作者 正文

fangang

等级:

文章: 185

积分: 416

来自: 北京

我现在离线

发表时间：2007-04-19

< > 猎头职位：上海：上海，北京：招聘java开发工程师

相关文章:

- 再析在spring框架中解决多数据源的问题
- 如何在spring框架中解决多数据源的问题（下）
- 动态切换多数据源

推荐圈子：JBPM @net

更多相关推荐

在我们的项目中遇到这样一个问题：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。我们以往在spring和hibernate框架中总是配置一个数据源，因而sessionFactory的dataSource属性总是指向这个数据源并且恒定不变，所有DAO在使用sessionFactory的时候都是通过这个数据源访问数据库。但是现在，由于项目的需要，我们的DAO在访问sessionFactory的时候都不得不在多个数据源中不断切换，问题就出现了：如何让sessionFactory在执行数据持久化的时候，根据客户的需求能够动态切换不同的数据源？我们能不能在spring的框架下通过少量修改得到解决？是否有什么设计模式可以利用呢？

问题的分析

我首先想到在spring的applicationContext中配置所有的dataSource。这些dataSource可能是各种不同类型的，比如不同的数据库：Oracle、SQL Server、MySQL等，也可能是不同的数据源：比如apache 提供的org.apache.commons.dbcp.BasicDataSource、spring提供的org.springframework.jndi.JndiObjectFactoryBean等。然后sessionFactory根据客户的每次请求，将dataSource属性设置成不同的数据源，以到达切换数据源的目的。

但是，我很快发现一个问题：当多用户同时并发访问数据库的时候会出现资源争用的问题。这都是“单例模式”惹的祸。众所周知，我们在使用spring框架的时候，在beanFactory中注册的bean基本上都是采用单例模式，即spring在启动的时候，这些bean就装载到内存中，并且每个bean在整个项目中只存在一个对象。正因为只存在一个对象，对象的所有属性，更准确说是实例变量，表现得就如同是个静态变量（实际上“静态”与“单例”往往是非常相似的两个东西，我们常常用“静态”来实现“单例”）。拿我们的问题来说，sessionFactory在整个项目中只有一个对象，它的实例变量dataSource也就只有一个，就如同一个静态变量一般。如果不同的用户都不断地去修改dataSource的值，必然会出现多用户争用一个变量的问题，对系统产生隐患。

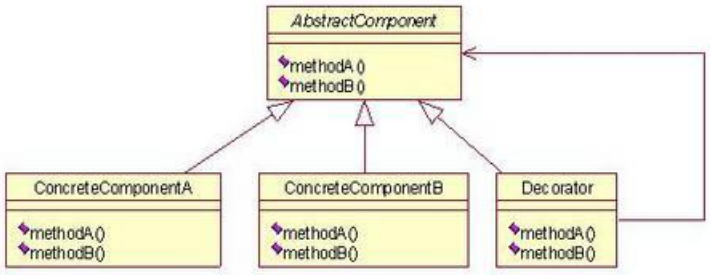
通过以上的分析，解决多数据源访问问题的关键，就集中在sessionFactory在执行数据持久化的时候，能够通过某段代码去根据客户的需要动态切换数据源，并解决资源争用的问题。

问题的解决

(一) 采用Decorator设计模式

要解决这个问题，我的思路锁定在了这个dataSource上了。如果sessionFactory指向的dataSource可以根据客户的需求去连接客户所需要的真正的数据源，即提供动态切换数据源的功能，那么问题就解决了。那么我们怎么做呢？去修改那些我们要使用的dataSource源码吗？这显然不是一个好的方案，我们希望我们的修改与原dataSource代码是分离的。根据以上的分析，使用GoF设计模式中的Decorator模式（装饰者模式）应当是我们可以选择的最佳方案。

什么是“Decorator模式”？简单点儿说就是当我们需要修改原有的功能，但我们又不愿直接去修改原有的代码时，设计一个Decorator套在原有代码外面。当我们使用Decorator的时候与原类完全一样，当Decorator的某些功能却已经修改为了我们需要修改的功能。Decorator模式的结构如图。

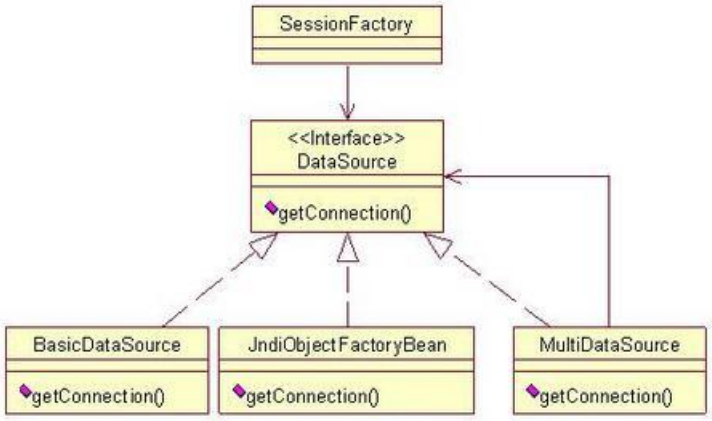


我们本来需要修改图中所有具体的Component类的一些功能，但却并不是去直接修改它们的代码，而是在它们的外面增加一个Decorator。Decorator与

具体的Component类都是继承的AbstractComponent，因此它长得和具体的Component类一样，也就是说我们在使用Decorator的时候就如同在使用ConcreteComponentA或者ConcreteComponentB一样，甚至那些使用ConcreteComponentA或者ConcreteComponentB的客户程序都不知道它们用的类已经改为了Decorator，但是Decorator已经对具体的Component类的部分方法进行了修改，执行这些方法的结果已经不同了。

(二) 设计MultiDataSource类

现在回到我们的问题，我们需要对dataSource的功能进行变更，但又不希望修改dataSource中的任何代码。我这里指的dataSource是所有实现javax.sql.DataSource接口的类，我们常用的包括apache 提供的org.apache.commons.dbcp.BasicDataSource、spring提供的org.springframework.jndi.JndiObjectFactoryBean等，这些类我们不可能修改它们本身，更不可能对它们一个个地修改以实现动态分配数据源的功能，同时，我们又希望使用dataSource的sessionFactory根本就感觉不到这样的变化。Decorator模式就正是解决这个问题的设计模式。
首先写一个Decorator类，我取名叫MultiDataSource，通过它来动态切换数据源。同时在配置文件中将sessionFactory的dataSource属性由原来的某个具体的dataSource改为MultiDataSource。如图：



对比原Decorator模式，AbstractComponent是一个抽象类，但在这里我们可以将这个抽象类用接口来代替，即DataSource接口，而ConcreteComponent就是那些DataSource的实现类，如BasicDataSource、JndiObjectFactoryBean等。MultiDataSource封装了具体的dataSource，并实现了数据源动态切换：

java 代码

```java
public class MultiDataSource implements DataSource {
    private DataSource dataSource = null;
public MultiDataSource(DataSource dataSource){
        this.dataSource = dataSource;
    }
    /* (non-Javadoc)
     * @see javax.sql.DataSource#getConnection()
     */
    public Connection getConnection() throws SQLException {
        return getDataSource().getConnection();
    }
    //其它DataSource接口应当实现的方法

    public DataSource getDataSource(){
        return this.dataSource;
        }
    }
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
}
```

客户在发出请求的时候，将dataSourceName放到request中，然后把request中的数据源名通过调用new `MultiDataSource(dataSource)`时可以告诉`MultiDataSource`客户需要的数据源，就可以实现动态切换数据源了。但细心的朋友会发现这在单例的情况下就是问题的，因为`MultiDataSource`在系统中只有一个对象，它的实例变量`dataSource`也只有一个，就如同一个静态变量一般。正因为如此，单例模式让许多设计模式都不得不需要更改，这将在我的《“单例”更改了我们的设计模式》中详细讨论。那么，我们在单例模式下如何设计呢？

(三) 单例模式下的MultiDataSource

在单例模式下，由于我们在每次调用MultiDataSource的方法的时候，dataSource都可能是不同的，所以我们不能将dataSource放在实例变量dataSource中，最简单的方式就是在方法getDataSource()中增加参数，告诉MultiDataSource我到底调用的是哪个dataSource：

java 代码

```java
public DataSource getDataSource(String dataSourceName){
        log.debug("dataSourceName:"+dataSourceName);
        try{
            if(dataSourceName==null||dataSourceName.equals("")){
                return this.dataSource;
```

```
            }
            return (DataSource)this.applicationContext.getBean(dataSourceName);
        }catch(NoSuchBeanDefinitionException ex){
            throw new DaoException("There is not the dataSource
        }
    }
```

值得一提的是，我需要的数据源已经都在spring的配置文件中注册，`dataSourceName`就是其对应的`id`。

xml 代码

```
<bean id="dataSource1"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName">
        <value>oracle.jdbc.driver.OracleDrivervalue>
    property>
    ......
bean>
<bean id="dataSource2"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName">
        <value>oracle.jdbc.driver.OracleDrivervalue>
    property>
    ......
bean>
```

为了得到spring的ApplicationContext，MultiDataSource类必须实现接口`org.springframework.context.ApplicationContextAware`，并且实现方法：

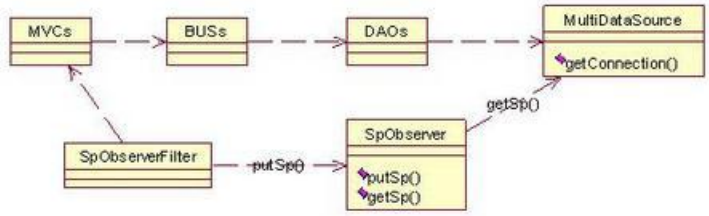
java 代码

```
private ApplicationContext applicationContext = null;
public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        this.applicationContext = applicationContext;
    }
```

如此这样，我就可以通过`this.applicationContext.getBean(dataSourceName)`得到`dataSource`了。

(四) 通过线程传递dataSourceName

查看以上设计，MultiDataSource依然无法运行，因为用户在发出请求时，他需要连接什么数据库，其数据源名是放在request中的，要将request中的数据源名传给MultiDataSource，需要经过BUS和DAO，也就是说为了把数据源名传给MultiDataSource，BUS和DAO的所有方法都要增加dataSourceName的参数，这是我们不愿看到的。写一个类，通过线程的方式跳过BUS和DAO直接传递给MultiDataSource是一个不错的设计：



java 代码

```
public class SpObserver {
    private static ThreadLocal local = new ThreadLocal();
    public static void putSp(String sp) {
        local.set(sp);
    }
    public static String getSp() {
        return (String)local.get();
    }
}
```

做一个filter，每次客户发出请求的时候就调用`SpObserver.petSp(dataSourceName)`，将request中的dataSourceName传递给`SpObserver`对象。最后修改MultiDataSource的方法getDataSource()：

java 代码

```
public DataSource getDataSource(){
        String sp = SpObserver.getSp();
        return getDataSource(sp);
    }
```

完整的MultiDataSource代码在附件中。

(五) 动态添加数据源

通过以上方案，我们解决了动态分配数据源的问题，但你可能提出疑问：方案中的数据源都是配置在spring的ApplicationContext中，如果我在程序运行过程中动态添加数据源怎么办？这确实是一个问题，而且在我们的项目中也确实遇到。spring的ApplicationContext是在项目启动的时候加载的。加载以后，我们如何动态地加载新的bean到ApplicationContext中呢？我想到如果用spring自己的方法解决这个问题就好了。所幸的是，在查

看spring的源代码后，我找到了这样的代码，编写了DynamicLoadBean类，只要调用loadBean()方法，就可以将某个或某几个配置文件中的bean加载到ApplicationContext中（见附件）。不通过配置文件直接加载对象，在spring的源码中也有，感兴趣的朋友可以自己研究。

(六) 在**spring**中配置

在完成了所有这些设计以后，我最后再唠叨一句。我们应当在spring中做如下配置：

xml 代码

```xml
<bean id="dynamicLoadBean" class="com.htxx.service.dao.DynamicLoadBean">bean>
<bean id="dataSource" class="com.htxx.service.dao.MultiDataSource">
        <property name="dataSource">
            <ref bean="dataSource1" />
        property>
    bean>
    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

        <property name="dataSource">
            <ref bean="dataSource" />
        property>
        ......
    bean>
```

其中dataSource属性实际上更准确地说应当是defaultDataSource，即spring启动时以及在客户没有指定数据源时应当指定的默认数据源。

该方案的优势

以上方案与其它方案相比，它有哪些优势呢？

首先，这个方案完全是在spring的框架下解决的，数据源依然配置在spring的配置文件中，sessionFactory依然去配置它的dataSource属性，它甚至都不知道dataSource的改变。唯一不同的是在真正的dataSource与sessionFactory之间增加了一个MultiDataSource。

其次，实现简单，易于维护。这个方案虽然我说了这么多东西，其实都是分析，真正需要我们写的代码就只有MultiDataSource、SpObserver两个类。MultiDataSource类真正要写的只有getDataSource()和getDataSource(sp)两个方法，而SpObserver类更简单了。实现越简单，出错的可能就越小，维护性就越高。

最后，这个方案可以使单数据源与多数据源兼容。这个方案完全不影响BUS和DAO的编写。如果我们的项目在开始之初是单数据源的情况下开发，随着项目的进行，需要变更为多数据源，则只需要修改spring配置，并少量修改MVC层以便在请求中写入需要的数据源名，变更就完成了。如果我们的项目希望改回单数据源，则只需要简单修改配置文件。这样，为我们的项目将增加更多的弹性。

特别说明：实例中的**DynamicLoadBean**在**web**环境下运行会出错，需要将类中**AbstractApplicationContext**改为**org.springframework.context.ConfigurableApplicationContext**。

相关博客：再析在spring框架中解决多数据源的问题

---

example.rar (32.1 KB)

描述: 源码及示例

下载次数: 2481

推荐链接

返回顶楼

jianfeng008cn

等级:

文章: 680

积分: 713

来自: 湖州

发表时间：2007-04-20

我觉得你配置多个sessionfactory bean从他向下都配置一套不就OK了 不知道你为什么要保证只有一个sessionfacty 不就是多加几个bean嘛

我现在离线

返回顶楼 回帖地址 0 2 请登录后投票

jianfeng008cn

等级:

文章: 680

积分: 713

来自: 湖州

我现在离线

发表时间：2007-04-20

意思意思吧 下面这样的东西分别搞几套就好了

Java代码

```
<bean id="dmDataSource">
<bean id="dmSessionFactory">
<bean id="dmTransactionInterceptor">
<bean id="dmTransactionManager">

<bean
                class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
                <property name="beanNames">
                        <value>*Service</value>
                </property>
                <property name="interceptorNames">
                        <list>
                                <value>dmTransactionInterceptor</value>
                        </list>
                </property>
</bean>

<bean
                class="org.springframework.transaction.interceptor.TransactionAttributeSourceAdvisor">
                <property name="transactionInterceptor"
                        ref="dmTransactionInterceptor" />
        </bean>

        <bean id="dmPersistence"
                class="">
                <property name="sessionFactory" ref="dmSessionFactory" />
</bean>
```

返回顶楼 回帖地址 0 2 请登录后投票

fangang

等级:

文章: 185

积分: 416

来自: 北京

我现在离线

发表时间：2007-04-23

感谢jianfeng008cn的评论，但是你也许还没有明白我要解决的问题。在spring的配置中，即使搞几套dataSource和sessionFactory，每个DAO使用哪个sessionFactory都是在配置中定死了的，比如：

Xml代码

```
<bean id="departmentDao" class="com.htxx.service.dao.departmentDao">
        <property name="sessionFactory"><ref bean="sessionFactory"/></property>
    </bean>
```

每个sessionFactory使用哪个dataSource也是在配置中定死了的

Xml代码

```
<bean id="sessionFactory"
                class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
                <property name="dataSource">
                        <ref bean="dataSource" />
                </property>
        </bean>
```

现在我希望在项目运行时动态分配每个DAO执行持久化使用哪个数据库，这样的需求并不是在spring中多搞几套dataSource和sessionFactory就能解决的。

返回顶楼 回帖地址 0 0 请登录后投票

byk

等级: 初级会员

文章: 31

积分: 66

来自: ...

我现在离线

发表时间: 2007-04-24

fangang 写道

> 感谢jianfeng008cn的评论，但是你也许还没有明白我要解决的问题。在spring的配置中，即使搞几套dataSource和sessionFactory，每个DAO使用哪个sessionFactory都是在配置中定死了的，比如：
>
> **Xml**代码
>
> ```
> 1.  <bean id="departmentDao" class="com.htxx.service.dao.departmentDao">
> 2.          <property name="sessionFactory"><ref bean="sessionFactory"/></property>
> 3.      </bean>
> ```
>
> 每个sessionFactory使用哪个dataSource也是在配置中定死了的
>
> **Xml**代码
>
> ```
> 1.  <bean id="sessionFactory"
> 2.                  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
> 3.                  <property name="dataSource">
> 4.                          <ref bean="dataSource" />
> 5.                  </property>
> 6.          </bean>
> ```
>
> 现在我希望在项目运行时动态分配每个DAO执行持久化使用哪个数据库，这样的需求并不是在spring中多搞几套dataSource和sessionFactory就能解决的。

根据数据源配多个同样的sessionFactory和dao,把dao注到一个map里,动态调用dao不就更简单???

返回顶楼 回帖地址 0 0 请登录后投票

fangang

等级: ☆☆☆☆

文章: 185

积分: 416

来自: 北京

我现在离线

发表时间: 2007-04-25

byk 写道

> 根据数据源配多个同样的sessionFactory和dao,把dao注到一个map里,动态调用dao不就更简单???

非常高兴byk跟我讨论这个问题。事实上，我将多个数据源配置到applicationContext其本身就是一个map。查看源码就可以发现spring的beanFactory就是使用map来管理配置在spring中的所有bean。我们这个问题的关键是“动态调用”，也就是dao如何去动态调用dataSource。解决了动态调用的问题，我们这个问题就解决了。而我通过创建MultiDataSource既解决了动态调用的问题，又与原系统兼容。

返回顶楼 回帖地址 0 0 请登录后投票

klyuan

等级: ☆☆☆☆☆

文章: 201

积分: 502

来自: 深圳

发表时间: 2007-04-25

把完整的贴出来看看吧

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

fangang

等级:

文章: 185

积分: 416

来自: 北京

发表时间：2007-04-26

klyuan 写道

> 把完整的贴出来看看吧

我已经修改了这篇文章，将原来的上、下两篇精简为一篇。完整的代码以及示例也在附件中了

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

huangxx

等级: 初级会员

文章: 18

积分: 30

来自: ...

发表时间：2007-04-26

这样岂不是不能用HIBERNATETEMPLATE？

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

rainlife

等级:

文章: 683

积分: 1034

来自: 我也来自火星?

发表时间：2007-04-26

怎么图片看不出来？那些空白的方框是图片吧？

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

北京: 益网诚聘高级软件开发工程师

□□: □□□□□□□□□□□□ java□□□□□

□□: □□□□□JAVA□□□□□

□□: The Netcircle□□JAVA□□□

□□: □□□□□□□□□□□□□□□□□ (□□8k)