**Spring配置多数据源实现数据库的读写分离**

## 实现读写分离，目前常用的两种方式：

1. 第一种方式是我们最常用的方式，就是定义2个数据库连接，一个是MasterDataSource,另一个是SlaveDataSource。更新数据时我们读取MasterDataSource，查询数据时我们读取SlaveDataSource。
2. 第二种方式动态数据源切换，就是在程序运行时，把数据源动态织入到程序中，从而选择读取主库还是从库。主要使用的技术是：annotation，Spring AOP ，反射。下面会详细的介绍实现方式。

## 介绍

在介绍实现方式之前，我们先准备一些必要的知识，spring 的****AbstractRoutingDataSource**** 类

　　   AbstractRoutingDataSource这个类 是spring2.0以后增加的，我们先来看下AbstractRoutingDataSource的定义：

**Java代码  [IMG_256](http://uule.iteye.com/blog/javascript:void())**

1. **public** **abstract** **class** AbstractRoutingDataSource **extends** AbstractDataSource **implements** InitializingBean  {}

**Java代码  [IMG_257](http://uule.iteye.com/blog/javascript:void())**

1. **public** **abstract** **class** AbstractRoutingDataSource **extends** AbstractDataSource **implements** InitializingBean {
3. **private** Map<Object, Object> targetDataSources;
5. **private** Object defaultTargetDataSource;
7. **private** DataSourceLookup dataSourceLookup = **new** JndiDataSourceLookup();
9. **private** Map<Object, DataSource> resolvedDataSources;
11. **private** DataSource resolvedDefaultDataSource;

　　  AbstractRoutingDataSource继承了AbstractDataSource ，而AbstractDataSource 又是DataSource 的子类。

DataSource   是javax.sql 的数据源接口，定义如下：

**Java代码  [IMG_258](http://uule.iteye.com/blog/javascript:void())**

1. **public** **interface** DataSource  **extends** CommonDataSource,Wrapper {
3. Connection getConnection() **throws** SQLException;
5. Connection getConnection(String username, String password)
6. **throws** SQLException;
7. }

 DataSource 接口定义了2个方法，都是获取数据库连接。我们在看下AbstractRoutingDataSource 如何实现了DataSource接口：

**Java代码  [IMG_259](http://uule.iteye.com/blog/javascript:void())**

1. **public** Connection getConnection() **throws** SQLException {
2. **return** determineTargetDataSource().getConnection();
3. }
5. **public** Connection getConnection(String username, String password) **throws** SQLException {
6. **return** determineTargetDataSource().getConnection(username, password);
7. }

 很显然就是调用自己的determineTargetDataSource()  方法获取到connection。determineTargetDataSource方法定义如下：

**Java代码  [IMG_260](http://uule.iteye.com/blog/javascript:void())**

1. **protected** DataSource determineTargetDataSource() {
2. Assert.notNull(**this**.resolvedDataSources, "DataSource router not initialized");
3. Object lookupKey = determineCurrentLookupKey();
4. DataSource dataSource = **this**.resolvedDataSources.get(lookupKey);
5. **if** (dataSource == **null** && (**this**.lenientFallback || lookupKey == **null**)) {
6. dataSource = **this**.resolvedDefaultDataSource;
7. }
8. **if** (dataSource == **null**) {
9. **throw** **new** IllegalStateException("Cannot determine target DataSource for lookup key [" + lookupKey + "]");
10. }
11. **return** dataSource;
12. }

我们最关心的还是下面2句话：

　　Object lookupKey = determineCurrentLookupKey();

    DataSource dataSource = this.resolvedDataSources.get(lookupKey);

    determineCurrentLookupKey方法返回lookupKey,resolvedDataSources方法就是根据lookupKey从Map中获得数据源。resolvedDataSources 和determineCurrentLookupKey定义如下：

　　private Map<Object, DataSource> resolvedDataSources;

　　protected abstract Object determineCurrentLookupKey()

　　看到以上定义，我们是不是有点思路了，resolvedDataSources是Map类型，我们可以把MasterDataSource和SlaveDataSource存到Map中，如下：

　　　　key　　　　　　　　value

　　　　master　　         MasterDataSource

　　　　slave              SlaveDataSource

　　我们在写一个类DynamicDataSource  继承AbstractRoutingDataSource，实现其determineCurrentLookupKey() 方法，该方法返回Map的key，master或slave。

　　好了，说了这么多，有点烦了，下面我们看下怎么实现。

 　上面已经提到了我们要使用的技术，我们先看下annotation的定义：

**Java代码  [IMG_261](http://uule.iteye.com/blog/javascript:void())**

1. @Retention(RetentionPolicy.RUNTIME)
2. @Target(ElementType.METHOD)
3. **public** **@interface** DataSource {
4. String value();
5. }

    我们还需要实现spring的抽象类AbstractRoutingDataSource，就是实现determineCurrentLookupKey方法：

**Java代码  [IMG_262](http://uule.iteye.com/blog/javascript:void())**

1. **public** **class** DynamicDataSource **extends** AbstractRoutingDataSource {
3. @Override
4. **protected** Object determineCurrentLookupKey() {
5. // TODO Auto-generated method stub
6. **return** DynamicDataSourceHolder.getDataSouce();
7. }
9. }

12. **public** **class** DynamicDataSourceHolder {
13. **public** **static** **final** ThreadLocal<String> holder = **new** ThreadLocal<String>();
15. **public** **static** **void** putDataSource(String name) {
16. holder.set(name);
17. }
19. **public** **static** String getDataSouce() {
20. **return** holder.get();
21. }
22. }

    从DynamicDataSource 的定义看出，他返回的是DynamicDataSourceHolder.getDataSouce()值，我们需要在程序运行时调用DynamicDataSourceHolder.putDataSource()方法，对其赋值。下面是我们实现的核心部分，也就是AOP部分，DataSourceAspect定义如下:

**Java代码  [IMG_263](http://uule.iteye.com/blog/javascript:void())**

1. **public** **class** DataSourceAspect {
3. **public** **void** before(JoinPoint point)
4. {
5. Object target = point.getTarget();
6. String method = point.getSignature().getName();
8. Class<?>[] classz = target.getClass().getInterfaces();
10. Class<?>[] parameterTypes = ((MethodSignature) point.getSignature())
11. .getMethod().getParameterTypes();
12. **try** {
13. Method m = classz[0].getMethod(method, parameterTypes);
14. **if** (m != **null** && m.isAnnotationPresent(DataSource.**class**)) {
15. DataSource data = m
16. .getAnnotation(DataSource.**class**);
17. DynamicDataSourceHolder.putDataSource(data.value());
18. System.out.println(data.value());
19. }
21. } **catch** (Exception e) {
22. // TODO: handle exception
23. }
24. }
25. }

    为了方便测试，我定义了2个数据库，shop模拟Master库，test模拟Slave库，shop和test的表结构一致，但数据不同，数据库配置如下：

**Xml代码  [IMG_264](http://uule.iteye.com/blog/javascript:void())**

1. **<bean** id="masterdataSource"
2. class="org.springframework.jdbc.datasource.DriverManagerDataSource"**>**
3. **<property** name="driverClassName" value="com.mysql.jdbc.Driver" **/>**
4. **<property** name="url" value="jdbc:mysql://127.0.0.1:3306/shop" **/>**
5. **<property** name="username" value="root" **/>**
6. **<property** name="password" value="yangyanping0615" **/>**
7. **</bean>**
9. **<bean** id="slavedataSource"
10. class="org.springframework.jdbc.datasource.DriverManagerDataSource"**>**
11. **<property** name="driverClassName" value="com.mysql.jdbc.Driver" **/>**
12. **<property** name="url" value="jdbc:mysql://127.0.0.1:3306/test" **/>**
13. **<property** name="username" value="root" **/>**
14. **<property** name="password" value="yangyanping0615" **/>**
15. **</bean>**
17. **<beans:bean** id="dataSource" class="com.air.shop.common.db.DynamicDataSource"**>**
18. **<property** name="targetDataSources"**>**
19. **<map** key-type="java.lang.String"**>**
20. <!-- write -->
21. **<entry** key="master" value-ref="masterdataSource"**/>**
22. <!-- read -->
23. **<entry** key="slave" value-ref="slavedataSource"**/>**
24. **</map>**
26. **</property>**
27. **<property** name="defaultTargetDataSource" ref="masterdataSource"**/>**
28. **</beans:bean>**
30. **<bean** id="transactionManager"
31. class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
32. **<property** name="dataSource" ref="dataSource" **/>**
33. **</bean>**

36. <!-- 配置SqlSessionFactoryBean -->
37. **<bean** id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>**
38. **<property** name="dataSource" ref="dataSource" **/>**
39. **<property** name="configLocation" value="classpath:config/mybatis-config.xml" **/>**
40. **</bean>**

　　在spring的配置中增加aop配置

**Xml代码  [IMG_265](http://uule.iteye.com/blog/javascript:void())**

1. <!-- 配置数据库注解aop -->
2. **<aop:aspectj-autoproxy></aop:aspectj-autoproxy>**
3. **<beans:bean** id="manyDataSourceAspect" class="com.air.shop.proxy.DataSourceAspect" **/>**
4. **<aop:config>**
5. **<aop:aspect** id="c" ref="manyDataSourceAspect"**>**
6. **<aop:pointcut** id="tx" expression="execution(\* com.air.shop.mapper.\*.\*(..))"**/>**
7. **<aop:before** pointcut-ref="tx" method="before"**/>**
8. **</aop:aspect>**
9. **</aop:config>**
10. <!-- 配置数据库注解aop -->

　　 下面是MyBatis的UserMapper的定义，为了方便测试，登录读取的是Master库，用户列表读取Slave库：

**Java代码  [IMG_266](http://uule.iteye.com/blog/javascript:void())**

1. **public** **interface** UserMapper {
2. @DataSource("master")
3. **public** **void** add(User user);
5. @DataSource("master")
6. **public** **void** update(User user);
8. @DataSource("master")
9. **public** **void** delete(**int** id);
11. @DataSource("slave")
12. **public** User loadbyid(**int** id);
14. @DataSource("master")
15. **public** User loadbyname(String name);
17. @DataSource("slave")
18. **public** List<User> list();
19. }