Использование EjbProxy

Вызывайте удалённые компоненты EJB с помощью простого вспомогательного класса.

Tips 'N Tricks

Горсен Хуанг

Эта подсказка познакомит вас со вспомогательным классом `EjbProxy`. Он создаёт и вызывает удалённый компонент EJB без непосредственной привязки вашего кода к определённой реализации конкретного компонента. *(В оригинальной версии на английском языке 1400 слов)*

Если вы когда-либо работали с корпоративными компонентами (Enterprise JavaBeans — EJB), то вам, вероятно, приходилось писать код для доступа к компоненту из клиентской програмы или другого компонента. Обычно, эта реализация похожа на все остальные аналогичные, за исключением небольших изменений. Хорошие программисты решают этот вопрос методом копирования и вставки одних и тех же фрагментов (как одним из видов повторного использования кода), тем не менее, после копирования и вставки одного и того же кода много раз это становится скучно. Поэтому было решено реализовать класс, названный `EjbProxy`, который бы решал эти проблемы. Для поддержания простоты и идеологической чистоты, этот класс вызывает любые компоненты не зная ничего, кроме имени компонента в JNDI (Java Naming and Directory Interface — интерфейс службы имён и каталогов), и вы можете использовать его на стороне клиента или в других компонентах EJB.

Вы должны выполнить четыре основных шага, чтобы вызвать компонент:

1. Создать начальный контекст окружения JNDI
2. Найти в полученном контексте имя компонента
3. Получить домашний (`home`) объект
4. Создать экземпляр компонента через метод создания (`create()`) домашнего

`home`) объекта

Реализация шагов с первого по третий элементарна:

```
InitialContext ctx = new InitialContext (prop);
Object home = ctx.lookup(beanName);
EJBHome obHome =
  (EJBHome) PortableRemoteObject.narrow (home, EJBHome.class);
```

Как вы видите, вы можете получить домашний объект (`home`) не зная специфики реализации конкретного компонента, за исключением имени для поиска в JNDI.

Шаг 4 не такой простой. Когда вы пытаетесь создать объект компонента, используя домашний объект, вы не можете вызвать метод `create()` из домашнего объекта, так как интерфейс `EJBHome` не определяет метода `create()`. Только реальная реализация компонента определяет и реализует этот метод.

Как одно из возможных решений, вы можете определить интерфейс, который расширяет `EJBHome` и объявляет метод `create()`. Тогда вы можете просто привести тип домашнего объекта к этому интерфейсу и вызвать метод `create()`. Тем не менее, такой подход имеет один недостаток: он требует, что бы все компоненты реализовывали этот специальный интерфейс. Вам может понадобиться вызвать метод `create()` из произвольного домашнего интерфейса.

Наилучшим решением будет использование возможностей прикладного интерфейса отражения Java (Java Reflection API). Вы можете обратиться к Ресурсам для получения подробной информации об отражении. Используя отражение, вы можете загружать заданный класс, используя `Class.forName`, и затем вызывать `getDeclaredMethod()` для получения определённого в классе метода. После получения объекта `Method`, вы можете вызывать метод с помощью `Method.invoke`.

Следующий код использует отражение для создания объекта компонента из домашнего объекта:

```
Method m = obHome.getClass().
        getDeclaredMethod("create", new Class[0]);
Object objRemote = m.invoke (obHome, new Object[0]);
Object obj = PortableRemoteObject.narrow(objRemote, theClass);
```

То есть, этот фрагмент кода получает класс из домашнего объекта и затем вызывает

`getDeclaredMethod()` с именем метода `create` для получения объекта `Method`, представляющего метод `create()`. Далее используется метод `invoke()` для выполнения метода `create()`, возвращающего экземпляр компонента EJB.

1. Класс EjbProxy

Класс `EjbProxy`, представленный в исходных кодах как EjbProxy.java, управляет этими неприятными деталями за вас.

Применять `EjbProxy` достаточно просто. Сперва вы создаёте объект `EjbProxy` — вы можете передавать параметры, такие как фабрика начального контекста и URL провайдера для создания начального контекста. После чего вы вызываете метод `getObj()` с поисковым JNDI именем компонента, чтобы получить экземпляр компонента. Помните, что если вы используете имя ссылки, когда вызываете один компонент EJB из другого, вам может понадобиться добавить строку поискового окружения к поисковому имени, такую как `java.comp/env/`.

Пример кода, который вы можете скачать, показывает, как использовать класс `EjbProxy` для создания экземпляра и последующей активизации компонента. С целью тестирования, также был реализован компонент, названный `EjbProxyExample`, имеющий один публичный метод:

```
public String hello (String name) throws RemoteException;
```

Этот метод просто принимает строку имени и возвращает строку, составленную из "Hello," и переданного имени. В примере, заместитель компонента EJB имеет JNDI имя `com.javaworldtip.ejbproxyexample.EjbProxyExample`. Предположим, вы используете WebLogic для сборки и размещения компонента из примера на локальной машине с портом 7001; для создания экземпляра `EjbProxyExample` клиентский код может быть таким:

```
EjbProxy myproxy =
        new EjbProxy ("weblogic.jndi.WLInitialContextFactory",
              "t3://localhost:7001");
EjbProxyExample obj = (EjbProxyExample)myproxy.getObj
("com.javaworldtip.ejbproxyexample.EjbProxyExample");
 // теперь получим строку "Hello, World!" и покажем её
 String strHello = obj.hello("World");
System.out.println (strHello);
```

Если вы вызываете этот компонент из другого компонента, расположенного на том же самом сервере, тогда ваш код должен выглядеть так:

```
EjbProxy myproxy = new EbjProxy ();
EjbProxyExample obj = (EjbProxyExample)myproxy.getObj
("com.javaworldtip.ejbproxyexample.EjbProxyExample ");
```

Если вы уже определили `EjbProxyExample` в секции `ejb-ref` вызывающего компонента, назвав связь `"EjbProxyExample"`, тогда вы можете использовать непосредственно имя связи в методе `getObj()`:

```
EjbProxyExample obj =
      (EjbProxyExample) myproxy.getObj (
        "java:comp/env/EjbProxyExmaple");
```

2. Как это работает

Вы можете использовать расширенный конструктор или метод `setContextProperties()` для установки любого свойства контекста, которое будет использовать `EJBProxy`:

```
public void setContextProperties (String initContextFactory,
                                  String providerUrl,
                                  String user,
                                  String password)
    {
        _prop = new Properties ();
        _prop.put (Context.INITIAL_CONTEXT_FACTORY, initContextFactory);
        _prop.put (Context.PROVIDER_URL, providerUrl);
        if (user != null)
        {
            _prop.put(Context.SECURITY_PRINCIPAL, user);
            if (password == null)
                        password = "";
            _prop.put(Context.SECURITY_CREDENTIALS, password);
        }
    }
```

Здесь все параметры являются переменными окружения, необходимыми для создания начального контекста, включая фабрику начального контекста, URL провайдера, идентификатор пользователя и его пароль. Если вы используете этот класс для

получения компонента в том же контейнере, вы можете просто передать null или проигнорировать их.

Теперь давайте рассмотрим наиболее важный метод в нашем классе посреднике, getObj():

```
public Object getObj (String beanJndiLookupName) throws EJBException
{
    try
    {
        EJBHome obHome = getHome (beanJndiLookupName);
        // получаем метод создания (create)
         Method m = obHome.getClass()
     .getDeclaredMethod("create", new Class[0]);
        // вызываем метод create
         Object obj = m.invoke (obHome, new Object[0]);
        return obj;
    }
    catch (NoSuchMethodException ne)
    {
        throw new EJBException (ne);
    }
    catch (InvocationTargetException ie)
    {
        throw new EJBException (ie);
    }
    catch (IllegalAccessException iae)
    {
        throw new EJBException (iae);
    }
}
```

В `getObj()` единственным необходимым параметром является JNDI имя компонента, которого мы ищем. Метод сначала вызывает `getHome()` для получения объекта `EJBHome`, затем вызывает метод `getDeclaredMethod()` для получения метода `create()` через ссылку на `Class` полученного `EBJHome`. Когда метод `create()` получен, вы можете вызывать его для получения экземпляра компонент. Если что-то пойдёт не так, вызов породит исключение.

Вы должны обработать некоторые возможные исключения. Например, если компонент

не содержит метода `create()`, то вызов `getDeclaredMethod()` выбросит `NoSuchMethodException`. В данном коде различные блоки перехвата исключений (`catch`) обрабатывают возможные исключения выбросом нового `EJBException`. Таким образом, код, вызывающий метод `getObj()` должен перехватывать `EJBException` и корректно обрабатывать это исключение.

Следующий метод `getHome()` реализует вызов `getObj()`:

```
public EJBHome getHome (String beanJndiLookupName) throws EJBException
{
    try
    {
          InitialContext ctx = null;

          if (_prop != null)
                    ctx = new InitialContext (_prop);
            else
                    ctx = new InitialContext ();

        Object home = ctx.lookup(beanJndiLookupName);
        EJBHome obHome = (EJBHome)
            PortableRemoteObject.narrow (home, EJBHome.class);
        return obHome;
    }
    catch (NamingException ne)
    {
        throw new EJBException (ne);
    }
}
```

Этот метод просто покрывает шаги с первого по третий для вызова компонента, как было упомянуто выше. К тому же, метод `getHome()` был сделан открытым просто для того, что бы сделать объект `home` более желанным, то есть, вы можете захотеть сохранить ссылку на объект `EJBHome` и использовать её в дальнейшем для повторного вызова методов (таких как вызов метода `create()` для создания нового сессионного компонента).

3. Расширение EjbProxy

В некоторых ситуациях, вы можете захотеть расширить класс `EjbProxy`. Например, метод `create()` компонента не может иметь каких-либо параметров. Вы можете легко расширить класс для обработки более общей ситуации изменённым методом `getObj()`:

```
public Object getObj (String beanJNDILookupName, Class[] classes, Object[]
objects) throws EJBException;
```

Внутри метода, дополнительные параметры применяются при вызове методов `getDeclaredMethod()` и `invoke()`:

```
Method m = obHome.getClass().getDeclaredMethod("create", classes);
Object obj = m.invoke (obHome, objects);
```

Используя отражение, вы можете также расширить этот класс для вызова произвольных методов компонент:

```
public Object executeMethod (String methodName, Class[] classes, Object[]
objects) throws EJBException;
```

Здесь `methodName` — имя вызываемого метода. Вам может потребоваться кэширование объектов компонент в вашем классе после получения объекта `EJBHome`, тогда используйте подход, аналогичный применённому в `getObj()`, пересмотренном для вызова метода.

4. Упростите ваше ежедневное программирование

Используя механизм отражения Java, вы можете использовать общий и гибкий класс компонентного посредника, который позволит вам вызывать компоненты из удалённых клиентов или из других компонент. Как вы видели, вы можете использовать класс `EjbProxy` или расшить его для облегчения вашего ежедневного программирования, связанного с созданием экземпляров и вызовом методов компонент EJB.

5. Об авторе

Горсен Хуанг (Gorsen Huang) является ведущим программистом в Wysdom Inc. Он работает в области архитектуры, дизайна и программирования программного обеспечения более 15 лет.

6. Ресурсы

- Получите исходные коды для данной подсказки: http://www.javaworld.com/javaworld/javatips/javatip118/EJBProxyTip.zip
- Информация о корпоративных компонентах (EJB) от компании Sun: http://java.sun.com/products/ejb
- Документация по прикладному интерфейсу(API) отражения Java: http://java.sun.com/products/ jdk/1.1/docs/guide/reflection/
- Также смотрите "Создание EJB из любого класса, используя отражение" Tony Loton (*JavaWorld*, Декабрь 2000): http://www.javaworld.com /jw-12-2000/jw-1215-anyclass.html
- Больше статей о **EJB**, смотрите в тематическом указателе *JavaWorld*: http://www.javaworld.com/ channel_content/jw-ejbs-index.shtml
- Больше статей о **JNDI**, смотрите в тематическом указателе: http://www.javaworld.com/ channel_content/jw-jndi-index.shtml
- Смотрите все предыдущие **Подсказки** и добавьте свои собственные: http://www.javaworld.com/ javatips/jw-javatips.index.html
- Изучайте Java в колонке **Java 101**: http://www.javaworld.com/javaworld/ topicalindex/jw-ti-java101.html
- Эксперты Java ответят на ваши вопросы в колонке **Java Q&A**: http://www.javaworld.com/javaworld/javaqa/javaqa-index.html
- Присоединяйтесь к бесплатной ежемесячной рассыке *JavaWorld*s: http://www.idg.net/jw-subscribe
- Вы можете найти множество статей, связанных с информационными технологиями на дружественном нам сайте IDG.net

View the original article at: http://www.javaworld.com/javatips/jw-javatip118.html