WebSphere 멀티플랫폼용 Process Server

버전 6.1.0

모듈 개발 및 전개

WebSphere® 멀티플랫폼용 Process Server

버전 **6.1.0**

모듈 개발 및 전개

주!

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

2008년 2월 1일

이 개정판은 새 개정판에서 별도로 명시하지 않는 한 멀티플랫폼용 WebSphere Process Server의 버전 6, 릴리스 1, 수정판 0(제품 번호: 5724-L01) 및 모든 후속 릴리스와 수정판에 적용됩니다.

이 문서에 대한 의견을 보내려면 doc-comments@us.ibm.com으로 전자 우편 메시지를 전송하십시오. 여러분의 의견을 기대하고 있습니다.

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

목차

그림

표

제 1 부 응용프로그램 개발

제 1 장 모듈 개발 개요

모듈은 WebSphere® Process Server 응용프로그램의 기본 전개 단위입니다. 모듈은 응용프로그램에서 사용되는 하나 이상의 컴포넌트 라이브러리 및 스테이징 모듈을 포함합니다. 컴포넌트는 다른 서비스 컴포넌트를 참조할 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 두 가지 유형의 서비스 모듈, 즉, 비즈니스 서비스에 대한 모듈과 중개 모듈을 지원합니다. 비즈니스 서비스에 대한 모듈은 프로세스의 로직을 구현합니다. 중개 모듈을 사용하면 서비스 호출을 대상이 이해하는 형식으로 변환하여 응용프로그램 간 통신을 할 수 있으며 목표로 요청을 전달하고 그 결과를 오리지네이터에게 리턴할 수 있습니다.

다음 섹션에서는 WebSphere Process Server에서 모듈을 구현하고 갱신하는 방법에 대해 설명합니다.

컴포넌트의 개요

컴포넌트는 재사용 가능한 비즈니스 로직을 캡슐화하는 기본 빌딩 블록입니다. 서비스 컴포넌트는 인터페이스, 참조 및 구현과 연관됩니다. 인터페이스는 서비스 컴포넌트와 호출 컴포넌트 간의 계약을 정의합니다. WebSphere Process Server를 사용하여, 서비스 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 정의하십시오.
2. 서비스 컴포넌트에서 사용되는 비즈니스 오브젝트를 정의, 수정 또는 조작하십시오.
3. 해당 인터페이스를 통해 서비스 컴포넌트를 정의 또는 수정하십시오.

   주: 서비스 컴포넌트는 해당 인터페이스를 통해 정의됩니다.
4. 선택적으로 서비스 컴포넌트를 내보내거나 가져오십시오.

5. 컴포넌트를 사용하는 모듈을 설치하는 데 사용할 EAR 파일을 작성하십시오. 서비스 컴포넌트를 사용하는 서비스 모듈을 설치하기 위해 EAR 파일을 작성하려면 WebSphere Integration Developer 또는 serviceDeploy 명령에서 내보내기 EAR 기능을 사용하여 파일을 작성하십시오.

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 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.

타스크 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 개발을 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 호출을 계속 수행하십시오.
3. 클라이언트 컴포넌트와 대상 컴포넌트를 연결하십시오.

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.

타스크 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 나열합니다.
3. 구현을 정의하는 클래스를 개발하십시오.
   - 컴포넌트가 장기 실행되는 경우(또는 비동기적 실행) 4단계를 계속 진행하십시오.
   - 컴포넌트가 장기 실행되지 않는 경우(또는 동기적으로 실행) 5단계를 계속 진행하십시오.
4. 비동기 구현을 개발하십시오.

   중요사항: 비동기 컴포넌트 인터페이스에서 joinsTransaction 특성을 `true`로 설정할 수 없습니다.
   a. 동기 서비스 컴포넌트를 나타내는 인터페이스를 정의하십시오.
   b. 서비스 컴포넌트의 구현을 정의하십시오.
   c. 6단계를 계속 진행하십시오.
5. 동기 구현을 개발하십시오.
   a. 동기 서비스 컴포넌트를 나타내는 인터페이스를 정의하십시오.
   b. 서비스 컴포넌트의 구현을 정의하십시오.
6. 컴포넌트 인터페이스 및 구현을 .java 확장자를 갖는 파일로 저장하십시오.
7. 서비스 모듈 및 필수 자원을 JAR 파일로 패키지하십시오.

   이 Information Center의 『프로덕션 서버로 모듈 전개』에서 7 - 9단계의 설명을 참조하십시오.
8. serviceDeploy 명령을 실행하여 응용프로그램을 포함하는 설치 가능한 EAR 파일을 작성하십시오.
9. 서버 노드에 응용프로그램을 설치하십시오.
10. 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;
 }
}
```

이 예는 비동기 서비스 컴포넌트를 개발합니다. 코드의 첫 번째 섹션은 getQuote 메소드를 구현하는 서비스 컴포넌트에 대한 인터페이스를 정의합니다.

```
public interface StockQuote {

 public float getQuote(String symbol);
}
```

다음 섹션은 StockQuote와 연관된 클래스의 구현입니다.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

코드의 다음 섹션은 비동기 인터페이스 StockQuoteAsync를 구현합니다.

```
public interface StockQuoteAsync {

 // deferred response
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);
```

```
 // callback
 public Ticket getQuoteAsync(String symbol, StockQuoteCallback callback);
}
```

이 섹션은 onGetQuoteResponse 메소드를 정의하는 StockQuoteCallback 인터페이스입니다.

```
public interface StockQuoteCallback {

 public void onGetQuoteResponse(Ticket ticket, float quote);
}
```

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.

타스크 정보

컴포넌트는 컴포넌트 이름을 사용하고 컴포넌트가 예상하는 데이터 유형을 전달하여 WebSphere Process Server 클러스터에서 사용할 수 있는 모든 서비스 컴포넌트를 사용할 수 있습니다. 이 환경에서 컴포넌트를 호출하는 것은 필요한 컴포넌트에 대한 참조를 찾아 작성하는 것을 포함합니다.

주: 모듈에 있는 컴포넌트는 같은 모듈에 있는 컴포넌트를 호출할 수 있는데, 이를 모듈 내 호출이라고 합니다. 제공 컴포넌트에 있는 인터페이스를 내보내고 이 인터페이스를 호출 컴포넌트에 가져옴으로써 외부 호출(모듈 간 호출)을 구현합니다.

중요사항: 호출 모듈이 실행 중인 서버와 다른 서버에 있는 컴포넌트를 호출할 때 이들 서버에 추가 구성을 수행해야 합니다. 필요한 구성은 컴포넌트가 비동기 또는 동기로 호출되는지에 따라 다릅니다. 이 경우 Application Server를 구성하는 방법은 관련 타스크에 설명되어 있습니다.

프로시저

1. 호출 모듈에서 필요한 컴포넌트를 결정하십시오.

   컴포넌트의 인터페이스 이름 및 인터페이스에 필요한 데이터 유형을 참고하십시오.
2. 데이터 오브젝트를 정의하십시오.

입력 또는 리턴이 Java™ 클래스일 수는 있지만 서비스 데이터 오브젝트가 최적입니다.

3. 컴포넌트를 찾으십시오.
   a. ServiceManager 클래스를 사용하여 호출 모듈에 사용 가능한 참조를 얻으십시오.
   b. locateService() 메소드를 사용하여 컴포넌트를 찾으십시오.

      컴포넌트에 따라 인터페이스는 WSDL(Web Service Descriptor Language) 포트 유형 또는 Java 인터페이스가 될 수 있습니다.
4. 컴포넌트를 동기 또는 비동기적으로 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
       StockQuoteAsync sQuote =
      (StockQuoteAsync)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuoteAsync(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.

타스크 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.

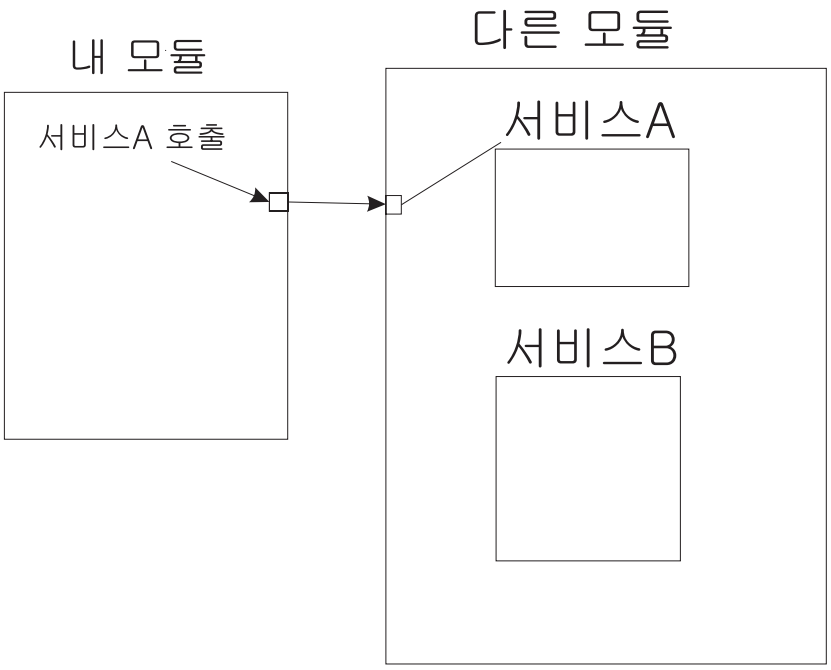


그림 1. 단순 호출 모델

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 응용 프로그램이 하나의 서비스를 호출하는 경우와 비슷하게 됩니다. 모듈A, 모듈B 및 모듈 C는 모두 CalculateFinalCost를 호출합니다.

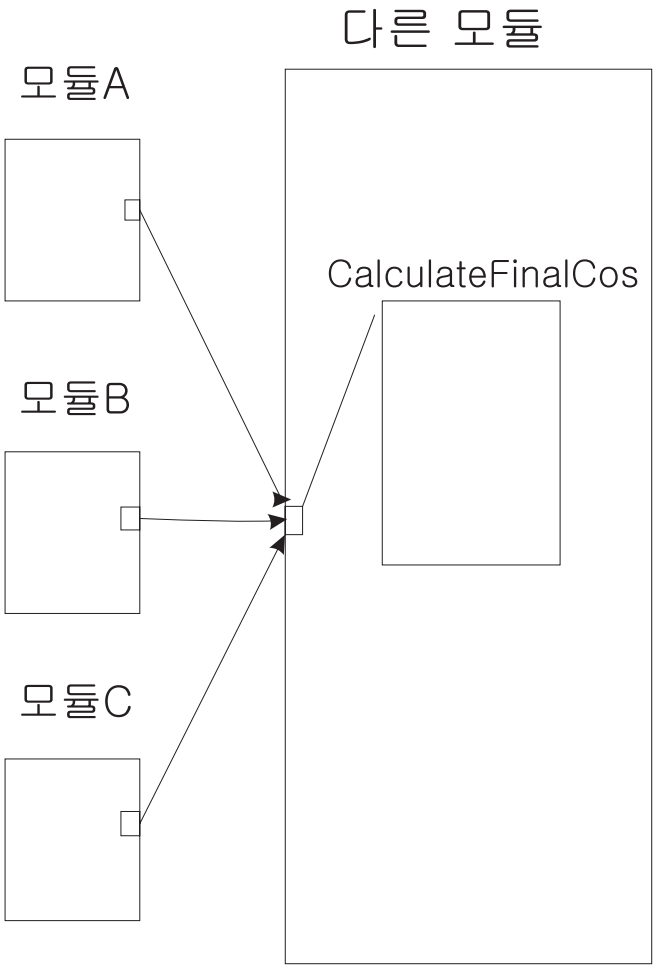


그림 2. 여러 개의 응용프로그램이 하나의 서비스를 호출

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 모델을 작성합니다.

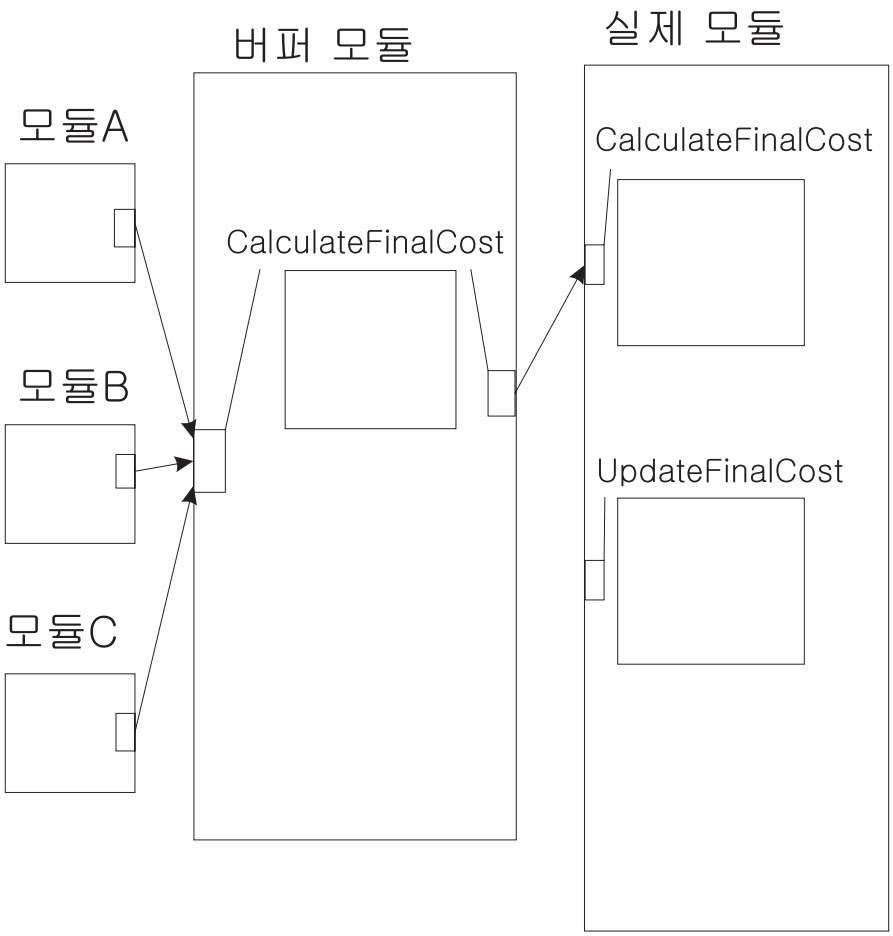


그림 3. UpdateCalculateFinal을 호출하는 분리 호출 모델

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

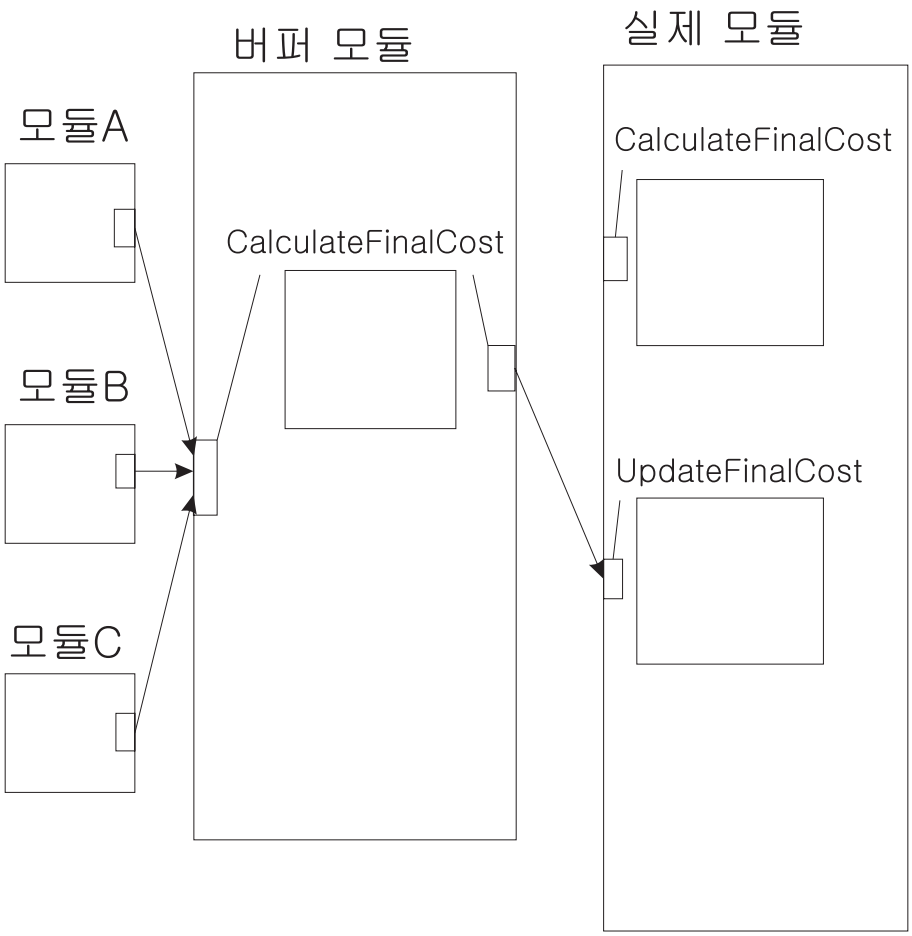


그림 4. UpdatedCalculateFinal을 호출하는 분리 호출 모델

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.

관련 태스크

대상 변경

참조 대상을 변경하면 응용프로그램을 다시 컴파일하여 다시 설치하지 않고도 컴포넌트에 고급 기능의 장점을 활용하는 탄력성을 응용프로그램에 제공할 수 있습니다.

HTTP 바인딩

HTTP 바인딩은 SCA(Service Component Architecture) 연결성을 HTTP에 제공하기 위해 설계되었습니다. 따라서 기존 또는 새로 개발된 HTTP 응용프로그램을 서비스 지향 아키텍처(SOA) 환경에 참여시킬 수 있습니다.

또한 SCA 런타임 환경의 네트워크는 기존 HTTP 하부 구조에서 통신할 수 있습니다.

HTTP 바인딩은 다음과 같은 여러 HTTP 기능을 제공합니다.

- 메시지는 HTTP 형식 및 메시지 헤더 정보를 보존하는 방식으로 중개 컴포넌트에 표시됩니다. 이는 HTTP 응용프로그램 프로그래머, 사용자 및 관리자에게 보다 친숙한 보기를 제공합니다.
- 기존 데이터 바인딩 프레임워크는 HTTP 규칙을 위해 확장되었으며, SCA 메시지 및 HTTP 메시지 헤더와 본문 간 맵핑을 제공합니다.
- 일련의 일반적인 HTTP 기능을 지원하도록 가져오기 및 내보내기를 구성할 수 있습니다.
- HTTP 가져오기 또는 내보내기를 포함하는 SCA 모듈을 설치하면 HTTP와의 연결성을 허용하도록 런타임 환경이 자동으로 구성됩니다.

HTTP 가져오기 및 내보내기 작성에 대한 지시사항은 Information Center의 **WebSphere Integration Developer** > **통합 응용프로그램 개발** > **HTTP 데이터 바인딩**에서 찾을 수 있습니다.

관련 태스크

HTTP 바인딩 표시
응용프로그램을 전개한 후 HTTP 바인딩이 올바른지 검사하고자 할 수 있습니다.

HTTP 내보내기 바인딩 변경
관리 콘솔에서 원래 소스를 변경한 다음 응용프로그램을 다시 전개하지 않고도 HTTP 내보내기 바인딩의 구성을 변경할 수 있습니다.

HTTP 가져오기 바인딩 변경
관리 콘솔에서 원래 소스를 변경한 다음 응용프로그램을 다시 전개하지 않고도 HTTP 가져오기 바인딩의 구성을 변경할 수 있습니다.

생성된 SCA(Service Component Architecture) 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 사용자의 것으로 바꿀 수 있습니다.

시작하기 전에

WebSphere Integration Developer 또는 genMapper 명령을 사용하여 Java 유형이 WSDL(Web Services Definition Language)로 변환되어 생성되었는지 확인하십시오.

타스크 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.

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);


}
```

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

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

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.

타스크 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.

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) ;

}
```

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

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

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 유형의 패키지 이름을 반전시켜 생성된 URI를 포함하고 Java 클래스 이름과 동일한 유형의 데이터 오브젝트가 작성됩니다. 예를 들어 Java 클래스 com.ibm.xsd.Customer는 Customer 유형의 URI http://xsd.ibm.com을 포함하는 SDO로 변환됩니다. 그러면 런타임에서 Java 클래스 구성원의 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 제공하는 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 구체적(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 유형의 패키지 이름을 반전시켜 생성된 URI를 포함하고 Java 클래스 이름과 동일한 유형의 데이터 오브젝트가 작성됩니다. 예를 들어 Java 클래스 com.ibm.xsd.Customer는 Customer 유형의 URI http://xsd.ibm.com을 포함하는 SDO로 변환됩니다. 그러면 런타임에서 Java 클래스 구성원의 컨텐츠를 검사하고 SDO의 특성에 값을 지정합니다.

두 경우 모두 런타임에서 변환을 완료할 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 스키마를 사용합니다.

Java.util 인터페이스

java.util 패키지의 특정 컨테이너 인터페이스의 경우 런타임에서는 다음과 같은 구체적(concrete) 클래스를 인스턴스화합니다.

표 1. WSDL 유형을 Java 클래스로 변환

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

관련 태스크

16 페이지의 『생성된 SCA(Service Component Architecture) 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 사용자의 것으로 바꿀 수 있습니다.

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

제 2 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.

타스크 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 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.

타스크 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 상호 작용하십시오.
      - 데이터를 조회하십시오.
      - 데이터에 대해 작업하십시오.

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.

타스크 정보

BusinessFlowManagerService 인터페이스는 비즈니스 프로세스 응용프로그램용 메소드를 제공하고 HumanTaskManagerService 인터페이스는 타스크 기반 응용프로그램용 메

소드를 제공합니다. 응용프로그램은 임의의 Java 응용프로그램(다른 EJB(Enterprise JavaBeans) 응용프로그램 포함)이 될 수 있습니다.

세션 Bean의 원격 인터페이스 액세스

EJB 클라이언트 응용프로그램은 Bean의 원격 홈 인터페이스를 통해 세션 Bean의 원격 인터페이스에 액세스합니다.

타스크 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.

   사용자의 응용프로그램이 BPEContainer 응용프로그램 또는 TaskContainer 응용프로그램이 실행되는 JVM(Java Virtual Machine)과 다른 JVM에서 실행되는 경우 다음 조치를 완료하십시오.

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.

d. 비즈니스 프로세스 또는 휴먼 타스크에서 복잡한 데이터 유형을 사용하고 EJB 응용프로그램 또는 웹 응용프로그램에서 클라이언트를 실행하지 않는 경우, 해당하는 XSD 또는 WSDL 파일을 응용프로그램의 EAR 파일에 패키지하십시오.

3. JNDI(Java Naming and Directory Interface)를 통해 세션 Bean의 원격 홈 인터페이스를 찾으십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the BusinessFlowManager bean
Object result =
       initialContext.lookup("java:comp/env/ejb/BusinessFlowManagerHome");

// Convert the lookup result to the proper type
   BusinessFlowManagerHome processHome =
           (BusinessFlowManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,BusinessFlowManagerHome.class);
```

   세션 Bean의 원격 홈 인터페이스는 EJB 오브젝트에 대한 작성 메소드를 포함합니다. 이 메소드는 세션 Bean의 원격 인터페이스를 리턴합니다.

4. 세션 Bean의 원격 인터페이스에 액세스하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

```
BusinessFlowManager process = processHome.create();
```

   세션 Bean에 액세스할 수 있다고 해서 호출자가 프로세스에 대한 모든 조치를 수행할 수 있는 것입니다. 호출자는 또한 해당 조치를 수행할 수 있는 권한을 가지고 있어야 합니다. 세션 Bean의 인스턴스를 작성할 때 컨텍스트는 세션 Bean의 인스턴스와 연관됩니다. 컨텍스트에는 호출자의 프린시펄 ID, 그룹 멤버쉽 목록이 포함되며 컨텍스트는 호출자가 Business Process Choreographer J2EE 역할 중 하나를 가지는지 여부를 표시합니다. 글로벌 보안을 설정하지 않은 경우라도 컨텍스트

를 사용하여 각 호출에 대한 호출자의 권한을 확인할 수 있습니다. 글로벌 보안을 설정하지 않은 경우, 호출자의 프린시펄 ID는 UNAUTHENTICATED 값을 가집니다.

5. 서비스 인터페이스에서 표시한 비즈니스 함수를 호출하십시오.

   다음 예는 프로세스 응용프로그램에 대한 해당 단계를 표시합니다.

   ```
   process.initiate("MyProcessModel",input);
   ```

   응용프로그램의 호출이 트랜잭션으로 실행됩니다. 트랜잭션은 다음 방법 중 하나로 설정 및 종료됩니다.

   - WebSphere Application 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("jta/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("jta/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

EJB 클라이언트 응용프로그램은 Bean의 로컬 홈 인터페이스를 통해 세션 Bean의 로컬 인터페이스에 액세스합니다.

타스크 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. 글로벌 보안을 설정하지 않은 경우라도 컨텍스트를 사용하여 각 호출에 대한 호출자의 권한을 확인할 수 있습니다. 글로벌 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("jta/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("jta/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.

타스크 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.

비즈니스 프로세스 및 타스크 관련 오브젝트에 대한 조회

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 제공해야 합니다. 저장된 조회를 실행할 때 즉, 런타임 시 데이터가 어셈블될 때 조회 기준이 동적으로 적용됩니다. 저장된 조회에 매개변수가 포함된 경우에는 조회 실행 시 매개변수를 분석합니다.

Business Process Choreographer API에 대한 자세한 정보는 프로세스 관련 메소드의 경우 com.ibm.bpe.api 패키지에서, 타스크 관련 메소드의 경우 com.ibm.task.api 패키지에서 Javadoc을 참조하십시오.

API query 메소드의 구문:

Business Process Choreographer API 조회의 구문은 SQL 조회와 유사합니다. Select 절, where 절, order-by 절, skip-tuple 매개변수, 임계값 매개변수 및 시간대 매개변수가 조회에 포함될 수 있습니다.

조회 구문은 오브젝트 유형에 따른 다릅니다. 다음 테이블은 각각의 서로 다른 오브젝트 유형의 구문을 표시합니다.

표 2.

오브젝트	구분
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 `null`을 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

  65 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:

결과 조회 세트에는 조회 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 항

목이 있는 오브젝트만을 검색합니다. 작업 항목은 타스크 또는 프로세스와 같은 비즈니스 오브젝트의 권한 역할에 사용자나 사용자 그룹을 지정하는 것을 나타냅니다. 예를 들어, John Smith가 제공된 타스크의 잠재적 소유자 역할 구성원이면 이 관계를 나타내는 작업 항목 오브젝트가 존재합니다.

시스템 관리자가 아닌 사용자가 타스크를 조회하는 경우 그룹 작업 항목이 사용 가능하지 않으면 다음 액세스 조건이 WHERE 절에 추가됩니다.

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'user'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

따라서 John Smith가 잠재적 소유자인 타스크의 목록을 가져오려는 경우 API where 절은 다음과 유사할 수 있습니다.

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

이 API 절로 인한 SQL 문의 액세스 조건은 다음과 같습니다.

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JohnSmith'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

이는 John Smith가 자신이 프로세스 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 타스크 오브젝트에서 조회용으로 사전 정의된 보기:

비즈니스 프로세스 및 휴먼 타스크 오브젝트에 대해 사전 정의된 데이터베이스 보기가 제공됩니다. 다음 오브젝트에 대한 참조 데이터를 조회하는 경우 다음 보기를 사용하십시오.

사전 정의된 보기를 사용할 때 보기 열에 대한 join 술부를 명시적으로 추가할 필요가 없으며 해당 구성체는 자동으로 추가됩니다. 서비스 API(BusinessFlowManagerService 또는 HumanTaskManagerService)의 일반 조회 함수를 사용하여 이 데이터를 조회할 수 있습니다. HumanTaskManagerDelegate API의 해당 메소드를 사용하거나 ExecutableQuery 인터페이스 구현으로 제공된 사용자 고유의 사전정의된 조회를 사용할 수도 있습니다.

주: 보기에는 설명되지 않은 열이 포함될 수도 있습니다. 이러한 열은 내부 사용을 위한 것일 뿐입니다.

ACTIVITY 보기:

활동의 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 3. ACTIVITY 보기의 열

열 이름	유형	설명
PIID	ID	프로세스 인스턴스 ID
AIID	ID	활동 인스턴스 ID
PTID	ID	프로세스 템플리트 ID
ATID	ID	활동 템플리트 ID
KIND	정수	활동의 종류입니다. 가능한 값은 다음과 같습니다. KIND_INVOKE (21) KIND_RECEIVE (23) KIND_REPLY (24) KIND_THROW (25) KIND_RETHROW (46) KIND_TERMINATE (26) KIND_WAIT (27) KIND_COMPENSATE (29) KIND_SEQUENCE (30) KIND_EMPTY (3) KIND_SWITCH (32) KIND_WHILE (34) KIND_PICK (36) KIND_FLOW (38) KIND_SCOPE (40) KIND_SCRIPT (42) KIND_STAFF (43) KIND_ASSIGN (44) KIND_CUSTOM (45) KIND_FOR_EACH_PARALLEL (49) KIND_FOR_EACH_SERIAL (47)
COMPLETED	시간소인	활동이 완료된 시간
ACTIVATED	시간소인	활동이 활성화된 시간
FIRST_ACTIVATED	시간소인	활동이 처음으로 활성화된 시간
STARTED	시간소인	활동이 시작된 시간

표 3. ACTIVITY 보기의 열 (계속)

열 이름	유형	설명
STATE	정수	활동의 상태입니다. 가능한 값은 다음과 같습니다. STATE_INACTIVE (1) STATE_READY (2) STATE_RUNNING (3) STATE_PROCESSING_UNDO (14) STATE_SKIPPED (4) STATE_FINISHED (5) STATE_FAILED (6) STATE_TERMINATED (7) STATE_CLAIMED (8) STATE_TERMINATING (9) STATE_FAILING (10) STATE_WAITING (11) STATE_EXPIRED (12) STATE_STOPPED (13)
OWNER	문자열	소유자의 프린시펄 ID
DESCRIPTION	문자열	활동 템플리트 설명에 플레이스홀더가 포함되는 경우 이 열에는 플레이스홀더가 해석된 활동 인스턴스의 설명이 포함됩니다.
TEMPLATE_NAME	문자열	연관된 프로세스 템플리트의 이름
TEMPLATE_DESCR	문자열	연관된 활동 템플리트의 이름
BUSINESS_RELEVANCE	부울	활동의 비즈니스와 연관 여부를 지정합니다. 가능한 값은 다음과 같습니다. **TRUE** 활동이 비즈니스와 관련됩니다. Business Process Choreographer 탐색기에서 활동 상태를 볼 수 있습니다. **FALSE** 활동이 비즈니스와 관련되지 않습니다.
EXPIRES	시간소인	활동이 만기되는 날짜 및 시간입니다. 활동 만기 시 이 이벤트가 발생한 날짜 및 시간입니다.

ACTIVITY_ATTRIBUTE 보기:

활동의 사용자 정의 특성 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 4. ACTIVITY_ATTRIBUTE 보기의 열

열 이름	유형	설명
AIID	ID	사용자 정의 특성이 있는 활동 인스턴스 ID

표 4. ACTIVITY_ATTRIBUTE 보기의 열 (계속)

열 이름	유형	설명
NAME	문자열	사용자 정의 특성의 이름
VALUE	문자열	사용자 정의 특성의 값

ACTIVITY_SERVICE 보기:

활동 서비스의 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 5. ACTIVITY_SERVICE 보기의 열

열 이름	유형	설명
EIID	ID	이벤트 인스턴스의 ID
AIID	ID	이벤트를 대기하는 활동 인스턴스의 ID
PIID	ID	이벤트를 포함하는 프로세스 인스턴스의 ID
VTID	ID	이벤트를 설명하는 서비스 템플리트의 ID
PORT_TYPE	문자열	포트 유형의 이름
NAME_SPACE_URI	문자열	네임 스페이스의 URI
OPERATION	문자열	서비스의 조작 이름

APPLICATION_COMP 보기:

응용프로그램 컴포넌트 ID 및 타스크의 기본 설정을 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 6. APPLICATION_COMP 보기의 열

열 이름	유형	설명
ACOID	문자열	응용프로그램 컴포넌트의 ID
BUSINESS_ RELEVANCE	부울	컴포넌트의 기본 타스크 비즈니스 연관 정책입니다. 이 값은 타스크 템플리트 또는 타스크의 정의로 겹쳐쓸 수 있습니다. 속성은 추적 감사에 대한 로깅에 영향을 줍니다. 가능한 값은 다음과 같습니다. **TRUE** 타스크가 비즈니스와 관련되고 감사됩니다. **FALSE** 타스크는 비즈니스와 연관되지 않으며 감사되지 않습니다.
NAME	문자열	응용프로그램 컴포넌트 이름
SUPPORT_ AUTOCLAIM	부울	컴포넌트의 기본 자동 청구 정책입니다. 이 속성이 TRUE로 설정되는 경우 단일 사용자가 잠재적 소유자이면 이 타스크가 자동 청구될 수 있습니다. 이 값은 타스크 템플리트 또는 타스크의 정의로 겹쳐쓸 수 있습니다.
SUPPORT_CLAIM_ SUSP	부울	일시중단된 타스크 청구 여부를 결정하는 컴포넌트의 기본 설정입니다. 이 속성이 TRUE로 설정되는 경우 일시중단된 타스크를 청구할 수 있습니다. 이 값은 타스크 템플리트 또는 타스크의 정의로 겹쳐쓸 수 있습니다.

표 6. APPLICATION_COMP 보기의 열 (계속)

열 이름	유형	설명
SUPPORT_ DELEGATION	부울	컴포넌트의 기본 타스크 대표 정책입니다. 이 속성을 TRUE로 설정하면 타스크에 대한 작업 항목 지정을 수정할 수 있습니다. 이는 작업 항목을 작성, 삭제 또는 전송할 수 있음을 의미합니다.
SUPPORT_ FOLLOW_ON	부울	컴포넌트의 기본 후속 타스크 정책입니다. 이 속성을 TRUE로 설정하면 타스크에 대한 후속 타스크를 작성할 수 있습니다. 이 값은 타스크 템플리트 또는 타스크의 정의로 겹쳐쓸 수 있습니다.
SUPPORT_ SUB_TASK	부울	컴포넌트의 기본 서브타스크 정책입니다. 이 속성이 TRUE로 설정되는 경우 서브타스크를 타스크에 대해 작성할 수 있습니다. 이 값은 타스크 템플리트 또는 타스크의 정의로 겹쳐쓸 수 있습니다.

ESCALATION 보기:

에스컬레이션 데이터를 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 7. ESCALATION 보기의 열

열 이름	유형	설명
ESIID	문자열	에스컬레이션 인스턴스 ID
ACTION	정수	에스컬레이션으로 트리거되는 조치입니다. 가능한 값은 다음과 같습니다. **ACTION_CREATE_WORK_ITEM (1)** 각 에스컬레이션 수신자의 작업 항목을 작성합니다. **ACTION_SEND_EMAIL (2)** 각 에스컬레이션 수신자에게 전자 우편을 전송합니다. **ACTION_CREATE_EVENT (3)** 이벤트를 작성 및 공개합니다.
ACTIVATION_STATE	정수	해당 타스크가 다음 상태 중 하나에 도달하는 경우 에스컬레이션 인스턴스가 작성됩니다. **ACTIVATION_STATE_READY (2)** 휴먼 또는 참여 타스크의 청구 준비 여부를 지정합니다. **ACTIVATION_STATE_RUNNING (3)** 원래 타스크가 시작되어 실행 중인지 여부를 지정합니다. **ACTIVATION_STATE_CLAIMED (8)** 타스크가 청구되도록 지정합니다. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** 서브타스크가 완료될 때까지 타스크가 대기하도록 지정합니다.

표 7. ESCALATION 보기의 열 (계속)

열 이름	유형	설명
ACTIVATION_TIME	시간소인	에스컬레이션이 활성화된 시간
AT_LEAST_ EXP_STATE	정수	에스컬레이션에서 예상되는 타스크의 상태입니다. 제한시간 초과가 발생하는 경우 타스크 상태는 이 속성 값과 비교됩니다. 가능한 값은 다음과 같습니다. **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** 타스크가 청구되도록 지정합니다. **AT_LEAST_EXPECTED_STATE_ENDED (20)** 타스크가 final 상태(FINISHED, FAILED, TERMINATED 또는 EXPIRED)인 것으로 지정합니다. **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** 타스크의 모든 서브타스크가 완료되도록 지정합니다.
ESTID	문자열	해당 에스컬레이션 템플리트 ID
FIRST_ESIID	문자열	체인에서 첫 번째 에스컬레이션의 ID
INCREASE_PRIORITY	정수	타스크 우선순위가 증가되는 방법을 표시합니다. 가능한 값은 다음과 같습니다. **INCREASE_PRIORITY_NO (1)** 타스크 우선순위가 증가되지 않습니다. **INCREASE_PRIORITY_ONCE (2)** 타스크 우선순위는 한 번에 하나씩 증가됩니다. **INCREASE_PRIORITY_REPEATED (3)** 타스크 우선순위는 에스컬레이션이 반복될 때마다 하나씩 증가됩니다.
NAME	문자열	에스컬레이션 이름
STATE	정수	에스컬레이션 상태입니다. 가능한 값은 다음과 같습니다. STATE_INACTIVE (1) STATE_WAITING (2) STATE_ESCALATED (3) STATE_SUPERFLUOUS (4)
TKIID	문자열	에스컬레이션이 속하는 타스크 인스턴스 ID

ESCALATION_CPROP 보기:

에스컬레이션의 사용자 정의 특성을 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 8. ESCALATION_CPROP 보기의 열

열 이름	유형	설명
ESIID	String	에스컬레이션 ID
NAME	문자열	특성의 이름
DATA_TYPE	문자열	비문자열 사용자 정의 특성 클래스의 유형
STRING_VALUE	문자열	문자열 유형의 사용자 정의 특성 값

ESCALATION_DESC 보기:

에스컬레이션의 다국어 설명 데이터를 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 9. ESCALATION_DESC 보기의 열

열 이름	유형	설명
ESIID	문자열	에스컬레이션 ID
LOCALE	문자열	설명 또는 표시 이름과 연관된 로케일 이름
DESCRIPTION	문자열	타스크 템플리트의 설명
DISPLAY_NAME	문자열	에스컬레이션의 설명 이름

ESC_TEMPL 보기:

에스컬레이션 템플리트에 대한 데이터를 조회하려면 이 사전 정의된 데이터베이스 보기를 사용하십시오.

표 10. ESC_TEMPL 보기의 열

열 이름	종류	설명
ESTID	문자열	에스컬레이션 템플리트의 ID
ACTION	정수	에스컬레이션으로 트리거되는 조치입니다. 가능한 값은 다음과 같습니다. **ACTION_CREATE_WORK_ITEM (1)** 각 에스컬레이션 수신자의 작업 항목을 작성합니다. **ACTION_SEND_EMAIL (2)** 각 에스컬레이션 수신자에게 전자 우편을 전송합니다. **ACTION_CREATE_EVENT (3)** 이벤트를 작성 및 공개합니다.

표 10. ESC_TEMPL 보기의 열 (계속)

열 이름	종류	설명
ACTIVATION_STATE	정수	해당 타스크가 다음 상태 중 하나에 도달하는 경우 에스컬레이션 인스턴스가 작성됩니다. **ACTIVATION_STATE_READY (2)** 휴먼 또는 참여 타스크의 청구 준비 여부를 지정합니다. **ACTIVATION_STATE_RUNNING (3)** 원래 타스크가 시작되어 실행 중인지 여부를 지정합니다. **ACTIVATION_STATE_CLAIMED (8)** 타스크가 청구되도록 지정합니다. **ACTIVATION_STATE_WAITING_ FOR_SUBTASK (20)** 서브타스크가 완료될 때까지 타스크가 대기하도록 지정합니다.
AT_LEAST_ EXP_STATE	정수	에스컬레이션에서 예상되는 타스크의 상태입니다. 제한시간 초과가 발생하는 경우 타스크 상태는 이 속성 값과 비교됩니다. 가능한 값은 다음과 같습니다. **AT_LEAST_EXPECTED_STATE_CLAIMED (8)** 타스크가 청구되도록 지정합니다. **AT_LEAST_EXPECTED_STATE_ENDED (20)** 타스크가 final 상태(FINISHED, FAILED, TERMINATED 또는 EXPIRED)인 것으로 지정합니다. **AT_LEAST_EXPECTED_STATE_ SUBTASKS_COMPLETED (21)** 타스크의 모든 서브타스크가 완료되도록 지정합니다.
CONTAINMENT_ CTX_ID	문자열	템플리트가 인라인 타스크 템플리트에 속하는 경우 포함 컨텍스트는 프로세스 템플리트입니다. 에스컬레이션 템플리트 컨텍스트가 독립형 타스크 템플리트에 속하는 경우에는 포함 컨텍스트가 타스크 템플리트입니다.
FIRST_ESTID	문자열	에스컬레이션 템플리트의 체인에서 첫 번째 에스컬레이션 템플리트의 ID

표 10. ESC_TEMPL 보기의 열 (계속)

열 이름	종류	설명
INCREASE_PRIORITY	정수	타스크 우선순위가 증가되는 방법을 표시합니다. 가능한 값은 다음과 같습니다. **INCREASE_PRIORITY_NO (1)** 타스크 우선순위가 증가되지 않습니다. **INCREASE_PRIORITY_ONCE (2)** 타스크 우선순위는 한 번에 하나씩 증가됩니다. **INCREASE_PRIORITY_REPEATED (3)** 타스크 우선순위는 에스컬레이션이 반복될 때마다 하나씩 증가됩니다.
NAME	문자열	에스컬레이션 템플리트의 이름
PREVIOUS_ESTID	문자열	에스컬레이션 템플리트의 체인에서 이전 에스컬레이션 템플리트의 ID
TKTID	문자열	에스컬레이션 템플리트가 속한 타스크 템플리트 ID

ESC_TEMPL_CPROP 보기:

에스컬레이션 템플리트의 사용자 정의 특성을 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 11. ESC_TEMPL_CPROP 보기의 열

열 이름	종류	설명
ESTID	문자열	에스컬레이션 템플리트의 ID
NAME	문자열	특성의 이름
TKTID	문자열	에스컬레이션 템플리트가 속한 타스크 템플리트 ID
DATA_TYPE	문자열	비문자열 사용자 정의 특성 클래스의 유형
VALUE	문자열	문자열 유형의 사용자 정의 특성 값

ESC_TEMPL_DESC 보기:

에스컬레이션 템플리트에 대한 다국어 설명 데이터를 조회하려면 이 사전 정의된 데이터베이스 보기를 사용하십시오.

표 12. ESC_TEMPL_DESC 보기의 열

열 이름	종류	설명
ESTID	문자열	에스컬레이션 템플리트의 ID
LOCALE	문자열	설명 또는 표시 이름과 연관된 로케일 이름
TKTID	문자열	에스컬레이션 템플리트가 속한 타스크 템플리트 ID
DESCRIPTION	문자열	타스크 템플리트의 설명
DISPLAY_NAME	문자열	에스컬레이션의 설명 이름

PROCESS_ATTRIBUTE 보기:

프로세스의 사용자 정의 특성 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 13. PROCESS_ATTRIBUTE 보기의 열

열 이름	유형	설명
PIID	ID	사용자 정의 특성이 있는 프로세스 인스턴스의 ID
NAME	문자열	사용자 정의 특성의 이름
VALUE	문자열	사용자 정의 특성의 값

PROCESS_INSTANCE 보기:

프로세스 인스턴스에 대한 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 14. PROCESS_INSTANCE 보기의 열

열 이름	유형	설명
PTID	ID	프로세스 템플리트 ID
PIID	ID	프로세스 인스턴스 ID
NAME	문자열	프로세스 인스턴스 이름
STATE	정수	프로세스 인스턴스 상태입니다. 가능한 값은 다음과 같습니다. STATE_READY (1) STATE_RUNNING (2) STATE_FINISHED (3) STATE_COMPENSATING (4) STATE_INDOUBT (10) STATE_FAILED (5) STATE_TERMINATED (6) STATE_COMPENSATED (7) STATE_COMPENSATION_FAILED (12) STATE_TERMINATING (8) STATE_FAILING (9) STATE_SUSPENDED (11)
CREATED	시간소인	프로세스 인스턴스가 작성된 시간
STARTED	시간소인	프로세스 인스턴스가 시작된 시간
COMPLETED	시간소인	프로세스 인스턴스가 완료된 시간
PARENT_PIID	ID	상위 프로세스 인스턴스의 ID
PARENT_NAME	문자열	상위 프로세스 인스턴스 이름
TOP_LEVEL_PIID	ID	최상위 레벨 프로세스 인스턴스의 프로세스 인스턴스 ID입니다. 최상위 레벨 프로세스 인스턴스가 없는 경우에는 현재 프로세스 인스턴스의 프로세스 인스턴스 ID입니다.
TOP_LEVEL_NAME	문자열	맨 위 레벨 프로세스 인스턴스 이름입니다. 맨 위 레벨 프로세스 인스턴스가 없는 경우 현재 프로세스 인스턴스의 이름입니다.
STARTER	문자열	프로세스 인스턴스 시작자의 프린시펄 ID

표 14. PROCESS_INSTANCE 보기의 열 (계속)

열 이름	유형	설명
DESCRIPTION	문자열	프로세스 템플리트 설명에 플레이스홀더가 포함되는 경우 이 열에는 플레이스홀더가 해석된 프로세스 인스턴스의 설명이 포함됩니다.
TEMPLATE_NAME	문자열	연관된 프로세스 템플리트 이름
TEMPLATE_DESCR	문자열	연관된 프로세스 템플리트의 설명
RESUMES	시간소인	프로세스 인스턴스가 자동으로 재개될 시간

PROCESS_TEMPLATE 보기:

프로세스 템플리트에 대한 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 15. PROCESS_TEMPLATE 보기의 열

열 이름	유형	설명
PTID	ID	프로세스 템플리트 ID
NAME	문자열	프로세스 템플리트의 이름
VALID_FROM	시간소인	프로세스 템플리트가 인스턴스화될 수 있는 시간
TARGET_NAMESPACE	문자열	프로세스 템플리트의 대상 네임 스페이스
APPLICATION_NAME	문자열	프로세스 템플리트가 속하는 엔터프라이즈 응용프로그램 이름
VERSION	문자열	사용자 정의 버전
CREATED	시간소인	프로세스 템플리트가 데이터베이스에서 작성된 시간
STATE	정수	프로세스 인스턴스를 작성하기 위해 프로세스 템플리트가 사용 가능한지 지정합니다. 가능한 값은 다음과 같습니다. STATE_STARTED (1) STATE_STOPPED (2)
EXECUTION_MODE	정수	이 프로세스 템플리트에서 파생된 프로세스 인스턴스가 실행될 수 있는 방법을 지정합니다. 가능한 값은 다음과 같습니다. EXECUTION_MODE_MICROFLOW (1) EXECUTION_MODE_LONG_RUNNING (2)
DESCRIPTION	문자열	프로세스 템플리트의 설명
COMP_SPHERE	정수	프로세스 템플리트에서 마이크로플로우 인스턴스의 보상 작동을 지정합니다. 기존 보상 스피어가 결합되거나 보상 스피어가 작성됩니다. 가능한 값은 다음과 같습니다. COMP_SPHERE_REQUIRED (2) COMP_SPHERE_SUPPORTS (4)
DISPLAY_NAME	문자열	프로세스의 설명 이름

QUERY_PROPERTY 보기:

프로세스 레벨 변수에 대한 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 16. QUERY_PROPERTY 보기의 열

열 이름	유형	설명
PIID	ID	프로세스 인스턴스 ID
VARIABLE_NAME	문자열	프로세스 레벨 변수의 이름
NAME	문자열	조회 특성의 이름
NAMESPACE	문자열	조회 특성의 네임 스페이스
GENERIC_VALUE	문자열	정의된 유형 중 하나로 맵핑되지 않는 특성 유형에 대한 문자열 표시: STRING_VALUE, NUMBER_VALUE, DECIMAL_VALUE 또는 TIMESTAMP_VALUE.
STRING_VALUE	문자열	특성 유형이 문자열 유형으로 맵핑되는 경우, 문자열 값입니다.
NUMBER_VALUE	정수	특성 유형이 정수 유형으로 맵핑되는 경우, 정수 값입니다.
DECIMAL_VALUE	10진수	특성 유형이 부동 소수점 유형으로 맵핑되는 경우, 10진수 값입니다.
TIMESTAMP_VALUE	시간소인	특성 유형이 시간소인 유형으로 맵핑되는 경우, 시간소인 값입니다.

TASK 보기:

타스크 오브젝트에 대한 조회에 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 17. TASK 보기의 열

열 이름	유형	설명
TKIID	ID	타스크 인스턴스의 ID
ACTIVATED	시간소인	타스크가 활성화된 시간
APPLIC_DEFAULTS_ID	ID	타스크의 기본값을 지정하는 응용프로그램 컴포넌트의 ID
APPLIC_NAME	문자열	타스크가 속하는 엔터프라이즈 응용프로그램 이름
BUSINESS_RELEVANCE	부울	타스크의 비즈니스 연관 여부를 지정합니다. 속성은 추적 감사에 대한 로깅에 영향을 줍니다. 가능한 값은 다음과 같습니다. **TRUE** 타스크가 비즈니스와 관련되고 감사됩니다. **FALSE** 타스크는 비즈니스와 연관되지 않으며 감사되지 않습니다.
COMPLETED	시간소인	타스크가 완료된 시간
CONTAINMENT_CTX_ID	ID	이 타스크의 포함 컨텍스트입니다. 이 속성은 타스크의 라이프 사이클을 결정합니다. 타스크의 포함 컨텍스트가 삭제된 경우 타스크도 삭제됩니다.

표 17. TASK 보기의 열 (계속)

열 이름	유형	설명
C T X _ AUTHORIZATION	정수	타스크 소유자가 타스크 컨텍스트에 액세스할 수 있도록 합니다. 가능한 값은 다음과 같습니다. **AUTH_NONE** 연관된 컨텍스트 오브젝트에 대한 권한이 없습니다. **AUTH_READER** 연관된 컨텍스트 오브젝트 조작에는 예를 들어 프로세스 인스턴스의 특성 읽기와 같이 독서자 권한이 필요합니다.
DUE	시간소인	타스크가 예정된 시간
EXPIRES	시간소인	타스크가 만기되는 날짜
FIRST_ACTIVATED	시간소인	타스크가 처음으로 활성화된 시간
FOLLOW_ON_TKIID	ID	후속타스크 인스턴스 ID
H I E R A R C H Y _ POSITION	정수	가능한 값은 다음과 같습니다. **HIERARCHY_POSITION_TOP_TASK (0)** 타스크 계층 구조에서 맨 위 레벨 타스크입니다. **HIERARCHY_POSITION_SUB_TASK (1)** 타스크는 타스크 계층 구조에서 서브타스크입니다. **HIERARCHY_POSITION_FOLLOW_ON_TASK (2)** 타스크는 타스크 계층 구조에서 후속 타스크입니다.
IS_AD_HOC	부울	이 타스크가 런타임 시 동적으로 작성되는지 또는 타스크 템플리트에서 작성되는지 여부를 표시합니다.
IS_ESCALATED	부울	이 타스크의 에스컬레이션이 발생했는지 여부를 표시합니다.
IS_INLINE	부울	타스크가 비즈니스 프로세스 인라인 타스크인지 여부를 표시합니다.
I S _ W A I T _ F O R _ SUB_TK	부울	하위 타스크가 종료 상태에 도달할 때까지 상위 타스크가 기다리는지 여부를 표시합니다.

표 17. TASK 보기의 열 (계속)

열 이름	유형	설명
KIND	정수	타스크의 종류입니다. 가능한 값은 다음과 같습니다. **KIND_HUMAN (101)** 타스크가 사용자가 작성하고 처리하는 공동 작업 타스크임을 설명합니다. **KIND_WPC_STAFF_ACTIVITY (102)** 타스크가 WebSphere Business Integration Server Foundation, 버전 5 비즈니스 프로세스의 staff 활동인 휴먼 타스크임을 설명합니다. **KIND_ORIGINATING (103)** 타스크가 사용자가 서비스를 작성, 초기화 및 시작할 수 있는 사용자 대 컴퓨터의 상호작용을 지원하는 호출 타스크임을 설명합니다. **KIND_PARTICIPATING (105)** 타스크가 사용자가 서비스를 구현할 수 있으며 컴퓨터 대 사용자의 상호작용을 지원하는 수행할 타스크임을 설명합니다. **KIND_ADMINISTRATIVE (106)** 타스크가 관리 타스크임을 설명합니다.
LAST_MODIFIED	시간소인	타스크가 마지막 변경된 시간
LAST_STATE_CHANGE	시간소인	타스크의 상태가 마지막 변경된 시간
NAME	문자열	타스크의 이름
NAME_SPACE	문자열	타스크를 카테고리화하는 데 사용되는 네임 스페이스
ORIGINATOR	문자열	타스크 시작자의 프린시펄 ID
OWNER	문자열	타스크 소유자의 프린시펄 ID
PARENT_CONTEXT_ID	문자열	이 타스크의 상위 컨텍스트입니다. 이 속성은 호출 응용프로그램 컴포넌트의 해당 컨텍스트에 키를 제공합니다. 상위 컨텍스트는 타스크를 작성하는 응용프로그램 컴포넌트로 설정됩니다.
PRIORITY	정수	타스크의 우선순위
RESUMES	시간소인	타스크가 자동으로 재개될 시간
STARTED	시간소인	타스크가 시작된 시간(STATE_RUNNING, STATE_CLAIMED)
STARTER	문자열	타스크 시작자의 프린시펄 ID

표 17. TASK 보기의 열 (계속)

열 이름	유형	설명
STATE	정수	타스크의 상태입니다. 가능한 값은 다음과 같습니다. **STATE_READY (2)** 타스크가 청구될 수 있는 상태입니다. **STATE_RUNNING (3)** 타스크가 시작되어 실행 중인 상태입니다. **STATE_FINISHED (5)** 타스크가 완료된 상태입니다. **STATE_FAILED (6)** 타스크가 성공적으로 완료되지 않은 상태입니다. **STATE_TERMINATED (7)** 외부 또는 내부 요청으로 타스크가 종료된 상태입니다. **STATE_CLAIMED (8)** 타스크가 청구된 상태입니다. **STATE_EXPIRED (12)** 지정된 지속 기간이 초과되어 타스크가 종료된 상태입니다. **STATE_FORWARDED (101)** 타스크가 후속 타스크와 함께 완료되었음을 설명합니다.
SUPPORT_AUTOCLAIM	부울	이 타스크가 단일 사용자에게 지정된 경우 자동으로 청구되는지 여부를 표시합니다.
SUPPORT_CLAIM_SUSP	부울	이 타스크가 일시중단되는 경우 청구될 수 있는지 여부를 표시합니다.
SUPPORT_DELEGATION	부울	이 타스크가 작업 항목 작성, 삭제 또는 전송을 통해 작업 위임을 지원하는지 여부를 표시합니다.
SUPPORT_FOLLOW_ON	부울	이 타스크가 후속 타스크 작성을 지원하는지 여부를 표시합니다.
SUPPORT_SUB_TASK	부울	이 타스크가 서브타스크 작성을 지원하는지 여부를 표시합니다.
SUSPENDED	부울	이 타스크가 일시중단되는지 여부를 표시합니다.
TKTID	ID	타스크 템플리트 ID
TOP_TKIID	ID	하위 타스크의 경우 최상위 타스크 인스턴스 ID
TYPE	문자열	타스크를 카테고리화하는 데 사용되는 유형

TASK_CPROP 보기:

타스크 오브젝트의 사용자 정의 특성을 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 18. TASK_CPROP 보기의 열

열 이름	유형	설명
TKIID	문자열	타스크 인스턴스 ID
NAME	문자열	특성의 이름
DATA_TYPE	문자열	비문자열 사용자 정의 특성 클래스의 유형
STRING_VALUE	문자열	문자열 유형의 사용자 정의 특성 값

TASK_DESC 보기:

타스크 오브젝트에 대한 다국어 설명 데이터를 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 19. TASK_DESC 보기의 열

열 이름	유형	설명
TKIID	문자열	타스크 인스턴스 ID
LOCALE	문자열	설명 또는 표시 이름과 연관된 로케일 이름
DESCRIPTION	문자열	타스크에 대한 설명
DISPLAY_NAME	문자열	타스크의 설명 이름

TASK_TEMPL 보기:

이 사전 정의 데이터베이스 보기는 타스크를 인스턴스화하는 데 사용할 수 있는 데이터를 보유합니다.

표 20. TASK_TEMPL 보기의 열

열 이름	유형	설명
TKTID	문자열	타스크 템플리트 ID
VALID_FROM	시간소인	타스크 템플리트가 인스턴스화에 사용할 수 있는 시간
APPLIC_DEFAULTS_ID	문자열	타스크 템플리트의 기본값을 지정하는 응용프로그램 컴포넌트의 ID
APPLIC_NAME	문자열	타스크 템플리트가 속하는 엔터프라이즈 응용프로그램 이름
BUSINESS_RELEVANCE	부울	타스크 템플리트의 비즈니스 연관 여부를 지정합니다. 속성은 추적 감사에 대한 로깅에 영향을 줍니다. 가능한 값은 다음과 같습니다. **TRUE** 타스크가 비즈니스와 관련되고 감사됩니다. **FALSE** 타스크는 비즈니스와 연관되지 않으며 감사되지 않습니다.
CONTAINMENT_CTX_ID	ID	이 타스크 템플리트의 포함 컨텍스트입니다. 이 속성은 타스크 템플리트의 라이프 사이클을 결정합니다. 포함 컨텍스트가 삭제된 경우 타스크 템플리트도 삭제됩니다.

표 20. TASK_TEMPL 보기의 열 (계속)

열 이름	유형	설명
C T X _ AUTHORIZATION	정수	타스크 소유자가 타스크 컨텍스트에 액세스할 수 있도록 합니다. 가능한 값은 다음과 같습니다. **AUTH_NONE** 연관된 컨텍스트 오브젝트에 대한 권한이 없습니다. **AUTH_READER** 연관된 컨텍스트 오브젝트 조작에는 예를 들어 프로세스 인스턴스의 특성 읽기와 같이 독서자 권한이 필요합니다.
DEFINITION_NAME	문자열	TEL(Task Execution Language) 파일의 타스크 템플리트 정의 이름
DEFINITION_NS	문자열	TEL 파일의 타스크 템플리트 정의의 네임 스페이스
IS_AD_HOC	부울	이 타스크 템플리트가 런타임 시 동적으로 작성되는지 여부 또는 타스크가 EAR 파일의 일부로 전개된 시점을 표시합니다.
IS_INLINE	부울	이 타스크 템플리트가 비즈니스 프로세스의 타스크로 모델링되었는지 여부를 나타냅니다.
KIND	정수	이 타스크 템플리트에서 파생된 타스크 종류입니다. 가능한 값은 다음과 같습니다. **KIND_HUMAN (101)** 타스크가 사용자가 작성하고 처리하는 *공동 작업 타스크*임을 설명합니다. **KIND_ORIGINATING (103)** 타스크가 사용자가 서비스를 작성, 초기화 및 시작할 수 있는 사용자 대 컴퓨터의 상호작용을 지원하는 *호출 타스크*임을 설명합니다. **KIND_PARTICIPATING (105)** 타스크가 사용자가 서비스를 구현할 수 있으며 컴퓨터 대 사용자의 상호작용을 지원하는 *수행할 타스크*임을 설명합니다. **KIND_ADMINISTRATIVE (106)** 타스크가 관리 타스크임을 설명합니다.
NAME	문자열	타스크 템플리트의 이름
NAMESPACE	문자열	타스크 템플리트를 카테고리화하는 데 사용되는 네임 스페이스
PRIORITY	정수	타스크 템플리트의 우선순위

표 20. TASK_TEMPL 보기의 열 (계속)

열 이름	유형	설명
STATE	정수	타스크 템플리트의 상태입니다. 가능한 값은 다음과 같습니다. **STATE_STARTED (1)** 타스크 인스턴스를 작성하는 데 타스크 템플리트를 사용할 수 있도록 지정합니다. **STATE_STOPPED (2)** 타스크 템플리트가 중지되도록 지정합니다. 이 상태의 타스크 템플리트에서 타스크 인스턴스를 작성할 수 없습니다.
SUPPORT_AUTOCLAIM	부울	이 타스크 템플리트에서 파생된 타스크가 단일 사용자에 지정된 경우 자동으로 청구될 수 있는지 여부를 나타냅니다.
SUPPORT_CLAIM_SUSP	부울	이 타스크 템플리트에서 파생된 타스크가 일시중단 상태인 경우 청구될 수 있는지 여부를 나타냅니다.
SUPPORT_DELEGATION	부울	이 타스크 템플리트에서 파생된 타스크가 작업 항목의 작성, 삭제 또는 전송을 사용하여 작업 위임을 지원하는지 여부를 나타냅니다.
SUPPORT_FOLLOW_ON	부울	이 타스크 템플리트가 후속 타스크 작성을 지원하는지 여부를 표시합니다.
SUPPORT_SUB_TASK	부울	이 타스크 템플리트가 서브타스크 작성을 지원하는지 여부를 표시합니다.
TYPE	문자열	타스크 템플리트를 카테고리화하는 데 사용되는 유형

TASK_TEMPL_CPROP 보기:

타스크 템플리트의 사용자 정의 특성을 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 21. TASK_TEMPL_CPROP 보기의 열

열 이름	유형	설명
TKTID	문자열	타스크 템플리트 ID
NAME	문자열	특성의 이름
DATA_TYPE	문자열	비문자열 사용자 정의 특성 클래스의 유형
STRING_VALUE	문자열	문자열 유형의 사용자 정의 특성 값

TASK_TEMPL_DESC 보기:

타스크 템플리트 오브젝트에 대한 다국어 설명 데이터를 조회하려면 이 사전 정의 데이터베이스 보기를 사용하십시오.

표 22. TASK_TEMPL_DESC 보기의 열

열 이름	유형	설명
TKTID	문자열	타스크 템플리트 ID

표 22. TASK_TEMPL_DESC 보기의 열 (계속)

열 이름	유형	설명
LOCALE	문자열	설명 또는 표시 이름과 연관된 로케일 이름
DESCRIPTION	문자열	타스크 템플리트의 설명
DISPLAY_NAME	문자열	타스크 템플리트의 설명 이름

WORK_ITEM 보기:

프로세스, 타스크 및 에스컬레이션의 작업 항목 및 권한 데이터에 대한 조회에 이 사전정의 데이터베이스 보기를 사용하십시오.

표 23. WORK_ITEM 보기의 열

열 이름	유형	설명
WIID	ID	작업 항목 ID
OWNER_ID	문자열	소유자의 프린시펄 ID
GROUP_NAME	문자열	연관된 그룹 작업 목록의 이름
EVERYBODY	부울	모든 사용자가 이 작업 항목을 소유하는지 여부를 지정합니다.
OBJECT_TYPE	정수	연관된 오브젝트의 유형입니다. 가능한 값은 다음과 같습니다. **OBJECT_TYPE_ACTIVITY (1)** 작업 항목이 활동에 대해 작성되었는지 지정합니다. **OBJECT_TYPE_PROCESS_INSTANCE (3)** 작업 항목이 프로세스 인스턴스에 대해 작성되었는지 지정합니다. **OBJECT_TYPE_TASK_INSTANCE (5)** 작업 항목이 타스크에 대해 작성되었는지 지정합니다. **OBJECT_TYPE_TASK_TEMPLATE (6)** 작업 항목이 타스크 템플리트에 대해 작성되었는지 지정합니다. **OBJECT_TYPE_ESCALATION_ INSTANCE (7)** 작업 항목이 에스컬레이션 인스턴스에 대해 작성되었는지 지정합니다. **OBJECT_TYPE_APPLICATION_ COMPONENT (9)** 작업 항목이 응용프로그램 컴포넌트에 대해 작성되었는지 지정합니다.
OBJECT_ID	ID	연관된 오브젝트(예: 연관된 프로세스 또는 타스크) ID

표 23. WORK_ITEM 보기의 열 (계속)

열 이름	유형	설명
ASSOC_OBJECT_TYPE	정수	ASSOC_OID 속성에서 참조한 오브젝트 유형(예: 타스크, 프로세스 또는 외부 오브젝트)입니다. OBJECT_TYPE 속성값을 사용하십시오.
ASSOC_OID	ID	작업 항목이 있는 오브젝트와 연관된 오브젝트의 ID입니다. 예를 들어, 이 작업 항목이 작성된 활동 인스턴스를 포함하는 프로세스 인스턴스의 프로세스 인스턴스 ID(PIID).
REASON	정수	작업 항목 지정 이유입니다. 가능한 값은 다음과 같습니다. REASON_POTENTIAL_STARTER (5) REASON_POTENTIAL_INSTANCE_ CREATOR (11) REASON_POTENTIAL_STARTER (1) REASON_EDITOR (2) REASON_READER (3) REASON_ORIGINATOR (9) REASON_OWNER (4) REASON_STARTER (6) REASON_ESCALATION_RECEIVER (10) REASON_ADMINISTRATOR (7)
CREATION_TIME	시간소인	작업 항목이 작성된 날짜 및 시간

조회에 변수를 사용하여 데이터 필터링

조회 결과에서 조회 기준에 일치하는 오브젝트를 리턴합니다. 변수 값에 따라 결과를 필터링하고자 할 수도 있습니다.

타스크 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 나열하십시오.

   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 나열하십시오.

   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.

타스크 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

37 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.

타스크 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 CustomerOrdersStartingWithA로 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));
   ```

   이 조치를 실행하면 기준에 맞는 오브젝트가 리턴됩니다. 이 경우 A로 시작하는 고객의 모든 주문이 리턴됩니다.

3. 사용 가능한 공용 저장된 조회의 이름을 나열하십시오.

   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. 타스크 기반 오브젝트에 대하여 해당 조회를 실행하려면, 리턴되는 오브젝트 유형으로 StoredQueryData를 StoredQuery 대신 지정하십시오.

   ```
   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.

타스크 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.

   ```
   StoredQuery 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();
   ```

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.

   ```
   StoredQuery storedQuery = process.getStoredQuery
      ("CustomerOrdersStartingWithA");
   String selectClause = storedQuery.getSelectClause();
   String whereClause = storedQuery.getWhereClause();
   String orderByClause = storedQuery.getOrderByClause();
   Integer threshold = storedQuery.getThreshold();String owner = storedQuery.getOwner();
   ```

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.

타스크 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 테이블에 특정 역할이 취할 수 있는 프로세스 인스턴스의 조치가 표시됩니다.

조치	호출자의 프린시펄 역할		
	독서자	시작자	관리자
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x

조치	호출자의 프린시펄 역할		
	독서자	시작자	관리자
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 테이블에 특정 역할이 취할 수 있는 활동 인스턴스의 조치가 표시됩니다.

조치	호출자의 프린시펄 역할				
	독서자	편집자	잠재적 소유자	소유자	관리자
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x

조치	호출자의 프린시펄 역할				
	독서자	편집자	잠재적 소유자	소유자	관리자
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.

타스크 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.

타스크 정보

예제에서는 시작 마이크로플로우 및 장기 실행 프로세스의 일반적인 시나리오에 대한 응용프로그램을 개발하는 방법을 보여줍니다.

고유 시작 서비스를 포함하는 마이크로플로우 실행:

마이크로플로우는 receive 활동 또는 pick 활동으로 시작됩니다. 마이크로플로우가 receive 활동으로 시작되거나 pick 활동에 하나의 onMessage 정의만 있는 경우 시작 서비스는 고유합니다.

타스크 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 나열하여 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 receive 활동 또는 pick 활동으로 시작됩니다. 마이크로플로우에서 여러 onMessage 정의를 갖는 pick 활동으로 시작할 경우 시작 서비스는 고유하지 않습니다.

타스크 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 나열하여 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 pick 활동으로 시작하고 단일 pick 활동이 하나의 onMessage 정의만 갖는 경우입니다.

프로시저

1. 옵션: 프로세스 템플리트를 나열하여 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, 독서자 및 편집자가 결정되어 프로세스 인스턴스에 대한 작업 항목을 수신합니다. 후속 활동 인스턴스가 결정됩니다. 이들은 자동으로 시작되거나 휴먼 타스크, receive 또는 pick 활동인 경우 잠재적 소유자에 대한 작업 항목이 작성됩니다.

비고유 시작 서비스를 포함하는 장기 실행 프로세스 시작:

장기 실행 프로세스는 여러 시작 receive 또는 pick 활동을 통해 시작할 수 있습니다. 시작 메소드를 사용하여 프로세스를 시작할 수 있습니다. 시작 서비스가 고유하지 않은 경우 예를 들어, 프로세스가 여러 개의 receive 또는 pick 활동으로 시작하거나 여러 onMessage 정의를 갖는 pick 활동으로 시작하면 호출할 서비스를 식별해야 합니다.

프로시저

1. 옵션: 프로세스 템플리트를 나열하여 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, 독서자 및 편집자가 결정되어 프로세스 인스턴스에 대한 작업 항목을 수신합니다. 후속 활동 인스턴스가 결정됩니다. 이들은 자동으로 시작되거나 휴먼 타스크, receive 또는 pick 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.

타스크 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 재시작:

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

시작하기 전에

호출자는 프로세스 인스턴스의 관리자 또는 비즈니스 프로세스 관리자여야 합니다.

타스크 정보

프로세스 인스턴스를 재시작하는 것은 프로세스 인스턴스를 처음으로 시작하는 것과 비슷합니다. 그러나 프로세스 인스턴스가 재시작되면 프로세스 인스턴스 ID가 인식되고 인스턴스의 입력 메시지가 사용 가능해집니다.

프로세스에 프로세스 인스턴스를 작성할 수 있는 둘 이상의 receive 활동 또는 pick 활동(receive choice 활동이라고도 함)이 있는 경우, 이 활동에 속한 모든 메시지를 사용하여 프로세스 인스턴스를 재시작합니다. 이 활동이 요청-응답 조작을 구현할 경우 연관된 reply 활동을 탐색하면 응답이 다시 전송됩니다.

프로시저

1. 재시작하려는 프로세스를 가져오십시오.

   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```

2. 프로세스 인스턴스를 재시작하십시오.

```
PIID piid = processInstance.getID();
process.restart( piid );
```

이 조치를 실행하면 지정된 프로세스 인스턴스가 재시작됩니다.

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. 종료해야하는 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 프

로세스 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.

타스크 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 나열하십시오.

   ```
   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.

타스크 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 나열하십시오.

```
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, (String)faultNames.get(0), myFault);
```

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.

타스크 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();
   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.

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

115 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(receive 활동, pick 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.

타스크 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 나열하십시오.

```
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 receive 활동 또는 pick 활동과 유사합니다.

타스크 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 나열하십시오.

```
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.

타스크 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.

타스크 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: 타스크 정보

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.

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.

Script 활동

force-complete 호출에서 매개변수를 전달할 수 없습니다. 그러나 복구해야 할 변수를 설정해야 합니다.

Invoke 활동

활동이 실행 상태인 경우 서브프로세스가 아닌 비동기 서비스를 호출하는 invoke 활동을 강제로 완료할 수도 있습니다. 비동기 서비스가 호출되고 응답되지 않는 경우 다음을 수행할 수 있습니다.

프로시저

1. 중지 상태에 있는 중지된 활동을 나열하십시오.

```
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: 타스크 정보

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.

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 나열하십시오.

```
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 24. 프로세스 템플리트의 API 메소드

메소드	설명
getProcessTemplate	지정된 프로세스 템플리트를 검색합니다.
queryProcessTemplates	데이터베이스에 저장되는 프로세스 템플리트를 검색합니다.

프로세스 인스턴스

다음 API 메소드는 프로세스 인스턴스 시작과 관련이 있습니다.

표 25. API 메소드는 프로세스 인스턴스 시작과 관련이 있습니다.

메소드	설명
call	마이크로플로우를 작성하고 실행합니다.

표 25. API 메소드는 프로세스 인스턴스 시작과 관련이 있습니다. (계속)

메소드	설명
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 26. 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 재시작합니다.
forceTerminate	지정된 맨 위 레벨 프로세스 인스턴스, 하위 자율성이 있는 서브프로세스 및 실행 중이거나 청구되거나 대기 중인 활동을 종료합니다.
delete	지정된 맨 위 레벨 프로세스 인스턴스 및 하위 자율성이 있는 서브프로세스를 삭제합니다.
query	검색 기준에 일치하는 데이터베이스의 특성을 검색합니다.

활동

invoke 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 있습니

다. 예를 들어, continueOnError 플래그 및 연관된 복구 기능은 invoke 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 27. 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	실행 또는 중지 상태의 활동 인스턴스를 강제로 완료합니다.
forceRetry	실행 또는 중지 상태의 활동 인스턴스를 강제 반복합니다.
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 28. 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.

타스크 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.

타스크 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 나열하여 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.

타스크 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 나열하여 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 나열해서 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.

타스크 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 나열하십시오.

```
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, 오리지네이터 또는 관리자여야 합니다.

타스크 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.

타스크 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 나열하십시오.

   ```
   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 나열하는 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.

타스크 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 나열하는 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.

타스크 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, 독서자, 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 독서자 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.

타스크 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 인스턴스를 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.

타스크 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.

타스크 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.

타스크 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.

타스크 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 나열됩니다. 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 29. 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 30. 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 31. 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 32. 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 테이블은 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

조치	타스크 유형			
	수행할 타스크	공동 작업 타스크	호출 타스크	관리 타스크
resume	X	X[1]		
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.

타스크 정보

비즈니스 프로세스 시나리오의 사용자를 호출하려면 다음 타스크 종류를 비즈니스 프로세스에 포함할 수 있습니다.

- 인라인 호출 타스크(API에서 *오리지네이팅 타스크*라고도 함)

  모든 receive 활동, pick 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.

타스크 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 이

를 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.

   에제에서 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.

타스크 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

81 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.

타스크 정보

BPEL(Business Process Execution Language) 결함은 다음에서 발생할 수 있습니다.

- 웹 서비스 호출(WSDL(Web Services Description Language) 결함)
- 처리 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

타스크 정보

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

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.

활동에 설정된 결함 확인

프로시저

1. 실패 또는 중지 상태의 task 활동을 나열하십시오.

```
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 invoke 활동에서 발생한 결함 확인

타스크 정보

활동에 결함이 발생하는 경우 결함 유형으로 활동 복구에 필요한 조치가 결정됩니다.

프로시저

1. 중지 상태에 있는 휴먼 타스크 활동을 나열하십시오.

```
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 invoke 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();
  }
}
```

웹 서비스 API 클라이언트 응용프로그램 개발

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

타스크 정보

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

소개: 웹 서비스

웹 서비스는 개방형 XML 기반 표준 및 전송 프로토콜을 사용하여 클라이언트 응용프로그램과 데이터를 교환하는 웹 기반 엔터프라이즈 응용프로그램입니다. 웹 서비스를 사용하여 언어 및 환경 중립 프로그래밍 모델을 사용할 수 있습니다.

웹 서비스는 다음 코어 기술을 사용합니다.

- XML(Extensible Markup Language). XML은 데이터 종속성의 문제점을 해결합니다. 이를 사용하여 데이터를 기술하며 또한 해당 데이터를 응용프로그램 또는 프로그래밍 언어와 맵핑합니다.
- WSDL(Web Services Description Language). 이 XML 기반 언어를 사용하여 기본 응용프로그램에 대한 설명을 작성합니다. 기본 응용프로그램 및 기타 웹 응용프로그램 간의 인터페이스로 작동하여 응용프로그램을 웹 서비스로 변환하는 설명입니다.
- SOAP(Simple Object Access Protocol). SOAP는 웹의 코어 통신 프로토콜이며, 대부분의 웹 서비스는 이 프로토콜을 사용하여 서로 통신합니다.

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 상호 작용할 수 있도록 허용합니다. 예를 들어, 다음과 같습니다.
  - 프로세스 템플리트 및 프로세스 인스턴스 작성
  - 기존 프로세스 요청
  - ID별 프로세스 조회

  가능한 조치의 전체 목록은 69 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 91 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.

타스크 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.

타스크 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 Application Server에 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"/>
```

관련 개념

139 페이지의『보안 추가(Java 웹 서비스)』
클라이언트 응용프로그램에서 보안 메커니즘을 구현하여 웹 서비스 통신을 보호해야 합니다.

관련 태스크

149 페이지의『보안 추가(.NET)』
보안 메커니즘을 클라이언트 응용프로그램에 통합하여 웹 서비스 통신을 보호할 수 있습니다.

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.

타스크 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.

타스크 정보

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

WebSphere Process Server에서 비즈니스 오브젝트는 비즈니스 프로세스 또는 휴먼 타스크와 상호 작용하는 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.

관련 태스크

145 페이지의『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.

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.

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.

타스크 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. 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 Application Server의 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"/>
```

관련 개념

139 페이지의『보안 추가(Java 웹 서비스)』
클라이언트 응용프로그램에서 보안 메커니즘을 구현하여 웹 서비스 통신을 보호해야 합니다.

관련 태스크

149 페이지의『보안 추가(.NET)』
보안 메커니즘을 클라이언트 응용프로그램에 통합하여 웹 서비스 통신을 보호할 수 있습니다.

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 Application Server의 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"/>
```

   관련 개념

   139 페이지의 『보안 추가(Java 웹 서비스)』
   클라이언트 응용프로그램에서 보안 메커니즘을 구현하여 웹 서비스 통신을 보호해야 합니다.

   관련 태스크

   149 페이지의 『보안 추가(.NET)』
   보안 메커니즘을 클라이언트 응용프로그램에 통합하여 웹 서비스 통신을 보호할 수 있습니다.

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.

타스크 정보

Java 웹 서비스의 프록시 클라이언트에는 클라이언트 응용프로그램이 호출하여 웹 서비스 요청을 수행하는 여러 Java Bean 클래스가 포함되어 있습니다. 프록시 클라이언

트는 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

네임 스페이스	패키지
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.

타스크 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.

타스크 정보

웹 서비스 API의 call() 및 sendMessage() 조작은 WebSphere Process Server에서 BPEL 프로세스와의 상호 작용을 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.

타스크 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 응용프로그램 코드화하십시오.
4. 프로젝트를 빌드하십시오.
5. 클라이언트 응용프로그램을 실행하십시오.

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());
```

관련 태스크

132 페이지의『프록시 클라이언트 생성(Java 웹 서비스)』
Java 웹 서비스 클라이언트 응용프로그램은 *프록시 클라이언트*를 사용하여 웹 서비스 API와 상호작용합니다.

136 페이지의『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.

보안 추가(Java 웹 서비스)

클라이언트 응용프로그램에서 보안 메커니즘을 구현하여 웹 서비스 통신을 보호해야 합니다.

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

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

관련 개념

휴먼 타스크의 권한 역할
휴먼 타스크에서 취할 수 있는 조치는 권한 역할에 따라 다릅니다. 이 역할은 시스템 레벨 J2EE 역할 또는 인스턴스 기반 역할일 수 있습니다.

비즈니스 프로세스의 권한 역할
역할은 같은 레벨의 권한을 공유하는 개인 세트입니다. 비즈니스 프로세스에서 취할 수 있는 조치는 권한 역할에 따라 다릅니다. 이 역할은 J2EE 역할 또는 인스턴스 기반 역할일 수 있습니다.

사용자 이름 토큰 구현:

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

타스크 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 나열될 수 있습니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService

      나열되는 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 나열됩니다.
      - 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 태스크

123 페이지의『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.

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.

타스크 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 나열될 수 있습니다.
      - 비즈니스 프로세스의 경우 service/BFMWSService
      - 휴먼 타스크의 경우 service/HTMWSService

      나열되는 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 나열됩니다.
      - 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 다음 토큰 생성기 클래스가 선택되었는지 확인하십시오. **com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**.
      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.

타스크 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 상호 작용합니다. 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.

타스크 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.

타스크 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.

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.

타스크 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;
```

관련 태스크

144 페이지의『프록시 클라이언트 생성(.NET)』

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

145 페이지의『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.

보안 추가(.NET)

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

타스크 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 수정사항은 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.

타스크 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.

비즈니스 프로세스 및 타스크 관련 오브젝트에 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);
      }
```

조회 매개변수:

각각의 조회는 다수의 SQL 유사 절 및 매개변수를 지정해야 합니다.

조회는 다음과 같이 구성됩니다.

- Select 절
- Where 절

- Order-by 절
- Skip-tuples 매개변수
- Threshold 매개변수
- Time-zone 매개변수

비즈니스 프로세스 및 휴먼 타스크 오브젝트에서 조회용으로 사전 정의된 보기

비즈니스 프로세스 및 휴먼 타스크 오브젝트에 대해 사전 정의된 데이터베이스 보기가 제공됩니다.

다음 오브젝트에 대한 참조 데이터를 조회하는 경우 다음 보기를 사용하십시오. 해당 보기를 사용할 때 보기 열에 대한 join 술부를 명시적으로 추가할 필요가 없으며 해당 구성체는 자동으로 추가됩니다. 웹 서비스 API의 조회 함수를 사용하여 이 데이터를 조회할 수 있습니다.

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.

타스크 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.

JMS 클라이언트 응용프로그램 개발

JMS(Java Messaging Service) API를 통해 비즈니스 프로세스 응용프로그램에 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

타스크 정보

JMS 소개

WebSphere Process Server 버전 6.1은 JMS(Java Messaging Service) 프로그래밍 인터페이스에 기초하여 통신 메소드로 비동기 메시징을 지원합니다.

JMS는 JMS 메시지로 요청을 작성, 송신, 수신하고 읽는 Java 클라이언트의 일반 방식(클라이언트 응용프로그램 또는 J2EE 응용프로그램)을 제공합니다.

JMS는 다음을 수행하는 비동기 메시지 기반 인터페이스입니다.

- **지점간 또는 publish/subscribe 메시징**을 사용합니다. 메시지 기반 프레임워크는 명시적 요청 없이 다른 응용프로그램에 정보를 푸시할 수 있습니다. 여러 등록자에게 동시에 동일한 정보를 전달할 수 있습니다. Business Process Choreographer의 JMS 인터페이스는 지점간 메시징만을 지원합니다.
- **리듬 독립**을 지원합니다. JMS는 비동기식으로 기능하지만 동기 요청/응답 모드를 시뮬레이트할 수도 있습니다. 이로 인해 소스 및 대상 시스템이 서로 기다리지 않고 동시에 작업할 수 있습니다. 이 기능은 장기 실행 비즈니스 프로세스와 비동기로 상호작용할 수 있는 기능을 제공하기 때문에 Business Process Choreographer에 대단히 유용합니다.
- **트랜잭션**을 지원합니다. 트랜잭션은 클라이언트 응용프로그램이 단일 원자 단위로 송신 또는 수신된 메시지 그룹을 핸들할 수 있게 합니다. JMS 트랜잭션은 서버의 트랜잭션 내에서 실행합니다. Business Process Choreographer의 JMS 인터페이스의 경우 일반적으로 각 트랜잭션에 대한 단일 메시지를 송신 및 수신합니다.
- **정보 전달을 보장합니다**. JMS 프레임워크는 트랜잭션 모드로 메시지를 관리하며 메시지 전달을 보장합니다(적시 전달은 보장되지 않음). Business Process Choreographer의 경우에는 비즈니스 프로세스를 다루기 때문에 신뢰할 수 있는 이 메시지 전달 기능이 특히 중요합니다.
- **이기종 프레임워크 간의 상호운영성**을 보장합니다. 소스 및 대상 응용프로그램이 각 프레임워크에 관련된 통신 및 실행 문제를 핸들할 필요없이 이기종 환경에서 작동할 수 있습니다.
- **보다 유연하게 교환합니다**. 메시지 모드로 전환하면 보다 세밀한 정보를 교환할 수 있습니다.

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 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.

타스크 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 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 API 개요

JMS 메시지 인터페이스(이후 ″JMS API″라 부름)를 사용하면 Business Process Choreographer 환경에서 실행하는 비즈니스 프로세스에 비동기로 액세스하는 클라이언트 응용프로그램을 개발할 수 있습니다.

JMS API는 클라이언트 응용프로그램이 마이크로플로우 및 장기 실행 프로세스와 비동기로 상호 작용할 수 있게 합니다.

JMS API는 웹 서비스 API와 동일한 인터페이스를 노출하며 예외는 다음과 같습니다.

- 웹 서비스 API의 경우 마이크로플로우를 호출할 때에만 call 조작을 사용할 수 있습니다. 그러나 JMS API에서는 call 조작을 사용하여 마이크로플로우 및 장기 실행 프로세스를 모두 호출할 수 있습니다.
- 다음 조작은 JMS API를 통해 노출되지 않습니다.
  - callAsync 조작(연관된 콜백 조작과 함께)
  - completeAndClaimSuccessor 및 getParticipatingTask 조작

예제 - 장기 실행 프로세스 실행

일반 클라이언트 응용프로그램이 장기 실행 프로세스에 대해 작업하려면 다음 단계를 따르십시오.

1. 155 페이지의 『JMS 인터페이스 액세스』에 설명된 대로 JMS 환경을 설정하십시오.
2. 설치된 프로세스 정의 목록을 확보하십시오.
   - queryProcessTemplates를 전송하십시오.
   - 그러면 **ProcessTemplate** 오브젝트의 목록이 리턴됩니다.
3. 시작 활동 목록을 확보하십시오(createInstance="yes"로 선택하거나 수신).
   - getStartActivities를 전송하십시오.
   - 그러면 **InboundOperationTemplate** 오브젝트의 목록이 리턴됩니다.
4. 입력 메시지를 작성하십시오. 이는 환경에 특정하며 사전에 전개된 프로세스 특정 아티팩트를 사용해야 할 수도 있습니다.
5. 프로세스 인스턴스를 작성하십시오.
   - sendMessage를 발행하십시오.

   JMS API의 경우 call 조작을 사용하여 비즈니스 프로세스가 제공하는 장기 실행 요청-응답 조작과 상호작용할 수도 있습니다. 이 조작은 오랜 기간 후에도 조작 결과나 결함을 지정된 응답 대상에 리턴합니다. 따라서 call 조작을 사용하면 프로세스 출력 또는 결함 메시지를 얻기 위해 query 및 getOutputMessage 조작을 사용할 필요가 없습니다.
6. 선택적으로 다음 단계를 반복하여 프로세스 인스턴스로부터 출력 메시지를 확보하십시오.
   - query를 발행하여 프로세스 인스턴스의 완료된 상태를 확보하십시오.
   - getOutputMessage를 발행하십시오.
7. 선택적으로 프로세스에 노출된 추가 조작에 대해 작업하십시오.
   - getWaitingActivities 또는 getActiveEventHandlers는 **InboundOperationTemplate** 오브젝트의 목록을 확보합니다.
   - 입력 메시지를 작성하십시오.

- sendMessage로 메시지를 전송하십시오.

8. 선택적으로 getCustomProperties 및 setCustomProperties를 사용하여 포함된 활동 또는 프로세스에 정의된 사용자 정의 특성을 확보하고 설정하십시오.
9. 선택적으로 프로세스 인스턴스에 대한 작업을 완료하십시오.
   - delete 및 terminate를 전송하여 장기 실행 프로세스에 대한 작업을 완료하십시오.

JMS 응용프로그램 개발

JMS 클라이언트 응용프로그램은 J2EE(Java 2 Enterprise Edition) 환경을 사용하여 Java에 개발해야 합니다.

타스크 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.

아티팩트 복사

WebSphere 환경에서 여러 아티팩트를 복사해서 JMS 클라이언트 응용프로그램을 보다 쉽게 작성할 수 있습니다.

이러한 아티팩트는 BOXMLSerializer를 사용하여 JMS 메시지 본문을 작성하는 경우에만 사용해야 합니다.

두 가지 방법으로 아티팩트를 가져올 수 있습니다.

- WebSphere Process Server 환경에서 아티팩트를 공개하고 내보냅니다.

  WebSphere Process Server 6.1의 경우 모든 클라이언트 아티팩트가 *install_root*₩ProcessChoreographer₩client 디렉토리에 있습니다. JMS API의 경우에는 아티팩트가 다음과 같습니다.

  BFMIF.wsdl

  BFMIF.xsd

  BPCGen.xsd

- WebSphere Process Server 클라이언트 CD에서 파일을 복사합니다.

서버 환경으로부터 아티팩트 공개:

JMS API에 액세스하는 클라이언트 응용프로그램을 개발하기 위해 WebSphere 서버 환경에서 여러 아티팩트를 공개할 수 있습니다.

타스크 정보

WebSphere Process Server 6.1의 경우 모든 클라이언트 아티팩트가 *was_home*₩ProcessChoreographer₩client 디렉토리에 있습니다. JMS API의 경우에는 아티팩트가 다음과 같습니다.

BFMIF.wsdl

BFMIF.xsd

BPCGen.xsd

이 아티팩트를 공개한 후 클라이언트 프로그래밍 환경으로 복사하십시오.

클라이언트 CD에서 파일 복사:

JMS API에 액세스하는 데 필요한 파일은 WebSphere Process Server 클라이언트 CD에서 사용할 수 있습니다.

프로시저

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

   WebSphere Process Server 6.1의 경우 모든 클라이언트 아티팩트가 ₩ProcessChoreographer₩client 디렉토리에 있습니다. JMS API의 경우에는 아티팩트가 다음과 같습니다.

   BFMIF.wsdl

   BFMIF.xsd

   BPCGen.xsd

비즈니스 예외의 응답 메시지 점검

JMS 클라이언트 응용프로그램은 비즈니스 예외에 대한 모든 응답 메시지의 메시지 헤더를 확인해야 합니다.

타스크 정보

JMS 클라이언트 응용프로그램은 먼저 응답 메시지 헤더의 **IsBusinessException** 특성을 확인해야 합니다.

예를 들면 다음과 같은 패널이 표시될 수 있습니다.

```
// 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
}
```

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.

타스크 정보

WebSphere Integration Developer를 사용하여 웹 응용프로그램을 빌드할 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 Explorer JSF 컴포넌트를 추가하십시오.

   a. 응용프로그램에 필요한 태그 라이브러리 참조를 JSP(JavaServer Pages) 파일에 추가하십시오. 일반적으로 JSF 및 HTML 태그 라이브러리 및 JSF 컴포넌트에 필요한 태그 라이브러리가 필요합니다.

      - `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>`
      - `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`
      - `<%@ taglib uri="http://com.ibm.bpe.jsf/taglib" prefix="bpe" %>`

   b. JSP 페이지의 본문에 `<f:view>` 태그를 추가하고 `<f:view>` 태그에 `<h:form>` 태그를 추가하십시오.

   c. JSP 파일에 JSF 컴포넌트를 추가하십시오.

      응용프로그램에 따라 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.

```
<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 상황에서는 예외가 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 테이블은 참조 바인딩 및 해당 기본 맵핑을 나열합니다.

표 33. 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

표 33. JNDI 이름에 대한 참조 바인딩 맵핑 (계속)

참조 바인딩	JNDI 이름	주석
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	로컬 세션 Bean

결과

전개된 웹 응용프로그램에는 Business Process Choreographer Explorer 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.

관련 개념

168 페이지의『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.

관련 태스크

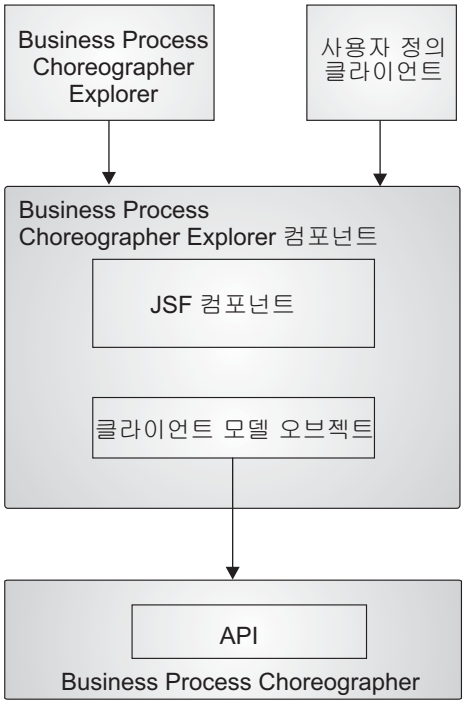
25 페이지의『세션 Bean의 원격 인터페이스 액세스』

EJB 클라이언트 응용프로그램은 Bean의 원격 홈 인터페이스를 통해 세션 Bean의 원격 인터페이스에 액세스합니다.

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();
}
```

관련 개념

174 페이지의『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.

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 테이블은 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 테이블은 Business Process Choreographer 인터페이스의 해당 클라이언트 모델 오브젝트에 대한 맵핑을 보여줍니다.

표 34. 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.
    //
    QueryResultSet queryResult = taskService.query(
     "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND, TASK.STATE, TASK.TYPE,"
      + "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 오리지네이터)를 표시하는 JavaServer 페이지를 포함합니다.

관련 개념

174 페이지의『사용자 특정 시간대 정보』

JSF(JavaServer Faces) 컴포넌트는 List 컴포넌트의 사용자 특정 시간대 정보를 핸들링할 유틸리티를 제공합니다.

관련 참조

175 페이지의『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 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 구현이 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 오류를 일으킨 오브젝트의 getID 메소드에서 리턴된 값이어야 합니다. 맵이 설정되고 모든 ID가 목록에 표시된 모든 항목과 일치하는 경우, 목록 핸들러는 오류 메시지를 포함한 열을 목록에 자동으로 추가합니다.

목록에 날짜가 지난 오류 메시지를 추가하지 않으려면 오류 맵을 재설정하십시오. 다음 상황에서는 맵이 자동으로 재설정됩니다.

- BPCListHandler 클래스의 refreshList 메소드가 호출되는 경우
- BPCListHandler 클래스에 새 조회가 설정되는 경우
- 목록의 항목에 조치를 트리거하는 데 CommandBar 컴포넌트를 사용하는 경우. CommandBar 컴포넌트는 이 메커니즘을 오류 핸들링의 한 방법으로 사용합니다.

  관련 개념

  168 페이지의『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.

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 35. bpe:list 속성

속성	필수	설명
buttonStyleClass	아니오	바닥글 영역의 단추를 렌더링하는 데 사용되는 계단식 스타일시트(CSS) 스타일 클래스
cellStyleClass	아니오	개별 테이블 셀을 표현하기 위한 CSS 스타일 클래스
checkbox	아니오	다중 항목을 선택하기 위한 선택란이 표현되는지 여부를 판별합니다. 이 속성은 true 또는 false 값을 가집니다. 값을 true로 설정하면 선택란 열이 렌더링됩니다.
headerStyleClass	아니오	테이블 헤더를 표현하기 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 표현하기 위한 CSS 스타일 클래스
selectAll	아니오	이 속성을 true로 설정하면 목록의 모든 항목이 기본적으로 선택됩니다.
styleClass	아니오	제목, 행 및 페이징 단추를 포함하여 전체 테이블을 렌더링하는 데 사용되는 CSS 스타일 클래스

표 36. 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 다음과 같습니다. String method().

표 36. bpe:column 속성 (계속)

속성	필수	설명
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 행에 해당 특성이 표시됩니다. Bean 클래스는 JavaBeans 클래스일 수 있습니다. Bean이 기본 변환기 및 특성 레이블을 제공하면 변환기 및 레이블을 사용하여 List 컴포넌트와 동일한 방식으로 렌더링합니다.

결과

JSF 응용프로그램은 이제 지정된 오브젝트의 세부사항(예: 타스크 인스턴스의 세부사항)을 표시하는 JavaServer 페이지를 포함합니다.

관련 참조

『세부사항 컴포넌트: 태그 정의』

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

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 37. 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
styleClass	아니오	HTML 요소를 렌더링하는 데 사용되는 CSS 클래스

표 38. 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.

타스크 정보

사용자가 사용자 인터페이스에서 단추를 누르면 해당 명령이 선택된 오브젝트에 대해 실행됩니다. JSF 응용프로그램에서 CommandBar 컴포넌트를 추가 및 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. 이들 핸들러 중 하나의 구성을 변경했거나 다른 관리 Bean을 사용한 경우, ItemProvider 인터페이스를 구현하는 관리 Bean을 JSF 구성 파일에 추가하십시오.

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 메소

드 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.

관련 참조

183 페이지의『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.

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:command`로 구성됩니다. `bpe:command` 태그는 `bpe:commandbar` 태그의 하위 요소입니다.

컴포넌트 클래스

com.ibm.bpe.jsf.component.CommandBarComponent

예제 구문

```
<bpe:commandbar model="#{TaskInstanceList}">

    <bpe:command
       commandID="Work on"
       label="Work on..."
       commandClass="com.ibm.bpc.explorer.command.WorkOnTaskCommand"
       context="#{TaskInstanceDetailsBean}"/>

    <bpe:command
       commandID="Cancel"
       label="Cancel"
       commandClass="com.ibm.task.clientmodel.command.CancelClaimTaskCommand"
       context="#{TaskInstanceList}"/>

</bpe:commandbar>
```

태그 속성

표 39. 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.

표 39. bpe:commandbar 속성 (계속)

속성	필수	설명
styleClass	아니오	명령 표시줄을 렌더링하는 데 사용되는 CSS 스타일 클래스

표 40. 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 프리미티브 유형을 표현하려면 Business Process Choreographer 탐색기 메시지 컴포넌트를 사용하십시오.

타스크 정보

메시지 유형이 프리미티브 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 프리미티브 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 프리미티브 유형을 표현할 수 있는 JavaServer 페이지를 포함할 수 있습니다.

관련 참조

187 페이지의『메시지 컴포넌트: 태그 정의』

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

메시지 컴포넌트: 태그 정의

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

메시지 컴포넌트는 JSF 컴포넌트 태그 bpe:form으로 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 41. 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 true로 설정하면 읽기 전용 양식이 표현됩니다. 기본적으로 이 속성은 false로 설정됩니다.
simplification	아니오	이 속성을 true로 설정하면 단순 유형을 포함하고 0 이상의 카디널리티가 있는 특성이 표시됩니다. 기본적으로 이 속성은 true로 설정됩니다.
style4validinput	아니오	유효한 입력을 표현하기 위한 캐스케이딩 스타일시트(CSS) 스타일
style4invalidinput	아니오	유효하지 않은 입력을 표현하기 위한 CSS 스타일
styleClass4invalidInput	아니오	유효하지 않은 입력을 렌더링하기 위한 CSS 스타일 클래스 이름
styleClass4output	아니오	출력 요소를 표현하기 위한 CSS 스타일 클래스 이름
styleClass4table	아니오	메시지 컴포넌트가 표현하는 테이블을 표현하기 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.

타스크 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 및 스태프 조회를 처리하는 사용자 정의된 플러그인을 작성할 수 있습니다.

타스크 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.

글로벌 레벨의 모든 타스크, 응용프로그램 컴포넌트의 타스크, 타스크 템플리트와 연관된 모든 타스크, 단일 타스크 인스턴스 등 다양한 레벨에서 플러그인을 등록할 수 있습니다.

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.

타스크 정보

다음 단계를 완료하여 API 이벤트 핸들러를 작성하십시오.

프로시저

1. APIEventHandlerPlugin2 인터페이스를 구현하거나 APIEventHandler 구현 클래스를 확장하는 클래스를 작성하십시오. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.
   - APIEventHandlerPlugin2 인터페이스를 사용할 경우, APIEventHandlerPlugin2 인터페이스 및 APIEventHandlerPlugin 인터페이스의 모든 메소드를 구현해야 합니다.
   - SPI 구현 클래스를 확장할 경우, 필요한 메소드를 대체하십시오.

   이 클래스는 J2EE(Java 2 Enterprise Edition) EJB(Enterprise JavaBeans) 응용프로그램의 컨텍스트에서 실행합니다. 이 클래스 및 클래스의 헬퍼 클래스는 반드시 EJB 스펙을 준수해야 합니다.

   팁: 해당 클래스에서 HumanTaskManagerService 인터페이스를 호출하려는 경우, 이벤트를 생성한 타스크를 갱신하는 메소드를 호출하지 마십시오. 이 조치로 데이터베이스 교착 상태가 발생할 수 있습니다.
2. 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.
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*APIEventHandlerPlugin이라는 이름으로 파일을 작성하십시오. 여기서 *plug-in_name*은 플러그인의 이름입니다.

      예를 들어, 플러그인의 이름이 Customer이고 com.ibm.task.spi.APIEventHandlerPlugin 인터페이스를 구현할 경우, 구성 파일의 이름은 com.ibm.task.spi.**Customer**APIEventHandlerPlugin입니다.

   b. 이 파일의 첫 번째 행(주석 행이나 빈 행이 아닌 행)에서는 1단계에서 작성된 플러그인 클래스의 완전한 이름을 지정하십시오.

      예를 들어, 플러그인 클래스 이름이 MyAPIEventHandler이고 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.

팁: 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.

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 APIEventHandlerPlugin2 인터페이스를 사용하거나 또는 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.

타스크 정보

공고 이벤트 핸들러를 구현하려면, NotificationEventHandlerPlugin 인터페이스를 사용하거나 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) EJB(Enterprise JavaBeans) 응용프로그램의 컨텍스트에서 실행합니다. 이 클래스 및 클래스의 헬퍼 클래스는 반드시 EJB 스펙을 준수해야 합니다.

   플러그인은 EscalationUser 역할의 권한으로 호출됩니다. 이 역할은 휴먼 타스크 컨테이너 구성 시 정의됩니다.

팁: 이 클래스에서 HumanTaskManagerService 인터페이스를 호출하는 경우, 타스크를 갱신하는 메소드 또는 이벤트를 생성한 에스컬레이션은 호출하지 마십시오. 이 조치로 데이터베이스 교착 상태가 발생할 수 있습니다.

2. 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.

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`*`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. 이 파일의 첫 번째 행(주석 행이나 빈 행이 아닌 행)에서는 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.

팁: 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 작성

해야 합니다(예: com.ibm.task.spi.CustomerNotificationEventHandlerPlugin 및 com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

플러그인 구현 및 헬퍼 클래스를 단일 JAR 파일에 패키지하십시오.

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.

개인 조회 결과를 사후 처리할 플러그인 작성

스태프 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.

타스크 정보

하나의 사후 처리 플러그인만이 있을 수 있으며, 이는 플러그인이 모든 타스크의 개인 조회 결과를 핸들해야 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 개인 분석 중에 제거된 사용자에 대한 정보를 수신합니다(HTM_REMOVED_USERS 맵 키에서). 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.

   모든 인터페이스 메소드를 구현해야 합니다. 이 클래스는 다른 클래스의 메소드를 호출할 수 있습니다.

   이 클래스는 J2EE(Java 2 Enterprise Edition) EJB(Enterprise JavaBeans) 응용프로그램의 컨텍스트에서 실행합니다. 이 클래스 및 클래스의 헬퍼 클래스는 반드시 EJB 스펙을 준수해야 합니다.

팁: 해당 클래스에서 HumanTaskManagerService 인터페이스를 호출하려는 경우, 이벤트를 생성한 타스크를 갱신하는 메소드를 호출하지 마십시오. 이 조치로 데이터베이스 교착 상태가 발생할 수 있습니다.

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.

   여러 J2EE 응용프로그램에서 헬퍼 클래스를 사용하는 경우, 공유 라이브러리로 등록한 별도의 JAR 파일로 해당 클래스를 패키지할 수 있습니다.

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. 이 파일의 첫 번째 행(주석 행이나 빈 행이 아닌 행)에서는 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.

5. 플러그인을 등록하십시오.
   a. 관리 콘솔에서 휴먼 타스크 관리자의 사용자 정의 특성 페이지로 이동하십시오 (**Application Server → *server_name* → 휴먼 타스크 컨테이너 → 사용자 정의 특성**).
   b. **Staff.PostProcessorPlugin** 이름의 사용자 정의 특성 및 플러그인에 부여한 이름의 값(이 예제에서는 `MyHandler`)을 추가하십시오.

플러그인 설치

플러그인을 사용하려면 타스크 컨테이너가 플러그인에 액세스할 수 있도록 먼저 플러그인을 설치해야 합니다.

타스크 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 각 서버에서 JAR 파일을 수동으로 분배한 후 각 셀에 대한 공유 라이브러리를 한 번 설치하십시오.

다음에 수행할 작업

이제 플러그인을 등록할 수 있습니다.

플러그인 등록

타스크 컨테이너 아티팩트 계층 구조의 다양한 레벨에서 플러그인을 등록할 수 있습니다. 예를 들어, 글로벌 레벨의 모든 타스크, 응용프로그램 컴포넌트의 타스크, 타스크 템플리트와 연관된 모든 타스크 또는 단일 타스크 인스턴스에 등록할 수 있습니다.

타스크 정보

여러 플러그인을 등록하는 경우 범위 지정이 지원됩니다. 즉, 타스크 컨테이너 아티팩트 계층 구조의 더 높은 레벨(예: 타스크 템플리트 또는 응용프로그램 컴포넌트)에서 등록된 플러그인 대신 해당 계층 구조의 더 낮은 레벨(예: 타스크 인스턴스)에서 등록된 플러그인을 사용합니다. 범위 지정은 모든 계층 구조 레벨에 지원됩니다. 타스크 컨테이너는 계층 구조의 가장 낮은 레벨에서 등록된 플러그인을 사용합니다.

다음 방법 중 한 가지로 플러그인을 등록할 수 있습니다.

- 타스크 모델의 플러그인을 등록하십시오.

  WebSphere Integration Developer의 타스크 편집기에서 타스크에 대한 특성 영역의 세부사항 페이지에 있는 **이벤트 핸들러 이름** 필드에 이벤트 핸들러의 이름을 지정하십시오.
- 런타임 시 작성한 임시 타스크 또는 타스크 템플리트에 대한 플러그인을 등록하십시오.

  TTask 클래스의 setEventHandlerName 메소드를 사용하여 이벤트 핸들러 이름을 등록하십시오.
- 런타임 시 타스크 인스턴스에 등록된 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.
- 글로벌 레벨에서 플러그인을 등록하십시오.

  관리 콘솔에서 휴먼 타스크 컨테이너에 대한 사용자 정의 특성 페이지에서 플러그인에 대한 사용자 정의 특성을 정의하십시오. 사용자 정의 특성 값은 플러그인 이름입니다.

제 2 부 응용프로그램 전개

제 3 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 설치할 수 있도록 하는 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 5에서는 응용프로그램이 컴포넌트 및 연관 라이브러리를 포함하는 방법을 보여줍니다.

그림 5. 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 응용프

로그램을 포함합니다. 응용프로그램을 완전히 설치하려면 이 JAR 파일, JAR 파일과 같은 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 재시작하십시오.

관련 참조

serviceDeploy 명령
serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

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.

관련 참조

serviceDeploy 명령
serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

서버로 전개 준비

모듈을 개발 및 테스트한 후에 모듈을 테스트 시스템에서 내보내어 전개용 프로덕션 환경으로 가져와야 합니다. 응용프로그램을 설치하려면 모듈을 내보낼 때 필요한 경로 및 모듈에서 필요한 라이브러리의 경로를 알고 있어야 합니다.

시작하기 전에

해당 타스크를 시작하기 전에 테스트 서버에서 모듈을 개발 및 테스트하여 문제점이나 성능에 관련된 모든 문제를 먼저 해결해야 합니다.

타스크 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 204 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 나열합니다. 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
```

프로덕션 서버에서 모듈 설치 중프로덕션 서버에서 모듈 설치 중에서 설명한 대로 모듈을 프로덕션 시스템에 설치하십시오.

관련 참조

serviceDeploy 명령

serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

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 4 장 프로덕션 서버에 모듈 설치

이 주제에서는 테스트 서버에서 응용프로그램을 가져오고 프로덕션 환경에 이를 배치하는 데 관련된 단계를 설명합니다.

시작하기 전에

프로덕션 서버에 서비스 응용프로그램을 전개하기 전에 테스트 서버에 응용프로그램을 어셈블하고 테스트하십시오. 테스트 후 모듈 개발 및 전개 PDF의 *서버에 전개 준비*에 설명된 대로 관련 파일을 내보내고 전개할 프로덕션 시스템으로 파일을 가져오십시오. 자세한 정보는 WebSphere Integration Developer 및 WebSphere Application Server Network Deployment의 Information Center를 참조하십시오.

프로시저

1. 프로덕션 서버에 모듈 및 기타 파일을 복사하십시오.

   응용프로그램에서 필요한 모듈 및 자원(EAR, JAR, RAR 및 WAR 파일)이 프로덕션 환경으로 이동됩니다.
2. serviceDeploy 명령을 실행하여 설치 가능한 EAR 파일을 작성하십시오.

   이 단계는 프로덕션에 응용프로그램을 설치하기 위한 준비로 서버에 모듈을 정의합니다.

   a. 전개할 모듈을 포함하는 JAR 파일을 찾으십시오.

   b. 이전 단계의 JAR 파일을 입력으로 사용하여 명령을 발행하십시오.
3. 2단계에서 작성한 EAR 파일을 설치하십시오. 응용프로그램 설치 방법은 응용프로그램을 독립형 서버에 설치하는지 또는 셀에 있는 서버에 설치하는지에 따라 달라집니다.

   주: 관리 콘솔 또는 스크립트를 사용하여 응용프로그램을 설치할 수 있습니다. 추가 정보는 WebSphere Application Server Information Center를 참조하십시오.
4. 구성을 저장하십시오. 이제 모듈이 응용프로그램으로 설치됩니다.
5. 응용프로그램을 시작하십시오.

결과

이제 응용프로그램이 활성화되고 작업은 모듈을 통해 플로우되어야 합니다.

다음에 수행할 작업

응용프로그램을 모니터하여 서버가 요청을 올바로 처리하는지 확인하십시오.

관련 참조

serviceDeploy 명령
serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

serviceDeploy를 사용하여 설치 가능한 EAR 파일 작성

프로덕션 환경에 응용프로그램을 설치하려면, 파일을 프로덕션 서버로 복사하고 설치 가능한 EAR 파일을 작성하십시오.

시작하기 전에

이 타스크를 시작하기 전에, 서버에 전개 중인 모듈 및 서비스를 포함하는 JAR 파일이 있어야 합니다. 자세한 정보를 보려면 서버에 전개 준비를 참조하십시오.

타스크 정보

serviceDeploy 명령은 JAR 파일, 기타 종속 EAR, JAR, RAR, WAR 및 ZIP 파일을 가져오고 서버에 설치할 수 있는 EAR 파일을 빌드합니다.

프로시저

1. 전개할 모듈을 포함하는 JAR 파일을 찾으십시오.
2. 이전 단계의 JAR 파일을 입력으로 사용하여 명령을 발행하십시오.

   이 단계는 EAR 파일을 작성합니다.

   주: 관리 콘솔에서 다음 단계를 수행하십시오.
3. 서버의 관리 콘솔에 설치할 EAR 파일을 선택하십시오.
4. **저장**을 클릭하여 EAR 파일을 설치하십시오.

관련 참조

serviceDeploy 명령
serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

Apache Ant 타스크를 사용하여 응용프로그램 전개

이 주제에서는 WebSphere Process Server로 응용프로그램의 전개를 자동화하기 위해 Apache™ Ant 타스크를 사용하는 방법에 대해 설명합니다. Apache Ant 타스크를 사용하여 여러 응용프로그램의 전개를 정의하고 서버에서 자동으로 실행되도록 할 수 있습니다.

시작하기 전에

이 타스크는 다음 사항을 가정합니다.
- 전개 중인 응용프로그램이 이미 개발 및 테스트되었습니다.
- 응용프로그램이 동일한 서버에 설치됩니다.
- Apache Ant 타스크에 대한 지식이 있습니다.
- 사용자는 전개 프로세스를 이해하고 있습니다.

응용프로그램 개발 및 테스트에 대한 정보는 WebSphere Integration Developer Information Center에서 보실 수 있습니다.

WebSphere Application Server Network Deployment의 Information Center 참조 부분에는 API(application programming interface)에 대한 섹션이 있습니다. Apache Ant 타스크는 com.ibm.websphere.ant.tasks 패키지에 설명되어 있습니다. 이 주제에서 연관되는 타스크는 ServiceDeploy 및 InstallApplication입니다.

타스크 정보

여러 응용프로그램을 동시에 설치해야 하는 경우, 전개하기 전에 Apache Ant 타스크를 개발하십시오. 그러면 Apache Ant 타스크는 프로세스에 사용자의 개입 없이 서버에서 응용프로그램을 전개하고 설치할 수 있습니다.

프로시저

1. 전개할 응용프로그램을 식별하십시오.
2. 각 응용프로그램에 대해 Jar 파일을 작성하십시오.
3. Jar 파일을 대상 서버에 복사하십시오.
4. 각 서버에 대한 EAR 파일을 작성하기 위해 ServiceDeploy 명령을 실행하려면 Apache Ant 타스크를 작성하십시오.
5. Application Server에서 4단계의 각 EAR 파일에 대해 InstallApplication 명령을 실행하려면 Apache Ant 타스크를 작성하십시오.
6. ServiceDeploy Apache Ant 타스크를 실행하여 응용프로그램에 대해 EAR 파일을 작성하십시오.
7. InstallApplication Apache Ant 타스크를 실행하여 6단계의 EAR 파일을 설치하십시오.

결과

응용프로그램이 대상 서버에서 올바르게 전개됩니다.

응용프로그램 자동 전개의 예제

이 예제는 myBuildScript.xml 파일에 포함된 Apache Ant 타스크를 나타내고 있습니다.

```
<?xml version="1.0">

<project name="OwnTaskExample" default="main" basedir=".">
 <taskdef name="servicedeploy"
   classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
 <target name="main" depends="main2">
  <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
   ignoreErrors="true"
   outputApplication="c:/synctest/SyncTargetEAREAR"
   workingDirectory="c:/synctest"
   noJ2eeDeploy="true"
   cleanStagingModules="true"/>
 </target>
</project>
```

이 명령문은 Apache Ant 타스크를 호출하는 방법을 나타냅니다.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

팁: 추가 프로젝트 명령문을 파일에 추가하여 여러 응용프로그램을 자동으로 전개할 수 있습니다.

다음에 수행할 작업

관리 콘솔을 사용하여 최근에 설치된 응용프로그램이 시작되어 워크플로우를 제대로 처리하고 있는지 확인하십시오.

관련 참조

serviceDeploy 명령

serviceDeploy 명령을 사용하여 SCA(Service Component Architecture) 준수 모듈을 서버에 설치할 수 있는 Java 응용프로그램으로 패키지할 수 있습니다. 이 명령은 wsadmin을 통해 일괄처리 설치를 수행한 경우 유용합니다.

제 5 장 비즈니스 프로세스 및 휴먼 타스크 응용프로그램 설치

비즈니스 프로세스나 휴먼 타스크 또는 둘 모두를 포함하는 SCA(Service Component Architecture) 모듈을 전개 대상에 분배할 수 있습니다. 전개 대상은 서버 또는 클러스터입니다.

시작하기 전에

응용프로그램을 설치하려는 각 Application Server 또는 클러스터에 대해 비즈니스 플로우 관리자, 휴먼 타스크 관리자 또는 둘 모두가 설치 및 구성되어 있는지 확인하십시오.

타스크 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.

관련 개념

212 페이지의『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.

212 페이지의『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.

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 스테이지에 비동기로 설치됩니다. 다음 스테이지를 시작하려면 각 스테이지를 완료해야 합니다.

1. Deployment Manager에서 응용프로그램 설치가 시작됩니다.

   이 스테이지 동안 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트는 WebSphere 구성 저장소에 구성됩니다. 응용프로그램의 유효성도 검사합니다. 오류가 발생하면 Deployment Manager의 FFDC 항목으로 또는 System.out 파일이나 System.err 파일에 오류가 보고됩니다.

2. Node Agent에서 응용프로그램 설치가 계속됩니다.

   이 스테이지 중에는 한 Application Server 인스턴스의 응용프로그램 설치가 트리거됩니다. 이 Application Server 인스턴스는 전개 대상이거나 대상의 일부입니다. 전개 대상이 여러 클러스터 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.

관련 태스크

211 페이지의 제 5 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.

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.

관련 태스크

211 페이지의 제 5 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.

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.

타스크 정보

비즈니스 프로세스 응용프로그램을 대화식으로 설치하려면 다음 단계를 수행하십시오.

프로시저

1. wsadmin 도구를 시작하십시오.

   profile_root/bin 디렉토리에 `wsadmin`을 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.

타스크 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 액세스해야 합니다.
- 응용프로그램이 관리 서버에 설치된 경우에는 Deployment Manager 및 이 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.
- 응용프로그램에 속하는 모든 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트가 중지 상태에 있어야 합니다.
- 임의의 상태에 있는 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없습니다.

개발 및 유닛 테스트 환경으로 사용되는 독립형 서버 환경의 경우 개발 모드에서 실행되도록 서버를 구성할 수 있습니다. 이 구성의 경우 템플리트를 중지하고 인스턴스를 표시하지 않을 필요가 없습니다. 그러나 프로덕션 환경에는 이 구성이 유효하지 않습니다.

타스크 정보

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

프로시저

1. 응용프로그램에서 모든 프로세스 및 타스크 템플리트를 중지하십시오.

   이 조치를 통해 프로세스 및 타스크 인스턴스 생성을 방지할 수 있습니다.

   a. 관리 콘솔 탐색 패널에서 **응용프로그램 → SCA 모듈**을 클릭하십시오.
   b. 중지하려는 템플리트가 포함된 모듈을 선택하십시오.
   c. 추가 특성에서 **비즈니스 프로세스** 또는 **휴먼 타스크** 또는 둘 다를 클릭하십시오.
   d. 해당 선택란을 클릭하여 모든 프로세스 및 타스크 템플리트를 선택하십시오.
   e. **중지**를 클릭하십시오.

   비즈니스 프로세스 템플리트 또는 휴먼 타스크 템플리트를 포함하는 모든 EJB 모듈에 대해 다음 단계를 반복하십시오.
2. 응용프로그램을 전개한 시스템에서 데이터베이스, 각 클러스터에 대한 하나 이상의 응용프로그램 서버, 독립형 서버가 실행 중인지 확인하십시오.

Network Deployment 환경에서 Depoyment Manager, 모든 관리 독립형 Application Server 및 최소 하나의 Application Server가 응용프로그램이 설치되는 각 클러스터에 대해 실행 중이어야 합니다.

3. 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.

4. 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 액세스해야 합니다.
- 응용프로그램이 관리 서버에 설치된 경우에는 Deployment Manager 및 이 서버가 실행 중이어야 합니다. 서버에 Business Process Choreographer 데이터베이스에 대한 액세스 권한이 있어야 합니다.

- 응용프로그램에 속하는 모든 비즈니스 프로세스 템플리트 및 휴먼 타스크 템플리트가 중지 상태에 있어야 합니다.
- 임의의 상태에 있는 비즈니스 프로세스 또는 휴먼 타스크 템플리트의 인스턴스가 없습니다.

개발 및 유닛 테스트 환경으로 사용되는 독립형 서버 환경의 경우 개발 모드에서 실행되도록 서버를 구성할 수 있습니다. 이 구성의 경우 템플리트를 중지하고 인스턴스를 표시하지 않을 필요가 없습니다. 그러나 프로덕션 환경에는 이 구성이 유효하지 않습니다.

글로벌 보안이 사용되는 경우에는 사용자 ID에 운영자 권한이 있는지도 확인하십시오.

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

타스크 정보

다음 단계에서는 bpcTemplates.jacl 스크립트를 사용하여 비즈니스 프로세스 템플리트 또는 휴먼 타스크 템플리트를 포함하는 응용프로그램을 설치 제거하는 방법에 대해 설명합니다. 해당 응용프로그램을 설치 제거하기 전에 템플리트를 중지해야 합니다. bpcTemplates.jacl 스크립트를 사용하여 위의 템플리트를 중지하고 설치 제거할 수 있습니다.

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
글로벌 보안이 사용 가능한 경우, 인증을 위한 사용자 ID입니다.

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

application_name
글로벌 보안이 사용 가능한 경우, 인증을 위한 사용자 암호입니다.

결과

응용프로그램이 설치 제거됩니다.

제 6 장 어댑터 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 7 장 EIS 응용프로그램 설치

EIS 응용프로그램 모듈을 J2EE 플랫폼에 전개할 수 있습니다. J2EE 플랫폼에 EIS 모듈을 전개하면 서버에 전개된 EAR 파일로 패키지된 응용프로그램이 됩니다. J2EE 아티팩트와 자원이 모두 작성되고, 응용프로그램이 구성되어 실행할 준비가 됩니다.

타스크 정보

J2EE 플랫폼에 전개하면 J2EE 아티팩트 및 자원이 작성됩니다.

표 42. 바인딩에서 J2EE 아티팩트로 맵핑

SCA 모듈의 바인딩	**생성된 J2EE 아티팩트**	**작성된 J2EE 자원**
EIS 가져오기	모듈 세션 EJB에 자원 참조 생성	ConnectionFactory
EIS 내보내기	자원 어댑터가 지원하는 리스너 인터페이스에 따라 MDB(Message Driven Bean) 생성 또는 전개	ActivationSpec
JMS 가져오기	런타임이 제공하는 MDB(Message Driven Bean)가 전개되고 모듈 세션 EJB에 자원 참조가 생성됩니다. 가져오기에 수신 목적지가 있는 경우에만 MDB가 작성되는 점을 참고하십시오.	• ConnectionFactory • ActivationSpec • Destinations
JMS 내보내기	런타임이 제공하는 MDB(Message Driven Bean)가 전개되고 모듈 세션 EJB에 자원 참조가 생성됩니다.	• ActivationSpec • ConnectionFactory • Destinations

가져오기 또는 내보내기가 ConnectionFactory 같은 자원을 정의할 때 모듈 Stateless 세션 EJB의 전개 설명자에 자원 참조가 생성됩니다. 또한 EJB 바인딩 파일에 해당 바인딩이 생성됩니다. 자원 참조가 바인드되는 이름은 대상 속성의 값(대상 속성이 존재하는 경우) 또는 모듈 이름 및 가져오기 이름을 기반으로 자원에 제공된 기본 JNDI 검색 이름입니다.

전개되면 구현이 모듈 세션 Bean을 찾아 자원을 검색합니다.

서버에 응용프로그램을 전개하는 중에 EIS 설치 타스크는 바인드된 요소 자원이 있는지 확인합니다. 요소 자원이 없고 SCDL 파일이 최소 하나의 특성을 지정하는 경우, EIS 설치 타스크에 의해 자원이 작성되고 구성됩니다. 자원이 없는 경우, 응용프로그램 실행 전에 자원이 작성된다고 가정하고 조치가 수행되지 않습니다.

수신 목적지를 사용하여 JMS 가져오기가 전개될 때 MDB(Message Driver Bean)가 전개됩니다. MDB는 전송된 요청에 대한 응답을 청취합니다. MDB는 JMS 메시지의

JMSreplyTo 헤더 필드에서 요청과 함께 전송된 목적지와 연관됩니다(청취합니다). 응답 메시지가 도착하면 MDB는 상관 ID를 사용하여 콜백 목적지에 저장된 콜백 정보를 검색한 후 콜백 오브젝트를 호출합니다.

설치 타스크는 가져오기 파일의 정보에서 ConnectionFactory 및 세 개의 목적지를 작성합니다. 또한 런타임 MDB가 수신 목적지에서 응답을 청취할 수 있도록 ActivationSpec을 작성합니다. ActivationSpec 특성은 목적지/ConnectionFactory 특성에서 파생됩니다. JMS 프로바이더가 SIBus 자원 어댑터인 경우, JMS 목적지에 대응하는 SIBus 목적지가 작성됩니다.

JMS 내보내기가 전개될 때 MDB(JMS 가져오기의 경우에 전개된 MDB와 동일하지 않음)가 전개됩니다. MDB는 수신 목적지에서 수신 요청을 청취한 후 SCA에서 처리하도록 요청을 디스패치합니다. 설치 타스크는 JMS 가져오기의 경우와 유사한 자원 세트(ActivationSpec, 응답 전송에 사용되는 ConnectionFactory 및 두 개의 목적지)를 작성합니다. 이 자원의 특성은 모두 내보내기 파일에 지정됩니다. JMS 프로바이더가 SIBus 자원 어댑터인 경우, JMS 목적지에 대응하는 SIBus 목적지가 작성됩니다.

J2SE 플랫폼에 EIS 응용프로그램 모듈 전개

EIS 모듈을 J2SE 플랫폼에 전개할 수는 있지만 EIS 가져오기만 지원됩니다.

시작하기 전에

이 타스크를 시작하기 전에 WebSphere Integration Development 환경에서 JMS 가져오기 바인딩을 사용하여 EIS 응용프로그램 모듈을 작성해야 합니다.

타스크 정보

메시지 대기열을 사용하여 비동기로 EIS 시스템에 액세스할 경우 JMS 가져오기와 함께 EIS 응용프로그램 모듈이 제공됩니다.

J2SE 플랫폼에 전개하는 것은 비관리 노드에서 바인딩 구현을 실행할 수 있는 유일한 인스턴스입니다. JMS 바인딩에는 비동기 및 JNDI 지원이 필요하며, 기본 SCA(Service Component Architecture) 또는 J2SE는 이 중 어느 것도 제공하지 않습니다. J2EE 커넥터 아키텍처는 비관리 인바운드 통신을 지원하지 않으므로 EIS 내보내기는 배제됩니다.

EIS 가져오기를 사용하는 EIS 응용프로그램 모듈을 J2SE에 전개할 때, 모듈 종속성 외에도 가져오기에 사용되는 WebSphere Adapter를 Manifest나 SCA에서 지원되는 기타 양식에서 종속성으로 지정해야 합니다.

J2EE 플랫폼에 EIS 응용프로그램 모듈 전개

J2EE 플랫폼에 EIS 모듈을 전개하면 서버에 전개된 EAR 파일로 패키지된 응용프로그램이 됩니다. J2EE 아티팩트와 자원이 모두 작성되고, 응용프로그램이 구성되어 실행할 준비가 됩니다.

시작하기 전에

이 타스크를 시작하기 전에 WebSphere Integration Development 환경에서 JMS 가져오기 바인딩을 사용하여 EIS 모듈을 작성해야 합니다.

타스크 정보

J2EE 플랫폼에 전개하면 J2EE 아티팩트 및 자원이 작성됩니다.

표 43. 바인딩에서 J2EE 아티팩트로 맵핑

SCA 모듈의 바인딩	생성된 J2EE 아티팩트	작성된 J2EE 자원
EIS 가져오기	모듈 세션 EJB에 자원 참조 생성	ConnectionFactory
EIS 내보내기	자원 어댑터가 지원하는 리스너 인터페이스에 따라 MDB(Message Driven Bean) 생성 또는 전개	ActivationSpec
JMS 가져오기	런타임이 제공하는 MDB(Message Driven Bean)가 전개되고 모듈 세션 EJB에 자원 참조가 생성됩니다. 가져오기에 수신 목적지가 있는 경우에만 MDB가 작성되는 점을 참고하십시오.	• ConnectionFactory • ActivationSpec • Destinations
JMS 내보내기	런타임이 제공하는 MDB(Message Driven Bean)가 전개되고 모듈 세션 EJB에 자원 참조가 생성됩니다.	• ActivationSpec • ConnectionFactory • Destinations

가져오기 또는 내보내기가 ConnectionFactory 같은 자원을 정의할 때 모듈 Stateless 세션 EJB의 전개 설명자에 자원 참조가 생성됩니다. 또한 EJB 바인딩 파일에 해당 바인딩이 생성됩니다. 자원 참조가 바인드되는 이름은 대상 속성의 값(대상 속성이 존재하는 경우) 또는 모듈 이름 및 가져오기 이름을 기반으로 자원에 제공된 기본 JNDI 검색 이름입니다.

전개되면 구현이 모듈 세션 Bean을 찾아 자원을 검색합니다.

서버에 응용프로그램을 전개하는 중에 EIS 설치 타스크는 바인드된 요소 자원이 있는지 확인합니다. 요소 자원이 없고 SCDL 파일이 최소 하나의 특성을 지정하는 경우, EIS 설치 타스크에 의해 자원이 작성되고 구성됩니다. 자원이 없는 경우, 응용프로그램 실행 전에 자원이 작성된다고 가정하고 조치가 수행되지 않습니다.

수신 목적지를 사용하여 JMS 가져오기가 전개될 때 MDB(Message Driver Bean)가 전개됩니다. MDB는 전송된 요청에 대한 응답을 청취합니다. MDB는 JMS 메시지의

JMSreplyTo 헤더 필드에서 요청과 함께 전송된 목적지와 연관됩니다(청취합니다). 응답 메시지가 도착하면 MDB는 상관 ID를 사용하여 콜백 목적지에 저장된 콜백 정보를 검색한 후 콜백 오브젝트를 호출합니다.

설치 타스크는 가져오기 파일의 정보에서 ConnectionFactory 및 세 개의 목적지를 작성합니다. 또한 런타임 MDB가 수신 목적지에서 응답을 청취할 수 있도록 ActivationSpec을 작성합니다. ActivationSpec 특성은 목적지/ConnectionFactory 특성에서 파생됩니다. JMS 프로바이더가 SIBus 자원 어댑터인 경우, JMS 목적지에 대응하는 SIBus 목적지가 작성됩니다.

JMS 내보내기가 전개될 때 MDB(JMS 가져오기의 경우에 전개된 MDB와 동일하지 않음)가 전개됩니다. MDB는 수신 목적지에서 수신 요청을 청취한 후 SCA에서 처리하도록 요청을 디스패치합니다. 설치 타스크는 JMS 가져오기의 경우와 유사한 자원 세트(ActivationSpec, 응답 전송에 사용되는 ConnectionFactory 및 두 개의 목적지)를 작성합니다. 이 자원의 특성은 모두 내보내기 파일에 지정됩니다. JMS 프로바이더가 SIBus 자원 어댑터인 경우, JMS 목적지에 대응하는 SIBus 목적지가 작성됩니다.

제 8 장 실패한 전개 문제점 해결

이 주제에서는 응용프로그램을 전개할 때 문제점의 원인을 판별하기 위해 수행하는 단계에 대해 설명합니다. 또한 몇 가지 가능한 해결 방법을 제시합니다.

시작하기 전에

이 주제에서는 다음 사항을 가정합니다.
- 사용자가 모듈 디버깅에 대한 기본사항을 이해하고 있습니다.
- 로깅 및 추적이 모듈이 전개되는 동안 활성화됩니다.

타스크 정보

전개의 문제점 해결 타스크는 오류 공고를 수신한 후에 시작됩니다. 조치를 수행하기 전에 검사해야 하는 실패한 전개의 증상은 여러 가지가 있습니다.

프로시저

1. 응용프로그램 설치가 실패하였는지 판별하십시오.

   장애 원인을 지정하는 메시지를 SystemOut.log 파일에서 검사하십시오. 응용프로그램이 설치되지 않는 몇 가지 이유에 다음이 포함됩니다.
   - 동일한 Network Deployment 셀의 다중 서버에 응용프로그램을 설치하려고 합니다.
   - 응용프로그램을 설치하려는 Network Deployment 셀에 기존 모듈과 동일한 이름의 응용프로그램을 설치하려고 합니다.
   - EAR 파일의 J2EE 모듈을 다른 대상 서버에 전개하려고 합니다.

   중요사항: 설치에 실패했으며 응용프로그램에 서비스가 포함된 경우, 응용프로그램을 재설치하기 전에 먼저 실패 이전에 작성된 모든 SIBus 목적지 또는 J2C 활성화 스펙을 제거해야 합니다. 이 아티팩트를 제거하는 가장 간단한 방법은 실패 후 **저장 > 모두 버리기**를 클릭하는 것입니다. 변경사항을 실수로 저장한 경우 SIBus 목적지 및 J2C 활성화 스펙을 수동으로 제거해야 합니다(관리 섹션의 SIBus 목적지 삭제 및 J2C 활성화 스펙 삭제 참조).
2. 응용프로그램이 설치된 경우, 시작되는지 확인하십시오.

   응용프로그램이 시작되지 않으면 서버가 응용프로그램에 대한 자원을 시작할 때 실패합니다.

   a. 진행 방법을 보여주는 메시지를 SystemOut.log 파일에서 검사하십시오.

   b. 응용프로그램에 필요한 자원이 사용 가능하고 시작되었는지 확인하십시오.

자원이 시작되지 않으면 응용프로그램이 실행되지 않습니다. 이는 정보를 유실하지 않도록 보호하기 위해서입니다. 자원이 시작되지 않는 이유는 다음과 같습니다.

- 바인딩이 잘못 지정되었습니다.
- 자원이 올바로 구성되지 않았습니다.
- 자원이 RAR(resource archive) 파일에 포함되지 않았습니다.
- 웹 자원이 WAR(Web services archive) 파일에 포함되지 않았습니다.

c. 누락된 컴포넌트가 있는지 판별하십시오.

컴포넌트 누락은 EAR(enterprise archive) 파일이 제대로 빌드되지 않았기 때문입니다. 모듈에 필요한 모든 컴포넌트가 JAR(Java Archive) 파일을 빌드한 테스트 시스템에 올바른 폴더에 있는지 확인하십시오. 『서버에 전개 준비』에는 추가 정보가 포함되어 있습니다.

3. 응용프로그램을 통해 플로우되는 정보가 있는지 확인하십시오.

실행 중인 응용프로그램도 정보 처리에 실패할 수 있습니다. 이유는 227 페이지의 2b단계에서 언급된 내용과 비슷합니다.

a. 응용프로그램이 다른 응용프로그램에 있는 서비스를 사용하는지 판별하십시오. 다른 응용프로그램이 설치되어 시작되었는지 확인하십시오.

b. 실행하지 못한 응용프로그램에서 사용하는 다른 응용프로그램에 있는 장치에 대한 가져오기 및 내보내기 바인딩이 올바로 구성되어 있는지 판별하십시오. 관리 콘솔을 사용하여 바인딩을 살펴보고 정정하십시오.

4. 문제점을 정정하고 응용프로그램을 다시 시작하십시오.

J2C 활성화 스펙 삭제

시스템에서는 서비스가 포함된 응용프로그램을 설치할 때 J2C 응용프로그램 스펙을 빌드합니다. 응용프로그램을 재설치하기 전에 이 스펙을 삭제해야 하는 경우가 종종 있습니다.

시작하기 전에

응용프로그램 설치에 실패하여 스펙을 삭제하는 경우 JNDI(Java Naming and Directory Interface) 이름에 있는 모듈이 설치에 실패한 모듈의 이름과 반드시 일치해야 합니다. JNDI 이름의 두 번째 부분은 이 목적지를 구현한 모듈의 이름입니다. 예를 들어, sca/SimpleBOCrsmA/ActivationSpec에서 **SimpleBOCrsmA**는 모듈 이름입니다.

이 타스크에 필요한 보안 역할: 보안 및 역할 기반 권한이 사용 가능한 경우, 이 타스크를 수행하려면 관리자 또는 구성자로 로그인해야 합니다.

타스크 정보

서비스가 들어 있는 응용프로그램을 설치한 후에 구성을 무심코 저장한 경우 또는 스펙이 필요하지 않은 경우에 J2C 활성화 스펙을 삭제하십시오.

프로시저

1. 삭제할 활성화 스펙을 찾으십시오.

   이 스펙은 자원 어댑터 패널에 들어 있습니다. **자원 > 자원 어댑터**를 눌러 이 패널을 탐색하십시오.

   a. **플랫폼 메시징 컴포넌트 SPI 자원 어댑터**를 찾으십시오.

      이 어댑터를 찾으려면 독립형 서버에 대한 **노드** 범위 또는 전개 환경의 **서버** 범위에 있어야 합니다.

2. 플랫폼 메시징 컴포넌트 SPI 자원 어댑터와 연관된 J2C 활성화 스펙을 표시하십시오.

   자원 어댑터 이름을 누르십시오. 다음 패널에 연관된 스펙이 표시됩니다.

3. 삭제 중인 모듈 이름과 일치하는 **JNDI 이름**이 있는 모든 스펙을 삭제하십시오.

   a. 해당 스펙 옆에 있는 선택란을 누르십시오.

   b. **삭제**를 누르십시오.

결과

시스템이 표시화면에서 선택된 스펙을 제거합니다.

다음에 수행할 작업

변경사항을 저장하십시오.

SIBus 목적지 삭제

SIBus 목적지는 응용프로그램에 서비스를 사용할 수 있게 하는 연결입니다. 목적지를 제거해야 하는 때가 있습니다.

시작하기 전에

응용프로그램 설치에 실패하여 목적지를 삭제하는 경우, 목적지 이름에 있는 모듈이 설치에 실패한 모듈의 이름과 반드시 일치해야 합니다. 목적지의 두 번째 부분은 이 목적지를 구현한 모듈의 이름입니다. 예를 들어, sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer에서 **SimpleBOCrsmA**는 모듈 이름입니다.

이 타스크에 필요한 보안 역할: 보안 및 역할 기반 권한이 사용 가능한 경우, 이 타스크를 수행하려면 관리자 또는 구성자로 로그인해야 합니다.

타스크 정보

서비스가 들어 있는 응용프로그램을 설치한 후에 구성을 무심코 저장한 경우 또는 목적지가 더 이상 필요하지 않은 경우에 SIBus 목적지를 삭제하십시오.

주: 이 타스크는 SCA 시스템 버스에서만 목적지를 삭제합니다. 서비스가 들어 있는 응용프로그램을 재설치하기 전에 응용프로그램 버스에서도 항목을 제거해야 합니다. (이 Information Center의 관리 섹션에 있는 J2C 활성화 스펙 삭제를 참조하십시오.)

프로시저

1. 관리 콘솔에 로그인하십시오.
2. SCA 시스템 버스의 목적지를 표시하십시오.

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

   표시화면에서 **SCA.SYSTEM.*cellname*.Bus**를 누르십시오. 여기서 *cellname*은 삭제 중인 목적지에 대한 모듈이 들어 있는 셀 이름입니다.
4. 제거 중인 모듈과 일치하는 모듈 이름이 들어 있는 목적지를 삭제하십시오.
   a. 해당 목적지 옆에 있는 선택란을 누르십시오.
   b. **삭제**를 누르십시오.

결과

패널에는 남아 있는 목적지만 표시됩니다.

다음에 수행할 작업

이들 목적지를 작성한 모듈에 관련된 J2C 활성화 스펙을 삭제하십시오.

제 3 부 부록

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 나라에서는 이 자료에 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.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 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 "현상태대로" 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 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

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

이 정보에 기술된 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 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 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 가지 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용 없이 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: (c) (회사명) (연도). 이 코드의 일부는 IBM Corp.의 샘플 프로그램에서 파생됩니다. (c) Copyright IBM Corp. _연도_. All rights reserved.

이 정보를 소프트카피로 확인하는 경우, 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 프로그래밍 인터페이스로서 사용될 수 없습니다.

상표 및 서비스표

IBM, IBM 로고, developerWorks, WebSphere 및 z/OS는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 등록상표입니다.

Adobe는 미국 또는 기타 국가에서 사용되는 Adobe Systems Incorporated의 등록상표입니다.

Java 및 모든 Java 기반 상표는 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)

IBM 멀티플랫폼용 WebSphere Process Server, 버전 6.1.0