**MyBatis整合Spring**

开发环境：

System：Windows

WebBrowser：IE6+、Firefox3+

JavaEE Server：tomcat5.0.2.8、tomcat6

IDE：eclipse、MyEclipse 8

Database：MySQL

开发依赖库：

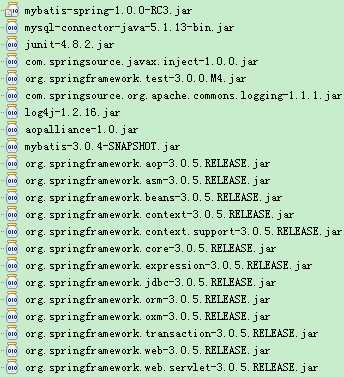
JavaEE5、Spring 3.0.5、Mybatis 3.0.4、myBatis-spring-1.0、junit4.8.2

Email：hoojo\_@126.com

Blog：<http://blog.csdn.net/IBM_hoojo>

<http://hoojo.cnblogs.com/>

1. 首先新建一个WebProject 命名为MyBatisForSpring，新建项目时，使用JavaEE5的lib库。然后手动添加需要的jar包，所需jar包如下：



1. 添加spring的监听及springMVC的核心Servlet，web.xml内容，内容如下：

|  |
| --- |
| <!-- 加载Spring容器配置 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 设置Spring容器加载配置文件路径 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:applicationContext-\*.xml</param-value>  </context-param>  <!-- 配置Spring核心控制器 -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/dispatcher.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  <!-- 解决工程编码过滤器 -->  <filter>  <filter-name>characterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>characterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

1. 在WEB-INF目录中添加dispatcher.xml，内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>  <!-- 注解探测器 -->  <context:component-scan base-package=*"com.hoo"*/>    <!-- annotation默认的方法映射适配器 -->  <bean id=*"handlerMapping"* class=*"org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"* />  <bean id=*"handlerAdapter"* class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"* />  </beans> |

1. 在src目录下添加applicationContext-common.xml，内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.0.xsd "*>    <!-- 配置DataSource数据源 -->  <bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"url"* value=*"jdbc:mysql://10.0.0.131:3306/ash2"*/>  <property name=*"username"* value=*"dev"*/>  <property name=*"password"* value=*"dev"*/>  </bean>    <!-- 配置SqlSessionFactoryBean -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  <property name=*"configLocation"* value=*"classpath:mybatis.xml"*/>  <!-- mapper和resultmap配置路径 -->  <property name=*"mapperLocations"*>  <list>  <!-- 表示在com.hoo.resultmap包或以下所有目录中，以-resultmap.xml结尾所有文件 -->  <value>classpath:com/hoo/resultmap/\*\*/\*-resultmap.xml</value>  <value>classpath:com/hoo/entity/\*-resultmap.xml</value>  <value>classpath:com/hoo/mapper/\*\*/\*-mapper.xml</value>  </list>  </property>  </bean>    <!-- 单独配置一个Mapper； 这种模式就是得给每个mapper接口配置一个bean -->  <!--  <bean id="accountMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">  <property name="mapperInterface" value="com.hoo.mapper.AccountMapper" />  <property name="sqlSessionFactory" ref="sqlSessionFactory" />  </bean>    <bean id="companyMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">  <property name="mapperInterface" value="com.hoo.mapper.CompanyMapper" />  <property name="sqlSessionFactory" ref="sqlSessionFactory" />  </bean>  -->    <!-- 通过扫描的模式，扫描目录在com/hoo/mapper目录下，所有的mapper都继承SqlMapper接口的接口， 这样一个bean就可以了 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.hoo.mapper"*/>  <property name=*"markerInterface"* value=*"com.hoo.mapper.SqlMapper"*/>  </bean>  </beans> |

上面的配置最先配置的是DataSource，这里采用的是jdbc的DataSource；

然后是SqlSessionFactoryBean，这个配置比较关键。SqlSessionFactoryBean需要注入DataSource数据源，其次还要设置configLocation也就是mybatis的xml配置文件路径，完成一些关于mybatis的配置，如settings、mappers、plugin等；

如果使用mapperCannerConfigurer模式，需要设置扫描根路径也就是你的mybatis的mapper接口所在包路径；凡是markerInterface这个接口的子接口都参与到这个扫描，也就是说所有的mapper接口继承这个SqlMapper。

如果你不使用自己的transaction事务，就使用MapperScannerConfigurer来完成SqlSession的打开、关闭和事务的回滚操作。在此期间，出现数据库操作的如何异常都会被转换成DataAccessException，这个异常是一个抽象的类，继承RuntimeException；

1. SqlMapper内容如下：

|  |
| --- |
| **package** com.hoo.mapper;  /\*\*  \* <b>function:</b>所有的Mapper继承这个接口  \* **@author** hoojo  \* **@createDate** 2011-4-12 下午04:00:31  \* **@file** SqlMapper.java  \* **@package** com.hoo.mapper  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** SqlMapper {  } |

1. 实体类和ResultMap.xml

|  |
| --- |
| **package** com.hoo.entity;  **import** java.io.Serializable;  **import** javax.persistence.Entity;  @Entity  **public** **class** Account **implements** Serializable {  **private** **static** **final** **long** *serialVersionUID* = -7970848646314840509L;  **private** Integer accountId;  **private** Integer status;  **private** String username;  **private** String password;  **private** String salt;  **private** String email;  **private** Integer roleId;    //getter、setter  @Override  **public** String toString() {  **return** **this**.accountId + "#" + **this**.status + "#" + **this**.username + "#" +  **this**.password + "#" + **this**.email + "#" + **this**.salt + "#" + **this**.roleId;  }  } |

account-resultmap.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"accountMap"*>  <resultMap type=*"com.hoo.entity.Account"* id=*"accountResultMap"*>  <id property=*"accountId"* column=*"account\_id"*/>  <result property=*"username"* column=*"username"*/>  <result property=*"password"* column=*"password"*/>  <result property=*"status"* column=*"status"*/>  </resultMap>  </mapper> |

1. 在src目录中添加applicationContext-beans.xml内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>  <!-- 注解探测器 ， 在JUnit测试的时候需要-->  <context:component-scan base-package=*"com.hoo"*/>    </beans> |

这里配置bean对象，一些不能用annotation注解的对象就可以配置在这里

1. 在src目录中添加mybatis.xml，内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 别名 -->  <typeAliases>  <typeAlias type=*"com.hoo.entity.Account"* alias=*"account"*/>  </typeAliases>  </configuration> |

在这个文件放置一些全局性的配置，如handler、objectFactory、plugin、以及mappers的映射路径（由于在applicationContext-common中的SqlSessionFactoryBean有配置mapper的location，这里就不需要配置）等

1. AccountMapper接口，内容如下：

|  |
| --- |
| **package** com.hoo.mapper;  **import** java.util.List;  **import** org.apache.ibatis.annotations.Select;  **import** com.hoo.entity.Account;  /\*\*  \* <b>function:</b>继承SqlMapper，MyBatis数据操作接口；此接口无需实现类  \* **@author** hoojo  \* **@createDate** 2010-12-21 下午05:21:20  \* **@file** AccountMapper.java  \* **@package** com.hoo.mapper  \* **@project** MyBatis  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** AccountMapper **extends** SqlMapper {    **public** List<Account> getAllAccount();    **public** Account getAccount();    **public** Account getAccountById(String id);    **public** Account getAccountByNames(String spring);    @Select("select \* from account where username = #{name}")  **public** Account getAccountByName(String name);    **public** **void** addAccount(Account account);    **public** **void** editAccount(Account account);    **public** **void** removeAccount(**int** id);  } |

这个接口我们不需要实现，由mybatis帮助我们实现，我们通过mapper文件配置sql语句即可完成接口的实现。然后这个接口需要继承SqlMapper接口，不然在其他地方就不能从Spring容器中拿到这个mapper接口，也就是说当我们注入这个接口的时候将会失败。

当然，你不继承这个接口也可以。那就是你需要给买个mapper配置一个bean。配置方法如下：

|  |
| --- |
| <bean id=*"accountMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>  <property name=*"mapperInterface"* value=*"com.hoo.mapper.AccountMapper"* />  <property name=*"sqlSessionFactory"* ref=*"sqlSessionFactory"* />  </bean> |

这里的MapperFactoryBean可以帮助我们完成Session的打开、关闭等操作

1. 在com.hoo.mapper也就是在AccountMapper接口的同一个包下，添加account-mapper.xml，内容如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- namespace和定义的Mapper接口对应，并实现其中的方法 -->  <mapper namespace=*"com.hoo.mapper.AccountMapper"*>  <!-- id和mapper接口中的方法名对应，resultType使用mybatis.xml中的别名 -->  <select id=*"getAccount"* resultType=*"account"*>  <![CDATA[  select \* from account limit 1  ]]>  </select>    <select id=*"getAllAccount"* resultType=*"list"* resultMap=*"accountResultMap"*>  <![CDATA[  select \* from account  ]]>  </select>    <!-- accountResultMap是account-resultmap.xml中定义的resultmap -->  <select id=*"getAccountById"* parameterType=*"string"* resultType=*"com.hoo.entity.Account"* resultMap=*"accountResultMap"*>  <![CDATA[  select \* from account where account\_id = #{id}  ]]>  </select>    <!-- accountMap.accountResultMap是account-resultmap.xml中定义的resultmap，通过namespace.id找到 -->  <select id=*"getAccountByNames"* parameterType=*"string"* resultMap=*"accountMap.accountResultMap"*>  <![CDATA[  select \* from account where username = #{name}  ]]>  </select>    <sql id=*"user\_name\_pwd"*>  username, password  </sql>    <!-- 自动生成id策略 -->  <insert id=*"addAccount"* useGeneratedKeys=*"true"* keyProperty=*"account\_id"* parameterType=*"account"*>  insert into account(account\_id, status, username, password)  values(#{accountId}, #{status}, #{username}, #{password})  </insert>    <!-- 根据selectKey语句生成主键 -->  <insert id=*"addAccount4Key"* parameterType=*"account"*>  <selectKey keyProperty=*"account\_id"* order=*"BEFORE"* resultType=*"int"*>  select cast(random() \* 10000 as Integer) a from #Tab  </selectKey>  insert into account(account\_id, status, username, password)  values(#{accountId}, #{status}, #{username}, #{password})  </insert>    <update id=*"editAccount"* parameterType=*"account"*>  update account set  status = #{status},  username = #{username},  password = #{password}  where account\_id = #{accountId}  </update>    <delete id=*"removeAccount"* parameterType=*"int"*>  delete from account where account\_id = #{id}  </delete>  </mapper> |

上面的namespace和定义接口类路径对应，所有的sql语句，如select、insert、delete、update的id和方法名称对应。关于更多MyBatis内容的讲解，这里就不赘述的。这里只完成整合！如果你不懂可以去阅读其他关于MyBatis的资料。

1. 为了测试发布，这里使用junit和spring官方提供的spring-test.jar，完成spring框架整合的测试，代码如下：

|  |
| --- |
| **package** com.hoo.mapper;  **import** java.util.List;  **import** javax.inject.Inject;  **import** org.springframework.test.context.ContextConfiguration;  **import** org.springframework.test.context.junit38.AbstractJUnit38SpringContextTests;  **import** com.hoo.entity.Account;  /\*\*  \* <b>function:</b> AccountMapper JUnit测试类  \* **@author** hoojo  \* **@createDate** 2011-4-12 下午04:21:50  \* **@file** AccountMapperTest.java  \* **@package** com.hoo.mapper  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @ContextConfiguration("classpath:applicationContext-\*.xml")  **public** **class** AccountMapperTest **extends** AbstractJUnit38SpringContextTests {    @Inject  //@Named("accountMapper")  **private** AccountMapper mapper;    **public** **void** testGetAccount() {  System.*out*.println(mapper.getAccount());  }    **public** **void** testGetAccountById() {  System.*out*.println(mapper.getAccountById("28"));  }    **public** **void** testGetAccountByName() {  System.*out*.println(mapper.getAccountByName("user"));  }    **public** **void** testGetAccountByNames() {  System.*out*.println(mapper.getAccountByNames("user"));  }    **public** **void** testAdd() {  Account account = **new** Account();  account.setEmail("temp@155.com");  account.setPassword("abc");  account.setRoleId(1);  account.setSalt("ss");  account.setStatus(0);  account.setUsername("Jack");  mapper.addAccount(account);  }    **public** **void** testEditAccount() {  Account acc = mapper.getAccountByNames("Jack");  System.*out*.println(acc);  acc.setUsername("Zhangsan");  acc.setPassword("123123");  mapper.editAccount(acc);  System.*out*.println(mapper.getAccountById(acc.getAccountId() + ""));  }    **public** **void** testRemoveAccount() {  Account acc = mapper.getAccountByNames("Jack");  mapper.removeAccount(acc.getAccountId());  System.*out*.println(mapper.getAccountByNames("Jack"));  }    **public** **void** testAccountList() {  List<Account> acc = mapper.getAllAccount();  System.*out*.println(acc.size());  System.*out*.println(acc);  }  } |

这里的注入并没有使用@Autowired、@Resource、@Qualifier注入，而是使用@Inject、@Named注入方式，Inject注入是JSR330的标准注入方式；而不局限于某个产品，使用于多个产品的使用，推荐使用这种方式；运行后，没有发现问题，就可以继续后续的编码工作了。

1. 定义AccountDao接口及实现代码，代码如下：

|  |
| --- |
| **package** com.hoo.dao;  **import** java.util.List;  **import** org.springframework.dao.DataAccessException;  /\*\*  \* <b>function:</b> Account数据库操作dao接口  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午10:21:38  \* **@file** AccountDao.java  \* **@package** com.hoo.dao  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** AccountDao<T> {    /\*\*  \* <b>function:</b> 添加Account对象信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:50:05  \* **@param** entity Account  \* **@return** boolean 是否成功  \* **@throws** DataAccessException  \*/  **public** **boolean** addAccount(T entity) **throws** DataAccessException;    /\*\*  \* <b>function:</b> 根据id对到Account信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:50:45  \* **@param** id 编号id  \* **@return** Account  \* **@throws** DataAccessException  \*/  **public** T getAccount(Integer id) **throws** DataAccessException;    /\*\*  \* <b>function:</b> 查询所有Account信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:51:45  \* **@param** id 编号id  \* **@return** Account  \* **@throws** DataAccessException  \*/  **public** List<T> getList() **throws** DataAccessException;  } |

接口实现

|  |
| --- |
| **package** com.hoo.dao.impl;  **import** java.util.List;  **import** javax.inject.Inject;  **import** org.springframework.dao.DataAccessException;  **import** org.springframework.stereotype.Repository;  **import** com.hoo.dao.AccountDao;  **import** com.hoo.entity.Account;  **import** com.hoo.mapper.AccountMapper;  /\*\*  \* <b>function:</b> Account数据库操作dao  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午10:25:02  \* **@file** AccountDaoImpl.java  \* **@package** com.hoo.dao.impl  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @SuppressWarnings("unchecked")  @Repository  **public** **class** AccountDaoImpl<T **extends** Account> **implements** AccountDao<T> {    @Inject  **private** AccountMapper mapper;    **public** **boolean** addAccount(T entity) **throws** DataAccessException {  **boolean** flag = **false**;  **try** {  mapper.addAccount(entity);  flag = **true**;  } **catch** (DataAccessException e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  **public** T getAccount(Integer id) **throws** DataAccessException {  T entity = **null**;  **try** {  entity = (T) mapper.getAccountById(String.*valueOf*(id));  } **catch** (DataAccessException e) {  **throw** e;  }  **return** entity;  }  **public** List<T> getList() **throws** DataAccessException {  **return** (List<T>) mapper.getAllAccount();  }  } |

1. 服务层AccountBiz接口及实现代码

接口：

|  |
| --- |
| **package** com.hoo.biz;  **import** java.util.List;  **import** org.springframework.dao.DataAccessException;  /\*\*  \* <b>function:</b> biz层Account接口  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:33:04  \* **@file** AccountBiz.java  \* **@package** com.hoo.biz  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** AccountBiz<T> {  /\*\*  \* <b>function:</b> 添加Account对象信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:50:05  \* **@param** entity Account  \* **@return** boolean 是否成功  \* **@throws** DataAccessException  \*/  **public** **boolean** addAccount(T entity) **throws** DataAccessException;    /\*\*  \* <b>function:</b> 根据id对到Account信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:50:45  \* **@param** id 编号id  \* **@return** Account  \* **@throws** DataAccessException  \*/  **public** T getAccount(Integer id) **throws** DataAccessException;    /\*\*  \* <b>function:</b> 查询所有Account信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:51:45  \* **@param** id 编号id  \* **@return** Account  \* **@throws** DataAccessException  \*/  **public** List<T> getList() **throws** DataAccessException;  } |

实现代码：

|  |
| --- |
| **package** com.hoo.biz.impl;  **import** java.util.List;  **import** javax.inject.Inject;  **import** org.springframework.dao.DataAccessException;  **import** org.springframework.stereotype.Service;  **import** com.hoo.biz.AccountBiz;  **import** com.hoo.dao.AccountDao;  **import** com.hoo.entity.Account;  **import** com.hoo.exception.BizException;  /\*\*  \* <b>function:</b> Account Biz接口实现  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:34:39  \* **@file** AccountBizImpl.java  \* **@package** com.hoo.biz.impl  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  //@Component  @Service  **public** **class** AccountBizImpl<T **extends** Account> **implements** AccountBiz<T> {    @Inject  **private** AccountDao<T> dao;    **public** **boolean** addAccount(T entity) **throws** DataAccessException {  **if** (entity == **null**) {  **throw** **new** BizException(Account.**class**.getName() + "对象参数信息为Empty！");  }  **return** dao.addAccount(entity);  }  **public** T getAccount(Integer id) **throws** DataAccessException {  **return** dao.getAccount(id);  }  **public** List<T> getList() **throws** DataAccessException {  **return** dao.getList();  }  } |

上面用到了一个自定义的异常信息，代码如下：

|  |
| --- |
| **package** com.hoo.exception;  **import** org.springframework.dao.DataAccessException;  /\*\*  \* <b>function:</b>自定义Biz层异常信息  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午11:42:19  \* **@file** BizException.java  \* **@package** com.hoo.exception  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **class** BizException **extends** DataAccessException {  /\*\*  \* **@author** Hoojo  \*/  **private** **static** **final** **long** *serialVersionUID* = 1L;    **public** BizException(String msg) {  **super**(msg);  }  **public** BizException(String msg, Throwable cause) {  **super**(msg, cause);  }  } |

这里只是简单的继承，如果还有其他的异常业务或需求可以进行具体的实现

1. springMVC的控制器，AccountController代码如下：

|  |
| --- |
| **package** com.hoo.controller;  **import** javax.inject.Inject;  **import** javax.servlet.http.HttpServletRequest;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.Model;  **import** org.springframework.web.bind.annotation.ExceptionHandler;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** com.hoo.biz.AccountBiz;  **import** com.hoo.entity.Account;  /\*\*  \* <b>function:</b> Account控制器  \* **@author** hoojo  \* **@createDate** 2011-4-13 上午10:18:02  \* **@file** AccountController.java  \* **@package** com.hoo.controller  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @Controller  @RequestMapping("/account")  **public** **class** AccountController {    @Inject  **private** AccountBiz<Account> biz;    @RequestMapping("/add")  **public** String add(Account acc) {  System.*out*.println(acc);  biz.addAccount(acc);  **return** "redirect:/account/list.do";  }    @RequestMapping("/get")  **public** String get(Integer id, Model model) {  System.*out*.println("##ID:" + id);  model.addAttribute(biz.getAccount(id));  **return** "/show.jsp";  }    @RequestMapping("/list")  **public** String list(Model model) {  model.addAttribute("list", biz.getList());  **return** "/list.jsp";  }    @ExceptionHandler(Exception.**class**)  **public** String exception(Exception e, HttpServletRequest request) {  //e.printStackTrace();  request.setAttribute("exception", e);  **return** "/error.jsp";  }  } |

1. 基本页面代码

index.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>MyBatis 整合 Spring 3.0.5</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  </head>    <body>  <h3>MyBatis 3.0.4 整合 Spring 3.0.5</h3>  <a href=*"account/list.do"*>查询所有</a><br/>  <a href=*"account/add.do?username=abcdef&password=123132&status=2"*>添加</a><br/>  <a href=*"account/get.do?id=25"*>查询</a><br/>  </body>  </html> |

List.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>all Account Result</title>    <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  </head>    <body>  <c:forEach items="${list}" var=*"data"*>  id: ${data.accountId }---name: ${data.username }---password: ${data.password }<hr/>  </c:forEach>  </body>  </html> |

show.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>show Account</title>    <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  </head>    <body>  ${account }<br/>  ${account.username }#${account.accountId }  </body>  </html> |

error.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>Error Page</title>    <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  </head>    <body>  <H2>Exception: ${exception }</H2>  <a href=*"javascript:document.getElementById('show').style.display = 'block';void(0);"*>  详细信息  </a>  <div id=*"show"* style=*"color: red; display: none;"*>  <% Exception ex = (Exception)request.getAttribute("exception"); %>    <% ex.printStackTrace(**new** java.io.PrintWriter(out)); %>  </div>  </body>  </html> |

1. 以上就基本上完成了整个Spring+SpringMVC+MyBatis的整合了。如果你想添加事务管理，得在applicationContext-common.xml中加入如下配置：

|  |
| --- |
| <!-- 配置事务管理器，注意这里的dataSource和SqlSessionFactoryBean的dataSource要一致，不然事务就没有作用了 -->  <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"* />  </bean> |

同时还需要加入aspectjweaver.jar这个jar包；

注意的是：Jdbc的TransactionManager不支持事务隔离级别，我在整个地方加入其它的TransactionManager，增加对transaction的隔离级别都尝试失败！

也许可以用于jpa、jdo、jta这方面的东西。不知道大家对MyBatis的事务是怎么处理的？

1. 对Dao进行扩展封装，运用SqlSessionDaoSupport进行模板的扩展或运用：

BaseDao代码如下：

|  |
| --- |
| **package** com.hoo.dao.impl;  **import** java.util.ArrayList;  **import** java.util.List;  **import** javax.inject.Inject;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.mybatis.spring.support.SqlSessionDaoSupport;  **import** org.springframework.stereotype.Repository;  **import** com.hoo.dao.BaseDao;  /\*\*  \* <b>function:</b> 运用SqlSessionDaoSupport封装Dao常用增删改方法，可以进行扩展  \* **@author** hoojo  \* **@createDate** 2011-4-13 下午06:33:37  \* **@file** BaseDaoImpl.java  \* **@package** com.hoo.dao.impl  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @Repository  @SuppressWarnings({ "unchecked", "unused" })  **public** **class** BaseDaoImpl<T **extends** Object> **extends** SqlSessionDaoSupport **implements** BaseDao<T> {    @Inject  **private** SqlSessionFactory sqlSessionFactory;    **public** **boolean** add(String classMethod, T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  flag = **this**.getSqlSession().insert(classMethod, entity) > 0 ? **true** : **false**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  **public** **boolean** edit(String classMethod, T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  flag = **this**.getSqlSession().update(classMethod, entity) > 0 ? **true** : **false**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  **public** T get(String classMethod, T entity) **throws** Exception {  T result = **null**;  **try** {  result = (T) **this**.getSqlSession().selectOne(classMethod, entity);  } **catch** (Exception e) {  **throw** e;  }  **return** result;  }  **public** List<T> getAll(String classMethod) **throws** Exception {  List<T> result = **new** ArrayList<T>();  **try** {  result = **this**.getSqlSession().selectList(classMethod);  } **catch** (Exception e) {  **throw** e;  }  **return** result;  }  **public** **boolean** remove(String classMethod, T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  flag = **this**.getSqlSession().delete(classMethod, entity) > 0 ? **true** : **false**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  } |

值得说明的是，这个类继承了SqlSessionDaoSupport，它需要我们帮助它注入SqlSessionFactory或是SqlSessionTemplate，如果两者都被注入将忽略SqlSessionFactory属性，使用SqlSessionTemplate模板。

继承SqlSessionDaoSupport后，可以拿到SqlSession完成数据库的操作；

1. 对Dao进行扩展封装，运用SqlSessionTemplate进行模板的扩展或运用：

首先看看这个组件中运用的一个Mapper的基类接口：

|  |
| --- |
| **package** com.hoo.mapper;  **import** java.util.List;  **import** org.springframework.dao.DataAccessException;  /\*\*  \* <b>function:</b> BaseSqlMapper继承SqlMapper，对Mapper进行接口封装，提供常用的增删改查组件；  \* 也可以对该接口进行扩展和封装  \* **@author** hoojo  \* **@createDate** 2011-4-14 上午11:36:41  \* **@file** BaseSqlMapper.java  \* **@package** com.hoo.mapper  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  **public** **interface** BaseSqlMapper<T> **extends** SqlMapper {    **public** **void** add(T entity) **throws** DataAccessException;    **public** **void** edit(T entity) **throws** DataAccessException;    **public** **void** remvoe(T entity) **throws** DataAccessException;    **public** T get(T entity) **throws** DataAccessException;    **public** List<T> getList(T entity) **throws** DataAccessException;  } |

该接口继承SqlMapper接口，但是该接口没有MyBatis的mapper实现。需要我们自己的业务mapper继承这个接口，完成上面的方法的实现。

看看继承SqlSessionTemplate的BaseMapperDao代码：

|  |
| --- |
| **package** com.hoo.dao.impl;  **import** java.util.List;  **import** javax.inject.Inject;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.mybatis.spring.SqlSessionTemplate;  **import** org.springframework.stereotype.Repository;  **import** com.hoo.dao.BaseMapperDao;  **import** com.hoo.mapper.BaseSqlMapper;  /\*\*  \* <b>function:</b>运用SqlSessionTemplate封装Dao常用增删改方法，可以进行扩展  \* **@author** hoojo  \* **@createDate** 2011-4-14 下午12:22:07  \* **@file** BaseMapperDaoImpl.java  \* **@package** com.hoo.dao.impl  \* **@project** MyBatisForSpring  \* **@blog** http://blog.csdn.net/IBM\_hoojo  \* **@email** hoojo\_@126.com  \* **@version** 1.0  \*/  @SuppressWarnings("unchecked")  @Repository  **public** **class** BaseMapperDaoImpl<T> **extends** SqlSessionTemplate **implements** BaseMapperDao<T> {    @Inject  **public** BaseMapperDaoImpl(SqlSessionFactory sqlSessionFactory) {  **super**(sqlSessionFactory);  }    **private** Class<? **extends** BaseSqlMapper> mapperClass;    **public** **void** setMapperClass(Class<? **extends** BaseSqlMapper> mapperClass) {  **this**.mapperClass = mapperClass;  }  **private** BaseSqlMapper<T> getMapper() {  **return** **this**.getMapper(mapperClass);  }    **public** **boolean** add(T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  **this**.getMapper().add(entity);  flag = **true**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  **public** **boolean** edit(T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  **this**.getMapper().edit(entity);  flag = **true**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  **public** T get(T entity) **throws** Exception {  **return** **this**.getMapper().get(entity);  }  **public** List<T> getAll() **throws** Exception {  **return** **this**.getMapper().getList(**null**);  }  **public** **boolean** remove(T entity) **throws** Exception {  **boolean** flag = **false**;  **try** {  **this**.getMapper().remvoe(entity);  flag = **true**;  } **catch** (Exception e) {  flag = **false**;  **throw** e;  }  **return** flag;  }  } |

上面这个类继承了SqlSessionTemplate，这个类需要提供一个构造函数。这里提供的是SqlSessionFactory的构造函数，通过该函数注入SqlSessionFactory即可完成数据库操作；

例外的是这个类还有一个关键属性mapperClass，这个class需要是BaseSqlMapper接口或是子接口，然后通过SqlSessionTemplate模板获得当前设置的Class的Mapper对象，完成数据库操作。

该类的测试代码：

|  |
| --- |
| @ContextConfiguration("classpath:applicationContext-\*.xml")  **public** **class** BaseMapperDaoImplTest **extends** AbstractJUnit38SpringContextTests {    @Inject  **private** BaseMapperDao<Company> dao;    **public** **void** init() {  dao.setMapperClass(CompanyMapper.**class**);  }    **public** **void** testGet() **throws** Exception {  init();  Company c = **new** Company();  c.setCompanyId(4);  System.*out*.println(dao.get(c));  }    **public** **void** testAdd() **throws** Exception {  init();  Company c = **new** Company();  c.setAddress("北京中关村");  c.setName("beijin");  System.*out*.println(dao.add(c));  }  } |

一般情况下，你可以在一个Dao中注入BaseMapperDao，紧跟着需要设置MapperClass。只有设置了MapperClass后，BaseMapperDao才能获取对应mapper，完成相关的数据库操作。当然你可以在这个Dao中将SqlSessionTemplate、SqlSession暴露出来，当BaseMapperDao的方法不够用，可以进行扩展。