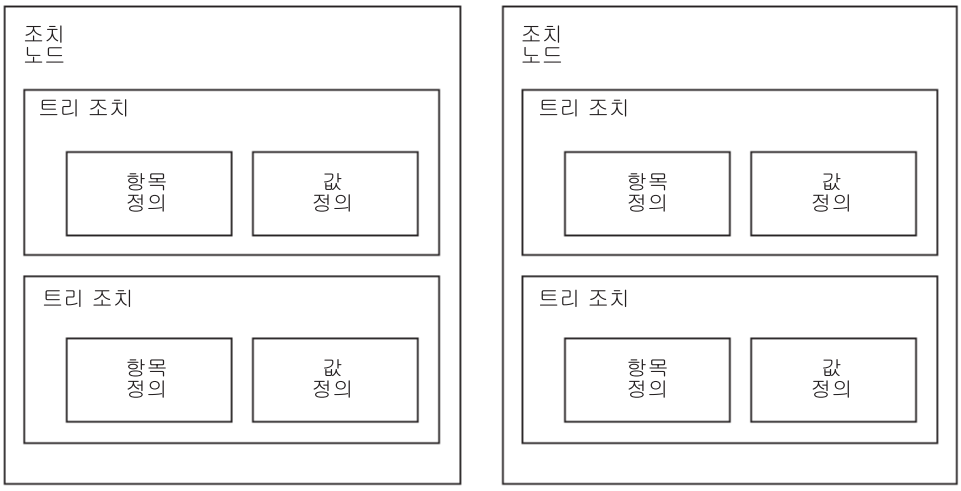
값 정의는 템플리트로 작성된 경우에만 변경될 수 있습니다. 클라이언트 응용프로그램의 경우, 사용자 프리젠테이션이 작성 시간에 지정된 경우 디스플레이에서 사용될 수 있습니다.

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 디자인을 상속하지 않습니다. 비어 있는 플레이스홀더 `TreeActions`만이 조치 노드에서 작성됩니다. 템플리트(`TreeActionValueTemplate`)는 조치 노드의 최소 하나의 항목 정의에 대한 `TemplateInstanceExpression`을 작성하여 조치 정의를 완료하는 데 사용되어야 합니다. 트리 조치가 `TemplateInstanceExpression`으로 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 결정 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, `TreeActionValueTemplate`의 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(결정 테이블에 있는 트리 조치의 경우에서 템플리트(TreeActionValueTemplate) 검색

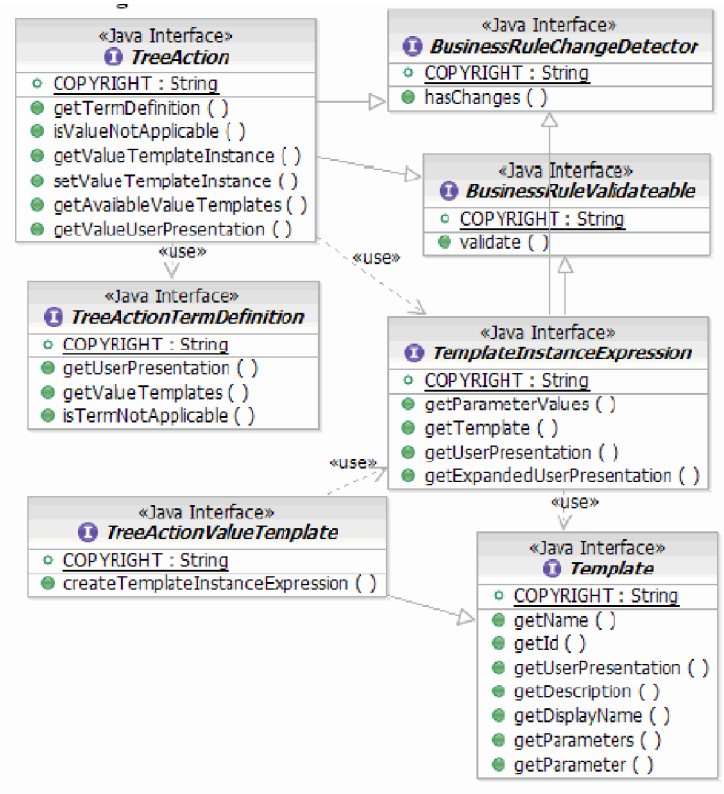


그림 17. TreeAction 및 관련 클래스의 클래스 다이어그램

결정 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 결정 테이블 검색

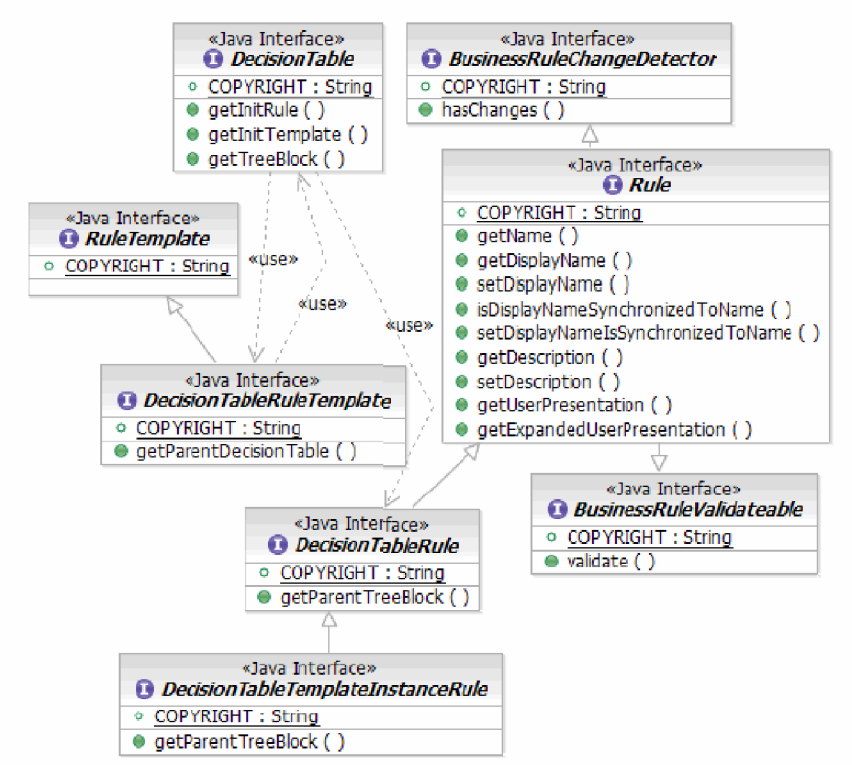


그림 18. DecisionTableRule 및 관련 클래스의 클래스 다이어그램

템플리트 및 매개변수

규칙 세트 및 결정 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 런타임에 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

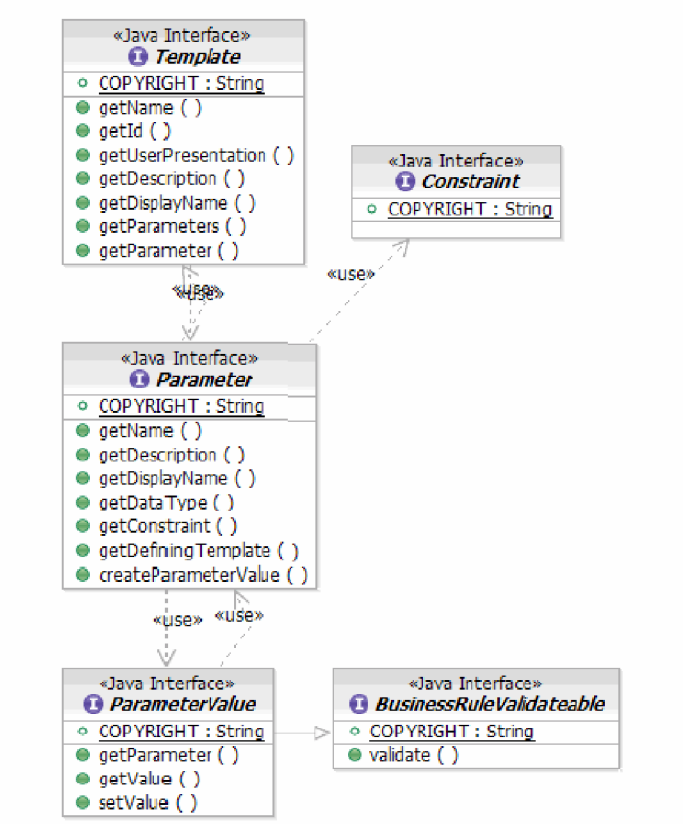


그림 19. 템플리트 및 매개변수와 관련 클래스의 클래스 다이어그램

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 런타임에 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 결정 테이블을 유효성 검증할 수 있습니다(규칙 그룹 컴포넌트 자체는 런타임에 편집 가능하지 않습니다). 비즈니스 규칙 그룹이 공개되면 규칙 그룹 선택사항 테이블, 규칙 세트 및 결정 테이블은 저장소로 공개되기 전에 유효성 검증됩니다.

아티팩트가 유효하지 않은 경우, 유효성 검증 문제점 목록과 함께 ValidationException이 발생합니다. 여러 유효성 검증 문제점이 예외 처리 섹션에 문서화됩니다.

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 결정 테이블을 이용하여 작업하는 기본 클래스입니다.

BusinessRuleManager에는 이름, 대상 네임 스페이스 또는 사용자 정의 특성에 의해 비즈니스 규칙을 검색하는 메소드가 있습니다. 또한 비즈니스 규칙 그룹, 규칙 세트 또는 결정 테이블에 작성한 변경사항을 공개하는 메소드가 있습니다.

BusinessRuleManager 클래스는 다음을 지원하는 메소드를 제공합니다.

- 모든 비즈니스 규칙 그룹 검색
- 특정 대상 네임 스페이스의 비즈니스 규칙 그룹 검색
- 특정 이름의 비즈니스 규칙 그룹 검색
- 특정 이름 및 대상 네임 스페이스의 비즈니스 규칙 그룹 검색
- 특정 특성이 들어 있는 비즈니스 규칙 그룹 검색
- 특정 특성이 들어 있는 비즈니스 규칙 그룹 검색
- 비즈니스 규칙 그룹 공개

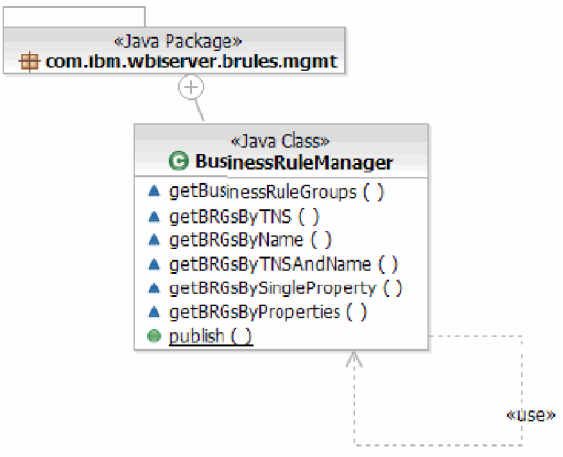


그림 20. BusinessRuleManager 및 패키지의 클래스 다이어그램

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 네임 스페이스
- 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. 이 논리 연산자 노드의 작성의 일부로, 연산자의 왼쪽 및 오른쪽은 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.

와일드 카드 검색은 LIKE 및 NOTLIKE 연산자가 사용될 때 지원됩니다. '%' 및 '_' 문자 둘 다는 와일드 카드 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 와일드 카드가 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.

와일드 카드는 왼쪽에서만(특성 값) 사용될 수 있습니다. 와일드 카드는 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 네임 스페이스에 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.

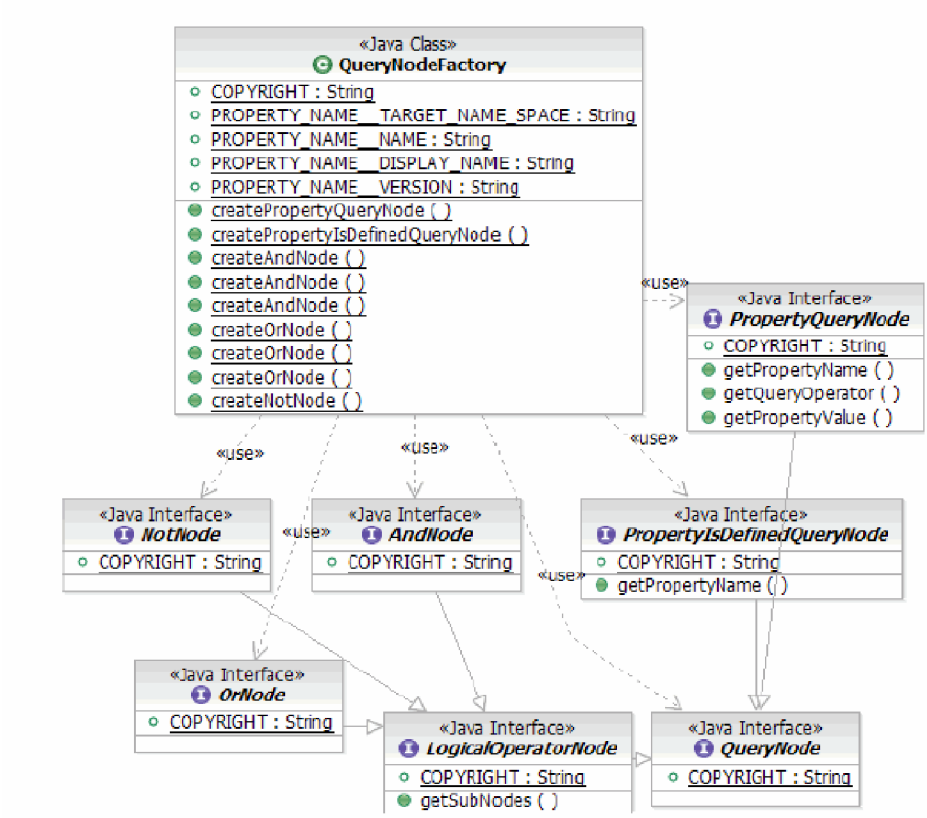


그림 21. QueryNodeFactory 및 관련 클래스의 클래스 다이어그램

트리의 노드는 부울 연산자, 와일드 카드(% 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, 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(예를 들어, 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(예를 들어, 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.

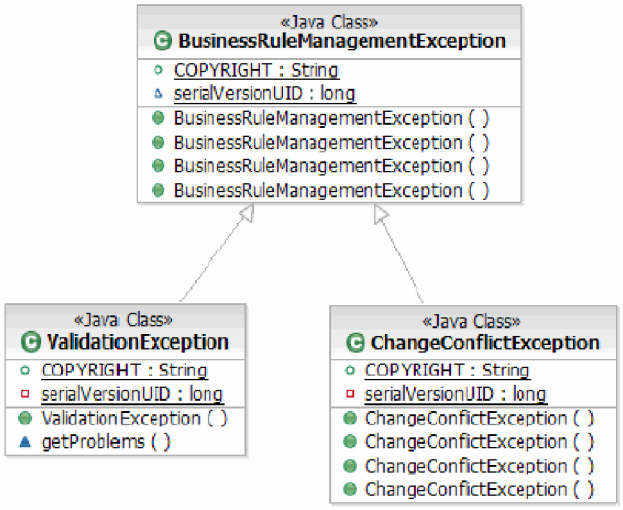


그림 22. BusinessRuleManagementException 및 관련 클래스의 클래스 다이어그램

비즈니스 규칙 그룹 문제점

비즈니스 규칙 그룹이 유효성 검증되거나 비즈니스 규칙 그룹을 공개하려는 시도가 있고 비즈니스 규칙 그룹의 일부가 유효하지 않으면 발생할 수 있는 문제점.

표 4. 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 다른 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 결정 테이블에 의해 이미 사용 중인 대상 네임 스페이스와 이름으로 새 비즈니스 규칙을 작성하면 발생하는 문제점. 저장소에 저장되는 규칙 아티팩트는 물론 현재 비즈니스 규칙 그룹과 연관된 모든 규칙 세트와 결정 테이블에서 검사가 수행됩니다. 이 문제점을 피하려면 다른 대상 네임 스페이스나 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 결정 테이블 문제점

표 5. 규칙 세트 및 결정 테이블 문제점

예외	설명
ProblemInvalidBooleanValue	규칙 세트의 규칙 템플리트의 매개변수나 결정 테이블의 조건 값이나 조치 값이 부울 유형의 매개변수에 대해 ″true″ 또는 ″false″이 아닌 다른 값을 수신할 때 발생하는 문제점. 잘못된 매개변수 값의 예제는 ″T″ 또는 ″F″입니다. 이 문제점을 피하려면 부울 유형의 매개변수로 작업 시 ″true″ 또는 ″false″ 값을 사용하십시오.

표 5. 규칙 세트 및 결정 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 결정 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 결정 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 달려 있습니다.

예

여러 가지 클래스가 비즈니스 규칙 그룹을 검색하고 규칙 세트 및 결정 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. 고기능의 웹 인터페이스가 됨을 의미하지는 않습니다. 클래스는 튼튼한 인터페이스를 빌드하기 위해 손쉽게 사용되거나 비즈니스 규칙을 수정하는 데 초점을 두는 다른 Java 응용프로그램에서 사용됩니다.

WebSphere Process Server v6.1에 응용 프로그램을 설치하거나 색인 페이지를 다음에서 액세스할 수 있습니다.

http://<hostname>:<port>/BRMgmtDriverWeb/

예를 들어, 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 대로의 결과는 물론 완료 코드를 이용하여 자세히 예제를 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 결정 테이블과 같은 모든 규칙 아티팩트를 변경하고 작업하는 것을 포함합니다. 이름과 네임 스페이스 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 결정 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
네임 스페이스: 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
네임 스페이스: 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
네임 스페이스: 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 결정 테이블 검색 및 인쇄

예제 1의 기능 이외에, 이 예제는 각 조작의 선택사항 테이블을 인쇄한 후 기본값 비즈니스 규칙 대상(규칙 세트나 결정 테이블 중 하나)과 조작을 위해 예정된 다른 비즈니스 규칙을 출력합니다. 규칙 세트와 결정 테이블 둘 다를 출력합니다.

예제 대부분은 동일하지만, 완벽을 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();
```

A specific business rule group is retrieved by name for this example.

```
  // 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();
 }
}
```

예

예제 2의 웹 브라우저 출력.

```
비즈니스 규칙 그룹
이름: DiscountRules
네임 스페이스: 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
네임 스페이스: 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:
결정 테이블
이름: calculateShippingDiscount
네임 스페이스: 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 항목, 오른쪽 항목 및 조건이 있습니다. 각 왼쪽 항목 및 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.printlnBold("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();
 }
}
```

예

예제 3의 웹 브라우저 출력.

example3 실행

```
비즈니스 규칙 그룹
이름: ApprovalValues
네임 스페이스: 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();
 }
}
```

예

예제 4의 웹 브라우저 출력.

example4 실행

```
비즈니스 규칙 그룹
이름: ApprovalValues
네임 스페이스: 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
네임 스페이스: 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");
```

조건의 왼쪽이 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();
 }
}
```

예

예제 5의 웹 브라우저 출력.

example5 실행

```
비즈니스 규칙 그룹
이름: ConfigurationValues
네임 스페이스: 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.printlnBold("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.printlnBold("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();
}
}
```

예

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.println1nBold("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();
}
}
```

예

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 결정 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();
 }
}
```

예

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 조작.

```
    // 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.printlnBold("Scheduled Destination:");
   record = opSelRecordIterator.next();
   out.println("Start Date: " + record.getStartDate()
```

```
   + " - End Date: " + record.getEndDate());
   BusinessRule ruleArtifact = record.getBusinessRuleTarget();
   out.println("Rule: " + ruleArtifact.getName());
   }
 }
}
```

예

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();
  }
}
```

예

예제 10의 웹 브라우저 출력.

```
예제 10 실행

공개 이전 규칙 세트:
규칙 세트
이름: getApprover
네임 스페이스: 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
네임 스페이스: 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();
 }
}
```

예

예제 11의 웹 브라우저 출력.

example11 실행

```
공개 이전 규칙 세트:
규칙 세트
이름: getApprover
네임 스페이스: 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
네임 스페이스: 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 결정 테이블에서 템플리트 수정 후 공개

이 예제에서, 조건 및 조치(둘 다 템플리트를 이용하여 정의)는 공개되기 전에 매개변수 값을 변경하여 결정 테이블에서 수정됩니다.

결정 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 ConditionValueTemplateInstance를 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(
 );
```

ConditionValueTemplateInstance를 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);
     }
    }
   }
  }
 }
}
```

예

예제 12의 웹 브라우저 출력.

예제 **12** 실행

```
공개 이전 규칙 세트:
결정 테이블
이름: getMessages
네임 스페이스: http://BRSamples/com/ibm/websphere/sample/brules

공개 이후 결정 테이블:
결정 테이블
이름: getMessages
네임 스페이스: http://BRSamples/com/ibm/websphere/sample/brules
```

예제 13: 조건 값 및 조치를 결정 테이블에 추가

이 예제에서, 조건 값 및 조치가 결정 테이블에 추가됩니다. 템플리트를 사용하여 결정 테이블에 조건 값을 추가할 수 있습니다.

조건 값을 조건 노드에 추가하면, 실제로 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. 우수 사례로, 해당 유형의 조건에 대해 올바른 템플리트를 검색하기 위해 각 조건 노드 레벨에서 템플리트 이름에 고유한 이름을 부여하는 것이 가장 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 결정 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;
```

결정 테이블에서 각 조건 노드 레벨의 고유한 템플리트 이름을 사용함으로써, 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 얻게 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 결정 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. 위로 이동           // 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.println1nBold("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;
}
}
```

예

예제 13의 웹 브라우저 출력.

예제 **13** 실행

```
공개 이전 결정 테이블:
결정 테이블
이름: getMessages
네임 스페이스: http://BRSamples/com/ibm/websphere/sample/brules
```

공개 이후 결정 테이블:
결정 테이블
이름: getMessages
네임 스페이스: 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")) {
```

문제점을 야기하기 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 (report error, correct
// error, etc.)
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();
}
}
```

예

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, as a
      // best practice, 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();
 }
}
```

예

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.
```

부록

부록은 공통 조작을 단순화하기 위해 예제에 사용된 추가 클래스와 와일드 카드를 사용하여 비즈니스 규칙 그룹을 검색하기 위한 복합 조회 작성의 추가 예제를 포함합니다.

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 printlnBold(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 결정 테이블을 인쇄하기 위해 사용됩니다. 이 메소드는 결정 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();
      }
}
```

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 와일드 카드 값('_', '%')은 다른 연산자(AND, OR, LIKE, NOT_LIKE, EQUAL 및 NOT_EQUAL)와 함께 사용됩니다.

예

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 와일드 카드별(%) 비즈니스 규칙 그룹 조회

특성별 및 값의 처음과 끝에 있는 와일드 카드별(%) 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 와일드 카드별('_') 비즈니스 규칙 그룹 조회

특성별 및 와일드 카드별(%) 비즈니스 규칙 그룹 조회의 예제입니다.

```
brgList = BusinessRuleManager.getBRGsBySingleProperty("ID",
QueryOperator.LIKE, "00000_3", 0, 0);

// Returns BRG3 and BRG4
```

다중 와일드 카드('_' 및 '%')와 특성별 비즈니스 규칙 그룹 조회

다중 와일드 카드('_' 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 와일드 카드('_')에 의한 비즈니스 규칙 그룹 조회

NOT_LIKE 연산자 및 와일드 카드('_')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 결합되어 있는 노드 목록별 비즈니스 규칙 그룹 조회

AND 연산자와 결합되어 있는 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 결합되어 있는 노드 목록별 비즈니스 규칙 그룹 조회

OR 연산자와 결합되어 있는 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
```

제 13 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 탐색기를 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 탐색기 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 하부 구조가 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 로드될 수 있습니다. RAL은 클라이언트 응용프로그램이 전체 WebSphere Process Server 서버 설치에서 실행하는 경우 교차 셀로도 사용될 수 있습니다. 그러나 RAL은 클라이언트 응용프로그램이 WebSphere Process Server 클라이언트 설치에서 실행하는 교차 셀 설치에서는 사용될 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.	Microsoft .NET 환경을 포함하여 웹 서비스 호출을 지원하는 모든 런타임 환경.	SCA JMS 가져오기를 사용하는 SCA 모듈을 포함하여 JMS 클라이언트를 지원하는 모든 런타임 환경.	REST 클라이언트를 지원하는 모든 런타임 환경.
보안	J2EE(Java 2, Enterprise Edition) 보안.	웹 서비스 보안.	WebSphere Process Server 설치의 J2EE(Java 2, Enterprise Edition) 보안. 예를 들어, WebSphere MQ 보안 메커니즘을 사용하여 클라이언트 응용프로그램이 API 메시지를 넣는 대기열을 보안할 수도 있습니다.	REST 메소드를 호출하는 클라이언트 응용프로그램은 적절한 HTTP 인증 메커니즘을 사용해야 합니다.

관련 태스크

231 페이지의 『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.

296 페이지의 『웹 서비스 API 클라이언트 응용프로그램 개발』

웹 서비스 API를 통해 비즈니스 프로세스 응용프로그램 및 휴먼 타스크 응용프로그램을 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

328 페이지의 『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.

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.

다음 EJB 조회 인터페이스는 Business Process Choreographer로 사용 가능합니다.

	설명
Business Process Choreographer EJB 조회 API	인스턴스 데이터 및 템플리트 데이터에 대한 액세스를 제공합니다. 시스템에서 모든 프로세스 및 타스크 관련 데이터는 이 인터페이스를 통해 액세스 가능합니다. 비즈니스 플로우 관리자 및/또는 휴먼 타스크 관리자의 관련 메소드는 다음을 포함합니다. • query • queryAll • queryProcessTemplates • queryTaskTemplates
Business Process Choreographer EJB 조회 테이블 API	인스턴스 데이터 및 템플리트 데이터에 대한 액세스를 제공합니다. 이 인터페이스는 프로세스 및 타스크 목록을 조회하는 데 특수화된 Business Process Choreographer 조회 테이블을 조회하는 데 사용됩니다. 비즈니스 플로우 관리자에서 관련 메소드는 다음과 같습니다. • queryEntities • queryEntityCount • queryRows • queryRowCount

프로세스나 타스크 관련 데이터를 액세스하는 클라이언트에 따라, 위에 나열된 인터페이스 중 하나 이상은 올바른 선택이 될 수 있습니다. 타스크 및 프로세스 목록 데이터를 조회하기 위해 Business Process Choreographer에서 사용 가능한 REST 및 웹 서비스 API도 있습니다. 그러나 높은 볼륨 프로세스 목록 및 타스크 목록 조회의 경우 성능상의 이유로 Business Process Choreographer EJB 조회 테이블 API를 사용하십시오.

Business Process Choreographer에서 테이블 조회

조회 테이블은 타스크 목록 및 프로세스 인스턴스 목록으로 참조되는 정보의 요약 정의이며, 타스크 또는 비즈니스 프로세스로 작업하는 사용자에게 표시됩니다. 조회 테이블은 사용자 정의될 수 있습니다. 예를 들어, 구성 옵션은 특정 시나리오에서 관련된 프로세스 인스턴스나 타스크만을 조회 테이블이 포함하는지 판별할 수 있습니다. 높은 볼륨 프로세스 목록 및 타스크 목록 조회가 있는 경우와 같이 성능이 중요한 곳에서는 조회 테이블을 사용하십시오.

조회 테이블은 기능성 및 성능 측면에서 Business Process Choreographer의 기존 조회 인터페이스와 사전정의된 데이터베이스 보기를 향상시킵니다.

- 성능 최적화 액세스 패턴을 사용하여, 조회 테이블은 프로세스 및 타스크 목록 조회 실행을 위해 최적화됩니다.
- 조회 테이블은 타스크나 프로세스 목록에 포함된 컨텐츠의 요약 정의입니다. 일단 정의되면, 조회 테이블은 필요한 정보에 대한 액세스를 단순화시키고 통합시킵니다.
- 조회 테이블은 권한 및 필터 옵션의 세밀한 구성을 가능하게 합니다.
- 조회 테이블의 컨텐츠 정의는 타스크 및 프로세스 목록에 용이합니다. 예를 들어, 타스크, 프로세스 또는 에스컬레이션과 같이 작업할 엔티티를 우선 선택하십시오. 그런 다음 타스크 설명이나 조회 특성과 같이, 엔티티를 표시하는 데 필요한 추가 정보를 선택하십시오.

세 가지 유형(사전 정의된 조회 테이블, 추가 조회 테이블 및 복합 조회 테이블)의 조회 테이블이 있습니다. 이러한 모든 유형의 테이블은 조회 테이블 API를 사용하여 조회됩니다. 복합 및 추가 조회 테이블은 도구 조회 테이블 빌더를 사용하여 개발됩니다.

관련 태스크

조회 테이블 관리
조회 테이블 빌더를 사용하여 개발된 Business Process Choreographer에서 조회 테이블을 관리하려면 manageQueryTable.py 스크립트를 사용하십시오.

조회 테이블 전개
Business Process Choreographer에서 추가 및 복합 조회 테이블을 전개하려면 manageQueryTable.py 스크립트를 사용하십시오. 조회 테이블은 실행 중인 독립형 서버나 최소한 하나의 구성원이 실행 중인 클러스터에서 전개됩니다. 추가 및 복합 조회 테이블의 전개 취소는 실행 중인 서버에서도 수행됩니다. 추가 조회 테이블의 경우, 관련된 물리적 데이터베이스 오브젝트, 일반적으로 데이터베이스 보기 또는 데이터베이스 테이블이 조회 테이블을 사용하기 전에 존재하지 않으면 작성되어야 합니다.

관련 참조

Business Process Choreographer에 대한 데이터베이스 보기
이 참조 정보는 사전 정의된 데이터베이스 보기의 열을 설명합니다.

사전정의된 조회 테이블

Business Process Choreographer의 사전정의된 조회 테이블은 TASK 또는 PROCESS_INSTANCE와 같이, 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. Business Process Choreographer 데이터베이스에서 간단한 데이터 보기를 제공합니다. 그러나 사전정의된 조회 테이블은 권한, 기능성 및 성능 측면에서 사전정의된 데이터베이스 보기와는 다릅니다.

조회 테이블 사용 시 권한 및 필터 옵션의 세밀한 구성을 가능하게 합니다.

사전정의된 조회 테이블은 동일한 기본적 물리 데이터를 사용하므로 사전정의된 데이터베이스 보기와 동일한 구조를 갖습니다. 그러나 사전정의된 조회 테이블은 실행 중인 프로세스와 타스크 목록 조회를 위해 최적화되어 사전정의된 데이터베이스 보기의 성능과 기능성을 향상시킵니다.

사전정의된 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다. 그러나 제안되는 조회 테이블의 사용은 조회 실행 시 검색되는 정보를 포함하는 복합 조회 테이블을 개발하는 것입니다.

조회 테이블에서 조회를 실행하기 위해 조회 테이블 API 사용 시 매개변수를 제출할 수 있습니다. 사전정의된 조회 테이블은 매개변수를 지원하지 않습니다.

다음과 같이 사전정의된 조회 테이블은 복합 조회 테이블의 기본 또는 첨부된 조회 테이블로서 또는 직접 조회를 위해 사용 가능합니다.

사전 정의된 조회 테이블에는 인스턴스 데이터가 들어 있으며 인증된 모든 사용자가 조회할 수 있습니다.

- ACTIVITY
- ACTIVITY_ATTRIBUTE
- ACTIVITY_SERVICE
- APPLICATION_COMP
- ESCALATION
- ESCALATION_CPROP
- ESCALATION_DESC
- PROCESS_ATTRIBUTE
- PROCESS_INSTANCE
- QUERY_PROPERTY
- TASK
- TASK_CPROP
- TASK_DESC

권한은 모든 작업 항목 즉, 모두, 개별, 그룹 및 상속 작업 항목에 사용 가능합니다. 인스턴스 데이터가 있는 사전정의된 조회 테이블에서는, 지정된 경우를 제외하고 조회 테이블 API가 모두, 개별 및 그룹 작업 항목에 기본값입니다.

사전 정의된 조회 테이블에는 템플리트 데이터가 들어 있으며 조회 테이블 API를 사용하는 관리 사용자가 조회할 수 있습니다.

- ESC_TEMPL

- ESC_TEMPL_CPROP
- ESC_TEMPL_DESC
- PROCESS_TEMPLATE
- TASK_TEMPL
- TASK_TEMPL_CPROP
- TASK_TEMPL_DESC

템플리트 데이터가 있는 사전정의된 조회 테이블은 작업 항목을 이용하여 권한을 적용하지 않으며, AdminAuthorizationOptions 오브젝트를 사용하는 관리자만이 조회할 수 있습니다.

관련 개념

205 페이지의『추가 조회 테이블』

Business Process Choreographer의 추가 조회 테이블은 일반적으로 외부 데이터베이스 테이블이나 뷰에서 또는 데이터베이스 오브젝트에서 조회 테이블 API로 데이터를 나타냅니다. 추가 조회 테이블에서, 이 외부 데이터는 비즈니스 프로세스 인스턴스 정보나 휴먼 타스크 정보와 결합될 수 있습니다. 추가 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다.

206 페이지의『복합 조회 테이블』

Business Process Choreographer의 복합 조회 테이블은 사전정의된 조회 테이블 및 추가 조회 테이블로 구성됩니다. 기존 테이블이나 보기로부터 데이터를 결합시킵니다. 일반적으로, 내가 수행할 작업과 같이, 프로세스 인스턴스 목록이나 타스크 목록에 표시되는 정보를 검색하기 위해 복합 조회 테이블이 사용됩니다.

209 페이지의『조회 테이블 개발』

Business Process Choreographer의 추가 및 복합 조회 테이블은 도구 조회 테이블 빌더를 사용하여 응용프로그램 개발 중 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.

212 페이지의『조회 테이블 API 개요』

조회 테이블 API에서 행 기반 조회 및 엔티티 기반 조회를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행할 수 있습니다.

관련 참조

Business Process Choreographer에 대한 데이터베이스 보기

이 참조 정보는 사전 정의된 데이터베이스 보기의 열을 설명합니다.

추가 조회 테이블

Business Process Choreographer의 추가 조회 테이블은 일반적으로 외부 데이터베이스 테이블이나 뷰에서 또는 데이터베이스 오브젝트에서 조회 테이블 API로 데이터를 나타냅니다. 추가 조회 테이블에서, 이 외부 데이터는 비즈니스 프로세스 인스턴스 정보나 휴먼 타스크 정보와 결합될 수 있습니다. 추가 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다.

추가 조회 테이블은 Business Process Choreographer에 의해 유지되는 데이터에 추가되는 데이터를 포함하는 데이터베이스에서 오브젝트를 기술합니다. 일반적으로 이것은 데이터베이스 뷰나 데이터베이스 테이블입니다. 추가 조회 테이블은 관련된 데이터베이스 오브젝트의 열을 기술합니다. 추가 조회 테이블의 이름은 접두부 및 이름으로 구성되어야 합니다(예: COMPANY.EXT_DATA).

주: 조회 테이블의 컨텍스트와 조회 테이블 API에서, 열은 일반적으로 속성으로 참조됩니다. 조회 테이블의 컨텐츠는 데이터베이스에 저장되므로, 용어 '열'도 사용될 수 있습니다.

조회 테이블에서 조회를 실행하기 위해 조회 테이블 API 사용 시 매개변수를 제출할 수 있습니다. 추가 조회 테이블은 매개변수를 지원하지 않습니다.

작업 항목이 있는 권한은 추가 조회 테이블에서 지원되지 않습니다. 인증된 모든 사용자는 추가 조회 테이블의 컨텐츠를 액세스할 수 있습니다.

관련 개념

202 페이지의『사전정의된 조회 테이블』

Business Process Choreographer의 사전정의된 조회 테이블은 TASK 또는 PROCESS_INSTANCE와 같이, 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. Business Process Choreographer 데이터베이스에서 간단한 데이터 보기를 제공합니다. 그러나 사전정의된 조회 테이블은 권한, 기능성 및 성능 측면에서 사전정의된 데이터베이스 보기와는 다릅니다.

206 페이지의『복합 조회 테이블』

Business Process Choreographer의 복합 조회 테이블은 사전정의된 조회 테이블 및 추가 조회 테이블로 구성됩니다. 기존 테이블이나 보기로부터 데이터를 결합시킵니다. 일반적으로, 내가 수행할 작업과 같이, 프로세스 인스턴스 목록이나 타스크 목록에 표시되는 정보를 검색하기 위해 복합 조회 테이블이 사용됩니다.

209 페이지의『조회 테이블 개발』

Business Process Choreographer의 추가 및 복합 조회 테이블은 도구 조회 테이블 빌더를 사용하여 응용프로그램 개발 중 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.

212 페이지의『조회 테이블 API 개요』
조회 테이블 API에서 행 기반 조회 및 엔티티 기반 조회를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행할 수 있습니다.

복합 조회 테이블

Business Process Choreographer의 복합 조회 테이블은 사전정의된 조회 테이블 및 추가 조회 테이블로 구성됩니다. 기존 테이블이나 보기로부터 데이터를 결합시킵니다. 일반적으로, 내가 수행할 작업과 같이, 프로세스 인스턴스 목록이나 타스크 목록에 표시되는 정보를 검색하기 위해 복합 조회 테이블이 사용됩니다.

복합 조회 테이블은 사전정의되고 추가된 조회 테이블에서 사용 가능한 정보의 조합입니다. 대부분 조회 응답 시간에 영향을 미치는 다양한 구성 옵션이 복합 조회 테이블에서 사용 가능한 정보를 지정하기 위해 사용 가능합니다.

구조

복합 조회 테이블은 기본 조회 테이블과 0개 이상의 첨부된 조회 테이블로 구성됩니다. 복합 조회 테이블의 이름은 접두부 및 이름으로 구성되어야 합니다
(예: COMPANY.TODO_TASK_LIST).

- 기본 조회 테이블은 복합 조회 테이블에 포함된 기본 정보를 구성합니다.

  권한이 필요한 경우, 기본 조회 테이블의 오브젝트는 사용 가능한 작업 항목에 대해 점검되며, 권한 옵션이 고려됩니다. 예를 들어, 사용자가 잠재적인 소유자인 휴먼 타스크만을 리턴하도록 조회가 정의되었습니다.

  또한 복합 조회 테이블의 각 오브젝트는 기본 조회 테이블의 기본 키에 의해 고유하게 식별될 수 있습니다. 예를 들어, TASK의 경우 이것은 타스크 ID TKIID입니다. 사전정의된 조회 테이블만이 기본 조회 테이블로 선택될 수 있습니다. 일반적으로, 기본 조회 테이블은 사전정의된 조회 테이블 TASK 또는 사전정의된 조회 테이블 PROCESS_INSTANCE입니다.
- 0개 이상 첨부된 조회 테이블은 복합 조회 테이블에 정의될 수 있습니다.

  기본 필터는 물론 권한의 필터 및 옵션 결과로 복합 조회 테이블에 포함된 각 오브젝트는 첨부된 조회 테이블에 포함된 추가 정보로 풍부해 질 수 있습니다. 예를 들어, 특정 로케일의 타스크 설명은 기본 조회 테이블 TASK를 이용해서 복합 조회 테이블에 추가될 수 있습니다.

  선택 기준을 사용하여 필요하다면, 일대일 또는 일대영 관계가 기본 조회 테이블 및 첨부된 조회 테이블 간에 유지되어야 합니다. 첨부된 조회 테이블은 이미 시스템에서 전개된 사전정의된 조회 테이블 및 추가 조회 테이블일 수 있습니다.

성능

조회 테이블에서의 조회 응답 시간은 주로 선택된 권한 옵션, 필터 및 선택 기준에 의해 좌우됩니다.

- 권한 옵션은 성능에 상당한 영향을 미칩니다. 개별 및 그룹 작업 항목과 같이 가능한 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.
- 기본 조회 테이블에서의 필터링은 효과적입니다. 예를 들어, TASK가 기본 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.

구현

복합 조회 테이블에는 데이터베이스의 물리적 표시가 없습니다. 복합 조회 테이블은 타스크 및 프로세스 목록 조회용으로 최적화되는 SQL로 실현됩니다.

권한

복합 조회 테이블은 권한을 요구하거나 요구하지 않도록 구성될 수 있습니다. 권한이 필수이면, 기본 조회 테이블의 오브젝트는 관련된 작업 항목의 WORK_ITEM 조회 테이블에 대해 SQL 결합을 사용하여 확인됩니다. 기본 조회 테이블이 TASK 또는 PROCESS_INSTANCE 조회 테이블과 같은 인스턴스 데이터를 포함하는 경우 이것이 기본값입니다.

다음 권한 옵션은 권한이 필수인 경우 복합 조회 테이블의 정의에서 사용 가능합니다.

- **모두 작업 항목**: 지정된 경우, 관련된 모두의 작업 항목이 있는 오브젝트는 복합 조회 테이블에 포함됩니다.
- **개별 작업 항목**: 지정된 경우, 관련된 개별 작업 항목이 있는 오브젝트는 복합 조회 테이블에 포함됩니다.
- **그룹 작업 항목**: 지정된 경우, 관련된 그룹 작업 항목이 있는 오브젝트는 복합 조회 테이블에 포함됩니다.
- **상속 작업 항목**: 지정된 경우, 참여하는 휴먼 타스크와 같이 구성된 대로 관련된 모두, 개별 또는 그룹 작업 항목을 가진 상위로서 프로세스 인스턴스를 갖는 오브젝트는 복합 조회 테이블에 포함됩니다. 일반적으로 상속된 작업 항목은 관리자에게만 유용합니다.

매개변수

매개변수는 정의된 필터 및 선택사항 기준의 파트를 동적으로 유지하기 위해 조회 테이블의 선택사항 기준 및 필터에서 사용될 수 있습니다.

필터

필터는 조회 테이블의 컨텐츠를 제한하기 위해 사용됩니다.

- **기본 필터**: 이 필터는 기본 조회 테이블에 정의됩니다. 기본 조회 테이블에 정의된 열에서 조건을 사용하여 복합 조회 테이블의 컨텐츠를 제한합니다.
- **권한 필터**: 이 필터는 권한을 실현하기 위해 사용되는 사전정의된 조회 테이블 WORK_ITEM에 정의된 열을 이용하여 복합 조회 테이블의 컨텐츠를 제한합니다. 작업 항목 작성은 Business Process Choreographer의 프로세스 및 휴먼 타스크에서 스태프 verb를 사용하여 정의됩니다.

주: 조회 테이블의 컨텍스트와 조회 테이블 API에서, 열은 일반적으로 속성으로 참조됩니다. 조회 테이블의 컨텐츠는 데이터베이스에 저장되므로, 용어 '열'도 사용됩니다.

선택 기준

일대일 또는 일대영 관계가 기본 조회 테이블의 오브젝트와 첨부된 조회 테이블의 오브젝트 간에 유지되어야 합니다. 이것은 첨부된 조회 테이블에서의 선택사항 기준을 사용하여 이루어집니다. 예를 들어, TASK가 기본 조회 테이블이고 TASK_DESC가 첨부된 조회 테이블인 경우, 선택사항 기준은 보통 복합 조회 테이블에서 휴먼 타스크에 추가된 설명과 관련하여 특정한 로케일을 하나 선택합니다. 이것의 예제는 `LOCALE='en_US'` 입니다.

관련 개념

202 페이지의『사전정의된 조회 테이블』

Business Process Choreographer의 사전정의된 조회 테이블은 TASK 또는 PROCESS_INSTANCE와 같이, 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. Business Process Choreographer 데이터베이스에서 간단한 데이터 보기를 제공합니다. 그러나 사전정의된 조회 테이블은 권한, 기능성 및 성능 측면에서 사전정의된 데이터베이스 보기와는 다릅니다.

205 페이지의『추가 조회 테이블』

Business Process Choreographer의 추가 조회 테이블은 일반적으로 외부 데이터베이스 테이블이나 뷰에서 또는 데이터베이스 오브젝트에서 조회 테이블 API로 데이터를 나타냅니다. 추가 조회 테이블에서, 이 외부 데이터는 비즈니스 프로세스 인스턴스 정보나 휴먼 타스크 정보와 결합될 수 있습니다. 추가 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다.

209 페이지의『조회 테이블 개발』

Business Process Choreographer의 추가 및 복합 조회 테이블은 도구 조회 테이

블 빌더를 사용하여 응용프로그램 개발 중 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.

212 페이지의『조회 테이블 API 개요』
조회 테이블 API에서 행 기반 조회 및 엔티티 기반 조회를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행할 수 있습니다.

조회 테이블 개발

Business Process Choreographer의 추가 및 복합 조회 테이블은 도구 조회 테이블 빌더를 사용하여 응용프로그램 개발 중 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 빌더 도구는 Eclipse 플러그인으로 사용 가능하며 WebSphere Business Process Management SupportPacs 사이트에서 다운로드될 수 있습니다. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크를 액세스하려면, 이 주제의 관련된 참조 섹션을 참조하십시오.

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 조회하기 위해 제공됩니다. 응용프로그램 개발에서, 사전정의된 조회 테이블을 직접 조회하기 보다는 복합 조회 테이블이 사용되어야 합니다.

예제 1

```
// get the naming context and lookup the business
// flow manager EJB home; note that the business flow
// manager EJB home should be cached for performance
// reasons; also, it is assumed that there's a EJB
// reference to the local business flow manager EJB
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
 (LocalBusinessFlowManagerHome)
 ctx.lookup("java:comp/env/ejb/BFM");

// create the business flow manager client-side stub
LocalBusinessFlowManager bfm = home.create();

// ************************************************
// ****************** example 1 *******************
// ************************************************

// execute a query against the predefined query table
// TASK; 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

202 페이지의 『사전정의된 조회 테이블』

Business Process Choreographer의 사전정의된 조회 테이블은 TASK 또는 PROCESS_INSTANCE와 같이, 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. Business Process Choreographer 데이터베이스에서 간단한 데이터 보기를 제공합니다. 그러나 사전정의된 조회 테이블은 권한, 기능성 및 성능 측면에서 사전정의된 데이터베이스 보기와는 다릅니다.

205 페이지의 『추가 조회 테이블』

Business Process Choreographer의 추가 조회 테이블은 일반적으로 외부 데이터베이스 테이블이나 뷰에서 또는 데이터베이스 오브젝트에서 조회 테이블 API로 데이터를 나타냅니다. 추가 조회 테이블에서, 이 외부 데이터는 비즈니스 프로세스 인스턴스 정보나 휴먼 타스크 정보와 결합될 수 있습니다. 추가 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다.

206 페이지의 『복합 조회 테이블』

Business Process Choreographer의 복합 조회 테이블은 사전정의된 조회 테이블 및 추가 조회 테이블로 구성됩니다. 기존 테이블이나 보기로부터 데이터를 결합시킵

니다. 일반적으로, 내가 수행할 작업과 같이, 프로세스 인스턴스 목록이나 타스크 목록에 표시되는 정보를 검색하기 위해 복합 조회 테이블이 사용됩니다.

『조회 테이블 API 개요』
조회 테이블 API에서 행 기반 조회 및 엔티티 기반 조회를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행할 수 있습니다.

관련 태스크


조회 테이블 관리
조회 테이블 빌더를 사용하여 개발된 Business Process Choreographer에서 조회 테이블을 관리하려면 manageQueryTable.py 스크립트를 사용하십시오.


조회 테이블 전개
Business Process Choreographer에서 추가 및 복합 조회 테이블을 전개하려면 manageQueryTable.py 스크립트를 사용하십시오. 조회 테이블은 실행 중인 독립형 서버나 최소한 하나의 구성원이 실행 중인 클러스터에서 전개됩니다. 추가 및 복합 조회 테이블의 전개 취소는 실행 중인 서버에서도 수행됩니다. 추가 조회 테이블의 경우, 관련된 물리적 데이터베이스 오브젝트, 일반적으로 데이터베이스 보기 또는 데이터베이스 테이블이 조회 테이블을 사용하기 전에 존재하지 않으면 작성되어야 합니다.

조회 테이블 API 개요

조회 테이블 API에서 행 기반 조회 및 엔티티 기반 조회를 사용하여 Business Process Choreographer의 조회 테이블에 대해 조회를 실행할 수 있습니다.

조회는 하나의 특정 조회 테이블에서만 실행됩니다. 다중 조회 테이블 간의 관계는, 즉 표준 조회 API, 데이터베이스 보기의 측면에서 복합 조회 테이블로 정의됩니다.

조회 테이블 API에서, 두 개의 다른 개념이 Business Process Choreographer의 조회 테이블에 대해 조회를 실행하는 데 사용 가능합니다.

- **엔티티 기반 조회**: queryEntities 메소드 및 queryEntityCount 메소드를 사용하는 엔티티 조회는, 기본 조회 테이블에 정의된 대로 조회 테이블이 고유하게 식별 가능한 엔티티를 포함한다고 가정합니다. 이들 엔티티는 기본 조회 테이블의 기본 키로 식별됩니다.
- **행 기반 조회**: queryRows 메소드 및 queryRowCount 메소드를 사용하는 행 조회는 JDBC와 같은 결과 세트를 리턴합니다. 동일한 엔티티(예를 들어, TKIID와 같이 해당 타스크 ID에 의해 식별되는 휴먼 타스크)는 결과 세트에서 여러 번 발생할 수 있습니다.

queryEntities 메소드에 의해 리턴되는 엔티티 결과 세트는 복합 조회 테이블에서 기본 조회 테이블의 의미를 강조합니다. 복합 조회 테이블은 기본 조회 테이블과 0개 이상의 첨부된 조회 테이블로 구성됩니다. 복합 조회 테이블의 기본 조회 테이블은 해당 엔티

티 유형을 판별합니다. 예를 들어, 기본 조회 테이블 TASK가 있는 복합 조회 테이블에는 유형 TASK의 엔티티가 들어 있습니다.

각 엔티티는 조회 테이블 내에서 고유하므로, 각 엔티티는 엔티티 결과 세트 내에서 고유합니다. 엔티티의 고유성은 기본 데이터의 기본 키로 유지됩니다. 예를 들어, TASK 엔티티의 경우 이것은 타스크 ID TKIID입니다.

queryRows 메소드로 리턴되는 행 결과 세트는 기본 데이터베이스 보기 및 테이블에 대해 실행되는 JDBC 조회로 리턴되는 행으로 구성됩니다. 표준 조회 API로 리턴되는 QueryResultSet와 비교될 수 있습니다. 일반적으로 행 수는 조회 테이블에 포함된 엔티티의 수보다 많습니다. 예를 들어, WI.REASON이 조회에서 선택되면 행 결과 세트는 특정 타스크의 중복 항목을 포함할 수 있습니다.

조회 테이블 API 개요

각 조회 테이블 API 메소드는 다음 매개변수를 갖습니다.

- **문자열 queryTableName**: 조회되는 조회 테이블의 이름. 사전정의된 조회 테이블의 경우, 이것은 사전정의된 조회 테이블의 이름입니다. 복합 및 추가 조회 테이블의 경우, 이것은 *prefix.name*입니다.
- **FilterOptions filterOptions**: 결과 세트를 제한하며 정렬 기준을 지정할 수 있는 옵션.
- **AuthorizationOptions authOptions**: 작업 항목이 고려되도록 지정하는 옵션, 다른 사용자를 위한 조회, AdminAuthorizationOptions을 사용하여 실행될 수 있는 관리 조회.
- **목록 매개변수**: 복합 조회 테이블은 필터 및 선택사항 기준의 매개변수를 이용하여 정의될 수 있으며, 그러한 매개변수의 값은 이 인수로 지정됩니다.

FilterOptions

- **구별**: 이 설정은 행 기반 조회가 실행될 때에만 유효합니다. 이것이 참으로 설정되면, 구별 행이 리턴됩니다.
- **로케일**: 로케일은 필터나 선택사항 기준에서 시스템 매개변수로 사용될 수 있습니다 (예: `'LOCALE=$LOCALE'`). 설정되지 않으면, 서버 로케일이 사용됩니다.
- **시간대**: 이것은 사전정의된 조회 테이블 TASK에서의 CREATED와 같이 날짜를 변환하는 데 사용됩니다. 지정하지 않으면, 서버 시간대가 사용됩니다.
- **임계값**: 이것은 리턴되는 행 또는 엔티티의 수를 제한합니다. 엔티티 기반 조회에서는 임계값 설정이 정확하지 않을 수 있습니다.
- **건너뛰기 계수**: 이것은 결과 세트에서 건너뛰는 엔티티의 행 수를 지정합니다.
- **선택된 속성**: 이것은 조회로 검색되는 속성을 지정하는 쉼표로 구분되는 속성 목록입니다. WI.REASON의 예와 같이, 'WI.'가 앞에 붙은 작업 항목 정보, 인스턴스 데이터를 포함하는 사전정의된 조회 테이블에서와 같이 권한이 필수이면, 정의된 속성

이외에 검색될 수 있습니다. 지정되는 선택된 속성이 없는 경우, 조회 테이블에 정의된 모든 속성은 리턴되며 작업 항목 정보는 리턴되지 않습니다.

주: 조회 테이블의 컨텍스트와 조회 테이블 API에서, 열은 일반적으로 속성으로 참조됩니다. 조회 테이블의 컨텐츠는 데이터베이스에 저장되므로, 용어 '열'도 사용될 수 있습니다.

- **조회 조건**: 이것은 결과 세트에서 추가 필터링을 수행합니다. 권한이 필수로 설정되면 조회 테이블에 정의된 속성이 참조될 수 있습니다. WORK_ITEM 조회 테이블에 정의된 열은 접두부 'WI.'를 사용하여서도 참조될 수 있습니다(예를 들어, WI.REASON=REASON_POTENTIAL_OWNER).
- **정렬 속성**: 이것은 정렬 기준을 정의하는 쉼표로 구분되는 속성 목록입니다(예를 들어, CREATED DESC).

AuthorizationOptions

- **모두**: 기본값인 참으로 설정되면, 조회 테이블에서 사용 가능한 경우 모두 작업 항목(스태프 verb 모두)이 고려됩니다.
- **개별**: 기본값인 참으로 설정되면, 조회 테이블에서 사용 가능한 경우 개별 작업 항목(예: 스태프 verb ″사용자″)이 고려됩니다.
- **그룹**: 기본값인 참으로 설정되면, 조회 테이블에서 사용 가능하고 휴먼 타스크 컨테이너에서 사용 가능한 경우 그룹 작업 항목(예: 스태프 verb ″그룹″)이 고려됩니다.
- **상속**: 참으로 설정되면, 상속 작업 항목이 고려됩니다. 예를 들어, 해당 조회 테이블에 대해 조회가 실행되는 경우 프로세스 인스턴스의 관리자는 해당 프로세스 인스턴스에 대해 작성되는 휴먼 타스크 인스턴스를 볼 수 있습니다.

AuthorizationOptions 오브젝트 대신, AdminAuthorizationOptions 오브젝트는 호출자가 J2EE 역할 BPESystemAdministrator을 가진 경우에만 사용 가능한 조회 테이블 API로 패스될 수 있습니다. AdminAuthorizationOptions 클래스는 AuthorizationOptions 클래스에서 파생됩니다. 다음 옵션을 사용할 수 있습니다.

- **onBehalfUser**: 기본값인 널로 설정되고, 조회가 권한을 요구하는 조회 테이블에 대해 실행되면, 이 특정 사용자의 경우 작업 항목에 근거한 권한을 사용하는 결과를 제한하지 않고 실행됩니다. 즉, 조회 테이블에 포함된 모든 오브젝트를 조회가 리턴합니다.
- **onBehalfUser**: 기본값인 널로 설정되고, 조회가 권한을 요구하지 않는 조회 테이블에 대해 실행되면, 인증된 모든 사용자는 조회 테이블의 모든 컨텐츠를 봅니다. 조회의 결과 세트는 AuthorizationOptions 또는 AdminAuthorizationOptions이 사용되었는지 여부와 관계없이 동일합니다.
- **onBehalfUser**: 특정 사용자 이름으로 설정된 경우, 조회는 지정된 사용자 측면에서 실행됩니다.

- **onBehalfUser**: 사전정의된 조회 테이블에 사용되며 템플리트 데이터가 조회되면, onBehalfUser가 널로 설정되어야 합니다.

매개변수

복합 조회 테이블은 필터나 선택사항 기준에 포함된 매개변수로 정의될 수 있습니다. 조회를 실행하는 데 필요한 모든 매개변수는 클래스 com.ibm.bpe.Parameter의 매개변수로서 java.util.List 목록에 포함된 조회 테이블 API로 패스됩니다.

QTCL(Query Table Condition Language)

QTCL(Query Table Condition Language)은 필터 및 선택사항 기준을 지정하기 위해 사용됩니다. 조회 테이블의 속성에 근거한 조건을 지정하려면 명확하게 정의된 이 언어를 사용하십시오. 이 섹션에서는 조회 테이블 API의 QTCL 특정 항목을 설명합니다. 전체 스펙은 WebSphere Business Process Management SupportPacs 사이트를 참조하십시오. PA71 WebSphere Process Server - 조회 테이블 빌더를 찾으십시오. 링크를 액세스하려면, 이 주제의 관련된 참조 섹션을 참조하십시오.

주: 조회 테이블의 컨텍스트와 조회 테이블 API에서, 열은 일반적으로 속성으로 참조됩니다. 그러나 조회 테이블의 컨텐츠는 데이터베이스에 저장되므로, 용어 '열'도 사용될 수 있습니다.

- QTCL 표현식의 부속표현식은 왼쪽 피연산자, 연산 및 오른쪽 피연산자, 또는 연산자 목록으로 구성됩니다. 또한 IS NULL과 같은 단항 연산자도 사용 가능합니다.
- 왼쪽 피연산자는 조회 테이블의 속성 이름입니다.
- 오른쪽 연산자는 왼쪽 연산자 속성에 정의된 상수이거나 리터럴입니다.
- QTCL 표현식은 표현식의 왼쪽에서 유효한 속성을 판별하는 특정 범위에서 실행됩니다. 조회 조건 또는 조회 필터는 조회가 실행되는 조회 테이블의 범위에서 실행됩니다. 표현식 왼쪽에 있는 유효한 속성은 조회 테이블의 속성입니다. 조회 테이블에서 권한이 필수인 경우, 'WI.'가 앞에 붙은 WORK_ITEM 조회 테이블의 속성도 유효합니다.
- 유효한 연산자: `<, >, <>, <=, >=, =, IN, NOT IN, IS NULL, IS NOT NULL, LIKE, IS NOT LIKE`.
- 부속표현식은 잘 알려진 의미가 있는 `AND` 및 `OR`을 사용하여 연결됩니다. 대괄호는 부속표현식을 그룹화하기 위해 사용됩니다. 이것은 사전정의된 TASK 조회 테이블에서 실행되는 조회를 위한 예입니다. `'(STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER) OR (WI.REASON=REASON_OWNER)'`.

계수 조회를 위한 조회 결과 세부사항

조회 테이블 API 메소드 queryEntityCount 및 queryRowCount는 단순 정수 값을 리턴합니다. 규정 오브젝트의 수 검색 시 성능을 위해 구현이 최적화됩니다.

queryEntities 메소드에 의해 리턴되는 EntityResultSet의 조회 결과 세부사항

다음 조회 결과 세부사항은 queryEntities 메소드에 의해 EntityResultSet에 대해 리턴됩니다.

- 조회 테이블 이름은 EntityResultSet에서 검색될 수 있습니다. 이것은 조회가 실행되는 조회 테이블의 이름입니다.
- 엔티티 유형 이름을 검색할 수 있습니다. 이것은 복합 조회 테이블에서 조회를 실행하는 경우 기본 조회 테이블의 이름입니다. 그렇지 않은 경우, 이것은 조회가 실행되는 조회 테이블의 이름입니다.
- EntityInfo 오브젝트를 검색할 수 있습니다. EntityInfo 오브젝트는 EntityResultSet에 포함된 엔티티의 세부사항을 설명합니다. 이것은 엔티티 유형과 마찬가지로 관련 유형인 속성입니다.
- FilterOptions에 정의된 경우, 지정된 순서의 엔티티 목록.
- EntityResultSet의 엔티티 수는 엔티티 목록에서 size() 메소드를 사용하여 검색됩니다.

queryRows 메소드에 의해 리턴되는 RowResultSet의 조회 결과 세부사항

다음 조회 결과 세부사항은 queryRows 메소드에 의해 RowResultSet에 대해 리턴됩니다.

- 조회 테이블 이름은 RowResultSet에서 검색될 수 있습니다. 이것은 조회가 실행되는 조회 테이블의 이름입니다.
- 기본 조회 테이블의 이름을 검색할 수 있습니다. 이것은 복합 조회 테이블에서 조회를 실행하는 경우 기본 조회 테이블의 이름이며, 그렇지 않은 경우 조회가 실행되는 조회 테이블의 이름입니다.
- 속성 및 관련 유형의 목록이 검색될 수 있습니다.
- RowResultSet은 FilterOptions에서 정의된 경우 지정된 순서로 next(), previous(), first() 및 last() 메소드로 탐색될 수 있습니다.
- RowResultSet의 크기가 검색될 수 있습니다.

관련 개념

202 페이지의『사전정의된 조회 테이블』

Business Process Choreographer의 사전정의된 조회 테이블은 TASK 또는 PROCESS_INSTANCE와 같이, 사전정의된 Business Process Choreographer 데이터베이스 보기의 조회 테이블 표시입니다. Business Process Choreographer 데이터베이스에서 간단한 데이터 보기를 제공합니다. 그러나 사전정의된 조회 테이블은 권한, 기능성 및 성능 측면에서 사전정의된 데이터베이스 보기와는 다릅니다.

205 페이지의『추가 조회 테이블』

Business Process Choreographer의 추가 조회 테이블은 일반적으로 외부 데이터베이스 테이블이나 뷰에서 또는 데이터베이스 오브젝트에서 조회 테이블 API로 데

이터를 나타냅니다. 추가 조회 테이블에서, 이 외부 데이터는 비즈니스 프로세스 인스턴스 정보나 휴먼 타스크 정보와 결합될 수 있습니다. 추가 조회 테이블은 조회 테이블 API를 사용하여 직접 조회될 수 있습니다.

206 페이지의『복합 조회 테이블』

Business Process Choreographer의 복합 조회 테이블은 사전정의된 조회 테이블 및 추가 조회 테이블로 구성됩니다. 기존 테이블이나 보기로부터 데이터를 결합시킵니다. 일반적으로, 내가 수행할 작업과 같이, 프로세스 인스턴스 목록이나 타스크 목록에 표시되는 정보를 검색하기 위해 복합 조회 테이블이 사용됩니다.

209 페이지의『조회 테이블 개발』

Business Process Choreographer의 추가 및 복합 조회 테이블은 도구 조회 테이블 빌더를 사용하여 응용프로그램 개발 중 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.

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 J2EE 역할 즉, BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor 또는 TaskSystemMonitor 중 하나가 있는 사용자만이 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.

- J2EE 역할 중 하나가 있는 사용자가 조회를 호출했는지 여부
- 조회한 오브젝트. 오브젝트 특성을 조회할 수 있도록 사전 정의된 데이터베이스 보기가 제공됩니다.
- from 절, 결합 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 제공

해야 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 6.

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 절의 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 구성체는 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 와일드 카드 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. 예를 들어, "TASK_CPROP.STRING_VALUE='d''automatisation'" 과 같습니다.
- 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 선택사항입니다. 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 예

- 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. Select 절에 있는 열을 지정하는 것이 가장 좋습니다.

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 또는 수백만 개의 항목이 있을 수 있으므로 임계값을 항상 지정하는 것이 좋습니다. 예를 들어, 한 번에 소수의 항목만 표시해야 하는 그래픽 사용자 인터페이스에서 임계값 매개변수가 유용합니다. 임계값 매개변수를 알맞게 설정하면 데이터베이스 조회가 더 빨라지고 더 적은 수의 데이터가 서버에서 클라이언트로 전송됩니다.

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.

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

241 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.

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
Integer	getInteger getShort getLong getString getBoolean
Boolean	getBoolean getShort getInteger getLong getString

속성 유형	메소드
byte[]	getBinary

예:

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 비즈니

스 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 판독기이거나 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 알아 보려 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 결합 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 스니핏은 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
```

그러면 조회가 다음 코드 스니펫처럼 됩니다.

```
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 결합 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 작업 로드를 점검하고자 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 보려 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 결합 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 네임 스페이스가 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 네임 스페이스와 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.

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 라이프 사이클 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 멤버쉽 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

199 페이지의『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.

관련 참조

Business Process Choreographer에 대한 데이터베이스 보기
이 참조 정보는 사전 정의된 데이터베이스 보기의 열을 설명합니다.

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.
   - J2EE(Java 2 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. 응용프로그램에서 생성된 스텁을 패키지하십시오.
   a. 프로세스 응용프로그램의 경우, <*install_root*>/ProcessChoreographer/client/bpe137650.jar 파일을 응용프로그램의 EAR(Enterprise Archive) 파일에 패키지하십시오.
   b. 타스크 응용프로그램의 경우, <*install_root*>/ProcessChoreographer/client/task137650.jar 파일을 응용프로그램의 EAR 파일에 패키지하십시오.
   c. JAR 파일을 포함하도록 응용프로그램 모듈의 Manifest 파일에 있는 **Classpath** 매개변수를 설정하십시오.

      응용프로그램 모듈은 J2EE 응용프로그램, 웹 응용프로그램 또는 EJB 응용프로그램일 수 있습니다.
3. 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 J2EE 관리 환경에서 실행되지 않는 경우 클라이언트 응용프로그램의 EAR 파일로 파일을 패키징하십시오.
- 클라이언트 응용프로그램이 관리 J2EE 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 투명하게 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();
   }
```

4. 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.

5. 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 ID, 그룹 멤버쉽 목록이 포함되며 컨텍스트는 호출자가 Business Process Choreographer J2EE 역할 중 하나를 가지는지 여부를 표시합니다. 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.

6. 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 Server에 의해 자동으로(전개 설명자가 TX_REQUIRED 지정)
   - 응용프로그램에 의해 명시적으로. 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.

예

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.
   - J2EE(Java 2 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 ID, 그룹 멤버쉽 목록이 포함되며 컨텍스트는 호출자가 Business Process Choreographer J2EE 역할 중 하나를 가지는지 여부를 표시합니다. 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 Server에 의해 자동으로(전개 설명자가 TX_REQUIRED 지정)
   - 응용프로그램에 의해 명시적으로. 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.

예

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 프로세스 및 타스크 관련 오브젝트 조회

클라이언트 응용프로그램은 비즈니스 프로세스 및 타스크 관련 오브젝트에 대해 작업합니다. 데이터베이스의 비즈니스 프로세스 및 타스크 관련 오브젝트를 조회하여 해당 오브젝트의 특정 특성을 검색할 수 있습니다.

이 태스크 정보

Business Process Choreographer 구성 중에 관계형 데이터베이스는 비즈니스 프로세스 컨테이너와 타스크 컨테이너 모두와 연관됩니다. 데이터베이스는 비즈니스 프로세스 및 타스크 관리에 필요한 모든 템플리트(모델) 및 인스턴스(런타임) 데이터를 저장합니다. SQL과 유사한 구문을 사용하여 이 데이터를 조회합니다.

One-Off 조회를 수행하여 오브젝트의 특정 특성을 검색할 수 있습니다. 자주 사용하는 조회를 저장하고 해당하는 저장된 조회를 응용프로그램에 포함시킬 수도 있습니다.

관련 참조

Business Process Choreographer에 대한 데이터베이스 보기
이 참조 정보는 사전 정의된 데이터베이스 보기의 열을 설명합니다.

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.

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

223 페이지의『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.

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 지정된 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 속해있는 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 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 표에 특정 역할이 취할 수 있는 프로세스 인스턴스의 조치가 표시됩니다.

조치	호출자의 프린시펄 역할		
	독자	시작자	관리자
createMessage	x	x	x

조치	호출자의 프린시펄 역할		
	독자	시작자	관리자
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

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 표에 특정 역할이 취할 수 있는 활동 인스턴스의 조치가 표시됩니다.

조치	호출자의 프린시펄 역할				
	독자	편집자	잠재적 소유자	소유자	관리자
cancelClaim				x	x
claim			x		x
complete				x	x

조치	호출자의 프린시펄 역할				
	독자	편집자	잠재적 소유자	소유자	관리자
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
setVariable					x
transferWorkItem				x (잠재적 소유자 또는 관리자 전용)	x

비즈니스 프로세스 라이프 사이클 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 라이프 사이클을 관리할 수 있습니다.

이 태스크 정보

예제에서는 프로세스에 대한 다음 일반적인 라이프 사이클 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 선택사항입니다.

   ```
   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 선택사항입니다.

   ```
   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 선택사항입니다.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
  ("PROCESS_TEMPLATE.EXECUTION_MODE =
       PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_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 선택사항입니다.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
  ("PROCESS_TEMPLATE.EXECUTION_MODE =
       PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_LONG_RUNNING",
   "PROCESS_TEMPLATE.NAME",
    new Integer(50),
    (TimeZone)null);
```

결과는 이름별로 정렬됩니다. 조회는 장기 실행 프로세스로 시작할 수 있는 처음 50개의 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

호출자는 프로세스 인스턴스의 관리자 또는 비즈니스 프로세스 관리자여야 합니다. 프로세스 인스턴스를 일시중단하려면 실행 또는 실패 상태여야 합니다.

이 태스크 정보

이후에 프로세스에서 사용되는 백엔드 시스템으로 액세스를 구성하는 경우와 같은 상황에 프로세스 인스턴스를 일시중단할 수 있습니다. 프로세스의 전제조건이 만족되면 프로세스 인스턴스를 재개할 수 있습니다. 프로세스 인스턴스의 실패 원인이 된 문제점을 해결하기 위해 프로세스를 일시중단했거나 문제점을 해결한 후 프로세스를 재개할 수도 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. 또한 자율 속성이 하위로 설정된 서브프로세스는 실행 중, 실패 중, 종료 중 또는 보상 중 상태에 있는 경우 일시중단됩니다. 이 프로세스 인스턴스와 연관된 인라인 타스크도 일시중단됩니다. 이 프로세스 인스턴스와 연관된 독립형 타스크는 일시중단되지 않습니다.

   이 상태에서, 시작된 활동은 여전히 완료할 수 있지만 새 활동은 활성화되지 않습니다. 예를 들어, 청구된 상태의 휴먼 타스크 활동은 완료할 수 있습니다.

3. 프로세스 인스턴스를 재개하십시오.

   ```
   process.resume( piid );
   ```

   이 조치를 실행하면 프로세스 인스턴스 및 서브프로세스가 일시중단되기 이전 상태로 돌아갑니다.

비즈니스 프로세스 다시 시작:

완료됨, 중단됨, 실패 또는 보상됨 상태의 프로세스 인스턴스를 다시 시작할 수 있습니다.

시작하기 전에

호출자는 프로세스 인스턴스의 관리자 또는 비즈니스 프로세스 관리자여야 합니다.

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.

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:

때때로 프로세스 관리자 권한이 있는 사용자가 복구 불가능 상태로 인식되는 맨 위 레벨 프로세스 인스턴스를 종료시킬 필요가 있습니다. 프로세스 인스턴스는 처리되지 않은 서브프로세스 또는 활동의 완료를 대기하지 않고 즉시 종료되기 때문에 예외적인 상황에서만 프로세스 인스턴스를 종료해야 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 강제 완료 또는 강제 재시도를 통해 활동을 복구할 수 있습니다.

단일 사용자 워크플로우 처리

일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예를 들어, 온라인 서점에서 책 주문). 이러한 유형의 워크플로우에는 병렬 경로가 없습니다.
completeAndClaimSuccessor API는 해당 유형의 워크플로우 처리를 지원합니다.

이 태스크 정보

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. 이러한 유형의 워크플로우는 사용자 인터페이스 정의가 사용자 인터페이스에서 대화 상자의 흐름을 제어하는 활동과 연관되므로 *페이지 플로우*라고도 합니다.

completeAndClaimSuccessor API는 휴먼 타스크 활동을 완료하며 로그온한 사용자에 대한 동일한 프로세스 인스턴스의 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 작동을 participates로 설정해야 합니다.

비즈니스 플로우 관리자 API와 휴먼 타스크 관리자 API를 모두 사용하는 예제와 이 예제를 비교하십시오.

프로시저

1. 활동 순서에서 첫 번째 활동을 청구하십시오.

```
//
//Query the list of activities that can be claimed by the logged-on user
//
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
                    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);
...
//
//Claim the first activity
//
if (result.size() > 0)
{
 result.first();
   AID aiid = (AID) 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.

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 따를 수 있는 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

291 페이지의『휴먼 타스크를 포함하는 단일 사용자 워크플로우 처리』
일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예를 들어, 온라인 서점에서 책 주문). 이 예제는 일련의 휴먼 타스크 활동(수행할 타스크)으로 서적을 주문하는 일련의 조치를 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 때때로 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

또한 예를 들어 활동이 응답하지 않는 경우 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(예를 들어 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 엔클로징 범위로 전달됩니다. 프로세스는 실패 중 상태가 되고 결국 실패됨 상태가 됩니다.

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 상태에 놓이게 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.

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. 해당 요청(예: sendMessage())을 발행하여 프로세스 템플리트를 인스턴스화하고 시작할 수 있습니다. 프로세스 인스턴스는 서버에서 자동으로 실행됩니다.

표 7. 프로세스 템플리트의 API 메소드

메소드	설명
getProcessTemplate	지정된 프로세스 템플리트를 검색합니다.

표 7. 프로세스 템플리트의 API 메소드 (계속)

메소드	설명
queryProcessTemplates	데이터베이스에 저장되는 프로세스 템플리트를 검색합니다.

프로세스 인스턴스

다음 API 메소드는 프로세스 인스턴스 시작과 관련이 있습니다.

표 8. 프로세스 인스턴스 시작과 관련이 있는 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.
initiate	프로세스 인스턴스를 작성하고 프로세스 인스턴스의 처리를 시작합니다. 장기 실행 프로세스에 이 메소드를 사용하십시오. 또한 해제하거나 잊어버리려는 마이크로플로우에 대해서도 이 메소드를 사용할 수 있습니다.
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 9. 프로세스 인스턴스의 라이프 사이클을 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.

표 9. 프로세스 인스턴스의 라이프 사이클을 제어하는 API 메소드 (계속)

메소드	설명
query	검색 기준에 일치하는 데이터베이스의 특성을 검색합니다.

활동

호출 활동의 경우 프로세스 모델에서 이 활동이 오류 상태에서 계속되도록 지정할 수 있습니다. continueOnError 플래그가 false로 설정되어 처리되지 않은 오류가 발생한 경우 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 10. 활동 인스턴스의 라이프 사이클 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.
query	검색 기준에 일치하는 데이터베이스의 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 11. 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.
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 표시해서 실행하려는 공동 작업 타스크의 이름을 찾으십시오.

타스크의 이름을 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.

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.

- 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(예를 들어, Business Process Choreographer 탐색기를 사용하여) 인스턴스를 작성합니다. 그러나 또한 런타임 시 휴먼 또는 참여 중인 타스크 인스턴스나 템플리트를 작성할 수도 있습니다.

이 태스크 정보

예를 들어, 응용프로그램이 전개될 때, 작업 흐름에 포함된 타스크가 아직 알려지지 않았거나 사용자 그룹 간의 일부 임시 협업을 다루는 타스크가 필요할 때 해당 타스크 정의를 사용할 수 없는 경우 이를 수행할 수 있습니다.

com.ibm.task.api.TaskModel 클래스의 인스턴스를 작성해서 임시 공동 작업 또는 수행할 타스크를 모델화하고 이들을 사용하여 재사용 가능한 타스크 템플리트를 작성하거나 일회 실행 타스크 인스턴스를 직접 작성할 수 있습니다. TaskModel 클래스의 인

스턴스를 작성하려면 팩토리 메소드 세트가 com.ibm.task.api.ClientTaskFactory 팩토리 클래스에서 사용 가능해야 합니다. 런타임의 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 12. 타스크 템플리트의 API 메소드

메소드	설명
getTaskTemplate	지정된 타스크 템플리트를 검색합니다.
createAndCallTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성 및 실행하고 그 결과를 동기적으로 대기합니다.
createAndStartTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성 및 시작합니다.
createTask	지정된 타스크 템플리트에서 타스크 인스턴스를 작성합니다.
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 13. 타스크 인스턴스의 API 메소드

메소드	설명
getTask	타스크 인스턴스를 검색합니다. 타스크 인스턴스는 모든 상태가 될 수 있습니다.
callTask	호출 타스크를 동기식으로 시작합니다.
startTask	이미 작성된 타스크를 시작합니다.
suspend	공동 작업 또는 수행할 타스크를 일시중단합니다.
resume	공동 작업 또는 수행할 타스크를 재개합니다.
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.
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 14. 에스컬레이션 작업에 사용되는 API 메소드

메소드	설명
getEscalation	지정된 에스컬레이션 인스턴스를 검색합니다.

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 15. 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.

타스크에 허용된 조치:

타스크에서 수행 가능한 조치는 타스크가 수행할 타스크, 공동 작업 타스크, 호출 타스크 또는 관리 타스크인지 여부에 따라 달라집니다.

HumanTaskManager 인터페이스에 제공된 모든 조치를 모든 유형의 타스크에 사용할 수는 없습니다. 다음 표는 각 타스크 유형에 대해 수행할 수 있는 조치를 보여줍니다.

	타스크 유형			
조치	수행할 타스크	공동 작업 타스크	호출 타스크	관리 타스크
callTask			X	
cancelClaim	X	X[1]		
claim	X	X[1]		
complete	X	X[1]		X
completeWithFollowOnTask[4]	X	X[1]		
completeWithFollowOnTask[5]		X[3]	X[3]	
createFaultMessage	X	X	X	X
createInputMessage	X	X	X	X
createOutputMessage	X	X	X	X
createWorkItem	X	X[1]	X	X
delete	X[1]	X[1]	X	X[1]
deleteWorkItem	X	X[1]	X	X
getCustomProperty	X	X[1]	X	X
getDocumentation	X	X[1]	X	X
getFaultNames	X	X[1]		
getFaultMessage	X	X[1]	X	
getInputMessage	X	X[1]	X	
getOutputMessage	X	X[1]	X	
getUsersInRole	X	X[1]	X	X
getTask	X	X[1]	X	X
getUISettings	X	X[1]	X	X
resume	X	X[1]		

조치	타스크 유형			
	수행할 타스크	공동 작업 타스크	호출 타스크	관리 타스크
setCustomProperty	X	X[1]	X	X
setFaultMessage	X	X[1]		
setOutputMessage	X	X[1]		
startTask	X[1]	X[1]	X	X
startTaskAsSubtask[6]	X	X[1]		
startTaskAsSubtask[7]		X[3]	X[3]	
suspend	X	X[1]		
suspendWithCancelClaim	X	X[1]		
terminate	X[1]	X[1]	X[1]	
transferWorkItem	X	X[1]	X	X
update	X	X[1]	X	X

참고:

1. 독립형 타스크, 임시 타스크 및 타스크 템플리트 전용
2. 독립형 타스크, 비즈니스 프로세스의 인라인 타스크 및 임시 타스크 전용
3. 독립형 타스크 및 임시 타스크 전용
4. 후속 타스크를 가질 수 있는 타스크 종류
5. 후속 타스크로 사용될 수 있는 타스크 종류
6. 서브타스크를 가질 수 있는 타스크 종류
7. 서브타스크로 사용될 수 있는 타스크 종류

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 이루어집니다.
  - 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 라이프 사이클은 프로세스 인스턴스에서 관리합니다. 프로세스 인스턴스가 삭제되면 타스크 인스턴스도 삭제됩니다. 즉, 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 제한 사항이 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. 템플리트의 완전한 목록을 얻으려면 프로세스 템플리트와 연관된 관리 타스크 템플리트를 읽고 로그온한 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(예를 들어, 온라인 서점에서 책 주문). 이 예제는 일련의 휴먼 타스크 활동(수행할 타스크)으로 서적을 주문하는 일련의 조치를 구현하는 방법을 보여줍니다. 비즈니스 플로우 관리자와 휴먼 타스크 관리자 API는 모두 워크플로우를 처리하는 데 사용됩니다.

이 태스크 정보

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 따를 수 있는 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

257 페이지의『단일 사용자 워크플로우 처리』
일부 워크플로우는 단일 사용자에 의해서만 수행됩니다(예를 들어, 온라인 서점에서 책 주문). 이러한 유형의 워크플로우에는 병렬 경로가 없습니다. completeAndClaimSuccessor API는 해당 유형의 워크플로우 처리를 지원합니다.

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.

API 예외 처리

BusinessFlowManagerService 인터페이스 또는 HumanTaskManagerService 인터페이스의 메소드가 성공적으로 완료되지 않는 경우 오류의 원인을 선언하는 예외가 처리됩니다. 이 예외를 처리하여 호출자에 대한 안내를 제공할 수 있습니다.

이 태스크 정보

그러나 예외의 서브세트만 처리하고 다른 잠재적 예외에 대해서는 일반적인 안내를 제공하는 것이 관례입니다. 모든 특정 예외는 일반 ProcessException 또는 TaskException에서 상속됩니다. 최종 `catch(ProcessException)` 또는 `catch(TaskException)` 문으로 일반 예외를 발견하는 것이 *좋습니다*. 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.

웹 서비스 API 클라이언트 응용프로그램 개발

웹 서비스 API를 통해 비즈니스 프로세스 응용프로그램 및 휴먼 타스크 응용프로그램을 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

이 태스크 정보

Java 웹 서비스 및 Microsoft .NET를 포함한 웹 서비스 클라이언트 환경에서 클라이언트 응용프로그램을 개발할 수 있습니다.

관련 개념

199 페이지의『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.

웹 서비스 컴포넌트 및 제어 순서

많은 클라이언트측 및 서버측 컴포넌트는 웹 서비스 요청 및 응답을 나타내는 제어 순서에 참여합니다.

제어의 일반 순서는 다음과 같습니다.

1. 클라이언트측:
   a. 클라이언트 응용프로그램(사용자가 제공)이 웹 서비스에 대한 요청을 발행합니다.
   b. 프록시 클라이언트(사용자가 제공하기도 하나, 클라이언트측 유틸리티를 사용하여 자동으로 생성할 수 있음)가 SOAP 요청 엔벌로프에 서비스 요청을 랩핑합니다.
   c. 클라이언트측 개발 하부 구조에서 웹 서비스 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 엔벌로프에 랩핑됩니다.
4. 네트워크에서 HTTP 또는 HTTPS를 사용하여 클라이언트측 환경으로 응답을 전송합니다.
5. 다시 클라이언트측:
   a. 클라이언트측 개발 하부 구조에서 SOAP 응답 엔벌로프를 랩핑 해제합니다.
   b. 프록시 클라이언트가 SOAP 응답에서 데이터를 추출하여 클라이언트 응용프로그램으로 전달합니다.

c. 클라이언트 응용프로그램이 필요에 따라 리턴된 데이터를 처리합니다.

웹 서비스 API 개요

웹 서비스 API를 사용하면 웹 서비스를 사용하여 Business Process Choreographer 환경에서 실행 중인 비즈니스 프로세스 및 휴먼 타스크를 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

Business Process Choreographer 웹 서비스 API는 다음 두 가지의 독립적인 웹 서비스 인터페이스(WSDL 포트 유형)를 제공합니다.

- 비즈니스 플로우 관리자 API. 클라이언트 응용프로그램이 마이크로플로우 및 장기 실행 프로세스와 상호작용할 수 있도록 허용합니다. 예를 들어, 다음과 같습니다.
  - 프로세스 템플리트 및 프로세스 인스턴스 작성
  - 기존 프로세스 요청
  - ID별 프로세스 조회

  가능한 조치의 전체 목록은 245 페이지의 『비즈니스 프로세스용 응용프로그램 개발』의 내용을 참조하십시오.
- 휴먼 타스크 관리자 API. 클라이언트 응용프로그램이 다음을 수행할 수 있도록 허용합니다.
  - 타스크 작성 및 시작
  - 기존 타스크 요청
  - 타스크 완료
  - ID별 타스크 조회
  - 타스크 콜렉션 조회

  가능한 조치의 전체 목록은 267 페이지의 『휴먼 타스크용 응용프로그램 개발』의 내용을 참조하십시오.

클라이언트 응용프로그램은 웹 서비스 인터페이스 중 하나 또는 둘 다를 사용할 수 있습니다.

예

다음은 휴먼 타스크 관리자 웹 서비스 API에 액세스하여 참여 중인 휴먼 타스크를 처리하는 클라이언트 응용프로그램에 대한 개략적인 설명입니다.

1. 클라이언트 응용프로그램은 사용자가 작업할 참여 중인 타스크의 목록을 요청하는 WebSphere Process Server에 대한 query 웹 서비스 호출을 발행합니다.
2. 참여 중인 타스크 목록은 SOAP/HTTP 응답 엔벌로프에 리턴됩니다.
3. 그러면 클라이언트 응용프로그램이 참여 중인 타스크 중 하나를 청구하기 위한 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 타스크 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.

1. 비즈니스 프로세스 및 휴먼 타스크의 인터페이스는 XML 기반 RPC용 Java API(JAX-RPC 1.1) 스펙에 정의된 "문서/리터럴 랩핑" 스타일을 사용하여 정의해야 합니다. 이것이 WID로 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

클라이언트 응용프로그램 개발

클라이언트 응용프로그램 개발 프로세스는 여러 단계로 구성됩니다.

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. WebSphere Process Server 클라이언트 CD에서 파일을 복사할 수도 있습니다.
3. 선택한 클라이언트 응용프로그램 개발 환경에서 내보낸 아티팩트를 사용하여 *프록시 클라이언트*를 생성하십시오.
4. 옵션: *헬퍼 클래스*를 생성하십시오. 클라이언트 응용프로그램이 WebSphere 서버에서 구체적 프로세스 또는 타스크와 직접 상호작용하는 경우 헬퍼 클래스가 필요합니다. 그러나 클라이언트 응용프로그램이 조회 발행과 같은 일반 타스크만을 수행하려고 할 경우 필요하지 않습니다.
5. 사용자 클라이언트 응용프로그램을 개발하십시오.

6. 필수 보안 메커니즘을 사용자 클라이언트 응용프로그램에 추가하십시오.

아티팩트 복사

클라이언트 응용프로그램을 작성하려면 WebSphere 환경에서 여러 아티팩트를 복사해야 합니다.

두 가지 방법으로 아티팩트를 가져올 수 있습니다.

- WebSphere Process Server 환경에서 아티팩트를 공개하고 내보냅니다.
- WebSphere Process Server 클라이언트 CD에서 파일을 복사합니다.

서버 환경에서 아티팩트 공개 및 내보내기

웹 서비스 API에 액세스하는 클라이언트 응용프로그램을 개발하려면 먼저 WebSphere 서버 환경에서 여러 아티팩트를 공개하고 내보내야 합니다.

이 태스크 정보

내보내는 아티팩트는 다음과 같습니다.

- 웹 서비스 API를 구성하는 조작 및 포트 유형을 설명하는 WSDL(Web Service Definition Language) 파일
- WSDL 파일의 서비스 및 메소드에서 참조한 데이터 유형 정의가 포함된 XML 스키마 정의(XSD) 파일
- 비즈니스 오브젝트를 설명하는 추가 WSDL 및 XSD 파일. 비즈니스 오브젝트는 WebSphere 서버에서 실행 중인 구체적 비즈니스 프로세스 또는 휴먼 타스크를 설명합니다. 추가 파일은 클라이언트 응용프로그램이 웹 서비스 API를 통해 구체적 비즈니스 프로세스 또는 휴먼 타스크와 직접 상호작용해야 하는 경우에만 필요합니다. 클라이언트 응용프로그램이 조회 발행과 같은 일반 타스크만 수행하는 경우에는 필요하지 않습니다.

해당 아티팩트를 공개한 후 클라이언트 프로그래밍 환경으로 이를 복사해야 합니다. 클라이언트 프로그래밍 환경에서는 이러한 아티팩트를 사용하여 프록시 클라이언트 및 헬퍼 클래스를 생성합니다.

웹 서비스 엔드포인트 주소 지정:

웹 서비스 엔드포인트 주소는 클라이언트 응용프로그램이 웹 서비스 API를 액세스하는 데 지정해야 하는 URL입니다. 클라이언트 응용프로그램에 대한 프록시 클라이언트를 생성하기 위해 내보내는 WSDL 파일에 엔드포인트 주소를 기록합니다.

이 태스크 정보

사용할 웹 서비스 엔드포인트 주소는 다음과 같이 WebSphere 서버 구성에 따라 달라집니다.

- 시나리오 1. 단일 WebSphere 서버가 존재합니다. 지정할 WebSphere 엔드포인트 주소는 서버의 호스트 이름 및 포트 번호입니다(예: **host1:9080**).
- 시나리오 2. WebSphere 클러스터가 몇 개의 서버로 구성됩니다. 지정할 WebSphere 엔드포인트 주소는 웹 서비스 API를 호스트하는 서버의 호스트 이름 및 포트입니다(예: **host2:9081**).
- 시나리오 3. 웹 서버가 프론트 엔드로 사용됩니다. 지정할 WebSphere 엔드포인트 주소는 웹 서버의 호스트 이름 및 포트입니다(예: **host:80**).

기본적으로, 웹 서비스 엔드포인트 주소는 *protocol*://*host*:*port*/*context_root*/*fixed_path* 형식입니다. 여기서:

- *protocol*은 클라이언트 응용프로그램 및 WebSphere 서버 간에 사용할 통신 프로토콜입니다. 기본 프로토콜은 HTTP입니다. 대신에 더 안전한 HTTPS(HTTP over SSL) 프로토콜을 선택할 수 있습니다. HTTPS를 사용하는 것이 좋습니다.
- *host:port*는 웹 서비스 API를 호스트하는 시스템을 액세스하는 데 사용되는 호스트 이름 및 포트 번호입니다. WebSphere 서버 구성(예를 들어, 클라이언트 응용프로그램이 직접 응용프로그램에 액세스하는지 또는 웹 서버 프론트 엔드를 통해 응용프로그램에 액세스하는지 여부)에 따라 값이 달라집니다.
- *context_root*는 컨텍스트 루트에 대해 임의의 값을 선택할 수 있습니다. 그러나 선택하는 값이 WebSphere 셀 내에서 고유해야 합니다. 기본값은 이름 지정 충돌 위험을 제거하는 ″node_server/cluster″ 접미부를 사용합니다.
- *fixed_path*는 `/sca/com/ibm/bpe/api/BFMWS`(비즈니스 플로우 관리자 API의 경우) 또는 `/sca/com/ibm/task/api/HTMWS`(휴먼 타스크 관리자 API의 경우)이며 수정할 수 없습니다.

웹 서비스 엔드포인트 주소는 비즈니스 프로세스 컨테이너 또는 휴먼 타스크 컨테이너 구성 시 초기에 지정됩니다.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.
2. **응용프로그램 → SCA 모듈**을 선택하십시오.

   주: 모든 사용 가능한 엔터프라이즈 응용프로그램 목록을 표시하려면 **응용프로그램 → 엔터프라이즈 응용프로그램**을 또한 선택할 수 있습니다.
3. SCA 모듈 또는 응용프로그램 목록에서 **BPEContainer**(비즈니스 프로세스 컨테이너의 경우) 또는 **TaskContainer**(휴먼 타스크 컨테이너의 경우)를 선택하십시오.
4. **추가 특성** 목록에서 **HTTP 엔드포인트 URL 정보 제공**을 선택하십시오.

5. 목록에서 기본 접두부 중 하나를 선택하거나 또는 사용자 정의 접두부를 입력하십시오. 클라이언트 응용프로그램이 웹 서비스 API를 호스트하는 응용프로그램 서버에 직접 연결할 경우 기본 접두부를 사용하십시오. 그렇지 않으면 사용자 정의 접두부를 지정하십시오.
6. **적용**을 클릭하여 선택한 접두부를 SCA 모듈로 복사하십시오.
7. **확인**을 클릭하십시오. URL 정보가 작업공간에 저장됩니다.

결과

관리 콘솔에서 현재 값을 볼 수 있습니다(예를 들어, 비즈니스 프로세스 컨테이너의 경우: **엔터프라이즈 응용프로그램 → BPEContainer → 전개 설명자 보기**).

내보낸 WSDL 파일에서 `soap:address` 요소의 `location` 속성은 지정된 웹 서비스 엔드포인트 주소를 포함합니다. 예를 들어 다음과 같습니다.

```
<wsdl:service name="BFMWSService">
  <wsdl:port name="BFMWSPort" binding="this:BFMWSBinding">
   <soap:address location=
         "https://myserver:9080/WebServicesAPIs/sca/com/ibm/bpe/api/BFMWS"/>
```

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. 먼저 해당 WSDL 파일을 공개한 후 WebSphere 환경에서 개발 환경(여기서 프록시 클라이언트를 생성하는 데 사용됨)으로 복사해야 합니다.

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

WSDL 파일은 한 번만 공개해야 합니다.

주: WebSphere Process Server 클라이언트 CD가 있을 경우, 대신 해당 CD에서 클라이언트 프로그래밍 환경으로 파일을 직접 복사할 수 있습니다.

비즈니스 프로세스 WSDL 공개:

WSDL 파일을 공개하려면 관리 콘솔을 사용하십시오.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.

2. **응용프로그램** → **SCA 모듈**을 선택하십시오.

   주: 모든 사용 가능한 엔터프라이즈 응용프로그램 목록을 표시하려면 **응용프로그램** → **엔터프라이즈 응용프로그램**을 또한 선택할 수 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 파일 이름은 BPEContainer_WSDLFiles.zip입니다. zip 파일에는 웹 서비스를 설명하는 WSDL 파일 및 WSDL 파일에서 참조된 모든 XSD 파일이 들어 있습니다.

휴먼 타스크 WSDL 공개:

WSDL 파일을 공개하려면 관리 콘솔을 사용하십시오.

프로시저

1. 관리자 권한이 있는 사용자 ID로 관리 콘솔에 로그온하십시오.
2. **응용프로그램** → **SCA 모듈**을 선택하십시오.

   주: 모든 사용 가능한 엔터프라이즈 응용프로그램 목록을 표시하려면 **응용프로그램** → **엔터프라이즈 응용프로그램**을 또한 선택할 수 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 파일 이름은 TaskContainer_WSDLFiles.zip입니다. zip 파일에는 웹 서비스를 설명하는 WSDL 파일 및 WSDL 파일에서 참조된 모든 XSD 파일이 들어 있습니다.

비즈니스 오브젝트 내보내기:

비즈니스 프로세스 및 휴먼 타스크 인터페이스에는 이들을 웹 서비스로 외부에서 액세스할 수 있게 해주는 잘 정의된 인터페이스가 있습니다. 해당 인터페이스가 비즈니스 오브젝트를 참조할 경우, 인터페이스 정의 및 비즈니스 오브젝트를 클라이언트 프로그래밍 환경으로 내보내야 합니다.

이 태스크 정보

클라이언트 응용프로그램이 상호작용해야 하는 각각의 비즈니스 오브젝트에 대해 이 프로시저를 반복해야 합니다.

WebSphere Process Server에서 비즈니스 오브젝트는 비즈니스 프로세스 또는 휴먼 타스크와 상호작용하는 요청, 응답 및 결함 메시지의 형식을 정의합니다. 이러한 메시지에는 복잡한 데이터 유형의 정의도 포함될 수 있습니다.

예를 들어, 휴먼 타스크를 작성하고 시작하려면, 다음 정보 항목을 타스크 인터페이스에 전달해야 합니다.

- 타스크 템플리트 이름
- 타스크 템플리트 네임 스페이스
- 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 환경으로부터 비즈니스 오브젝트를 WSDL(Web Service Definition Language) 및 XSD(XML Schema Definition) 파일로 내보내고, 데이터 유형 정의를 클라이언트 프로그래밍 환경으로 가져온 다음 클라이언트 응용프로그램이 사용할 헬퍼 클래스로 변환하면 가능합니다.

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.

예

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 응용프로그램이 수행하는 모든 *일반* 조작(call, sendMessage 등)에서 <xsd:any> 매개변수를 사용하여 웹 서비스 API로(에서) 전달 및 수신해야 합니다. 해당 사용자 정의 요소는 내보낸 XSD 파일로 생성된 헬퍼 클래스에 의해 클라이언트 응용프로그램에서 작성되고, 직렬화되며, 직렬화 해제됩니다.

클라이언트 CD 파일 사용

WebSphere 서버 환경에서 아티팩트를 내보내는 대신, 클라이언트 응용프로그램을 생성하는 데 필요한 파일을 WebSphere Process Server 클라이언트 CD에서 복사할 수 있습니다.

이러한 경우, 비즈니스 플로우 관리자 API 또는 휴먼 타스크 관리자 API의 기본 웹 서비스 엔드포인트 주소를 수동으로 수정해야 합니다.

클라이언트 응용프로그램이 두 API 모두에 액세스하는 경우, 두 API 모두의 기본 엔드포인트 주소를 편집해야 합니다.

클라이언트 CD에서 파일 복사:

웹 서비스 API에 액세스하는 데 필요한 파일은 WebSphere Process Server 클라이언트 CD에서 가져올 수 있습니다.

프로시저

1. 클라이언트 CD에 액세스하여 ProcessChoreographer₩client 디렉토리를 찾아보십시오.
2. 필요한 파일을 클라이언트 응용프로그램 개발 환경으로 복사하십시오.

   비즈니스 플로우 관리자 API의 경우 다음을 복사하십시오.

   BFMWS.wsdl
   비즈니스 플로우 관리자 웹 서비스 API에서 사용할 수 있는 웹 서비스를 설명합니다. 이 파일에는 엔드포인트 주소가 나와 있습니다.

   BFMIF.wsdl
   비즈니스 플로우 관리자 웹 서비스 API에 있는 각 웹 서비스의 매개변수 및 데이터 유형을 설명합니다.

   BFMIF.xsd
   비즈니스 플로우 관리자 웹 서비스 API에 사용된 데이터 유형을 설명합니다.

   BPCGEN.xsd
   비즈니스 플로우 관리자 및 휴먼 타스크 관리자 웹 서비스 API에서 공통적인 데이터 유형이 포함되어 있습니다.

   휴먼 타스크 관리자 API의 경우 다음을 복사하십시오.

   HTMWS.wsdl
   휴먼 타스크 관리자 웹 서비스 API에서 사용할 수 있는 웹 서비스를 설명합니다. 이 파일에는 엔드포인트 주소가 나와 있습니다.

   HTMIF.wsdl
   휴먼 타스크 관리자 웹 서비스 API에 있는 각 웹 서비스의 매개변수 및 데이터 유형을 설명합니다.

   HTMIF.xsd
   휴먼 타스크 관리자 웹 서비스 API에 사용된 데이터 유형을 설명합니다.

   BPCGEN.xsd
   비즈니스 플로우 관리자 및 휴먼 타스크 관리자 웹 서비스 API에서 공통적인 데이터 유형이 포함되어 있습니다.

   주: BPCGen.xsd 파일은 두 API 모두에서 공통입니다.

다음에 수행할 작업

파일을 복사한 후, BFMWS.wsdl 또는 HTMWS.wsdl 파일의 웹 서비스 API 엔드포인트 주소를 웹 서비스 API를 호스트하는 WebSphere Application Server의 주소로 직접 변경해야 합니다.

웹 서비스 엔드포인트 주소 수동으로 변경:

클라이언트 CD에서 파일을 복사한 경우, WSDL 파일에 지정된 기본 웹 서비스 엔드포인트 주소를 웹 서비스 API를 호스트하는 서버의 주소로 변경해야 합니다.

이 태스크 정보

WSDL 파일을 내보내기 전에 관리 콘솔을 사용하여 웹 서비스 엔드포인트 주소를 설정할 수 있습니다. 그러나 WebSphere Process Server 클라이언트 CD에서 WSDL 파일을 복사한 경우, 기본 웹 서비스 엔드포인트 주소를 수동으로 수정해야 합니다.

사용할 웹 서비스 엔드포인트 주소는 다음과 같이 WebSphere 서버 구성에 따라 달라집니다.

- 시나리오 1. 단일 WebSphere 서버가 존재합니다. 지정할 WebSphere 엔드포인트 주소는 서버의 호스트 이름 및 포트 번호입니다(예: **host1:9080**).
- 시나리오 2. WebSphere 클러스터가 몇 개의 서버로 구성됩니다. 지정할 WebSphere 엔드포인트 주소는 웹 서비스 API를 호스트하는 서버의 호스트 이름 및 포트입니다(예: **host2:9081**).
- 시나리오 3. 웹 서버가 프론트 엔드로 사용됩니다. 지정할 WebSphere 엔드포인트 주소는 웹 서버의 호스트 이름 및 포트입니다(예: **host:80**).

비즈니스 플로우 관리자 API 엔드포인트 변경:

WebSphere Process Server 클라이언트 CD에서 비즈니스 플로우 관리자 API 파일을 복사한 경우, 기본 엔드포인트 주소를 수동으로 편집해야 합니다.

프로시저

1. 클라이언트 CD에서 복사한 파일이 있는 디렉토리를 탐색하십시오.
2. 문서 편집기 또는 XML 편집기에서 BFMWS.wsdl 파일을 여십시오.
3. `soap:address` 요소를 찾으십시오(파일 맨 아래 쪽).
4. 웹 서비스 API를 실행 중인 서버의 HTTP URL이 포함된 `location` 속성 값을 수정하십시오. 다음을 수행하십시오.
   a. `http`를 `https`로 대체하여 더 안전한 HTTPS 프로토콜을 사용할 수도 있습니다.
   b. *localhost*를 웹 서비스 API 서버 엔드포인트 주소의 호스트 이름 또는 IP 주소로 바꾸십시오.
   c. *9080*을 응용프로그램 서버의 포트 번호로 바꾸십시오.
   d. *BPEContainer_N1_server1*을 웹 서비스 API를 실행 중인 응용프로그램의 컨텍스트 루트로 바꾸십시오. 기본 컨텍스트 루트는 다음 요소로 구성됩니다.
      - *BPEContainer*는 응용프로그램 이름입니다.

- *N1*은 노드 이름입니다.
- *server1*은 서버 이름입니다.

e. URL에서 고정된 부분(/sca/com/ibm/bpe/api/BFMWS)은 수정하지 마십시오.

예를 들어, **s1.n1.ibm.com** 서버에서 응용프로그램을 실행 중이고 서버가 **9080** 포트에서 SOAP/HTTP 요청을 승인하는 경우, soap:address 요소를 다음과 같이 수정하십시오.

```
<soap:address location="http://si.n1.ibm.com:9080/
                  BPEContainer_N1_server1/sca/com/ibm/bpe/api/BFMWS"/>
```

휴먼 타스크 관리자 API 엔드포인트 변경:

WebSphere Process Server 클라이언트 CD에서 휴먼 타스크 관리자 API 파일을 복사한 경우, 기본 엔드포인트 주소를 수동으로 편집해야 합니다.

프로시저

1. 클라이언트 CD에서 복사한 파일이 있는 디렉토리를 탐색하십시오.
2. 문서 편집기 또는 XML 편집기에서 HTMWS.wsdl 파일을 여십시오.
3. soap:address 요소를 찾으십시오(파일 맨 아래 쪽).
4. 올바른 엔드포인트 주소가 포함된 location 속성 값을 수정하십시오. 다음을 수행하십시오.
   a. http를 https로 대체하여 더 안전한 HTTPS 프로토콜을 사용할 수도 있습니다.
   b. *localhost*를 웹 서비스 API 서버 엔드포인트 주소의 호스트 이름 또는 IP 주소로 바꾸십시오.
   c. *9080*을 응용프로그램 서버의 포트 번호로 바꾸십시오.
   d. *HTMContainer_N1_server1*을 웹 서비스 API를 실행 중인 응용프로그램의 컨텍스트 루트로 바꾸십시오. 기본 컨텍스트 루트는 다음 요소로 구성됩니다.
      - *HTMContainer*는 응용프로그램 이름입니다.
      - *N1*은 노드 이름입니다.
      - *server1*은 서버 이름입니다.
   e. URL에서 고정된 부분((/sca/com/ibm/task/api/HTMWS)은 수정하지 마십시오.

예를 들어, **s1.n1.ibm.com** 서버에서 응용프로그램을 실행 중이고 서버가 **9081** 포트에서 SOAP/HTTPS 요청을 승인하는 경우, soap:address 요소를 다음과 같이 수정하십시오.

```
<soap:address location="https://si.n1.ibm.com:9081/
                  HTMContainer_N1_server1/sca/com/ibm/task/api/HTMWS"/>
```

Java 웹 서비스 환경에서 클라이언트 응용프로그램 개발

Java 웹 서비스와 호환 가능한 Java 기반 개발 환경을 사용하여 웹 서비스 API에 대한 클라이언트 응용프로그램을 개발할 수 있습니다.

프록시 클라이언트 생성(Java 웹 서비스)

Java 웹 서비스 클라이언트 응용프로그램은 *프록시 클라이언트*를 사용하여 웹 서비스 API와 상호작용합니다.

이 태스크 정보

Java 웹 서비스의 프록시 클라이언트에는 클라이언트 응용프로그램이 호출하여 웹 서비스 요청을 수행하는 여러 Java Bean 클래스가 포함되어 있습니다. 프록시 클라이언트는 서비스 매개변수의 어셈블리를 SOAP 메시지로 처리하고 HTTP를 통해 웹 서비스로 SOAP 메시지를 전송하고 웹 서비스에서 응답을 수신하며 리턴된 데이터를 클라이언트 응용프로그램으로 전달합니다.

그러므로 프록시 클라이언트를 사용하는 경우 기본적으로 클라이언트 응용프로그램은 웹 서비스를 로컬 함수처럼 호출할 수 있습니다.

주: 프록시 클라이언트는 한 번만 생성해야 합니다. 그런 다음 동일한 웹 서비스 API에 액세스하는 모든 클라이언트 응용프로그램이 동일한 프록시 클라이언트를 사용할 수 있습니다.

IBM 웹 서비스 환경에서는 두 가지 방법으로 프록시 클라이언트를 생성할 수 있습니다.

- Rational® Application Developer 또는 WebSphere Integration Developer 통합 개발 환경 사용
- WSDL2Java 명령행 도구 사용

기타 Java 웹 서비스 개발 환경에는 일반적으로 WSDL2Java 도구 또는 독점 클라이언트 응용프로그램 생성 기능 중 하나가 포함되어 있습니다.

Rational Application Developer를 사용한 프록시 클라이언트 생성:

Rational Application Developer 통합 개발 환경을 통해 클라이언트 응용프로그램의 프록시 클라이언트를 생성할 수 있습니다.

시작하기 전에

프록시 클라이언트를 생성하기 전에 먼저 비즈니스 프로세스 또는 휴먼 타스크 웹 서비스 인터페이스를 설명하는 WSDL 파일을 WebSphere 환경(또는 WebSphere Process Server 클라이언트 CD)에서 내보낸 후 클라이언트 프로그래밍 환경으로 복사해야 합니다.

프로시저

1. 해당하는 WSDL 파일을 프로젝트에 추가하십시오.
   - 비즈니스 프로세스의 경우:
     a. 내보낸 파일 BPEContainer_*nodename_servername*_WSDLFiles.zip을 임시 디렉토리에 압축을 푸십시오.
     b. 하위 디렉토리 META-INF를 압축을 푼 디렉토리 BPEContainer_*nodename_servername*.ear/b.jar에서 가져오십시오.
   - 휴먼 타스크의 경우:
     a. 내보낸 파일 TaskContainer_*nodename_servername*_WSDLFiles.zip을 임시 디렉토리에 압축을 푸십시오.
     b. 하위 디렉토리 META-INF를 압축을 푼 디렉토리 TaskContainer_*nodename_servername*.ear/h.jar에서 가져오십시오.

   새 디렉토리 wsdl 및 하위 디렉토리 구조가 프로젝트에 작성됩니다.
2. 웹 서비스 마법사 특성을 수정하십시오.
   a. Rational Application Developer에서 **환경 설정 → 웹 서비스 → 코드 생성 → IBM WebSphere 런타임**을 선택하십시오.
   b. **비랩핑 스타일을 사용하여 WSDL에서 Java 생성** 옵션을 선택하십시오.

   주: 환경 설정 메뉴에서 **웹 서비스** 옵션을 선택할 수 없는 경우에는 먼저 필요한 기능을 **창 → 환경 설정 → Workbench → 기능**의 단계에 따라 사용 가능하게 설정해야 합니다. **웹 서비스 개발자**를 클릭하고 **확인**을 클릭하십시오. 그런 다음 환경 설정 창을 다시 열고 **코드 생성** 옵션을 변경하십시오.
3. 새로 작성된 wsdl 디렉토리에 있는 BFMWS.WSDL 또는 HTMWS.WSDL 파일을 선택하십시오.
4. 마우스 오른쪽 단추를 클릭한 후 **웹 서비스 → 클라이언트 생성**을 선택하십시오.

   나머지 단계를 계속하기 전에 서버가 시작되었는지 확인하십시오.
5. 웹 서비스 창에서 **다음**을 클릭하여 모든 기본값을 승인하십시오.
6. 웹 서비스 선택 창에서 **다음**을 클릭하여 모든 기본값을 승인하십시오.
7. 클라이언트 환경 구성 창에서 다음을 수행하십시오.
   a. **편집**을 클릭하고 웹 서비스 런타임 옵션을 IBM WebSphere로 변경하십시오.
   b. J2EE 버전 옵션을 1.4로 변경하십시오.
   c. **확인**을 클릭하십시오.
   d. **다음**을 클릭하십시오.

8. 이 단계는 비즈니스 프로세스 및 휴먼 타스크 웹 서비스 API를 포함하는 웹 서비스 클라이언트를 생성해야 하는 경우에만 필요합니다. 두 WSDL 파일 모두에 중복 메소드가 있기 때문입니다.
   a. 웹 서비스 프록시 창에서 패키지에 대한 네임 스페이스의 사용자 정의 맵핑 정의를 선택한 다음 **확인**을 클릭하십시오.
   b. 웹 서비스 클라이언트 네임 스페이스-패키지 맵핑 창에서 다음 네임 스페이스 및 패키지를 추가하십시오.

      BFMWS.wsdl:

네임 스페이스	패키지
http://www.ibm.com/xmlns/prod/websphere/business-process/types/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0/Binding	com.ibm.sca.bpe
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.bpe

      HTMWS.wsdl:

네임 스페이스	패키지
http://www.ibm.com/xmlns/prod/websphere/human-task/types/6.0	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/human-task/services/6.0/Binding	com.ibm.sca.task
http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/6.0	com.ibm.sca.task

      겹쳐쓰기를 확인하도록 요청되면 **YesToAll**을 클릭하십시오.
9. **완료**를 클릭하십시오.

결과

프록시, 위치 지정자 및 헬퍼 Java 클래스로 구성된 프록시 클라이언트가 생성되고 클라이언트에 추가됩니다. 전개 설명자도 갱신되었습니다.

WSDL2Java를 사용한 프록시 클라이언트 생성:

WSDL2Java는 프록시 클라이언트를 생성하는 명령행 도구입니다. 프록시 클라이언트는 클라이언트 응용프로그램을 보다 쉽게 프로그래밍할 수 있게 해줍니다.

시작하기 전에

프록시 클라이언트를 생성하기 전에 먼저 비즈니스 프로세스 또는 휴먼 타스크 웹 서비스 API를 설명하는 WSDL 파일을 WebSphere 환경(또는 WebSphere Process Server 클라이언트 CD)에서 내보낸 후 클라이언트 프로그래밍 환경으로 복사해야 합니다.

이 태스크 정보

프로시저

1. WSDL2Java 도구를 사용하여 프록시 클라이언트를 생성하십시오. 유형은 다음과 같습니다.

   wsdl2java *options WSDLfilepath*

   여기서:

   - *options*는 다음과 같습니다.

     -noWrappedOperations (-w)
     랩핑된 조작의 감지를 사용 불가능하게 합니다. 요청 및 응답 메시지에 대한 Java Bean이 생성됩니다.

     주: 기본값이 아닙니다.

     -role (-r)
     클라이언트측 개발을 위한 파일 및 바인딩 파일을 생성하려면 **client** 값을 지정합니다.

     -container (-c)
     사용할 클라이언트측 컨테이너입니다. 올바른 인수는 다음과 같습니다.

     client 클라이언트 컨테이너

     ejb EJB(Enterprise JavaBeans) 컨테이너

     none 컨테이너 없음

     web 웹 컨테이너

     -output (-o)
     생성된 파일을 저장할 폴더입니다.

     WSDL2Java 매개변수에 대한 완전한 목록을 보려면 **-help** 명령행 스위치를 사용하거나 WID/RAD의 WSDL2Java 도구에 대한 온라인 도움말을 참조하십시오.

   - *WSDLfilepath*는 WebSphere 환경에서 내보내거나 클라이언트 CD에서 복사한 WSDL 파일의 경로 및 이름입니다.

   다음 예제는 휴먼 타스크 활동 웹 서비스 API에 대한 프록시 클라이언트를 생성합니다.

   ```
   call wsdl2java.bat -r client -c client -noWrappedOperations
                      -output c:₩ws₩proxyClient  c:₩ws₩bin₩HTMWS.wsdl
   ```

2. 프로젝트에 생성된 클래스 파일을 포함시키십시오.

BPEL 프로세스에 대한 헬퍼 클래스 작성(Java 웹 서비스)

구체적 API 요청(예를 들어, sendMessage 또는 call)은 클라이언트 응용프로그램이 ″문서/리터럴 랩핑″ 스타일 요소를 사용하도록 요구합니다. 클라이언트 응용프로그램은 헬퍼 클래스에 필요한 랩퍼 요소 생성을 돕도록 요구합니다.

시작하기 전에

헬퍼 클래스를 작성하려면 먼저 WebSphere Process Server 환경에서 웹 서비스 API의 WSDL 파일을 내보내야 합니다.

이 태스크 정보

웹 서비스 API의 call() 및 sendMessage() 조작은 WebSphere Process Server에서 BPEL 프로세스와의 상호작용을 허용합니다. call() 조작의 입력 메시지는 프로세스 입력 메시지의 문서/리터럴 랩퍼가 제공될 것으로 예상합니다.

다음과 같은 방법으로 BPEL 프로세스 또는 휴먼 타스크에 대한 헬퍼 클래스를 작성할 수 있습니다.

1. SoapElement 오브젝트를 사용하십시오.

   WebSphere Integration Developer에서 사용할 수 있는 Rational Application Developer 환경에서 웹 서비스 엔진은 JAX-RPC 1.1을 지원합니다. JAX-RPC 1.1에서 SoapElement 오브젝트는 DOM(Document Object Model) 요소를 확장하므로 DOM API를 사용하여 SOAP 메시지를 작성, 읽기, 로드 및 저장할 수 있습니다.

   예를 들어, WSDL 파일에 워크플로우 프로세스 또는 휴먼 타스크에 대한 다음 입력 메시지가 있다고 가정하십시오.

```
<xsd:element name="operation1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input1" nillable="true" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

   WSDL 파일은 프로세스 또는 휴먼 타스크 모듈을 개발할 때 작성됩니다.

   DOM API를 사용하여 클라이언트 응용프로그램에 해당 SOAP 메시지를 작성하는 경우는 다음과 같습니다.

```
SOAPFactory soapfactoryinstance = SOAPFactory.newInstance();
SOAPElement soapmessage = soapfactoryinstance.createElement
                          ("operation1", namespaceprefix, interfaceURI);
SOAPElement inputelement = soapfactoryinstance.createElement("input1");
inputelement.addTextNode( message value);
soapmessage.addChildElement(outputelement);
```

다음 예제는 클라이언트 응용프로그램에 sendMessage 조작의 입력 매개변수를 작성하는 방법을 보여줍니다.

```
SendMessage inWsend = new SendMessage();
inWsend.setProcessTemplateName(processtemplatename);
inWsend.setPortType(porttype);
inWsend.setOperation(operationname);
inWsend.set_any(soapmessage);
```

2. WebSphere Custom Data Binding 기능을 사용하십시오.

   이 기술은 다음 developerWorks 부분에 설명되어 있습니다.

   - 웹 서비스용 사용자 정의 맵핑 기술을 선택하는 방법
   - 복잡한 XML 스키마용 EMF SDO를 사용하여 웹 서비스 개발

    문서 기반 웹 서비스에 대한 패턴 및 전략의 상호 운영성

    선택적 JAX-RPC 1.0/1.1 XML 스키마 유형이 있는 스키마/WSDL에 대한 웹 서비스 지원

클라이언트 응용프로그램 작성(Java 웹 서비스)

클라이언트 응용프로그램은 웹 서비스 API에 요청을 전송하거나 웹 서비스 API에서 응답을 수신합니다. 프록시 클라이언트를 사용하여 통신을 관리하고 헬퍼 클래스를 사용하여 복잡한 데이터 유형을 형식화하여 클라이언트 응용프로그램은 웹 서비스 메소드를 로컬 함수처럼 호출할 수 있습니다.

시작하기 전에

클라이언트 응용프로그램 작성을 시작하려면 먼저 프록시 클라이언트 및 필요한 헬퍼 클래스를 생성하십시오.

이 태스크 정보

웹 서비스 호환 개발 도구를 사용하여(예를 들어, IBM RAD(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. 프록시 클라이언트를 생성하고 Java 헬퍼 클래스를 프로젝트에 추가하십시오.
3. 클라이언트 응용프로그램을 코딩하십시오.
4. 프로젝트를 빌드하십시오.
5. 클라이언트 응용프로그램을 실행하십시오.

예

다음 예는 비즈니스 플로우 관리자 웹 서비스 API를 사용하는 방법을 보여줍니다.

```
// create the proxy
       BFMIFProxy proxy = new BFMIFProxy();
 // prepare the input data for the operation
      GetProcessTemplate iW = new GetProcessTemplate();
      iW.setIdentifier(your_process_template_name);

 // invoke the operation
      GetProcessTemplateResponse oW = proxy.getProcessTemplate(iW);

 // process output of the operation
      ProcessTemplateType ptd = oW.getProcessTemplate();
      System.out.println("getName= " + ptd.getName());
      System.out.println("getPtid= " + ptd.getPtid());
```

보안 추가(Java 웹 서비스)

클라이언트 응용프로그램에서 보안 메커니즘을 구현하여 웹 서비스 통신을 보호해야 합니다.

이 태스크 정보

WebSphere Application Server는 현재 웹 서비스 API에 대한 다음 보안 메커니즘을 지원합니다.

- 사용자 이름 토큰
- LTPA(Lightweight Third Party Authentication)

사용자 이름 토큰 구현:

사용자 이름 토큰 보안 메커니즘은 사용자 이름 및 암호 신임을 제공합니다.

이 태스크 정보

사용자 이름 토큰 보안 메커니즘을 사용하여 다양한 *콜백 핸들러*를 구현할 수 있습니다. 선택사항에 따라 다음이 수행됩니다.

- 클라이언트 응용프로그램을 실행할 때마다 사용자 이름 및 암호를 제공하도록 프롬프트합니다.
- 사용자 이름 및 암호가 전개 설명자에 작성됩니다.

두 경우 모두, 제공된 사용자 이름 및 암호는 해당 비즈니스 프로세스 컨테이너 또는 휴먼 타스크 컨테이너에서 부여한 역할의 사용자 이름 및 암호와 일치해야 합니다.

사용자 이름 및 암호는 요청 메시지 엔벌로프에 캡슐화되며 SOAP 메시지 헤더에 ″분명하게″ 표시됩니다. 그러므로 HTTPS(HTTP over SSL) 통신 프로토콜을 사용하도록

클라이언트 응용프로그램을 구성하는 것이 가장 좋습니다. 그러면 모든 통신이 암호화됩니다. 웹 서비스 API의 엔드포인트 URL 주소 지정 시 HTTPS 통신 프로토콜을 선택할 수 있습니다.

사용자 이름 토큰을 정의하려면 다음을 수행하십시오.

프로시저

1. 보안 토큰을 작성하십시오.
   a. 모듈의 **전개 편집기**를 여십시오.
   b. **WS 확장** 탭을 클릭하십시오.
   c. **서비스 참조** 아래에 다음 웹 서비스 참조가 표시될 수 있습니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService

      표시되는 사항은 프록시 클라이언트를 생성할 때 BFMWS.wsdl(비즈니스 프로세스), HTMWS.wsdl(휴먼 타스크) 또는 둘 모두가 추가되었는지 여부에 따라 다릅니다.
   d. 두 서비스 참조 모두에 대해 다음을 수행하십시오.
      1) **서비스 참조** 중 하나를 선택하십시오.
      2) **요청 생성기 구성** 섹션을 펼치십시오.
      3) **보안 토큰** 하위 섹션을 펼치십시오.
      4) **추가**를 클릭하십시오. 보안 토큰 창이 열립니다.
      5) **이름** 필드에 새 보안 토큰의 이름을 입력하십시오(**UserNameTokenBFM** 또는 **UserNameTokenHTM**).
      6) **토큰 유형** 드롭 다운 목록에서 **Username**을 선택하십시오. (**로컬 이름** 필드는 기본값으로 자동으로 채워집니다.)
      7) **URI** 필드는 공백으로 두십시오. 사용자 이름 토큰에는 URI 값이 필요하지 않습니다.
      8) **확인**을 클릭하십시오.
2. 토큰 생성기를 작성하십시오.
   a. 모듈의 **전개 편집기**를 여십시오.
   b. **WS 바인딩** 탭을 클릭하십시오.
   c. **서비스 참조** 아래에 이전 단계와 동일한 웹 서비스 참조가 표시됩니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService
   d. 두 서비스 참조 모두에 대해 다음을 수행하십시오.
      1) **서비스 참조** 중 하나를 선택하십시오.

2) **보안 요청 생성기 바인딩 구성** 섹션을 펼치십시오.
3) **토큰 생성기** 하위 섹션을 펼치십시오.
4) **추가**를 클릭하십시오. 토큰 생성기 창이 열립니다.
5) **이름** 필드에 새 토큰 생성자의 이름을 입력하십시오 (예: ″UserNameTokenGeneratorBFM″ 또는 ″UserNameTokenGeneratorHTM″).
6) **토큰 생성기 클래스** 필드에서 다음 토큰 생성기 클래스가 선택되었는지 확인하십시오. **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**.
7) **보안 토큰** 드롭 다운 목록에서 이전에 작성한 해당 보안 토큰을 선택하십시오.
8) **값 유형 사용** 선택란을 선택하십시오.
9) **값 유형** 필드에서 **사용자 이름 토큰**을 선택하십시오. (**로컬 이름** 필드는 선택한 **Username 토큰**이 반영되도록 자동으로 채워집니다.)
10) **콜백 핸들러** 필드에 ″com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler″(클라이언트 응용프로그램을 실행할 때 사용자 이름 및 암호에 대한 프롬프트를 표시함) 또는 ″com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler″를 입력하십시오.
11) **NonPromptCallbackHandler**를 선택한 경우 전개 설명자의 해당 필드에 유효한 사용자 이름 및 암호를 지정해야 합니다.
12) **확인**을 클릭하십시오.

관련 정보

IBM WebSphere Developer 기술 저널: WebSphere Application Server V6에 대한 웹 서비스 보안

LTPA 보안 메커니즘 구현:

이미 확립한 보안 컨텍스트에서 클라이언트 응용프로그램을 실행 중인 경우 LTPA(Lightweight Third Party Authentication) 보안 메커니즘을 사용할 수 있습니다.

이 태스크 정보

LTPA 보안 메커니즘은 보안 컨텍스트를 이미 확립한 보안 환경에서 클라이언트 응용프로그램을 실행 중인 경우에만 사용할 수 있습니다. 예를 들어, EJB(Enterprise JavaBeans) 컨테이너에서 클라이언트 응용프로그램을 실행 중인 경우 클라이언트 응용프로그램을 호출하려면 먼저 EJB 클라이언트에 로그인해야 합니다. 그러면 보안 컨텍스트가 확립됩니다. 그런 다음, EJB 클라이언트 응용프로그램이 웹 서비스를 호출하면

LTPA 콜백 핸들러가 보안 컨텍스트에서 LTPA 토큰을 검색하고 이를 SOAP 요청 메시지에 추가합니다. 서버측에서 LTPA 토큰은 LTPA 메커니즘에 의해 처리됩니다.

LTPA 보안 메커니즘을 구현하려면 다음을 수행하십시오.

프로시저

1. WebSphere Integration Developer에서 사용할 수 있는 Rational Application Developer 환경에서 **WS 바인딩 → 보안 요청 생성자 바인딩 구성 → 토큰 생성자**를 선택하십시오.
2. 보안 토큰을 작성하십시오.
   a. 모듈의 **전개 편집기**를 여십시오.
   b. **WS 확장** 탭을 클릭하십시오.
   c. **서비스 참조** 아래에 다음 **웹 서비스 참조**가 표시될 수 있습니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService

      표시되는 사항은 프록시 클라이언트를 생성할 때 BFMWS.wsdl(비즈니스 프로세스), HTMWS.wsdl(휴먼 타스크) 또는 둘 모두가 추가되었는지 여부에 따라 다릅니다.
   d. 두 서비스 참조 모두에 대해 다음을 수행하십시오.
      1) **서비스 참조** 중 하나를 선택하십시오.
      2) **요청 생성기 구성** 섹션을 펼치십시오.
      3) **보안 토큰** 하위 섹션을 펼치십시오.
      4) **추가**를 클릭하십시오. 보안 토큰 창이 열립니다.
      5) **이름** 필드에 새 보안 토큰의 이름을 입력하십시오(**LTPATokenBFM** 또는 **LTPATokenHTM**).
      6) **토큰 유형** 드롭 다운 목록에서 **LTPAToken**을 선택하십시오. (**URI** 및 **로컬 이름** 필드는 기본값으로 자동으로 채워집니다.)
      7) **확인**을 클릭하십시오.
3. 토큰 생성기를 작성하십시오.
   a. 모듈의 **전개 편집기**를 여십시오.
   b. **WS 바인딩** 탭을 클릭하십시오.
   c. **서비스 참조** 아래에 이전 단계와 동일한 웹 서비스 참조가 표시됩니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService
   d. 두 서비스 참조 모두에 대해 다음을 수행하십시오.
      1) **서비스 참조** 중 하나를 선택하십시오.

2) **보안 요청 생성기 바인딩 구성** 섹션을 펼치십시오.
3) **토큰 생성기** 하위 섹션을 펼치십시오.
4) **추가**를 클릭하십시오. 토큰 생성기 창이 열립니다.
5) **이름** 필드에 새 토큰 생성기의 이름을 입력하십시오 (예: ″LTPATokenGeneratorBFM″ 또는 ″LTPATokenGeneratorHTM″).
6) **토큰 생성기 클래스** 필드에서 토큰 생성기 클래스 **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**가 선택되었는지 확인하십시오.
7) **보안 토큰** 드롭 다운 목록에서 이전에 작성한 해당 보안 토큰을 선택하십시오.
8) **값 유형 사용** 선택란을 선택하십시오.
9) **값 유형** 필드에서 **LTPAToken**을 선택하십시오. (**URI** 및 **로컬 이름** 필드는 선택한 **LTPA 토큰**이 반영되도록 자동으로 채워집니다.)
10) **콜백 핸들러** 필드에 ″com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler″를 입력하십시오.
11) **확인**을 클릭하십시오.

결과

런타임 시, **LTPATokenCallbackHandler**는 기존 보안 컨텍스트에서 LTPA 토큰을 검색하고 이를 SOAP 요청 메시지에 추가합니다.

트랜잭션 지원 추가(Java 웹 서비스)

서비스 요청의 일부로 클라이언트 응용프로그램 컨텍스트를 전달하여 서버측 요청 처리가 클라이언트의 트랜잭션에 참여할 수 있도록 Java 웹 서비스 클라이언트 응용프로그램을 구성할 수 있습니다. 이러한 원자적 트랜잭션 지원은 WS-AT(Web Services-Atomic Transaction) 스펙에 정의됩니다.

이 태스크 정보

WebSphere Application Server는 각 웹 서비스 API 요청을 개별적인 원자적 트랜잭션으로 실행합니다. 다음 방법 중 하나로 트랜잭션 지원을 사용하도록 클라이언트 응용프로그램을 구성할 수 있습니다.

- 트랜잭션에 참여하도록 구성합니다. 서버측 요청 처리가 클라이언트 응용프로그램 트랜잭션 컨텍스트에서 수행됩니다. 그런 다음 웹 서비스 API 요청의 실행 및 롤백 도중 문제점이 발생할 경우, 클라이언트 응용프로그램의 요청이 또한 롤백됩니다.
- 트랜잭션 지원을 사용하지 않도록 구성합니다. WebSphere Application Server가 요청을 실행할 새 트랜잭션을 작성하지만, 서버측 요청 처리는 클라이언트 응용프로그램 트랜잭션 컨텍스트로 수행되지 않습니다.

.NET 환경에서 클라이언트 응용프로그램 개발

Microsoft .NET는 웹 서비스를 통해 응용프로그램에 연결하는 강력한 개발 환경을 제공합니다.

프록시 클라이언트 생성(.NET)

.NET 클라이언트 응용프로그램은 *프록시 클라이언트*를 사용하여 웹 서비스 API와 상호작용합니다. 프록시 클라이언트는 웹 서비스 메시징 프로토콜의 복잡성으로부터 클라이언트 응용프로그램을 보호합니다.

시작하기 전에

프록시 클라이언트를 작성하려면, 먼저 다수의 WSDL 파일을 WebSphere 환경으로부터 내보낸 후 클라이언트 프로그래밍 환경으로 복사해야 합니다.

주: WebSphere Process Server 클라이언트 CD가 있는 경우에는 대신 해당 CD에서 파일을 복사할 수 있습니다.

이 태스크 정보

프록시 클라이언트는 C# Bean 클래스 세트를 구성합니다. 각각의 클래스는 단일 웹 서비스에서 사용하는 모든 메소드 및 오브젝트를 포함합니다. 서비스 메소드는 매개변수 어셈블리를 완전한 SOAP 메시지로 처리하며, HTTP를 통해 SOAP 메시지를 전송하고, 웹 서비스의 응답을 수신한 다음 리턴된 데이터를 처리합니다.

주: 프록시 클라이언트는 한 번만 생성해야 합니다. 그런 다음 웹 서비스 API를 액세스 중인 모든 클라이언트가 동일한 프록시 클라이언트를 사용할 수 있습니다.

프로시저

1. WSDL 명령을 사용하여 프록시 클라이언트를 생성하십시오. 유형은 다음과 같습니다.

   wsdl *options WSDLfilepath*

   여기서:

   - *options*는 다음과 같습니다.

     /language

     프록시 클래스를 작성하는 데 사용되는 언어를 지정할 수 있습니다. 기본값은 C#입니다. 또한 언어 인수로 **VB**(Visual Basic), **JS**(JScript) 또는 **VJS** (Visual J#)를 지정할 수 있습니다.

     /output

     해당하는 접미부를 갖는 출력 파일의 이름입니다(예: proxy.cs).

/protocol

프록시 클래스에서 구현되는 프로토콜입니다. **SOAP**는 기본 설정입니다.

WSDL.exe 매개변수에 대한 완전한 목록을 보려면 **/?** 명령행 스위치를 사용하거나 Visual Studio의 WSDL 도구에 대한 온라인 도움말을 참조하십시오.

- *WSDLfilepath*는 WebSphere 환경에서 내보내거나 클라이언트 CD에서 복사한 WSDL 파일의 경로 및 이름입니다.

다음 예제는 휴먼 타스크 관리자 웹 서비스 API에 대한 프록시 클라이언트를 생성합니다.

```
wsdl /language:cs /output:proxyclient.cs c:¥ws¥bin¥HTMWS.wsdl
```

2. 프록시 클라이언트를 DLL(Dynamic Link Library) 파일로 컴파일하십시오.

BPEL 프로세스에 대한 헬퍼 클래스 작성(.NET)

특정 웹 서비스 API 조작은 클라이언트 응용프로그램에 "문서/리터럴" 스타일 랩핑 요소를 사용하도록 요구합니다. 클라이언트 응용프로그램은 헬퍼 클래스에 필요한 랩퍼 요소 생성을 돕도록 요구합니다.

시작하기 전에

헬퍼 클래스를 작성하려면 먼저 WebSphere Process Server 환경에서 웹 서비스 API의 WSDL 파일을 내보내야 합니다.

이 태스크 정보

웹 서비스 API의 call() 및 sendMessage() 조작은 BPEL 프로세스를 WebSphere Process Server 내에서 시작하도록 합니다. call() 조작의 입력 메시지는 BPEL 프로세스 입력 메시지의 문서/리터럴 랩퍼가 제공될 것으로 예상합니다. BPEL 프로세스에 필요한 Bean 및 클래스를 생성하려면 `<wsdl:types>` 요소를 새 XSD 파일로 복사한 후 xsd.exe 도구를 사용하여 헬퍼 클래스를 생성하십시오.

프로시저

1. 아직 이를 수행하지 않은 경우, WebSphere Integration Developer에서 BPEL 프로세스 인터페이스의 WSDL 파일을 내보내십시오.
2. 문서 편집기 또는 XML 편집기에서 WSDL 파일을 여십시오.
3. `<wsdl:types>` 요소의 모든 하위 요소의 컨텐츠를 복사한 후 새 스켈레톤 XSD 파일에 붙여넣으십시오.
4. XSD 파일에서 `xsd.exe` 도구를 실행하십시오.

   call xsd.exe file.xsd /classes /o

   여기서:

file.xsd

변환할 XSD(XML Schema Definition) 파일입니다.

/classes (/c)

지정된 XSD 파일의 컨텐츠에 해당하는 헬퍼 클래스를 생성합니다.

/output (/o)

생성된 파일의 출력 디렉토리를 지정합니다. 이 디렉토리를 생략하면 기본값은 현재 디렉토리입니다.

예를 들어 다음과 같습니다.

call xsd.exe ProcessCustomer.xsd /classes /output:c:₩temp

5. 생성된 클래스 파일을 클라이언트 응용프로그램에 추가하십시오. 예를 들어, Visual Studio를 사용 중인 경우, **프로젝트 → 기존 항목 추가** 메뉴 옵션을 사용하여 이를 수행할 수 있습니다.

예

ProcessCustomer.wsdl 파일이 다음과 같은 컨텐츠를 포함할 경우:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                  xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  name="ProcessCustomer"
                  targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
  <wsdl:types>
    <xsd:schema targetNamespace="http://ProcessTypes/bpel/ProcessCustomer"
                xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
                schemaLocation="xsd-includes/http.com.ibm.bpe.unittest.sca.xsd"/>
      <xsd:element name="doit">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input1" nillable="true" type="bons1:Customer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="doitResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="output1" nillable="true" type="bons1:Customer"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
    <wsdl:message name="doitRequestMsg">
    <wsdl:part element="tns:doit" name="doitParameters"/>
  </wsdl:message>
    <wsdl:message name="doitResponseMsg">
    <wsdl:part element="tns:doitResponse" name="doitResult"/>
  </wsdl:message>
    <wsdl:portType name="ProcessCustomer">
    <wsdl:operation name="doit">
      <wsdl:input message="tns:doitRequestMsg" name="doitRequest"/>
```

```
      <wsdl:output message="tns:doitResponseMsg" name="doitResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

결과 XSD 파일의 컨텐츠:

```
<xsd:schema xmlns:bons1="http://com/ibm/bpe/unittest/sca"
                      xmlns:tns="http://ProcessTypes/bpel/ProcessCustomer"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://ProcessTypes/bpel/ProcessCustomer">
 <xsd:import namespace="http://com/ibm/bpe/unittest/sca"
              schemaLocation="Customer.xsd"/>
 <xsd:element name="doit">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="input1" type="bons1:Customer" nillable="true"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="doitResponse">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="output1" type="bons1:Customer" nillable="true"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

관련 정보

XML 스키마 정의 도구에 관한 Microsoft 문서(XSD.EXE)

클라이언트 응용프로그램 작성(.NET)

클라이언트 응용프로그램은 웹 서비스 API에 요청을 전송하거나 웹 서비스 API에서 응답을 수신합니다. 프록시 클라이언트를 사용하여 통신을 관리하고 헬퍼 클래스를 사용하여 복잡한 데이터 유형을 형식화하여 클라이언트 응용프로그램은 웹 서비스 메소드를 로컬 함수처럼 호출할 수 있습니다.

시작하기 전에

클라이언트 응용프로그램 작성을 시작하려면 먼저 프록시 클라이언트 및 필요한 헬퍼 클래스를 생성하십시오.

이 태스크 정보

.NET 호환 개발 도구(예: Visual Studio .NET)를 사용하여 .NET 클라이언트 응용프로그램을 개발할 수 있습니다. 일반 웹 서비스 API를 호출하는 모든 종류의 .NET 응용프로그램을 빌드할 수 있습니다.

프로시저

1. 새 클라이언트 응용프로그램 프로젝트를 작성하십시오. 예를 들어, **WinFX Windows® 응용프로그램**을 Visual Studio에 작성하십시오.

2. 프로젝트 옵션에서 프록시 클라이언트의 DLL(Dynamic Link Library) 파일에 대한 참조를 추가하십시오. 비즈니스 오브젝트 정의가 포함된 모든 헬퍼 클래스를 프로젝트에 추가하십시오. 예를 들어, Visual Studio에서는 **Project → 기존 항목 추가** 옵션을 사용하여 이를 수행할 수 있습니다.
3. 프록시 클라이언트 오브젝트를 작성하십시오. 예를 들어 다음과 같습니다.

```
HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService service =
    new HTMClient.HTMReference.HumanTaskManagerComponent1Export_HumanTaskManagerHttpService();
```

4. 웹 서비스에서 전송하거나 수신할 메시지에 사용되는 비즈니스 오브젝트 데이터 유형을 선언하십시오. 예를 들어 다음과 같습니다.

```
HTMClient.HTMReference.TKIID id = new HTMClient.HTMReference.TKIID();

ClipBG bg = new ClipBG();
Clip clip = new Clip();
```

5. 특정 웹 서비스 함수를 호출하고 필요한 매개변수를 지정하십시오. 예를 들어, 휴먼 타스크를 작성하고 시작하려면 다음을 지정하십시오.

```
HTMClient.HTMReference.createAndStartTask task =
   new HTMClient.HTMReference.createAndStartTask();
HTMClient.HTMReference.StartTask sTask = new HTMClient.HTMReference.StartTask();

sTask.taskName = "SimpleTask";
sTask.taskNamespace = "http://myProcess/com/acme/task";
sTask.inputMessage = bg;
task.inputTask = sTask;

id = service.createAndStartTask(task).outputTask;
```

6. 원격 프로세스 및 타스크는 지속적 ID(이전 단계의 예제에서 `id`)로 식별됩니다. 예를 들어, 이전에 작성한 휴먼 타스크를 선언하려면 다음을 지정하십시오.

```
HTMClient.HTMReference.claimTask claim = new HTMClient.HTMReference.claimTask();
claim.inputTask = id;
```

보안 추가(.NET)

보안 메커니즘을 클라이언트 응용프로그램에 통합하여 웹 서비스 통신을 보호할 수 있습니다.

이 태스크 정보

이러한 보안 메커니즘에는 사용자 이름 토큰(사용자 이름 및 암호) 또는 사용자 정의 2진 및 XML 기반 보안 토큰이 있습니다.

프로시저

1. WSE(Web Services Enhancements) 2.0 SP3 for Microsoft .NET를 다운로드한 후 설치하십시오. 다음 웹 사이트에서 사용 가능합니다.

   http://www.microsoft.com/downloads/details.aspx?familyid=1ba1f631-c3e7-420a-bc1e-ef18bab66122&displaylang=en
2. 생성된 프록시 클라이언트 코드를 다음과 같이 수정하십시오.

다음 코드를 찾으십시오.

```
public class Export1_MyMicroflowHttpService : System.Web.Services.Protocols.SoapHttpClientProtocol {
```

다음과 같이 변경하십시오.

```
public class Export1_MyMicroflowHttpService : Microsoft.Web.Services2.WebServicesClientProtocol {
```

주: WSDL.exe 도구를 실행하여 프록시 클라이언트를 재생성한 경우 이러한 수정 사항은 유실됩니다.

3. 파일 맨 앞에 다음 행을 추가하여 클라이언트 응용프로그램 코드를 수정하십시오.

```
using System.Web.Services.Protocols;
using Microsoft.Web.Services2;
using Microsoft.Web.Services2.Security.Tokens;
...
```

4. 원하는 보안 메커니즘을 구현하는 코드를 추가하십시오. 예를 들어, 다음 코드는 사용자 이름 및 암호 보호를 추가합니다.

```
string user = "U1";
string pwd = "password";
UsernameToken token =
   new UsernameToken(user, pwd, PasswordOption.SendPlainText);

me._proxy.RequestSoapContext.Security.Tokens.Clear();
me._proxy.RequestSoapContext.Security.Tokens.Add(token);
```

비즈니스 프로세스 및 타스크 관련 오브젝트 조회

웹 서비스 API를 사용하여 해당 오브젝트의 특정 특성을 검색할 수 있는 Business Process Choreographer 데이터베이스의 비즈니스 프로세스 및 타스크 관련 오브젝트를 조회할 수 있습니다.

이 태스크 정보

Business Process Choreographer 데이터베이스는 비즈니스 프로세스 및 타스크 관리에 필요한 템플리트(모델) 및 인스턴스(런타임) 데이터를 저장합니다.

웹 서비스 API를 통해 클라이언트 응용프로그램은 비즈니스 프로세스 및 타스크에 관한 정보를 데이터베이스에서 검색하는 조회를 발행할 수 있습니다.

클라이언트 응용프로그램은 One-Off 조회를 수행하여 오브젝트의 특정 특성을 검색할 수 있습니다. 또한 사용하는 조회를 저장할 수 있습니다. 그런 다음 클라이언트 응용프로그램에서 이러한 저장된 조회를 검색하고 사용합니다.

웹 서비스 API를 사용하는 타스크 관련 오브젝트와 비즈니스 프로세스에 대한 조회

웹 서비스 API의 조회 인터페이스를 사용하여 비즈니스 프로세스 및 타스크에 대한 정보를 확보합니다.

클라이언트 응용프로그램은 SQL 유사 구문을 사용하여 데이터베이스를 조회합니다.

Java 웹 서비스의 예

```
string processTemplateName = "ProcessCustomerLR";
query query1 = new query();
query1.selectClause = "DISTINCT PROCESS_INSTANCE.STARTED, PROCESS_INSTANCE.PIID";
query1.whereClause =
       "PROCESS_INSTANCE.TEMPLATE_NAME = '" + processTemplateName + "'";
query1.orderByClause = "PROCESS_INSTANCE.STARTED";
query1.threshold = null;
query1.timeZone = "UTC"; query1.skipTuples = null;
queryResponse queryResponse1 = proxy.query(query1);
```

데이터베이스에서 검색한 정보는 웹 서비스 API를 통해 *조회 결과 세트*로 리턴됩니다.

예를 들면,

```
QueryResultSetType queryResultSet = queryResponse1.queryResultSet;
if (queryResultSet != null) {
   Console.WriteLine("--> QueryResultSetType");
   Console.WriteLine(" . size= " + queryResultSet.size);
   Console.WriteLine(" . numberColumns= " + queryResultSet.numberColumns);
   string indent = " . ";

   // -- the query column info
   QueryColumnInfoType[] queryColumnInfo = queryResultSet.QueryColumnInfo;
   if (queryColumnInfo.Length > 0) {
      Console.WriteLine();
      Console.WriteLine("= . QueryColumnInfoType size= " + queryColumnInfo.Length);
      Console.Write( " | tableName ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].tableName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | columnName ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         Console.Write( " | " + queryColumnInfo[i].columnName.PadLeft(20) );
      }
      Console.WriteLine();
      Console.Write( " | data type ");
      for (int i = 0; i < queryColumnInfo.Length ; i++) {
         QueryColumnInfoTypeType tt = queryColumnInfo[i].type;
         Console.WriteLine( " | " + tt.ToString());
      }
         Console.WriteLine();
   }
   else {
      Console.WriteLine("--> queryColumnInfo= <null>");
   }

   // - the query result values
   string[][] result = queryResultSet.result;
   if (result !=null) {
      Console.WriteLine();
      Console.WriteLine("= . result size= " + result.Length);
      for (int i = 0; i &lt; result.Length; i++) {
        Console.Write(indent +i );
        string[] row = result[i];
        for (int j = 0; j &lt; row.Length; j++ ) {
           Console.Write(" | " + row[j]);
        }
        Console.WriteLine();
      }
   }
   else {
      Console.WriteLine("--> result= <null>");
```

```
   }
}
else {
   Console.WriteLine("--> QueryResultSetType= <null>");
}
```

조회 함수는 호출자의 권한에 따라 오브젝트를 리턴합니다. 결과 조회 세트에는 호출자가 참조할 권한이 있는 오브젝트의 특성만 포함됩니다.

오브젝트 특성을 조회할 수 있도록 사전 정의된 데이터베이스 보기가 제공됩니다. 프로세스 템플리트의 경우 조회 함수에는 다음 구문이 있습니다.

```
ProcessTemplateData[] queryProcessTemplates
                       (java.lang.String whereClause,
                        java.lang.String orderByClause,
                        java.lang.Integer threshold,
                        java.util.TimeZone timezone);
```

타스크 템플리트의 경우 조회 함수에는 다음 구문이 있습니다.

```
TaskTemplate[] queryTaskTemplates
                (java.lang.String whereClause,
                 java.lang.String orderByClause,
                 java.lang.Integer threshold,
                 java.util.TimeZone timezone);
```

기타 비즈니스 프로세스 및 타스크 관련 오브젝트의 경우 조회 함수에는 다음 구문이 있습니다.

```
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);
```

조회 인터페이스에는 queryAll 메소드도 포함됩니다. 이 메소드를 사용하여 예를 들어 모니터링 목적으로 오브젝트의 연관된 데이터를 모두 검색할 수 있습니다. queryAll 메소드 호출자는 BPESystemAdministrator, BPESystemMonitor, TaskSystemAdministrator 또는 TaskSystemMonitor.method와 같은 J2EE(Java 2 Platform, Enterprise Edition) 역할 중 하나를 가지고 있어야 합니다. 오브젝트의 해당 작업 항목을 사용한 권한 점검은 적용되지 않습니다.

.NET의 예

```
ProcessTemplateType[] templates = null;

   try {
        queryProcessTemplates iW = new queryProcessTemplates();
        iW.whereClause = "PROCESS_TEMPLATE.STATE=PROCESS_TEMPLATE.STATE.STATE_STARTED";
        iW.orderByClause = null;
        iW.threshold = null;
        iW.timeZone = null;

        Console.WriteLine("--> queryProcessTemplates ... ");
        Console.WriteLine("--> query: WHERE " + iW.whereClause + " ORDER BY " +
```

```
    iW.orderByClause + " THRESHOLD " + iW.threshold + " TIMEZONE" + iW.timeZone);

  templates = proxy.queryProcessTemplates(iW);

  if (templates.Length < 1) {
    Console.WriteLine("--> No templates found :-(");
  }
  else {
    for (int i = 0; i < templates.Length ; i++) {
      Console.Write("--> found template with ptid: " + templates[i].ptid);
      Console.WriteLine(" and name: " + templates[i].name);
     /*  ... other properties of ProcessTemplateType ... */
    }
  }
}
catch( Exception e ) {
  Console.WriteLine("exception= " + e);
}
```

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.

공용 저장된 조회 관리
공용 저장된 조회는 시스템 관리자가 작성합니다. 해당 조회는 모든 사용자에게 사용 가능합니다.

기타 사용자에 대한 개인용 저장된 조회 관리
모든 사용자가 개인용 조회를 작성할 수 있습니다. 해당 조회는 조회 소유자 및 시스템 관리자에게만 사용 가능합니다.

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.

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.

관련 개념

199 페이지의『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.

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 들면 다음과 같은 패널이 표시될 수 있습니다.

   ```
   <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. JMS 클라이언트를 J2SE 환경에서 실행하려면 http://www-1.ibm.com/support/docview.wss?uid=swg24012804에서 ″IBM Client for JMS on J2SE with IBM WebSphere Application Server″를 참조하십시오.

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 작성

할 수 있습니다. 이 경우 JNDI(Java Naming and Directory Interface) 찾아보기를 사용하여 연결 팩토리를 검색할 수 있습니다. JNDI 찾아보기 이름은 Business Process Choreographer의 외부 요청 대기열을 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-acknowledgement.
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. 메시지를 전송할 메시지 생성자를 작성하십시오. JNDI 찾아보기 이름은 Business Process Choreographer의 외부 요청 대기열을 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 찾아보기 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 프로세

스 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(예를 들어, 메시지 직렬화를 해제하는 중) 장애가 일찍 발생하면 SCA 내보내기의 수신 대상에 지정된 실패한 전달의 최대 수까지 재시도가 시도됩니다. 실패한 전달의 최대 수에 도달하면 Business Process Choreographer 버스의 시스템 예외 대상에 요청 메시지가 추가됩니다. 그러나 Business Flow Manager의 SCA 컴포넌트에 의한 요청을 실제로 처리하는 중 장애가 발생하면 WebSphere Process Server의 실패한 이벤트 관리 하부 구조에서 실패한 요청 메시지가 핸들됩니다. 즉, 재시도가 예외 상황을 해결하지 못하면 실패한 이벤트 관리 데이터베이스에서 메시지가 끝날 수 있습니다.

메시지 본문

JMS 메시지 본문은 조작의 문서/리터럴 랩퍼 요소를 나타내는 XML 문서가 포함된 문자열입니다.

다음은 유효한 요청 메시지 본문의 간단한 예제입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<_6:queryProcessTemplates xmlns:_6="http://www.ibm.com/xmlns/prod/
      websphere/business-process/services/6.0">
<whereClause>PROCESS_TEMPLATE.STATE IN (1)</whereClause>
</_6:queryProcessTemplates>
```

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 경우에는 아티팩트가 다음과 같습니다.

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

wsa.xsd

다음 방식으로 이 아티팩트를 획득할 수 있습니다.

- WebSphere Process Server 환경에서 아티팩트를 공개하고 내보냅니다.

  이 클라이언트 아티팩트는 *install_root*₩ProcessChoreographer₩client 디렉토리에 있습니다.
- WebSphere Process Server 클라이언트 CD의 *install_root*₩ProcessChoreographer₩client 디렉토리에서 파일을 복사하십시오.

결과

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 들면 다음과 같은 패널이 표시될 수 있습니다.

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
}
```

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. 330 페이지의 『JMS 인터페이스 액세스』에 설명된 대로 JMS 환경을 설정하십시오.
2. 설치된 프로세스 정의 목록을 확보하십시오.
   - queryProcessTemplates를 전송하십시오.
   - 이것은 ProcessTemplate 오브젝트 목록을 리턴합니다.
3. 시작 활동 목록을 확보하십시오(createInstance="yes"로 선택하거나 수신).
   - getStartActivities를 전송하십시오.
   - 이것은 InboundOperationTemplate 오브젝트의 목록을 리턴합니다.
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 생성할 수 있습니다. JSF 클라이언트 생성에 대한 자세한 정보는 WebSphere Integration Developer의 Information Center를 참조하십시오.

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. 전제조건 Business Process Choreographer 탐색기 Java archive(JAR 파일)를 추가하십시오.

   프로젝트의 WEB-INF/lib 디렉토리에 다음 파일을 추가하십시오.

   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar

- bpcjsfcomponents.jar

웹 응용프로그램을 원격 서버에서 전개 중이면 다음 파일도 추가하십시오. Business Process Choreographer API에 원격으로 액세스하려면 이 파일이 필요합니다.

- bpe137650.jar
- task137650.jar

WebSphere Process Server에서 이 파일은 모두 다음 디렉토리에 있습니다.

- Windows 시스템: *install_root*₩ProcessChoreographer₩client
- UNIX®, Linux® 및 i5/OS® 시스템: *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 탐색기 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 목록 컴포넌트, 세부사항 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 핸들링을 위해서는 JSF 구성 파일의 오류 페이지에 대한 탐색 대상 및 오류 Bean을 모두 정의해야 합니다. 오류 Bean의 이름에 대해 BPCError를 사용하고 오류 페이지의 탐색 대상 이름에 대해 error를 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 16. 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 탐색기 컴포넌트가 제공하는 기능이 들어 있습니다.

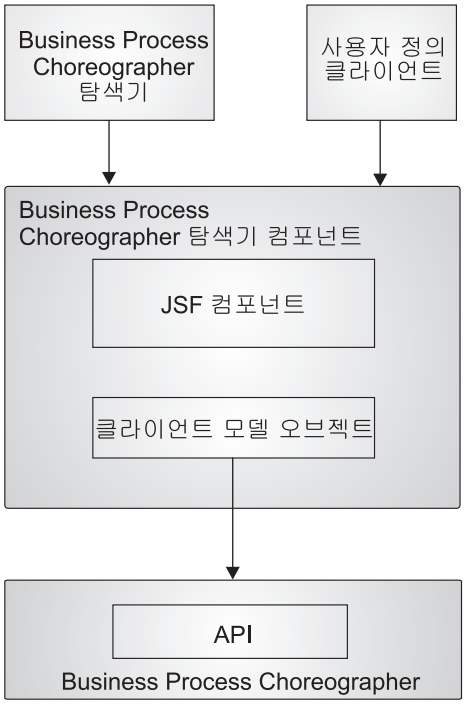
다음에 수행할 작업

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 탐색기 컴포넌트

Business Process Choreographer 탐색기 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 탐색기 및 기타 사용자 정의 클라이언트에 대한 컴포넌트의 관계는 다음 그림으로 표시됩니다.



JSF 컴포넌트

Business Process Choreographer 탐색기 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 임베디드합니다.

- 목록 컴포넌트

  목록 컴포넌트는 응용프로그램 오브젝트의 목록을 테이블에 표시합니다(예: 타스크, 활동, 프로세스 인스턴스, 프로세스 템플리트, 작업 항목 또는 에스컬레이션). 이 컴포넌트에는 연관된 목록 핸들러가 들어 있습니다.
- 세부사항 컴포넌트

  세부사항 컴포넌트는 타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시합니다. 이 컴포넌트에는 연관된 세부사항 핸들러가 들어 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.
- 메시지 컴포넌트

  메시지 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 핸들링

JSF(JavaServer Faces) 컴포넌트는 사전 정의된 관리 Bean, BPCError를 오류 핸들링에 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();
}
```

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 17. 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 목록 컴포넌트 추가

클라이언트 모듈 오브젝트의 목록(예를 들어, 비즈니스 프로세스 인스턴스 또는 타스크 인스턴스)을 표시하려면 Business Process Choreographer 탐색기 목록 컴포넌트를 사용하십시오.

프로시저

1. JSP(JavaServer Pages) 파일에 목록 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 목록 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.

   목록 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 보여줍니다.

```
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

목록 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 목록 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 구

현이 리턴하는 오브젝트 목록에 따라 다르게 표시됩니다. BPCListHandler 클래스의 setQuery 메소드를 사용하여 쿼리를 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 핸들링할 유틸리티를 제공합니다.

BPCListHandler 클래스는 com.ibm.bpc.clientcore.util.User 인터페이스를 사용하여 각 사용자의 시간대와 로케일 정보를 가져옵니다. 목록 컴포넌트는 JSF(JavaServer Faces) 구성 파일에서 관리 Bean 이름 **user**로 구성된 인터페이스의 구현을 기대합니다. 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 핸들링

List 컴포넌트를 사용하면 JSF 응용프로그램의 목록을 표시하면 com.ibm.bpe.jsf.handler.BPCListHandler 클래스가 제공하는 오류 핸들링 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. 목록 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 오

류를 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 핸들링의 한 방법으로 사용합니다.

목록 컴포넌트: 태그 정의

Business Process Choreographer 탐색기 목록 컴포넌트는 테이블에 있는 오브젝트의 목록을 표시합니다(예: 타스크, 활동, 프로세스 인스턴스, 프로세스 템플리트, 작업 항목 및 에스컬레이션).

목록 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 18. bpe:list 속성

속성	필수	설명
buttonStyleClass	아니오	바닥글 영역의 단추를 렌더링하는 데 사용되는 계단식 스타일시트(CSS) 스타일 클래스
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 19. bpe:column 속성

속성	필수	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 세부사항 컴포넌트 추가

타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시하려면 Business Process Choreographer 탐색기 세부사항 컴포넌트를 사용하십시오.

프로시저

1. JSP(JavaServer Pages) 파일에 세부사항 컴포넌트를 추가하십시오.

   <h:form> 태그에 bpe:details 태그를 추가하십시오. bpe:details 태그는 **model** 속성을 포함해야 합니다. bpe:property 태그를 사용하여 세부사항 컴포넌트에 특성을 추가할 수 있습니다.

   다음 예는 타스크 인스턴스에 대한 일부 특성을 표시하기 위해 세부사항 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.

   세부사항 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 행

에 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.

세부사항 컴포넌트: 태그 정의

Business Process Choreographer 탐색기 세부사항 컴포넌트는 타스크, 작업 항목, 활동, 프로세스 인스턴스 및 프로세스 템플리트의 특성을 표시합니다.

세부사항 컴포넌트는 JSF 컴포넌트 태그 `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 20. bpe:details 속성

속성	필수	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

표 20. bpe:details 속성 (계속)

속성	필수	설명
styleClass	아니오	HTML 요소를 렌더링하는 데 사용되는 CSS 클래스

표 21. bpe:property 속성

속성	필수	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 탐색기 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 세부사항 컴포넌트 또는 목록 컴포넌트와 함께 사용됩니다. 일반적으로 이 태그에 지정된 모델은 동일한 페이지의 세부사항 컴포넌트 또는 목록 컴포넌트에 지정된 모델과 동일합니다. 예를 들어 목록 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(예를 들어 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. **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. 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 핸들링합니다.

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 제공자를 사용합니다.

응용프로그램의 사용자 인터페이스의 명령 막대 섹션에서 단추가 눌러지면, 다음과 같이 CommandBar 컴포넌트가 연관된 이벤트를 핸들링합니다.

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 탐색기 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:command로 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 22. bpe:commandbar 속성

속성	필수	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 목록 컴포넌트 또는 세부사항 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 23. bpe:command 속성

속성	필수	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
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 메시지 컴포넌트 추가

JSF(JavaServer Faces) 응용프로그램에서 데이터 오브젝트 및 기본 유형을 표현하려면 Business Process Choreographer 탐색기 메시지 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 메시지 컴포넌트를 추가하십시오.

   `<h:form>` 태그에 `bpe:form` 태그를 추가하십시오. `bpe:form` 태그는 model 속성을 포함해야 합니다.

   다음 예는 메시지 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>
   ```

   메시지 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 메시지 구형 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(예를 들어 메시지가 설정되지 않았기 때문에) 핸들러는 비어 있는 새 메시지를 작성하여 저장합니다. 메시지 컴포넌트는 MyHandler Bean이 제공하는 메시지를 표시합니다.

결과

이제 JSF 응용프로그램은 데이터 오브젝트 및 기본 유형을 표현할 수 있는 JavaServer 페이지를 포함할 수 있습니다.

메시지 컴포넌트: 태그 정의

Business Process Choreographer 탐색기 메시지 컴포넌트는 JSF(JavaServer Faces) 응용프로그램에 `commonj.sdo.DataObject` 오브젝트 및 기본 유형(예: 정수 및 문자열)을 표현합니다.

메시지 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 24. 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

표 24. bpe:form 속성 (계속)

속성	필수	설명
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 탐색기 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 역할에 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 탐색기가 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 탐색기에서 유효성이 검증됩니다. 유효성 검증은 제공된 특성의 유형 및 브라우저에서 사용되는 로케일을 기반으로 합니다. 입력의 유효성을 검증할 수 없는 경우, 같은 페이지가 다시 나타나며 messageValidationErrors request 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 탐색기에 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 탐색기의 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 탐색기 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 탐색기에 표현됩니다.

사용자 정의 JSP 단편

사용자 정의 JSP(JavaServer Pages) 단편은 HTML 양식 태그에 임베디드됩니다. 런타임 시, Business Process Choreographer 탐색기는 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 탐색기에서 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 하부 구조를 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.

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. APIEventHandlerPlugin3 인터페이스를 구현하거나 APIEventHandler 구현 클래스를 확장하는 클래스를 작성하십시오. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.
   - APIEventHandlerPlugin3 인터페이스를 사용할 경우, APIEventHandlerPlugin3 인터페이스 및 APIEventHandlerPlugin 인터페이스의 모든 메소드를 구현해야 합니다.
   - APIEventHandler 구현 클래스를 확장할 경우, 필요한 메소드를 대체하십시오.

   이 클래스는 J2EE(Java 2 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 2 서비스 프로바이더 인터페이스 스펙을 준수합니다.

   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.APIEventHandlerPlugin3 인터페이스를 구현할 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 무시됩니다.

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.

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 J2EE(Java 2 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.

여러 J2EE 응용프로그램에서 헬퍼 클래스를 사용하는 경우, 공유 라이브러리로 등록한 별도의 JAR 파일로 해당 클래스를 패키지할 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 2 서비스 프로바이더 인터페이스 스펙을 준수합니다.

   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 `MyEventAPIHandler`이고 `com.customer.plugins` 패키지에 위치할 경우, 구성 파일의 첫 번째 행에 `com.customer.plugins.MyAPIEventHandler`와 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.

API 이벤트 핸들러 및 공고 이벤트 플러그인 설치

API 이벤트 핸들러 또는 공고 이벤트 핸들러 플러그인을 사용하려면 타스크 컨테이너가 플러그인에 액세스할 수 있도록 먼저 플러그인을 설치해야 합니다.

이 태스크 정보

단일 J2EE(Java 2 Enterprise Edition) 응용프로그램에서만 플러그인을 사용할지 또는 여러 응용프로그램에서 플러그인을 사용할지 여부에 따라 플러그인을 설치하는 방법이 다릅니다.

플러그인을 설치하려면 다음 단계 중 하나를 완료하십시오.

- 단일 J2EE 응용프로그램에서 사용할 플러그인을 설치하십시오.

  플러그인 JAR 파일을 응용프로그램 EAR 파일에 추가하십시오. WebSphere Integration Developer의 전개 설명자 편집기에서 기본 EJB(Enterprise JavaBeans) 모듈의 J2EE 응용프로그램에 대한 프로젝트 유틸리티 JAR 파일로 플러그인의 JAR 파일을 설치하십시오.
- 여러 J2EE 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 구성원을 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 여러 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.

- 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.

사용자 조회 결과를 사후 처리하는 플러그인 작성, 설치 및 실행

개인 분석은 특정 역할(예를 들어, 타스크의 잠재적 소유자)에 지정된 사용자의 목록을 리턴합니다. 개인 분석에서 리턴한 개인 조회 결과를 변경하는 플러그인을 작성할 수 있습니다. 예를 들어, 워크로드 밸러싱을 개선하려면, 조회 결과에서 이미 워크로드가 많은 사용자를 제거하는 플러그인을 작성할 수도 있습니다.

이 태스크 정보

하나의 사후 처리 플러그인만 있을 수 있으며, 이는 플러그인이 모든 타스크의 사용자 조회 결과를 처리해야 함을 의미합니다. 플러그인은 사용자를 추가 또는 제거하거나, 사용자 또는 그룹 정보를 변경할 수 있습니다. 또한 결과 유형을 변경할 수 있습니다(예를 들어, 사용자 목록에서 그룹으로 또는 전체 사용자로).

개인 분석이 완료된 후 플러그인이 실행되기 때문에 기밀성 또는 보안을 유지해야 하는 모든 규칙을 이미 적용했어야 합니다. 플러그인은 (HTM_REMOVED_USERS 맵 키에서) 개인 분석 중에 제거된 사용자에 대한 정보를 수신합니다. 플러그인에서 이 컨텍스트 정보를 사용하여 기밀성이나 보안 규칙을 보존해야 합니다.

개인 조회 결과의 사후 처리를 구현하려면 StaffQueryResultPostProcessorPlugin 인터페이스를 사용합니다. 인터페이스에는 타스크, 에스컬레이션, 타스크 템플리트 및 응용프로그램 컴포넌트에 대한 조회 결과를 수정하는 메소드가 있습니다.

다음 단계를 완료하여 개인 조회 결과를 사후 처리할 플러그인을 작성하십시오.

프로시저

1. StaffQueryResultPostProcessorPlugin 인터페이스를 구현하는 클래스를 쓰십시오.

   이 클래스는 J2EE(Java 2 Enterprise Edition) 엔터프라이즈 응용프로그램의 컨텍스트에서 실행됩니다. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.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.

   인터페이스에서 모든 메소드를 구현해야 합니다. 이들 메소드는 특정 타스크, 템플리트, 타스크 또는 에스컬레이션 역할의 사용자 지정 기준과 관련된 정보를 포함합니다.

   - 사용자 지정 기준 정의는 유형 Map의 **context** 매개변수 항목으로 지정됩니다. 이 정보를 액세스하려면 다음과 같이 진행하십시오.

     ```
     Map pacAsMap = (Map) context.get("HTM_VERB");

     // to retrieve the name of the PAC
     String pacName = (String) pacAsMap.get("HTM_VERB_NAME");

     // to retrieve the PAC parameter names
     Set paramNames = pacAsMap.keySet();

     // to retrieve the value of a specific parameter
     String paramValue = (String) pacAsMap.get(paramName);
     ```

- 사용자 지정 기준 매개변수 값으로 지정되는 교체 변수는 유형 Map의 **context** 매개변수 항목입니다. 이 정보를 액세스하려면 다음과 같이 진행하십시오.

```
Object replVarObj = pacAsMap.get(replVarName);
if (replVarObj instanceof String)
     String replVarValue = (String) replVarObj;
if (replVarObj instanceof String[])
     String[] replVarValues = (String[]) replVarObj;
```

- 예를 들어, 가상 구성원 관리자 사용자 디렉토리를 액세스하여, 사용자 분석 중 사용자 디렉토리를 액세스하여 작성되는 StaffQueryResult 오브젝트.

  StaffQueryResult 오브젝트에는 사용자 분석 중 검색되는 사용자 항목에 대한 정보가 들어 있습니다. 자세한 정보는 StaffQueryResultPostProcessorPlugin 인터페이스의 Javadoc 참조 정보를 참조하십시오.

- 사용자 분석에 의해 명시적으로 포함된 사용자 목록은 유형 Map의 **context** 매개변수의 항목으로 포함됩니다. 이 정보를 액세스하려면 다음과 같이 진행하십시오.

```
String[] removedUserIDs = (String[]) context.get("HTM_REMOVED_USERS");
```

다음 예제는 SpecialTask라는 타스크의 편집자 역할을 변경할 수 있는 방법을 표시합니다.

```
public StaffQueryResult processStaffQueryResult
                        (StaffQueryResult originalStaffQueryResult,
                                          Task task,
                                          int  role,
                                          Map  context)
{
  StaffQueryResult newStaffQueryResult = originalStaffQueryResult;
  StaffQueryResultFactory staffResultFactory =
                          StaffQueryResultFactory.newInstance();
  if (role == com.ibm.task.api.WorkItem.REASON_EDITOR &&
      task.getName() != null &&
      task.getName().equals("SpecialTask"))
  {
      UserData user = staffResultFactory.newUserData
                           ("SuperEditor",
                             new Locale("en-US"),
                            "SuperEditor@company.com");
      ArrayList userList = new ArrayList();
      userList.add(user);

      newStaffQueryResult = staffResultFactory.newStaffQueryResult(userList);
   }
   return(newStaffQueryResult);
}
```

2. 플러그인 클래스 및 헬퍼 클래스를 JAR 파일에 어셈블하십시오.

   JAR을 공유 라이브러리로 사용 가능하게 설정하고 타스크 컨테이너와 연관시킬 수 있습니다. 이 방식에서는, 사용자의 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 2 서비스 프로바이더 인터페이스 스펙을 준수합니다.

a. `com.ibm.task.spi.`*`plug-in_name`*`StaffQueryResultPostProcessorPlugin` 이름(여기서 *plug-in_name*은 플러그인의 이름)의 파일을 작성하십시오.

   예를 들어, 플러그인이 `MyHandler`이고 com.ibm.task.spi.StaffQueryResultPostProcessorPlugin 인터페이스를 구현하는 경우 구성 파일의 이름은 `com.ibm.task.spi.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`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 `MyEventAPIHandler`이고 `com.customer.plugins` 패키지에 위치할 경우, 구성 파일의 첫 번째 행에 `com.customer.plugins.MyAPIEventHandler`와 같은 항목이 포함되어야 합니다. 플러그인을 로드하는 데 사용할 수 있는 서비스 프로바이더 구성 파일 및 개인 조회 결과를 처리하는 플러그인이 포함된 설치 가능한 JAR 파일이 생성됩니다.

4. 플러그인을 설치하십시오.

   개인 조회 결과에 대한 사후 처리 플러그인은 하나만 있을 수 있습니다. 플러그인을 공유 라이브러리로 설치해야 합니다.

   a. 플러그인의 WebSphere Application Server 공유 라이브러리를 정의하십시오. Business Process Choreographer가 구성된 서버 또는 클러스터의 범위에 대해 공유 라이브러리를 정의하십시오. 그런 다음 이 공유 라이브러리를 TaskContainer 응용프로그램과 연관시키십시오. 이 단계는 한 번만 수행해야 합니다.

   b. 플러그인 JAR 파일을 서버 또는 클러스터 구성원을 호스트하는 각각의 영향을 받는 WebSphere Process Server 설치 시 사용할 수 있도록 설정하십시오.

5. 플러그인을 등록하십시오.

   a. 관리 콘솔에서 휴먼 타스크 관리자의 사용자 정의 특성 페이지로 이동하십시오.

      독립형 환경에서는 **서버 → 응용프로그램 서버 →** ***server_name***, 또는 Business Process Choreographer가 클러스터에서 구성된 경우 **서버 → 클러스터 →** ***cluster_name***을 클릭하십시오. **비즈니스 통합** 아래에서 **휴먼 타스크 관리자**를 선택하십시오. **추가 특성** 아래에서 **사용자 정의 특성**을 선택하십시오.

   b. **Staff.PostProcessorPlugin** 이름의 사용자 정의 특성 및 플러그인에 부여한 이름의 값(이 예제에서는 `MyHandler`)을 추가하십시오.

이제 사용자 조회 결과 사후 처리에 플러그인을 사용할 수 있습니다. JAR 파일을 변경하는 경우, 공유 라이브러리의 파일을 바꾸고 서버를 다시 시작하십시오.

6. 플러그인을 실행하십시오. 사용자 지정 및 사용자 대체 둘 다가 실행된 이후 사후 처리 플러그인이 호출됩니다. StaffQueryResultPostProcessorPlugin 인터페이스에 의해 지정된 정보로 플러그인이 호출됩니다.

제 2 부 응용프로그램 전개

제 14 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 설치하기 위한 프로세스는 WebSphere Application Server Network Deployment, 버전 6 Information Center의 『응용프로그램 개발 및 전개』에 설명된 프로세스와 비슷합니다. 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 이동할 수 있도록 돕는 파일을 설명합니다. 이 파일과 파일에 포함된 내용을 이해하여 모듈을 올바로 설치하는 것이 중요합니다.

라이브러리 및 Jar 파일 개요

모듈에서 종종 라이브러리에 있는 아티팩트를 사용합니다. 아티팩트 및 라이브러리는 모듈을 전개할 때 식별하는 JAR(Java Archive) 파일에 포함됩니다.

모듈을 개발하는 동안 다양한 모듈에서 사용할 수 있는 특정 자원 또는 컴포넌트를 식별했을 수 있습니다. 이 자원 또는 컴포넌트는 서버에 이미 전개된 라이브러리에 있는 기존 오브젝트 또는 모듈을 개발하는 동안 작성한 오브젝트일 수 있습니다. 이 주제에서는 응용프로그램 설치에 필요한 라이브러리 및 파일에 대해 설명합니다.

라이브러리 개념

라이브러리에는 WebSphere Integration Developer의 다중 모듈에서 사용되는 오브젝트 또는 자원이 들어 있습니다. 아티팩트는 JAR, RAR(resource archive) 또는 WAR(Web service archive) 파일 형식으로 작성됩니다. 이런 아티팩트 중 일부는 다음을 포함합니다.

- 확장자가 .wsdl인 인터페이스 또는 웹 서비스 설명자 파일
- 확장자가 .xsd인 비즈니스 오브젝트 XML 스키마 정의 파일
- 확장자가 .map인 비즈니스 오브젝트 맵 파일

• 확장자가 .rel 및 .rol인 관계 및 역할 정의 파일

모듈에 아티팩트가 필요한 경우 메모리에 로드되지 않았으면 서버는 아티팩트를 EAR 클래스 경로에서 찾아 로드합니다. 해당 지점에서 아티팩트에 대한 요청은 바뀔 때까지 이 사본을 사용합니다. 그림 23에서는 응용프로그램이 컴포넌트 및 연관 라이브러리를 포함하는 방법을 보여줍니다.

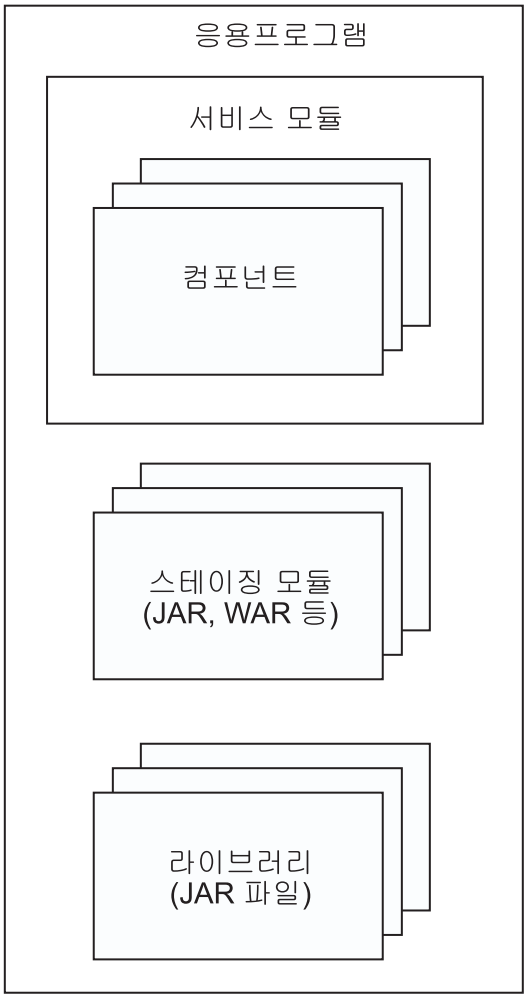


그림 23. 모듈, 컴포넌트 및 라이브러리의 관계

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, JAR 파일과 같

은 기타 라이브러리, 웹 서비스 아카이브(WAR) 파일, 자원 아카이브(RAR) 파일, 스테이징 라이브러리(EJB(Enterprise Java Bean)) JAR 파일 또는 기타 아카이브가 필요하며 serviceDeploy 명령을 사용하여 설치 가능한 EAR 파일을 작성해야 합니다.

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*EJB
- *myService*EJBClient
- *myService*Web

주: serviceDeploy 명령은 서비스에 WSDL 포트 유형 서비스가 포함된 경우에만 *myService* 웹 스테이징 모듈을 작성합니다.

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 포함하는 실행 중인 응용프로그램을 중지하고 다시 설치하십시오.
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 프로세스에 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 380 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 중프로덕션 서버에서 모듈 설치 중에서 설명한 대로 모듈을 프로덕션 시스템에 설치하십시오.

클러스터에 서비스 응용프로그램 설치에 대한 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 15 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 응용프로그램 서버 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 응용프로그램 서버 인스턴스의 응용프로그램 설치가 트리거됩니다. 이 응용프로그램 서버 인스턴스는 전개 대상이거나 대상의 일부입니다. 전개 대상이 여러 클러스터 구성원이 있는 클러스터인 경우 이 클러스터의 클러스터 구성원에서 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 서비스 개발이 프로세스나 타스크에 대한 전개 코드를 생성할 때 각 프로세스 컴포넌트 또는 타스크 컴포넌트는 하나의 세션 엔터프라이즈 Bean으로 맵핑됩니다. 모든 전개 코드는 엔터프라이즈 응용프로그램(EAR) 파일에 패키지됩니다. 또한 엔터프라이즈 응용프로그램의 설치 중 각 프로세스에 대해 이 프로세스의 Java 코드를 나타내는 Java 클래스가 생성되어 EAR 파일에 임베디드됩니다. 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. 새 엔터프라이즈 응용프로그램을 설치하면 설치되는 템플리트의 버전이 다음과 같이 판별됩니다.

- 템플리트의 이름 및 대상 네임 스페이스가 아직 존재하지 않는 경우 새 템플리트가 설치됩니다.
- 템플리트 이름 및 대상 네임 스페이스가 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 하부 구조에 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.

타스크[24]: 데이터 소스 갱신

```
//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
타스크[25]: 세트 참조 갱신

// 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
스키마 접두부:  []:
// 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
테이블 접두부:  []:// 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 타스크를 포함하는 응용프로그램을 설치 제거하려면 다음 전제조건을 충족시켜야 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 구성원이 실행 중이어야 합니다. 클러스터 구성원이 Business Process Choreographer 데이터베이스에 대한 액세스 권한을 가져야 합니다.
- 응용프로그램이 관리 서버에 설치된 경우에는 Deployment Manager 및 이 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.
- **-force** 옵션을 사용하는 경우를 제외하고, 임의의 상태에 있는 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없습니다.

이 태스크 정보

비즈니스 프로세스 또는 휴먼 타스크를 포함하는 엔터프라이즈 응용프로그램을 설치 제거하려면 다음 조치를 수행하십시오.

프로시저

1. 응용프로그램을 전개한 시스템에서 데이터베이스, 각 클러스터에 대한 하나 이상의 응용프로그램 서버, 독립형 서버가 실행 중인지 확인하십시오.

   Network Deployment 환경에서 Depoyment Manager, 모든 관리 독립형 응용프로그램 서버 및 최소 하나의 응용프로그램 서버가 응용프로그램이 설치되는 각 클러스터에 대해 실행 중이어야 합니다.
2. 응용프로그램에 비즈니스 프로세스 인스턴스 또는 휴먼 타스크 인스턴스가 없는지 확인하십시오.

   필요한 경우 관리자는 Business Process Choreographer 탐색기를 사용하여 프로세스 또는 타스크 인스턴스를 삭제할 수 있습니다. 응용프로그램을 설치 제거할 때 자동으로 중지되므로 프로세스 및 타스크 템플리트를 중지시킬 필요가 없습니다.
3. 응용프로그램을 중지하고 설치 제거하십시오.
   a. 관리 콘솔 탐색 패널에서 **응용프로그램 → 엔터프라이즈 응용프로그램**을 클릭하십시오.
   b. 설치 제거하려는 응용프로그램을 선택하고 **중지**를 클릭하십시오.

      프로세스 인스턴스 또는 타스크 인스턴스가 응용프로그램에 여전히 존재하면 이 단계가 실패합니다.
   c. 설치 제거하려는 응용프로그램을 다시 선택하고 **설치 제거**를 클릭하십시오.
   d. **저장**을 클릭하여 변경사항을 저장하십시오.

결과

응용프로그램이 설치 제거됩니다.

관리 명령을 사용하여 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치 제거

관리 명령은 비즈니스 프로세스 또는 휴먼 타스크를 포함하는 응용프로그램을 제거하는 데 사용되는 관리 콘솔을 대체합니다.

시작하기 전에

비즈니스 프로세스 또는 휴먼 타스크를 포함하는 응용프로그램을 설치 제거하려면 다음 전제조건을 충족시켜야 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 구성원이 실행 중이어야 합니다. 클러스터 구성원이 Business Process Choreographer 데이터베이스에 대한 액세스 권한을 가져야 합니다.
- 응용프로그램이 관리 서버에 설치된 경우에는 Deployment Manager 및 이 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.
- **-force** 옵션을 사용하는 경우를 제외하고, 임의의 상태에 있는 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없습니다.

그리고 관리 보안이 사용되는 경우에는 사용자 ID가 관리자 또는 운영자 권한을 가지는지 확인하십시오. **-force** 옵션을 사용하려면, 관리자 권한이 필요합니다.

관리 클라이언트 연결로의 서버 프로세스가 실행 중인지 확인하십시오. 관리 클라이언트가 자동으로 서버 프로세스에 연결되는지를 확인하려면 명령 옵션으로 `-conntype NONE` 옵션을 사용하지 마십시오.

이 태스크 정보

다음 단계에서는 bpcTemplates.jacl 스크립트를 사용하여 비즈니스 프로세스 템플리트 또는 휴먼 타스크 템플리트를 포함하는 응용프로그램을 설치 제거하는 방법에 대해 설명합니다.

응용프로그램을 설치 제거하기 전에, 예를 들면 Business Process Choreographer 탐색기를 사용하여 응용프로그램의 템플리트와 연관된 타스크 인스턴스 또는 프로세스 인스턴스를 삭제할 수 있습니다. 또한 bpcTemplates.jacl 스크립트와 함께 **-force** 옵션을 사용하여 템플리트와 연관된 인스턴스를 삭제하고 템플리트를 중지하고 템플리트를 설치 제거하는 작업을 한 번에 수행할 수 있습니다.

주의:
-force 옵션은 모든 프로세스 인스턴스와 타스크 인스턴스 데이터를 삭제하므로 이 옵션을 사용할 때는 주의해야 합니다.

프로시저

1. Business Process Choreographer 샘플 디렉토리로 이동하십시오.

   Windows 플랫폼에서는 다음을 입력하십시오.

   ```
   cd install_root¥ProcessChoreographer¥admin
   ```

   Linux, UNIX 및 i5/OS 플랫폼에서 다음을 입력하십시오.

   ```
   cd install_root/ProcessChoreographer/admin
   ```

2. 템플리트를 중지하고 해당 응용프로그램을 설치 제거하십시오.

   Windows 플랫폼에서는 다음을 입력하십시오.

```
install_root¥bin¥wsadmin -f bpcTemplates.jacl
                              [-user user_name]
                              [-password user password]
                              -uninstall application_name
                              [-force]
```

Linux, UNIX 및 i5/OS 플랫폼에서 다음을 입력하십시오.

```
install_root/bin/wsadmin -f bpcTemplates.jacl
                              [-user user_name]
                              [-password user password]
                              -uninstall application_name
                              [-force]
```

여기서:

user_name
관리 보안이 사용 가능한 경우, 인증을 위한 사용자 ID입니다.

user_password
관리자 보안이 사용 가능한 경우, 인증을 위한 사용자 암호입니다.

application_name
설치 제거할 응용프로그램의 이름을 입력하십시오.

결과

응용프로그램이 설치 제거됩니다.

제 16 장 어댑터 및 설치

어댑터를 사용하여 사용자의 응용프로그램과 엔터프라이즈 정보 시스템의 다른 컴포넌트를 통신할 수 있습니다.

어댑터 설치 프로세스에 대한 설명은 WebSphere Integration Developer Information Center의 어댑터 구성 및 사용에 있습니다.

제 17 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 J2EE 모듈을 다른 대상 서버에 전개하려고 하는 중입니다.

   중요사항: 설치가 되지 않고 응용프로그램에 서비스가 있는 경우 응용프로그램을 다시 설치하기 전에 장애 이전에 작성된 모든 SIBus 대상이나 J2C 활성화 스펙을 제거해야 합니다. 이러한 아티팩트를 가장 쉽게 제거하는 방법은 장애가 발생한 후에 **저장 > 모두 버리기**를 클릭하는 것입니다. 실수로 변경사항을 저장한 경우에는 SIBus 대상과 J2C 활성화 스펙을 수동으로 제거해야 합니다. (자세한 사항은 관리 섹션에서 SIBus 대상 삭제 및 J2C 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.

J2C 활성화 스펙 삭제

시스템은 서비스가 포함된 응용프로그램 설치 시 J2C 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/ActivationSpec에서 **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 스펙이 필요 없는 경우 J2C 활성화 스펙을 삭제하십시오.

프로시저

1. 삭제할 활성화 스펙을 찾으십시오.

   스펙은 자원 어댑터 패널에 들어 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 J2C 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

SIBus 대상은 응용프로그램에서 서비스를 사용 가능하게 하는 연결입니다. 대상을 제거해야 하는 경우가 있습니다.

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/Customer에서 **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 들어 있는 응용프로그램을 다시 설치하려면 먼저 응용프로그램에서 항목을 제거해야 합니다(해당 Information Center의 관리 섹션에 있는 J2C 활성화 스펙 삭제 참조).

프로시저

1. 관리 콘솔에 로그인하십시오.
2. SCA 시스템 버스에서 대상을 표시하십시오.

   서비스 통합 > 버스를 클릭하여 패널을 탐색하십시오.
3. SCA 시스템 버스 대상을 선택하십시오.

   화면에서 **SCA.SYSTEM.*cellname*.Bus**를 클릭하십시오. 여기서 *cellname*은 삭제하려는 대상이 포함된 모듈이 들어 있는 셀의 이름입니다.
4. 제거하려는 모듈과 일치하는 모듈 이름이 들어 있는 대상을 삭제하십시오.
   a. 해당 대상 옆에 있는 선택란을 클릭하십시오.
   b. **삭제**를 클릭하십시오.

결과

패널에 남아 있는 대상만 표시됩니다.

다음에 수행할 작업

해당 대상을 작성한 모듈과 관련된 J2C 활성화 스펙을 삭제하십시오.

제 3 부 부록

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트 문자 세트(DBCS) 정보에 관한 라이센스 조회는 한국 IBM 지적재산국에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 이 책을 ″현 상태대로″ 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

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

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함)간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 원하는 프로그램 라이센스 사용자는 다음 주소로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 문서에 기술된 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 라이센스가 있는 모든 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약 또는 모든 동등한 계약 하에서 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 단계의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을 통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 비IBM 제품을 반드시 테스트하지 않았으므로, 이들 제품과 관련된 성능의 정확성, 호환성 또는 기타 주장에 대해서는 확인할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM이 제시하는 방향 또는 의도에 관한 모든 언급은 특별한 통지 없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이센스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 IBM에 비용을 지불하지 않고 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건 하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이러한 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 진술하지 않습니다.

이러한 샘플 프로그램이나 파생 작업의 각 사본이나 부분에는 다음과 같은 저작권 표시가 있어야 합니다: ⓒ (회사명) (연도). 이 코드의 일부는 IBM Corp.의 샘플 프로그램에서 파생됩니다. ⓒ Copyright IBM Corp. _enter the year or years_. All rights reserved.

이 정보를 소프트카피로 확인하는 경우에는 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보는 본 프로그램을 사용하는 응용프로그램 소프트웨어 작성을 돕기 위해 제공됩니다.

귀하는 범용 프로그래밍 인터페이스를 통해 본 프로그램 툴의 서비스를 제공하는 응용프로그램 소프트웨어를 작성할 수 있습니다.

그러나 본 정보에는 진단, 수정 및 성능 조정 정보도 포함되어 있습니다. 진단, 수정 및 성능 조정 정보는 응용프로그램 소프트웨어의 디버그를 돕기 위해 제공된 것입니다.

경고: 본 진단, 수정 및 조정 정보는 변경될 수 있으므로 프로그램 인터페이스로서 사용될 수 없습니다.

상표 및 서비스표

IBM, IBM 로고 및 ibm.com은 미국 또는 기타 국가에서 사용되는 International Business Machines Corporation의 상표 또는 등록상표입니다. 이들 및 기타 IBM 상표 용어가 이 정보에서 처음 나타날 때 상표 기호(® 또는 ™)와 함께 표시되는 경우, 이들 기호는 이 정보가 출판된 당시 IBM이 소유한 미국 등록 또는 관습법 상표를 표시합니다. 또한 이러한 상표는 기타 국가에서 등록상표 또는 일반 법적 상표입니다. IBM 상표의 현재 목록은 www.ibm.com/legal/copytrade.shtml 웹 페이지의 ″저작권 및 상표 정보″에 있습니다.

Java는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.

기타 회사, 제품 및 서비스 이름은 해당 회사의 상표 또는 서비스표입니다.

이 제품은 Eclipse 프로젝트에서 개발한 소프트웨어를 포함합니다. (http://www.eclipse.org 웹 사이트 참조)

멀티플랫폼용 IBM WebSphere Process Server, 버전 6.2