

---

## 知识点列表

编号	名称	描述	级别
1	注解方式实现 SSH 整合	通过案例掌握使用注解方式整合 SSH	**
2	Spring 管理事务的策略	了解 Spring 管理事务策略	*
3	AOP 动态代理	了解动态代理原理，属于偏难的知识点	*
4	Spring 框架结构	了解并掌握 Spring 框架结构	*
5	Spring MVC	掌握 Spring MVC 的开发流程，明白学习框架技术要首先了解其流程	*
6	SSH 重构当当注意事项	作为课后作业。使用 SSH 重构当当网项目	*

注：    \*\*"理解级别    \*\*\*"掌握级别    \*\*\*\*"应用级别

---

## 目录

1. 注解方式实现 SSH 整合 **.....	3
【案例 1】注解方式实现 SSH 整合 **.....	3
2. Spring 管理事务的策略 *.....	18
3. AOP 动态代理 *.....	18
【案例 2】动态代理 *.....	18
4. Spring 框架结构 *.....	27
5. Spring MVC *.....	28
【案例 3】Spring MVC **.....	29
6. ssh2 重构当当注意事项 **.....	41

---

## 1. 注解方式实现 SSH 整合 \*\*

### 【案例 1】注解方式实现 SSH 整合 \*\*

#### 1) 拷贝 spring4 工程为 spring5

请下载 spring4

使用注解方式

首先我们开启组件扫描，将 bean 们扫描进来

#### 2) 修改 ssh.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver">
        </property>
        <property name="url" value="jdbc:mysql:///test"> </property>
        <property name="username" value="root"> </property>
        <property name="password" value="root"> </property>
        <property name="maxActive" value="10"> </property>
        <property name="initialSize" value="2"> </property>
```

```

</bean>

<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"></property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<!-- 注释掉需要注解注入的bean
<bean id="userDao" class="tarena.dao.impl.HibernateUserDAOImpl">
    <property name="sessionFactory"
ref="mySessionFactory"></property>
</bean>

<bean id="userService" class="tarena.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"></property>
</bean>
-->

<!-- 启用自动扫描注解 -->
<context:component-scan base-package="tarena">
</context:component-scan>

<!-- 声明式事务控制 -->
<bean id="txManager"

```

```

class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory"
ref="mySessionFactory"> </property>
    </bean>

    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="find*" read-only="true"
propagation="NOT_SUPPORTED"/>
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>

    <aop:config>
        <aop:pointcut expression="within(tarena.service..*)"
id="servicePointcut"/>
        <aop:advisor advice-ref="txAdvice"
pointcut-ref="servicePointcut"/>
    </aop:config>

</beans>

```

需要**注意**的是注解方式只适用于自定义的 bean，所以 dataSource、SessionFactory 仍然保留

### 3) 修改 UserServiceImpl

加入注解，将 Service 作为 bean 纳入 Spring 容器中管理

```

package tarena.service.impl;

import org.springframework.stereotype.Service;
import tarena.dao.UserDAO;
import tarena.pojo.User;
import tarena.service.UserService;

@Service("userService")
public class UserServiceImpl implements UserService {
    //默认采用名称对应规则将Spring容器中dao注入

```

```

private UserDao userDao;

public UserDao getUserDao() {
    return userDao;
}

public void setUserDao(UserDao userDao) {
    this.userDao = userDao;
}

public boolean findLogin(User user) {

    User usr = userDao.findByEmail(user.getEmail());
    if(usr != null){
        if(usr.getPassword().equals(user.getPassword())){
            return true;
        }
    }
    return false;
}
}

```

#### 4) 修改 HibernateUserDAOImpl

加入注解，将 DAO 作为 bean 纳入 Spring 容器中管理

```

package tarena.dao.impl;

import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.stereotype.Repository;
import tarena.dao.UserDAO;
import tarena.pojo.User;

@Repository("userDao")
public class HibernateUserDAOImpl
    extends HibernateDaoSupport implements UserDAO {

    public User findByEmail(String email) {
        String hql = "from User where email=?";

```

```

        List<User> list =
            this.getHibernateTemplate().find(hql,new Object[]{email});
        User user = null;
        if(!list.isEmpty()){
            user = list.get(0);
        }
        return user;
    }
}

```

其次，将 DAO 注入到 Service 中

### 5) 修改 UserServiceImpl

加入注解，将 DAO 注入到 Service 中

```

package tarena.service.impl;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;
import tarena.dao.UserDAO;
import tarena.pojo.User;
import tarena.service.UserService;

@Service("userService")
public class UserServiceImpl implements UserService {
    //默认采用名称对应规则将Spring容器中dao注入
    @Resource(name="userDao")
    private UserDAO userDao;

    public UserDAO getUserDao() {return userDao;}
    public void setUserDao(UserDAO userDao) {
        this.userDao = userDao;}

    public boolean findLogin(User user) {

        User usr = userDao.findByEmail(user.getEmail());
        if(usr != null){
            if(usr.getPassword().equals(user.getPassword())){

```

```

        return true;
    }
}
return false;
}
}

```

但是，现在我们又遇到一个问题，HibernateUserDAOImpl 需要使用 [SessionFactory](#)，那么我们需要将 SessionFactory 注入到其中。  
如何做？

## 6) 修改 HibernateUserDAOImpl

首先，

随便写一个方法，这样我们就可以使用注解@Resource 从 Spring 容器中取出配置好的 bean 名为 mySessionFactory；

其次，

取出 SessionFactory 后，我们可以调用父类 HibernateUserSupport 的 set 方法（**super**.setSessionFactory(sessionFactory)），将 sessionFactory 对象注入给父类对象。

最后，如此两个步骤，我们就完成了 sessionFactory 的注入。

```

package tarena.dao.impl;

import java.util.List;
import javax.annotation.Resource;
import org.hibernate.SessionFactory;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.stereotype.Repository;
import tarena.dao.UserDAO;
import tarena.pojo.User;

@Repository("userDao")
public class HibernateUserDAOImpl
    extends HibernateDaoSupport implements UserDAO {

    @Resource(name="mySessionFactory")
    private void setMySessionFactory(SessionFactory sessionFactory){

```



```

        super.setSessionFactory(sessionFactory);
    }

    public User findByEmail(String email) {
        String hql = "from User where email=?";
        List<User> list =
            this.getHibernateTemplate().find(hql,new Object[]{email});
        User user = null;
        if(!list.isEmpty()){
            user = list.get(0);
        }
        return user;
    }
}

```

接下来该使用注解替换事务控制了

注解方式实现事务控制

## 7) 修改 ssh.xml

```

<bean id="myDataSource"
class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver">
    </property>
    <property name="url" value="jdbc:mysql:///test"> </property>
    <property name="username" value="root"> </property>
    <property name="password" value="root"> </property>
    <property name="maxActive" value="10"> </property>
    <property name="initialSize" value="2"> </property>
</bean>

<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"> </property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml </value>

```

```

        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>

<!-- 启用自动扫描注解 -->
<context:component-scan base-package="tarena">
</context:component-scan>

<!-- 启用事务控制注解 -->
<bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory"
        ref="mySessionFactory"></property>
</bean>

<tx:annotation-driven transaction-manager="txManager"/>

<!--
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="find*" read-only="true"
            propagation="NOT_SUPPORTED"/>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<aop:config>

```

```

        <aop:pointcut expression="within(tarena.service..*)"
        id="servicePointcut"/>
        <aop:advisor advice-ref="txAdvice"
        pointcut-ref="servicePointcut"/>
    </aop:config>
-->
</beans>

```

配置好“启用事务控制注解”，接下来使用它  
我们可以写在类的前面，表示该类中所有的方法都是用事务控制

```

11
12 @Service
13 @Transactional
14 public class UserServiceImpl implements UserService {
15     //默认采用名称对应规则将Spring容器中dao注入
16     @Resource(name="userDao")
17     private UserDAO userDao;
18
19     public UserDAO getUserDao() {
20

```

在@Transactional 中有很多的属性可以使用，使用方法是相同的

```

12
13 @Service
14 @Transactional(readOnly=true, propagation=Propagation.NESTED)
15 public class UserServiceImpl implements UserService {
16     //默认采用名称对应规则将Spring容器中dao注入
17     @Resource(name="userDao")
18     private UserDAO userDao;

```

当然，我们可以直接在方法上增加事务控制，这样更好

## 8) 修改 UserServiceImpl

```

package tarena.service.impl;

@Service("userService")
public class UserServiceImpl implements UserService {
    //默认采用名称对应规则将Spring容器中dao注入
    @Resource(name="userDao")
    private UserDAO userDao;

```

```

    public UserDao getUserDao() {
        return userDao;
    }

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

    @Transactional(readOnly=true,propagation=Propagation.NOT_SUPPORTED)
    public boolean findLogin(User user) {

        User usr = userDao.findByEmail(user.getEmail());
        if(usr != null){
            if(usr.getPassword().equals(user.getPassword())){
                return true;
            }
        }
        return false;
    }

    @Transactional(propagation=Propagation.REQUIRED)
    public void update(){
        //....
    }
}

```

再回顾下 ssh.xml 文件中，有大批的配置信息，我们可以拆分一下

## 9) 新建 ssh-base.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

```

---

```
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
```

```
<bean id="myDataSource"
class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver">
    </property>
    <property name="url" value="jdbc:mysql:///test"> </property>
    <property name="username" value="root"> </property>
    <property name="password" value="root"> </property>
    <property name="maxActive" value="10"> </property>
    <property name="initialSize" value="2"> </property>
</bean>

<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"> </property>
    <property name="mappingResources">
        <list>
            <value>tarena/mapping/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
        </props>
    </property>
</bean>
</beans>
```

## 10) 新建 ssh-annotation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
    <!-- 启用自动扫描注解 -->
    <context:component-scan base-package="tarena">
    </context:component-scan>

    <!-- 启用事务控制注解 -->
    <bean id="txManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="mySessionFactory">
        </property>
    </bean>
    <tx:annotation-driven transaction-manager="txManager"/>
</beans>
```

## 11) 修改 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
       xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <context-param>
        <param-name>contextConfigLocation</param-name>
```

```
<param-value>classpath:ssh-annotation.xml,classpath:ssh-base.xml
</param-value>
</context-param>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

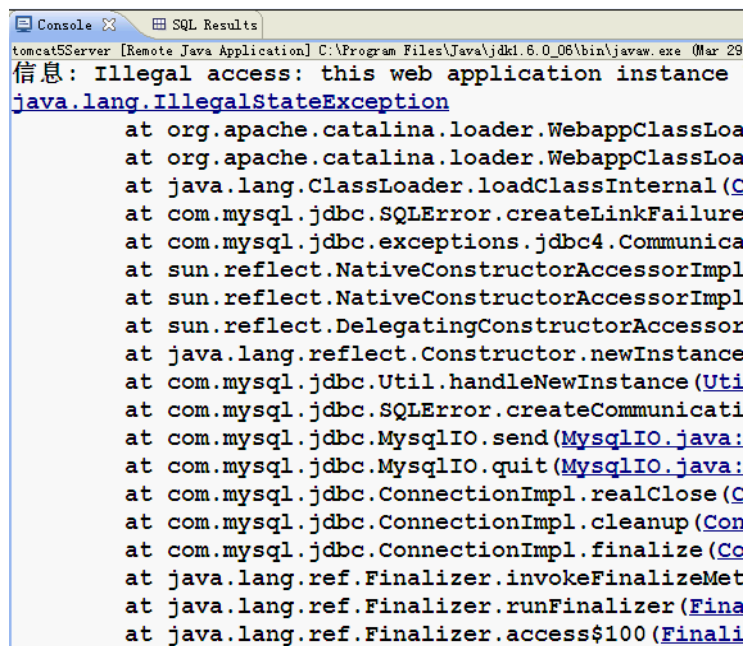
<filter>
  <filter-name>StrutsFilter</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>StrutsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>
```

采用注解方式启用事务配置成功，就可以使用了

## 12) 部署项目

**注意：**部署项目的时候可能遇到这个异常，这是因为 Jar 包的问题



```
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 29
信息: Illegal access: this web application instance
java.lang.IllegalStateException
    at org.apache.catalina.loader.WebappClassLoader
    at org.apache.catalina.loader.WebappClassLoader
    at java.lang.ClassLoader.loadClassInternal(C
    at com.mysql.jdbc.SQLException.createLinkFailure
    at com.mysql.jdbc.exceptions.jdbc4.Communicat
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.NativeConstructorAccessorImpl
    at sun.reflect.DelegatingConstructorAccessor
    at java.lang.reflect.Constructor.newInstance
    at com.mysql.jdbc.Util.handleNewInstance(Uti
    at com.mysql.jdbc.SQLException.createCommunicati
    at com.mysql.jdbc.MysqlIO.send(MysqlIO.java:
    at com.mysql.jdbc.MysqlIO.quit(MysqlIO.java:
    at com.mysql.jdbc.ConnectionImpl.realClose(C
    at com.mysql.jdbc.ConnectionImpl.cleanup(Con
    at com.mysql.jdbc.ConnectionImpl.finalize(Co
    at java.lang.ref.Finalizer.invokeFinalizeMet
    at java.lang.ref.Finalizer.runFinalizer(Fina
    at java.lang.ref.Finalizer.access$100(Finali
```

我们可以这样解决

步骤 1, 将项目中的 mysql 的 Jar 包 mysql-connector-java-5.1.8-bin.jar 删除

步骤 2, 拷贝该 mysql 的 Jar 包到 Tomcat 服务器的 common/lib 目录下

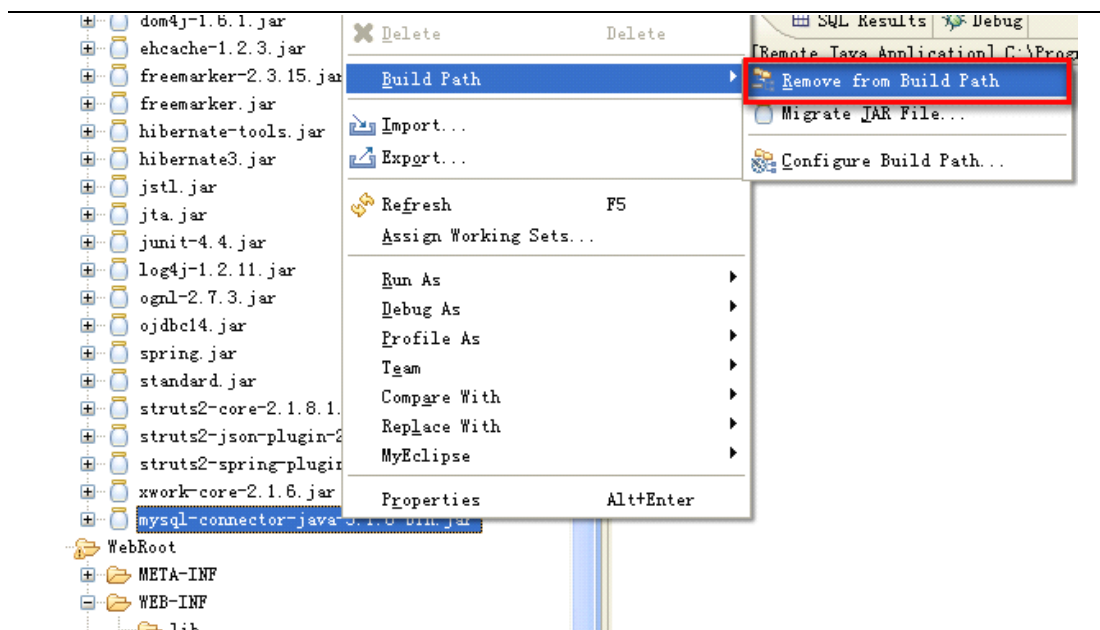
步骤 3, 重启 tomcat 服务器 ( \* )

步骤 4, 删除原先部署在 Tomcat 的项目, 再重新部署 ( \* )

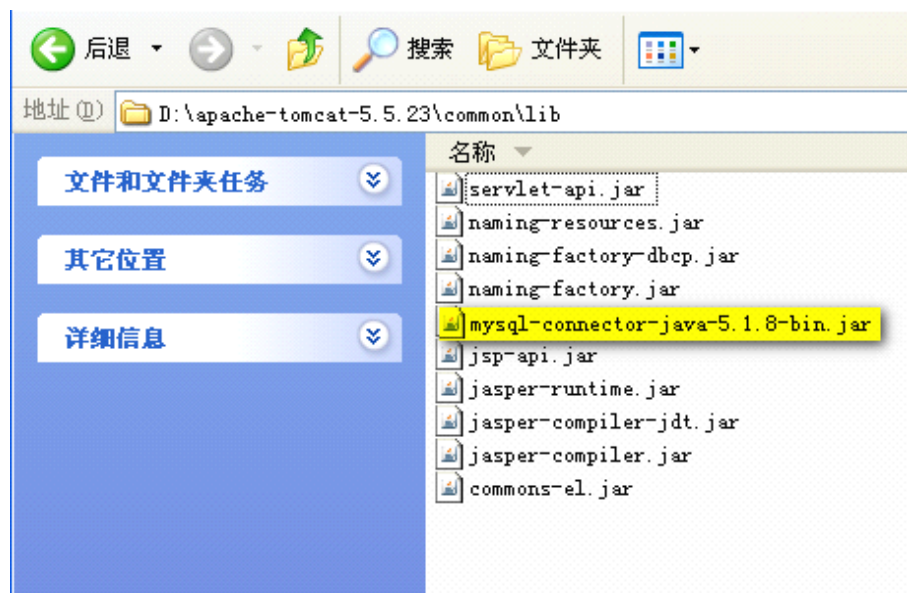
OK

删除项目中的 mysql 的 Jar 包





将 mysql 的 jar 包拷贝到 tomcat 服务器的 common/lib 目录下



**13) 测试 (略)**

**(案例结束)**

---

## 2. Spring 管理事务的策略 \*

### Spring 当中常用事务类型

- ✓ REQUIRED  
支持当前事务，如果当前没有事务，则新建事务，常用
- ✓ SUPPORTS  
支持当前事务，如果当前没有事务，就以非事务方式执行
- ✓ MANDATORY  
支持当前事务，如果当前没有事务，就抛出异常
- ✓ REQUIRES-NEW  
新建一个事务，如果当前存在事务，则将当前事务挂起
- ✓ NOT-SUPPORTED  
以非事务方式执行，如果当前存在事务，则挂起当前事务
- ✓ NEVER  
和 NOT-SUPPORTED 差不多，以非事务方式执行，  
区别在于如果当前存在事务，则抛出异常
- ✓ NESTED  
如果当前存在事务，则嵌套在事务内执行；如果当前没有事务，则执行 REQUIRED 相同操作。

## 3. AOP 动态代理 \*

### 动态代理 (AutoProxy)

采用了 AOP 后，容器返回的对象是代理对象。

用户在使用时，由代理对象调用切面组件和目标对象的功能。

一般采用两种方式

- 1) 如果目标对象有接口，采用 JDK 代理
- 2) 如果目标对象没有接口，采用 CGLIB 代理

### 【案例 2】动态代理 \*

- 1) 新建工程 spring5\_02
- 2) 配置好 spring 环境

---

### 3) 定义组件 UserService

```
package proxy;

public interface UserService {
    public void save();
    public void delete();
}
```

### 4) 新建实现类 UserServiceImpl

```
package proxy;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息！");
    }

    public void save() {
        System.out.println("保存用户信息！");
    }
}
```

### 5) 新建测试类 Test01

```
package proxy;

import org.junit.Test;

public class Test01 {

    @Test
    public void tst1(){
        UserService userService = new UserServiceImpl();
        userService.save();
        userService.delete();
    }
}
```

```
}
```

## 6) 运行 Test01

现在，我们提出需求，我们希望用户在每个操作的时候都记录日志  
该怎么做？

我们这时就可以使用[动态代理机制](#)，通过动态代理 Factory( 相当于 Spring 容器 )来生成代理实例。  
由代理对象来完成该功能。

动态代理 Facotory

## 7) 新建 JDKProxyFactory

```
package proxy;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

public class JDKProxyFactory implements InvocationHandler{

    private Object target;

    //单例模式
    private JDKProxyFactory() {}
    public static JDKProxyFactory getInstance(){
        return new JDKProxyFactory();
    }

    /**
     * @param clazz 目标对象类型
     * @return      代理对象
     * @throws Exception
     */
}
```

```

public Object getProxy(Class clazz) throws Exception{
    //获得目标类型的实例对象
    target = clazz.newInstance();

    //根据目标类型对象创建代理对象
    Object proxy = Proxy.newProxyInstance(
        clazz.getClassLoader(),
        clazz.getInterfaces(), this);
    return proxy;
}

/**
 * 每次通过代理对象调用目标方法时，执行该方法
 * @param arg0
 * @param arg1
 * @param arg2
 * @return
 * @throws Throwable
 */
public Object invoke(Object proxy, Method method, Object[] params)
    throws Throwable {
    Object retVal = null;
    try{
        System.out.println("调用前置通知");
        //调用目标对象处理
        retVal = method.invoke(target, params);
        System.out.println("调用后置通知");
    }catch(Exception e){
        System.out.println("调用异常通知");
    }finally{
        System.out.println("调用最终通知");
    }
    return retVal;
}
}

```

## 8) 修改 Test01

```

package proxy;

```

```

import org.junit.Test;

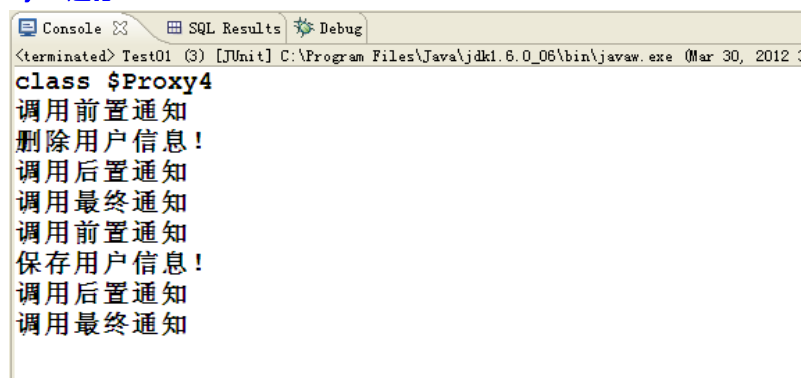
public class Test01 {

    // @Test
    public void tst1(){
        UserService userService = new UserServiceImpl();
        userService.save();
        userService.delete();
    }

    @Test
    public void testJDKProxy() throws Exception {
        JDKProxyFactory proxyFactory = JDKProxyFactory.getInstance();
        UserService userService =
            (UserService)proxyFactory.getProxy(UserServiceImpl.class);
        System.out.println(userService.getClass());
        userService.delete();
        userService.save();
    }
}

```

## 9) 运行 Test01



```

<terminated> Test01 (3) [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 30, 2012)
class $Proxy4
调用前置通知
删除用户信息!
调用后置通知
调用最终通知
调用前置通知
保存用户信息!
调用后置通知
调用最终通知

```

如果没有接口，比如更新部门时，只有 DeptServiceImpl，没有 DeptService 接口，如果我们还使用这种方式就会出错。

---

测试在没有接口情况下，使用 JDK 代理会出错的情况

#### 10) 新建 DeptServiceImpl

```
package proxy;

public class DeptServiceImpl {
    public void update(){
        System.out.println("更新部门信息！");
    }
    public void add(){
        System.out.println("添加部门信息！");
    }
}
```

#### 11) 修改 Test01

```
package proxy;

import org.junit.Test;

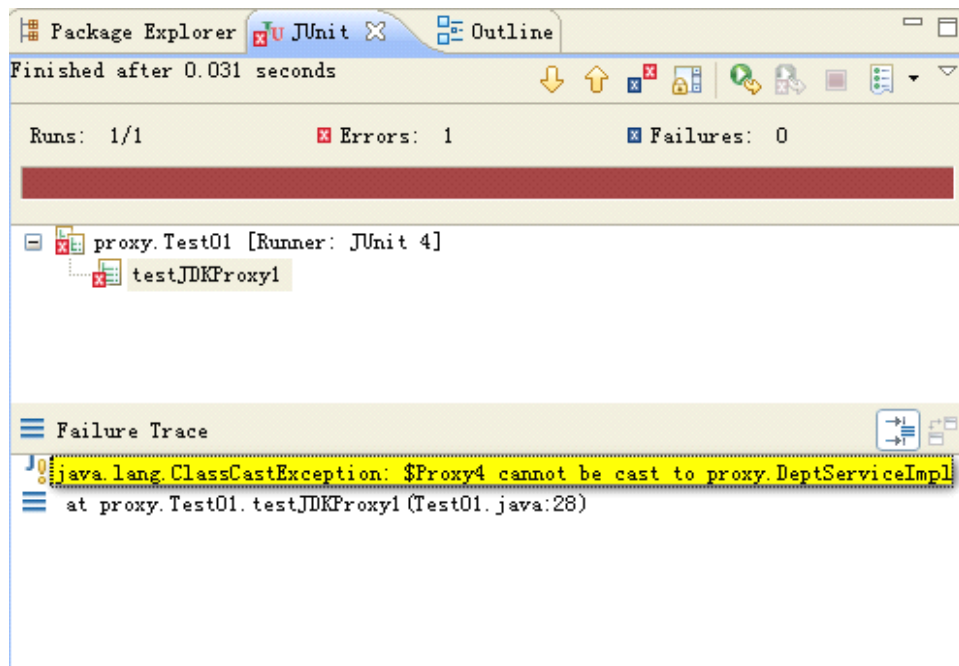
public class Test01 {
    // @Test
    public void testJDKProxy() throws Exception {
        JDKProxyFactory proxyFactory = JDKProxyFactory.getInstance();
        UserService userService =
            (UserService)proxyFactory.getProxy(UserServiceImpl.class);
        System.out.println(userService.getClass());
        userService.delete();
        userService.save();
    }

    @Test
    public void testJDKProxy1() throws Exception{
        JDKProxyFactory proxyFactory = JDKProxyFactory.getInstance();
        DeptServiceImpl deptService =
            (DeptServiceImpl)proxyFactory.getProxy(DeptServiceImpl.class);
        deptService.add();
        deptService.update();
    }
}
```

```
}  
}
```

## 12) 运行 Test01

会提示异常



这种情况下，我们就需要使用 cglib 代理

## 13) 新建 CGLIBProxyFactory

```
package proxy;  
  
import java.lang.reflect.Method;  
import net.sf.cglib.proxy.Enhancer;  
import net.sf.cglib.proxy.MethodInterceptor;  
import net.sf.cglib.proxy.MethodProxy;  
  
public class CGLIBProxyFactory implements MethodInterceptor{  
    private Object target;  
  
    private CGLIBProxyFactory(){  
    }  
    public static CGLIBProxyFactory getInstance(){  
        return new CGLIBProxyFactory();  
    }  
}
```



```

    }

    public Object getProxy(Class clazz) throws Exception{
        target = clazz.newInstance();
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(clazz);
        enhancer.setCallback(this);
        return enhancer.create();
    }

    public Object intercept(Object proxy, Method method,
        Object[] params,
        MethodProxy methodProxy) throws Throwable {

        Object retVal = null;
        try{
            System.out.println("调用前置通知");
            //调用目标对象处理
            retVal = method.invoke(target, params);
            System.out.println("调用后置通知");
        }catch(Exception e){
            System.out.println("调用异常通知");
        }finally{
            System.out.println("调用最终通知");
        }
        return retVal;
    }
}

```

#### 14) 修改 Test01

```

package proxy;

import org.junit.Test;

public class Test01 {

    // @Test

```

```

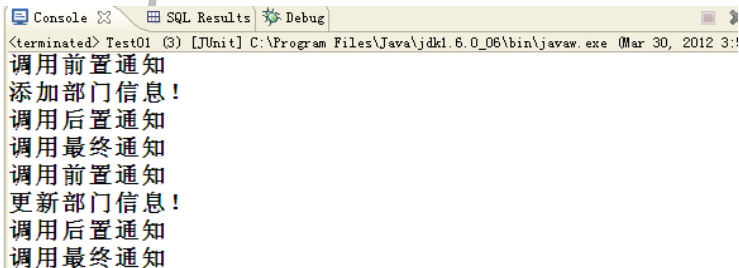
    public void testJDKProxy() throws Exception {
        JDKProxyFactory proxyFactory = JDKProxyFactory.getInstance();
        UserService userService =
            (UserService)proxyFactory.getProxy(UserServiceImpl.class);
        System.out.println(userService.getClass());
        userService.delete();
        userService.save();
    }

//    @Test
    public void testJDKProxy1() throws Exception{
        JDKProxyFactory proxyFactory = JDKProxyFactory.getInstance();
        DeptServiceImpl deptService =
            (DeptServiceImpl)proxyFactory.getProxy(DeptServiceImpl.class);
        deptService.add();
        deptService.update();
    }

    @Test
    public void testGglibProxy() throws Exception{
        CGLIBProxyFactory proxyFactory = CGLIBProxyFactory.getInstance();
        DeptServiceImpl deptService =
            (DeptServiceImpl)proxyFactory.getProxy(DeptServiceImpl.class);
        deptService.add();
        deptService.update();
    }
}

```

## 15) 运行 Test01



```

<terminated> Test01 (3) [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 30, 2012 3:11:11 PM)
调用前置通知
添加部门信息!
调用后置通知
调用最终通知
调用前置通知
更新部门信息!
调用后置通知
调用最终通知

```

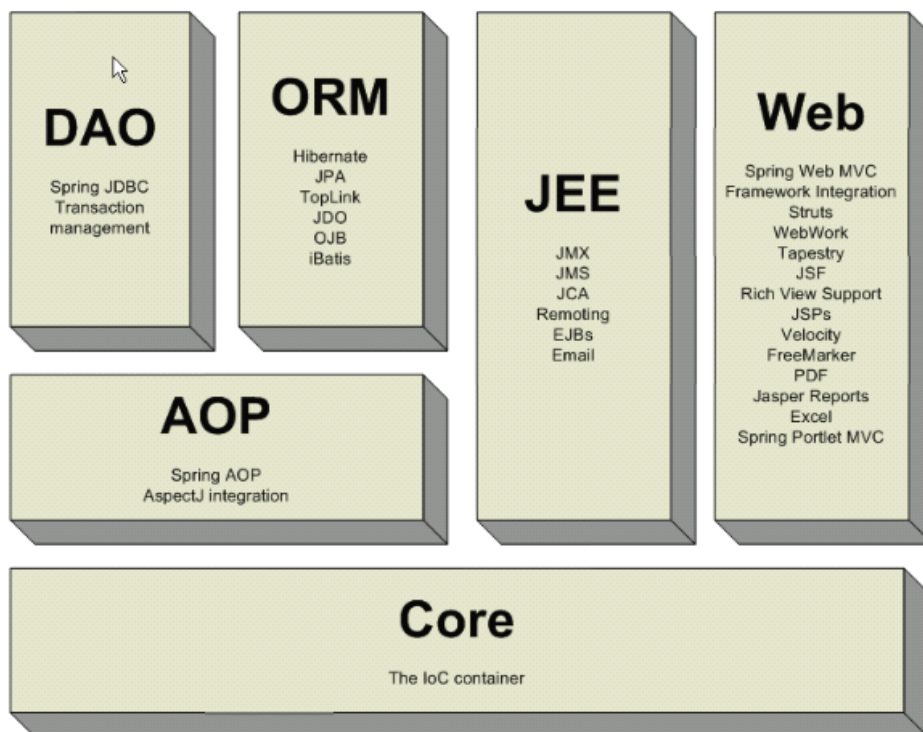
( 案例结束 )

---

## 4. Spring 框架结构 \*

我们在使用学习 Spring 框架时，只是重点介绍了 IoC 和 AOP，这只是 Spring 强大功能的核心，Spring 还可以做很多工作。

Spring 框架整体结构图



Spring 框架概述

Core(The IoC container)和 AOP 只是 Spring 框架中的一部分。

Spring 对 JDBC 的整合，属于 DAO。

如上所示，Spring 的功能共 6 大模块。（请查看 [Spring 帮助手册](#)）

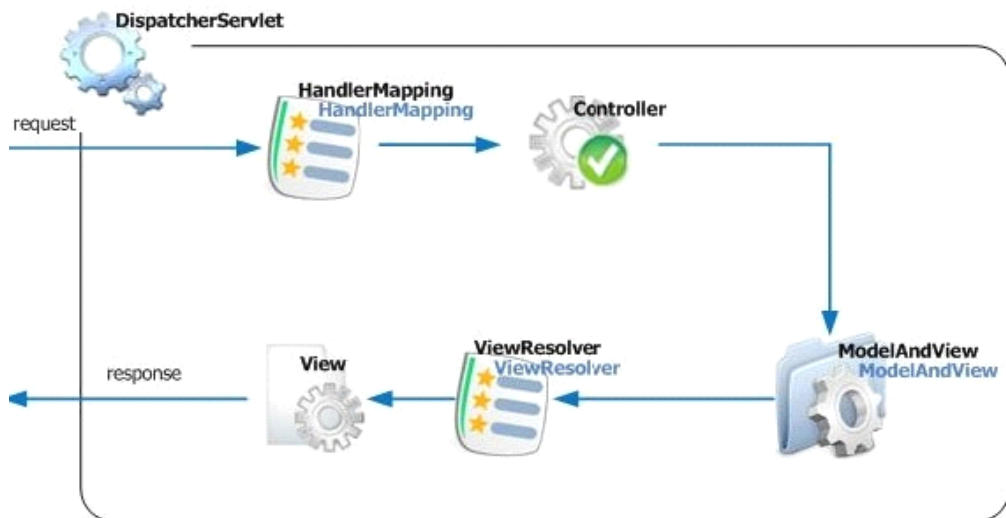
- 1) Core 封装包是 Spring 框架的最基础部分，提供 IoC 和依赖注入。
- 2) DAO 提供了 JDBC 的抽象层。还有注解声明类型的事务管理方法。
- 3) ORM 该封装包提供了常用的“对象/关系” APIs。  
我们频繁使用的是 Hibernate，除此之外还可以整合 JPA、JDO、iBatis 等。
- 4) AOP 提供了符合 AOP 规范的面向切面的编程实现

- 
- 5) Web 该封装包提供了基础的针对 web 开发的集成特性。  
此外 Spring 也提供了一种 MVC 实现。
  - 6) J2EE 一般使用较少。提供了 JMS、Email 等集成。Spring 可以将发送邮件等功能整合。

## 5. Spring MVC \*

Spring 的作用是整合，但不仅仅限于整合，Spring 框架可以被看做是一个[企业解决方案级别的框架](#)。

### Spring MVC 流程图



1. 客户端发送请求
2. 客户端发送的请求到达服务器控制器  
服务器控制器由 Servlet ( DispatcherServlet ) 实现的，来完成请求的转发
3. 该控制器 ( DispatcherServlet ) 调用了一个用于映射的类 HandlerMapping ，  
该类用于将请求映射到对应的处理器来处理请求。
4. HandlerMapping 将请求映射到对应的处理器 Controller ( 相当于 Action )  
在 Spring 当中如果写一些处理器组件，一般实现 Controller 接口
5. 在 Controller 中就可以调用一些 Service 或 DAO 来进行数据操作
6. ModelAndView 用于存放从 DAO 中取出的数据，还可以存放响应视图的一些数据。
7. 如果想将处理结果返回给用户，那么在 Spring 框架中还提供一个视图组件 ViewResolver ，  
该组件根据 Controller 返回的标示，找到对应的视图，将响应 response 返回给用户

---

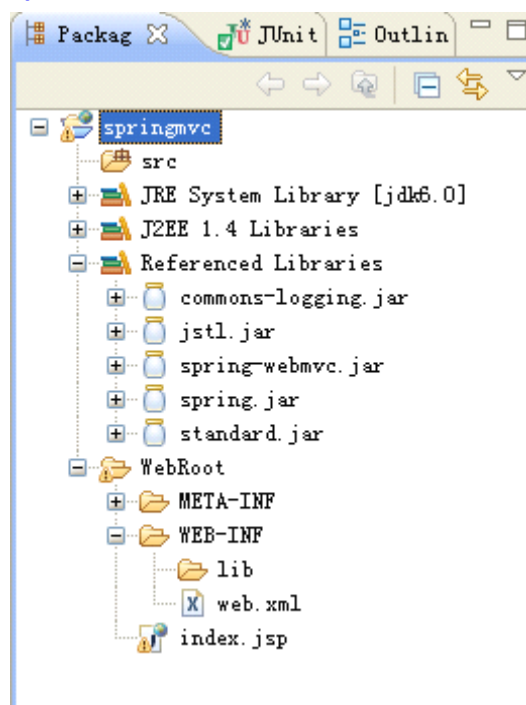
如上所示是 Spring 处理 request--response 流程中的几个重要组件。

如何学习框架技术？

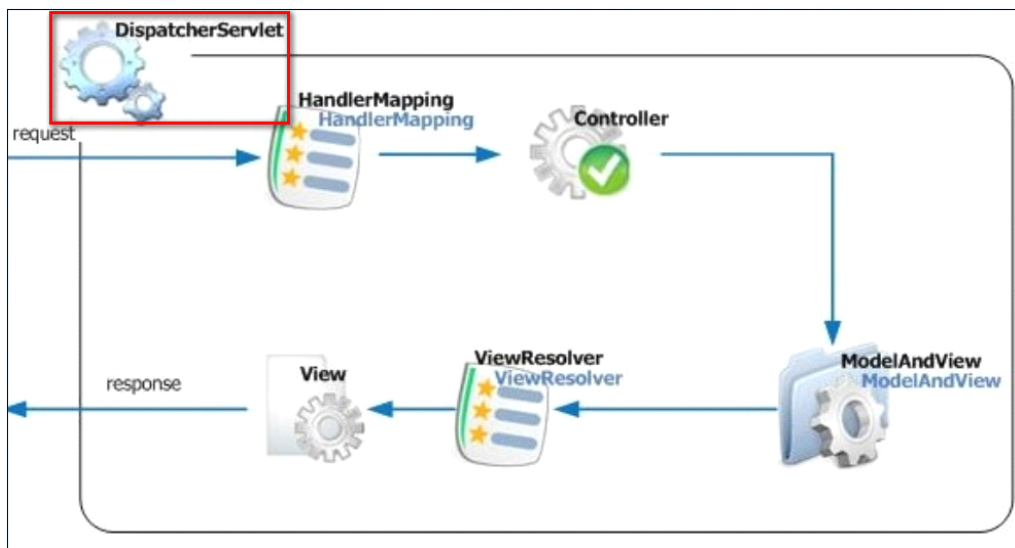
就像如上的图示一样，先掌握新技术的[体系流程图](#)。在快速弄明白程序执行流程后，在使用过程中了解细节。

### 【案例 3】Spring MVC \*\*

- 1) 新建工程 springmvc
- 2) 导入 Jar 包



首先配置 [DispatcherServlet](#)



### 3) web.xml

配置中央控制器 DispatcherServlet

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>SpringMVC</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <!-- 指定配置文件的位置 -->
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <!-- 服务器启动即加载 -->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>

```

```
<servlet-name>SpringMVC</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>

</web-app>
```

#### 4) 新建 spring-mvc.xml

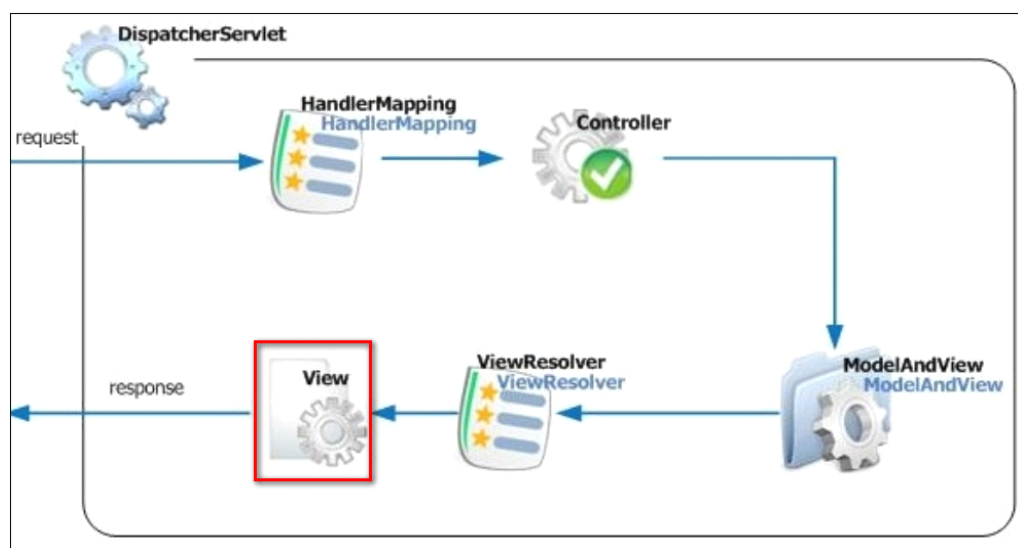
和之前的 Spring 配置文件头相同

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

  <!-- 待添加 -->

</beans>
```

写视图 **View** ( 严格的说只是 jsp 页面 )



## 5) 新建视图页面 login.jsp

需要注意的是，没有添加 Struts2 框架，  
就不能这样调用了 `user.email`，  
同时 action 发往 `login.do`，而不是 `login.action`

```
<%@ page contentType="text/html; charset=utf-8"
    isELIgnored="false" pageEncoding="utf-8"%>
<html>
<head>
<title>Insert title here</title>
</head>
<body style="font-size:30px;">
    <form action="login.do" method="post">
        用户名：<input type="text" name="email"><br/>
        密码：<input type="text" name="password"><br/>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

## 6) 新建视图页面 ok.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" %>
<html>
<head>
```

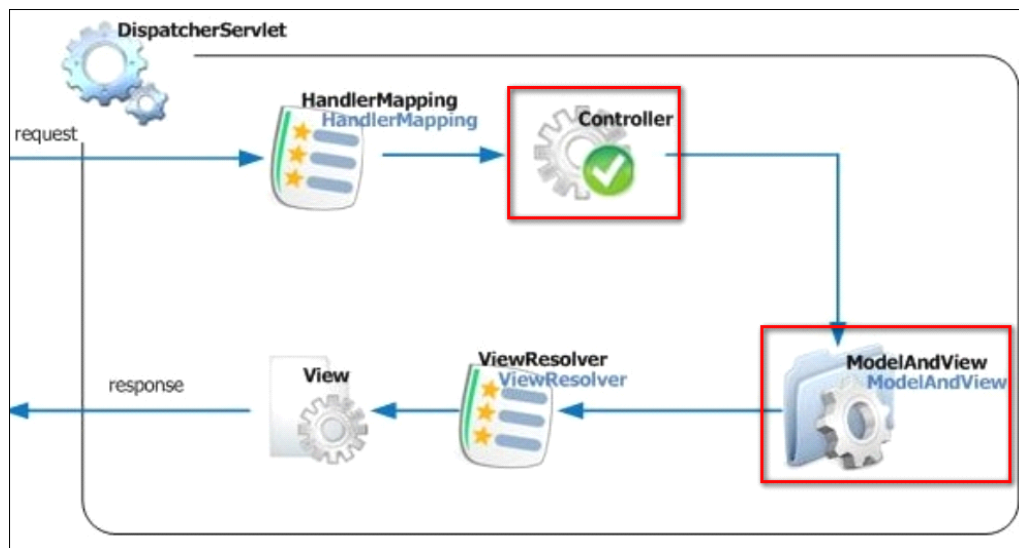


```

<title>登录成功页面</title>
</head>
<body>
  <h2>登录成功！
</body>
</html>

```

接下来新建 **Controller**，返回值是 **ModelAndView** 类型



## 7) LoginControl

LoginControl 等价于我们之前写的 LoginAction，handleRequest() 相当于 execute() 方法

```

package tarena.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class LoginControl implements Controller {

    @Override
    public ModelAndView handleRequest(
        HttpServletRequest arg0,
        HttpServletResponse arg1) throws Exception {

```

```

String email = arg0.getParameter("email");
String password = arg0.getParameter("password");

if("scott@163.com".equals(email)
    && "1111".equals(password)){

    return new ModelAndView("ok");
}

return new ModelAndView("login");
}
}

```

在 Spring 配置文件中，注入 bean ( loginControl )

## 8) spring-mvc.xml

```

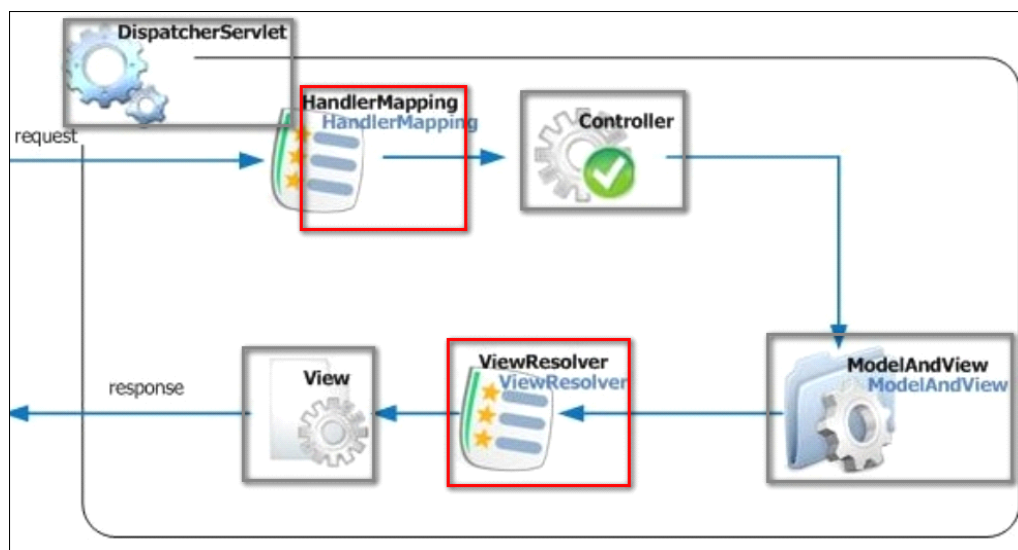
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="loginControl" class="tarena.control.LoginControl"> </bean>

</beans>

```

那么，从如上步骤中，我们已经完成了 Spring MVC 控制流程中的 [DispatcherServlet](#)、[Controller](#)、[ModelAndView](#)、[View](#)，  
还有 [HandlerMapping](#)、[ViewResolver](#)，它们不需要自己写，需要配置



配置 HandlerMapping、ViewResolver

## 9) 修改 spring-mvc.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="loginControl" class="tarena.control.LoginControl">
        <property name="commandClass" value="tarena.control.User">
        </property>
    </bean>

    <!-- 配置HandlerMapping组件, 用于实现请求与处理器之间的映射 -->

```

```

<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="login.do">loginControl</prop>
        </props>
    </property>
</bean>

<!-- 定义ViewResolver组件, 实现根据视图标识获取JSP响应 -->
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- View中用到的相关技术 -->
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"> </property>
    <!-- 后缀 -->
    <property name="suffix" value=".jsp"> </property>
    <!-- 前缀 -->
    <property name="prefix" value="/"> </property>
</bean>

</beans>

```

如此设置“前缀”和“后缀”表明，

ViewResolver 对象将到 WEB-INF/jsp 目录下找\*.jsp 文件

```

22 <bean id="viewResolver" class="org.springframework.w
23     <property name="viewClass" value="org.springfram
24     <property name="suffix" value=".jsp"> </property>
25     <property name="prefix" value="/WEB-INF/jsp"> <
26 </bean>
27

```

jsp 文件的名字，是根据我们在 LoginControl 中的具体业务返回的字符串决定  
“success” 或者 “login”

```

@Override
public ModelAndView handleRequest(
    HttpServletRequest arg0,
    HttpServletResponse arg1) throws Exception {

    if("scott@163.com".equals(email)
        && "1111".equals(password)){

        return new ModelAndView("success" );
    }

    return new ModelAndView("login" );
}

```

需要注意的是

[HandleMapping 组件](#)有很多的实现技术，我们使用的是较为简单的  
[org.springframework.web.servlet.handler.SimpleUrlHandlerMapping](#)

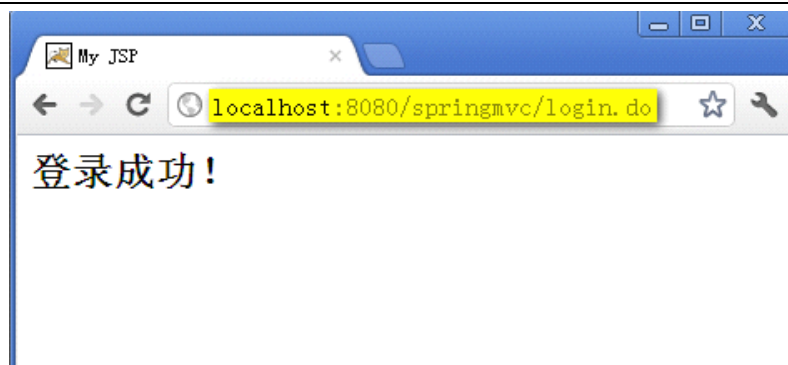
[ViewResolver 组件](#)也有很多实现技术，我们使用  
[org.springframework.web.servlet.view.InternalResourceViewResolver](#)  
 此外，在 [ViewResolver 组件](#)中使用的主要解析技术是 [JstlView](#)  
 （以预防&&配合页面使用 jstl 标签的情况）  
[org.springframework.web.servlet.view.JstlView](#)

#### 10) 部署项目

#### 11) 访问 <http://localhost:8080/springmvc/login.jsp>

#### 12) 输入 “scott@163.com” “1111”





测试成功

如果有需要将数据传入下一个视图的需求的话，我们这样写  
借助于 [ModelMap](#)，我们就可以传参数

### 13) 修改 LoginControl

```
package tarena.control;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.ui.ModelMap;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class LoginControl implements Controller {

    @Override
    public ModelAndView handleRequest(
        HttpServletRequest arg0,
        HttpServletResponse arg1) throws Exception {

        String email = arg0.getParameter("email");
        String password = arg0.getParameter("password");

        ModelMap map = new ModelMap();
        if("scott@163.com".equals(email)
            && "1111".equals(password)){
            map.put("msg", email);
        }
    }
}
```

```

        return new ModelAndView("ok", map);
    }
    map.put("msg", "用户名或密码错误！");
    return new ModelAndView("login", map);
}
}

```

#### 14) 修改 login.jsp

```

<%@ page contentType="text/html; charset=utf-8"
    isELIgnored="false" pageEncoding="utf-8"%>
<html>
<head>
<title>Insert title here</title>
</head>
<body style="font-size:30px;">
    <!--错误提示信息-->
    ${msg}
    <form action="login.do" method="post">
        用户名 : <input type="text" name="email"> <br/>
        密码 : <input type="text" name="password"> <br/>
        <input type="submit" value="登录">
    </form>
</body>
</html>

```

#### 15) 修改 ok.jsp

```

<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" %>
<html>
<head>
    <title>My JSP </title>
</head>

<body>
    <h2>登录成功 ! <%=request.getAttribute("msg") %> </h2> <br>
</body>
</html>

```

## 16) Test

访问 <http://localhost:8080/springmvc/login.jsp>

输入错误信息会提示



输入正确信息，跳转成功页面



有一点还需要了解，默认是采用 dispatcher 方式跳转，  
如果出错了，页面跳转到 login.do

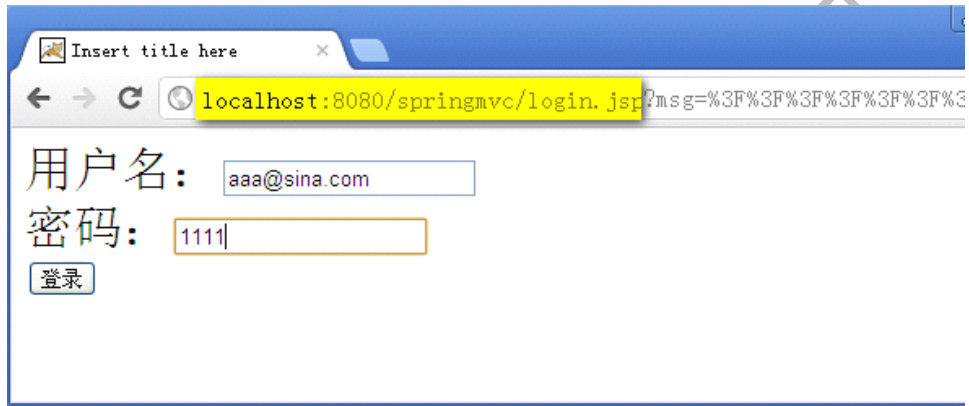


我们可以这样设置为 [redirect 方式](#)



```
18 ModelMap map = new ModelMap();
19 if("scott@163.com".equals(email) && "1111".equals(pa
20     map.put("msg", email);
21     return new ModelAndView("ok",map);
22 }
23 map.put("msg", "用户名或密码错误！");
24 return new ModelAndView("redirect:/login.jsp",map);
25
```

登录不成功时，会跳转到 login.jsp 页面，并且带着 msg 消息作为参数



**( 案例结束 )**

## 6. ssh2 重构当注意事项 \*\*

- 1) 主要工作
  - ✓ 将 Action 逻辑提到 Service 组件中
  - ✓ 重构 DAO 组件
  - ✓ ssh2 框架搭建及相关配置
  - ✓ 使用 Spring 控制事务
- 2) 注意事项
  - ✓ JSON 响应问题
    - 避免 Action 组件中的 Service 参与 JSON 响应
    - 可以在 getXXX 方法前加@JSON(serialize=false)，也可以使用 xml 配置。
  - ✓ CartService 购物车组件要使用 scope="session"，需要在 web.xml 中添加 RequestContextHolder 配置

---

```
<listener>
  <listener-class>
    org.springframework.web.context.request.RequestContextListener
  </listener-class>
</listener>
```