<http://blog.csdn.net/a1610770854/article/details/53572337>

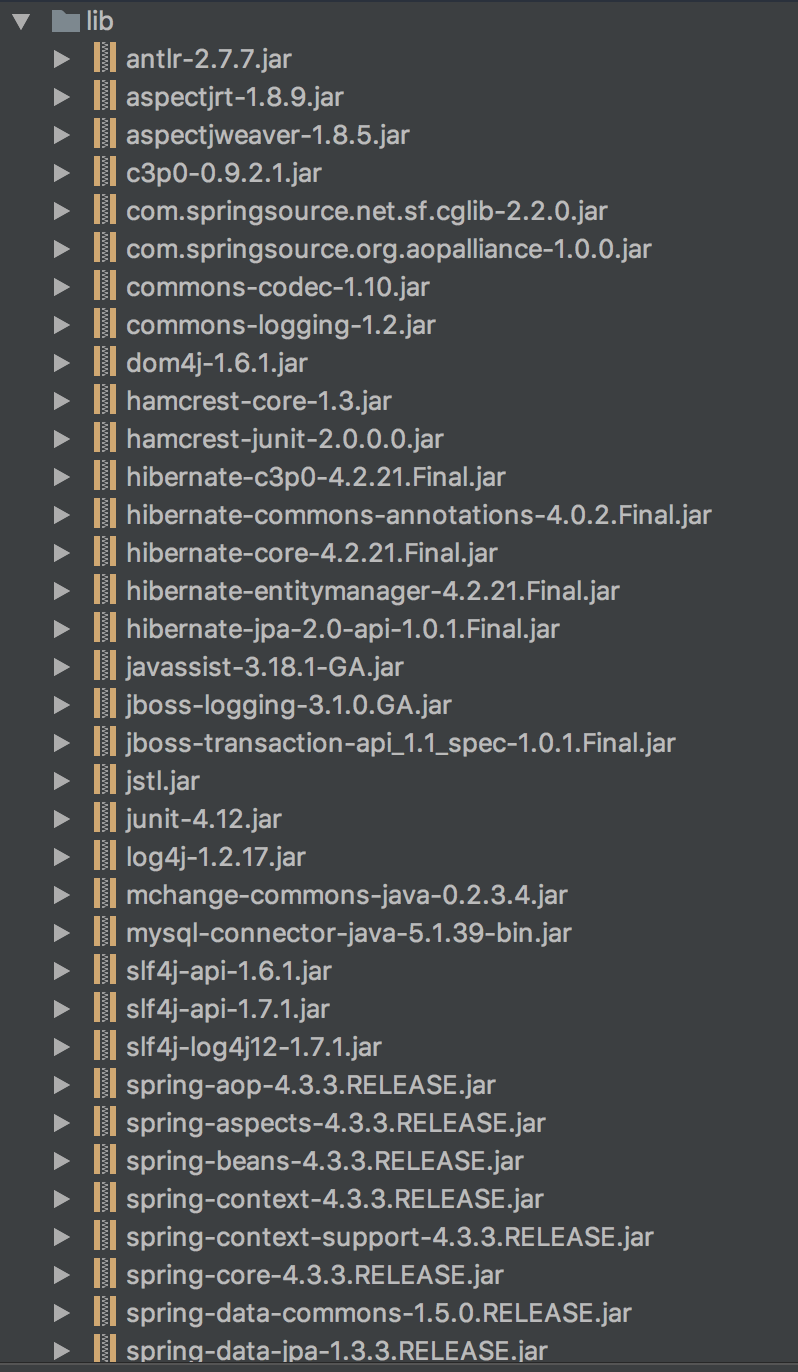
版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/a1610770854/article/details/53572337)

# 1.添加依赖包：

[**spring**](http://lib.csdn.net/base/javaee)和**[hibernate](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "_blank)**的包还是好找的，记得加入hibernate/lib/jpa里面的jar包，接着最麻烦的事jpa了，需要添加spring-data的jar包了，给个网址吧：[下载spring-data](http://repo.spring.io/release/org/springframework/data/" \t "_blank)

我们需要的是spring-data-commons和spring-data-jpa，记得 如果包倒入的不对的话，会出现包冲突的一场，我个人使用的是commoms-1.5.0和jpa-1.3.3，这两个是兼容的，还需要的是slf4j的jar，因为这个也是一个依赖包。附上整个项目的包结构吧：





# 2.配置xml

先是配置applicationContext.xml

**[html]** [view plain](http://blog.csdn.net/a1610770854/article/details/53572337) [copy](http://blog.csdn.net/a1610770854/article/details/53572337)

[print?](http://blog.csdn.net/a1610770854/article/details/53572337)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:tx="http://www.springframework.org/schema/tx"
6. xmlns:aop="http://www.springframework.org/schema/aop"
7. xmlns:jpa="http://www.springframework.org/schema/data/jpa"
8. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
9. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
10. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
11. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd
12. http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.2.xsd"**>**
14. <!--自动扫描包-->
15. **<context:component-scan** base-package="cn.limbo"**>**
16. <!--不要将Controller扫进来，否则aop无法使用-->
17. **<context:exclude-filter** type="annotation" expression="org.springframework.stereotype.Controller"**/>**
18. **</context:component-scan>**
20. <!--使Aspect注解起作用,自动为匹配的类生成代理对象-->
21. **<aop:aspectj-autoproxy** proxy-target-class="true"**/>**
23. <!--引入properties-->
24. **<context:property-placeholder** location="classpath:hibernate.properties"**/>**
26. <!--配置数据源-->
27. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>**
28. **<property** name="user" value="${dataSource.username}"**/>**
29. **<property** name="password" value="${dataSource.password}"**/>**
30. **<property** name="jdbcUrl" value="${dataSource.url}"**/>**
31. **<property** name="driverClass" value="${dataSource.driverClassName}"**/>**
32. **</bean>**
34. <!--<!–sessionFactory–>-->
35. <!--<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">-->
36. <!--<!– 配置数据源属性 –>-->
37. <!--<property name="dataSource" ref="dataSource"/>-->
38. <!--<!– 配置扫描的实体包(pojo) –>-->
39. <!--<property name="namingStrategy">-->
40. <!--<bean class="org.hibernate.cfg.ImprovedNamingStrategy"/>-->
41. <!--</property>-->
42. <!--<property name="packagesToScan" value="cn.limbo.entity"/>-->
44. <!--<property name="hibernateProperties">-->
45. <!--<props>-->
46. <!--<prop key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>-->
47. <!--<prop key="hibernate.show\_sql">true</prop>-->
48. <!--<prop key="hibernate.format\_sql">true</prop>-->
49. <!--<prop key="hibernate.hbm2ddl.auto">update</prop>-->
50. <!--</props>-->
51. <!--</property>-->
52. <!--</bean>-->
54. **<bean** id="persistenceProvider"
55. class="org.hibernate.ejb.HibernatePersistence"**/>**
57. **<bean** id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"**>**
58. **<property** name="database" value="MYSQL"**/>**
59. **</bean>**
61. **<bean** id="jpaDialect" class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"**/>**
63. <!--jpa工厂-->
64. **<bean** id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"**>**
65. <!--数据源-->
66. **<property** name="dataSource" ref="dataSource"**/>**
67. **<span** style="white-space:pre"**>**  **</span>**<!--持久层提供者-->
68. **<property** name="persistenceProvider" ref="persistenceProvider"**/>**
69. <!--适配器-->
70. **<property** name="jpaVendorAdapter" ref="jpaVendorAdapter"**/>**
72. **<property** name="jpaDialect" ref="jpaDialect"**/>**
74. **<property** name="jpaProperties"**>**
75. **<props>**
76. **<prop** key="hibernate.dialect"**>**${dataSource.dialect}**</prop>**
77. **<prop** key="hibernate.hbm2ddl.auto"**>**${dataSource.hbm2ddl.auto}**</prop>**
78. **<prop** key="hibernate.show\_sql"**>**${dataSource.show\_sql}**</prop>**
79. **<prop** key="hibernate.format\_sql"**>**${dataSource.format\_sql}**</prop>**
80. **</props>**
81. **</property>**
83. **<property** name="packagesToScan"**>**
84. **<list>**
85. **<value>**cn.limbo.entity**</value>**
86. **</list>**
87. **</property>**
88. **</bean>**
90. **<jpa:repositories** base-package="cn.limbo.dao"
91. entity-manager-factory-ref="entityManagerFactory"
92. transaction-manager-ref="transactionManager"**/>**
94. <!-- 配置Hibernate 的事物管理器 -->
95. **<bean** id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"**>**
96. **<property** name="entityManagerFactory" ref="entityManagerFactory"**/>**
97. **</bean>**

100. **<tx:annotation-driven** transaction-manager="transactionManager"**/>**
101. **</beans>**

可以看到，如果引入jpa之后原先的hibernate的sessionFactory的bean配置就不需要了，转而是配置entityManagerFactory了，而且事务的管理的bean也变成了JpaTransactionManager了，这个比较简单。

如果项目是web项目的话，一定一定要记得更改web.xml原先对sessionFactory的fliter，否则项目启动的时候会报找不到sessionFactory的错误，诶，当时有点粗心了。

web.xml

**[html]** [view plain](http://blog.csdn.net/a1610770854/article/details/53572337) [copy](http://blog.csdn.net/a1610770854/article/details/53572337)

[print?](http://blog.csdn.net/a1610770854/article/details/53572337)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"
5. version="3.1"**>**
7. **<display-name>**Google Authenticator**</display-name>**
8. **<welcome-file-list>**
9. **<welcome-file>**login.html**</welcome-file>**
10. **</welcome-file-list>**
12. <!--配置IOC容器-->
13. **<context-param>**
14. **<param-name>**contextConfigLocation**</param-name>**
15. **<param-value>**classpath:applicationContext.xml**</param-value>**
16. **</context-param>**
17. **<listener>**
18. **<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>**
19. **</listener>**
21. <!--配置SpringMVC 的 DispatcherServlet-->
22. **<servlet>**
23. **<servlet-name>**springmvc**</servlet-name>**
24. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
25. **<init-param>**
26. **<param-name>**contextConfigLocation**</param-name>**
27. **<param-value>**classpath:spring-mvc.xml**</param-value>**
28. **</init-param>**
29. **<load-on-startup>**1**</load-on-startup>**
30. **</servlet>**
31. **<servlet-mapping>**
32. **<servlet-name>**springmvc**</servlet-name>**
33. **<url-pattern>**\*.do**</url-pattern>**
34. **</servlet-mapping>**
36. <!-- 配置编码方式过滤器,注意一点:要配置在所有过滤器的前面 -->
37. **<filter>**
38. **<filter-name>**characterEncodingFilter**</filter-name>**
39. **<filter-class>**org.springframework.web.filter.CharacterEncodingFilter**</filter-class>**
40. **<init-param>**
41. **<param-name>**encoding**</param-name>**
42. **<param-value>**UTF-8**</param-value>**
43. **</init-param>**
44. **</filter>**
45. **<filter-mapping>**
46. **<filter-name>**characterEncodingFilter**</filter-name>**
47. **<url-pattern>**\*.do**</url-pattern>**
48. **</filter-mapping>**
50. <!-- 为了使用SpringMVC框架实现REST风格,需要配置  HiddenHttpMethodFilter-->
51. **<filter>**
52. **<filter-name>**hiddenHttpMethodFilter**</filter-name>**
53. **<filter-class>**org.springframework.web.filter.HiddenHttpMethodFilter**</filter-class>**
54. **</filter>**
55. **<filter-mapping>**
56. **<filter-name>**hiddenHttpMethodFilter**</filter-name>**
57. **<url-pattern>**\*.do**</url-pattern>**
58. **</filter-mapping>**
60. <!--hibernate对对象的管理是基于session的，如果开启了延迟加载，
61. 对于关联对象的查询可能会在渲染jsp的时候才发生，但是这个时候hibernate的当前session默认已经关闭了，
62. 就会抛异常，所以我们经常需要hibernate在渲染jsp页面的时候还开着session，
63. 这就需要在web.xml中配置一个拦截所有请求的filter--**>**
64. <!--如果是hibernate的话这样配置-->
65. <!--<filter>-->
66. <!--<filter-name>hibernateFilter</filter-name>-->
67. <!--<filter-class>org.springframework.orm.hibernate4.support.OpenSessionInViewFilter</filter-class>-->
68. <!--</filter>-->
69. <!--<filter-mapping>-->
70. <!--<filter-name>hibernateFilter</filter-name>-->
71. <!--<url-pattern>\*.do</url-pattern>-->
72. <!--</filter-mapping>-->
74. <!--jpa配置的方式，需要配置entityManager-->
75. **<filter>**
76. **<filter-name>**entityManagerInViewFilter**</filter-name>**
77. **<filter-class>**org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter**</filter-class>**
78. **</filter>**
79. **<filter-mapping>**
80. **<filter-name>**entityManagerInViewFilter**</filter-name>**
81. **<url-pattern>**\*.do**</url-pattern>**
82. **</filter-mapping>**
84. **</web-app>**

# 3.编写dao层

经过上面的配置之后，已经可以使用jpa了，接下来就是编写dao层的东西了

先贴上我的dao层实现

**[java]** [view plain](http://blog.csdn.net/a1610770854/article/details/53572337) [copy](http://blog.csdn.net/a1610770854/article/details/53572337)

[print?](http://blog.csdn.net/a1610770854/article/details/53572337)

1. **package** cn.limbo.dao;
3. **import** cn.limbo.entity.User;
4. **import** org.springframework.data.jpa.repository.Modifying;
5. **import** org.springframework.data.jpa.repository.Query;
6. **import** org.springframework.data.repository.RepositoryDefinition;
7. **import** org.springframework.data.repository.query.Param;
9. **import** java.util.List;
11. /\*\*
12. \* user表的数据操作
13. \* Created by limbo on 2016/11/26.
14. \*/
16. @RepositoryDefinition(domainClass = User.**class**, idClass = Integer.**class**)
17. **public** **interface** UserDao{
19. @Query("select u from User u where u.ID = ?1")
20. **public** User getUserByID(**int** ID);
22. @Query("select u from User u where u.name = ?1")
23. **public** User getUserByName(String userName);
25. @Query("select u from User u")
26. **public** List<User> getAllUsers();
28. **public** **void** save(User user);
30. @Modifying
31. @Query("delete from User u where u.ID = ?1")
32. **public** **void** delete(**int** ID);
34. @Modifying
35. @Query("update User u set u.name = :name , u.password = :password where u.ID = :ID")
36. **public** **void** update(@Param("ID") Integer ID ,@Param("name")  String name , @Param("password") String password);
38. }

实现jpa的方式有两种，一个是集成Repository<T,ID>接口，其中T表示你要操作的实体类，而ID表示实体类里面的ID的类型；还有一种就是加上注解@RepositoryDefinition

其中有两个参数一定要传值，domainClass表示实体类的Class，idClass表示ID的类型的class，就如我上面进行配置的。

jpa的接口类型有很多，一般用的最多的是JpaRepository，还有CrudRepository,而Repository只是最基础的，没有实现任何的方法，是一个空的接口，所以可以进行自定义。

接下来就是使用dao的东西了，贴上我的service层

UserService.java

**[java]** [view plain](http://blog.csdn.net/a1610770854/article/details/53572337) [copy](http://blog.csdn.net/a1610770854/article/details/53572337)

[print?](http://blog.csdn.net/a1610770854/article/details/53572337)

1. **package** cn.limbo.service;
3. **import** cn.limbo.entity.User;
5. **import** java.util.List;
7. /\*\*
8. \* Created by limbo on 2016/11/26.
9. \*/
10. **public** **interface** UserService {
12. **public** User getUserByID(**int** ID);
14. **public** List<User> getAllUsers();
16. **public** **void** addUser(String name,String password);
18. **public** **void** deleteUserByID(**int** ID);
20. **public** **void** updateUser(User user);
22. **public** **boolean** isExist(String userName);
24. }

UserServiceImpl.java

**[java]** [view plain](http://blog.csdn.net/a1610770854/article/details/53572337) [copy](http://blog.csdn.net/a1610770854/article/details/53572337)

[print?](http://blog.csdn.net/a1610770854/article/details/53572337)

1. **package** cn.limbo.service.impl;
3. **import** cn.limbo.dao.UserDao;
4. **import** cn.limbo.entity.User;
5. **import** cn.limbo.service.UserService;
6. **import** org.springframework.beans.factory.annotation.Autowired;
7. **import** org.springframework.stereotype.Service;
8. **import** org.springframework.transaction.annotation.Transactional;
10. **import** java.util.List;
12. /\*\*
13. \* Created by limbo on 2016/11/26.
14. \*/
15. @Service("userService")
16. @Transactional
17. **public** **class** UserServiceImpl **implements** UserService{
19. @Autowired
20. **private** UserDao userDao;
22. @Override
23. **public** User getUserByID(**int** ID) {
24. **return** userDao.getUserByID(ID);
25. }
27. @Override
28. **public** List<User> getAllUsers() {
29. **return** userDao.getAllUsers();
30. }
32. @Override
33. **public** **void** addUser(String name,String password) {
34. User user = **new** User(name,password);
35. userDao.save(user);
36. }
38. @Override
39. **public** **void** deleteUserByID(**int** ID) {
40. userDao.delete(ID);
41. }
43. @Override
44. **public** **void** updateUser(User user) {
45. userDao.update(user.getID(),user.getName(),user.getPassword());
46. }
48. @Override
49. **public** **boolean** isExist(String userName) {
50. **if**(userDao.getUserByName(userName) != **null**)
51. **return** **true**;
52. **return** **false**;
53. }
54. }

其实最重要的是就是在需要在实现类前面加上@Transactional注解。到此，jpa配置完成咯。

附上当时我配置的时候的参考文章

<http://www.cnblogs.com/dreamroute/p/5173896.html>

<http://www.ibm.com/developerworks/cn/opensource/os-cn-spring-jpa/>

整个项目我发在了github上了，有兴趣可以看看

<https://github.com/NetFilx/Google-Authenticator>