

Tarena Teaching System

# Spring 核心

## 学员用书



Java企业应用及互联网  
高级工程师培训课程

# 目 录

<b>Unit01</b> .....	1
1. Spring IOC 容器 .....	3
1.1. Spring 框架简介 .....	3
1.1.1. 什么是 Spring .....	3
1.1.2. Spring 的主要功能 .....	3
1.2. 容器和 Bean 管理 .....	3
1.2.1. Spring 容器简介 .....	3
1.2.2. Spring 容器的实例化 .....	4
1.2.3. Spring 容器的使用 .....	4
1.2.4. Bean 的实例化 .....	4
1.2.5. Bean 的命名 .....	6
1.2.6. Bean 的作用域 .....	6
1.2.7. Bean 的生命周期回调 .....	6
1.2.8. Bean 延迟实例化 .....	7
1.2.9. 指定 bean 依赖关系 .....	7
1.3. 容器的 IOC 应用 .....	8
1.3.1. IOC 概念 .....	8
1.3.2. Setter 注入 .....	8
1.3.3. 构造器注入 .....	9
1.3.4. 自动装配 .....	10
经典案例 .....	11
1. 实例化 Spring 容器示例 .....	11
2. 利用 Spring 容器创建 JavaBean 对象 .....	14
3. 如何控制 Bean 实例化 .....	18
4. 利用 Spring 实现 bean 属性 setter 方式注入 .....	28
5. 利用构造器参数实现依赖属性的注入 .....	34
6. 利用 Spring 的自动装配功能实现自动属性注入 .....	45
课后作业 .....	51
<b>Unit02</b> .....	52
1. Spring IOC 容器 .....	53
1.1. 参数值注入 .....	53
1.1.1. 注入基本值 .....	53
1.1.2. 注入 Bean 对象 .....	53
1.1.3. 注入集合 .....	54
1.1.4. 注入 Spring 表达式值 .....	56

---

1.1.5. 注入 null 或空字符串 .....	56
1.2. 基于注解的组件扫描 .....	57
1.2.1. 什么是组件扫描 .....	57
1.2.2. 指定扫描类路径 .....	57
1.2.3. 自动扫描的注解标记 .....	57
1.2.4. 自动扫描组件的命名 .....	58
1.2.5. 指定组件的作用域 .....	58
1.2.6. 初始化和销毁回调的控制 .....	58
1.2.7. 指定依赖注入关系 .....	59
1.2.8. 注入 Spring 表达式值 .....	61
经典案例 .....	62
1. 给 MessageBean 注入参数值 .....	62
2. 测试 Spring 自动组件扫描方式 .....	75
3. 如何控制 ExampleBean 实例化方式 .....	80
4. 使用注解方式重构 JdbcDataSource, UserDao, UserService .....	84
课后作业 .....	114
<b>Unit03</b> .....	115
1. Spring Web MVC 基础 .....	116
1.1. Spring Web MVC 简介 .....	116
1.1.1. MVC 模式简介 .....	116
1.1.2. 什么是 Spring Web MVC .....	116
1.1.3. Spring Web MVC 的核心组件 .....	117
1.1.4. Spring Web MVC 的处理流程 .....	117
1.2. 基于 XML 配置的 MVC 应用 .....	118
1.2.1. 搭建 Spring Web MVC 环境 .....	118
1.2.2. DispatcherServlet 控制器配置 .....	118
1.2.3. HandlerMapping 组件 .....	118
1.2.4. Controller 组件 .....	119
1.2.5. ModelAndView 组件 .....	120
1.2.6. ViewResolver 组件 .....	120
1.3. 基于注解配置的 MVC 应用 .....	121
1.3.1. RequestMapping 注解应用 .....	121
1.3.2. Controller 注解应用 .....	122
经典案例 .....	123
1. 构建一个 helloworld 应用案例 .....	123
2. 重构 helloworld 应用案例 .....	134
课后作业 .....	139
<b>Unit04</b> .....	140

---

1. Spring Web MVC 实战 .....	141
1.1. 基于注解配置的 MVC 应用--续.....	141
1.1.1. 接收请求参数值 .....	141
1.1.2. 向页面传值 .....	142
1.1.3. Session 存储 .....	144
1.1.4. 重定向视图 .....	144
1.2. 实战技巧 .....	145
1.2.1. 中文乱码解决方案 .....	145
1.2.2. 使用拦截器实现登录检查 .....	146
1.2.3. 异常处理.....	148
1.3. 文件上传 .....	150
1.3.1. SpringMVC 文件上传简介.....	150
1.3.2. CommonsMultipartResolver 组件 .....	150
1.3.3. 视图表单实现 .....	150
1.3.4. Controller 实现 .....	151
1.3.5. 限制上传文件大小 .....	151
经典案例 .....	153
1. 登录案例.....	153
2. 为登录案例解决中文乱码问题 .....	181
3. SomeInterceptor 拦截器入门示例.....	182
4. 为 Hello 示例添加登录检查拦截器 .....	186
5. 为案例添加异常处理 .....	192
6. 文件上传案例 .....	206
7. 限制上传文件大小案例.....	218
课后作业 .....	223



# Spring 核心

## Unit01

知识体系.....**Page 3**

Spring IOC 容器	Spring 框架简介	什么是 Spring
		Spring 的主要功能
	容器和 Bean 管理	Spring 容器简介
		Spring 容器的实例化
		Spring 容器的使用
		Bean 的实例化
		Bean 的命名
		Bean 的作用域
		Bean 的生命周期回调
		Bean 延迟实例化
		指定 bean 依赖关系
	容器的 IOC 应用	IOC 概念
		Setter 注入
		构造器注入
		自动装配

经典案例.....**Page 11**

实例化 Spring 容器示例	Spring 容器的实例化
利用 Spring 容器创建 JavaBean 对象	Spring 容器的使用
	Bean 的实例化
	Bean 的命名
如何控制 Bean 实例化	Bean 的作用域
	Bean 的生命周期回调
	Bean 延迟实例化
	指定 Bean 依赖关系
利用 Spring 实现 bean 属性 setter 方式注入	Setter 注入
利用构造器参数实现依赖属性的注入	构造器注入
利用 Spring 的自动装配功能实现自动属性注入	自动装配



## 1. Spring IOC 容器

### 1.1. Spring 框架简介

#### 1.1.1. 【Spring 框架简介】什么是 Spring

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

**什么是Spring**

**知识讲解**

**为什么要用Spring?**

Spring的本质是管理软件中的对象,如何**创建对象**和**维护对象之间的关系**

#### 1.1.2. 【Spring 框架简介】Spring 的主要功能

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

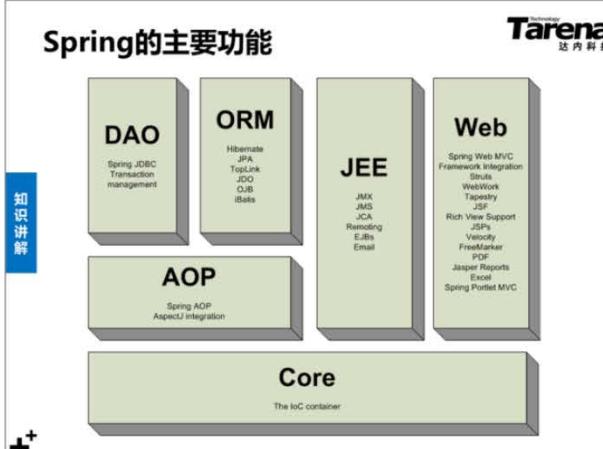
\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

**Spring的主要功能**

**知识讲解**



The diagram illustrates the main components of the Spring framework:

- DAO**: Spring JDBC, Transaction management
- ORM**: Hibernate, JPA, TopLink, JDO, OJB, iBatis
- JEE**: JMS, JNDI, JCA, Remoting, EJBs, Email
- Web**: Spring Web MVC, Framework integration, Struts, WebWork, Tapestry, JSF, Rich View Support, JSPs, Velocity, Freemarker, PDF, Jasper Reports, Excel, Spring Portlet MVC
- AOP**: Spring AOP, AspectJ integration
- Core**: The IoC container

### 1.2. 容器和 Bean 管理

#### 1.2.1. 【容器和 Bean 管理】Spring 容器简介

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

**Spring容器简介**

**知识讲解**

• 在Spring中,任何的Java类和JavaBean都被当成Bean处理,这些Bean通过容器管理和应用。

• Spring容器实现了IOC和AOP机制,这些机制可以简化Bean对象创建和Bean对象之间的解耦;

• Spring容器有BeanFactory和ApplicationContext两种类型;

**什么是JavaBean: 一种简单规范的Java对象**

**何时使用Spring?**

当需要管理JavaBean对象时候就可以使用,  
Spring是最简洁的对象管理方案之一.

如何**创建对象** 如何**管理对象之间关系**

### 1.2.2. 【容器和 Bean 管理】Spring 容器的实例化

知识讲解

Spring容器的实例化 如何创建对象

• ApplicationContext继承自BeanFactory接口，拥有更多的企业级方法，推荐使用该类型，实例化方法如下：

```
//加载文件系统中的配置文件实例化
String conf = "C:\applicationContext.xml";
ApplicationContext ac =
    new FileSystemXmlApplicationContext(conf);

//加载工程classpath下的配置文件实例化
String conf = "applicationContext.xml";
ApplicationContext ac =
    new ClassPathXmlApplicationContext(conf);
```

如何使用Spring? 遵守Spring定义的规则, 基于配置和默认规则, 减少了代码的书写

+

### 1.2.3. 【容器和 Bean 管理】Spring 容器的使用

知识讲解

Spring容器的使用

• 从本质上讲，BeanFactory和ApplicationContext仅仅只是一个维护Bean定义以及相互依赖关系的高级工厂接口。通过BeanFactory和ApplicationContext我们可以访问bean定义。

• 首先在容器配置文件applicationContext.xml中添加Bean定义

```
<bean id="标识符" class="Bean类型"/>
```

• 然后在创建BeanFactory和ApplicationContext容器对象后，调用getBean()方法获取Bean的实例即可

```
getBean("标识符")
```

+

### 1.2.4. 【容器和 Bean 管理】Bean 的实例化

知识讲解

Bean的实例化

• Spring容器创建Bean对象的方法有以下3种

- 用构造器来实例化
- 使用静态工厂方法实例化
- 使用实例工厂方法实例化

将对象创建规则告诉Spring, Spring会帮你去创建对象;  
基于配置和默认规则, 减少了代码的书写!

+

知识讲解

### Bean的实例化 ( 续1 )

• 用构造器来实例化

```
<bean id="calendarObj1"
      class="java.util.GregorianCalendar"/>
```

```
<bean name="calendarObj2"
      class="java.util.GregorianCalendar "/>
```

id或name属性用于指定Bean名称, 用于从Spring中查找这个 Bean对象  
class用于指定Bean类型, 会自动调用无参数构造器创建对象

+

知识讲解

### Bean的实例化 ( 续2 )

• 使用静态工厂方法实例化

```
<bean id="calendarObj2" class="java.util.Calendar"
      factory-method="getInstance"/>
```

id属性用于指定Bean名称 ;  
class属性用于指定工厂类型 ;  
factory-method属性用于指定工厂中创建Bean对象的方法 , 必须用static修饰的方法。

+

知识讲解

### Bean的实例化 ( 续3 )

• 使用实例工厂方法实例化

```
<bean id="calendarObj3" class="java.util.GregorianCalendar"/>
```

```
<bean id="dateObj" factory-bean="calendarObj3"
      factory-method="getTime"/>
```

id用于指定Bean名称 ;  
factory-bean属性用于指定工厂Bean对象 ;  
factory-method属性用于指定工厂中创建Bean对象的方法。

+

### 1.2.5. 【容器和 Bean 管理】Bean 的命名

_____ _____ _____ _____ _____ _____	<p><b>Bean的命名</b></p> <p><b>知识讲解</b></p> <ul style="list-style-type: none"> <li>• Bean的名称             <ul style="list-style-type: none"> <li>- 在Spring容器中，每个Bean都需要有名字（即标识符），该名字可以用&lt;bean&gt;元素的id或name属性指定</li> <li>- id属性比name严格，要求具有唯一性，不允许用“/”等特殊字符</li> </ul> </li> <li>• Bean的别名             <p>为已定义好的Bean，再增加另外一个名字引用，可以使用&lt;alias&gt;指定</p> <pre>&lt;alias name="fromName" alias="toName"/&gt;</pre> </li> </ul> <p style="text-align: center;">+</p>
--	--

### 1.2.6. 【容器和 Bean 管理】Bean 的作用域

_____ _____ _____ _____ _____ _____	<p><b>Bean的作用域</b></p> <p><b>知识讲解</b></p> <ul style="list-style-type: none"> <li>• Spring容器在实例化Bean时，可以创建以下作用域的Bean对象</li> </ul> <table border="1" style="margin-top: 10px; width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">作用域</th> <th style="text-align: left; padding: 2px;">描述</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">singleton</td> <td style="padding: 2px;">在每个Spring IoC容器中一个bean定义对应一个对象实例，默认项</td> </tr> <tr> <td style="padding: 2px;">prototype</td> <td style="padding: 2px;">一个bean定义对应多个对象实例</td> </tr> <tr> <td style="padding: 2px;">request</td> <td style="padding: 2px;">在一次HTTP请求中，一个bean定义对应一个实例，仅限于Web环境</td> </tr> <tr> <td style="padding: 2px;">session</td> <td style="padding: 2px;">在一个HTTP Session中，一个bean定义对应一个实例，仅限于Web环境</td> </tr> <tr> <td style="padding: 2px;">global Session</td> <td style="padding: 2px;">在一个全局的HTTP Session中，一个bean定义对应一个实例；仅在基于portlet的Web应用中才有意义，Portlet规范定义了全局Session的概念</td> </tr> </tbody> </table> <p style="text-align: center;">上面的Bean作用域，可以通过&lt;bean&gt;定义的scope属性指定</p> <p style="text-align: center;">+</p>	作用域	描述	singleton	在每个Spring IoC容器中一个bean定义对应一个对象实例，默认项	prototype	一个bean定义对应多个对象实例	request	在一次HTTP请求中，一个bean定义对应一个实例，仅限于Web环境	session	在一个HTTP Session中，一个bean定义对应一个实例，仅限于Web环境	global Session	在一个全局的HTTP Session中，一个bean定义对应一个实例；仅在基于portlet的Web应用中才有意义，Portlet规范定义了全局Session的概念
作用域	描述												
singleton	在每个Spring IoC容器中一个bean定义对应一个对象实例，默认项												
prototype	一个bean定义对应多个对象实例												
request	在一次HTTP请求中，一个bean定义对应一个实例，仅限于Web环境												
session	在一个HTTP Session中，一个bean定义对应一个实例，仅限于Web环境												
global Session	在一个全局的HTTP Session中，一个bean定义对应一个实例；仅在基于portlet的Web应用中才有意义，Portlet规范定义了全局Session的概念												

### 1.2.7. 【容器和 Bean 管理】Bean 的生命周期回调

_____ _____ _____ _____ _____ _____	<p><b>Bean的生命周期回调</b></p> <p><b>知识讲解</b></p> <ul style="list-style-type: none"> <li>• 指定初始化回调方法             <pre>&lt;bean id="exampleBean" class="com.foo.ExampleBean"       init-method= "init" &gt;     &lt;/bean&gt;</pre> </li> <li>• 指定销毁回调方法，仅适用于singleton模式的bean             <pre>&lt;bean id="exampleBean" class="com.foo.ExampleBean"       destroy-method= "destroy" &gt;     &lt;/bean&gt;</pre> </li> </ul> <p style="text-align: center;">提示：指定销毁回调方法，仅适用于singleton模式的bean</p> <p style="text-align: center;"><b>Spring会管理对象的创建过程</b></p> <p style="text-align: center;">+</p>
--	---

知识讲解

+

**Bean的生命周期回调 ( 续1 )**

• 在顶级的<beans/>元素中的default-init-method属性，可以为容器所有<bean>指定初始化回调方法

```
<beans default-init-method="init">
  <bean id="exampleBean" class="com.foo.ExampleBean" />
</beans>
```

• 在顶级的<beans/>元素中的default-destroy-method属性，可以为容器所有<bean>指定销毁回调方法

```
<beans default-destroy-method="destroy">
  <bean id="exampleBean" class="com.foo.ExampleBean" />
</beans>
```

+

### 1.2.8. 【容器和 Bean 管理】Bean 延迟实例化

知识讲解

+

**Bean延迟实例化**

• 在ApplicationContext实现的默认行为就是在启动时将所有singleton bean提前进行实例化

• 如果不想让一个singleton bean在ApplicationContext初始化时被提前实例化，可以使用<bean>元素的lazy-init="true"属性改变

• 一个延迟初始化bean将在第一次被用到时实例化

```
<bean id="exampleBean"
  lazy-init="true"
  class="com.foo.ExampleBean" />
```

• 在顶级的<beans/>元素中的default-lazy-init属性，可以为容器所有<bean>指定延迟实例化特性

+

### 1.2.9. 【容器和 Bean 管理】指定 bean 依赖关系

知识讲解

+

**指定bean依赖关系**

• 当一个bean对另一个bean存在依赖关系时，可以利用<bean>元素的depends-on属性指定

```
<bean id="beanOne" class="ExampleBean"
  depends-on="manager"/>
<bean id="manager" class="ManagerBean" />
```

• 当一个bean对多个bean存在依赖关系时，depends-on属性可以指定多个bean名，用逗号隔开

```
<bean id="beanOne" class="ExampleBean"
  depends-on="manager1, manager2"/>
```

+

### 1.3. 容器的 IOC 应用

#### 1.3.1. 【容器的 IOC 应用】 IOC 概念

**IOC概念**

**Tarena**  
达内科技

**知识讲解**

- IoC全称是Inversion of Control，被译为控制反转；
- IoC是指程序中对象的获取方式发生反转，由最初的new方式创建，转变为由第三方框架创建、注入。第三方框架一般是通过配置方式指定注入哪一个具体实现，从而降低了对象之间的耦合度
- IOC按实现方法不同，可以分为依赖注入DI和依赖查找两种
- Spring容器是采用DI方式实现了IOC控制，IOC是Spring框架的基础和核心；

+

**IOC概念（续1）**

**Tarena**  
达内科技

**知识讲解**

- DI全称是Dependency Injection，被译为依赖注入；
- DI的基本原理就是将一起工作具有关系的对象，通过构造方法参数或方法参数传入建立关联，因此容器的工作就是创建bean时注入那些依赖关系。
- IOC是一种思想，而DI是实现IOC的主要技术途径
- DI主要有两种注入方式，即 Setter注入和构造器注入

+

#### 1.3.2. 【容器的 IOC 应用】 Setter 注入

**Setter注入**

**Tarena**  
达内科技

**知识讲解**

- 通过调用无参构造器或无参static工厂方法实例化bean之后，调用该bean的setter方法，即可实现setter方式的注入。

```
public class JDBCDataSource {  
    private String driver;  
    public void setDriver(String driver) {  
        try{//注册数据库驱动  
            Class.forName(driver);  
            this.driver = driver;  
        }catch(Exception e){  
            throw new RuntimeException(e);  
        }  
    }  
    //其他代码... ...  
}
```

+

## Setter注入 (续1)

- 在容器xml配置中，配置注入参数。

```
<bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
<property name="driver" value="oracle.jdbc.OracleDriver"/>
<property name="url"
  value="jdbc:oracle:thin:@localhost:1521:XE"/>
<property name="user" value="openlab"/>
<property name="pwd" value="open123"/>
</bean>
```

知识讲解



+

### 1.3.3. 【容器的 IOC 应用】构造器注入

## 构造器注入

- 基于构造器的注入是通过调用带参数的构造器来实现的，容器在bean被实例化的时候，根据参数类型执行相应的构造器。

```
public class OracleUserDAO implements UserDAO{
    private JDBCDataSource dataSource;
    public OracleUserDAO(JDBCDataSource dataSource) {
        this.dataSource = dataSource;
    }
    //其他代码... ...
}
```

知识讲解



+

## 构造器注入 (续1)

- 按构造参数索引指定注入

```
<bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
<property name="driver" value="oracle.jdbc.OracleDriver"/>
<property name="url"
  value="jdbc:oracle:thin:@localhost:1521:XE"/>
<property name="user" value="openlab"/>
<property name="pwd" value="open123"/>
</bean>
<bean id="userDAO" class="org.tarena.dao.OracleUserDAO">
    <constructor-arg index="0" ref="dataSource" />
</bean>
```

知识讲解



+

### 1.3.4. 【容器的 IOC 应用】自动装配

[手写区]

#### 自动装配

• Spring IoC容器可以自动装配 (autowire) 相互协作 bean之间的关联关系，autowire可以针对单个bean进行设置，autowire的方便之处在于减少xml的注入配置

• 在xml配置文件中，可以在<bean/>元素中使用 autowire属性指定自动装配规则，一共有五种类型值

属性值	描述
no	禁用自动装配，默认值
byName	根据属性名自动装配。此选项将检查容器并根据名字查找与属性完全一致的 bean，并将其与属性自动装配
byType	如果容器中存在一个与指定属性类型相同的bean，那么将与该属性自动装配
constructor	与byType的方式类似，不同之处在于它应用于构造器参数
autodetect	通过bean类来决定是使用constructor还是byType方式进行自动装配。如果发现默认的构造器，那么将使用byType方式

[知识讲解]

[+]

[手写区]

#### 自动装配 (续1)

• 配置示例

```
<bean id="userDAO" class="org.tarena.dao.OracleUserDAO">
    <constructor-arg index="0" ref="dataSource"/>
</bean>
<bean id="userService" class="org.tarena.service.UserService"
    autowire="byType"/>
```

上述配置，在UserService中如果存在接收UserDao类型的Setter方法，Spring就可以自动将userDAO对象注入进去

[知识讲解]

[+]

## 经典案例

### 1. 实例化 Spring 容器示例

- 问题

使用 ApplicationContext 的方式实例化 Spring 容器。

- 方案

使用 ApplicationContext 的方式实例化 Spring 容器的核心代码如下：

```
String conf = "applicationContext.xml";
ApplicationContext ac =
    new ClassPathXmlApplicationContext(conf);
```

- 步骤

#### 步骤一：新建工程，导入 jar 包

新建名为 SpringIoC\_Day01\_Part1 的 web 工程，在该工程导入如图-1 所示的 5 个 Spring 相关 jar 包。

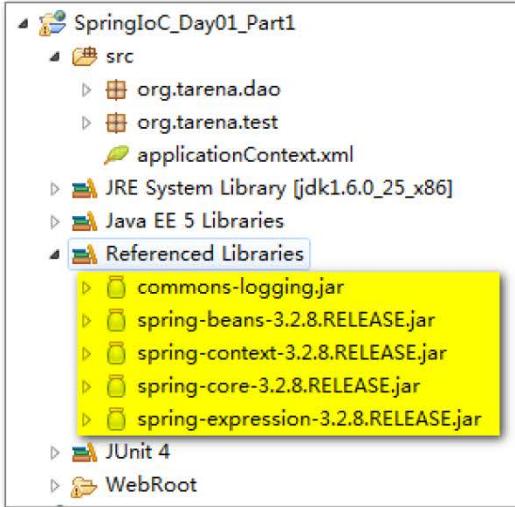


图 - 1

#### 步骤二：新建 Spring 配置文件

新建 Spring 配置文件 applicationContext.xml。该文件名为 Spring 默认的配置文件名，也可以自由定义名称，如图-2 所示：

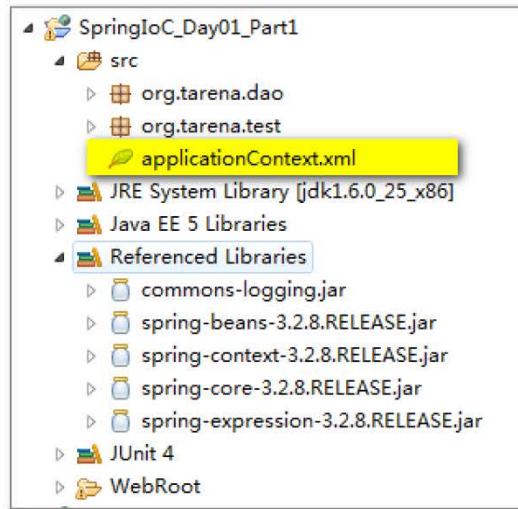


图 - 2

`applicationContext.xml` 文件中的代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd">
</beans>
```

### 步骤三：新建类 Test1

导入 JUnit4，用于软件的单元测试。

新建类 TestCase，在类中使用 ApplicationContext 的方式实例化 Spring 容器。

TestCase 在工程中的位置如图-3 所示：

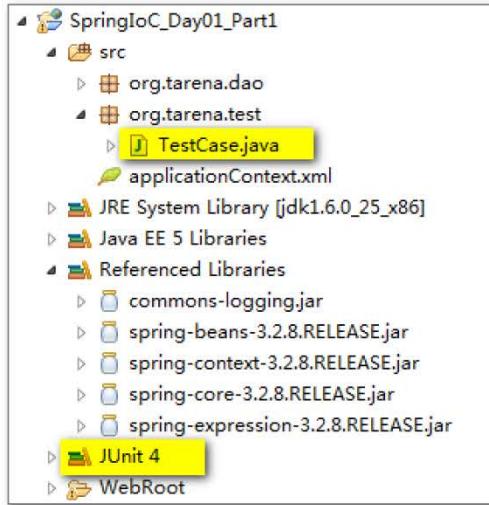


图 - 3

在 TestCase 类中添加测试方法 testInitContext()，代码如图-4 所示：

```
public class TestCase {
    /** 测试实例化Spring容器示例 */
    @Test
    public void testInitContext(){
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        System.out.println(ac);
    }
}
```

图 - 4

#### 步骤四：运行 testInitContext()方法

运行 testInitContext()方法，控制台输出结果如图-5 所示：

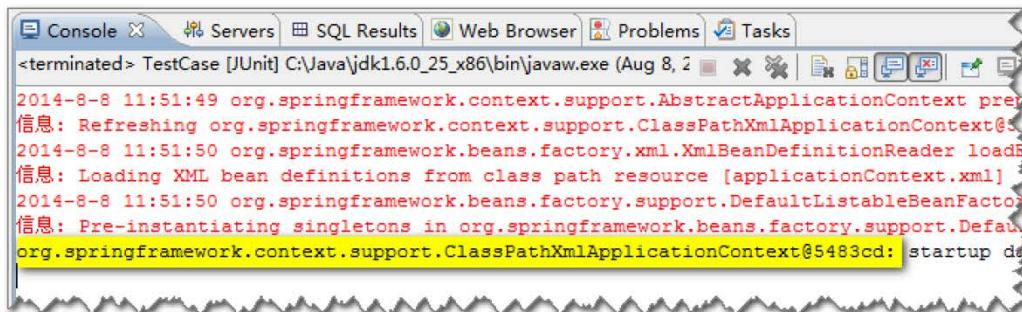


图 - 5

控制台输出如图-5 所示的信息，说明实例化 Spring 容器成功。

- **完整代码**

TestCase 类的完整代码如下：

```

package org.tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestCase {
    /** 测试实例化 Spring 容器示例 */
    @Test
    public void testInitContext(){
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        System.out.println(ac);
    }
}

```

**applicationContext.xml 完整代码如下：**

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd">
</beans>

```

## 2. 利用 Spring 容器创建 JavaBean 对象

- **问题**

测试 Spring 支持的多种 JavaBean 对象创建方式：

1. 用构造器来实例化的方式。

    利用 Spring 调用构造器 GregorianCalendar 创建 Calendar 实例。

2. 使用静态工厂方法实例化的方式。

    利用 Spring 调用 Calendar 的静态工厂方法 getInstance() 创建 Calendar 实例。

3. 使用实例工厂方法实例化的方式。

    利用 Spring 创建 GregorianCalendar 对象作为工厂，调用 getTime()方法创建 Date 类型对象实例。

## • 方案

1. 用构造器来实例化的方式的配置代码如下：

```
<bean id="calendarObj1" class="java.util.GregorianCalendar"></bean>
```

bean 标记中 id 属性 calendarObj1 用于定义 bean 名字，是程序代码中获得 Spring 管理 bean 对象的标识，这个名字不能重复，class 用于指定创建对象的类 GregorianCalendar，Spring 会自动的调用 GregorianCalendar 类的默认构造器创建 bean 对象实例。

2. 使用静态工厂方法实例化的方式的配置代码如下：

```
<bean id="calendarObj2"
      class="java.util.Calendar" factory-method="getInstance">
</bean>
```

bean 标记中 id 属性 calendarObj2 用于定义 bean 名字，是程序代码中获得 Spring 管理 bean 对象的标识，这个名字不能重复，class 属性用于指定创建对象的工厂类 Calendar，factory-method 属性用于指定创建对象的静态工厂方法 getInstance，Spring 会自动的调用工厂类 Calendar 静态工厂方法 getInstance 创建 bean 对象实例。

3. 使用实例工厂方法实例化的方式的配置代码如下：

```
<bean id="calendarObj3" class="java.util.GregorianCalendar"></bean>
<bean id="dateObj"
      factory-bean="calendarObj3" factory-method="getTime">
</bean>
```

这里定义了两个 bean，其中一个 bean calendarObj3 是用于创建 dateObj 对象的实例工厂。

另外一个 bean 标记中 id 属性 dateObj 用于定义 bean 名字，是程序代码中获得 Spring 管理 bean 对象的标识，这个名字不能重复，factory-bean 属性用于指定创建对象的工厂对象 calendarObj3，前面定义的一个 bean，factory-method 属性用于指定创建对象的工厂方法 getTime，Spring 会自动的调用工厂类 Calendar 静态工厂方法 getInstance 创建 bean 对象实例。

## • 步骤

**步骤一：配置 applicationContext.xml，增加 Bean 对象创建声明**

代码如下所示：

```
<!-- 1. 用构造器来实例化的方式的配置代码如下： -->
<bean id="calendarObj1" class="java.util.GregorianCalendar"></bean>

<!-- 2. 使用静态工厂方法实例化的方式的配置代码如下： -->
<bean id="calendarObj2" class="java.util.Calendar"
```

```
factory-method="getInstance">  
</bean>  
  
<!-- 3. 使用实例工厂方法实例化的方式的配置代码如下： -->  
<bean id="calendarObj3" class="java.util.GregorianCalendar"></bean>  
<bean id="dateObj" factory-bean="calendarObj3"  
      factory-method="getTime">  
</bean>
```

## 步骤二：在 TestCase 类中增加测试方法 testCreateBeanObject，测试 Spring 创建对象的结果

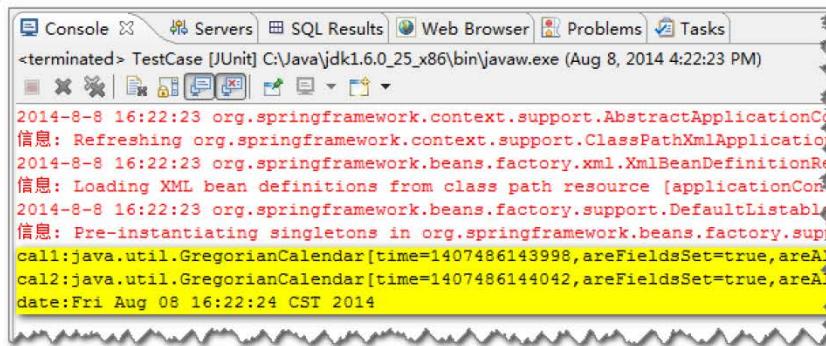
先创建 Spring 容器对象，再调用 getBean 方法获得 Spring 创建的对象实例，并且利用输出语句测试对象是否存在。这个代码中要注意： getBean 方法的参数必须是上一个步骤中定义的 bean 标记上的 id 属性的值，否则会出现运行异常。

代码如下所示：

```
/** 测试 Spring 支持的多种 JavaBean 对象创建方式 */  
@Test  
public void testCreateBeanObject(){  
    //实例化 Spring 容器示例  
    String conf = "applicationContext.xml";  
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);  
  
    //1. 用构造器来实例化的方式。  
    //利用 Spring 调用构造器 GregorianCalendar 创建 Calendar 实例。  
    //Calendar call1 = (Calendar)ac.getBean("calendarObj1"); //方式 1  
    Calendar call1 = ac.getBean("calendarObj1", Calendar.class); //方式 2  
    System.out.println("call1:"+call1);  
  
    //2. 使用静态工厂方法实例化的方式。  
    //利用 Spring 调用 Calendar 的静态工厂方法 getInstance() 创建 Calendar 实例。  
    Calendar cal2 = ac.getBean("calendarObj2", Calendar.class);  
    System.out.println("cal2:"+cal2);  
  
    //3. 使用实例工厂方法实例化的方式。  
    //利用 Spring 创建 GregorianCalendar 对象作为工厂，调用getTime()方法创建 Date  
    //类型对象实例。  
    Date date = ac.getBean("dateObj", Date.class);  
    System.out.println("date:"+date);  
}
```

## 步骤三：运行测试方法测试 bean 实例化

控制台输出结果如图-6 所示：



The screenshot shows the Eclipse IDE's Console view with the following log output:

```

2014-8-8 16:22:23 org.springframework.context.support.AbstractApplicationContext
信息: Refreshing org.springframework.context.support.ClassPathXmlApplication
2014-8-8 16:22:23 org.springframework.beans.factory.xml.XmlBeanDefinitionRe
信息: Loading XML bean definitions from class path resource [applicationCon
2014-8-8 16:22:23 org.springframework.beans.factory.support.DefaultListabl
信息: Pre-instantiating singletons in org.springframework.beans.factory.sup
call1:java.util.GregorianCalendar[time=1407486143998,areFieldsSet=true,areAl
call2:java.util.GregorianCalendar[time=1407486144042,areFieldsSet=true,areAl
date:Fri Aug 08 16:22:24 CST 2014

```

图 - 6

### • 完整代码

TestCase 类的 testCreateBeanObject 方法完整代码如下所示：

```

package org.tarena.test;

import java.util.Calendar;
import java.util.Date;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestCase {

    /** 测试 Spring 支持的多种 JavaBean 对象创建方式 */
    @Test
    public void testCreateBeanObject(){
        //实例化 Spring 容器示例
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);

        //1. 用构造器来实例化的方式。
        //利用 Spring 调用构造器 GregorianCalendar 创建 Calendar 实例。
        //Calendar call1 = (Calendar)ac.getBean("calendarObj1"); //方式1
        Calendar call1 = ac.getBean("calendarObj1", Calendar.class); //方式2
        System.out.println("call1:"+call1);

        //2. 使用静态工厂方法实例化的方式。
        //利用 Spring 调用 Calendar 的静态工厂方法 getInstance() 创建 Calendar 实例。
        Calendar cal2 = ac.getBean("calendarObj2", Calendar.class);
        System.out.println("cal2:"+cal2);

        //3. 使用实例工厂方法实例化的方式。
        //利用 Spring 创建 GregorianCalendar 对象作为工厂，调用getTime()方法创建Date
        //类型对象实例。
        Date date = ac.getBean("dateObj", Date.class);
        System.out.println("date:"+date);
    }
}

```

applicationContext.xml 文件的完整代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
    http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd"
    default-lazy-init="true" default-init-method="init"
    default-destroy-method="destroy">

    <!-- 1. 用构造器来实例化的方式的配置代码如下： -->
<bean id="calendarObj1" class="java.util.GregorianCalendar"></bean>

    <!-- 2. 使用静态工厂方法实例化的方式的配置代码如下： -->
<bean id="calendarObj2" class="java.util.Calendar"
    factory-method="getInstance">
</bean>

    <!-- 3. 使用实例工厂方法实例化的方式的配置代码如下： -->
<bean id="calendarObj3" class="java.util.GregorianCalendar"></bean>
<bean id="dateObj" factory-bean="calendarObj3"
    factory-method="getTime">
</bean>

</beans>

```

### 3. 如何控制 Bean 实例化

- **问题**

测试 Bean 的作用域、Bean 的生命周期回调、Bean 对象的创建时机以及如何指定 bean 依赖关系。

- **步骤**

#### 步骤一：Bean 对象的创建模式

1. 新建 ExampleBean，该类在工程中的位置如图-7 所示：

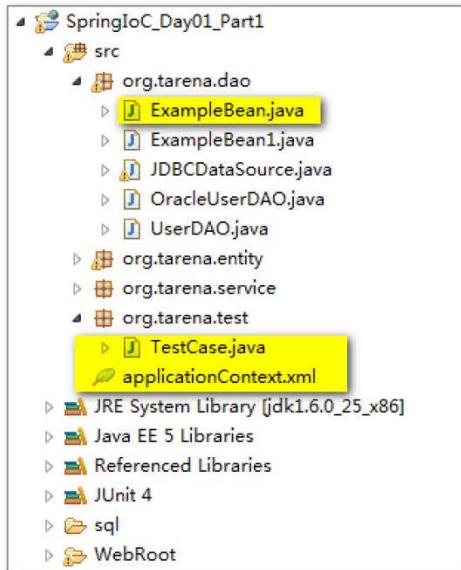


图 - 7

ExampleBean 类的代码如图-8 所示：

```
public class ExampleBean {
    public ExampleBean(){
        System.out.println("实例化ExampleBean");
    }
    public void execute(){
        System.out.println("执行ExampleBean处理");
    }
}
```

图 - 8

2 . 在 applicationContext.xml 文件中 , 配置 ExampleBean , 代码如图-9 所示 :

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"/>
```

图 - 9

3 . 在 TestCase 中新建测试方法 testExampleBean() , 在方法中从 Spring 中获取两个 ExampleBean 类型对象 , 通过比较操作符 “ == ” 进行比较 , 如果输出结果为 true , 则表明两次获取的是同一个对象 , 即创建对象的方式单例模式 , 代码如图-10 所示 :

```
/** 测试 Bean 对象的作用域 */
@Test
public void testExampleBean() {
    //实例化Spring容器示例
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取ExampleBean对象
    ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
    ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
    System.out.println(bean1==bean2);
}
```

图 - 10

4 . 运行 `testExampleBean()`方法 , 控制台输出结果如下 :

```
实例化 ExampleBean
true
```

上述运行结果可以看得出在软件运行期间 `ExampleBean` 的构造器只被调用过一次 , 创建过一个对象 , 两次获得引用变量 `bean1` , `bean2` , 通过比较操作符 “`==`” 进行比较的输出结果为 `true` , 说明是引用了同一个对象 , 也就说明 Spring 容器创建 Bean 对象是唯一实例 , 是单例对象。

5 . 修改 `applicationContext.xml` , 设置创建 Bean 的模式为原型模式( `prototype` ) , 代码如图-11 所示 :

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
scope="prototype" ></bean>
```

图 - 11

6 . 再次运行 `testExampleBean()`方法 , 控制台输出结果如下 :

```
实例化 ExampleBean
实例化 ExampleBean
false
```

这个结果说明调用了 2 次 `ExampleBean` 类的构造方法创建了两个 Bean 对象 , 比较结果是 `false` 表示 `bean1` 和 `bean2` 引用了这两个不同的对象 , 这样创建 bean 就不再是单例模式了。

## 步骤二 : Bean 对象的初始化和销毁

1. 修改 `ExampleBean` 类 , 加入方法 `init` 和方法 `destroy` , 代码如下所示 :

```
package org.tarena.dao;

public class ExampleBean {

    public ExampleBean(){
```

```

        System.out.println("实例化 ExampleBean");
    }

    public void execute(){
        System.out.println("执行 ExampleBean 处理");
    }

    public void init(){
        System.out.println("初始化 ExampleBean 对象");
    }
    public void destroy(){
        System.out.println("销毁 ExampleBean 对象");
    }

}

```

2. 修改 applicationContext.xml 希望在 bean 对象创建后自动调用 init()方法，代码如图-12 所示：

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
scope="prototype" init-method="init"></bean>
```

图 - 12

3. 运行 testExampleBean()方法，自定义的初始化的方法在对象被创建后调用，如图-13 所示：

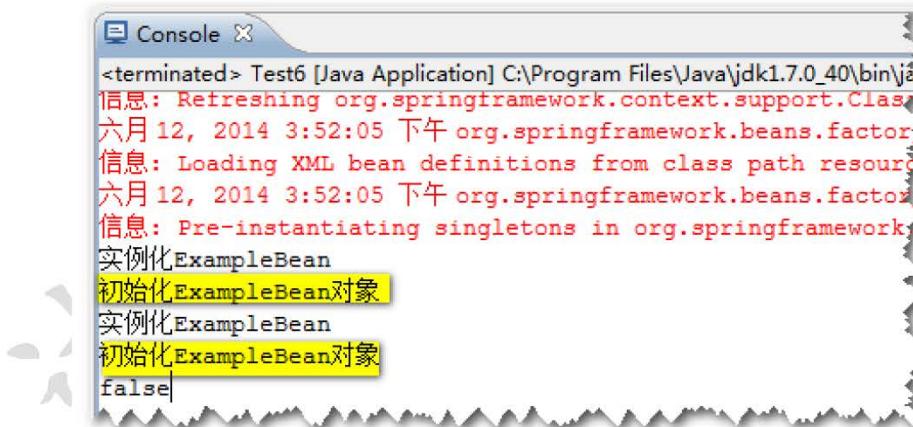


图 - 13

4. 修改 applicationContext.xml 希望在 bean 对象销毁前自动调用 destroy 方法，bean 对象在 spring 容器关闭的时候被销毁，代码如图-14 所示：

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
scope="prototype" init-method="init" destroy-method="destroy">
</bean>
```

图 - 14

5.修改 testExampleBean()方法，关闭 ApplicationContext 对象，代码如图-15 所示：

```
/** 测试 Bean 对象的作用域*/
@Test
public void testExampleBean(){
    //实例化Spring容器示例
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取ExampleBean对象
    ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
    ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
    System.out.println(bean1==bean2);
    //关闭Spring容器，注意AbstractApplicationContext类型定义了 close()方法
    AbstractApplicationContext ctx = (AbstractApplicationContext)ac;
    ctx.close();
}
```

图 - 15

6.运行 testExampleBean()方法，控制台输出结果如图-16 所示：

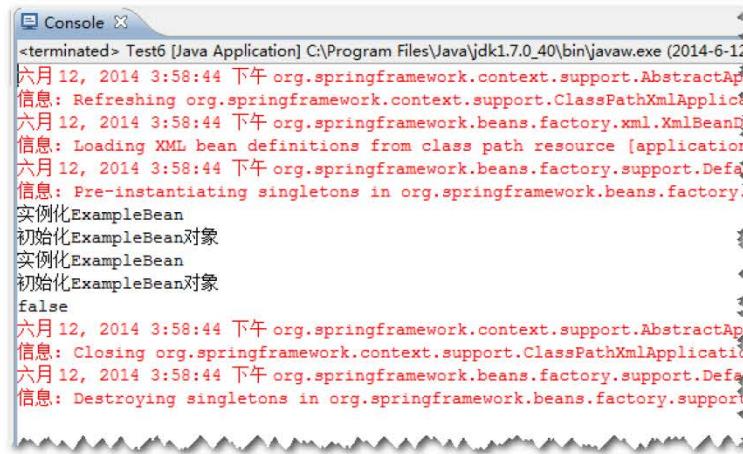


图 - 16

控制台没有输出预期的“销毁 ExampleBean 对象”的结果。原因在于 applicationContext.xml 文件中设置的 destroy-method 属性仅仅对单例模式起作用，在 prototype 模式下没有意义。

7.修改 applicationContext.xml，使用 singleton 模式创建 Bean 对象，代码如图-17 所示：

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
scope="singleton" init-method="init" destroy-method="destroy">
</bean>
```

图 - 17

8.运行 testExampleBean()方法，控制台输出了“销毁 ExampleBean 对象”，如图-18 所示：



图 - 18

9. 在顶级的 `<beans/>` 元素中的 `default-init-method` 属性以及 `default-destroy-method` 属性，可以为容器所有 `<bean>` 指定初始化回调方法以及指定销毁回调方法，代码如图-19 所示：

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="ht...
  xmlns:context="http://www.springframework.org/schema/context" xmlns:...
  xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http...
  xmlns:jpa="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="...
    http://www.springframework.org/schema/beans http://www.springfra...
    http://www.springframework.org/schema/context http://www.springf...
    http://www.springframework.org/schema/jdbc http://www.springfram...
    http://www.springframework.org/schema/jee http://www.springfram...
    http://www.springframework.org/schema/tx http://www.springfram...
    http://www.springframework.org/schema/data/jpa http://www.spring...
  default-init-method="init" default-destroy-method="destroy">
```

图 - 19

### 步骤三：Bean 对象的创建时机

1. 注释 `testExampleBean` 中如图-20 所示的代码。

```
/** 测试 Bean 对象的作用域*/
@Test
public void testExampleBean(){
    //实例化Spring容器示例
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取ExampleBean对象
    //ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
    //ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
    //System.out.println(bean1==bean2);
    //关闭Spring容器，注意AbstractApplicationContext类型定义了 close() 方法
    //AbstractApplicationContext ctx = (AbstractApplicationContext)ac;
    //ctx.close();
}
```

图 - 20

2. 运行 testExampleBean 方法，控制台输出结果如图-21 所示：

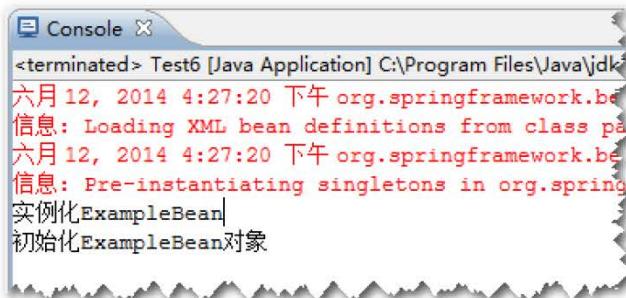


图- 21

控制台打印结果，说明默认情况下 ExampleBean 在 Spring 容器被创建时就会创建。

3. 修改 applicationContext.xml，通过设置配置文件属性 lazy-init="true"，可以改变 Spring 容器创建对象的时机，代码如图-22 所示：

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
  scope="singleton" init-method="init" destroy-method="destroy"
  lazy-init="true">
</bean>
```

图 - 22

4. 运行 testExampleBean 方法，控制台没有输出信息，因为对象并没有被实例化，或者说，实例化被延迟了。

5. 去除 testExampleBean 方法注释掉的代码，如图-23 中所示。

```
@Test
public void testExampleBean() {
    //实例化Spring容器示例
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取ExampleBean对象
    ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
    ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
    System.out.println(bean1==bean2);
    //关闭Spring容器，注意AbstractApplicationContext类型定义了 close()方法
    AbstractApplicationContext ctx = (AbstractApplicationContext)ac;
    ctx.close();
}
```

图 - 23

6. 运行 testExampleBean 方法，控制台输出结果如图-24 所示：

```

Console <terminated> Test6 [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe (2
六月 12, 2014 4:39:33 下午 org.springframework.context.support.Abst...
信息: Refreshing org.springframework.context.support.ClassPathXml...
六月 12, 2014 4:39:33 下午 org.springframework.beans.factory.xml.Xml...
信息: Loading XML bean definitions from class path resource [appli...
六月 12, 2014 4:39:33 下午 org.springframework.beans.factory.suppor...
信息: Pre-instantiating singletons in org.springframework.beans.fa...
实例化ExampleBean
初始化ExampleBean对象
true
六月 12, 2014 4:39:33 下午 org.springframework.context.support.Abst...
信息: Closing org.springframework.context.support.ClassPathXmlAppl...
销毁ExampleBean对象
六月 12, 2014 4:39:33 下午 org.springframework.beans.factory.suppor...
信息: Destroying singletons in org.springframework.beans.factory...

```

图- 24

从输出结果可以看出，当使用 ExampleBean 对象时，才被创建，即，设置 `lazy-init="true"` 属性后，对象不使用不创建。

7. 在顶级的`<beans/>`元素中的 `default-lazy-init` 属性，可以为容器所有`<bean>`指定延迟实例化特性，代码如图-25 所示：

```

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema...
xmlns:context="http://www.springframework.org/schema/context" xmlns:jdbc="http://www.springfram...
xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org...
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/sprin...
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context...
    http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring...
    http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring...
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx...
    http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/j...
    default-lazy-init="true" default-init-method="init" default-destroy-method="destroy">

```

图 - 25

#### 步骤四：指定 bean 依赖关系

1. 新建类 ExampleBean1，代码如下所示：

```

package org.tarena.dao;

public class ExampleBean1 {
    public ExampleBean1(){
        System.out.println("实例化 ExampleBean1");
    }
}

```

2. 修改 `applicationContext.xml` 文件，将 `ExampleBean` 依赖 `ExampleBean1`，代码如图-26 所示：

```
<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
    scope="singleton" init-method="init" destroy-method="destroy"
    depends-on="bean1">
</bean>
<bean id="bean1" class="org.tarena.dao.ExampleBean1" lazy-init="true">
</bean>
```

图 - 26

3. 运行 testExampleBean 方法，控制台输出结果如图-27 所示：

```
Console X
<terminated> Test6 [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe (20)
六月 12, 2014 4:59:06 下午 org.springframework.context.support.Abst...
信息: Refreshing org.springframework.context.support.ClassPathXmlAp...
六月 12, 2014 4:59:06 下午 org.springframework.beans.factory.xml.Xm...
信息: Loading XML bean definitions from class path resource [appli...
六月 12, 2014 4:59:06 下午 org.springframework.beans.factory.support...
信息: Pre-instantiating singletons in org.springframework.beans.fa...
实例化ExampleBean1
实例化ExampleBean
初始化ExampleBean对象
true
六月 12, 2014 4:59:06 下午 org.springframework.context.support.Abst...
信息: Closing org.springframework.context.support.ClassPathXmlAp...
六月 12, 2014 4:59:06 下午 org.springframework.beans.factory.support...
信息: Destroying singletons in org.springframework.beans.factory...
销毁ExampleBean对象
```

图 - 27

从图-27 可以看出，由于 ExampleBean 依赖于 ExampleBean1，因此在创建 ExampleBean 的同时，也创建了 ExampleBean1。

- **完整代码**

ExampleBean 类的完整代码如下所示：

```
package org.tarena.dao;

public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化 ExampleBean");
    }

    public void execute(){
        System.out.println("执行 ExampleBean 处理");
    }
    public void init(){
        System.out.println("初始化 ExampleBean 对象");
    }
    public void destroy(){
        System.out.println("销毁 ExampleBean 对象");
    }
}
```

ExampleBean1 类的完整代码如下所示：

```
package org.tarena.dao;

public class ExampleBean1 {
    public ExampleBean1(){
        System.out.println("实例化 ExampleBean1");
    }
}
```

testExampleBean 方法的完整代码如下所示：

```
@Test
public void testExampleBean(){
    //实例化 Spring 容器示例
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取 ExampleBean 对象
    ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
    ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
    System.out.println(bean1==bean2);
    //关闭 Spring 容器，注意 AbstractApplicationContext 类型定义了 close() 方法
    AbstractApplicationContext ctx = (AbstractApplicationContext)ac;
    ctx.close();
}
```

applicationContext.xml 文件的完整代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd"
           default-lazy-init="true" default-init-method="init"
           default-destroy-method="destroy">

    <!-- 1. 用构造器来实例化的方式的配置代码如下： -->
    <bean id="calendarObj1" class="java.util.GregorianCalendar"></bean>

    <!-- 2. 使用静态工厂方法实例化的方式的配置代码如下： -->
    <bean id="calendarObj2" class="java.util.Calendar"
          factory-method="getInstance">
    </bean>

    <!-- 3. 使用实例工厂方法实例化的方式的配置代码如下： -->

```

```

<bean id="calendarObj3" class="java.util.GregorianCalendar"></bean>
<bean id="dateObj" factory-bean="calendarObj3"
      factory-method="getTime">
</bean>

<bean id="exampleBean" class="org.tarena.dao.ExampleBean"
      scope="singleton"
      init-method="init" destroy-method="destroy"
      depends-on="bean1"/>
<bean id="bean1" class="org.tarena.dao.ExampleBean1" lazy-init="true"/>

</beans>

```

## 4. 利用 Spring 实现 bean 属性 setter 方式注入

- 问题

JDBCDataSource 类封装了管理数据库连接的方法 `getConnection()`，这个方法在执行之前需要数据库连接参数：数据库驱动，连接 URL，用户名和密码。

JDBCDataSource 代码如下：

```

package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCDataSource implements Serializable{

    private String driver;
    private String url;
    private String user;
    private String pwd;

    public String getDriver() {
        return driver;
    }

    public void setDriver(String driver) {
        try{
            //注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {

```

```

        this.user = user;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public Connection getConnection() throws SQLException{
        Connection conn = DriverManager.getConnection(url, user, pwd);
        return conn;
    }

    public void close(Connection conn){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

利用 Spring 实现 JDBCDataSource 对象的创建，再使用 setter 注入的方式将数据库连接参数注入给 JDBCDataSource。这样就可以正常的调用 getConnection()方法获得数据库连接了。

### • 方案

利用 Spring 配置文件 applicationContext.xml 配置 bean，并且 setter 参数注入 JDBCDataSource 的连接参数，这样 Spring 在创建 JDBCDataSource 对象以后就会自动化的调用 setter 方法注入数据库连接参数。

applicationContext.xml 配置 bean 参考代码如下：

```

<!-- setter 注入 -->
<bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
    <property name="driver" value="oracle.jdbc.OracleDriver"></property>
    <property name="url"
        value="jdbc:oracle:thin:@192.168.0.20:1521:XE"></property>
    <property name="user" value="openlab"></property>
    <property name="pwd" value="open123"></property>
</bean>

```

### • 步骤

#### 步骤一：新建工程，导入 jar 包

新建名为 SpringIoC\_01\_Part2 的 web 工程，在该工程导入如图-28 所示的 6 个 jar 包，包括 Spring API 和 Oracle JDBC Driver。

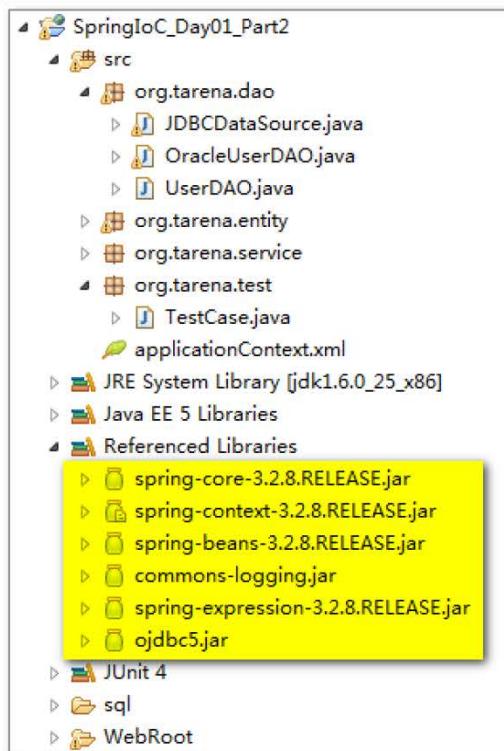


图 - 28

## 步骤二：创建被 Spring 管理的 JDBCDataSource 类，用于连接到数据库

代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCDataSource implements Serializable{

    private String driver;
    private String url;
    private String user;
    private String pwd;

    public String getDriver() {
        return driver;
    }

    public void setDriver(String driver) {
        try{
            //注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }
}
```

```

public void setUrl(String url) {
    this.url = url;
}

public String getUser() {
    return user;
}

public void setUser(String user) {
    this.user = user;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public Connection getConnection() throws SQLException{
    Connection conn = DriverManager.getConnection(url, user, pwd);
    return conn;
}

public void close(Connection conn){
    if(conn!=null){
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}

```

### 步骤三：新建 applicationContext.xml，并且增加 setter 代码注入 JDBC 参数

在配置文件中声明 JDBCDataSource 实例的 bean ID 为“dataSource”，该文件的核心代码如图-29 所示：

```

<!-- setter注入 -->
<bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
    <property name="driver" value="oracle.jdbc.OracleDriver"></property>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"></property>
    <property name="user" value="openlab"></property>
    <property name="pwd" value="open123"></property>
</bean>

```

图 - 29

### 步骤四：新建 TestCase 类，添加测试方法 testJDBCDataSource()

新建的 testJDBCDataSource() 方法中，从 Spring 中获取 ID 为“dataSource”的 JDBCDataSource 对象，Spring 会在创建 JDBCDataSource 对象之后调用 setter 方法注入 JDBC 连接参数，代码如图-30 所示：

```
public class TestCase {
    /** Setter 注入测试 */
    @Test
    public void testJDBCDataSource() throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        System.out.println(ac);
        JDBCDataSource ds = ac.getBean("dataSource", JDBCDataSource.class);
        Connection conn = ds.getConnection();
        System.out.println(conn);
    }
}
```

图 - 30

### 步骤五：运行 testJDBCDataSource 方法

运行 testJDBCDataSource 方法，控制台的输出结果如图-31 所示：

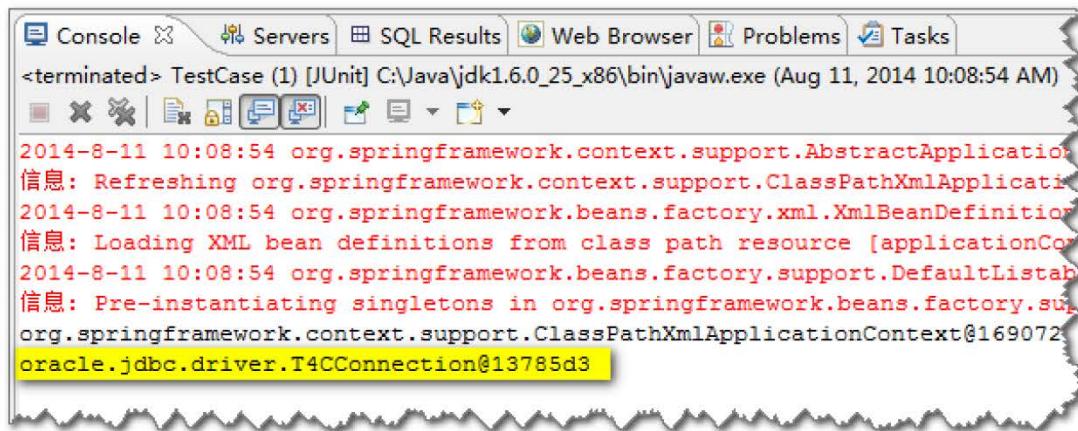


图 - 31

如果能够输出如上结果，说明能够成功的获取 Oracle JDBC 连接，也就说明 Spring 成功的调用 Setter 方法注入了数据库连接参数。

- 完整代码

JDBCDataSource 类的完整代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCDataSource implements Serializable{

    private String driver;
    private String url;
    private String user;
    private String pwd;

    public String getDriver() {
```

```

        return driver;
    }

    public void setDriver(String driver) {
        try{
            //注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public Connection getConnection() throws SQLException{
        Connection conn = DriverManager.getConnection(url, user, pwd);
        return conn;
    }

    public void close(Connection conn){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

TestCase 类的完整代码如下所示：

```

package org.tarena.test;

import java.sql.Connection;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.dao.JDBCDataSource;

public class TestCase {
    /** Setter 注入测试 */
    @Test

```

```

public void testJDBCDataSource() throws Exception{
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    System.out.println(ac);
    JDBCDataSource ds = ac.getBean("dataSource", JDBCDataSource.class);
    Connection conn = ds.getConnection();
    System.out.println(conn);
}
}

```

**applicationContext.xml 文件的完整代码如下所示：**

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd"
        default-init-method="init" default-destroy-method="destroy">

    <!-- setter 注入 -->
    <bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
        <property name="driver" value="oracle.jdbc.OracleDriver"/></property>
        <property
            name="url"
            value="jdbc:oracle:thin:@localhost:1521:XE"/></property>
        <property name="user" value="openlab"/></property>
        <property name="pwd" value="open123"/></property>
    </bean>
</beans>

```

## 5. 利用构造器参数实现依赖属性的注入

- **问题**

OracleUserDAO 是经典的数据访问接口实现类，这个类工作必须依赖 Oracle 数据库连接来工作，JDBCDataSource 实例可以提供 Oracle 数据库的连接，OracleUserDAO 类采用构造器参数的方式，依赖 JDBCDataSource 类，这样的好处是创建 OracleUserDAO 类必须有参数 JDBCDataSource 对象实例。Spring 支持利用构造器注入参数创建 Bean 对象的方式。

UserDAO 接口参考如下：

```
/**
```

```
* 用户数据访问对象接口
*/
public interface UserDAO {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
}
```

User 类型参考如下：

```
public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public User(String name, String pwd, String phone) {
        super();
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
```

```

public int hashCode() {
    return id;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (obj instanceof User) {
        User o = (User) obj;
        return this.id == o.id;
    }
    return true;
}

@Override
public String toString() {
    return id+","+name+","+pwd+","+phone;
}
}

```

### OracleUserDAO 类参考如下：

```

public class OracleUserDAO implements UserDAO{

    private JDBCDataSource dataSource;

    /** 创建 OracleUserDAO 对象必须依赖于 JDBCDataSource 实例 */
    public OracleUserDAO(JDBCDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}

```

- 方案

Spring 支持利用构造器注入参数实例化 Bean 方式。只要在 Spring 的配置文件中增加构造器参数 `constructor-arg` , Spring 就会自动的调用有参数的构造器创建 bean 对象实例，整个过程无需程序编码只需要配置 `applicationContext.xml` 文件即可，代码参考如下：

```
<!-- setter 注入 -->
<bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
    <property name="driver" value="oracle.jdbc.OracleDriver"></property>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"></property>
    <property name="user" value="openlab"></property>
    <property name="pwd" value="open123"></property>
</bean>

<!-- 构造器注入 -->
<bean id="userDAO" class="org.tarena.dao.OracleUserDAO">
    <!-- 利用构造器参数注入 bean 的属性 -->
    <constructor-arg name="dataSource" ref="dataSource"/>
</bean>
```

- 步骤

### 步骤一：新建业务实体类：User 类

User 类代表软件中的用户实例类型，用户对象信息存储在 Oracle 数据库中。User 类代码如下所示：

```
package org.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public User(String name, String pwd, String phone) {
        super();
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }
```

```

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

@Override
public int hashCode() {
    return id;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (obj instanceof User) {
        User o = (User) obj;
        return this.id == o.id;
    }
    return true;
}

@Override
public String toString() {
    return id+","+name+","+pwd+","+phone;
}
}

```

## 步骤二：创建 Oracle 数据库初始化 SQL 脚本，并且执行

创建 Oracle 数据库的初始化 SQL 脚本，在数据库中创建 Users 表，并且存入实例数据用于测试需要，在 Oracle 上执行这个 SQL 脚本。Oracle 初始化 SQL 脚本参考如下：

```

-- 创建用户表
CREATE TABLE USERS
(
    ID NUMBER(7, 0),
    NAME VARCHAR2(50),
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50),
    PRIMARY KEY (id),
)

```

```
-- 登录用户名唯一约束
constraint name unique unique(name)
);

-- 用户 ID 生成序列
CREATE SEQUENCE SEQ_USERS;

-- 向数据库插入模拟数据
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');
```

### 步骤三：创建 UserDAO 接口，定义用户查询功能

创建 UserDAO 接口，声明用户数据查询方法 `findByName`，该方法从数据库中根据唯一的用户名查询用户对象，如果没有查询到对象返回 null。参考代码如下：

```
package org.tarena.dao;

import org.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDAO {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
}
```

### 步骤四：创建 OracleUserDAO 类，实现 UserDAO 接口定义的功能

创建 OracleUserDAO 类，实现 UserDAO 接口的 `findByName` 方法，该方法用户从数据库中根据唯一的用户名查询用户对象，如果没有查询到对象返回 null。这个方法的实现必须依赖于 `JDBCDataSource` 属性，需要利用 `JDBCDataSource` 获得数据库连接，进行数据查询得到用户数据。参考代码如下：

```
package org.tarena.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.tarena.entity.User;

public class OracleUserDAO implements UserDAO{

    private JDBCDataSource dataSource;

    /** 创建 OracleUserDAO 对象必须依赖于 JDBCDataSource 实例 */
    public OracleUserDAO(JDBCDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
    }
}
```

```
String sql = "select id, name, pwd, phone from USERS where name=?";
Connection conn = null;
try {
    conn = dataSource.getConnection();
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setString(1, name);
    ResultSet rs = ps.executeQuery();
    User user=null;
    while(rs.next()){
        user = new User();
        user.setId(rs.getInt("id"));
        user.setName(rs.getString("name"));
        user.setPwd(rs.getString("pwd"));
        user.setPhone(rs.getString("phone"));
    }
    rs.close();
    ps.close();
    return user;
} catch (SQLException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    dataSource.close(conn);
}
}
```

##### 步骤五：配置 Spring 添加 OracleUserDAO 的 bean 定义

在 applicationContext.xml 文件中增加 OracleUserDAO 的 Bean 定义 ,利用构造器初始化属性 dataSource , Spring 会自动调用 OracleUserDAO 有参数构造器创建 OracleUserDAO 实例 , 其中 userDao 是 bean ID。

该文件的核心代码如图-32 所示：



图 - 32

dataSource 为按构造参数注入 , 这个参数是引用了 id 为 dataSource 的 Bean 对象。

##### 步骤六：创建测试方法 testOracleUserDAO() 测试构造器注入

新建 testOracleUserDAO 方法 , 在方法中利用 userDao 作为 Bean ID 获取 OracleUserDAO 对象 , Spring 会自动的调用 OracleUserDAO 的有参数构造器 , 注入 dataSource 对象 , 创建 OracleUserDAO 对象。再调用 findByName 方法测试是否能够正

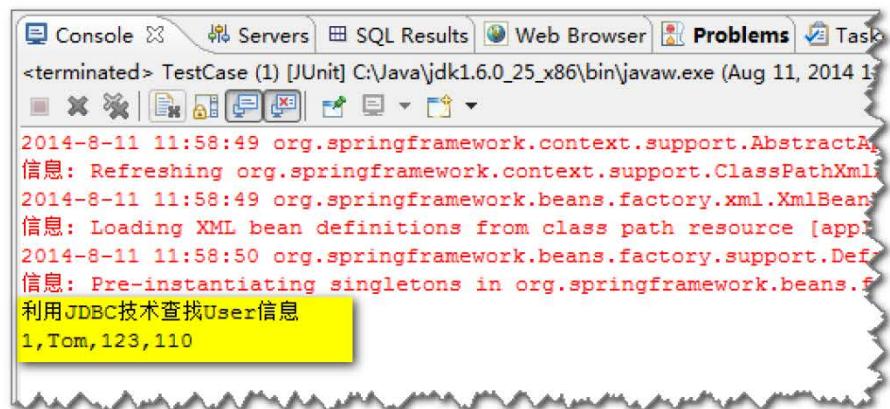
确连接 Oracle 并查询得到用户信息。代码如图-33 所示：

```
/** 构造器参数注入 */
@Test
public void testOracleUserDAO() {
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取OracleUserDAO的实例
    UserDAO userDAO = ac.getBean("userDAO", UserDAO.class);
    //查询用户对象
    User tom = userDAO.findByName("Tom");
    System.out.println(tom);
}
```

图 - 33

### 步骤七：运行 testOracleUserDAO 方法

运行 testOracleUserDAO 方法，控制台的输出结果如图-34 所示：



```
<terminated> TestCase (1) [JUnit] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 11, 2014 11:58:49)
2014-8-11 11:58:49 org.springframework.context.support.AbstractAp
信息: Refreshing org.springframework.context.support.ClassPathXml
2014-8-11 11:58:49 org.springframework.beans.factory.xml.XmlBean
信息: Loading XML bean definitions from class path resource [appl
2014-8-11 11:58:50 org.springframework.beans.factory.support.Def
信息: Pre-instantiating singletons in org.springframework.beans.f
利用JDBC技术查找User信息
1, Tom, 123, 110
```

图 - 34

从输出结果可以看出调用到了 OracleUserDAO 对象的 findByName 方法 ,说明 Spring 正确的使用构造器注入的方式 bean 对象将 dataSource 注入 OracleUserDAO 对象内部，并且有正确的执行结果。

- 完整代码

User 类的完整代码如下所示：

```
package org.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }
```

```
public User(int id, String name, String pwd, String phone) {
    this.id = id;
    this.name = name;
    this.pwd = pwd;
    this.phone = phone;
}

public User(String name, String pwd, String phone) {
    super();
    this.name = name;
    this.pwd = pwd;
    this.phone = phone;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

@Override
public int hashCode() {
    return id;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (obj instanceof User) {
        User o = (User) obj;
        return this.id == o.id;
    }
    return true;
}

@Override
public String toString() {
    return id+","+name+","+pwd+","+phone;
}
```

}

oracle.sql 文件的完整代码如下所示：

```
-- 创建用户表
CREATE TABLE USERS
(
    ID NUMBER(7, 0) ,
    NAME VARCHAR2(50) ,
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50) ,
    PRIMARY KEY (id),
    -- 登录用户名唯一约束
    constraint name unique unique(name)
);

-- 用户 ID 生成序列
CREATE SEQUENCE SEQ_USERS;

-- 向数据库插入模拟数据
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');
```

UserDAO 文件的完整代码如下所示：

```
package org.tarena.dao;

import org.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDAO {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
}
```

OracleUserDAO 文件的完整代码如下所示：

```
package org.tarena.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.tarena.entity.User;

public class OracleUserDAO implements UserDAO{

    private JDBCDataSource dataSource;

    /** 创建 OracleUserDAO 对象必须依赖于 JDBCDataSource 实例 */
    public OracleUserDAO(JDBCDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
```

```

System.out.println("利用 JDBC 技术查找 User 信息");
String sql = "select id, name, pwd, phone from USERS where name=?";
Connection conn = null;
try {
    conn = dataSource.getConnection();
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setString(1, name);
    ResultSet rs = ps.executeQuery();
    User user=null;
    while(rs.next()){
        user = new User();
        user.setId(rs.getInt("id"));
        user.setName(rs.getString("name"));
        user.setPwd(rs.getString("pwd"));
        user.setPhone(rs.getString("phone"));
    }
    rs.close();
    ps.close();
    return user;
} catch (SQLException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    dataSource.close(conn);
}
}
}

```

applicationContext.xml 文件的完整代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd"
        default-lazy-init="true"                               default-init-method="init"
        default-destroy-method="destroy">

    <!-- setter 注入 -->
    <bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
        <property name="driver" value="oracle.jdbc.OracleDriver"/></property>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"/></property>
        <property name="user" value="openlab"/></property>
        <property name="pwd" value="open123"/></property>
    </bean>

    <!-- 构造器注入 -->
    <bean id="userDAO" class="org.tarena.dao.OracleUserDAO">

```

```

<!-- 利用构造器参数注入 bean 的属性 -->
<constructor-arg index="0" ref="dataSource"/>
</bean>

</beans>

```

TestCase 文件的完整代码如下所示：

```

package org.tarena.test;

import java.sql.Connection;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.dao.JDBCDataSource;
import org.tarena.dao.UserDAO;
import org.tarena.entity.User;

public class TestCase {
    /** Setter 注入测试 */
    //@Test
    public void testJDBCDataSource() throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        System.out.println(ac);
        JDBCDataSource ds = ac.getBean("dataSource", JDBCDataSource.class);
        Connection conn = ds.getConnection();
        System.out.println(conn);
    }

    /** 构造器参数注入 */
    @Test
    public void testOracleUserDAO(){
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        //获取 OracleUserDAO 的实例
        UserDAO userDAO = ac.getBean("userDAO", UserDAO.class);
        //查询用户对象
        User tom = userDAO.findByName("Tom");
        System.out.println(tom);
    }
}

```

## 6. 利用 Spring 的自动装配功能实现自动属性注入

- 问题

UserService 类需要依赖 UserDAO 接口的实例实现登录功能 login()，也就是说要能够正确执行 UserService 类型对象的 login 方法必须为 UserService 对象注入 UserDAO 类型的实例。UserService 代码参考如下：

```

package org.tarena.service;

import org.tarena.dao.UserDAO;
import org.tarena.entity.User;

public class UserService {
    private UserDAO userDAO;

```

```

public void setUserDAO(UserDAO userDAO) {
    this.userDAO = userDAO;
}

public UserDAO getUserDAO() {
    return userDAO;
}
/** 登录系统功能 */
public User login(String name, String pwd){
    try{
        User user = userDAO.findByName(name);
        if(user.getPwd().equals(pwd)){
            return user;
        }
        return null;
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
}

```

使用 Spring 自动装配的方式可以自动化的为 UserService 提供 UserDAO 对象实例。

## • 方案

Spring IoC 容器可以自动装配( autowire )相互协作 bean 之间的关联关系 ,autowire 可以针对单个 bean 进行设置 , autowire 的方便之处在于减少 xml 的注入配置。

在 xml 配置文件中 , 可以在<bean/>元素中使用 autowire 属性指定自动装配规则 , 一共有五种类型值 , 如图-35 所示 :

属性值	描述
no	禁用自动装配, 默认值
byName	根据属性名自动装配。此选项将检查容器并根据名字查找与属性完全一致的 bean, 并将其与属性自动装配
byType	如果容器中存在一个与指定属性类型相同的bean, 那么将与该属性自动装配
constructor	与byType的方式类似, 不同之处在于它应用于构造器参数
autodetect	通过bean类来决定是使用constructor还是byType方式进行自动装配。如果发现默认的构造器, 那么将使用byType方式

图 - 35

配置代码 ,Spring 就会自动的根据类型自动调用 setter 方法完成 UserService 依赖的对象 userDAO 的注入 , 不必进行完整的配置 , 参考配置如下 :

```

<!-- 构造器注入 -->
<bean id="userDAO" class="org.tarena.dao.OracleUserDAO">
    <!-- 利用构造器参数注入 bean 的属性 -->
    <constructor-arg name="dataSource" ref="dataSource"/>
</bean>

<!-- 按照类中自动注入属性 -->
<bean id="userService" class="org.tarena.service.UserService"
    autowire="byType"/>

```

上述配置，在 UserService 中如果存在接收 userDao 类型的方法 setter 方法 setUserDAO(UserDAO dao) , Spring 就可以自动按照类型 UserDAO 匹配，将 userDao 类型的 bean 对象 userDao 注入 userService 中。

- 步骤

### 步骤一：创建 UserService 类

新建 UserService 类，UserService 类包含依赖 UserDAO 类型的属性，并且有 Bean 属性的访问方法 setUserDAO，该类中包含登录业务功能方法 login，该方法依赖 userDao 对象调用了 userDao 对象的 findByName 方法查询用户对象，代码如下所示：

```
package org.tarena.service;

import org.tarena.dao.UserDAO;
import org.tarena.entity.User;

public class UserService {

    private UserDAO userDao;

    public void setUserDAO(UserDAO userDao) {
        this.userDAO = userDao;
    }

    public UserDAO getUserDAO() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd){
        try{
            User user = userDao.findByName(name);
            if(user.getPwd().equals(pwd)){
                return user;
            }
            return null;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

### 步骤二：修改 applicationContext.xml

增加 userService 的配置信息，该文件的核心代码如图-36 所示：



```
<!-- 按照类中自动注入属性 -->
<bean id="userService" class="org.tarena.service.UserService" autowire="byType"/>
```

图 - 36

### 步骤三：新建测试方法 testUserService

新建 testUserService 方法，从 Spring 中获取 id 为 userService 类型的对象，调

用 login 方法，代码如图-37 所示：

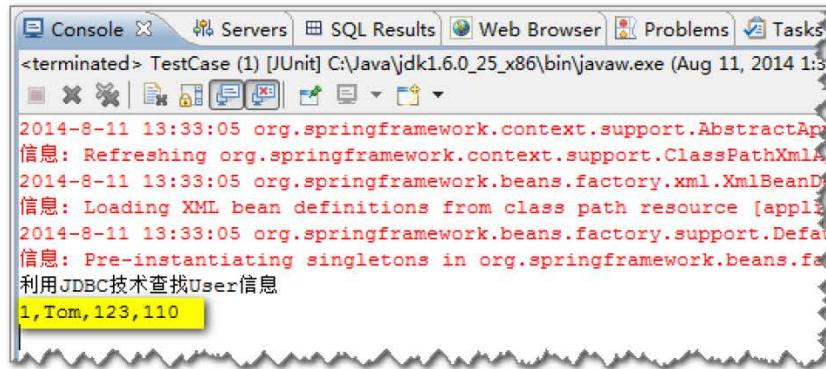


```
/** 自动属性注入测试 */
@Test
public void testUserService(){
    String conf = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    //获取UserService实例
    UserService userService = ac.getBean("userService", UserService.class);
    //调用登录方法， 测试自动注入结果
    User tom = userService.login("Tom", "123");
    System.out.println(tom);
}
```

图 - 37

#### 步骤四：运行 testUserService 方法

运行 testUserService 方法，控制台的输出结果如图-38 所示：



```
Console Servers SQL Results Web Browser Problems Tasks
<terminated> TestCase (1) [JUnit] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 11, 2014 1:3
2014-8-11 13:33:05 org.springframework.context.support.AbstractAp
信息: Refreshing org.springframework.context.support.ClassPathXmlA
2014-8-11 13:33:05 org.springframework.beans.factory.xml.XmlBeanD
信息: Loading XML bean definitions from class path resource [appli
2014-8-11 13:33:05 org.springframework.beans.factory.support.Defat
信息: Pre-instantiating singletons in org.springframework.beans.fa
利用JDBC技术查找User信息
1, Tom, 123, 110
```

图 - 38

从输出结果如果能够正确得到 User 对象实例，说明 userService 能够正确访问 userDao 对象的 login 方法，也说明 userDao 被自动注入到 userService 对象内部。

- 完整代码

UserService 类的完整代码：

```
package org.tarena.service;

import org.tarena.dao.UserDAO;
import org.tarena.entity.User;

public class UserService {
    private UserDAO userDAO;

    public void setUserDAO(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public UserDAO getUserDAO() {
        return userDAO;
    }
}
```

```
/** 登录系统功能 */
public User login(String name, String pwd){
    try{
        User user = userDao.findByName(name);
        if(user.getPwd().equals(pwd) ){
            return user;
        }
        return null;
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```

TestCase 类的完整代码如下所示：

```
package org.tarena.test;

import java.sql.Connection;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.dao.JDBCDataSource;
import org.tarena.dao.UserDAO;
import org.tarena.entity.User;
import org.tarena.service.UserService;

public class TestCase {
    /** Setter 注入测试 */
    //@Test
    public void testJDBCDataSource() throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        System.out.println(ac);
        JDBCDataSource ds = ac.getBean("dataSource", JDBCDataSource.class);
        Connection conn = ds.getConnection();
        System.out.println(conn);
    }

    /** 构造器参数注入 */
    //@Test
    public void testOracleUserDAO(){
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        //获取 OracleUserDAO 的实例
        UserDAO userDAO = ac.getBean("userDAO", UserDAO.class);
        //查询用户对象
        User tom = userDAO.findByName("Tom");
        System.out.println(tom);
    }

    /** 自动属性注入测试 */
    @Test
    public void testUserService(){
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        //获取 UserService 实例
        UserService userService = ac.getBean("userService", UserService.class);
        //调用登录方法， 测试自动注入结果
        User tom = userService.login("Tom", "123");
        System.out.println(tom);
    }
}
```

applicationContext.xml 文件的完整代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd"
        default-lazy-init="true" default-init-method="init"
        default-destroy-method="destroy">

    <!-- setter 注入 -->
    <bean id="dataSource" class="org.tarena.dao.JDBCDataSource">
        <property name="driver" value="oracle.jdbc.OracleDriver"/></property>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"/></property>
        <property name="user" value="openlab"/></property>
        <property name="pwd" value="open123"/></property>
    </bean>

    <!-- 构造器注入 -->
    <bean id="userDAO" class="org.tarena.dao.OracleUserDAO">
        <!-- 利用构造器参数注入 bean 的属性 -->
        <constructor-arg name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 按照类中自动注入属性 -->
    <bean id="userService" class="org.tarena.service.UserService" autowire="byType"/>
</beans>
```

## 课后作业

1. 描述 Spring 框架的作用和优点。
2. 如何控制 Bean 对象的作用域，默认作用域是什么？
3. 基于 Setter 注入实现 DeptDAO 和 DeptService 的 IoC 控制，并编写测试程序。

达内IT培训集团

# Spring 核心

## Unit02

知识体系.....Page 53

Spring IOC 容器	参数值注入	注入基本值
		注入 Bean 对象
		注入集合
		注入 Spring 表达式值
		注入 null 或空字符串
	基于注解的组件扫描	什么是组件扫描
		指定扫描类路径
		自动扫描的注解标记
		自动扫描组件的命名
		指定组件的作用域
		初始化和销毁回调的控制
		指定依赖注入关系
		注入 Spring 表达式值

经典案例.....Page 62

给 MessageBean 注入参数值	注入 Bean 对象
	注入集合
	注入 Spring 表达式值
	注入 null 或空字符串
测试 Spring 自动组件扫描方式	定义扫描类路径
	自动扫描的注解标记
	自动扫描组件的命名
如何控制 ExampleBean 实例化方式	指定组件的作用域
	指定初始化和销毁回调
使用注解方式重构 JdbcDataSource, UserDAO, UserService	指定依赖注入关系
	注入 Spring 表达式值

课后作业.....Page 114

## 1. Spring IOC 容器

### 1.1. 参数值注入

#### 1.1.1. 【参数值注入】注入基本值

**注入基本值**

<value/>元素可以通过字符串指定属性或构造器参数的值。  
容器将字符串从java.lang.String类型转化为实际的属性或参数类型后给Bean对象注入

```
<bean id="myDataSource"
      class="org.apache.commons.dbcp.BasicDataSource">
    <property name="username">
      <value>root</value>
    </property>
    <property name="password">
      <value>1234</value>
    </property>
    ...
  </bean>
```

**注入基本值 (续1)**

也可以通过value属性指定基本值

```
<bean id="myDataSource"
      class="org.apache.commons.dbcp.BasicDataSource">
    <property name="username" value="root"/>
    <property name="password" value="1234"/>
    ...
  </bean>
```

#### 1.1.2. 【参数值注入】注入 Bean 对象

**注入Bean对象**

注入Bean对象，定义格式有内部Bean和外部Bean两种

- 注入内部Bean

```
<bean id="userService" class="com.service.userService">
  <property name="userDAO">
    <bean class="com.dao.OralceUserDao"></bean>
  </property>
</bean>
```

## 注入Bean对象 (续1)

- 注入外部Bean(引用方式, 方便重用)

```
<bean id="userDAO" class="com.dao.OracleUserDao"/>  
  
<bean id="userService" class="com.service.UserService">  
    <property name="userDAO" ref="userDAO"/>  
</bean>
```

知识讲解



### 1.1.3. 【参数值注入】注入集合

## 注入集合

通过<list>、<set>、<map>及<props>元素可以定义和设置与Java类型中对应List、Set、Map及Properties的属性值。

知识讲解



## 注入集合 (续1)

- List集合注入

```
<bean id="messageBean" class="com.util.MessageBean">  
    <property name="friends">  
        <list>  
            <value>Jack</value>  
            <value>Tom</value>  
        </list>  
    </property>  
    ...  
</bean>
```

知识讲解



知识讲解

## 注入集合 (续2)

- Set集合注入

```
<bean id="messageBean" class="com.util.MessageBean">
    <property name="cities">
        <set>
            <value>北京</value>
            <value>上海</value>
            <value>广州</value>
        </set>
    </property>
    ...
</bean>
```

+

知识讲解

## 注入集合 (续3)

- Map集合注入

```
<bean id="messageBean" class="com.util.MessageBean">
    <property name="books">
        <map>
            <entry key="1001" value="Java语言基础"/>
            <entry key="1002" value="Java Web基础"/>
            <entry key="1003" value="Spring使用基础"/>
        </map>
    </property>
    ...
</bean>
```

+

知识讲解

## 注入集合 (续4)

- Properties集合注入

```
<bean id="messageBean" class="com.util.MessageBean">
    <property name="dbParams">
        <props>
            <prop key="username">root</prop>
            <prop key="password">1234</prop>
            <prop key="driverClassName">
                com.mysql.jdbc.Driver
            </prop>
        </props>
    </property>
    ...
</bean>
```

+

## 注入集合 (续5)

- 引用方式 List集合注入
- ```
<util:list id="oneList">
    <value>Jack</value>
    <value>Tom</value>
</util:list>
<bean id="messageBean" class="com.util.MessageBean" >
    <property name="friends" ref="oneList">
        ...
    </property>
</bean>
```
- Set Map Properties 都可以采用引用方式注入
- ```
<util:list /> <util:set/> <util:props />
```

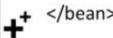


### 1.1.4. 【参数值注入】注入 Spring 表达式值

## 注入Spring表达式值

Spring引入了一种表达式语言，这和统一的EL在语法上很相似，这种表达式语言可以用于定义基于XML和注解配置的Bean，注入一个properties文件信息。

```
<util:properties id="jdbcProperties"
    location="classpath:org/config/jdbc.properties"/>
<bean id="myDataSource"
    class="org.apache.commons.dbcp.BasicDataSource" >
    <property name="username"
        value="#{jdbcProperties.username}"/>
    <property name="password"
        value="#{jdbcProperties.password}"/>
    ...
</bean>
```



### 1.1.5. 【参数值注入】注入 null 或空字符串

## 注入null或空字符串

Spring将属性的空参数当作空String，下面给email属性设置了空String值（""）

```
<bean id="exampleBean" class="com.bean.ExampleBean" >
    <property name="email" value="" />
</bean>
```

如果需要注入null值，可以使用<null/>元素

```
<bean id="exampleBean" class="com.bean.ExampleBean" >
    <property name="email">
        <null/>
    </property>
</bean>
```



## 1.2. 基于注解的组件扫描

### 1.2.1. 【基于注解的组件扫描】什么是组件扫描

**什么是组件扫描**

Technology **Tarena**  
 达内科技

知识讲解

- 指定一个包路径, Spring会自动扫描该包及其子包所有组件类, 当发现组件类定义前有特定的注解标记时, 就将该组件纳入到Spring容器。等价于原有XML配置中的<bean>定义功能
- 组件扫描可以替代大量XML配置的<bean>定义



### 1.2.2. 【基于注解的组件扫描】指定扫描类路径

**指定扫描类路径**

Technology **Tarena**  
 达内科技

使用组件扫描, 首先需要在XML配置中指定扫描类路径

```
<context:component-scan
  base-package="org.example"/>
```

上面配置, 容器实例化时会自动扫描org.example包及其子包下所有组件类。



### 1.2.3. 【基于注解的组件扫描】自动扫描的注解标记

**自动扫描的注解标记**

Technology **Tarena**  
 达内科技

指定扫描类路径后, 并不是该路径下所有组件类都扫描到Spring容器的, 只有在组件类定义前面有以下注解标记时, 才会扫描到Spring容器。

注解标记	描述
@Component	通用注解
@Name	通用注解
@Repository	持久化层组件注解
@Service	业务层组件注解
@Controller	控制层组件注解



#### 1.2.4. 【基于注解的组件扫描】自动扫描组件的命名

知识讲解

**自动扫描组件的命名**

Tarena  
达内科技

当一个组件在扫描过程中被检测到时，会生成一个默认id值，默认id为小写开头的类名。也可以在注解标记中自定义id。下面两个组件id名字分别是oracleUserDao和loginService

```
@Repository
public class OracleUserDao implements UserDao {
    //...
}

@Service("loginService")
public class UserService {
    //...
}
```

#### 1.2.5. 【基于注解的组件扫描】指定组件的作用域

知识讲解

**指定组件的作用域**

Tarena  
达内科技

通常受Spring管理的组件，默认的作用域是"singleton"。如果需要其他的作用域可以使用@Scope注解，只要在注解中提供作用域的名称即可

```
@Scope("prototype")
@Repository
public class OracleUserDao implements EmpDao {
    //...
}
```

#### 1.2.6. 【基于注解的组件扫描】初始化和销毁回调的控制

知识讲解

**初始化和销毁回调的控制**

Tarena  
达内科技

@PostConstruct和@PreDestroy注解标记分别用于指定初始化和销毁回调方法，使用示例

```
public class ExampleBean {
    @PostConstruct
    public void init() {
        //初始化回调方法
    }

    @PreDestroy
    public void destroy() {
        //销毁回调方法
    }
}
```

### 1.2.7. 【基于注解的组件扫描】指定依赖注入关系

知识讲解

#### 指定依赖注入关系

具有依赖关系的Bean对象，利用下面任意一种注解都可以实现关系注入

- @Resource
- @Autowired/@Qualifier
- @Inject/@Named

+

知识讲解

#### 指定依赖注入关系（续1）

@Resource注解标记可以用在字段定义或setter方法定义前面，默认首先按名称匹配注入，然后类型匹配注入

```
public class UserService {  
    // @Resource  
    private UserDao userDao;  
    @Resource  
    public void setUserDao(UserDao dao) {  
        this.userDao = dao;  
    }  
}
```

当遇到多个匹配Bean时注入会发生错误，可以显式指定名称，例如@Resource(name="empDao1")

+

知识讲解

#### 指定依赖注入关系（续2）

@Autowired注解标记也可以用在字段定义或setter方法定义前面，默认按类型匹配注入

```
public class UserService {  
    // @Autowired  
    private UserDao userDao;  
  
    @Autowired  
    public void setUserDao(UserDao dao) {  
        this.userDao = dao;  
    }  
}
```

+

### 指定依赖注入关系 (续3)

@Autowired当遇到多个匹配Bean时注入会发生错误，可以使用下面方法指定名称

```
public class UserService {  
    // @Autowired  
    // @Qualifier("mysqlUserDao")  
    private UserDao userDao;  
  
    @Autowired  
    public void setUserDao(@Qualifier("mysqlUserDao")  
                           UserDao dao){  
        userDao = dao;  
    }  
}
```

知识讲解



### 指定依赖注入关系 (续4)

@Inject注解标记是Spring 3.0开始增添的对JSR-330标准的支持，使用前需要添加JSR-330的jar包，使用方法与@.Autowired相似，具体如下

```
public class UserService {  
    // @Inject  
    private UserDao userDao;  
  
    @Inject  
    public void setUserDao(UserDao dao) {  
        userDao = dao;  
    }  
}
```

知识讲解



### 指定依赖注入关系 (续5)

@Inject当遇到多个匹配Bean时注入会发生错误，可以使用@Named指定名称限定，使用方法如下

```
public class UserService {  
  
    private UserDao userDao;  
  
    @Inject  
    public void setUserDao(@Named("mysqlUserDao")  
                           UserDao dao){  
        userDao = dao;  
    }  
}
```

知识讲解



### 1.2.8. 【基于注解的组件扫描】注入 Spring 表达式值

注入Spring表达式值



@Value注解可以注入Spring表达式值，使用方法

- 首先在XML配置中指定要注入的properties文件

```
<util:properties id="jdbcProps"  
location="classpath:db.properties"/>
```

- 然后在setter方法前使用@Value注解

```
public class JDBCDataSource{  
    @Value("${jdbcProps.url}")  
    private String url;  
    @Value("${jdbcProps.driver}")  
    public void setUrl(String driver) {  
        try{Class.forName(driver)}catch(.....)...  
    }  
}
```

知识讲解

+

达内IT培训

## 经典案例

### 1. 给 MessageBean 注入参数值

- 问题

Spring 可以通过配置文件为 bean 注入多种类型的属性，MessageBean 类用于演示 Spring 的多种类型数据的注入方式，这些类型数据和注入方式包括：

1. 注入基本值。
2. 注入 Bean 对象(请参考 Unit 1 案例)。
3. 直接集合注入。
4. 引用方式集合注入
5. 注入 Spring 表达式值。
6. 注入 null 或空字符串。

- 步骤

#### 步骤一：新建工程，导入 Spring API jar 包

新建名为 SpringIoC\_02\_Part1 的 web 工程，在该工程导入如图-1 所示的 5 个 jar 包。

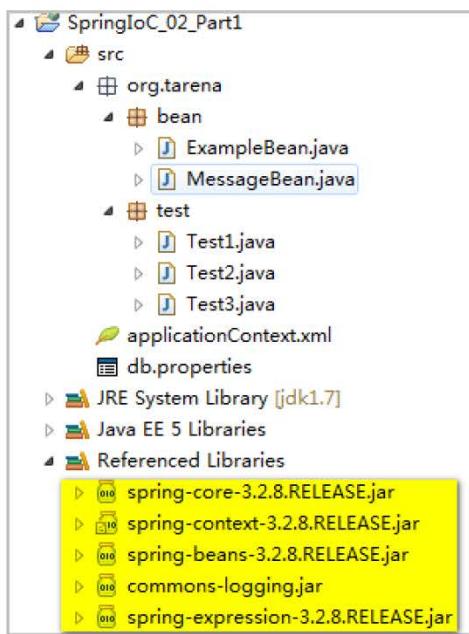


图 - 1

#### 步骤二：新建 Spring 配置文件

新建 Spring 配置文件 applicationContext.xml。该文件名为 Spring 默认的配置文件名，也可以自由定义名称。如图-2 所示：

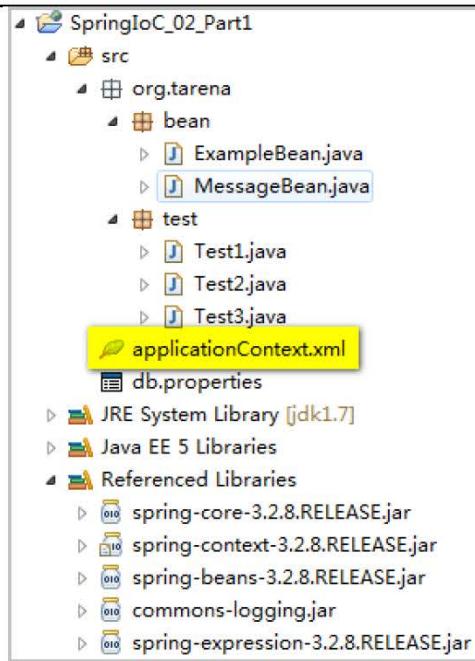


图- 2

`applicationContext.xml` 文件中的代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd">
</beans>
```

### 步骤三：新建类 MessageBean 添加基本值属性

新建类 `MessageBean`，在该类中添加如下代码中的四个属性，并生成这几个属性对应的 `getter` 和 `setter` 方法；最后在 `execute` 方法获取这几个属性的信息，代码如下所示：

```
package org.tarena.bean;

public class MessageBean {
    private String moduleName;
    private int pageSize;
    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;
}
public String execute(){
    System.out.println("moduleName="+moduleName);
    System.out.println("pageSize="+pageSize);
    System.out.println("username="+username);
    System.out.println("password="+password);
    System.out.println("----List 信息如下---");
    return "success";
}
public String getModuleName() {
    return moduleName;
}

public void setModuleName(String moduleName) {
    this.moduleName = moduleName;
}

public int getPageSize() {
    return pageSize;
}

public void setPageSize(int pageSize) {
    this.pageSize = pageSize;
}
}

```

#### 步骤四：修改 applicationContext.xml 文件，增加属性基本值注入

修改 applicationContext.xml 文件，为 MessageBean 的四个属性注入基本参数值，代码如图-3 所示：

```

<bean id="messagebean" class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="资源管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="scott"></property>
    <property name="password" value="tiger"></property>
</bean>

```

图 - 3

#### 步骤五：新建类 Test1，添加测试方法获取 MessageBean 对象测试注入结果

新建类 Test1，在类中使用 ApplicationContext 的方式实例化 Spring 容器，获取 MessageBean 对象，并调用 execute 方法。Test1 类的代码如图-4 所示：

```

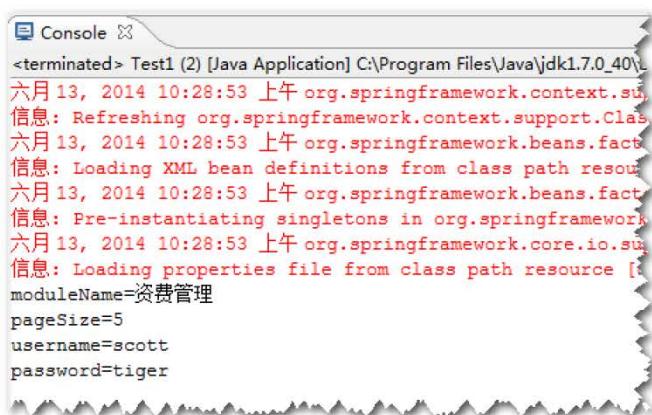
public class Test1 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        MessageBean bean = ac.getBean("messagebean",MessageBean.class);
        bean.execute();
    }
}

```

图 - 4

### 步骤六：运行 Test1 类，测试注入结果

运行 Test1 类，控制台输入结果如图-5 所示：



```

Console 
<terminated> Test1 (2) [Java Application] C:\Program Files\Java\jdk1.7.0_40\
六月 13, 2014 10:28:53 上午 org.springframework.context.su
信息: Refreshing org.springframework.context.support.Clas
六月 13, 2014 10:28:53 上午 org.springframework.beans.fact
信息: Loading XML bean definitions from class path resou
六月 13, 2014 10:28:53 上午 org.springframework.beans.fact
信息: Pre-instantiating singletons in org.springframework
六月 13, 2014 10:28:53 上午 org.springframework.core.io.su
信息: Loading properties file from class path resource [i
moduleName=资费管理
pageSize=5
username=scott
password=tiger

```

图 - 5

控制台输出如图-5 所示的信息，说明获取到了注入的基本属性值。

### 步骤七：为 MessageBean 添加集合属性，用于注入集合测试

1. 修改类 MessageBean，在该类中添加如下加粗代码中的四个属性，并生成这几个属性对应的 getter 和 setter 方法；最后在 execute 方法获取这几个属性的信息，代码如下所示：

```

package org.tarena.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class MessageBean {
    private String moduleName;
    private int pageSize;
    private String username;
    private String password = "";

private List<String> someList;
private Set<String> someSet;
private Map<String, Object> someMap;
private Properties someProps;

    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;
}

public List<String> getSomeList() {
    return someList;
}

public void setSomeList(List<String> someList) {
    this.someList = someList;
}

public Set<String> getSomeSet() {
    return someSet;
}

public void setSomeSet(Set<String> someSet) {
    this.someSet = someSet;
}

public Map<String, Object> getSomeMap() {
    return someMap;
}

public void setSomeMap(Map<String, Object> someMap) {
    this.someMap = someMap;
}

public Properties getSomeProps() {
    return someProps;
}

public void setSomeProps(Properties someProps) {
    this.someProps = someProps;
}

public String execute(){
    System.out.println("moduleName="+moduleName);
    System.out.println("pageSize="+pageSize);
    System.out.println("username="+username);
    System.out.println("password="+password);

    System.out.println("---List 信息如下---");
    for(String s : someList){
        System.out.println(s);
    }
    System.out.println("---Set 信息如下---");
    for(String s : someSet){
        System.out.println(s);
    }
    System.out.println("---Map 信息如下---");
    Set<String> keys = someMap.keySet();
    for(String key : keys){
}

```

```

        System.out.println(key+"="
                           +someMap.get(key));
    }
    System.out.println("----Properties 信息如下---");
    Set<Object> keys1 = someProps.keySet();
    for(Object key : keys1){
        System.out.println(key+"="
                           +someProps.getProperty(key.toString()));
    }

    return "success";
}

public String getModuleName() {
    return moduleName;
}

public void setModuleName(String moduleName) {
    this.moduleName = moduleName;
}

public int getPageSize() {
    return pageSize;
}

public void setPageSize(int pageSize) {
    this.pageSize = pageSize;
}
}

```

2. 修改 applicationContext.xml 文件，为 MessageBean 的四个属性注入集合参数值，代码如图-6 所示：



```

<!-- 配置属性 -->
<property name="password" value="tiger"></property>
<property name="someList">
    <list>
        <value>北京</value>
        <value>上海</value>
        <value>广州</value>
    </list>
</property>
<property name="someSet">
    <set>
        <value>James Gosling</value>
        <value>Martin Fowler</value>
        <value>Larry Ellison</value>
    </set>
</property>
<property name="someMap">
    <map>
        <entry key="1001" value="Java语言基础"></entry>
        <entry key="1002" value="Java Web基础"></entry>
        <entry key="1003" value="Spring使用基础"></entry>
    </map>
</property>
<property name="someProps">
    <props>
        <prop key="username">root</prop>
        <prop key="password">1234</prop>
    </props>
</property>
</bean>

```

图 - 6

3. 运行 Test1 类 , 控制台输出结果如图-7 所示 :

```

Console
<terminated> Test1 (2) [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe
六月 13, 2014 10:57:31 上午 org.springframework.context.support.AbstractApplicationContext publishEvent
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1c3...
六月 13, 2014 10:57:31 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
信息: Loading XML bean definitions from class path resource [applicationContext.xml]
六月 13, 2014 10:57:31 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
信息: Pre-instantiating singletons in org.springframework.beans.factory.xml.XmlBeanDefinitionReader
六月 13, 2014 10:57:31 上午 org.springframework.core.io.support.PropertiesLoaderUtils loadProperties
信息: Loading properties file from class path resource [db.properties]
实例化ExampleBean
初始化ExampleBean对象
moduleName=资费管理
pageSize=5
username=scott
password=tiger
---List信息如下---
北京
上海
广州
---Set信息如下---
James Gosling
Martin Fowler
Larry Ellison
---Map信息如下---
1001=Java语言基础
1002=Java Web基础
1003=Spring使用基础
---Properties信息如下---
password=1234
username=root

```

图 - 7

控制台输出如图-7 所示的信息 , 说明只需要通过配置 Spring 就可以为 Bean 注入集合参数值。

#### 步骤八：测试引用方式集合注入

1. 为 Spring 配置文件 applicationContext.xml 增加如下配置 , 采用引用方式注入集合对象 , 代码参考如下 :

```

<!-- 定义集合 Bean -->
<util:list id="oneList">
    <value>Tom</value>
    <value>Andy</value>
</util:list>
<util:set id="oneSet">
    <value>James Gosling</value>
    <value>Martin Fowler</value>
</util:set>
<util:map id="oneMap">
    <entry key="1001" value="Java 语言基础"></entry>
    <entry key="1002" value="Java Web 基础"></entry>
</util:map>
<util:properties id="oneProps">
    <prop key="username">root</prop>
    <prop key="password">1234</prop>

```

```

</util:properties>
<!-- 引用方式注入集合属性 -->
<bean id="messagebean2" class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="资费管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="andy"></property>
    <property name="password" value="123"></property>
    <!-- 引用方式注入集合属性 -->
    <property name="someList" ref="oneList" />
    <property name="someSet" ref="oneSet" />
    <property name="someMap" ref="oneMap" />
    <property name="someProps" ref="oneProps" />
</bean>

```

## 2. 增加 Test2 类测试如上配置 , Test2 代码如下:

```

package org.tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.bean.MessageBean;

/** 引用方式注入集合 */
public class Test2 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        MessageBean bean = ac.getBean("messagebean2",MessageBean.class);
        bean.execute();
    }
}

```

## 3. 执行 Test2 有如下结果 , 如图-8 所示:



```

Console X Servers SQL Results Web Browser Problems
<terminated> Test2 (1) [Java Application] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe
2014-8-11 17:13:10 org.springframework.context.support.AbstractApplicationContext: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d3f33: startup date [2014-08-11T17:13:10+08:00]; root of context hierarchy
2014-8-11 17:13:10 org.springframework.beans.factory.xml.XmlBeanDefinitionReader: Loading XML bean definitions from class path resource [db.properties]
2014-8-11 17:13:10 org.springframework.beans.factory.support.DefaultListableBeanFactory: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1d3f33: this factory has 1 beans which are defined by XML and 0 beans which are defined internally
2014-8-11 17:13:10 org.springframework.core.io.support.PropertiesLoaderUtils: Loading properties file from class path resource [db.properties]
实例化ExampleBean
初始化ExampleBean对象
moduleName=资费管理
pageSize=5
username=andy
password=123
---List信息如下---
Tom
Andy
---Set信息如下---
James Gosling
Martin Fowler
---Map信息如下---
1001=Java语言基础
1002=Java Web基础
---Properties信息如下---
password=1234
username=root

```

图 - 8

这个结果说明，通过引用方式也可以为 bean 注入集合属性。

### 步骤九：利用 Spring 表达式注入属性值

1. 在工程的 src 下，新建属性文件 db.properties，文件内容如下：

```
user=scott
```

2. 修改 applicationContext.xml 文件，注入 Spring 表达式值，代码如图-9 所示：

```
<util:properties id="jdbcProperties"
    location="classpath:db.properties"/>

<bean id="messagebean"
    class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="资费管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="#{jdbcProperties.user}"></property>
    <property name="password" value="tiger"></property>
```

图 - 9

3. 运行 Test1 类，控制台输出结果如图-10 所示：



```
Console X
terminated> Test1 (2) [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin
六月 13, 2014 11:03:23 上午 org.springframework.context.support.ClassPathXmlApplicationContext
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@109...
六月 13, 2014 11:03:24 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader
信息: Loading XML bean definitions from class path resource [applicationContext.xml]
六月 13, 2014 11:03:24 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader
信息: Pre-instantiating singletons in org.springframework.beans.factory.xml.XmlBeanDefinitionReader
六月 13, 2014 11:03:24 上午 org.springframework.core.io.support.PropertiesLoaderSupport
信息: Loading properties file from class path resource [db.properties]
实例化ExampleBean
初始化ExampleBean对象
moduleName=资费管理
pageSize=5
username=scott
password=tiger
---List信息如下---
北京
上海
广州
---Set信息如下---
James Gosling
Martin Fowler
Larry Ellison
---Map信息如下---
1001=Java语言基础
1002=Java Web基础
1003=Spring使用基础
---Properties信息如下---
password=1234
username=root
```

图 - 10

控制台输出如图-10 所示的信息，说明获取到了注入的 Spring 表达式值。

### 步骤十：注入 null 值

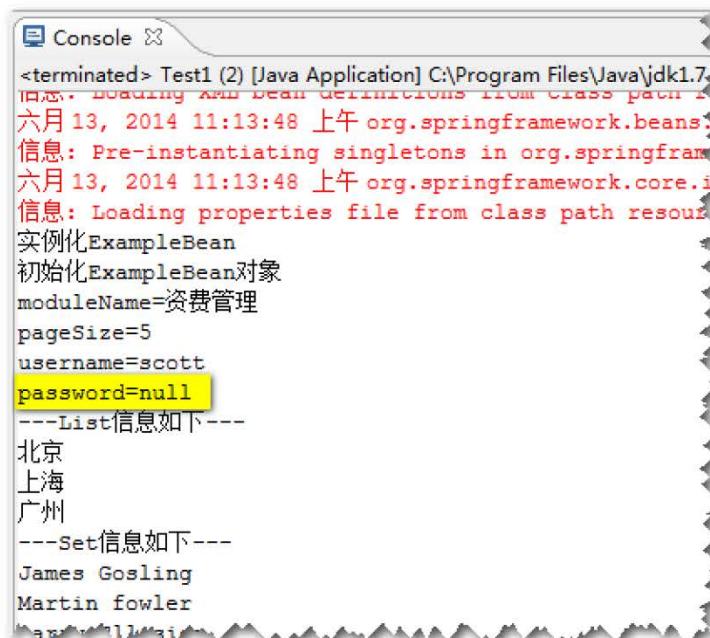
1. 修改 applicationContext.xml 文件，将属性 password 注入 null 值，代码如图

-11 所示：

```
<bean id="messagebean"
      class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="资费管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="#{jdbcProperties.user}"></property>
    <property name="password" >
      <null/>
    </property>
```

图 - 11

2. 运行 Test1 类，控制台输出结果如图-12 所示：



```
Console
<terminated> Test1 [2] [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin
信息: Loading XML Bean definitions from class path resource [org/springframework/beans/XmlBeanDefinitionReader.java:150]
六月 13, 2014 11:13:48 上午 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
信息: Pre-instantiating singletons in org.springframework.beans.factory.xml.XmlBeanDefinitionReader
六月 13, 2014 11:13:48 上午 org.springframework.core.io.support.PropertiesLoaderSupport loadProperties
信息: Loading properties file from class path resource [org/springframework/core/io/support/PropertiesLoaderSupport.java:123]
实例化ExampleBean
初始化ExampleBean对象
moduleName=资费管理
pageSize=5
username=scott
password=null
---List信息如下---
北京
上海
广州
---Set信息如下---
James Gosling
Martin Fowler
axwell
```

图 - 12

控制台输出如图-12 所示的信息，说明获取到了注入的 null 值。

### • 完整代码

MessageBean 类的完整代码如下所示：

```
package org.tarena.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class MessageBean {
    private String moduleName;
    private int pageSize;
    private String username;
    private String password = "";

    private List<String> someList;
```

```
private Set<String> someSet;
private Map<String, Object> someMap;
private Properties someProps;

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;
}

public List<String> getSomeList() {
    return someList;
}

public void setSomeList(List<String> someList) {
    this.someList = someList;
}

public Set<String> getSomeSet() {
    return someSet;
}

public void setSomeSet(Set<String> someSet) {
    this.someSet = someSet;
}

public Map<String, Object> getSomeMap() {
    return someMap;
}

public void setSomeMap(Map<String, Object> someMap) {
    this.someMap = someMap;
}

public Properties getSomeProps() {
    return someProps;
}

public void setSomeProps(Properties someProps) {
    this.someProps = someProps;
}

public String execute(){
    System.out.println("moduleName="+moduleName);
    System.out.println("pageSize="+pageSize);
    System.out.println("username="+username);
    System.out.println("password="+password);
    System.out.println("----List 信息如下---");
    for(String s : someList){
        System.out.println(s);
    }
    System.out.println("----Set 信息如下---");
    for(String s : someSet){
        System.out.println(s);
    }
    System.out.println("----Map 信息如下---");
    Set<String> keys = someMap.keySet();
}
```

```
for(String key : keys){
    System.out.println(key+"="+
        +someMap.get(key));
}
System.out.println("----Properties 信息如下---");
Set<Object> keys1 = someProps.keySet();
for(Object key : keys1){
    System.out.println(key+"="
        +someProps.getProperty(key.toString()));
}
return "success";
}

public String getModuleName() {
    return moduleName;
}

public void setModuleName(String moduleName) {
    this.moduleName = moduleName;
}

public int getPageSize() {
    return pageSize;
}

public void setPageSize(int pageSize) {
    this.pageSize = pageSize;
}
```

Test1 类的完整代码如下：

```
package org.tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.bean.MessageBean;

public class Test1 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        MessageBean bean = ac.getBean("messagebean",MessageBean.class);
        bean.execute();
    }
}
```

Test2 类的完整代码如下：

```
package org.tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.bean.MessageBean;

/** 引用方式注入集合 */
public class Test2 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        MessageBean bean = ac.getBean("messagebean2",MessageBean.class);
        bean.execute();
    }
}
```

applicationContext.xml 完整代码如下：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.2.xsd">

    <!-- 加载 properties 文件为 bean -->
    <util:properties id="jdbcProperties"
        location="classpath:db.properties" />

    <bean id="messagebean" class="org.tarena.bean.MessageBean">
        <property name="moduleName" value="资费管理"></property>
        <property name="pageSize" value="5"></property>
        <!-- 表达式注入 -->
        <property name="username" value="#{jdbcProperties.user}"></property>
        <property name="password">
            <null />
        </property>
        <property name="someList">
            <list>
                <value>北京</value>
                <value>上海</value>
                <value>广州</value>
            </list>
        </property>
        <property name="someSet">
            <set>
                <value>James Gosling</value>
                <value>Martin Fowler</value>
                <value>Larry Ellison</value>
            </set>
        </property>
        <property name="someMap">
            <map>
                <entry key="1001" value="Java 语言基础"></entry>
                <entry key="1002" value="Java Web 基础"></entry>
                <entry key="1003" value="Spring 使用基础"></entry>
            </map>
        </property>
        <property name="someProps">
            <props>
                <prop key="username">root</prop>
                <prop key="password">1234</prop>
            </props>
        </property>
    </bean>

```

```
</property>
</bean>

<!-- 定义集合 Bean -->
<util:list id="oneList">
    <value>Tom</value>
    <value>Andy</value>
</util:list>
<util:set id="oneSet">
    <value>James Gosling</value>
    <value>Martin Fowler</value>
</util:set>
<util:map id="oneMap">
    <entry key="1001" value="Java 语言基础"></entry>
    <entry key="1002" value="Java Web 基础"></entry>
</util:map>
<util:properties id="oneProps">
    <prop key="username">root</prop>
    <prop key="password">1234</prop>
</util:properties>
<!-- 引用方式注入集合属性 -->
<bean id="messagebean2" class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="财务管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="andy"></property>
    <property name="password" value="123"></property>
    <!-- 引用方式注入集合属性 -->
    <property name="someList" ref="oneList" />
    <property name="someSet" ref="oneSet" />
    <property name="someMap" ref="oneMap" />
    <property name="someProps" ref="oneProps" />
</bean>
</beans>
```

## 2. 测试 Spring 自动组件扫描方式

- 问题

如何使用组件扫描的方式和“注解标记”配合获取容器中的 ExampleBean 对象。

- 方案

使用组件扫描，首先需要在 XML 配置中指定扫描类路径，配置代码如下：

```
<context:component-scan
    base-package="com.tarena"/>
```

上述配置，容器实例化时会自动扫描 com.tarena 包及其子包下所有组件类。

指定扫描类路径后，并不是该路径下所有组件类都扫描到 Spring 容器的，只有在组件类定义前面有以下注解标记时，才会扫描到 Spring 容器，如图-13，图-14 所示：

注解标记	描述
@Component	通用注解
@Name	通用注解
@Repository	持久化层组件注解
@Service	业务层组件注解
@Controller	控制层组件注解

图 - 13

```

@Component
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化ExampleBean");
    }
}
    
```

图 - 14

- 步骤

### 步骤一：新建类 ExampleBean

新建类 ExampleBean ,在该类前使用通用组件注解 “@Component” ,表明 ExampleBean 为可被扫描的组件 ,代码如下所示 :

```

package org.tarena.bean;

import javax.annotation.PostConstruct;

@Component
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化ExampleBean");
    }

    public void execute(){
        System.out.println("执行 ExampleBean 处理");
    }
}
    
```

### 步骤二：修改 applicationContext.xml

修改 applicationContext.xml ,使容器实例化时自动扫描 org.tarena 包及其子包下所有组件类 ,代码如图-15 所示 :

```

<context:component-scan base-package="org.tarena"/>
    
```

图 - 15

### 步骤三：新建类 Test3

Test3 类的代码如图-16 所示：

```
package org.tarena.test;

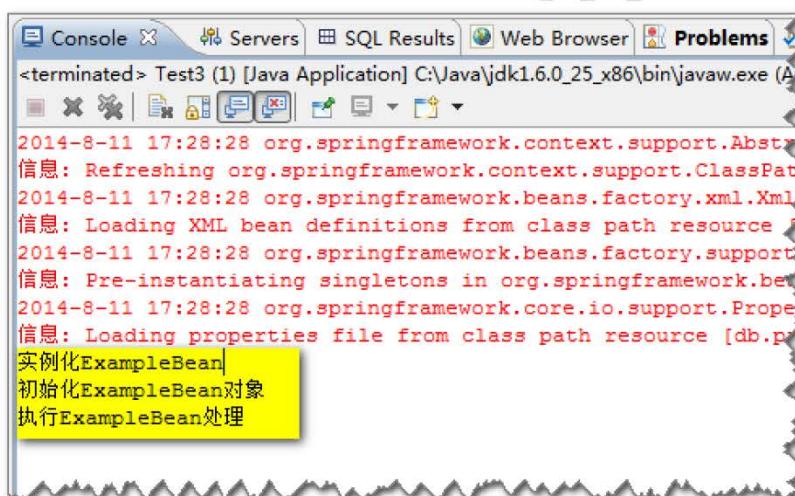
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.bean.ExampleBean;

/** 自动组件扫描方式 */
public class Test3 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        ExampleBean bean = ac.getBean("exampleBean",ExampleBean.class);
        bean.execute();
    }
}
```

图 - 16

#### 步骤四：运行 Test3 类

运行 Test3 类，控制台输出结果如图-17 所示：



```
<terminated> Test3 (1) [Java Application] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (A)
2014-8-11 17:28:28 org.springframework.context.support.AbstractApplicationContext: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d3...
2014-8-11 17:28:28 org.springframework.beans.factory.xml.XmlBeanDefinitionReader: Loading XML bean definitions from class path resource [applicationContext.xml]
2014-8-11 17:28:28 org.springframework.beans.factory.support.DefaultListableBeanFactory: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1d3...
2014-8-11 17:28:28 org.springframework.core.io.support.PropertiesLoaderUtils: Loading properties file from class path resource [db.properties]
实例化ExampleBean
初始化ExampleBean对象
执行ExampleBean处理
```

图 - 17

控制台输出上述结果，说明 Spring 会自动使用组件扫描的方式创建 ExampleBean 实例，并且 Test3 中获取到了这个 ExampleBean 对象。

- **完整代码**

ExampleBean 类的完整代码如下所示：

```
package org.tarena.bean;

import org.springframework.stereotype.Component;

@Component
public class ExampleBean {
```

```

public ExampleBean(){
    System.out.println("实例化 ExampleBean");
}

public void execute(){
    System.out.println("执行 ExampleBean 处理");
}

```

Test3 类的完整代码如下所示：

```

package org.tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.bean.ExampleBean;

public class Test3 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(conf);
        ExampleBean bean = ac.getBean("exampleBean",ExampleBean.class);
        bean.execute();
    }
}

```

applicationContext.xml 文件的完整代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.2.xsd">

    <!-- 加载 properties 文件为 bean -->
    <util:properties id="jdbcProperties"
                    location="classpath:db.properties" />

    <bean id="messagebean" class="org.tarena.bean.MessageBean">
        <property name="moduleName" value="资费管理"></property>
        <property name="pageSize" value="5"></property>
        <!-- 表达式注入 -->
        <property name="username" value="#{jdbcProperties.user}"></property>
        <property name="password">

```

```

        <null />
    </property>
    <property name="someList">
        <list>
            <value>北京</value>
            <value>上海</value>
            <value>广州</value>
        </list>
    </property>
    <property name="someSet">
        <set>
            <value>James Gosling</value>
            <value>Martin Fowler</value>
            <value>Larry Ellison</value>
        </set>
    </property>
    <property name="someMap">
        <map>
            <entry key="1001" value="Java 语言基础"></entry>
            <entry key="1002" value="Java Web 基础"></entry>
            <entry key="1003" value="Spring 使用基础"></entry>
        </map>
    </property>
    <property name="someProps">
        <props>
            <prop key="username">root</prop>
            <prop key="password">1234</prop>
        </props>
    </property>
</bean>


<util:list id="oneList">
    <value>Tom</value>
    <value>Andy</value>
</util:list>
<util:set id="oneSet">
    <value>James Gosling</value>
    <value>Martin Fowler</value>
</util:set>
<util:map id="oneMap">
    <entry key="1001" value="Java 语言基础"></entry>
    <entry key="1002" value="Java Web 基础"></entry>
</util:map>
<util:properties id="oneProps">
    <prop key="username">root</prop>
    <prop key="password">1234</prop>
</util:properties>

<bean id="messagebean2" class="org.tarena.bean.MessageBean">
    <property name="moduleName" value="财务管理"></property>
    <property name="pageSize" value="5"></property>
    <property name="username" value="andy"></property>
    <property name="password" value="123"></property>
    
    <property name="someList" ref="oneList" />
    <property name="someSet" ref="oneSet" />
    <property name="someMap" ref="oneMap" />
    <property name="someProps" ref="oneProps" />
</bean>


<context:component-scan base-package="org.tarena" />
</beans>

```

### 3. 如何控制 ExampleBean 实例化方式

- 问题

使用组件扫描的方式，重构如何控制 ExampleBean 实例化为单例或非单例模式。

- 方案

1. 通常受 Spring 管理的组件，默认的作用域是“singleton”。如果需要其他的作用域可以使用@Scope 注解，只要在注解中提供作用域的名称即可，代码如下：

```
@Component
@Scope("singleton")
public class ExampleBean {
    ...
}
```

2. @PostConstruct 和@PreDestroy 注解标记分别用于指定初始化和销毁回调方法，代码如下：

```
public class ExampleBean {
    @PostConstruct
    public void init() {
        // 初始化回调方法
    }

    @PreDestroy
    public void destroy() {
        // 销毁回调方法
    }
}
```

- 步骤

#### 步骤一：Bean 对象的创建模式

1. 修改 ExampleBean，使用 prototype 模式创建 ExampleBean 对象，修改的代码如图-18 所示：

```
@Component
@Scope("prototype")
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化ExampleBean");
    }
}
```

图 - 18

2. 新建类 Test4 在该类中创建两个 ExampleBean 对象，通过比较操作符==进行比较，代码如图-19 所示：

```

public static void main(String[] args) throws Exception{
    String conf = "applicationContext.xml";
    AbstractApplicationContext ac = new ClassPathXmlApplicationContext(conf);
    ExampleBean bean1 = ac.getBean("exampleBean",ExampleBean.class);
    ExampleBean bean2 = ac.getBean("exampleBean",ExampleBean.class);
    System.out.println(bean1==bean2);
    ac.close();
}

```

图 - 19

3. 运行 Test4，控制台输出结果如下：

```

实例化 ExampleBean
实例化 ExampleBean
false

```

通过前面的讲解，了解到默认情况下 Spring 容器是通过单例模式创建 Bean 对象的。通过本案例的运行结果，可以看出使用原型方式，调用了 2 次 ExampleBean 类的构造方法，说明创建了 2 个 ExampleBean 对象。

4. 如果想要使用单例模式创建对象，也可以使用注解的方式进行显示标注，修改 ExampleBean 类的代码如图-20 所示：

```

@Component
@Scope("singleton")
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化ExampleBean");
    }
}

```

图 - 20

5. 再次运行 Test4，控制台输出结果如下：

```

实例化 ExampleBean
true

```

上述运行结果可以看得出，两个 ExampleBean 对象，通过比较操作符 “ == ” 进行比较的输出结果为 true，说明 Spring 容器是通过单例模式创建 Bean 对象的。

## 步骤二：Bean 对象的初始化和销毁

1. 修改 ExampleBean 类，加入方法 init 和方法 destroy，并使用注解“@PostConstruct”和注解“@PreDestroy”指定 init 为初始化的回调方法、destroy 为销毁的回调方法，代码如下所示：

```
package org.tarena.bean;
```

```

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("singleton")
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化 ExampleBean");
    }

    public void execute(){
        System.out.println("执行 ExampleBean 处理");
    }

    @PostConstruct
    public void init(){
        System.out.println("初始化 ExampleBean 对象");
    }
    @PreDestroy
    public void destroy(){
        System.out.println("销毁 ExampleBean 对象");
    }
}

```

2.运行 Test4 类，控制台输出结果如图-21所示：

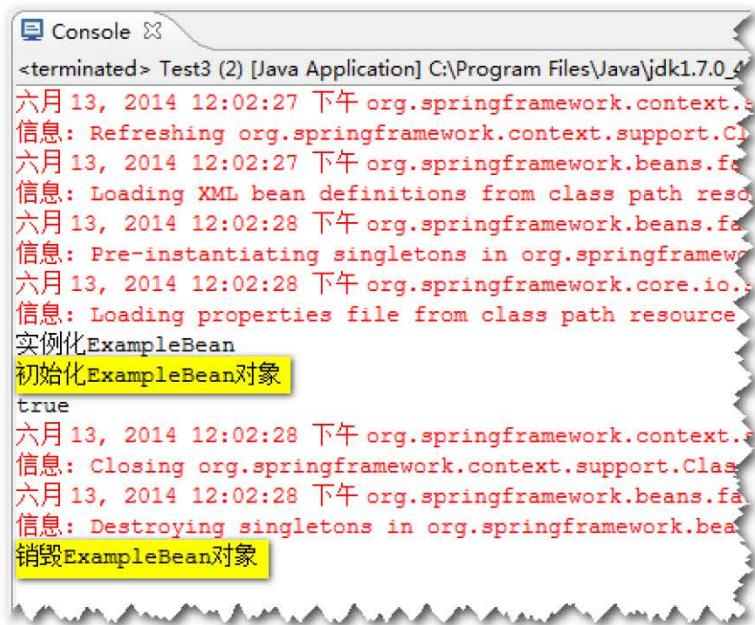


图 - 21

控制台输出了“初始化 ExampleBean 对象”和“销毁 ExampleBean 对象”，说明使用组件扫描的方式为 ExampleBean 类添加初始化和销毁的回调方法成功。

- 完整代码

ExampleBean 类的完整代码如下所示：

```
package org.tarena.bean;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("singleton")
public class ExampleBean {

    public ExampleBean(){
        System.out.println("实例化 ExampleBean");
    }

    public void execute(){
        System.out.println("执行 ExampleBean 处理");
    }

    @PostConstruct
    public void init(){
        System.out.println("初始化 ExampleBean 对象");
    }
    @PreDestroy
    public void destroy(){
        System.out.println("销毁 ExampleBean 对象");
    }
}
```

Test4 类的完整代码如下所示：

```
package org.tarena.test;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.dao.ExampleBean;

public class Test4 {
    public static void main(String[] args) throws Exception{
        String conf = "applicationContext.xml";
        AbstractApplicationContext ac =
            new ClassPathXmlApplicationContext(conf);
        ExampleBean bean1 = ac.getBean("exampleBean", ExampleBean.class);
        ExampleBean bean2 = ac.getBean("exampleBean", ExampleBean.class);
        System.out.println(bean1==bean2);
        ac.close();
    }
}
```

applicationContext.xml 文件的完整代码与前一案例相同。

## 4. 使用注解方式重构 JdbcDataSource, UserDao, UserService

- 问题

使用注解的方式解决业务服务组件 UserService 对象和数据访问层 UserDao 对象以及数据库连接组件 JdbcDataSource 之间的依赖关系，详细要求如下：

1. 使用注解方式在 Spring 中声明每个组件；
2. 使用注解方式将组件进行合理的注入解决依赖关系；
3. 使用注解将数据库连接配置信息注入到 JdbcDataSource 中。

User 类的参考代码如下：

```
package org.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public User(String name, String pwd, String phone) {
        super();
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
```

```
        this.pwd = pwd;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public int hashCode() {
        return id;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (obj instanceof User) {
            User o = (User) obj;
            return this.id == o.id;
        }
        return true;
    }

    @Override
    public String toString() {
        return id+","+name+","+pwd+","+phone;
    }
}
```

UserService 类的参考代码如下：

```
package org.tarena.service;

import org.tarena.dao.UserDao;
import org.tarena.entity.User;

/** 业务层 */
public class UserService implements Serializable {

    private UserDao userDao;

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

    public UserDao getUserDao() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd){
        try{
            User user = userDao.findByName(name);
            if(user.getPwd().equals(pwd)){
                return user;
            }
            return null;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

```
}
```

### UserDao 接口参考代码如下：

```
package org.tarena.dao;

import org.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDao {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
    // 以下是可以扩展的功能
    // public User add(String name, String pwd, String phone);
    // public User find(int id);
    // public User delete(int id);
    // public void update(User user);
}
```

### OracleUserDao 类参考代码如下：

```
package org.tarena.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.tarena.entity.User;

/** 持久层 */
public class OracleUserDao implements UserDao, Serializable {

    private JdbcDataSource dataSource;

    public OracleUserDao() {
    }

    public void setDataSource(JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
        rs.close();
        ps.close();
        return user;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }finally{
        dataSource.close(conn);
    }
}
}
```

MysqlUserDao 类参考代码如下：

```
package org.tarena.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.tarena.entity.User;

/** 持久层注解 */
public class MysqlUserDao implements UserDao , Serializable{

    private JdbcDataSource dataSource;

    public MysqlUserDao() {
    }

    public void setDataSource(JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}
```

JdbcDataSource 类参考代码如下：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/** 数据库连接源 */
public class JdbcDataSource implements Serializable{

    private String driver;
    private String url;
    private String user;
    private String pwd;

    public String getDriver() {
        return driver;
    }

    /** 必须使用 Bean 属性输入，否则不能执行 JDBC Driver 注册 */
    public void setDriver(String driver) {
        try{
            //注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public Connection getConnection() throws SQLException{
        Connection conn = DriverManager.getConnection(url, user, pwd);
        return conn;
    }

    public void close(Connection conn){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Oracle 初始化脚本参考代码如下：

```

drop table users;
CREATE TABLE USERS
(
    ID NUMBER(7, 0) ,
    NAME VARCHAR2(50) ,
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50),
    PRIMARY KEY (id),
    constraint name_unique unique(name)
);

drop SEQUENCE SEQ_USERS;
CREATE SEQUENCE SEQ_USERS;

    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');

```

**db.properties 初始化脚本参考代码如下：**

```

# config for Oracle
driver=oracle.jdbc.OracleDriver
url=jdbc:oracle:thin:@192.168.0.26:1521:XE
user=openlab
pwd=open123

# config for mysql
# driver=com.mysql.jdbc.Driver
# url=jdbc:mysql://localhost/demo
# user=root
# pwd=root

```

## • 方案

使用 Spring 注解方式实现组件的声明，并且利用注解注入方式解决依赖关系。

1. 使用@Service 将 UserService 声明为服务组件
2. 使用@Repository 将 OracleUserDao 和 MysqlUserDao 声明为存储组件
3. 使用@Component 将 JdbcDataSource 声明为通用组件
4. 使用注解@Resource 将 OracleUserDAO 对象注入到 UserService 的 userDao 属性
5. 使用注解@.Autowired 和 @Qualifier 将 JdbcDataSource 对象注入到 OracleUserDao 的 dataSource 属性
6. 使用注解 @Inject 和 @Named 将 JdbcDataSource 对象注入到 MysqlUserDao 的 dataSource 属性
7. 使用注解 @Value 将 db.properties 中的数据库连接参数注入到 JdbcDataSource 的 bean 属性中
8. 测试每个注入的结果

- 步骤

### 步骤一：新建工程，导入 jar 包

新建名为 SpringIoC\_Day02\_Part2 的 web 工程，在该工程导入如图-22 所示的 8 个 jar 包。

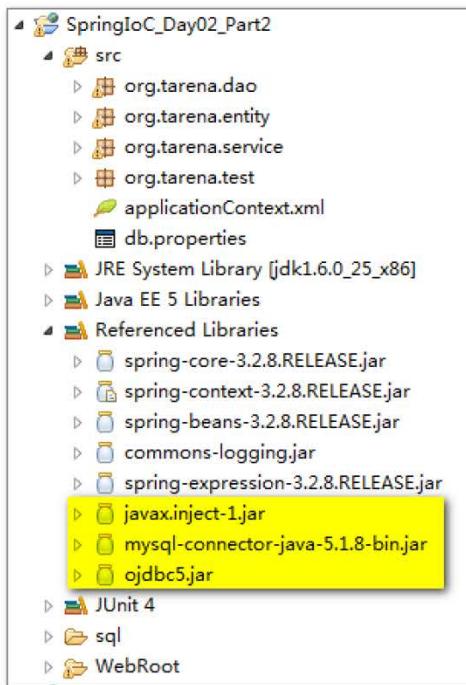


图 - 22

包含 Spring 必须的 5 个 jar 包，以及数据库驱动程序。

其中 javax.inject-1.jar 包，是 Spring3.0 开始增添的对 JSR-330 标准的支持，使用 @Inject 注解标记需要添加此 jar 包。

### 步骤二：新建 User 类，代表系统用户

代码如下所示：

```
package org.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }
}
```

```
public User(String name, String pwd, String phone) {
    super();
    this.name = name;
    this.pwd = pwd;
    this.phone = phone;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

@Override
public int hashCode() {
    return id;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (obj instanceof User) {
        User o = (User) obj;
        return this.id == o.id;
    }
    return true;
}

@Override
public String toString() {
    return id+","+name+","+pwd+","+phone;
}
}
```

### 步骤三：新建数据访问接口 UserDao

代码如下所示：

```

package org.tarena.dao;

import org.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDao {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
    // 以下是可以扩展的功能
    // public User add(String name, String pwd, String phone);
    // public User find(int id);
    // public User delete(int id);
    // public void update(User user);
}

```

#### 步骤四：新建 UserService 类，并注册为 Service 组件

新建 UserService 类 在该类实现了登录功能 login 方法 ,login 方法中调用 UserDAO 接口的 findByName 方法，代码如下所示：

```

package org.tarena.service;

import java.io.Serializable;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;
import org.tarena.dao.UserDao;
import org.tarena.entity.User;

/** 业务层 注解 */
@Service //默认的 Bean ID 是 userService
public class UserService implements Serializable {

    private UserDao userDao;

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

    public UserDao getUserDao() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd){
        try{
            User user = userDao.findByName(name);
            if(user.getPwd().equals(pwd)){
                return user;
            }
            return null;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}

```

#### 步骤五：新建 applicationContext.xml

增加配置信息，实现组件扫描功能。该文件的核心代码如图-23 所示：

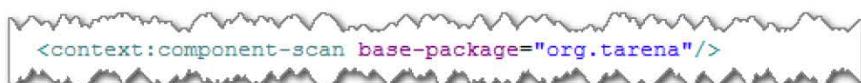


图 - 23

### 步骤六：新建测试方法 testUserService()，测试组件扫描结果

为项目导入 JUnit4 测试框架。

新建测试类 TestCase，添加测试方法 testUserService，利用 userService 作为 ID 从 Spring 获取 UserService 对象，代码如图-24 所示：

```
public class TestCase {

    /** 测试创建userService 组件 */
    @Test
    public void testUserService(){
        String config = "applicationContext.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(config);
        UserService service = ac.getBean("userService", UserService.class);
        System.out.println(service);
        User u = service.login("Tom", "123");
        System.out.println(u);
    }
}
```

图 - 24

### 步骤七：运行 testUserService 方法

运行 testUserService 方法，控制台的输出结果如图-25 所示：

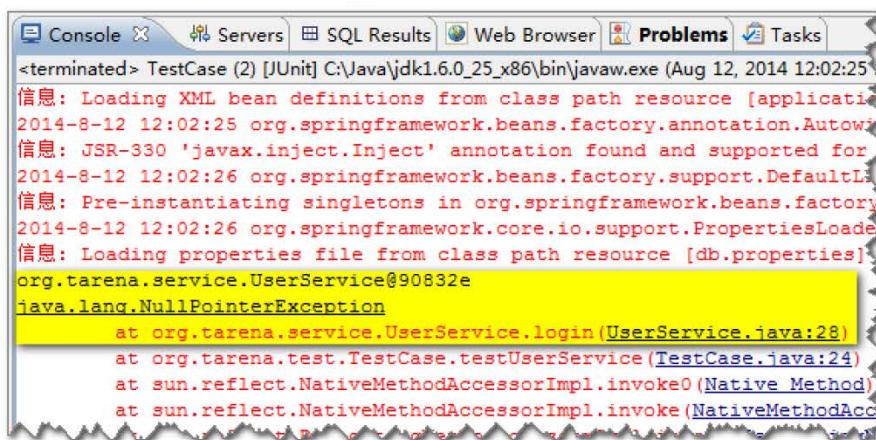


图- 25

从输出结果可以看出@Service 可以将自动参加对象注册 bean 组件，bean 组件的 ID 为 userService，可以输出这个组件。

空指针异常的原因是登录方法 login 依赖于 UserDao 对象，但是还没有注入组件，所以引用变量 userDao 的值是 null，在值为 null 的 userDao 变量上调用了 findByName 方法就抛出了空指针异常，注入 UserDao 实例即可解决。

## 步骤八：使用注解将 OracleUserDao 声明为 Bean，并且注入到 UserService 中

1. 新建 OracleUserDao 实现 UserDao 接口，在该类中实现 findByName 方法，findByName 方法调用 JdbcDataSource 的 getConnection 方法获取数据库连接。使用注解@Repository 将 Oracle 声明为存储组件，并且设置组件 ID 为 userDao，这样便于自动注入到 UserService 对象中。

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.tarena.entity.User;

/** 持久层注解 */
@Repository("userDao") //指定特定的 Bean ID 方便 setUserDAO 注入
public class OracleUserDao implements UserDao, Serializable{

    private JdbcDataSource dataSource;

    public OracleUserDao() {
    }

    public void setDataSource(JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}
```

2. 重构 UserService 类，在属性设置方法 setUserDao() 上使用注解@Resource，这

样 Spring 就会自动的将前一步声明的 userDao 对象注入到当前 UserService 对象中。 userDao 引用变量不再为 null , 调用 login 方法时不会在 findByName 方法上发生空指针异常了。代码如图-26 所示 :

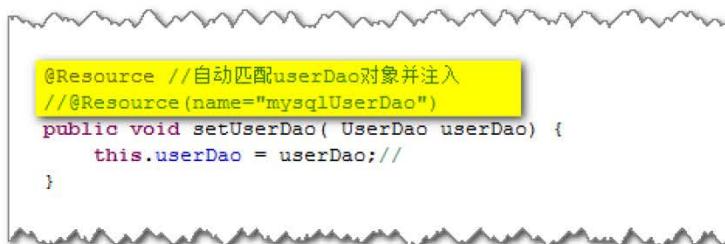


图 - 26

3. 在 TestCase 类中 , 创建测试方法 testUserDao() , 从 Spring 中获取 userDao 对象 , 以及 userService 对象 , 调用 userService 对象的 login 方法。参考代码如下 :

```

/** 测试 UserDao Bean 的创建和注入 */
@Test
public void testUserDao(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    UserDao dao =
        ac.getBean("userDao", UserDao.class);
    UserService service =
        ac.getBean("userService", UserService.class);
    //有对象被输出说明 Spring 正确的创建了 userDao 组件
    System.out.println(dao);
    //返回 true 说明 Spring 正确的将 userDao 组件注入到 service 组件中。
    System.out.println(dao == service.getUserDao());
    User u = service.login("Tom", "123");
}

```

运行 testUserDao 方法 , 控制台的输出结果如图-27 所示 :



图 - 27

### 输出结果分析：

1. 输出了 userDao 对象，说明 Spring 正确的创建了 OracleUserDao 对象；
2. 输出了 true 说明 userDao 对象被正确的注入到 userService 中的 userDao 属性；
3. 输出 null 指针异常原因是调用 login 方法时候，login 方法又调用了 findByName 方法，而 findByName 方法在 dataSource 引用上调用 getConnection() 方法获取数据库连接时候发生了异常。如果给 dataSource 属性正确注入 JdbcDataSource 对象就可以解决异常问题。

### 步骤九：使用注解@Component 将 JdbcDataSource 声明为通用组件，并且重构 JdbcDataSource，使用注解@Autowired 和@Qualifier 将 jdbcDataSource 组件注入到 dataSource 属性中

1. 新建类 JdbcDataSource 类，并且使用@Component 注册到 Spring 中，默认的 bean ID 为 jdbcDataSource，代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import org.springframework.stereotype.Component;

/** 组件注解 默认 id 为 jdbcDataSource */
@Component
public class JdbcDataSource implements Serializable{

    private String driver;
    private String url;
    private String user;
    private String pwd;

    public String getDriver() {
        return driver;
    }

    /** 必须使用 Bean 属性输入，否则不能执行 JDBC Driver 注册 */
    public void setDriver(String driver) {
        try{
            //注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
```

```

        this.user = user;
    }

    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public Connection getConnection() throws SQLException{
        Connection conn = DriverManager.getConnection(url, user, pwd);
        return conn;
    }

    public void close(Connection conn){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

2. 重构 OracleUserDao 类，在 setDataSource 方法前使用 @Autowired 和 @Qualifier 将 jdbcDataSource 组件注入到 dataSource 属性中，可以省略 @Qualifier，如省略 Spring 将自动按照类型匹配到 jdbcDataSource 对象。参考代码如下：

```

@.Autowired //按照类型自动装配
public void setDataSource(
    @Qualifier("jdbcDataSource") JdbcDataSource dataSource) {
    this.dataSource = dataSource;
}

```

3. 新建测试方法 testJdbcDataSource()，在测试方法中获取 jdbcDataSource 对象和 userDao 对象并且测试 jdbcDataSource 是否注入到 userDao 中，代码如图-28 所示：

```

/** 测试jdbcDataSource的创建和注入 */
@Test
public void testJdbcDataSource() {
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    JdbcDataSource dataSource =
        ac.getBean("jdbcDataSource", JdbcDataSource.class);
    OracleUserDao userDao = ac.getBean("userDao", OracleUserDao.class);
    System.out.println(dataSource);
    System.out.println(dataSource==userDao.getDataSource());
    User user = userDao.findByName("Tom");
}

```

图 - 28

4. 运行 testJdbcDataSource 方法，控制台的输出结果如图-29 所示：

```

Console X
<terminated> TestCase (2) [JUnit] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 12, 2014 4:12:08 PM)
2014-8-12 16:12:08 org.springframework.context.support.AbstractApplicationContext
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
2014-8-12 16:12:08 org.springframework.beans.factory.xml.XmlBeanDefinitionReader
信息: Loading XML bean definitions from class path resource [applicationContext.xml]
2014-8-12 16:12:08 org.springframework.beans.factory.annotation.AutowiredAnnotation
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
2014-8-12 16:12:08 org.springframework.beans.factory.support.DefaultListableBeanFactory
信息: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
2014-8-12 16:12:08 org.springframework.core.io.support.PropertiesLoaderSupport
信息: Loading properties file from class path resource [db.properties]
org.tarena.dao.JdbcDataSource@bcda2d
true
利用JDBC技术查找User信息
java.sql.SQLException: The url cannot be null
    at java.sql.DriverManager.getConnection(DriverManager.java:554)
    at java.sql.DriverManager.getConnection(DriverManager.java:185)
    at org.tarena.dao.JdbcDataSource.getConnection(JdbcDataSource.java:60)
    at org.tarena.dao.UserDao.findByName(OracleUserDao.java:45)

```

图 - 29

### 运行结果分析：

1. 输出了 `JdbcDataSource`，说明 Spring 自动创建 `JdbcDataSource` 对象；
2. 输出 `true` 说明 Spring 已经自动的将 `JdbcDataSource` 对象注入到了 `userDao` 的 `dataSource` 属性中；
3. 输出了空指针异常原因是 `JdbcDataSource` 中的 `url` 属性还没有注入合适的值，默认值是 `null` 的当连接数据库时候发生了空指针异常。将 `jdbc` 连接参数进行合理的注入，就可以解决问题。

### 步骤十：使用@Value 将 db.properties 文件中的 jdbc 参数注入到 JdbcDataSource 中

1. 新建 `db.properties` 文件，保存 `jdbc` 配置参数，完整代码如下所示：

```

# config for Oracle
driver=oracle.jdbc.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:XE
user=openlab
pwd=open123

# config for mysql
# driver=com.mysql.jdbc.Driver
# url=jdbc:mysql://localhost/demo
# user=root
# pwd=root

```

2. 更新 `applicationContext.xml` 文件，将 `db.properties` 读取到 Spring 中，代码片段如图-30 所示：

```

<util:properties id="jdbcProps" location="classpath:db.properties"/>

```

图 - 30

3. 重构 JdbcDataSource 类，使用@Value 和 Spring 表达式配合将 db.properties 文件中的 JDBC 配置信息注入到 JdbcDataSource 属性中，其中 driver 属性一定在 setDriver 方法上声明，否则不能正确注册 JDBC Driver。代码参考如下：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/** 组件注解 */
@Component
public class JdbcDataSource implements Serializable{

    private String driver;

    @Value("#{jdbcProps.url}")
    private String url;

    @Value("#{jdbcProps.user}")
    private String user;

    @Value("#{jdbcProps.pwd}")
    private String pwd;

    public String getDriver() {
        return driver;
    }
    /** 必须使用 Bean 属性输入，否则不能进行 JDBC Driver 注册 */
    @Value("#{jdbcProps.driver}")
    public void setDriver(String driver) {
        try{
            // 注册数据库驱动
            Class.forName(driver);
            this.driver = driver;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
    }

    public String getUrl() {
        return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }

    public String getUser() {
        return user;
    }
    public void setUser(String user) {
        this.user = user;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
}
```

```

public Connection getConnection() throws SQLException{
    Connection conn = DriverManager.getConnection(url, user, pwd);
    return conn;
}

public void close(Connection conn){
    if(conn!=null){
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

4. 在 TestCase 类中增加测试方法 testJdbcProps() , 测试 JDBC 属性的注入。参考代码如图-31 所示:

```

/** 测试Spring表达式注入 */
@Test
public void testJdbcProps() throws Exception{
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    Properties props =
        ac.getBean("jdbcProps", Properties.class);
    JdbcDataSource dataSource =
        ac.getBean("jdbcDataSource", JdbcDataSource.class);
    System.out.println(props);
    System.out.println(dataSource.getDriver());
    System.out.println(dataSource.getUrl());
    System.out.println(dataSource.getUser());
    System.out.println(dataSource.getPwd());
    Connection conn = dataSource.getConnection();
    System.out.println(conn);
}

```

图 - 31

5. 执行测试方法 testJdbcProps() , 测试结果如图-32 所示 :

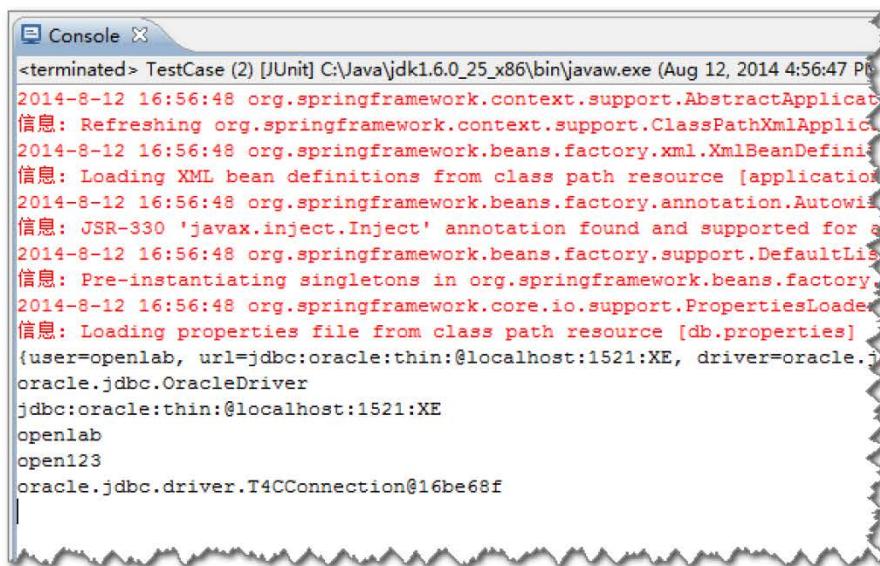


图 - 32

### 输入结果分析：

1. 输出了 `jdbcProps` 对象说明 Spring 成功的加载了 `db.properties`；
2. 输出了 `driver` `url` `user` `pwd` 4 个属性说明 Spring 通过@value 表达式将属性信息成功的注入到 `jdbcTemplate` 对象中；
3. 输出数据库连接对象，说明 `jdbcTemplate` 已经利用数据库参数正确连接到数据库。

### 步骤十一：完整测试注入配置

1. 利用 Oracle SQL 脚本在数据库中初始化适当的数据用于测试，完整的 SQL 脚本如下：

```

drop table users;
CREATE TABLE USERS
(
    ID NUMBER(7, 0) ,
    NAME VARCHAR2(50) ,
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50),
    PRIMARY KEY (id),
    constraint name_unique unique(name)
);

drop SEQUENCE SEQ_USERS;
CREATE SEQUENCE SEQ_USERS;

    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');

```

2. 在 `TestCase` 类中创建测试方法 `testLogin()`，在测试方法从 Spring 中获取 `userService` 对象，调用 `login` 方法，尝试是否能够通过数据库中的数据完成登录验证功能，并且返回 `User` 对象。测试代码如图-33：

```

/** 测试login方法的功能 */
@Test
public void testLogin(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    UserService service = ac.getBean("userService", UserService.class);
    UserDao dao = service.getUserDao();
    System.out.println(dao);
    User u = service.login("Tom", "123");
    System.out.println(u);
}

```

图 - 33

3. 执行 `testLogin` 方法，得到结果如图-34 所示：

```

Console X
<terminated> TestCase (2) [JUnit] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 12, 2014 5:34:0
2014-8-12 17:34:00 org.springframework.context.support.AbstractAppl
信息: Refreshing org.springframework.context.support.ClassPathXmlApp
2014-8-12 17:34:00 org.springframework.beans.factory.xml.XmlBeanDefi
信息: Loading XML bean definitions from class path resource [appliac
2014-8-12 17:34:01 org.springframework.beans.factory.annotation.Auto
信息: JSR-330 'javax.inject.Inject' annotation found and supported f
2014-8-12 17:34:01 org.springframework.beans.factory.support.Default
信息: Pre-instantiating singletons in org.springframework.beans.facto
2014-8-12 17:34:01 org.springframework.core.io.support.PropertiesLo
信息: Loading properties file from class path resource [db.properties]
org.tarena.dao.OracleUserDao@ae3364
利用JDBC技术查找User信息
1, Tom, 123, 110

```

图 - 34

运行结果说明：

1. 输出了 OracleUserDao 对象，说明利用 OracleUserDao 访问了数据库；
2. 输出 user 对象，说明 login 方法功能一切正常，也说明与 userService 对象相关联的注入配置正确。

#### 步骤十二：重构配置，利用 MysqlUserDao 将数据库切换到 MySQL 数据库，利用注解 @Inject 和@Named 实现属性注入

1 利用 SQL 脚本在 MySQL 数据库中生产测试数据。SQL 脚本代码如下：

```

DROP TABLE users;
CREATE TABLE `Users` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(50) NOT NULL unique ,
  `pwd` VARCHAR(50) NULL ,
  `phone` VARCHAR(50) NULL ,
  PRIMARY KEY (`id`)
);
insert into users (name, pwd, phone) values ('Tom', '123', '110');
insert into users (name, pwd, phone) values ('Jerry', 'abc', '119');
insert into users (name, pwd, phone) values ('Andy', '456', '112');

```

2. 重构 db.properties 文件，更新为 MySQL 数据库连接参数。完整代码如下所示：

```

# config for Oracle
# driver=oracle.jdbc.OracleDriver
# url=jdbc:oracle:thin:@localhost:1521:XE
# user=openlab
# pwd=open123

# config for mysql
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost/demo
user=root
pwd=root

```

3. 添加 MysqlUserDao 类实现 UserDao 接口，在 MySQL 上实现 findByName 方法，

并且使用@Repository 声明为存储组件，在 setDataSource 方法上使用@Inject 声明 dataSource 属性注入。完整代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.inject.Inject;
import javax.inject.Named;

import org.springframework.stereotype.Repository;
import org.tarena.entity.User;

/** 持久层注解 */
@Repository //指定特定的 Bean ID
public class MysqlUserDao implements UserDao, Serializable{

    private JdbcDataSource dataSource;

    public MysqlUserDao() {
    }

    @Inject
    public void setDataSource(
        @Named("jdbcDataSource") JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }
    public JdbcDataSource getDataSource() {
        return dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}
```

4. 重构 UserService 的 set UserDao() 方法指定注入 jdbc UserDao 对象，代码如图 -35 所示：



图 - 35

5. 重新执行测试案例方法 testLogin()，输出结果如图-36 所示：



图 - 36

运行结果说明：

1. 输出了 MysqlUserDao 对象，说明利用 MysqlUserDao 访问了数据库；
2. 输出 user 对象，说明 login 方法功能一切正常，也说明与 userService 对象相关联的注入配置正确。

- 完整代码

Oracle SQL 脚本完整代码如下所示：

```
drop table users;
CREATE TABLE USERS
(
    ID NUMBER(7, 0),
    NAME VARCHAR2(50),
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50),
    PRIMARY KEY (id),
    constraint name_unique unique(name)
);
```

```

drop SEQUENCE SEQ_USERS;
CREATE SEQUENCE SEQ_USERS;

    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');

```

MySQL SQL 脚本完整代码如下所示：

```

DROP TABLE users;
CREATE TABLE `Users` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(50) NOT NULL unique ,
  `pwd` VARCHAR(50) NULL ,
  `phone` VARCHAR(50) NULL ,
  PRIMARY KEY (`id`)
);

insert into users (name, pwd, phone) values ('Tom', '123', '110');
insert into users (name, pwd, phone) values ('Jerry', 'abc', '119');
insert into users (name, pwd, phone) values ('Andy', '456', '112');

```

db.properties 文件的内容如下：

```

# config for Oracle
# driver=oracle.jdbc.OracleDriver
# url=jdbc:oracle:thin:@localhost:1521:XE
# user=openlab
# pwd=open123

# config for mysql
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost/demo
user=root
pwd=root

```

applicationContext.xml 文件的完整代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.2.xsd
    " >

```

```
http://www.springframework.org/schema/util/spring-util-3.2.xsd
    http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd">

<context:component-scan base-package="org.tarena"/>

<util:properties id="jdbcProps"
    location="classpath:db.properties"/>

</beans>
```

User 类的完整代码如下所示：

```
package org.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public User(String name, String pwd, String phone) {
        super();
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String getPhone() {
        return phone;
    }
}
```

```
}

public void setPhone(String phone) {
    this.phone = phone;
}

@Override
public int hashCode() {
    return id;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (obj instanceof User) {
        User o = (User) obj;
        return this.id == o.id;
    }
    return true;
}

@Override
public String toString() {
    return id+","+name+","+pwd+","+phone;
}
}
```

UserDao 接口的完整代码如下所示：

```
package org.tarena.dao;

import org.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDao {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
    // public User add(String name, String pwd, String phone);
    // public User find(int id);
    // public User delete(int id);
    // public void update(User user);
}
```

JdbcDataSource 类的完整代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/** 组件注解 */
@Component
public class JdbcDataSource implements Serializable{
    private String driver;
```

```

@Value("${jdbcProps.url}")
private String url;

@Value("${jdbcProps.user}")
private String user;

@Value("${jdbcProps.pwd}")
private String pwd;

public String getDriver() {
    return driver;
}

/** 必须使用 Bean 属性输入，否则不能进行 JDBC Driver 注册 */
@Value("${jdbcProps.driver}")
public void setDriver(String driver) {
    try{
        //注册数据库驱动
        Class.forName(driver);
        this.driver = driver;
    }catch(Exception e){
        throw new RuntimeException(e);
    }
}

public String getUrl() {
    return url;
}

public void setUrl(String url) {
    this.url = url;
}

public String getUser() {
    return user;
}

public void setUser(String user) {
    this.user = user;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public Connection getConnection() throws SQLException{
    Connection conn = DriverManager.getConnection(url, user, pwd);
    return conn;
}

public void close(Connection conn){
    if(conn!=null){
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}

```

OralceUserDao 类的完整代码如下所示：

```
package org.tarena.dao;
```

```
import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Repository;
import org.tarena.entity.User;

/** 持久层注解 */
@Repository("userDao") //指定特定的 Bean ID 方便 setUserDAO 注入
public class OracleUserDao implements UserDao, Serializable{

    private JdbcDataSource dataSource;

    public OracleUserDao() {
    }

    /** 创建 OracleUserDAO 对象必须依赖于 JDBCDataSource 实例 */
    public OracleUserDao(JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Autowired //按照类型自动装配
    public void setDataSource(
        @Qualifier("jdbcDataSource") JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    public JdbcDataSource getDataSource() {
        return dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}
```

MysqlUserDao 类的完整代码如下所示：

```
package org.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.inject.Inject;
import javax.inject.Named;

import org.springframework.stereotype.Repository;
import org.tarena.entity.User;

/** 持久层 注解 */
@Repository //指定特定的 Bean ID
public class MysqlUserDao implements UserDao, Serializable{

    private JdbcDataSource dataSource;

    public MysqlUserDao() {
    }

    @Inject
    public void setDataSource(
        @Named("jdbcDataSource") JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }
    public JdbcDataSource getDataSource() {
        return dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}
```

UserService 类的完整代码如下所示：

```
package org.tarena.service;

import java.io.Serializable;

import javax.annotation.Resource;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;
import org.tarena.dao.UserDao;
import org.tarena.entity.User;

/** 业务层 注解 */
@Service("prototype") //非单例对象
@Service //默认的 Bean ID 是 userService
public class UserService implements Serializable {

    private UserDao userDao;

    //@Resource //自动匹配 userDao 对象并注入
    @Resource(name="mysqlUserDao")
    public void setUserDao( UserDao userDao) {
        this.userDao = userDao;//
    }

    public UserDao getUserDao() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd){
        try{
            User user = userDao.findByName(name);
            if(user.getPwd().equals(pwd)){
                return user;
            }
            return null;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

TestCase 类的完整代码如下所示：

```
package org.tarena.test;

import java.sql.Connection;
import java.util.Properties;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.tarena.dao.JdbcDataSource;
import org.tarena.dao.MysqlUserDao;
import org.tarena.dao.OracleUserDao;
import org.tarena.dao.UserDao;
import org.tarena.entity.User;
import org.tarena.service.UserService;

public class TestCase {
```

```

/** 测试创建userService 组件 */
@Test
public void testUserService(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    UserService service = ac.getBean("userService", UserService.class);
    System.out.println(service);
    User u = service.login("Tom", "123");
    System.out.println(u);
}

@Test
public void testMysqlUserDao(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    MysqlUserDao dao =
        ac.getBean("mysqlUserDao", MysqlUserDao.class);
    User user = dao.findByName("Tom");
    System.out.println(user);
    System.out.println(dao);
}

/** 测试 UserDao Bean 的创建和注入 */
@Test
public void testUserDao(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    UserDao dao =
        ac.getBean("userDao", UserDao.class);
    UserService service =
        ac.getBean("userService", UserService.class);
    //有对象被输出说明 Spring 正确的创建了 userDao 组件
    System.out.println(dao);
    //返回 true 说明 Spring 正确的将 userDao 组件注入到 service 组件中。
    System.out.println(dao == service.getUserDao());
    User u = service.login("Tom", "123");
}

/** 测试 jdbcDataSource 的创建和注入 */
@Test
public void testJdbcDataSource(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    JdbcDataSource dataSource =
        ac.getBean("jdbcDataSource", JdbcDataSource.class);
    OracleUserDao userDao = ac.getBean("userDao", OracleUserDao.class);
    System.out.println(dataSource);
    System.out.println(dataSource==userDao.getDataSource());
    User user = userDao.findByName("Tom");
}

/** 测试 Sping 表达式注入 */
@Test
public void testJDBCProps() throws Exception{
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    Properties props =
        ac.getBean("jdbcProps", Properties.class);
    JdbcDataSource dataSource =
        ac.getBean("jdbcDataSource", JdbcDataSource.class);
    System.out.println(props);
    System.out.println(dataSource.getDriver());
    System.out.println(dataSource.getUrl());
    System.out.println(dataSource.getUser());
    System.out.println(dataSource.getPwd());
    Connection conn = dataSource.getConnection();
    System.out.println(conn);
}

```

```
}

/** 测试 login 方法的功能 */
@Test
public void testLogin(){
    String config = "applicationContext.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(config);
    UserService service = ac.getBean("userService", UserService.class);
    UserDao dao = service.getUserDao();
    System.out.println(dao);
    User u = service.login("Tom", "123");
    System.out.println(u);
}

}
```

达内IT培训集团

## 课后作业

1. 编写 List、Map 和 Properties 集合类型的注入配置格式。
2. 基于组件扫描技术实现 DeptDAO 和 DeptService 的 IoC 控制，并编写测试程序。
3. 描述下列注解标记的作用。

@Component、@Repository、@Service、@Scope、@Autowired、@Inject、@Value

# Spring 核心

## Unit03

知识体系.....Page 116

Spring Web MVC 基础	Spring Web MVC 简介	MVC 模式简介
		什么是 Spring Web MVC
		Spring Web MVC 的核心组件
基于 XML 配置的 MVC 应用	Spring Web MVC 的处理流程	Spring Web MVC 的处理流程
		搭建 Spring Web MVC 环境
		DispatcherServlet 控制器配置
		HandlerMapping 组件
		Controller 组件
		ModelAndView 组件
		ViewResolver 组件
基于注解配置的 MVC 应用	@RequestMapping 注解应用	@RequestMapping 注解应用
		@Controller 注解应用

经典案例.....Page 123

构建一个 helloworld 应用案例	DispatcherServlet 控制器配置
	HandlerMapping 组件
	Controller 组件
	ModelAndView 组件
	ViewResolver 组件
重构 helloworld 应用案例	@RequestMapping 注解应用
	@Controller 注解应用

课后作业.....Page 139

## 1. Spring Web MVC 基础

### 1.1. Spring Web MVC 简介

#### 1.1.1. 【Spring Web MVC 简介】MVC 模式简介

知识讲解

#### MVC模式简介

**M-Model 模型**  
模型 ( Model ) 的职责是负责业务逻辑。包含两层：业务数据和业务处理逻辑。比如实体类、DAO、Service都属于模型层。

**V-View 视图**  
视图 ( View ) 的职责是负责显示界面和用户交互（收集用户信息）。属于视图的组件是不包含业务逻辑和控制逻辑的JSP。

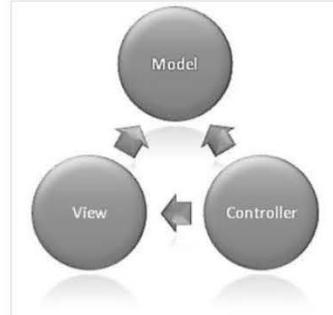
**C-Controller 控制器**  
控制器是模型层M和视图层V之间的桥梁，用于控制流程  
比如：在Servlet项目中的单一控制器ActionServlet。

+

知识讲解

#### MVC模式简介 ( 续1 )

M、V、C三部分关系如下图所示



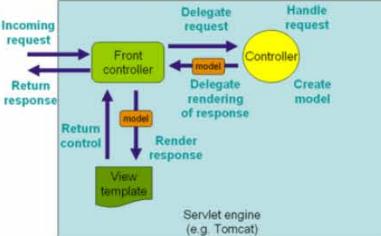
+

#### 1.1.2. 【Spring Web MVC 简介】什么是 Spring Web MVC

知识讲解

#### 什么是Spring Web MVC

Spring web MVC是Spring框架一个非常重要的功能模块。实现了MVC结构，便于简单、快速开发MVC结构的Web程序。Spring web MVC 提供的API封装了Web开发中常用的功能，简化了Web过程。



+

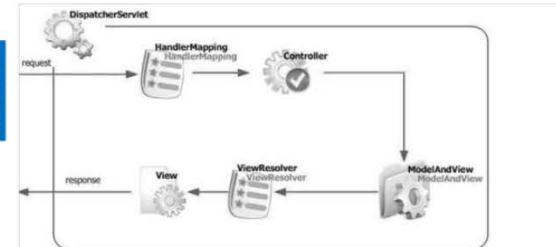
### 1.1.3. 【Spring Web MVC 简介】Spring Web MVC 的核心组件

Spring Web MVC提供了M、V和C相关的主要实现组件，具体如下

- DispatcherServlet（控制器, 请求入口）
- HandlerMapping（控制器, 请求派发）
- Controller（控制器, 请求处理流程）
- ModelAndView（模型, 封装业务处理结果和视图）
- ViewResolver（视图, 视图显示处理器）

### 1.1.4. 【Spring Web MVC 简介】Spring Web MVC 的处理流程

Spring Web MVC 的主要处理流程如下



Spring Web MVC的处理流程（续1）

- 浏览器向Spring发出请求，请求交给前端控制器DispatcherServlet处理
- 控制器通过HandlerMapping找到相应的Controller组件处理请求
- 执行Controller组件约定方法处理请求，在约定方法调用模型组件完成业务处理。约定方法可以返回一个 ModelAndView对象，封装了处理结果数据和视图名称信息
- 控制器接收ModelAndView之后，调用ViewResolver组件，定位View(JSP)并传递数据信息，生成响应界面结果

## 1.2. 基于 XML 配置的 MVC 应用

### 1.2.1. 【基于 XML 配置的 MVC 应用】搭建 Spring Web MVC 环境

知识讲解

#### 搭建Spring Web MVC环境

搭建Spring Web MVC开发环境的主要步骤如下

- 创建Web工程，导入Spring Web MVC相关开发包
  - Spring API、web、webmvc等开发包
- 在src下添加Spring 的XML配置文件
  - 名称可以自定义，例如spring-mvc.xml
- 在web.xml中配置DispatcherServlet前端控制器组件
  - DispatcherServlet组件在spring mvc中已提供，只需要配置即可
  - 配置DispatcherServlet时，同时指定XML配置文件

+

### 1.2.2. 【基于 XML 配置的 MVC 应用】DispatcherServlet 控制器配置

知识讲解

#### DispatcherServlet控制器配置

```
<server>
  <servlet-name>springmvc</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-mvc.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</server>
<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <url-pattern>*.form</url-pattern>
</servlet-mapping>
```

+

### 1.2.3. 【基于 XML 配置的 MVC 应用】HandlerMapping 组件

知识讲解

#### HandlerMapping组件

通过HandlerMapping组件，DispatcherServlet控制器可以将客户HTTP请求映射到Controller组件上。

- SimpleUrlHandlerMapping
  - 维护一个HTTP请求和Controller映射关系列表(map)，根据列表对应关系调用Controller

+

知识讲解

### HandlerMapping组件 (续1)

SimpleUrlHandlerMapping使用定义如下

```
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/login.form">loginController</prop>
            <prop key="/hello.form">helloController</prop>
        </props>
    </property>
</bean>

<bean id="helloController"
    class="org.tarena.web.HelloController"/>
```

+

知识讲解

### HandlerMapping组件 (续2)

- RequestMappingHandlerMapping  
RequestMappingHandlerAdapter
  - 在Controller类和方法上使用@RequestMapping注解  
指定对应的客户HTTP请求 (后续注解配置部分将详细介绍)

+

#### 1.2.4. 【基于 XML 配置的 MVC 应用】Controller 组件

知识讲解

### Controller组件

Controller组件负责执行具体的业务处理，可调用DAO等组件，编写时需要实现Controller接口及约定方法

```
public class HelloController implements Controller{
    public ModelAndView handleRequest(
        HttpServletRequest req,
        HttpServletResponse res) throws Exception {
        System.out.println("Hello Spring!");
        return new ModelAndView("hello");//下一个PPT
    }
}
```

+

### 1.2.5. 【基于 XML 配置的 MVC 应用】 ModelAndView 组件

**ModelAndView组件**

Tarena 达内科技

Controller组件约定的handleRequest方法执行后返回一个 ModelAndView 对象，该对象可封装模型数据和视图响应信息。 ModelAndView 构造器如下

- ModelAndView(String viewName)
- ModelAndView(String viewName, Map model)

viewName 是 jsp 页面的名字  
model 的数据存储到 request 的 attribute 中

+

### 1.2.6. 【基于 XML 配置的 MVC 应用】 ViewResolver 组件

**ViewResolver组件**

Tarena 达内科技

所有 Controller 组件都返回一个 ModelAndView 实例，封装了视图名。 Spring 中的视图以名字为标识，视图解析器 ViewResolver 通过名字来解析视图。

Spring 提供了多种视图解析器，具体如下

ViewResolver 类型	功能描述
UrlBasedViewResolver	将视图名直接解析成对应的 URL，不需要显式的映射定义。如果你的视图名和视图资源的名字是一致的，就可使用该解析器，而无需进行映射
InternalResourceViewResolver	UrlBasedViewResolver 的子类，它支持 InternalResourceView （对 Servlet 和 JSP 的包装），以及其子类 JstlView 和 TilesView 响应类型
XmlViewResolver	支持用 xml 文件定义具体的响应视图文件
VelocityViewResolver / FreeMarkerViewResolver	UrlBasedViewResolver 的子类，它能支持 Velocity 和 FreeMarker 等视图技术

+

**ViewResolver组件（续1）**

Tarena 达内科技

InternalResourceViewResolver 使用示例如下

```
<bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

如：视图名 **hello** 通过以上配置可以映射到  
/WEB-INF/jsp/hello.jsp

+

### 1.3. 基于注解配置的 MVC 应用

#### 1.3.1. 【基于注解配置的 MVC 应用】RequestMapping 注解应用

**RequestMapping注解应用**

**Tarena**  
达内科技

- @RequestMapping可以用在类定义和方法定义上
- @RequestMapping标明这个类或方法与哪一个客户请求对应

**知识讲解**

```
@Controller
@RequestMapping("/day01")
public class HelloController{
    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        return "hello";
    }
}
```

+

**RequestMapping注解应用 (续1)**

**Tarena**  
达内科技

- 开启@RequestMapping注解映射，需要在Spring的XML配置文件中定义 *RequestMappingHandlerMapping* (类定义前) 和 *RequestMappingHandlerAdapter* (方法定义前) 两个bean组件

**提示：**Spring 3.1版本之前需要定义 DefaultAnnotationHandlerMapping 和 AnnotationMethodHandlerAdapter 两个组件

+

**RequestMapping注解应用 (续2)**

**Tarena**  
达内科技

*RequestMappingHandlerMapping* 和 *RequestMappingHandlerAdapter* 两个bean组件定义示例

```
<bean
    class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

<bean
    class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>
```

+

### RequestMapping注解应用（续3）

从Spring 3.2版本开始可以使用下面XML配置简化

RequestMappingHandlerMapping和

RequestMappingHandlerAdapter定义

<mvc:annotation-driven/>

知识讲解



### 1.3.2. 【基于注解配置的 MVC 应用】Controller 注解应用

### Controller注解应用

推荐使用@Controller注解声明Controller组件，这样可以使Controller定义更加灵活，可以不用实现Controller接口，请求处理的方法也可以灵活定义

```
@Controller  
@RequestMapping("/day01")  
public class HelloController{
```

```
    @RequestMapping("/hello.form")  
    public String execute() throws Exception {  
        return "hello";  
    }  
}
```



### Controller注解应用（续1）

为了使@Controller注解生效，需要在Spring的XML配置文件中开启组件扫描定义，并指定Controller组件所在包

```
<context:component-scan  
    base-package="com.tarena.controller"/>
```



## 经典案例

### 1. 构建一个 helloworld 应用案例

- 问题

使用 Spring Web MVC 构建 helloworld Web 应用案例。

- 方案

解决本案例的方案如下：

1. 创建 Web 工程，导入 Spring Web MVC 相关开发包。

- Spring API、web、webmvc 等开发包。

2. 在 src 下添加 Spring Web MVC 的 XML 配置文件。

- 名称可以自定义，例如 spring-mvc.xml。

3. 在 web.xml 中配置 DispatcherServlet 前端控制器组件。

- DispatcherServlet 组件在 spring mvc 中已提供，只需要配置即可。
- 配置 DispatcherServlet 时，同时指定 XML 配置文件。

配置代码如下所示：

```
<servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

4. 配置 HandlerMapping 组件，通过 HandlerMapping 组件，DispatcherServlet 控制器可以将客户 HTTP 请求映射到 Controller 组件上。

使用如下配置可以实现 当客户发出 hello.form 请求时，会调用 HelloController 组件进行处理。具体的代码如下：

```
<bean class="org.springframework.web.servlet.handler.
SimpleUrlHandlerMapping">
    <property name="mappings">
```

```
<props>
    <prop key="/hello.form">helloController</prop>
</props>
</property>
</bean>
```

5. 编写 Controller 组件，它负责执行具体的业务处理，可调用 DAO 等组件，编写时需要实现 Controller 接口及约定方法，代码如下：

```
public class HelloController implements Controller {
    public ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        System.out.println("接收到请求");
        return new ModelAndView("hello");
    }
}
```

Controller 组件约定的 handleRequest 方法执行后返回一个 ModelAndView 对象，该对象可封装模型数据和视图名响应信息。

并且将 HelloController 配置到 spring-mvc.xml，代码如下：

```
<bean id="helloController"
      class="org.tarena.web.HelloController"/>
```

6. 配置视图解析器 ViewResolver。所有 Controller 组件都返回一个 ModelAndView 实例，封装了视图名。Spring 中的视图以名字为标识，视图解析器 ViewResolver 通过名字来解析视图。InternalResourceViewResolver 使用示例代码如下：

```
<bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.
      InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/">
    <property name="suffix" value=".jsp"/>
</bean>
```

- 步骤

#### 步骤一：新建 WEB 工程，并且配置 Spring MVC

1. 创建新的 Web 工程，导入 Spring API 包，和 Spring Web 相关的包，如图-1 所示：

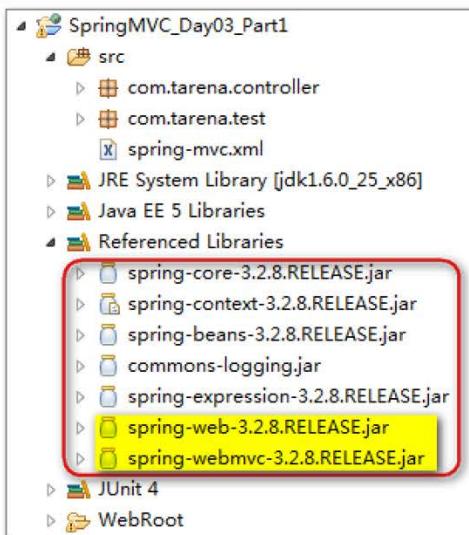


图 - 1

2. 在 src 下添加 Spring Web MVC 的 XML 配置文件，文件名为 `spring-mvc.xml`。代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.2.xsd" >

</beans>
```

3. 在 `web.xml` 中配置 `DispatcherServlet` 前端控制器组件，代码如下所示：

```
<?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/web-app_2_5.xsd">
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>
```

```

        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <!-- 指定 Spring 的配置文件 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.form</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

4. 部署项目到 Tomcat 中， 并且启动 Tomcat , 控制台输出如图-2 所示：

```

Console [ Servers ] SQL Results Web Browser Problems Tasks
tomcat6Server [Remote Java Application] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 13, 2014 6:12:52 PM)
信息: Starting service Catalina
2014-8-13 18:12:53 org.apache.catalina.core.StandardEngine startInternal
信息: Starting Servlet Engine: Apache Tomcat/7.0.6
2014-8-13 18:12:53 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory day03
2014-8-13 18:12:54 org.apache.catalina.core.ApplicationContext log
信息: Initializing Spring FrameworkServlet 'springmvc'
2014-8-13 18:12:54 org.springframework.web.servlet.FrameworkServlet initServletBean
信息: FrameworkServlet 'springmvc': initialization started
2014-8-13 18:12:54 org.springframework.context.support.AbstractApplicationContext r
信息: Refreshing WebApplicationContext for namespace 'springmvc-servlet': startup d
2014-8-13 18:12:54 org.springframework.beans.factory.xml.XmlBeanDefinitionReader l
信息: Loading XML bean definitions from class path resource [spring-mvc.xml]
2014-8-13 18:12:54 org.springframework.beans.factory.support.DefaultListableBeanFa
信息: Pre-instantiating singletons in org.springframework.beans.factory.support.Del
2014-8-13 18:12:54 org.springframework.web.servlet.FrameworkServlet initServletBe
信息: FrameworkServlet 'springmvc': initialization completed in 473 ms
2014-8-13 18:12:54 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory docs
2014-8-13 18:12:54 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory examples
2014-8-13 18:12:54 org.apache.catalina.core.ApplicationContext log
信息: ContextListener: contextInitialized()
2014-8-13 18:12:54 org.apache.catalina.core.ApplicationContext log

```

图 - 2

输出结果分析：输出结果中包含 Spring MVC 初始化信息以及加载配置文件 spring-mvc.xml 的信息。这个结果说明 Spring MVC 配置正确成功。如果有异常输出就要检查配置文件是否正确。

## 步骤二：配置 Spring HandlerMapping 组件

1. Spring HandlerMapping 用于分发 Web 请求到 Controller 的映射，这个类来自 Spring MVC API ， 只需要配置到 Spring 中即可。修改 spring-mvc.xml 添加 HandlerMapping 配置。配置代码如下：

```

<!-- 定义请求处理映射 HandlerMapping -->
<bean id="handlerMapping"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">

```

```

<!-- 指定请求和 Controller 对应关系 -->
<property name="mappings" ref="urlMappings"/>
</bean>

<!-- 定义请求映射表 map -->
<util:properties id="urlMappings">
    <prop key="/hello.form">helloController</prop>
</util:properties>

```

其中 HandlerMapping 实现类是 Spring 提供的 SimpleUrlHandlerMapping。SimpleUrlHandlerMapping 的 mappings 属性引用了 id 为 urlMappings 的 Properties 集合。

2. 为当前项目增加 JUnit4 API，并且在项目中添加测试案例类 TestCase。在 TestCase 类中添加测试方法 testHandlerMapping() 测试 HandlerMapping 配置结果，测试代码如下所示：

```

package com.tarena.test;

import java.util.Properties;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.web.servlet.HandlerMapping;

public class TestCase {
    /**
     * HandlerMapping 映射测试
     */
    @Test
    public void testHandlerMapping(){
        String cfg = "spring-mvc.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
        HandlerMapping obj = ac.getBean("handlerMapping",
                HandlerMapping.class);
        Properties map = ac.getBean("urlMappings", Properties.class);
        System.out.println(obj);
        System.out.println(map);
    }
}

```

执行测试案例，输出结果如图-3 所示：

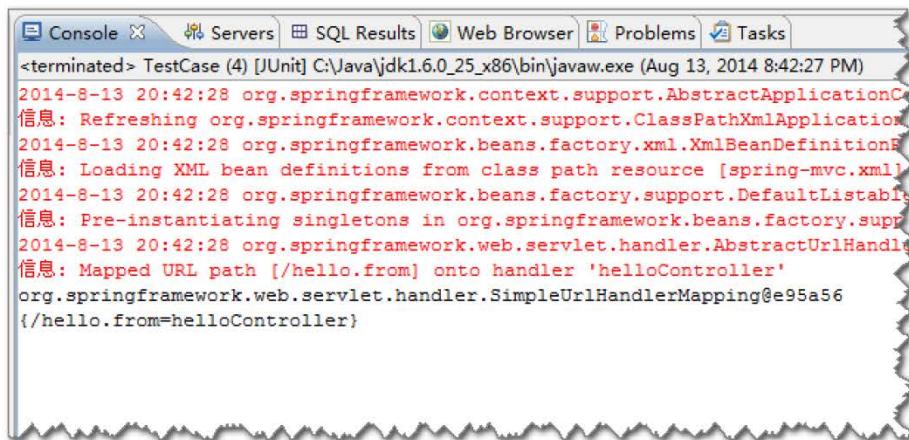


图 - 3

如果能够输出 handlerMapping 和 urlMappings 两个对象就说明这个配置上正确的。

### 步骤三：新建控制器处理类 HelloController，并且配置

1. 控制用于处理 Web URL 请求，负责执行具体的业务处理，控制器需要实现 Controller 接口，并且实现业务处理方法。控制器要返回 ModelAndView 对象，这个对象表示控制处理结果选择了那个 View ( JSP 页面作为 View ) 用于显示。控制器实现代码如下所示：

```
package com.tarena.controller;

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 HelloController implements Controller{

    public ModelAndView handleRequest(
        HttpServletRequest req,
        HttpServletResponse res) throws Exception {
        System.out.println("处理 hello.form 请求");
        ModelAndView mv = new ModelAndView("hello");
        return mv;//调用 hello.jsp
    }
}
```

2. 修改 spring-mvc.xml 文件，将控制器 HelloController 配置为一个 bean。这里要注意 bean 的 ID 要与 urlMappings 中的映射名字对应，控制器的配置如图-4 所示：



图 - 4

3. 在 TestCase 类中添加测试方法 testHelloController() 方法测试配置 bean 结果，测试方法代码如下所示：

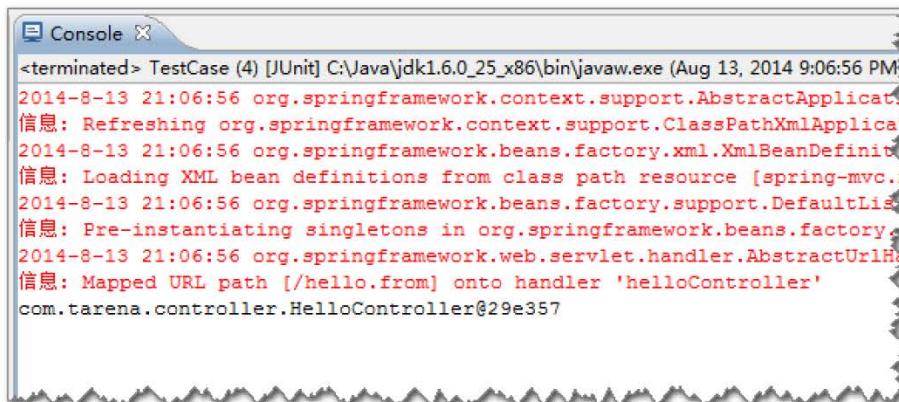
```
/** 控制器测试 */
@Test
public void testHelloController(){
    String cfg = "spring-mvc.xml";
```

```

ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
Controller obj = ac.getBean("helloController", Controller.class);
System.out.println(obj);
}

```

4. 执行测试方法，执行结果如图-5 所示：



```

Console X
<terminated> TestCase (4) [JUnit] C:\Java\jdk1.6.0_25_x86\bin\javaw.exe (Aug 13, 2014 9:06:56 PM)
2014-8-13 21:06:56 org.springframework.context.support.AbstractApplicat
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicat
2014-8-13 21:06:56 org.springframework.beans.factory.xml.XmlBeanDefinit
信息: Loading XML bean definitions from class path resource [spring-mvc.
2014-8-13 21:06:56 org.springframework.beans.factory.support.DefaultList
信息: Pre-instantiating singletons in org.springframework.beans.factory.
2014-8-13 21:06:56 org.springframework.web.servlet.handler.AbstractUrlHa
信息: Mapped URL path [/hello.from] onto handler 'helloController'
com.tarena.controller.HelloController@29e357

```

图 - 5

如果配置正确，将在控制台输出 helloController 对象。

#### 步骤四：添加 ViewResolver 组件配置

1. 在 spring-mvc.xml 文件中增加 ViewResolver 组件的配置，ViewResolver 用于视图的显示结果处理。代码如下所示：

```

<!-- 定义视图解析器 ViewResolver -->
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>

```

2. 在 TestCase 类中增加 testViewResolver() 方法，用于测试 ViewResolver 的配置结果，测试方法代码如下所示：

```

/** 测试 ViewResolver 配置 */
@Test
public void testViewResolver(){
    String cfg = "spring-mvc.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
    ViewResolver obj = ac.getBean("viewResolver", ViewResolver.class);
    System.out.println(obj);
}

```

3. 执行测试方法，执行结果如图-6 所示：



图 - 6

如果配置正确，将在控制台输出 viewResolver 对象。

#### 步骤五：新建视图组件 hello.jsp

在 viewResolver 声明的前缀文件夹/WEB-INF/jsp 中，创建 hello.jsp 页面。JSP 代码如下：

```

<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Spring Hello World!</title>
</head>
<body>
    欢迎进入 Spring Web MVC 世界!
</body>
</html>

```

#### 步骤六：测试 Spring MVC 流程

部署 SpringMVC 项目到 Tomcat6 中，在浏览器地址栏中输入 URL “<http://localhost:8080/工程名/hello.form>” 发起请求，会得到结果如图-7 所示：

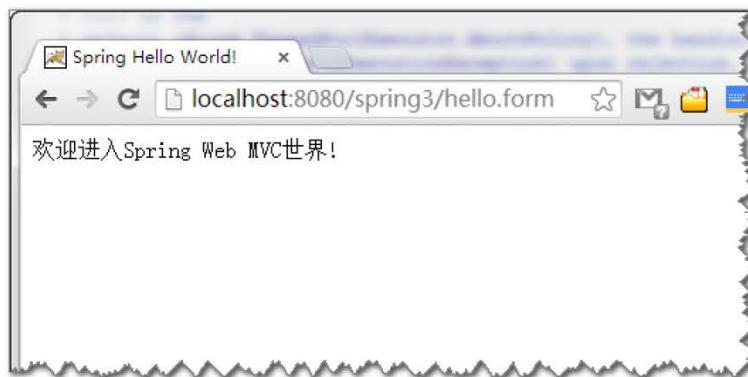


图 - 7

由图-7 可以说明 Spring MVC HelloWorld 开发部署测试成功，请求执行流程是：

1. 浏览器向 Tomcat6 服务器发起 Web 请求 “hello.form”。

2. Tomcat6 会根据 web.xml 的配置将 “hello.form” 请求交给 Spring 核心控制器 DispatcherServlet 处理。
3. DispatcherServlet 根据 HandlerMapping 中的 urlMappings 将“/hello.form”请求转给 helloController 处理。
4. helloController 执行 handlerRequest() 方法，处理请求，并且返回 ModelAndView 对象代表处理结果， ModelAndView 对象中包含了目标视图 “hello”。
5. Spring 核心控制器收到 ModelAndView 中包含的视图 “hello”。
6. Spring 核心控制器利用 ViewResolver 中的前后缀应用 “hello” 到 /WEB-INF/jsp/hello.jsp 文件。
7. 执行 hello.jsp 作为响应结果发送到浏览器。
8. 浏览器收到 hello.jsp 的执行结果：欢迎进入 Spring Web MVC 世界！

- **完整代码**

HelloController 类的完整代码如下所示：

```
package com.tarena.controller;

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 HelloController implements Controller{

    public ModelAndView handleRequest(
        HttpServletRequest req,
        HttpServletResponse res) throws Exception {
        System.out.println("处理hello.form 请求");
        ModelAndView mv = new ModelAndView("hello");
        return mv;//调用 hello.jsp
    }
}
```

TestCase 类的完整代码如下所示：

```
package com.tarena.test;

import java.util.Properties;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.web.servlet.HandlerMapping;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.mvc.Controller;

public class TestCase {
    /** HandlerMapping 映射测试 */
    @Test
    public void testHandlerMapping(){
        String cfg = "spring-mvc.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
```

```

HandlerMapping      obj      =      ac.getBean("handlerMapping",
HandlerMapping.class);
Properties map = ac.getBean("urlMappings", Properties.class);
System.out.println(obj);
System.out.println(map);
}

/** 控制器测试 */
@Test
public void testHelloController(){
    String cfg = "spring-mvc.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
    Controller obj = ac.getBean("helloController", Controller.class);
    System.out.println(obj);
}

/** 测试 ViewResolver 配置 */
@Test
public void testViewResolver(){
    String cfg = "spring-mvc.xml";
    ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
    ViewResolver obj = ac.getBean("viewResolver", ViewResolver.class);
    System.out.println("OK"+obj);
}
}

```

**spring-mvc.xml 配置代码如下：**

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.2.xsd" >

    <!-- 定义请求处理映射 HandlerMapping -->
    <bean id="handlerMapping"

          class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <!-- 指定请求和 Controller 对应关系 -->
        <property name="mappings" ref="urlMappings" />
    </bean>
    <!-- 定义 url 请求映射表 map -->
    <util:properties id="urlMappings">
        <prop key="/hello.form">helloController</prop>
    
```

```
</util:properties>

<!-- 控制器 bean 配置 -->
<bean id="helloController"
      class="com.tarena.controller.HelloController" />

<!-- 定义视图解析器 ViewResolver -->
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver"
>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>

</beans>
```

Hello.jsp 页面的完整代码如下：

```
<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE html>
<html>
<head>
<title>Spring Hello World!</title>
</head>
<body>
    欢迎进入 Spring Web MVC 世界!
</body>
</html>
```

web.xml 的完整代码如下所示：

```
<?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>
    <!-- 指定 Spring 的配置文件 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.form</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

## 2. 重构 helloworld 应用案例

- 问题

使用注解的方式重构 helloworld 应用案例。

- 方案

### 1. @RequestMapping 注解应用

@RequestMapping 可以用在类定义和方法定义上，它标明这个类或方法与哪一个客户请求对应。实例代码如下：

```
@RequestMapping("/day01")
public class HelloController{
    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        return "hello";
    }
}
```

2. 开启@RequestMapping 注解映射，需要在 Spring 的 XML 配置文件进行配置，配置代码如下：

```
<mvc:annotation-driven/>
```

### 3. @Controller 注解应用

推荐使用@Controller 注解声明 Controller 组件，这样可以使得 Controller 定义更加灵活，可以不用实现 Controller 接口，请求处理的方法也可以灵活定义。代码如下：

```
@Controller

@RequestMapping("/day01")
public class HelloController{
    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        return "hello";
    }
}
```

4. 为了使@Controller 注解生效，需要在 Spring 的 XML 配置文件中开启组件扫描定义，并指定 Controller 组件所在包，配置代码如下：

```
<context:component-scan base-package="com.tarena.controller"/>
```

- 步骤

**步骤一： 复制名为 SpringMVC\_03\_Part1 工程创建一个新工程，工程名为 SpringMVC\_03\_Part2**

选中名为 SpringMVC\_03\_Part1 工程，先按 Ctrl+C 复制快捷键，然后按 Ctrl+V 粘贴

快捷键，弹出如图-8所示提示框，将工程名改为 SpringMVC\_03\_Part2，如图-9所示：

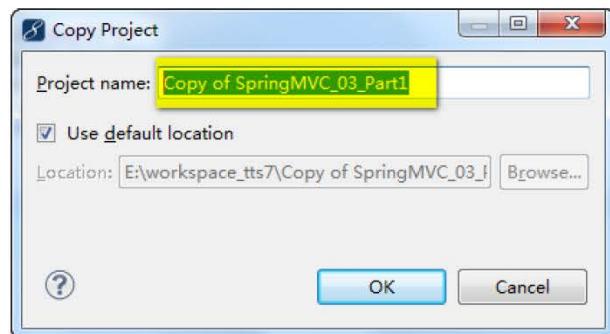


图 - 8

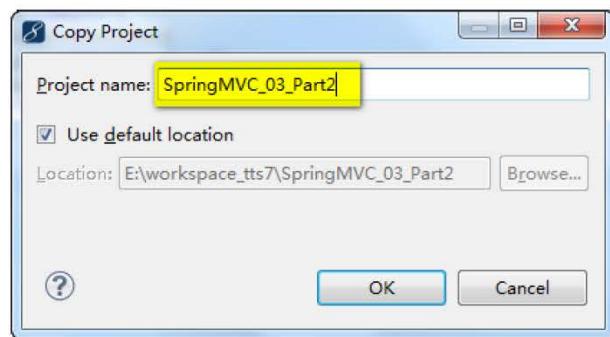


图 - 9

将工程的部署名也设置为 spring03。设置方法为：选中工程后鼠标右键，在菜单中选择 Properties 选项，会弹出图-10 所示的对话框。

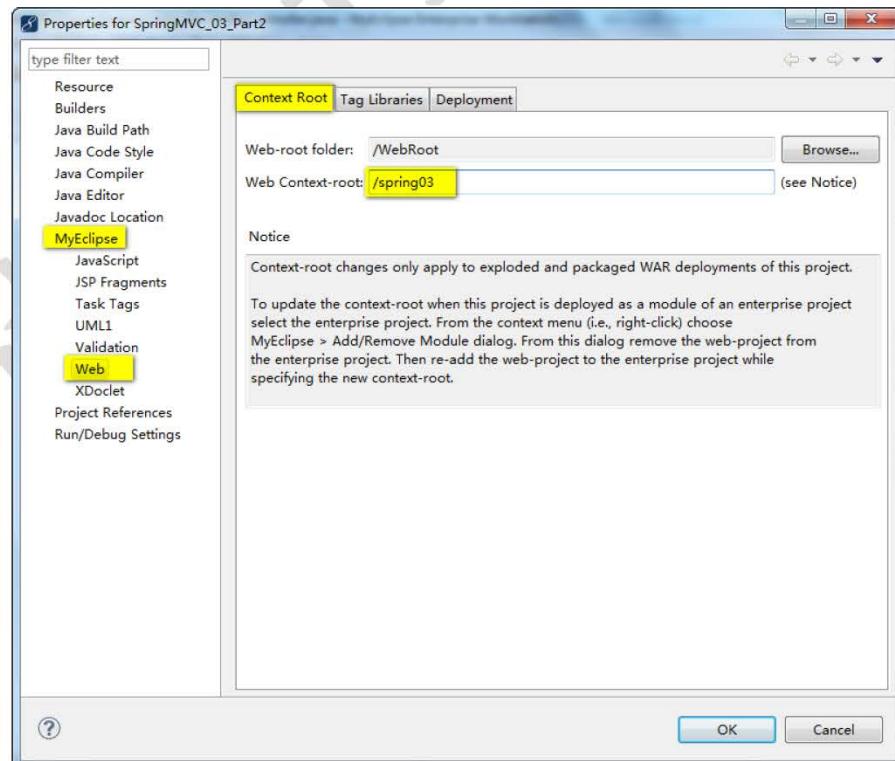


图 - 10

在图-10 的对话框中，左边列表选择 “MyEclipse” → “Web” 后，将右侧的 Web Context-root 文本框中的值修改为 spring03，点击 OK 按钮完成设置。

### 步骤二：修改 HelloController 类

使用注解的方式，修改 HelloController 类，代码如下所示：

```
package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/day01")
public class HelloController{
    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        return "hello";
    }
}
```

### 步骤三：修改 spring-mvc.xml 的配置

在 spring-mvc.xml 文件中，开启@RequestMapping 注解映射以及组件扫描，代码如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        </property>
    
```

```

<property name="suffix" value=".jsp">
</property>
</bean>

</beans>

```

#### 步骤四：测试

通过地址 “<http://localhost:8080/spring03/day01/hello.form>” 访问 HelloController，如图-11 所示：

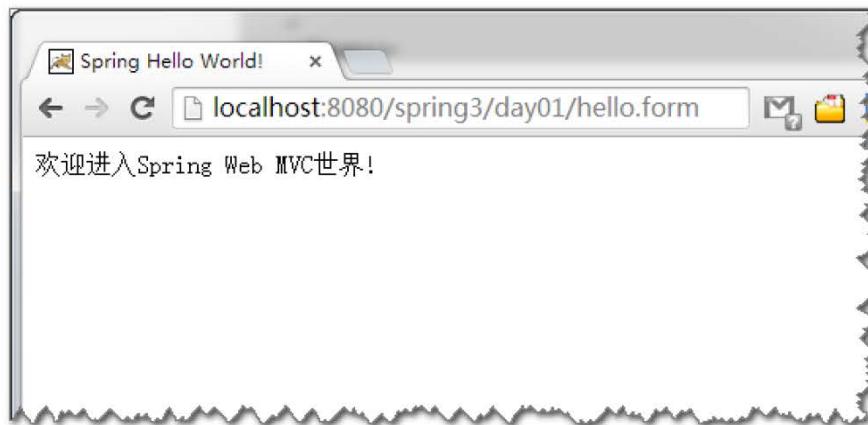


图 - 11

#### • 完整代码

HelloController 类的完整代码如下所示：

```

package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/day01")
public class HelloController{
    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        return "hello";
    }
}

```

spring-mvc.xml 文件的完整代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="

```

```
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>
    <!--
        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

        <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>
        -->
        <bean id="viewResolver"

            class="org.springframework.web.servlet.view.InternalResourceViewResolver">
            <property name="prefix" value="/WEB-INF/jsp/">
            </property>
            <property name="suffix" value=".jsp">
            </property>
        </bean>
    </beans>
```

## 课后作业

1. 描述 Spring Web MVC 的工作流程。
2. 基于注解配置实现城市列表显示功能，要求和提示如下：

- 1 ) City 城市类有 id 和 name 两个属性；
- 2 ) 自行构建 List<City>集合，并存入自定义的数据，不需要从数据库查询；
- 3 ) 在 JSP 中可以采用 JSTL 和 EL 表达式显示集合数据。

# Spring 核心

## Unit04

知识体系.....Page 141

Spring Web MVC 实战	基于注解配置的 MVC 应用--续	接收请求参数值
		向页面传值
		Session 存储
		重定向视图
	实战技巧	中文乱码解决方案
		使用拦截器实现登录检查
		异常处理
	文件上传	SpringMVC 文件上传简介
		CommonsMultipartResolver 组件
		视图表单实现
		Controller 实现
		限制上传文件大小

经典案例.....Page 153

登录案例	接收请求参数值
	向页面传值
	Session 存储
	重定向视图
为登录案例解决中文乱码问题	中文乱码解决方案
SomeInterceptor 拦截器入门示例	使用拦截器实现登录检查
为 Hello 示例添加登录检查拦截器	
为案例添加异常处理	异常处理
文件上传案例	SpringMVC 文件上传简介
	CommonsMultipartResolver 组件
	视图表单处理
	Controller 处理
限制上传文件大小案例	限制上传文件大小

课后作业.....Page 223

# 1. Spring Web MVC 实战

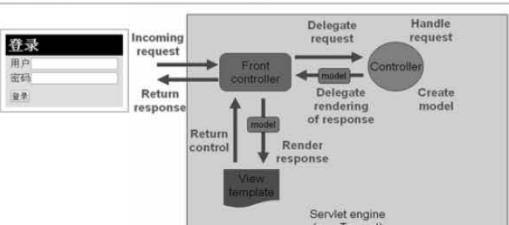
## 1.1. 基于注解配置的 MVC 应用--续

### 1.1.1. 【基于注解配置的 MVC 应用--续】接收请求参数值

**接收请求参数值**

Spring MVC Web请求提交数据到控制器有下面几种方法

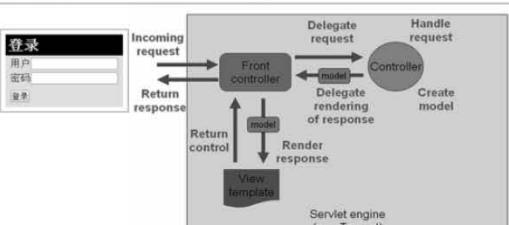
- 使用HttpServletRequest获取
- 使用@RequestParam注解
- 使用自动机制封装成Bean对象



**接收请求参数值 ( 续1 )**

- 使用HttpServletRequest获取示例
- Spring自动参数注入HttpServletRequest
- 优点直接，缺点需要自己处理数据类型转换

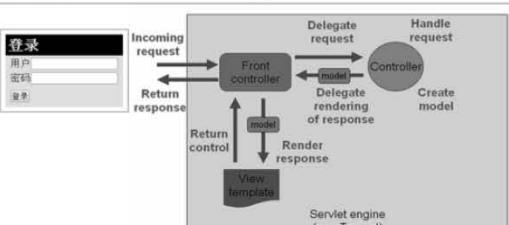
```
@RequestMapping("/login-action1.form")
public String checkLogin1(HttpServletRequest req){
    String name = req.getParameter("name");
    String pwd = req.getParameter("pwd");
    System.out.println(name);
    System.out.println(pwd);
    User user = userService.login(name, pwd);
    //... 省略处理过程
    return "success";
}
```



**接收请求参数值 ( 续2 )**

- Spring会自动将表单参数注入到方法参数(名称一致)
- 使用@RequestParam注解，映射不一致的名称
- 优点参数类型自动转换，但可能出现类型转换异常

```
@RequestMapping("/login-action2.form")
public String checkLogin2(
    String name,
    @RequestParam("pwd")String password,
    HttpServletRequest req){
    System.out.println(name);
    System.out.println(password);
    User user = userService.login(name, password);
    //... 省略处理过程
    return "success";
}
```



141

### 接收请求参数值（续3）

使用自动机制封装成Bean属性示例

- 定义User实体，属性名与<form>表单组件的name相同

```
<form action="login-action3.form">
    用户名: <input type="text" name="name"><br/>
    密码: <input type="password" name="pwd"><br/>
    <input type="submit" value="登录">
</form>

public class User {
    private String name;
    private String pwd;
    //省略setter和getter方法
}
```



### 接收请求参数值（续4）

使用自动机制封装成实体参数示例

- 在Controller组件处理方法定义User类型参数

```
@RequestMapping("/login-action3.form")
public String checkLogin3(User user){
    System.out.println(user.getName());
    System.out.println(user.getPwd());
    User u = userService.login(user.getName(), user.getPwd());
    //... 省略处理过程
    return "success";
}
```

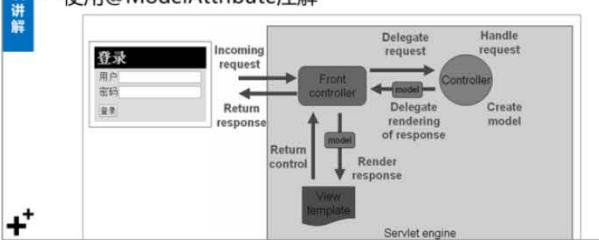


#### 1.1.2. 【基于注解配置的 MVC 应用--续】向页面传值

### 向页面传值

当Controller组件处理后，需要向JSP传值时，用下面方法

- 直接使用HttpServletRequest和Session
- 使用ModelAndView对象
- 使用ModelMap参数对象
- 使用@ModelAttribute注解



## 向页面传值（续1）

使用ModelAndView对象示例

- 在Controller处理方法完成后返回一个 ModelAndView 对象，包含显示视图名和模型数据。

```
@RequestMapping("/login-action.form")
public ModelAndView checkLogin4(String name, String pwd){
    User user = userService.login(name, pwd);
    Map<String, Object> data = new HashMap<String, Object>();
    data.put("user", user);
    return new ModelAndView("success", data);
}
```

Model数据会利用HttpServletRequest 的 Attribute 传递到JSP页面中。



## 向页面传值（续2）

使用ModelMap参数对象示例

- 在Controller处理方法中追加一个ModelMap类型参数

```
@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model){
    User user = userService.login(name, pwd);
    model.addAttribute("user", user);
    return "success";
}
```

ModelMap数据会利用HttpServletRequest 的 Attribute 传递到JSP页面中。



## 向页面传值（续3）

使用@ModelAttribute示例

- 在Controller方法的参数部分或Bean属性方法上使用

```
@RequestMapping("/login-action6.from")
public String checkLogin6(
    @ModelAttribute("user") User user){
    //TODO
    return "success";
}
@ModelAttribute("name")
public String getName(){
    return name;
}
```

@ModelAttribute数据会利用  
HttpServletRequest 的  
Attribute 传递到JSP页面中。



### 1.1.3. 【基于注解配置的 MVC 应用--续】Session 存储

#### Session存储



可以利用HttpServletRequest的getSession()方法访问

```
@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
ModelMap model, HttpServletRequest req){
User user = userService.login(name, pwd);
req.getSession().setAttribute("loginUser", user);
model.addAttribute("user", user);
return "success";
}
```



### 1.1.4. 【基于注解配置的 MVC 应用--续】重定向视图

#### 重定向视图



Spring MVC默认采用转发方式定位视图，如果需要重定向方式可采用下面几种方法

- 使用RedirectView
- 使用redirect:前缀



#### 重定向视图（续1）



如果Controller的请求处理方法返回的是 ModelAndView 对象，可以使用RedirectView方法重定向，示例代码

```
public ModelAndView checkLogin(){
//TODO
RedirectView view = new RedirectView("login.form");
return new ModelAndView(view);
}
```



知识讲解

重定向视图 ( 续2 )

如果Controller的请求处理方法返回的是String类型，可以使用“redirect:前缀”方法重定向，示例代码

```
public String checkLogin(){  
    //TODO  
    return "redirect:login.form";  
}
```

+

## 1.2. 实战技巧

### 1.2.1. 【实战技巧】中文乱码解决方案

知识讲解

中文乱码解决方案

在表单提交时，如果遇到中文字符会出现乱码现象，Spring提供了一个CharacterEncodingFilter过滤器，可用于解决乱码问题。

CharacterEncodingFilter使用时需要注意以下问题

- 表单数据以POST方式提交
- 在web.xml中配置CharacterEncodingFilter过滤器
- 页面编码和过滤器指定编码要保持一致

+

知识讲解

中文乱码解决方案 ( 续1 )

CharacterEncodingFilter配置示例

```
<filter>  
    <filter-name>encodingFilter</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>encodingFilter</filter-name>  
        <url-pattern>/*</url-pattern>  
    </filter-mapping>
```

+

### 1.2.2. 【实战技巧】使用拦截器实现登录检查

知识讲解

+  
+

### 使用拦截器实现登录检查

Spring的HandlerMapping处理器支持拦截器应用。当需要为某些请求提供特殊功能时，例如对用户进行身份认证，就非常适用。

知识讲解

+  
+

### 使用拦截器实现登录检查（续1）

拦截器必须实现HandlerInterceptor接口，这个接口有以下3个方法

- preHandle(..)
- postHandle(..)
- afterCompletion(..)

处理器执行前被调用。方法返回true表示会继续调用其他拦截器和处理器；返回false表示中断流程，不会执行后续拦截器和处理器

处理器执行后、视图处理前调用。此时可以通过modelAndView对象对模型数据进行处理或对视图进行处理

知识讲解

+  
+

### 使用拦截器实现登录检查（续2）

- afterCompletion(..)

整个请求处理完毕后调用，如性能监控中我们可以在此记录结束时间并输出消耗时间，还可以进行一些资源清理。只有preHandle返回true时才会执行afterCompletion方法

知识讲解

### 使用拦截器实现登录检查 ( 续3 )

自定义拦截器的示例代码如下

```
public class SomeInterceptor implements HandlerInterceptor{  
    public boolean preHandle(HttpServletRequest req,  
        HttpServletResponse res, Object handler)  
        throws Exception {  
        // TODO 处理器执行前调用  
        return true;  
    }  
    public void postHandle(HttpServletRequest req,  
        HttpServletResponse res, Object handler,  
        ModelAndView mv) throws Exception {  
        // TODO 处理器执行后调用  
    }  
}
```

+

知识讲解

### 使用拦截器实现登录检查 ( 续4 )

```
public void afterCompletion(HttpServletRequest req,  
    HttpServletResponse res, Object handler, Exception e)  
    throws Exception {  
    // TODO 请求完成处理后调用  
}
```

提示：自定义拦截器时，实现HandlerInterceptor接口需要实现接口定义的所有方法，如果只需要某一个方法可以继承HandlerInterceptorAdapter

+

知识讲解

### 使用拦截器实现登录检查 ( 续5 )

自定义拦截器的配置如下

```
<mvc:interceptors>  
    <mvc:interceptor>  
        <mvc:mapping path="/day04/**"/>  
        <mvc:exclude-mapping path="/login/**"/>  
        <bean class="com.tarena.interceptor.SomeInterceptor"/>  
    </mvc:interceptor>  
</mvc:interceptors>
```

+

### 1.2.3. 【实战技巧】异常处理

### 异常处理

Spring MVC处理异常方式有以下三种

- 使用Spring MVC提供的简单异常处理器 SimpleMappingExceptionResolver
- 实现HandlerExceptionResolver接口自定义异常处理器
- 使用@ExceptionHandler注解实现异常处理



### 异常处理（续1）

SimpleMappingExceptionResolver使用时，只需要在Spring的XML配置文件中定义下就可以了，定义示例如下

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
<property name="exceptionMappings">
<props>
<prop key="java.lang.Exception">error</prop>
<prop key="com.tarena.TimeoutException">login</prop>
</props>
</property>
</bean>
```

异常类型                  视图页面

异常处理页面获取异常对象名exception  
适合全局处理简单异常

### 异常处理（续2）

实现HandlerExceptionResolver接口自定义异常处理器，  
定义示例如下

```
public class MyMappingExceptionResolver
implements HandlerExceptionResolver{
public ModelAndView resolveException(
HttpServletRequest req,HttpServletResponse res,
Object handler, Exception ex) {
Map<String, Object> model =
new HashMap<String, Object>();
model.put("ex",ex);
//TODO 根据不同异常类型返回不同视图
return new ModelAndView("error", model);
}
}
```

适合全局处理有“处理过程”的异常

知识讲解

### 异常处理 ( 续3 )

自定义的异常处理器需要在Spring的XML配置文件中定义下才可应用，定义示例如下

```
<bean id="exceptionHandler"
      class="com.tarena.interceptor.MyMappingExceptionResolver"/>
```

+

知识讲解

### 异常处理 ( 续4 )

@ExceptionHandler注解实现异常处理，使用方法如下

- 首先编写一个BaseController类，定义如下

```
public class BaseController {
```

```
    @ExceptionHandler
    public String execute(
        HttpServletRequest request, Exception ex){
        request.setAttribute("ex", ex);
        //TODO 可根据异常类型不同返回不同视图名
        return "error";
    }
}
```

适合局部处理有“处理过程”的异常

- 然后其他的Controller继承BaseController类即可

+

知识讲解

### 异常处理 ( 续5 )

对于框架内部异常或代码无法捕获的异常，可以在web.xml中通过<error-page>定义，目前绝大多数服务器都支持此配置。配置示例如下

```
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/WEB-INF/views/error/500.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/views/error/404.jsp</location>
</error-page>
```

+

### 1.3. 文件上传

#### 1.3.1. 【文件上传】SpringMVC 文件上传简介

知识讲解

SpringMVC文件上传简介

在SpringMVC中，文件上传功能可以由即插即用的 CommonsMultipartResolver解析器组件实现，它定义在org.springframework.web.multipart包里。Spring 提供的CommonsMultipartResolver解析器可以支持 Commons FileUpload和COS FileUpload两种上传组件。

背景知识：  
Servlet 标准没有API支持RFC1867  
<http://www.ietf.org/>  
RFC 1945 Hypertext Transfer Protocol HTTP/1.0  
RFC 1867 Form-based File Upload in HTML  
RFC 2616 Hypertext Transfer Protocol HTTP/1.1

+

#### 1.3.2. 【文件上传】CommonsMultipartResolver 组件

知识讲解

CommonsMultipartResolver组件

CommonsMultipartResolver解析器组件可以调用 common-fileupload.jar的功能，将请求提交的文件信息解析出来，该组件使用步骤如下

- 引入common-fileupload.jar和common-io.jar开发包
- 在Spring配置文件中添加CommonsMultipartResolver 组件的bean定义

```
<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>
```

+

#### 1.3.3. 【文件上传】视图表单实现

知识讲解

视图表单实现

在JSP视图表单中，`<form>`标记必须追加 `enctype="multipart/form-data"`设置，指定表单数据的提交格式。默认情况，提交格式是application/x-www-form-urlencoded，不能用于文件上传；必须使用multipart/form-data设置才可以。  
**method属性也必须设置为post方式提交**

```
<form action="upload.form" method="post" enctype="multipart/form-data">
<input type="file" name="file" />
<input type="submit" value="Submit" />
</form>
```

+

### 1.3.4. 【文件上传】Controller 实现

知识讲解

Controller实现

在Controller处理方法中，可以使用@RequestParam注解将CommonsMultipartResolver解析出的文件赋值给MultipartFile参数对象，该对象包含了上传的文件信息。

```
@RequestMapping(value = "/upload.from")
public String upload(
    @RequestParam(value = "file", required = false) MultipartFile file,
    HttpServletRequest request, ModelMap model) {
    //TODO 将file文件对象保存到指定目录下
}
```

+

### 1.3.5. 【文件上传】限制上传文件大小

知识讲解

限制上传文件大小

CommonsMultipartResolver解析器可以设置对上传文件大小的限定，配置示例如下

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons
.CommonsMultipartResolver">
<property name="maxUploadSize" value="102400"/>
</bean>
```

+

知识讲解

限制上传文件大小（续1）

当设置了maxUploadSize大小限制后，如果上传文件大于指定大小，会抛出MaxUploadSizeExceeded异常，可以采用异常处理给客户显示友好提示。

在处理上传的Controller组件中定义@ExceptionHandler异常处理方法

+

---

---

---

---

---

---

---

---

---

## 限制上传文件大小 ( 续2 )

@ExceptionHandler异常处理示例

```
@ExceptionHandler  
public ModelAndView doException(Exception ex){  
    Map<String, Object> model = new HashMap<String, Object>();  
    if (ex instanceof MaxUploadSizeExceededException){  
        long size = ((MaxUploadSizeExceededException)ex).getMaxUploadSize();  
        model.put("errors", "文件应小于 "+size +"字节");  
    } else{  
        model.put("errors", "未知错误: " + ex.getMessage());  
    }  
    return new ModelAndView("upload", model);  
}
```

提示 : CommonsMultipartResolver的resolveLazily属性  
指定为true , 将文件解析延迟 , 才能触发上面的异常处理。



## 经典案例

### 1. 登录案例

- 问题

利用 Spring MVC 实现基于网页的登入认证过程（登录业务模块利用前面 UserService）。在该案例中，测试接收请求参数值的方式、向页面传值的方式、Session 存储以及重定向视图。

- 方案

1. 利用学习 Spring 核心阶段构建的业务层 Bean UserService 做为业务模型，这个业务模型提供了登录认证的方法，这个模型依赖 UserDao，OracleUserDao，JdbcDataSource，请参考“完整代码”。UserService 类的代码参考如下：

```
package com.tarena.service;

import java.io.Serializable;
import javax.annotation.Resource;
import org.springframework.stereotype.Service;
import com.tarena.dao.UserDao;
import com.tarena.entity.User;

/** 业务层注解 */
@Service //默认的 Bean ID 是 userService
public class UserService implements Serializable {

    private UserDao userDao;

    //@Resource //自动匹配 userDao 对象并注入
    @Resource(name="userDao")
    public void setUserDao( UserDao userDao) {
        this.userDao = userDao;//
    }

    public UserDao getUserDao() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd)
        throws NameOrPwdException, NullParamException{
        if(name == null || name.equals("") ||
           pwd==null || pwd.equals("")){
            throw new NullParamException("登录参数不能为空！");
        }
        User user = userDao.findByName(name);
        if(user != null && pwd.equals(user.getPwd())){
            return user;
        }
        throw new NameOrPwdException("用户名或者密码错误");
    }
}
```

这个代码中包含两个业务异常对象一个是 NameOrPwdException , 代码如下所示 :

```
package com.tarena.service;

/** 用户名或者密码错误 */
public class NameOrPwdException extends Exception {

    public NameOrPwdException() {
    }

    public NameOrPwdException(String message) {
        super(message);
    }

    public NameOrPwdException(Throwable cause) {
        super(cause);
    }

    public NameOrPwdException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

另外一个是 NullParamException , 代码如下所示 :

```
package com.tarena.service;
/** 参数为空 */
public class NullParamException extends Exception {

    public NullParamException() {
    }

    public NullParamException(String message) {
        super(message);
    }

    public NullParamException(Throwable cause) {
        super(cause);
    }

    public NullParamException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

2. 创建一个多表单的登录界面用于测试 Spring MVC 多种参数传递方式 , 界面参考如图-1 所示 :

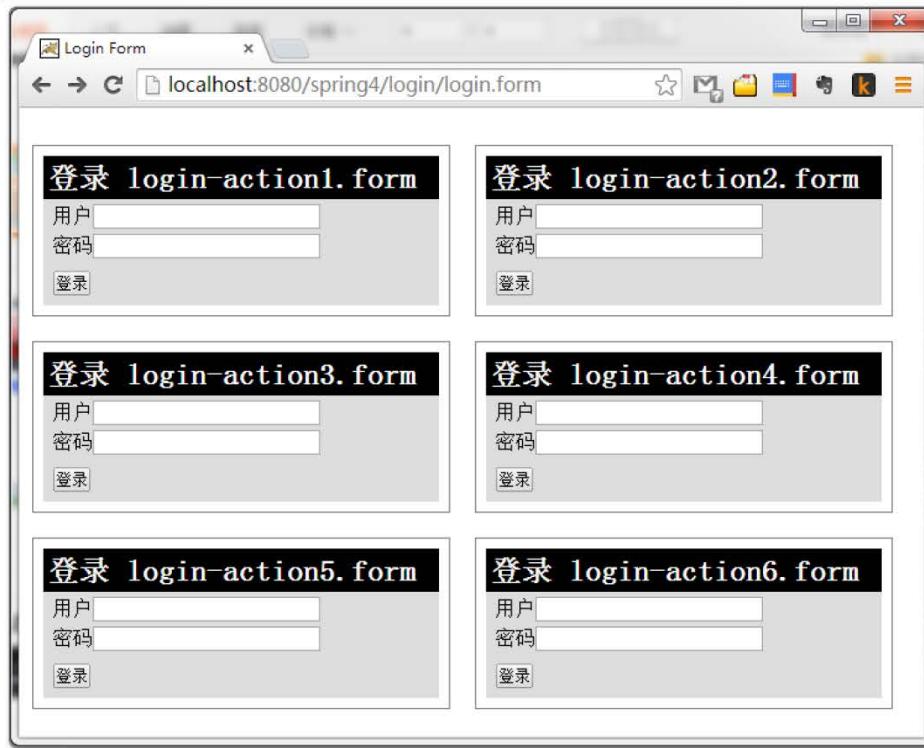


图 - 1

3. 上述页面前 3 个 Web 表单分别用于测试 Spring MVC 接收请求提交的参数值的几种方法：

- 使用 HttpServletRequest 获取。
- 使用@RequestParam 注解。
- 使用自动机制封装成实体参数。

4. 页面中后 3 个 Web 表单用于测试：当 Controller 组件处理后，需要向响应 JSP 传值时，可以使用的方法

- 使用 ModelAndView 对象
- 使用 ModelMap 参数对象
- 使用@ModelAttribute 注解
- 步骤

#### 步骤一：创建新 Web 项目，导入 Spring MVC 包和业务层 UserService

1. 创建 Web 项目导入相关的 jar 包，项目布局如图-2 所示：

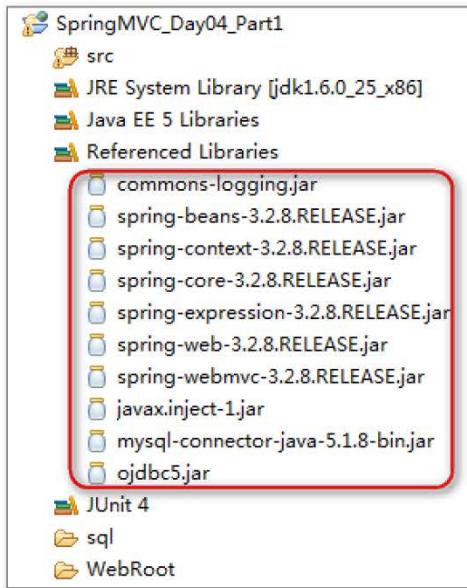


图 - 2

这里包括数据库驱动 jar 包。

## 2. 导入前述业务层 UserService 类以及依赖的类，等。

User 类代码如下：

```
package com.tarena.entity;

import java.io.Serializable;

public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
    private String phone;

    public User() {
    }

    public User(int id, String name, String pwd, String phone) {
        this.id = id;
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public User(String name, String pwd, String phone) {
        super();
        this.name = name;
        this.pwd = pwd;
        this.phone = phone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPwd() {
        return pwd;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public int hashCode() {
        return id;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (obj instanceof User) {
            User o = (User) obj;
            return this.id == o.id;
        }
        return true;
    }

    @Override
    public String toString() {
        return id+","+name+","+pwd+","+phone;
    }
}
```

### UserDao 接口代码如下：

```
package com.tarena.dao;

import com.tarena.entity.User;
/**
 * 用户数据访问对象接口
 */
public interface UserDao {
    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name);
    // public User add(String name, String pwd, String phone);
    // public User find(int id);
    // public User delete(int id);
    // public void update(User user);
}
```

### UserService 类代码如下：

```
package com.tarena.service;

import java.io.Serializable;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.tarena.dao.UserDao;
import com.tarena.entity.User;

/** 业务层 注解 */
@Service //默认的 Bean ID 是 userService
public class UserService implements Serializable {

    private UserDao userDao;

    //@Resource //自动匹配 userDao 对象并注入
    @Resource(name="userDao")
    public void setUserDao( UserDao userDao) {
        this.userDao = userDao;//
    }

    public UserDao getUserDao() {
        return userDao;
    }

    /** 登录系统功能 */
    public User login(String name, String pwd)
        throws NameOrPwdException, NullParamException{
        if(name == null || name.equals("") ||
           pwd==null || pwd.equals("")){
            throw new NullParamException("登录参数不能为空！");
        }
        User user = userDao.findByName(name);
        if(user != null && pwd.equals(user.getPwd())){
            return user;
        }
        throw new NameOrPwdException("用户名或者密码错误");
    }
}
```

### NameOrPwdException 类代码如下：

```
package com.tarena.service;

/** 用户名或者密码错误 */
public class NameOrPwdException extends Exception {

    public NameOrPwdException() {
    }

    public NameOrPwdException(String message) {
        super(message);
    }

    public NameOrPwdException(Throwable cause) {
        super(cause);
    }
}
```

```
public NameOrPwdException(String message, Throwable cause) {
    super(message, cause);
}

}
```

NullParamException 类代码如下：

```
package com.tarena.service;
/** 参数为空 */
public class NullParamException extends Exception {

    public NullParamException() {
    }

    public NullParamException(String message) {
        super(message);
    }

    public NullParamException(Throwable cause) {
        super(cause);
    }

    public NullParamException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

JdbcDataSource 类代码如下：

```
package com.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/** 组件注解 */
@Component
public class JdbcDataSource implements Serializable{

    private String driver;

    @Value("#{jdbcProps.url}")
    private String url;

    @Value("#{jdbcProps.user}")
    private String user;

    @Value("#{jdbcProps.pwd}")
    private String pwd;

    public String getDriver() {
        return driver;
    }
    /** 必须使用 Bean 属性输入，否则不能进行 JDBC Driver 注册 */
    @Value("#{jdbcProps.driver}")
    public void setDriver(String driver) {
```

```

try{
    //注册数据库驱动
    Class.forName(driver);
    this.driver = driver;
} catch(Exception e){
    throw new RuntimeException(e);
}

public String getUrl() {
    return url;
}
public void setUrl(String url) {
    this.url = url;
}

public String getUser() {
    return user;
}
public void setUser(String user) {
    this.user = user;
}

public String getPwd() {
    return pwd;
}

public void setPwd(String pwd) {
    this.pwd = pwd;
}

public Connection getConnection() throws SQLException{
    Connection conn = DriverManager.getConnection(url, user, pwd);
    return conn;
}

public void close(Connection conn){
    if(conn!=null){
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

OralceUserDao 类代码如下：

```

package com.tarena.dao;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Repository;

import com.tarena.entity.User;

/** 持久层 注解 */

```

```

@Repository("userDao") //指定特定的 Bean ID 方便 setUserDao 注入
public class OracleUserDao implements UserDao, Serializable{

    private JdbcDataSource dataSource;

    public OracleUserDao() {
    }

    /** 创建 OracleUserDAO 对象必须依赖于 JDBCDataSource 实例 */
    public OracleUserDao(JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Autowired //按照类型自动装配
    public void setDataSource(
        @Qualifier("jdbcDataSource") JdbcDataSource dataSource) {
        this.dataSource = dataSource;
    }

    public JdbcDataSource getDataSource() {
        return dataSource;
    }

    /** 根据唯一用户名查询系统用户，如果没有找到用户信息返回 null */
    public User findByName(String name) {
        System.out.println("利用 JDBC 技术查找 User 信息");
        String sql = "select id, name, pwd, phone from USERS where name=?";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, name);
            ResultSet rs = ps.executeQuery();
            User user=null;
            while(rs.next()){
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPwd(rs.getString("pwd"));
                user.setPhone(rs.getString("phone"));
            }
            rs.close();
            ps.close();
            return user;
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }finally{
            dataSource.close(conn);
        }
    }
}

```

db.properties 文件内容如下：

```

# config for Oracle
driver=oracle.jdbc.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:XE
user=openlab
pwd=open123

```

spring-mvc.xml 文件代码如下：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.2.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <util:properties id="jdbcProps" location="classpath:db.properties"/>
    <context:component-scan base-package="com.tarena"/>
    <!-- 视图处理 -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/">
        </property>
        <property name="suffix" value=".jsp">
        </property>
    </bean>
</beans>

```

Oracle 数据库初始化 SQL 代码如下：

```

drop table users;
CREATE TABLE USERS
(
    ID NUMBER(7, 0) ,
    NAME VARCHAR2(50) ,
    PWD VARCHAR2(50),
    PHONE VARCHAR2(50) ,
    PRIMARY KEY (id),
    constraint name_unique unique(name)
);

drop SEQUENCE SEQ_USERS;
CREATE SEQUENCE SEQ_USERS;

    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Tom',
'123', '110');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Jerry',
'abc', '119');
    insert into Users (id, name, pwd, phone) values (SEQ_USERS.nextval, 'Andy',
'456', '112');

```

3. 为项目添加 JUnit4 API , 然后添加测试类 TestCase 和测试方法 testUserService() 用于测试上述配置是否正确。 TestCase 类代码如下 :

```

package com.tarena.test;

import java.util.Properties;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.tarena.dao.JdbcDataSource;
import com.tarena.entity.User;
import com.tarena.service.UserService;

public class TestCase {

    @Test
    public void testUserService() throws Exception{
        String cfg = "spring-mvc.xml";
        ApplicationContext ac = new ClassPathXmlApplicationContext(cfg);
        Properties obj = ac.getBean("jdbcProps", Properties.class);
        JdbcDataSource ds = ac.getBean("jdbcDataSource", JdbcDataSource.class);
        System.out.println(obj);
        System.out.println(ds);
        System.out.println(ds.getConnection());
        UserService service = ac.getBean("userService", UserService.class);
        User user = service.login("Tom", "123");
        System.out.println(user);
    }
}

```

执行测试方法 testUserService() 结果如图 -3 所示 :

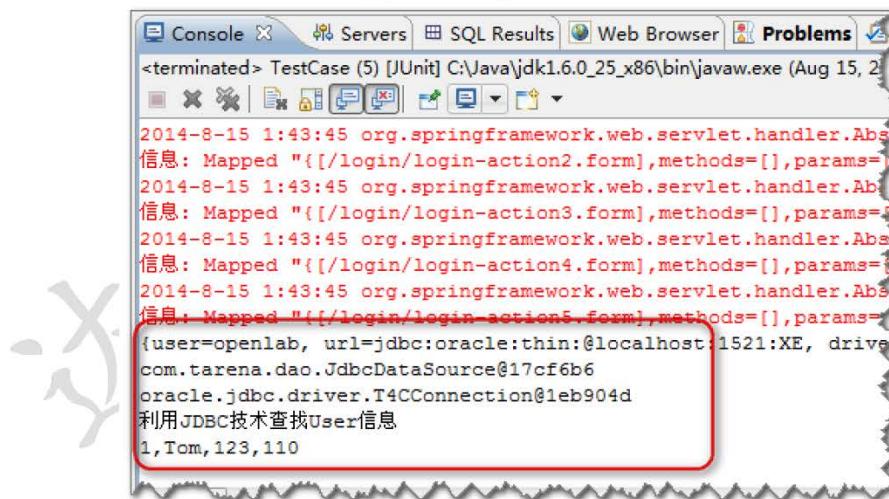


图 - 3

这个结果说明业务层 UserService 工作正常。

4. 配置 Spring MVC 核心控制器 DispatcherServlet 到 web.xml 。 web.xml 配置部分代码参考如下 :

```

<servlet>
    <servlet-name>springmvc</servlet-name>

```

```

<servlet-class>
    org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-mvc.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.form</url-pattern>
</servlet-mapping>

```

5. 部署项目到 Tomcat 并且启动，测试 Spring MVC 配置是否正常。控制台输出结果如图-4 所示：

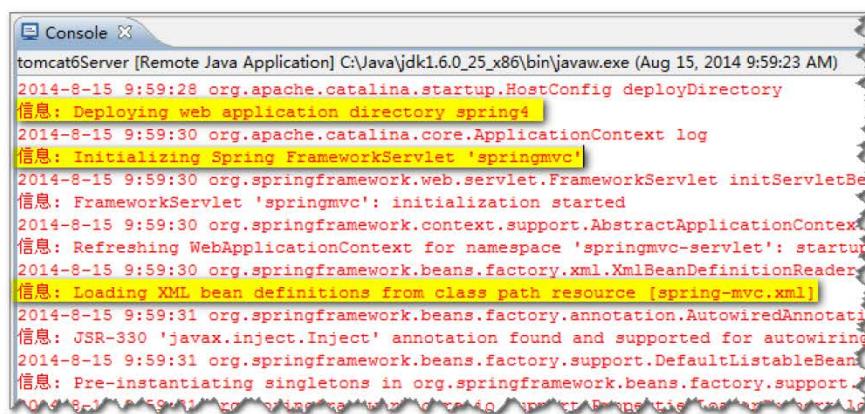


图 - 4

在输出结果中出现内容，并且没有异常就会说明 Spring MVC 部署正常。

## 步骤二：实现 login-action1.form 登录流程，测试利用 HttpServletRequest 传值方法

1. 在 WEB-INF/jsp 文件夹下添加 login-form.jsp 文件，代码如下所示：

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE HTML>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="base" value="/"></c:url>
<html>
    <head>
        <title>Login Form</title>
        <link rel="stylesheet" type="text/css" href="${base}styles.css">
    </head>
    <body>
        <h6>${message}</h6>
        <form method="post" action="${base}login/login-action1.form">
            <div>
                <h2>登录 login-action1.form</h2>
                <p><label>用户</label><input type="text" name="name"></p>
                <p><label>密码</label><input type="password" name="pwd"></p>
                <h3><input type="submit" value="登录"></h3>
            </div>
        </form>
    </body>
</html>

```

```

<form method="post" action="${base}login/login-action2.form">
  <div>
    <h2>登录 login-action2.form</h2>
    <p><label>用户</label><input type="text" name="name"></p>
    <p><label>密码</label><input type="password" name="pwd"></p>
    <h3><input type="submit" value="登录"></h3>
  </div>
</form>

<form method="post" action="${base}login/login-action3.form">
  <div>
    <h2>登录 login-action3.form</h2>
    <p><label>用户</label><input type="text" name="name"></p>
    <p><label>密码</label><input type="password" name="pwd"></p>
    <h3><input type="submit" value="登录"></h3>
  </div>
</form>

<form method="post" action="${base}login/login-action4.form">
  <div>
    <h2>登录 login-action4.form</h2>
    <p><label>用户</label><input type="text" name="name"></p>
    <p><label>密码</label><input type="password" name="pwd"></p>
    <h3><input type="submit" value="登录"></h3>
  </div>
</form>

<form method="post" action="${base}login/login-action5.form">
  <div>
    <h2>登录 login-action5.form</h2>
    <p><label>用户</label><input type="text" name="name"></p>
    <p><label>密码</label><input type="password" name="pwd"></p>
    <h3><input type="submit" value="登录"></h3>
  </div>
</form>

<form method="post" action="${base}login/login-action6.form">
  <div>
    <h2>登录 login-action6.form</h2>
    <p><label>用户</label><input type="text" name="name"></p>
    <p><label>密码</label><input type="password" name="pwd"></p>
    <h3><input type="submit" value="登录"></h3>
  </div>
</form>

</body>
</html>

```

2. 为页面添加样式文件 `styles.css`，样式文件保存在 `WebRoot` 文件夹下，`styles.css` 文件代码如下所示：

```

*{margin: 0;padding: 0;}
h2{text-align: center;color: red; padding: 10px;font-size: 14px;}
form{padding: 10px;float: left;}
form div{ border: 1px gray solid;width: 320px;padding: 8px;}

```

```

form p input{ width: 180px}
form h2 input { text-align: center; }
form h2{ background: black; color: white; padding: 4px; }
form p{ background: #ddd; padding: 4px 8px 0 8px; }
form h3{ background: #ddd; padding: 8px; }

```

3. 在 WEB-INF/jsp 文件夹下添加 success.jsp 文件，这个文件是登录成功以后显示的界面，代码如下所示：

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE HTML>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="base" value="/"></c:url>
<html>
    <head>
        <title>Success</title>
        <link rel="stylesheet" type="text/css" href="${base}styles.css">
    </head>
    <body>
        <h6>${user.name} 登录成功！</h6>
    </body>
</html>

```

上述页面中，使用 EL 表达式和标准标签库配合显示界面数据，其中<c:url var="base" value="/"></c:url>和\${base}用于解决绝对路径问题。

4. 创建控制器类 LoginController，在该类中使用注解@Controller 的方式进行配置：

- 1 ) 使用@Controller 将 LoginController 声明为控制器 Bean 组件。
- 2 ) 使用@RequestMapping("/login") 声明对 LoginController 组件的请求在 /login 路径下。
- 3 ) 流程控制方法 loginForm()，用于显示登录表单页面。使用@RequestMapping 注解将映射请求/login-form.form 到 loginForm()方法。
- 4 ) 增加 userService 属性，并且使用@Resource 注解声明在运行期间注入 userService 对象。
- 5 ) 增加控制流程方法 checkLogin1()，使用@RequestMapping 注解将请求 /login-action1.form 映射到 checkLogin1() 方法。checkLogin1() 方法调用 userService 的 login 方法，实现登录流程。checkLogin1()方法主要是测试 JSP 页面到控制器的数据传输方式。

LoginController 类代码如下所示：

```

package com.tarena.controller;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import com.tarena.entity.User;
import com.tarena.service.NameOrPwdException;
import com.tarena.service.NullParamException;
import com.tarena.service.UserService;

```

```
@Controller
@RequestMapping("/login")
public class LoginController {

    @Resource //请求 Spring 注入资源 userService
    private UserService userService;

    @RequestMapping("/login.form")
    public String loginForm(){
        //可以向表单界面传递一些参数
        return "login-form"; //映射到 login-form.jsp
    }

    @RequestMapping("/login-action1.form")
    //Spring MVC 自动参数注入 HttpServletRequest
    public String checkLogin1(HttpServletRequest req){
        System.out.println("----方法----");
        //优点直接简洁，缺点需要自己处理数据类型转换
        String name = req.getParameter("name");
        String pwd = req.getParameter("pwd");
        System.out.println(name);
        System.out.println(pwd);
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success"; //映射到 success.jsp
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }
}
```

## 5. 测试 login-action1.form 登录流程

通过网址 "<http://localhost:8080/spring04/login/login-form.form>" 请求 Tomcat 服务器，显示如下表单界面，如图-5 所示：

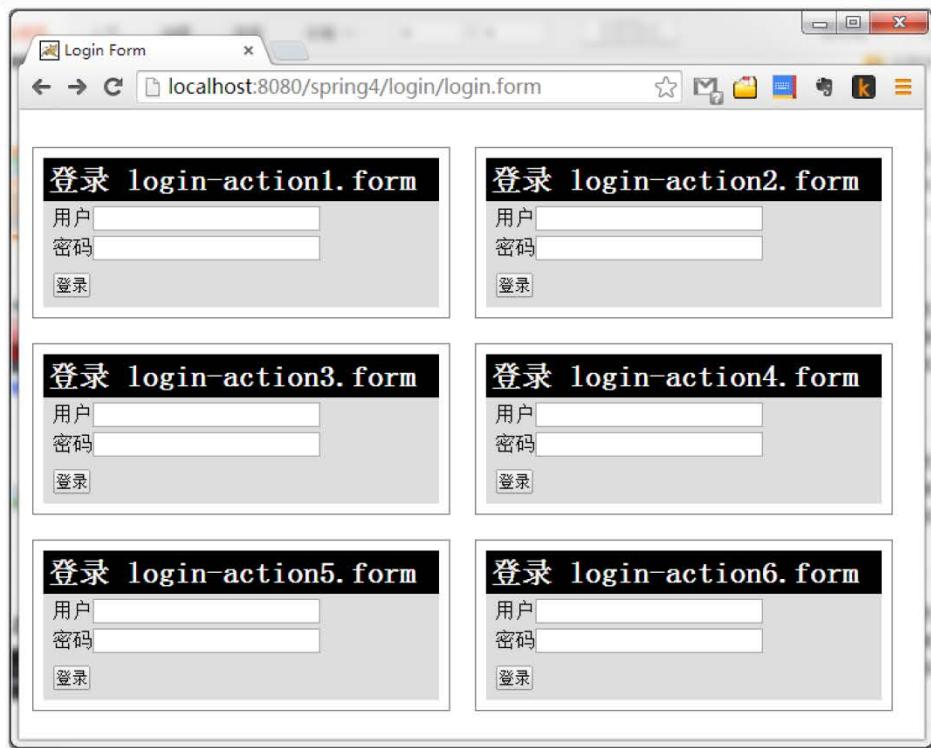


图 - 5

在“登录 login-action1.form”中填上测试数据“Tom”，“123”点击“登录”按钮提交表单，执行结果如图-6所示：

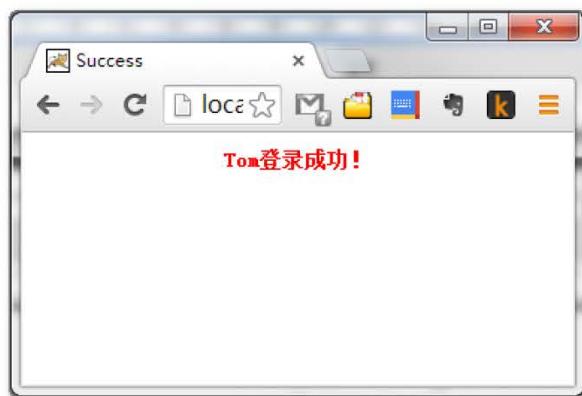


图 - 6

执行结果说明通过浏览器表单向 Spring 控制器提交数据正确能够完成完整的登录流程，也就是说可以利用 HttpServletRequest 对象进行浏览器页面到控制器传值。

### 步骤三：实现 login-action2.form 登录流程，测试使用@RequestParam 注解获取请求参数的值

1. 修改 LoginController 类，添加 checkLogin2()方法，增加的 checkLogin2()方法代码如下图-7 所示：

```

@RequestMapping("/login-action2.form")
public String checkLogin2(
    String name,
    @RequestParam("pwd") String password, //映射表单属性不同的参数
    HttpServletRequest req){
    System.out.println("---方法二---");
    //优点，自动转换数据类型，缺点可能出现数据类型转换异常
    System.out.println(name);
    System.out.println(password);
    try {
        User user = userService.login(name, password);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

```

图 - 7

在上述代码中，使用@RequestParam注解声明表单密码输入框 pwd 的值注入到password变量，表单中用户名输入框根据名字映射注入name变量。@RequestMapping注解将login-action2.form映射到了checkLogin2()方法。这样就与login-form.jsp表单对应。login-form.jsp文件上对应的表单代码如下所示：

```

<form method="post" action="${base}login/login-action2.form">
    <div>
        <h2>登录 login-action2.form</h2>
        <p><label>用户</label><input type="text" name="name"></p>
        <p><label>密码</label><input type="password" name="pwd"></p>
        <h3><input type="submit" value="登录"></h3>
    </div>
</form>

```

login-action2表单与控制器方法checkLogin2传输关系如图-8所示：



图 - 8

2. 重新部署应用程序并且启动 Tomcat , 填写 “Tom” , “123” 提交 login-action2 表单如果再次得到与前次一样的结果 , 就说明再次成功的将数据传输到 checkLogin2() 方法中。测试结果如图-9 所示 :

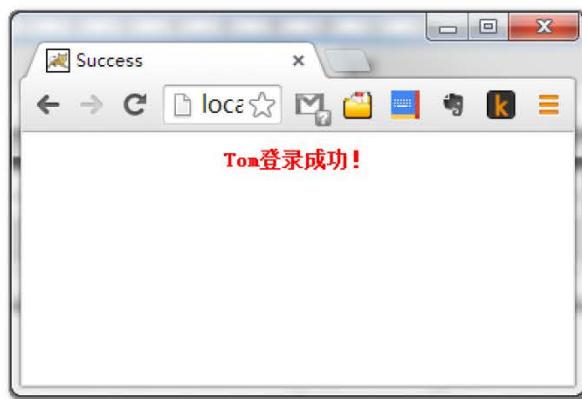


图 - 9

#### 步骤四：实现 login-action3.form 登录流程，使用自动机制封装成实体参数的方式获取请求参数的值

1. 修改 LoginController , 添加 checkLogin3() 方法 , 代码如图-10 所示 :

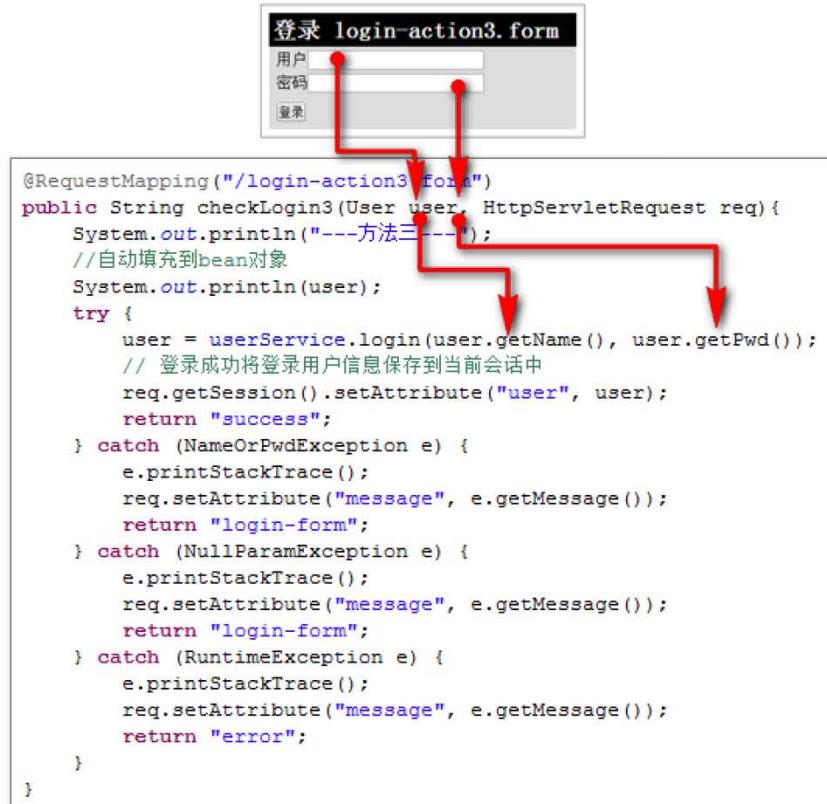
```

@RequestMapping("/login-action3.form")
public String checkLogin3(User user, HttpServletRequest req) {
    System.out.println("----方法三----");
    //自动填充到bean对象
    System.out.println(user);
    try {
        user = userService.login(user.getName(), user.getPwd());
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

```

图 - 10

这里采用 user 作为参数， Spring 会自动的将页面表单参数 name , pwd 注入到 user 对象的相应属性 name, pwd 传递到方法中。@RequestMapping 将请求 login-action3.form 映射到方法 checkLogin3()。表单与数据的对应关系如图-11 所示：



```

@RequestMapping("/login-action3.form")
public String checkLogin3(User user, HttpServletRequest req) {
    System.out.println("----方法三----");
    //自动填充到bean对象
    System.out.println(user);
    try {
        user = userService.login(user.getName(), user.getPwd());
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

```

图 - 11

2. 再次启动 Tomcat , 测试 login-action3 表单 , 如果能够正常执行也会得到正确结果。

**步骤五：实现 login-action4.form 登录流程，使用 ModelAndView 实现控制器向页面传值**

1. 修改 LoginController , 添加 checkLogin4() 方法如图-12 所示的代码：

```
@RequestMapping("/login-action4.form")
public ModelAndView checkLogin4(
    String name, String pwd, HttpServletRequest req) {
    System.out.println("----方法四---");
    Map<String, Object> data = new HashMap<String, Object>();
    try {
        User user = userService.login(name, pwd);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return new ModelAndView("success", data);
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (NullParamException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (RuntimeException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("error", data);
    }
}
```

图- 12

上述代码中，在处理方法完成后返回一个 ModelAndView 对象。

ModelAndView 返回值与页面 login-form.jsp 之间的数据对应关系如图-13 所示：

```

@RequestMapping("/login-action4.form")
public ModelAndView checkLogin4(
    String name, String pwd, HttpServletRequest req){
    System.out.println("----方法四----");
    Map<String, Object> data = new HashMap<String, Object>();
    try {
        User user = userService.login(name, pwd);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return new ModelAndView("success", data);
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (NullParamException e) {
        e.p:
        dat:
        ret:
    } catch (e.p:
        e.p:
        dat:
        ret:
    }
}

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>  
 <!DOCTYPE HTML>  
 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
 <c:url var="base" value="/" /></c:url>  
 <html>  
 <head>  
 <title>Login Form</title>  
 <link rel="stylesheet" type="text/css" href="\${base}styles.css">
 </head>  
 <body>  
 <h6>\${message}</h6>  
 <form method="post" action="\${base}login/login-action1.form">  
 <div>  
 <h2>登录 login-action1.form</h2>  
 <p><label>用户名</label><input type="text" name="name"></p>  
 <p><label>密码</label><input type="password" name="pwd"></p>  
 <h3><input type="submit" value="登录"></h3>
 </div>
 </form>
 </body>

图 - 13

2. 再次启动 Tomcat , 测试 login-action4 表单 , 但用户名或者密码错误时候出现结果如下图-14 所示 :

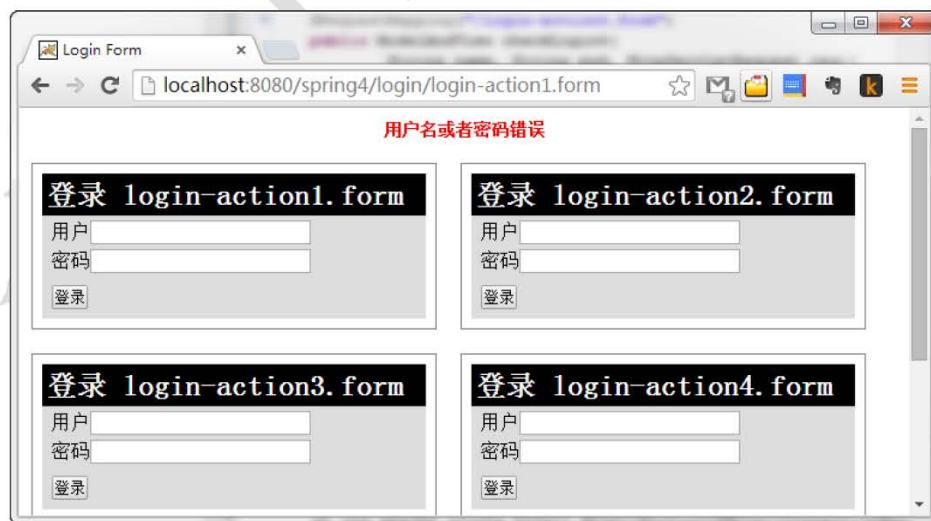


图 - 14

这个结果中红色字体消息就来自 ModelAndView 中封装的消息。

#### 步骤六：实现 login-action4.form 登录流程，使用 ModelMap 参数对象向页面传值

1. 修改 LoginController , 添加 checkLogin4() 方法 , 如图-15 所示的代码。

```

@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model, HttpServletRequest req) {
    System.out.println("----方法五---");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}

```

图- 15

ModelMap 属性值与页面 login-form.jsp 之间的数据对应关系如图-16 所示：

```

@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model, HttpServletRequest req) {
    System.out.println("----方法五---");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=UTF-8" %>
 <!DOCTYPE HTML>
 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
 <c:url var="base" value="/"></c:url>
 <html>
 <head>
 <title>Login Form</title>
 <link rel="stylesheet" type="text/css" href="\${base}css/login.css" />
 </head>
 <body>
 <h6>\${message}</h6>
 <form method="post" action="\${base}login/login-action1.form">
 <div>
 <h2>登录 login-action1.form</h2>
 <p><label>用户名</label><input type="text" name="username" /></p>
 <p><label>密码</label><input type="password" name="password" /></p>
 </div>
 </form>
 </body>
 </html>

图- 16

2. 再次启动 Tomcat , 测试 login-action5 表单 , 但用户名或者密码错误时候出现结果如下图-17 所示 :

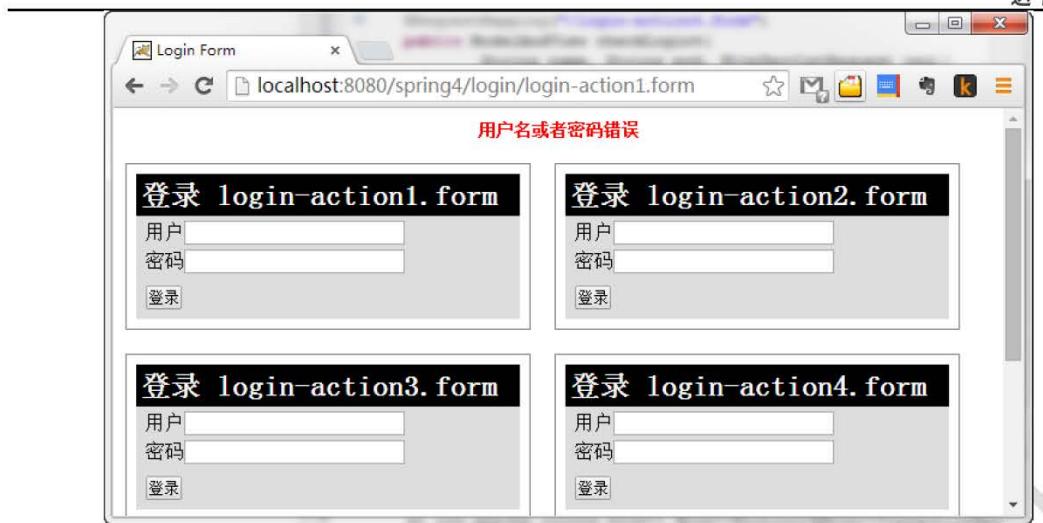


图 - 17

这个结果中红色字体消息就来自 ModelMap 中封装的消息。

#### 步骤七：实现 login-action6.form 登录流程，使用@ModelAttribute 注解向页面传值

1. 修改 LoginController，添加 checkLogin6()方法，如图-18 所示的代码。

```

@RequestMapping("/login-action6.form")
public String checkLogin6(
    @ModelAttribute("name") String name,
    @ModelAttribute("password") String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("----方法六---");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}
private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
@ModelAttribute("next")
public String getNext(){
    Random r = new Random();
    return msg[r.nextInt(msg.length)];
}

```

图 - 18

@ModelAttribute 声明的属性与 login-form.jsp 页面的值关系如图-19 所示：

```

@RequestMapping("/login-action6.form")
public String checkLogin6(
    @ModelAttribute("name") String name,
    @ModelAttribute("password") String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("---方法六---");
    try {
        User user = userService.login(name, pwd);
        if (user != null) {
            model.addAttribute("message", "登录成功");
            return "success";
        } else {
            model.addAttribute("message", "用户名或密码错误");
            return "error";
        }
    }
    private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
    @ModelAttribute("next")
    public String getNext(){
        Random r = new Random();
        return msg[r.nextInt(msg.length)];
    }
}

```

图 - 19

2. 再次启动 Tomcat , 测试 login-action6 表单 , 但用户名或者密码错误时候出现结果如下图-20 所示 :



图 - 20

这个结果中能够记住用户名。

#### 步骤八：Session 存储

在前述案例中 , 用户登录成功以后 , 可以利用 HttpServletRequest 对象的 getSession() 方法访问 Session 对象 , 这样就可以保持用户登录状态了。如图-21 所示的代码。

```

System.out.println(pwd);
try {
    User user = userService.login(name, pwd);
    //登录成功将登录用户信息保存到当前会话中
    req.getSession().setAttribute("user", user);
    return "success";
} catch (NameOrPwdException e) {
    e.printStackTrace();
    req.setAttribute("message", e.getMessage());
    return "login-form";
}

```

图 - 21

### 步骤十一：重构 checkLogin1 方法，测试重定向视图

修改 LoginController，修改 checkLogin1 方法的代码如图-22 所示：

```

@RequestMapping("/login-action1.form")
//Spring MVC 自动参数注入HttpServletRequest
public String checkLogin1(HttpServletRequest req){
    System.out.println("----方法一----");
    //优点直接间接，缺点需要自己处理数据类型转换，不支持文件上传功能
    String name = req.getParameter("name");
    String pwd = req.getParameter("pwd");
    System.out.println(name);
    System.out.println(pwd);
    try {
        User user = userService.login(name, pwd);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "redirect:login.form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

```

图 - 22

上述代码中，使用“redirect:login.form”但提交 login-form1 表单时候，如果没有填写参数，就会发生参数为 null 异常，就会重新定向到 login.form 请求。

- 完整代码

LoginController 类的完整代码如下所示：

```

package com.tarena.controller;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

```

```

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.tarena.entity.User;
import com.tarena.service.NameOrPwdException;
import com.tarena.service.NullParamException;
import com.tarena.service.UserService;

@Controller
//@SessionAttributes("user")
@RequestMapping("/login")
public class LoginController {

    @Resource //请求 Spring 注入资源 userService
    private UserService userService;

    @RequestMapping("/login.form")
    public String loginForm(){
        //可以向表单界面传递一些参数
        return "login-form";
    }

    @RequestMapping("/login-action1.form")
    //Spring MVC 自动参数注入 HttpServletRequest
    public String checkLogin1(HttpServletRequest req){
        System.out.println("----方法一---");
        //优点直接简洁，缺点需要自己处理数据类型转换， 不支持文件上传功能
        String name = req.getParameter("name");
        String pwd = req.getParameter("pwd");
        System.out.println(name);
        System.out.println(pwd);
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "redirect:login.form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }

    @RequestMapping("/login-action2.form")
    public String checkLogin2(
        String name,
        @RequestParam("pwd")String password, //映射表单属性不同的参数
        HttpServletRequest req){
        System.out.println("----方法二---");
        //优点， 自动转换数据类型， 缺点可能出现数据类型转换异常
        System.out.println(name);
    }
}

```

```
System.out.println(password);
try {
    User user = userService.login(name, password);
    //登录成功将登录用户信息保存到当前会话中
    req.getSession().setAttribute("user", user);
    return "success";
} catch (NameOrPwdException e) {
    e.printStackTrace();
    req.setAttribute("message", e.getMessage());
    return "login-form";
} catch (NullParamException e) {
    e.printStackTrace();
    req.setAttribute("message", e.getMessage());
    return "login-form";
} catch (RuntimeException e) {
    e.printStackTrace();
    req.setAttribute("message", e.getMessage());
    return "error";
}
}

@RequestMapping("/login-action3.form")
public String checkLogin3(User user, HttpServletRequest req){
    System.out.println("----方法三----");
    //自动填充到bean 对象
    System.out.println(user);
    try {
        user = userService.login(user.getName(), user.getPwd());
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action4.form")
public ModelAndView checkLogin4(
    String name, String pwd, HttpServletRequest req){
    System.out.println("----方法四----");
    Map<String, Object> data = new HashMap<String, Object>();
    try {
        User user = userService.login(name, pwd);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return new ModelAndView("success", data);
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (NullParamException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (RuntimeException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
    }
}
```

```

        return new ModelAndView("error", data);
    }
}

@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("----方法五----");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action6.form")
public String checkLogin6(
    @ModelAttribute("name") String name,
    @ModelAttribute("password") String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("----方法六----");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}

private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
@ModelAttribute("next")
public String getNext(){
    Random r = new Random();
    return msg[r.nextInt(msg.length)];
}

```

spring-mvc.xml 文件、 web.xml 文件，以及业务层代码略。

## 2. 为登录案例解决中文乱码问题

- 问题

在上次课的登录案例中，如果用户输入中文信息，LoginController 接收到的是乱码。

- 方案

使用 Spring 框架提供的 CharacterEncodingFilter 组件控制字符编码。

- 步骤

**步骤一：**

**修改 web.xml 配置文件**

打开工程 WebRoot/WEB-INF 目录下的 web.xml 文件，添加如图 -23 所示的 CharacterEncodingFilter 配置。

```
<!-- 处理编码问题 -->
<filter>
    <filter-name>encodingFilter</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>encodingFilter</filter-name>
    <url-pattern>*.form</url-pattern>
</filter-mapping>
```

图 - 23

**步骤三：部署并访问工程**

部署工程后启动 Tomcat，在浏览器中输入地址访问登录页面，输入中文信息后点击“登录”按钮，查看服务器端获取参数值的输出结果。

- 完整代码

web.xml 文件的代码如下：

```
<?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>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-mvc.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.form</url-pattern>
</servlet-mapping>

<!-- 处理编码问题 -->
<filter>
    <filter-name>encodingFilter</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>encodingFilter</filter-name>
    <url-pattern>*.form</url-pattern>
</filter-mapping>

<!-- 出错页面定义 -->
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/WEB-INF/jsp/error.jsp</location>
</error-page>
<!--
    <error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/jsp/error404.jsp</location>
    </error-page>
-->

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

### 3. SomeInterceptor 拦截器入门示例

- 问题

利用 Spring 的拦截器可以在处理器 Controller 方法执行前和后增加逻辑代码，了解拦截器中 preHandle、postHandle 和 afterCompletion 方法执行时机。

- 方案

自定义一个拦截器类 SomeInterceptor，实现 HandlerInterceptor 接口及其方法，然后在 spring-mvc.xml 中添加拦截器配置，来指定拦截哪些请求。

- 步骤

### 步骤一：创建 SomeInterceptor 拦截器组件

新建一个 com.tarena.interceptor 包，在该包中新建一个 SomeInterceptor 类。

SomeInterceptor 类要实现 HandlerInterceptor 接口及其约定方法，代码如下：

```
public class SomeInterceptor implements HandlerInterceptor{

    public void afterCompletion(HttpServletRequest req,
                                HttpServletResponse res, Object handler, Exception e)
        throws Exception {
        System.out.println("请求处理完成后调用");
    }

    public void postHandle(HttpServletRequest req,
                           HttpServletResponse res, Object handler, ModelAndView mv)
        throws Exception {
        System.out.println("处理器执行后调用");
    }

    public boolean preHandle(HttpServletRequest req, HttpServletResponse res,
                             Object handler) throws Exception {
        System.out.println("处理器执行前调用");
        return true;
    }
}
```

### 步骤二：修改 spring-mvc.xml 配置

打开工程 src 下的 spring-mvc.xml 文件，追加 SomeInterceptor 配置。

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/spring/*"/>
        <mvc:exclude-mapping path="/login/*"/>
        <bean class="com.tarena.interceptor.SomeInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

### 步骤三：创建控制器 HelloController 中添加控制台打印信息

创建 HelloController，在 execute 方法增加控制台打印语句。

```
package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/spring")
public class HelloController{

    @RequestMapping("/hello.form")
    public String execute() throws Exception {
        System.out.println("执行 HelloController");
        return "hello";
    }
}
```

#### 步骤四：部署并访问工程

部署工程后启动Tomcat，在浏览器中输入地址 <http://localhost:8080/spring4/spring/hello.form>，观察控制台输出结果。输出结果参考如图-24所示：

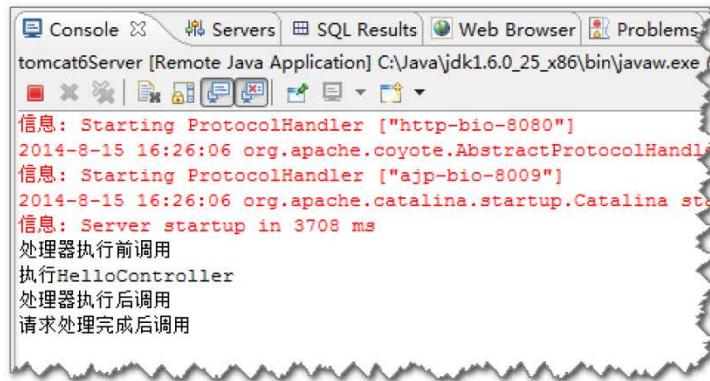


图 - 24

#### • 完整代码

SomeInterceptor.java 完整代码如下：

```
package com.tarena.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class SomeInterceptor implements HandlerInterceptor{

    public void afterCompletion(HttpServletRequest req,
                               HttpServletResponse res, Object handller, Exception e)
        throws Exception {
        System.out.println("请求处理完成后调用");
    }

    public void postHandle(HttpServletRequest req,
                           HttpServletResponse res, Object handller,
                           ModelAndView mv) throws Exception {
        System.out.println("处理器执行后调用");
    }

    public boolean preHandle(HttpServletRequest req,
                             HttpServletResponse res, Object handller) throws Exception {
        System.out.println("处理器执行前调用");
        return true;
    }
}
```

### spring-mvc.xml 完整代码如下：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/day01/*"/>
            <mvc:exclude-mapping path="/login/*"/>
            <bean class="com.tarena.interceptor.SomeInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/">
        </property>
        <property name="suffix" value=".jsp">
        </property>
    </bean>
</beans>
```

### HelloController.java 完整代码如下：

```

package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/spring")
public class HelloController{

    @RequestMapping("/hello.from")
    public String execute() throws Exception {
```

```
        System.out.println("执行HelloController");
        return "hello";
    }
}
```

## 4. 为 Hello 示例添加登录检查拦截器

- **问题**

为了提高系统安全性，有些功能要求用户必须登录才能访问，以 hello.jsp 为例，追加登录检查功能。

- **方案**

编写一个登录检查拦截器，在 preHandle 方法进行登录检测。如果未登录，中断请求处理，定位到登录页面。

- **步骤**

### 步骤一：编写 LoginInterceptor 拦截器实现类

新建一个名为 LoginInterceptor 类，注意要实现 HandlerInterceptor 接口，在 preHandle 方法添加 session 登录信息的检查（一般登录成功后都会将用户信息写入 session），文件源代码如下：

```
package com.tarena.interceptor;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.ModelAndView;

public class LoginInterceptor implements HandlerInterceptor{

    public void afterCompletion(HttpServletRequest req,
                               HttpServletResponse res, Object handler, Exception e)
        throws Exception {
    }

    public void postHandle(HttpServletRequest req,
                           HttpServletResponse res, Object handler,
                           ModelAndView mv) throws Exception {
    }

    public boolean preHandle(HttpServletRequest req,
                             HttpServletResponse res, Object handler) throws Exception {
        Object user = req.getSession().getAttribute("user");
        if(user != null){
            return true;
        }
        ServletContext ctx = req.getSession().getServletContext();
        res.sendRedirect(ctx.getContextPath() + "/login/login.form");
    }
}
```

```
        return false;
    }
}
```

## 步骤二：修改 spring-mvc.xml 配置

打开 spring-mvc.xml，添加 LoginInterceptor 配置。

```
<mvc:interceptors>
    <!--省略其他拦截器配置-->
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/login/**"/>
        <bean class="com.tarena.interceptor.LoginInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

## 步骤三：部署并访问工程

部署工程，启动服务器，访问 <http://localhost:8080/spring4/spring/hello.form>，会发现定位到 <http://localhost:8080/spring4/login/login.form>。

### • 完整代码

LoginInterceptor.java 完整代码如下：

```
package com.tarena.interceptor;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class LoginInterceptor implements HandlerInterceptor{

    public void afterCompletion(HttpServletRequest req,
                               HttpServletResponse res, Object handler, Exception e)
        throws Exception {
    }

    public void postHandle(HttpServletRequest req,
                           HttpServletResponse res, Object handler,
                           ModelAndView mv) throws Exception {
    }

    public boolean preHandle(HttpServletRequest req,
                           HttpServletResponse res, Object handler) throws Exception {
        Object user = req.getSession().getAttribute("user");
        if(user != null){
            return true;
        }
        ServletContext ctx = req.getSession().getServletContext();
        res.sendRedirect(ctx.getContextPath() + "/login/login.form");
        return false;
    }
}
```

LoginController.java 完整代码如下：

```
package com.tarena.controller;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.tarena.entity.User;
import com.tarena.service.NameOrPwdException;
import com.tarena.service.NullParamException;
import com.tarena.service.UserService;

@Controller
//@SessionAttributes("user")
@RequestMapping("/login")
public class LoginController {

    @Resource //请求 Spring 注入资源 userService
    private UserService userService;

    @RequestMapping("/login.form")
    public String loginForm(){
        //可以向表单界面传递一些参数
        return "login-form";
    }

    @RequestMapping("/login-action1.form")
    //Spring MVC 自动参数注入 HttpServletRequest
    public String checkLogin1(HttpServletRequest req){
        System.out.println("----方法----");
        //优点直接简洁，缺点需要自己处理数据类型转换，不支持文件上传功能
        String name = req.getParameter("name");
        String pwd = req.getParameter("pwd");
        System.out.println(name);
        System.out.println(pwd);
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "redirect:login.form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }
}
```

```

        }

    }

    @RequestMapping("/login-action2.form")
    public String checkLogin2(
        String name,
        @RequestParam("pwd")String password, //映射表单属性不同的参数
        HttpServletRequest req){
        System.out.println("----方法二----");
        //优点，自动转换数据类型，缺点可能出现数据类型转换异常
        System.out.println(name);
        System.out.println(password);
        try {
            User user = userService.login(name, password);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }

    @RequestMapping("/login-action3.form")
    public String checkLogin3(User user, HttpServletRequest req){
        System.out.println("----方法三----");
        //自动填充到bean对象
        System.out.println(user);
        try {
            user = userService.login(user.getName(), user.getPwd());
            // 登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }

    @RequestMapping("/login-action4.form")
    public ModelAndView checkLogin4(
        String name, String pwd, HttpServletRequest req){
        System.out.println("----方法四----");
        Map<String, Object> data = new HashMap<String, Object>();
        try {
            User user = userService.login(name, pwd);

```

```

    //登录成功将登录用户信息保存到当前会话中
    req.getSession().setAttribute("user", user);
    return new ModelAndView("success", data);
} catch (NameOrPwdException e) {
    e.printStackTrace();
    data.put("message", e.getMessage());
    return new ModelAndView("login-form", data);
} catch (NullParamException e) {
    e.printStackTrace();
    data.put("message", e.getMessage());
    return new ModelAndView("login-form", data);
} catch (RuntimeException e) {
    e.printStackTrace();
    data.put("message", e.getMessage());
    return new ModelAndView("error", data);
}
}

@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model, HttpServletRequest req){
System.out.println("----方法五---");
try {
    User user = userService.login(name, pwd);
    // 登录成功将登录用户信息保存到当前会话中
    req.getSession().setAttribute("user", user);
    return "success";
} catch (NameOrPwdException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "login-form";
} catch (NullParamException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "login-form";
} catch (RuntimeException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "error";
}
}

@RequestMapping("/login-action6.form")
public String checkLogin6(
    @ModelAttribute("name") String name,
    @ModelAttribute("password") String pwd,
    ModelMap model, HttpServletRequest req){
System.out.println("----方法六---");
try {
    User user = userService.login(name, pwd);
    // 登录成功将登录用户信息保存到当前会话中
    req.getSession().setAttribute("user", user);
    return "success";
} catch (NameOrPwdException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "login-form";
} catch (NullParamException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "login-form";
} catch (RuntimeException e) {
    e.printStackTrace();
    model.addAttribute("message", e.getMessage());
    return "error";
}
}

```

```

    }
}

private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
@ModelAttribute("next")
public String getNext(){
    Random r = new Random();
    return msg[r.nextInt(msg.length)];
}
}

```

### spring-mvc.xml 文件完整代码如下：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/spring/*"/>
            <mvc:exclude-mapping path="/login/*"/>
            <bean class="com.tarena.interceptor.SomeInterceptor"/>
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <mvc:exclude-mapping path="/login/*"/>
            <bean class="com.tarena.interceptor.LoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/">
        </property>
        <property name="suffix" value=".jsp">
        </property>
    </bean>
</beans>

```

## 5. 为案例添加异常处理

- 问题

Spring MVC 框架提供了异常处理机制，当 Controller 组件抛出异常后会触发异常处理。

- 方案

Spring 提供了三种异常处理方法，具体如下：

- 1 ) 使用 SimpleMappingExceptionResolver
- 2 ) 使用自定义 ExceptionResolver 组件
- 3 ) 使用@ExceptionHandler 注解

- 步骤

### 步骤一：实现第一种方法，修改 spring-mvc.xml 配置

打开工程的 spring-mvc.xml 文件，添加 SimpleMappingExceptionResolver 定义，如图-25 所示：



图 - 25

### 步骤二：在 LoginController 方法中制造异常

打开 LoginController.java，在 checkLogin1 处理方法中添加下面代码。

```
//制造空指针异常
String s = null;
s.length();
```

### 步骤三：添加 error.jsp 异常处理页面

在工程中新建一个 error.jsp 页面，error.jsp 位置如图-26 所示：

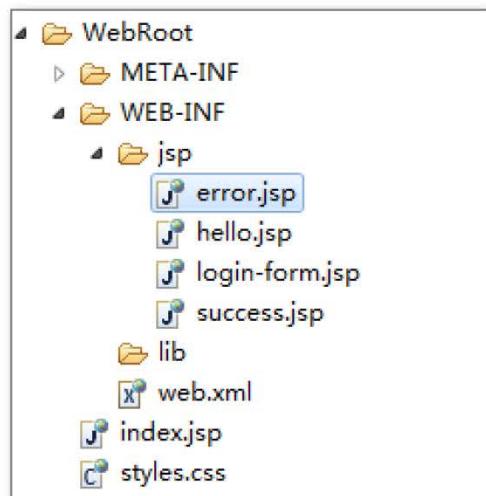


图 - 26

`error.jsp` 代码如下：

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE HTML>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="base" value="/"></c:url>
<html>
  <head>
    <title>Login Form</title>
    <link rel="stylesheet" type="text/css" href="${base}styles.css">
  </head>
  <body>
    <h6>发生了意外${message}${exception}</h6>
  </body>
</html>
    
```

#### 步骤四：部署并访问工程，测试第一种方法

部署并访问工程，进入登录页面，输入登录信息后，点击“登录”按钮，会显示 `error.jsp` 页面，如图-27 所示：

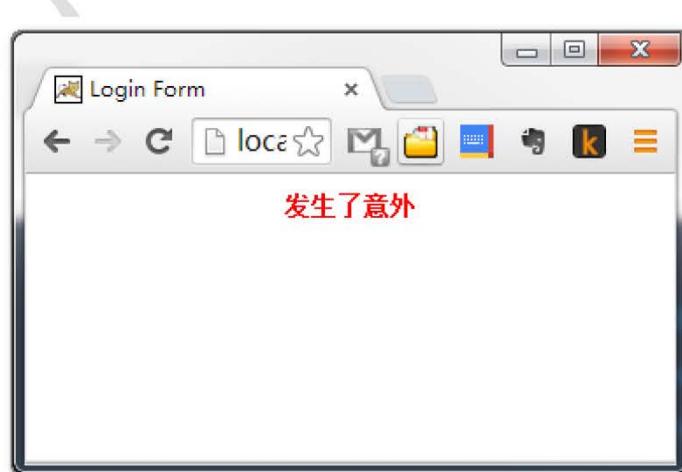


图 - 27

### 步骤五：实现第二种方法，自定义 MyMappingExceptionResolver 异常处理器

在 com.tarena.interceptor 下新建一个 MyMappingExceptionResolver 类，实现 HandlerExceptionResolver 接口，如图-28 所示：

```
public class MyMappingExceptionResolver implements HandlerExceptionResolver {  
    public ModelAndView resolveException(HttpServletRequest req,  
                                         HttpServletResponse res, Object handler, Exception ex) {  
        Map<String, Object> model = new HashMap<String, Object>();  
        model.put("ex", ex);  
        //TODO 根据不同错误转向不同页面  
        return new ModelAndView("error", model);  
    }  
}
```

图 - 28

### 步骤六：修改 spring-mvc.xml 文件

打开 spring-mvc.xml 文件，注释掉 SimpleMappingExceptionResolver 配置，追加 MyMappingExceptionResolver 配置，如图-29 所示：

```
<!--  
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">  
    <property name="exceptionAttribute" value="ex"/>  
    <property name="exceptionMappings">  
        <props>  
            <prop key="java.lang.Exception">error</prop>  
        </props>  
    </property>  
</bean>  
-->  
  
<bean id="exceptionHandler" class="com.tarena.interceptor.MyMappingExceptionResolver"/>
```

图 - 29

### 步骤七：部署并访问工程，测试第二种方法

部署并访问工程，进入登录页面，输入登录信息后，点击“登录”按钮，会显示 error.jsp 页面，效果与第一种相同。

### 步骤八：实现第三种方法，定义一个 BaseController 类

在 com.tarena.controller 下新建一个 BaseController 类，定义一个异常处理方法，然后使用 @ExceptionHandler 标注该方法，代码如下所示：

```
package com.tarena.controller;  
  
import java.io.Serializable;  
  
import javax.servlet.http.HttpServletRequest;  
  
import org.springframework.web.bind.annotation.ExceptionHandler;  
  
public class BaseController implements Serializable{  
    @ExceptionHandler
```

```
//@ResponseBody
public String execute(HttpServletRequest request, Exception ex){
    request.setAttribute("ex", ex);
    request.setAttribute("message", ex.getMessage());
    // 可根据异常类型不同返回不同视图名
    return "error";
}
}
```

### 步骤九：将 LoginController 继承 BaseController

当 LoginController 需要异常处理时，可以继承 BaseController，这样可以实现异常处理功能的重复利用，如图-30 所示：

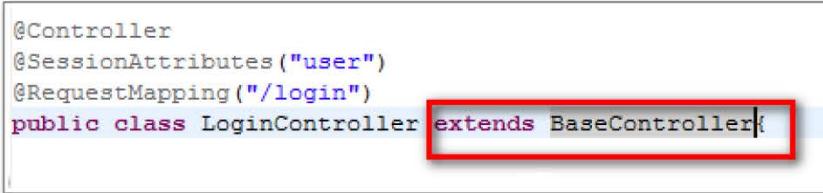


图 - 30

### 步骤十：修改 spring-mvc.xml 文件

打开 spring-mvc.xml 文件，注释掉第二种的 MyMappingExceptionResolver 配置

### 步骤十一：部署并访问工程，测试第三种方法

部署并访问工程，进入登录页面，输入登录信息后，点击“登录”按钮，会显示 error.jsp 页面，效果与第一种相同。

- 第一种方法完整代码

spring-mvc.xml 文件的完整实现代码：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/mvc
    "
```

[http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">](http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd)

```

<context:component-scan base-package="com.tarena.controller"/>

<mvc:annotation-driven/>

<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/spring/*"/>
        <mvc:exclude-mapping path="/login/*"/>
        <bean class="com.tarena.interceptor.SomeInterceptor"/>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/login/*"/>
        <bean class="com.tarena.interceptor.LoginInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>

<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"/>
    </property>
    <property name="suffix" value=".jsp">
    </property>
</bean>

<bean
      class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionAttribute" value="ex"/>
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.Exception">error</prop>
        </props>
    </property>
</bean>
</beans>

```

**error.jsp 文件的完整代码如下：**

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<!DOCTYPE HTML>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="base" value="/"></c:url>
<html>
    <head>
        <title>Login Form</title>
        <link rel="stylesheet" type="text/css" href="${base}styles.css">
    </head>
    <body>
        <h6>发生了意外${message}${ex}</h6>
    </body>
</html>

```

**LoginController.java 文件的完整代码如下：**

```

package com.tarena.controller;

import java.util.HashMap;

```

```
import java.util.Map;
import java.util.Random;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.tarena.entity.User;
import com.tarena.service.NameOrPwdException;
import com.tarena.service.NullParamException;
import com.tarena.service.UserService;

@Controller
//@SessionAttributes("user")
@RequestMapping("/login")
public class LoginController {

    @Resource //请求 Spring 注入资源 userService
    private UserService userService;

    @RequestMapping("/login.form")
    public String loginForm(){
        //可以向表单界面传递一些参数
        return "login-form";
    }

    @RequestMapping("/login-action1.form")
    //Spring MVC 自动参数注入 HttpServletRequest
    public String checkLogin1(HttpServletRequest req){

        String s = null;
        s.length();

        System.out.println("----方法----");
        //优点直接简洁，缺点需要自己处理数据类型转换， 不支持文件上传功能
        String name = req.getParameter("name");
        String pwd = req.getParameter("pwd");
        System.out.println(name);
        System.out.println(pwd);
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "redirect:login.form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "error";
        }
    }
}
```

```

@RequestMapping("/login-action2.form")
public String checkLogin2(
    String name,
    @RequestParam("pwd") String password, //映射表单属性不同的参数
    HttpServletRequest req){
    System.out.println("----方法二---");
    //优点，自动转换数据类型，缺点可能出现数据类型转换异常
    System.out.println(name);
    System.out.println(password);
    try {
        User user = userService.login(name, password);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action3.form")
public String checkLogin3(User user, HttpServletRequest req){
    System.out.println("----方法三---");
    //自动填充到bean对象
    System.out.println(user);
    try {
        user = userService.login(user.getName(), user.getPwd());
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action4.form")
public ModelAndView checkLogin4(
    String name, String pwd, HttpServletRequest req){
    System.out.println("----方法四---");
    Map<String, Object> data = new HashMap<String, Object>();
    try {
        User user = userService.login(name, pwd);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
    }
}

```

```
        return new ModelAndView("success", data);
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (NullParamException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("login-form", data);
    } catch (RuntimeException e) {
        e.printStackTrace();
        data.put("message", e.getMessage());
        return new ModelAndView("error", data);
    }
}

@RequestMapping("/login-action5.form")
public String checkLogin5(String name, String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("---方法五---");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action6.form")
public String checkLogin6(
    @ModelAttribute("name") String name,
    @ModelAttribute("password") String pwd,
    ModelMap model, HttpServletRequest req){
    System.out.println("---方法六---");
    try {
        User user = userService.login(name, pwd);
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        model.addAttribute("message", e.getMessage());
        return "error";
    }
}
```

```

private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
@ModelAttribute("next")
public String getNext(){
    Random r = new Random();
    return msg[r.nextInt(msg.length)];
}
}

```

### • 第二种方法完整代码

MyMappingExceptionResolver.java 文件的完整实现代码：

```

package com.tarena.interceptor;

import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;

public class MyMappingExceptionResolver implements HandlerExceptionResolver{

    public ModelAndView resolveException(HttpServletRequest req,
                                         HttpServletResponse res, Object handler, Exception ex) {
        Map<String, Object> model = new HashMap<String, Object>();
        model.put("ex", ex);
        //TODO 根据不同错误转向不同页面
        return new ModelAndView("error", model);
    }
}

```

spring-mvc.xml 文件的完整实现代码：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/mvc
       "
/>

```

```

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

<context:component-scan base-package="com.tarena.controller"/>

<mvc:annotation-driven/>

<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/day01/**"/>
        <mvc:exclude-mapping path="/login/**"/>
        <bean class="com.tarena.interceptor.SomeInterceptor"/>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/login/**"/>
        <bean class="com.tarena.interceptor.LoginInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>

<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"/>
    </property>
    <property name="suffix" value=".jsp">
    </property>
</bean>

<!--
<bean
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolve
r">
    <property name="exceptionAttribute" value="ex"/></property>
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.Exception">error</prop>
        </props>
    </property>
</bean>
-->

<bean id="exceptionHandler"
      class="com.tarena.interceptor.MyMappingExceptionResolver"/>

</beans>

```

### • 第三种方法完整代码

BaseController.java 文件的完整实现代码：

```

package com.tarena.controller;

import javax.servlet.http.HttpServletRequest;
import org.springframework.web.bind.annotation.ExceptionHandler;

public class BaseController {
    @ExceptionHandler
    public String execute(HttpServletRequest request, Exception ex){
        request.setAttribute("ex", ex);
        //TODO 可根据异常类型不同返回不同视图名
        return "error";
    }
}

```

## LoginController.java 文件的完整实现代码：

```
package com.tarena.controller;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.tarena.entity.User;
import com.tarena.service.NameOrPwdException;
import com.tarena.service.NullParamException;
import com.tarena.service.UserService;

@Controller
//@SessionAttributes("user")
@RequestMapping("/login")
public class LoginController extends BaseController{

    @Resource //请求 Spring 注入资源 userService
    private UserService userService;

    @RequestMapping("/login.form")
    public String loginForm(){
        //可以向表单界面传递一些参数
        return "login-form";
    }

    @RequestMapping("/login-action1.form")
    //Spring MVC 自动参数注入 HttpServletRequest
    public String checkLogin1(HttpServletRequest req){

        String s = null;
        s.length();

        System.out.println("----方法----");
        //优点直接简洁，缺点需要自己处理数据类型转换， 不支持文件上传功能
        String name = req.getParameter("name");
        String pwd = req.getParameter("pwd");
        System.out.println(name);
        System.out.println(pwd);
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            req.setAttribute("message", e.getMessage());
            return "redirect:login.form";
        }
    }
}
```

```

    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action2.form")
public String checkLogin2(
    String name,
    @RequestParam("pwd")String password, //映射表单属性不同的参数
    HttpServletRequest req){
    System.out.println("----方法二----");
    //优点， 自动转换数据类型， 缺点可能出现数据类型转换异常
    System.out.println(name);
    System.out.println(password);
    try {
        User user = userService.login(name, password);
        //登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action3.form")
public String checkLogin3(User user, HttpServletRequest req){
    System.out.println("----方法三----");
    //自动填充到bean 对象
    System.out.println(user);
    try {
        user = userService.login(user.getName(), user.getPwd());
        // 登录成功将登录用户信息保存到当前会话中
        req.getSession().setAttribute("user", user);
        return "success";
    } catch (NameOrPwdException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (NullParamException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "login-form";
    } catch (RuntimeException e) {
        e.printStackTrace();
        req.setAttribute("message", e.getMessage());
        return "error";
    }
}

@RequestMapping("/login-action4.form")
public ModelAndView checkLogin4(
    String name, String pwd, HttpServletRequest req){

```

```

        System.out.println("----方法四---");
        Map<String, Object> data = new HashMap<String, Object>();
        try {
            User user = userService.login(name, pwd);
            //登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return new ModelAndView("success", data);
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            data.put("message", e.getMessage());
            return new ModelAndView("login-form", data);
        } catch (NullParamException e) {
            e.printStackTrace();
            data.put("message", e.getMessage());
            return new ModelAndView("login-form", data);
        } catch (RuntimeException e) {
            e.printStackTrace();
            data.put("message", e.getMessage());
            return new ModelAndView("error", data);
        }
    }

    @RequestMapping("/login-action5.form")
    public String checkLogin5(String name, String pwd,
        ModelMap model, HttpServletRequest req){
        System.out.println("----方法五---");
        try {
            User user = userService.login(name, pwd);
            // 登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "login-form";
        } catch (RuntimeException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "error";
        }
    }

    @RequestMapping("/login-action6.form")
    public String checkLogin6(
        @ModelAttribute("name") String name,
        @ModelAttribute("password") String pwd,
        ModelMap model, HttpServletRequest req){
        System.out.println("----方法六---");
        try {
            User user = userService.login(name, pwd);
            // 登录成功将登录用户信息保存到当前会话中
            req.getSession().setAttribute("user", user);
            return "success";
        } catch (NameOrPwdException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "login-form";
        } catch (NullParamException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "login-form";
        }
    }
}

```

```

        } catch (RuntimeException e) {
            e.printStackTrace();
            model.addAttribute("message", e.getMessage());
            return "error";
        }
    }

    private String[] msg = {"再来一次", "下次就对了", "没关系还有机会"};
    @ModelAttribute("next")
    public String getNext(){
        Random r = new Random();
        return msg[r.nextInt(msg.length)];
    }
}

```

### spring-mvc.xml 文件的完整实现代码：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc
           http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/data
           http://www.springframework.org/schema/data/jpa
           http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/day01/**"/>
            <mvc:exclude-mapping path="/login/**"/>
            <bean class="com.tarena.interceptor.SomeInterceptor"/>
        </mvc:interceptor>
        <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <mvc:exclude-mapping path="/login/**"/>
            <bean class="com.tarena.interceptor.LoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        </property>
    
```

```
<property name="suffix" value=".jsp">
</property>
</bean>

<!--
<bean
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolve
r">
    <property name="exceptionAttribute" value="ex"></property>
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.Exception">error</prop>
        </props>
    </property>
</bean>

<bean id="exceptionHandler"
      class="com.tarena.interceptor.MyMappingExceptionResolver"/>

</beans>
-->
```

## 6. 文件上传案例

- **问题**

如何在 Spring Web MVC 中实现文件上传操作。

- **方案**

利用 Spring Web MVC 提供的 CommonsMultipartResolver 组件实现。

- **步骤**

实现此案例需要按照如下步骤进行。

**步骤一：新建工程，导入 jar 包**

新建名为 SpringMVC\_Day05\_Part1 的 Web 工程，在该工程导入如图-31 所示的 jar 包：

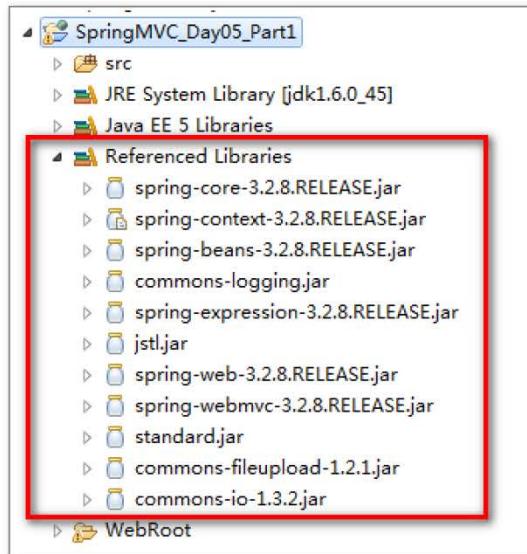


图-31

## 步骤二：新建 SpringMVC 控制类 UploadController

新建 SpringMVC 控制类 UploadController，如图-2 所示的 jar 包：

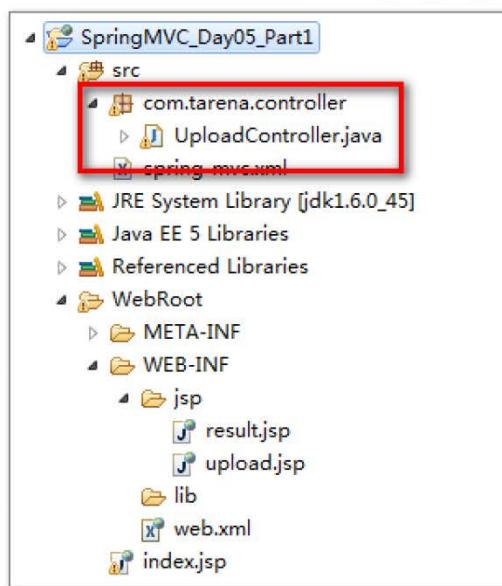


图-32

SpringMVC 控制类 UploadController 的代码如下所示：

```
package com.tarena.controller;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
```

```

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MaxUploadSizeExceedededException;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class UploadController {

    @RequestMapping("/toUpload.from")
    public String toUpload() {
        return "upload";
    }

    @RequestMapping(value = "/upload.from")
    public String upload(
        @RequestParam(value = "file", required = false)
        MultipartFile file,
        HttpServletRequest request,
        ModelMap model) {

        String path = request.getSession().getServletContext().getRealPath(
            "upload");
        String fileName = file.getOriginalFilename();
        System.out.println(path);
        File targetFile = new File(path, fileName);
        if (!targetFile.exists()) {
            targetFile.mkdirs();
        }
        // 保存
        try {
            file.transferTo(targetFile);
            model.addAttribute("fileUrl", request.getContextPath() + "/upload/" +
                fileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "result";
    }

    @RequestMapping(value = "/uploads.from")
    public String uploads(
        @RequestParam(value = "file", required = false)
        MultipartFile[] files,
        HttpServletRequest request,
        ModelMap model) {
        List<String> urls = new ArrayList<String>();
        for (MultipartFile file : files) {
            String path = request.getSession().getServletContext().getRealPath(
                "upload");
            String fileName = file.getOriginalFilename();
            System.out.println(path);
            File targetFile = new File(path, fileName);
            if (!targetFile.exists()) {
                targetFile.mkdirs();
            }
            // 保存
            try {
                file.transferTo(targetFile);
                urls.add(request.getContextPath() + "/upload/" + fileName);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

        }
        model.addAttribute("fileUrls", urls);
        return "result";
    }

}

```

### 步骤三：新建 Spring MVC 配置文件 spring-mvc.xml

新建 SpringMVC 配置文件 spring-mvc.xml，如图-33 所示：

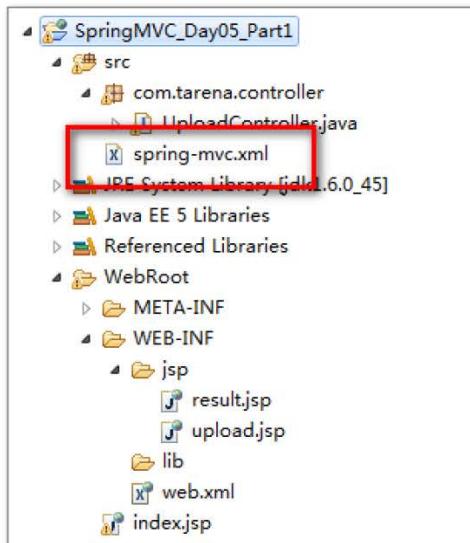


图-33

SpringMVC 配置文件 spring-mvc.xml 的代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>
    <bean id="viewResolver" />

```

```
<class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/">
    </property>
    <property name="suffix" value=".jsp">
    </property>
</bean>

<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>

</beans>
```

#### 步骤四：修改 web.xml 配置文件

修改 web.xml 配置文件为如下代码：

```
<?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>
        <!-- 指定 Spring 的配置文件 -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-mvc.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.from</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

#### 步骤五：修改 index.jsp 文件

修改 index.jsp 配置文件为如下代码：

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

    <body>
        <a href="toUpload.from">文件上传示例</a>
    </body>
</html>
```

## 步骤六：新建 jsp 文件 upload.jsp

新建 jsp 文件 upload.jsp , 如图-34 所示：

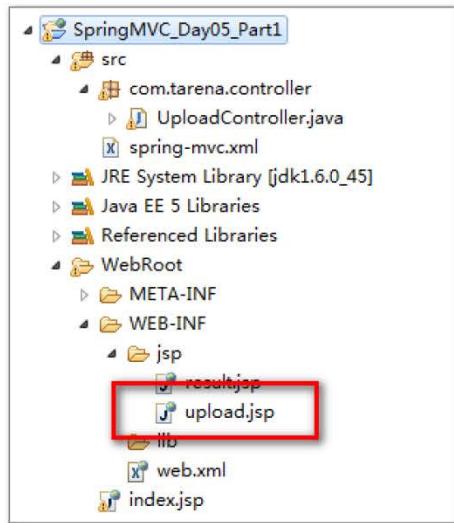


图-34

jsp 文件 upload.jsp 的代码如下所示：

```
<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>
<html>
    <title>上传图片</title>
    <body>
        ${errors }
        <form action="upload.from" method="post"
            enctype="multipart/form-data">
            <input type="file" name="file" /> <input type="submit"
            value="Submit" />
        </form>
        <hr/>
        <form action="uploads.from" method="post"
            enctype="multipart/form-data">
            <input type="file" name="file" /><br/>
            <input type="file" name="file" />
            <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

## 步骤七：新建 jsp 文件 result.jsp

新建 jsp 文件 result.jsp , 如图-35 所示：

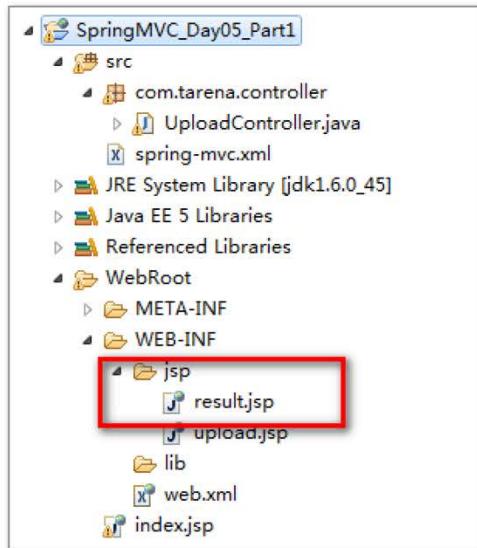


图-35

jsp 文件 result.jsp 的代码如下所示：

```
<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head></head>
<body style="font-size: 30px; font-style: italic;">
    <c:if test="${fileUrl != null}">
        <a href="${fileUrl }">查看</a>
    </c:if>
    <hr />
    <ul>
        <c:forEach items="${fileUrls}" var="url">
            <li><a href="${url }">查看</a></li>
        </c:forEach>
    </ul>
</body>
</html>
```

#### 步骤八：Web 方式运行 SpringMVC\_Day05\_Part1 项目

将 SpringMVC\_Day05\_Part1 项目以 Web 项目方式运行，如图-36 所示：

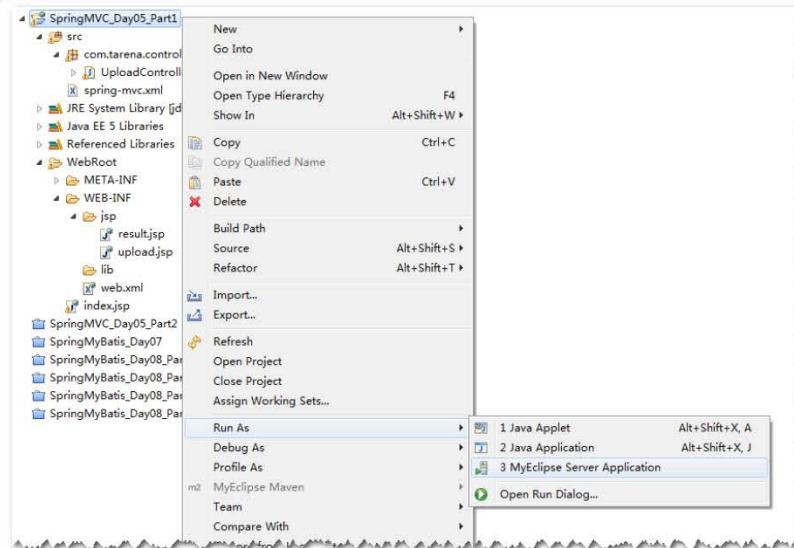


图-36

打开浏览器输入 URL :

[http://localhost:8080/SpringMVC\\_Day05\\_Part1/index.jsp](http://localhost:8080/SpringMVC_Day05_Part1/index.jsp), 如图-37 所示 :

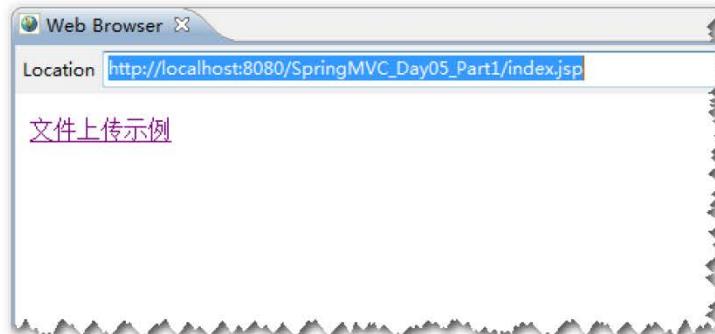


图-37

点击 “文件上传示例” 链接，显示图-38 界面：

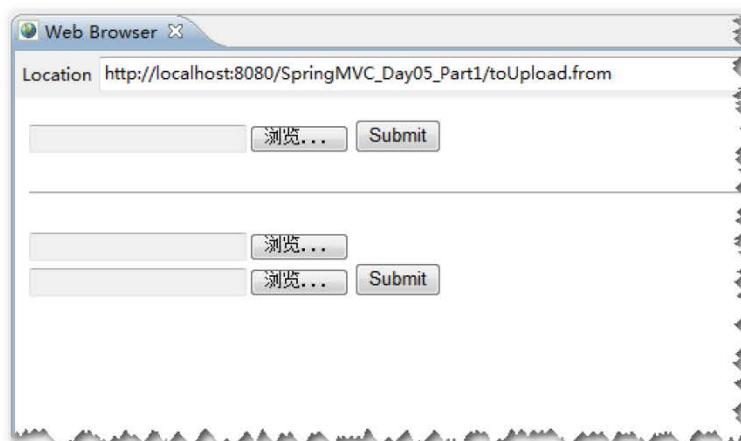


图-38

选择一个文本文件并点击“Submit”，如图-39所示：

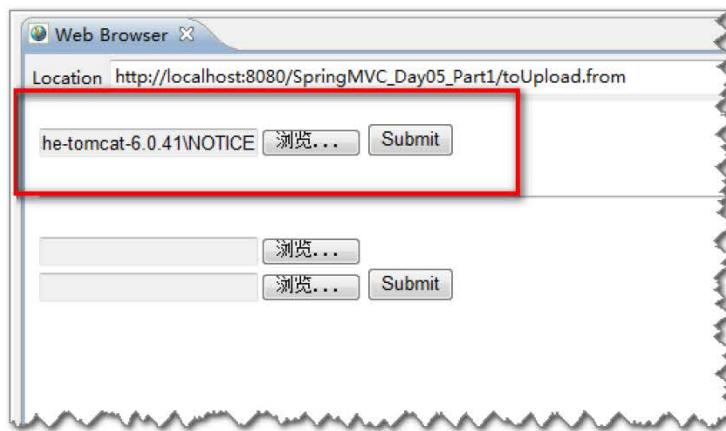


图-39

提交后显示如图-40的页面，点击“查看”链接，可以显示刚才上传的文本文件内容：

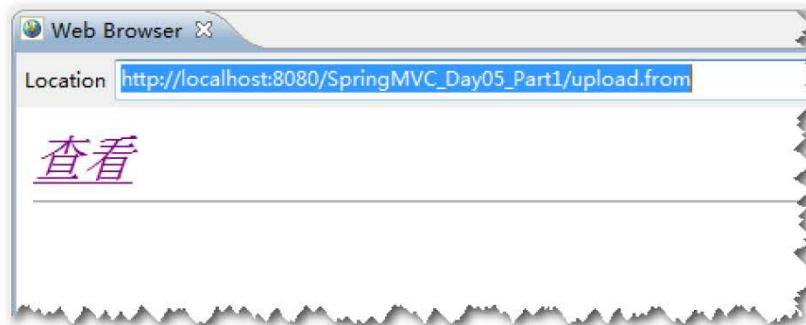


图-40

正确显示上传的文本文件内容则测试成功。

对图-41方框内的多文件提交，同样采取上述方法测试：

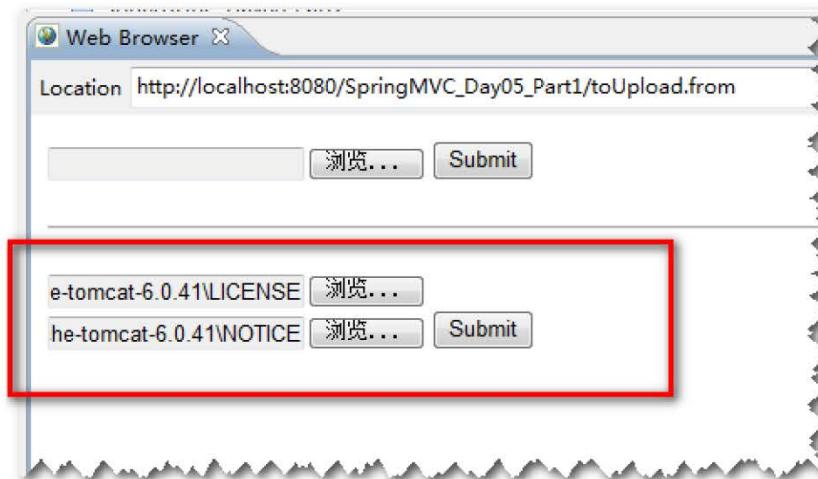


图-41

- 完整代码

SpringMVC 控制类 UploadController 代码如下所示：

```
package com.tarena.controller;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MaxUploadSizeExceedededException;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class UploadController {

    @RequestMapping("/toUpload.from")
    public String toUpload() {
        return "upload";
    }

    @RequestMapping(value = "/upload.from")
    public String upload(
        @RequestParam(value = "file", required = false)
        MultipartFile file,
        HttpServletRequest request,
        ModelMap model) {

        String path = request.getSession().getServletContext().getRealPath(
            "upload");
        String fileName = file.getOriginalFilename();
        System.out.println(path);
        File targetFile = new File(path, fileName);
        if (!targetFile.exists()) {
            targetFile.mkdirs();
        }
        // 保存
        try {
            file.transferTo(targetFile);
            model.addAttribute("fileUrl", request.getContextPath() + "/upload/" +
                + fileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    return "result";
}

@RequestMapping(value = "/uploads.from")
public String uploads(
    @RequestParam(value = "file", required = false)
    MultipartFile[] files,
    HttpServletRequest request,
    ModelMap model) {
    List<String> urls = new ArrayList<String>();
}
```

```

        for (MultipartFile file : files) {
            String path = request.getSession().getServletContext().getRealPath(
                "upload");
            String fileName = file.getOriginalFilename();
            System.out.println(path);
            File targetFile = new File(path, fileName);
            if (!targetFile.exists()) {
                targetFile.mkdirs();
            }
            // 保存
            try {
                file.transferTo(targetFile);
                urls.add(request.getContextPath() + "/upload/" + fileName);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        model.addAttribute("fileUrls", urls);
        return "result";
    }
}

```

SpringMVC 配置文件 spring-mvc.xml 代码如下所示：

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller"/>

    <mvc:annotation-driven/>

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        </property>
        <property name="suffix" value=".jsp">
        </property>
    </bean>

```

```

<bean
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
  />
</beans>

```

配置文件 web.xml 代码如下所示：

```

<?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>
    <!-- 指定 Spring 的配置文件 -->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.from</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>

```

index.jsp 代码如下所示：

```

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

  <body>
    <a href="toUpload.from">文件上传示例</a>
  </body>
</html>

```

upload.jsp 代码如下所示：

```

<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>
<html>
  <title>上传图片</title>
  <body>
    ${errors }
    <form           action="upload.from"           method="post"
      enctype="multipart/form-data">
      <input type="file" name="file" />    <input type="submit"
      value="Submit" />
    </form>

```

```

<hr/>
<form action="uploads.from" method="post"
enctype="multipart/form-data">
  <input type="file" name="file" /><br/>
  <input type="file" name="file" />
  <input type="submit" value="Submit" />
</form>
</body>
</html>
  
```

**result.jsp 代码如下所示：**

```

<%@page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <head></head>
  <body style="font-size: 30px; font-style: italic;">
    <c:if test="${fileUrl != null}">
      <a href="#">查看</a>
    </c:if>
    <hr />
    <ul>
      <c:forEach items="${fileUrls}" var="url">
        <li><a href="#">查看</a></li>
      </c:forEach>
    </ul>
  </body>
</html>
  
```

## 7. 限制上传文件大小案例

- **问题**

在上传功能中如何对上传文件大小进行限制？

- **方案**

在 CommonsMultipartResolver 组件中有一个 maxUploadSize 属性，用于限制上传文件大小。在配置 CommonsMultipartResolver 时指定 maxUploadSize 属性值，然后对在 Controller 组件中处理文件超限异常即可。

- **步骤**

实现此案例需要按照如下步骤进行。

**步骤一：修改 Spring 配置文件 spring-mvc.xml**

修改 Spring 配置文件 spring-mvc.xml，在 multipartResolver 标签内加入图-42 方框所示代码。

```

<?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:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="

        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller" />

    <mvc:annotation-driven />

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <bean id="multipartResolver"
        class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="maxUploadSize" value="51200" />
        <property name="resolveLazily" value="true" />
    </bean>
</beans>

```

图-42

修改 Spring 配置文件 spring-mvc.xml，在 multipartResolver 标签内加入代码如下所示：

```

<property name="maxUploadSize" value="51200" />
<property name="resolveLazily" value="true" />

```

## 步骤二：Web 方式运行 SpringMVC\_Day05\_Part1 项目

将 SpringMVC\_Day05\_Part1 项目以 Web 项目方式运行，如图-43 所示：

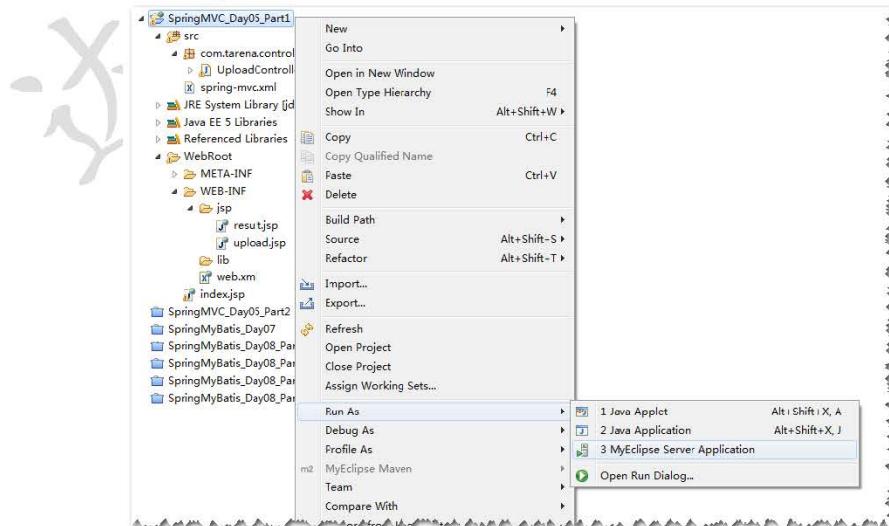


图-43

打开浏览器输入 URL :

[http://localhost:8080/SpringMVC\\_Day05\\_Part1/index.jsp](http://localhost:8080/SpringMVC_Day05_Part1/index.jsp), 如图-44 所示 :

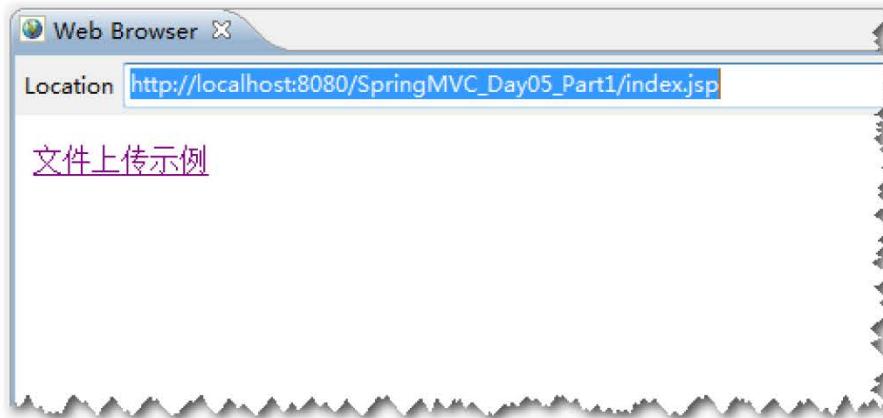


图-44

点击 “文件上传示例” 链接，显示图-45 界面：

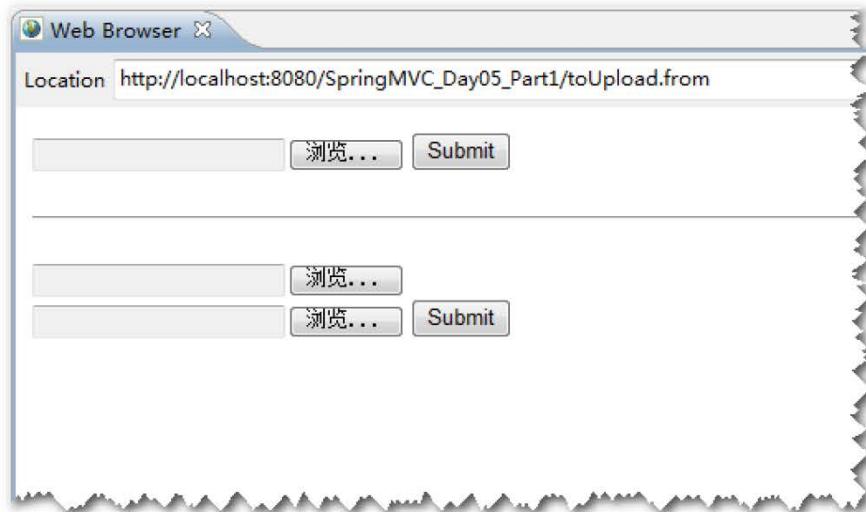


图-45

选择一个文件大小超过 “ 51200byte” 文本文件并点击 “Submit”，如图-46 所示：

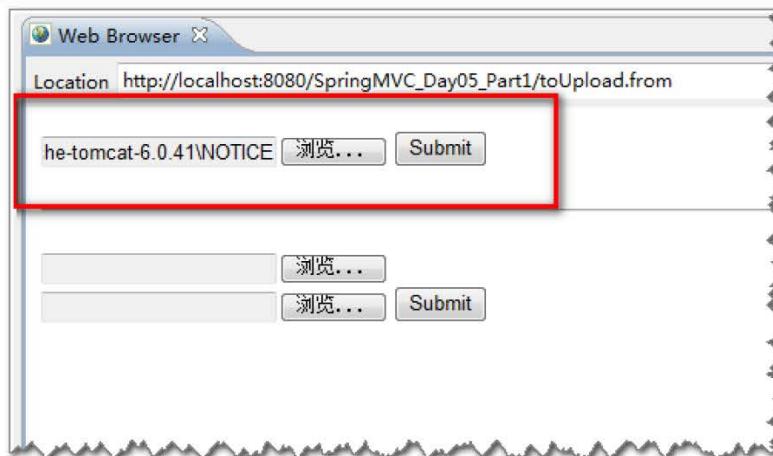


图-46

提交后显示如图-47 的出错页面，则测试成功：

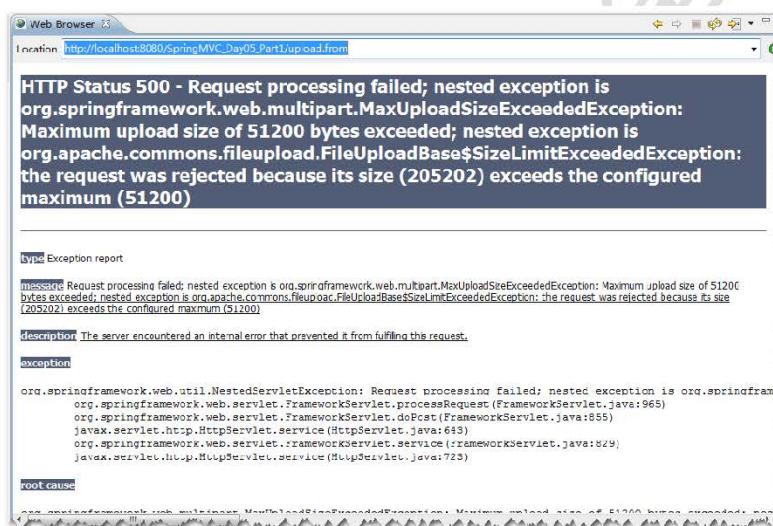


图-47

### • 完整代码

Spring 配置文件 `spring-mvc.xml` 如下所示：

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jdbc"
```

```
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <context:component-scan base-package="com.tarena.controller" />

    <mvc:annotation-driven />

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    >
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <bean id="multipartResolver"
        class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
    >
        <property name="maxUploadSize" value="51200"></property>
        <property name="resolveLazily" value="true" />
    </bean>
</beans>
```

## 课后作业

1. 描述利用 Spring MVC 如何实现登录检查控制。
2. 介绍几种 Spring MVC 的异常处理方法。

达内IT培训集团