### 建立一个Maven web工程

建立后包含main文件和test文件，工程的类写在main类中，将需要测试的类写在test类中，可以使用Junit和TestNG对框架内容进行测试

@ContextConfiguration("classpath\*:/smart-context.xml") //测试前先启动spring容器  
public class UserServiceTest extends AbstractTransactionalTestNGSpringContextTests{

//这个例子中对数据库的操作都是在事务下进行的，所以测试时使用AbstractTransactionalTestNGSpringContextTests  
 private UserService userService;  
 @Autowired  
 public void setUserService(UserService userService){  
 this.userService = userService;  
 }  
  
 @Test //可单独对该方法进行测试  
 public void hasMatchUser(){  
 boolean b1 = userService.hasMatchUser("admin", "123456");  
 boolean b2 = userService.hasMatchUser("admin", "1111");  
 *assertTrue*(b1);  
 *assertTrue*(!b2);  
 }  
  
 @Test  
 public void findUserByUserName(){  
 User user = userService.findUserByUserName("admin");  
 *assertEquals*(user.getUserName(),"admin");  
 }  
}

在main文件下有java文件resources文件（存放各种资源配置文件），webapp文件（页面为文件）

在mian文件下分别建立包domain(持久化对象)，dao(存放对数据库进行操作的类)属于持久层，service(对dao的操作进行组合操作的业务逻辑)属于业务层，web(前端页面操作在后台的操作)属于可视层。

### 建立领域对象（实体类）

依照数据库表格信息建立对象类，并且只设置get 和 set 方法。领域对象一般要实现Serializable接口，实现序列化，

**Java序列化**是指把Java对象转换为字节序列的过程；而**Java反序列化**是指把字节序列恢复为Java对象的过程。

**2.什么需要序列化与反序列化**

 我们知道，当两个进程进行远程通信时，可以相互发送各种类型的数据，包括文本、图片、音频、视频等， 而这些数据都会以二进制序列的形式在网络上传送。那么当两个Java进程进行通信时，能否实现进程间的对象传送呢？答案是可以的。如何做到呢？这就需要Java序列化与反序列化了。换句话说，一方面，发送方需要把这个Java对象转换为字节序列，然后在网络上传送；另一方面，接收方需要从字节序列中恢复出Java对象。

 当我们明晰了为什么需要Java序列化和反序列化后，我们很自然地会想Java序列化的好处。其好处一是实现了数据的持久化，通过序列化可以把数据永久地保存到硬盘上（通常存放在文件里），二是，利用序列化实现远程通信，即在网络上传送对象的字节序列。

### Dao文件

Dao文件主要包括对数据增删改查操作封装，Spring框架对简单的JDBC事务操作进行了薄层封装，spring JDBC模板类org.springframework.jdbc.core.JdbcTemplate，

@Repository //通过spring注解定义一个Dao  
public class UserDao {  
 private JdbcTemplate jdbcTemplate;  
  
 @Autowired //自动注入JdbcTemplate的Bean  
 public void setJdbcTemplate(JdbcTemplate jdbcTemplate){  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 private final static String *MATCH\_COUNT\_SQL* = "select *count*(\*) from t\_user "  
 +"where user\_name = ? and password = ?";  
  
 private final static String *UPDATE\_LOGIN\_INFO\_SQL* = "update t\_user set "  
 +"last\_visit = ?,last\_ip = ?,credits = ? where user\_id = ?";  
  
 private final static String *QUERY\_BY\_USERNAME* = "select user\_id,user\_name,credits " +  
 "from t\_user where user\_name =?";  
  
 public int getMatchCount(String userName,String password){  
 return jdbcTemplate.queryForObject(*MATCH\_COUNT\_SQL*,new Object[]{userName,password},Integer.class); //通过传入的数组参数对+"where user\_name = ? and password = ?";中的？占位符进行赋值  
 }  
  
 public User findUserByUserName(final String userName){  
 final User user = new User();  
 jdbcTemplate.query(*QUERY\_BY\_USERNAME*, new Object[]{userName},  
 //匿名类方式实现的回掉函数  
 new RowCallbackHandler() {  
 public void processRow(ResultSet resultSet) throws SQLException {  
 user.setUserId(resultSet.getInt("user\_id"));  
 user.setUserName(userName);  
 user.setCredits(resultSet.getInt("credits"));  
 }  
 });  
 return user;  
 }  
  
 public void updateLofinInfo(User user){  
 jdbcTemplate.update(*UPDATE\_LOGIN\_INFO\_SQL*,new Object[]{user.getLastVisit(),user.getLastIp(),user.getCredits(),user.getUserId()});  
 }  
}

#### 在Spring中装配Dao

<context:component-scan base-package="com.smart.dao"></context:component-scan>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  
 destroy-method="close"  
 p:driverClassName="com.mysql.jdbc.Driver"  
 p:url="jdbc:mysql://localhost:3306/sampledb"  
 p:username="root"  
 p:password="Lxb090288"/>  
  
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate"  
 p:dataSource-ref="dataSource"/>

### 业务层

业务层主要时对Dao对象中方法的组合使用，这里使用

@Transactional  
public void loginSuccess(User user){  
 user.setCredits(5+user.getCredits());  
 LoginLog loginLog = new LoginLog();  
 loginLog.setUserId(user.getUserId());  
 loginLog.setIp(user.getLastIp());  
 loginLog.setLoginDate(user.getLastVisit());  
 userDao.updateLofinInfo(user);  
 loginLogDao.insertLoginLog(loginLog);  
}

@Transactional将方法标注为事务注解，可以简单的完成事务的所有操作。

#### 在Spring中装配业务层

<context:component-scan base-package="com.smart.service"></context:component-scan>

//配置扫描

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"  
 p:dataSource-ref="dataSource"/>//装配事务管理器

<aop:config proxy-target-class="true">  
 <aop:pointcut id="serviceMethod" expression=" (execution(\* com.smart.service..\*(..))) and   
 (@annotation(org.springframework.transaction.annotation.Transactional))"/>  
</aop:config>//通过aop面向切面配置增强事务功能，让service包下的所有bean方法拥有事务  
  
<tx:advice id="txAdvice" transaction-manager="transactionManager">  
 <tx:attributes>  
 <tx:method name="\*"/>  
 </tx:attributes>  
</tx:advice>

### web文件（展现层）

//页面信息提交对象

public class LoginCommand {

private String userName;

private String password;

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

//标注 contorller

@Controller  
public class LoginController {  
 private UserService userService;  
  
 @RequestMapping(value = "/index.html") //通过RequestMapping来实现对页面action或href的对应操作  
 public String LoginPage(){  
 return "login";  
 }  
  
 @RequestMapping(value = "/loginCheck.html")  
 public ModelAndView loginCheck(HttpServletRequest request,LoginCommand loginCommand){  
 boolean isValidUser = userService.hasMatchUser(loginCommand.getUserName(),loginCommand.getPassword());  
 if(!isValidUser){  
 return new ModelAndView("login","error","用户名或密码错误");  
 }  
 else {  
 User user = userService.findUserByUserName(loginCommand.getUserName());  
 user.setLastIp(request.getLocalAddr());  
 user.setLastVisit(new Date());  
 userService.loginSuccess(user);  
 request.getSession().setAttribute("user",user);  
 return new ModelAndView("main");  
 }  
 }  
 @Autowired  
 public void setUserSrevice(UserService userService){  
 this.userService = userService;  
 }  
}