Spring MVC 学习笔记

Spring MVC 概述

Spring MVC是Spring提供的一个强大而灵活的web框架。

借助于注解，Spring MVC提供了几乎是POJO的开发模式，使得控制器的开发和测试更加简单。

这些控制器一般不直接处理请求，而是将其委托给Spring上下文中的其他bean，通过Spring的依赖注入功能，这些bean被注入到控制器中。

Spring MVC 主要由**DispatcherServlet**、**处理器映射**、**处理器(控制器)**、**视图解析器**、**视图**组成。

Spring MVC 两个核心：

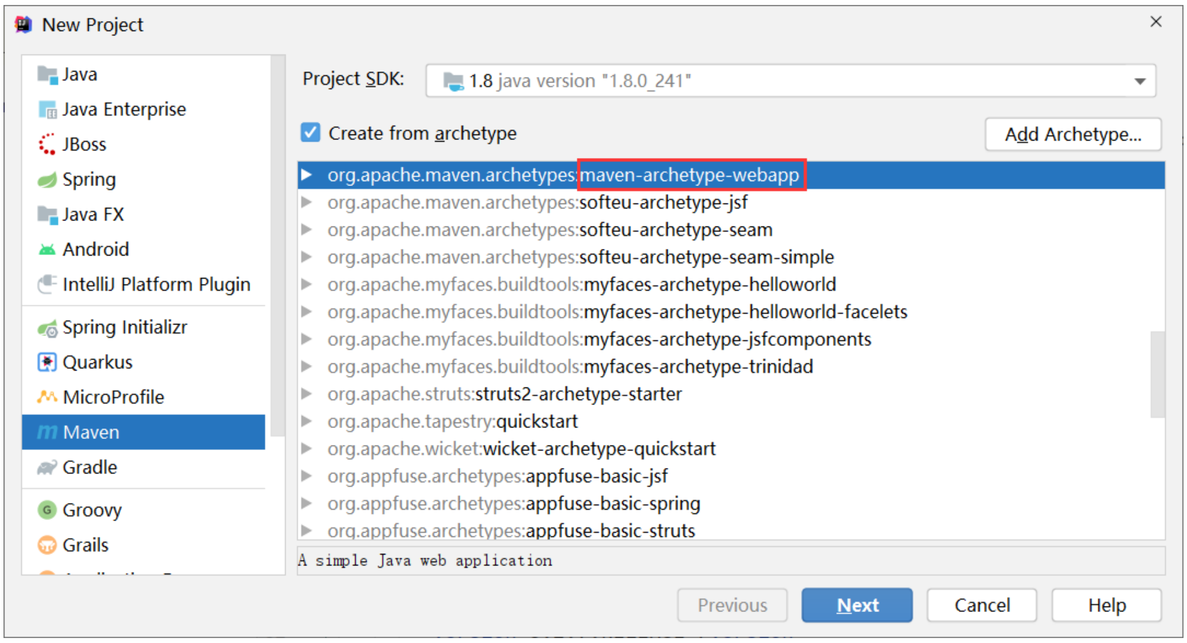
- **处理器映射**：选择使用哪个控制器来处理请求 。
- **视图解析器**：选择结果应该如何渲染。

通过以上两点，Spring MVC保证了如何选择**控制处理请求**和如何选择**视图展现输出**之间的松耦合。

一、开发准备

1. 新建一个Maven Web项目

新建 `maven-archetype-webapp` 模板项目：

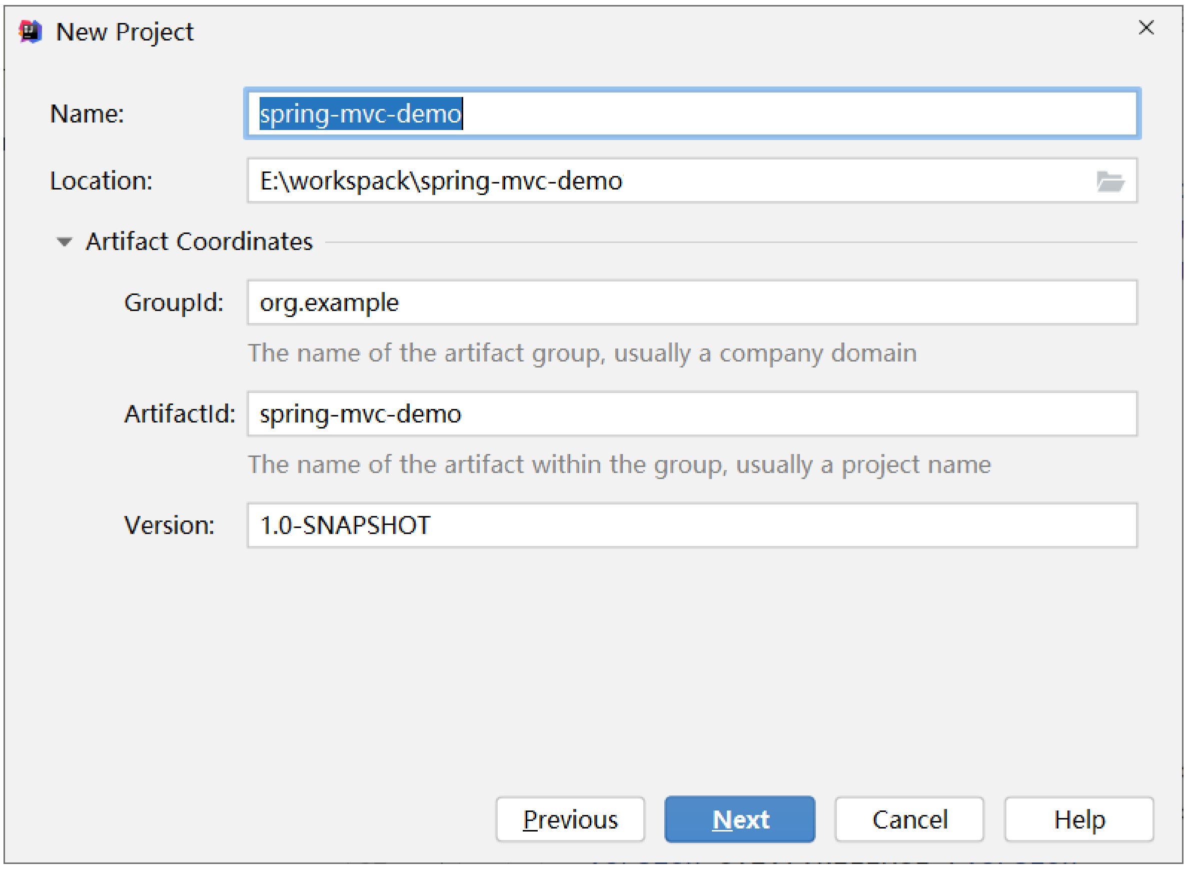


Project SDK: 1.8 java version "1.8.0_241"

☑ Create from archetype

- org.apache.maven.archetypes:maven-archetype-webapp
- org.apache.maven.archetypes:softeu-archetype-jsf
- org.apache.maven.archetypes:softeu-archetype-seam
- org.apache.maven.archetypes:softeu-archetype-seam-simple
- org.apache.myfaces.buildtools:myfaces-archetype-helloworld
- org.apache.myfaces.buildtools:myfaces-archetype-helloworld-facelets
- org.apache.myfaces.buildtools:myfaces-archetype-jsfcomponents
- org.apache.myfaces.buildtools:myfaces-archetype-trinidad
- org.apache.struts:struts2-archetype-starter
- org.apache.tapestry:quickstart
- org.apache.wicket:wicket-archetype-quickstart
- org.appfuse.archetypes:appfuse-basic-jsf
- org.appfuse.archetypes:appfuse-basic-spring
- org.appfuse.archetypes:appfuse-basic-struts

```
A simple Java web application
```



Name: spring-mvc-demo

Location: E:\workspack\spring-mvc-demo

Artifact Coordinates

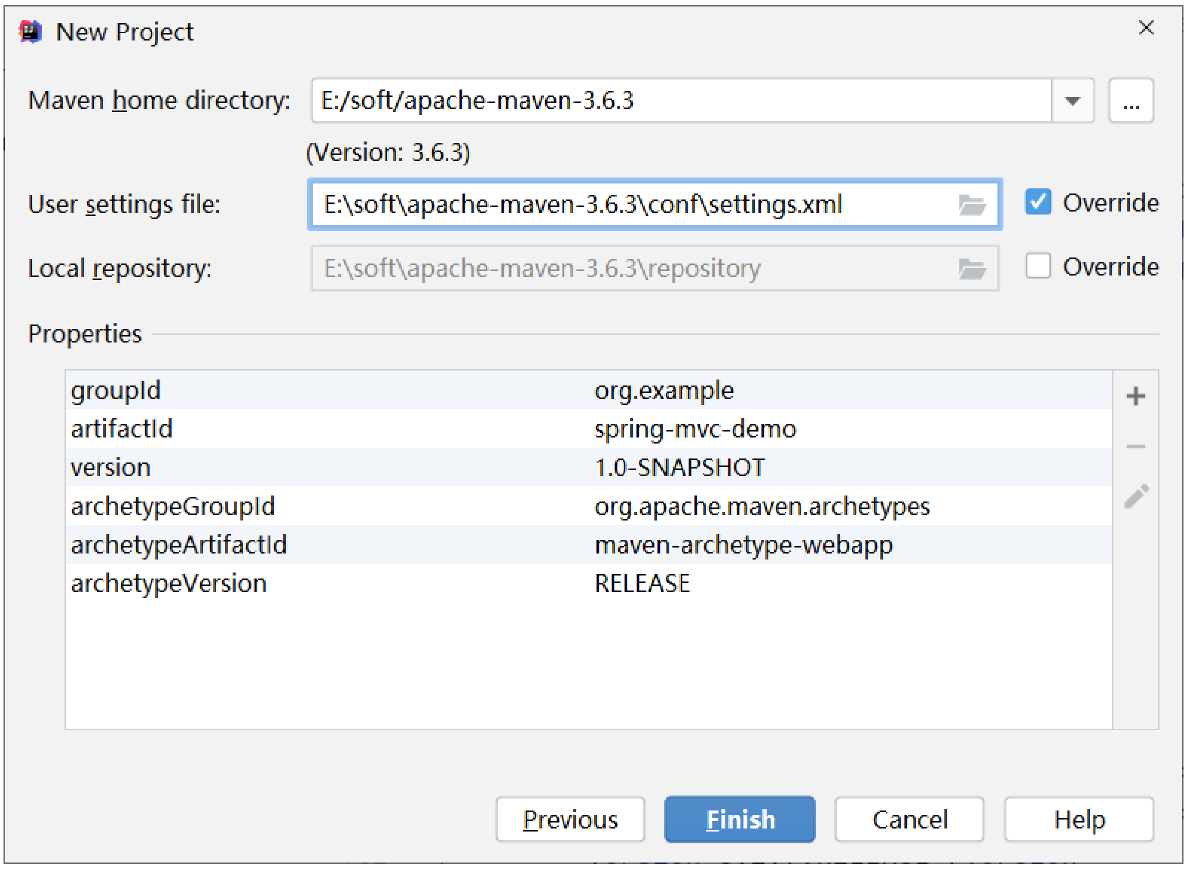
GroupId: org.example

The name of the artifact group, usually a company domain

ArtifactId: spring-mvc-demo

The name of the artifact within the group, usually a project name

Version: 1.0-SNAPSHOT



Maven home directory: E:/soft/apache-maven-3.6.3

(Version: 3.6.3)

User settings file: E:\soft\apache-maven-3.6.3\conf\settings.xml [✓ Override]

Local repository: E:\soft\apache-maven-3.6.3\repository Override

Properties

groupId	org.example
artifactId	spring-mvc-demo
version	1.0-SNAPSHOT
archetypeGroupId	org.apache.maven.archetypes
archetypeArtifactId	maven-archetype-webapp
archetypeVersion	RELEASE

2. 添加项目依赖及配置文件

2.1 完整的pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>spring-mvc-demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>spring-mvc-demo Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>

    <tomcat.port>8080</tomcat.port>

    <spring.version>5.2.7.RELEASE</spring.version>
    <jsp-api.version>2.3.3</jsp-api.version>
    <servlet-api.version>3.1.0</servlet-api.version>
    <jstl.version>1.2</jstl.version>
    <taglibs.version>1.1.2</taglibs.version>

    <junit.version>4.13</junit.version>
  </properties>

  <dependencies>
    <!-- compile范围:默认，是指编译范围内有效,在编译和打包时都会将依赖存储进去。 -->
    <!-- spring-webmvc framework -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- provided范围:在编译和测试过程中有效,最后生成的war包时不会加入。-->
    <!-- jsp+servlet支持  -->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>${jsp-api.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>${servlet-api.version}</version>
      <scope>provided</scope>
    </dependency>
    <!-- jstl 和 standard taglibs-->
    <dependency>
      <groupId>javax.servlet.jsp.jstl</groupId>
      <artifactId>jstl</artifactId>
```

```
        <version>${jstl.version}</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>${taglibs.version}</version>
      </dependency>
      <!-- test范围:是指测试范围有效,在编译和打包时都不会使用这个依赖。 -->
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
      </dependency>

      <!-- runtime范围:在运行时候依赖,在编译时候不依赖。-->
    </dependencies>

    <build>
      <finalName>spring-mvc-demo</finalName>
        <plugins>

          <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
              <source>${maven.compiler.source}</source>
              <target>${maven.compiler.target}</target>
            </configuration>
          </plugin>

          <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.2.2</version>
          </plugin>

          <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
            <configuration>
              <port>${tomcat.port}</port>
              <uriEncoding>${project.build.sourceEncoding}</uriEncoding>
            </configuration>
          </plugin>
        </plugins>
    </build>
  </project>
```

2.1.1 增加`spring-webmvc`依赖:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.7.RELEASE</version>
</dependency>
```

2.1.2 使用JSP和Servlet,JSTL和标准标签的 需要添加以下依赖：

```
<!-- jsp-api 和 servlet-api-->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
 <!-- jstl 和 standard taglibs-->
<dependency>
    <groupId>javax.servlet.jsp.jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
</dependency>
```

1.**test**范围：是指测试范围有效,在编译和打包时都不会使用这个依赖。
2.**compile**范围：**默认**，是指编译范围内有效,在编译和打包时都会将依赖存储进去。
3.**provided**范围：在编译和测试过程中有效,最后生成的war包时不会加入 。
4.**runtime**范围：在运行时候依赖,在编译时候不依赖。

例如:servlet-api,因为servlet-api tomcat服务器已经存在了,如果再打包会冲突，所以是provided。

2.1.4 增加tomcat7-maven-plugin插件

```
<build>
    <finalName>spring-mvc-demo</finalName>
      <plugins>

        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
          <configuration>
            <source>${maven.compiler.source}</source>
            <target>${maven.compiler.target}</target>
          </configuration>
        </plugin>

        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.2.2</version>
        </plugin>

        <plugin>
          <groupId>org.apache.tomcat.maven</groupId>
          <artifactId>tomcat7-maven-plugin</artifactId>
          <version>2.2</version>
          <configuration>
            <port>8080</port>
            <uriEncoding>UTF-8</uriEncoding>
          </configuration>
        </plugin>
      </plugins>
  </build>
```

2.2 修改web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">

  <display-name>spring-mvc-demo</display-name>

  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

升级为web 3.1的配置 并 配置spring mvc的DispatcherServlet。

DispatcherServlet是整个Spring MVC的核心。它负责接收HTTP请求组织协调Spring MVC的各个组成部分。

其主要工作有以下三项：

- （1）截获符合特定格式的URL请求。
- （2）初始化DispatcherServlet上下文对应WebApplicationContext，并将其与业务层、持久化层的WebApplicationContext建立关联。
- （3）初始化Spring MVC的各个组成组件，并装配到DispatcherServlet中。

> servlet-name为**springmvc**,默认自动到src\main\webapp\WEB-INF目录下加载：
>
> `servlet-name`+`-servlet.xml` 后缀的配置文件 `springmvc-servlet.xml`。

不使用默认配置文件可以指定自定义配置文件：

```
<servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.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>
```

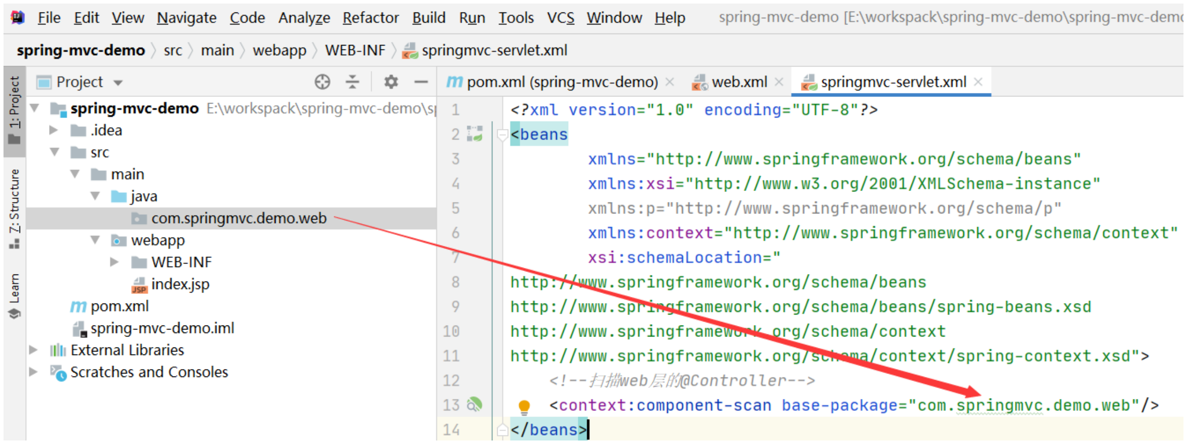
2.3 增加spring mvc配置文件

在src\main\webapp\WEB-INF目录下创建`springmvc-servlet.xml`。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

    <!--扫描web层的@Controller-->
    <context:component-scan base-package="com.springmvc.demo.web"/>

</beans>
```

增加了mvc的命名空间 用于 编写mvc的标签配置。

注意：根据自己的项目的包路径来配置`component-scan`扫描Web层。需要创建对应的包目录：


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
spring-mvc-demo [E:\workspack\spring-mvc-demo\spring-mvc-demo
spring-mvc-demo 〉 src 〉 main 〉 webapp 〉 WEB-INF 〉 springmvc-servlet.xml
Project
pom.xml (spring-mvc-demo)
web.xml
springmvc-servlet.xml
1: Project
Z: Structure
Learn

```
spring-mvc-demo  E:\workspack\spring-mvc-demo\s
  .idea
  src
    main
      java
        com.springmvc.demo.web
      webapp
        WEB-INF
        index.jsp
  pom.xml
  spring-mvc-demo.iml
External Libraries
Scratches and Consoles
```

```
<?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:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
    <!--扫描web层的@Controller-->
    <context:component-scan base-package="com.springmvc.demo.web"/>
</beans>
```

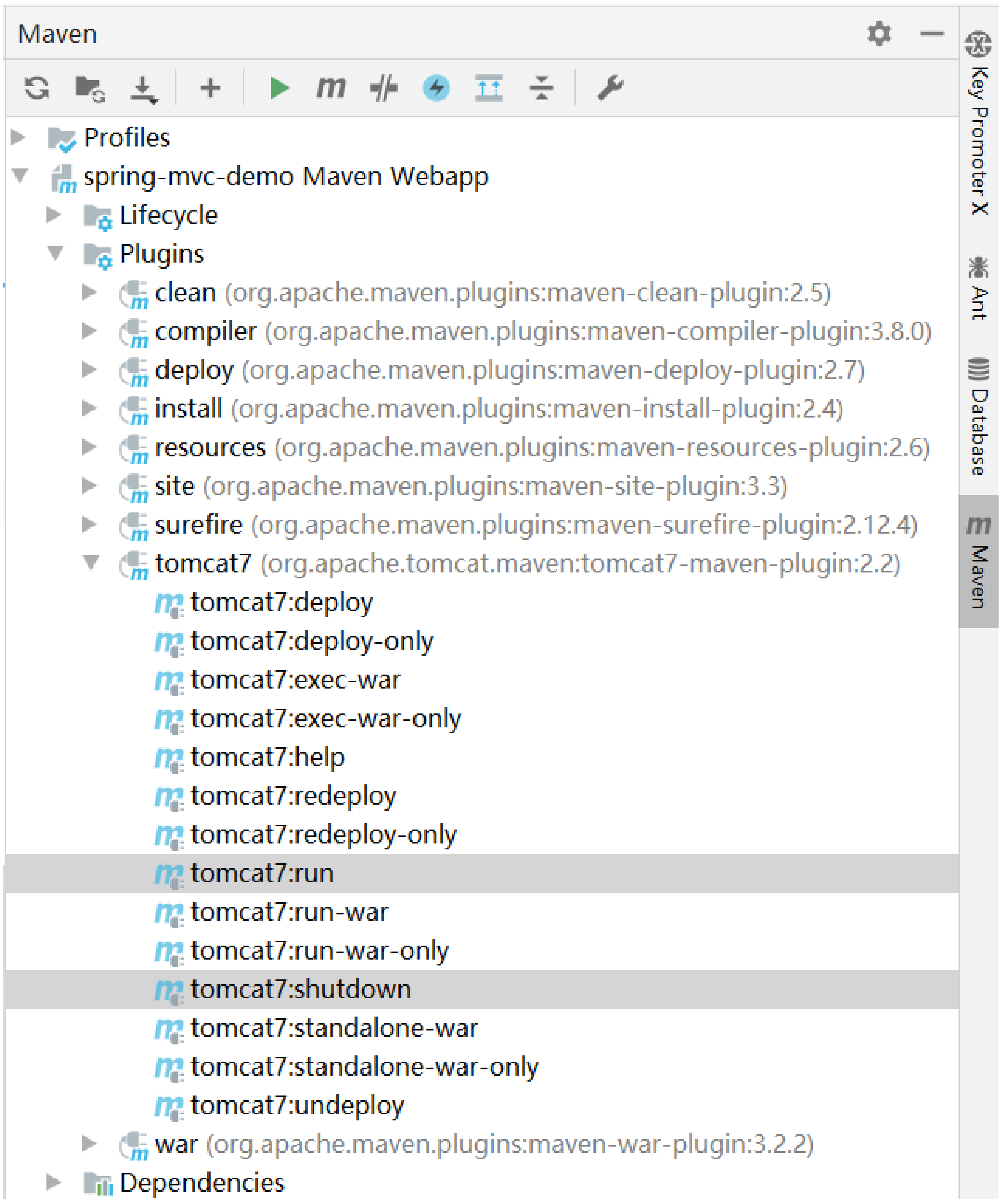
3. 编写处理器(控制器)Controller

```
package com.springmvc.demo.web.ex01;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloContreller {
    @RequestMapping("/hello")
    public  @ResponseBody String sayHello() {
        return "Hello Spring MVC!";
    }
}
```

4. 运行项目

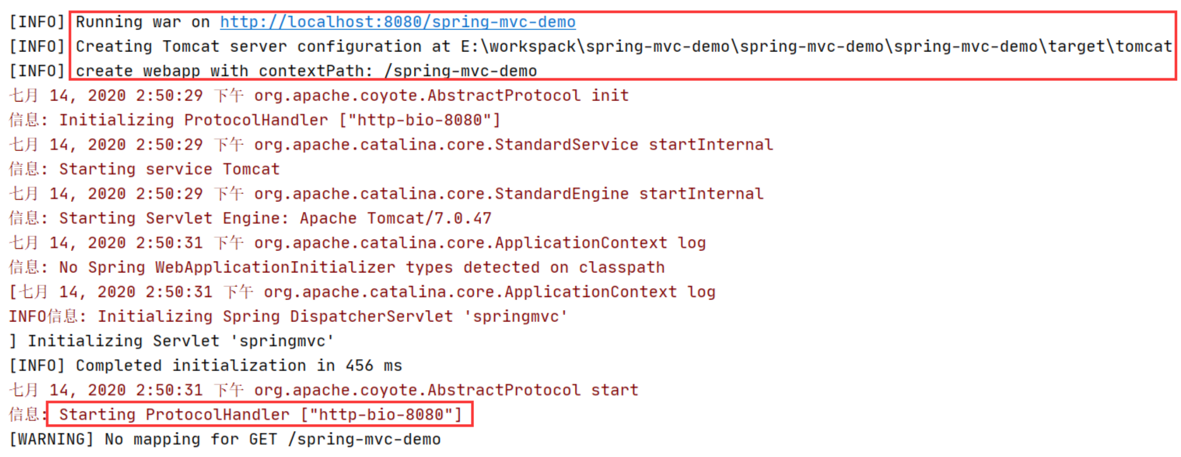
4.1 使用tomcat7-maven-plugin运行项目。

打开Maven选项卡双击tomcat7:run运行项目



```
▶ Profiles
▼ spring-mvc-demo Maven Webapp
    ▶ Lifecycle
    ▼ Plugins
        ▶ clean (org.apache.maven.plugins:maven-clean-plugin:2.5)
        ▶ compiler (org.apache.maven.plugins:maven-compiler-plugin:3.8.0)
        ▶ deploy (org.apache.maven.plugins:maven-deploy-plugin:2.7)
        ▶ install (org.apache.maven.plugins:maven-install-plugin:2.4)
        ▶ resources (org.apache.maven.plugins:maven-resources-plugin:2.6)
        ▶ site (org.apache.maven.plugins:maven-site-plugin:3.3)
        ▶ surefire (org.apache.maven.plugins:maven-surefire-plugin:2.12.4)
        ▼ tomcat7 (org.apache.tomcat.maven:tomcat7-maven-plugin:2.2)
            tomcat7:deploy
            tomcat7:deploy-only
            tomcat7:exec-war
            tomcat7:exec-war-only
            tomcat7:help
            tomcat7:redeploy
            tomcat7:redeploy-only
            tomcat7:run
            tomcat7:run-war
            tomcat7:run-war-only
            tomcat7:shutdown
            tomcat7:standalone-war
            tomcat7:standalone-war-only
            tomcat7:undeploy
        ▶ war (org.apache.maven.plugins:maven-war-plugin:3.2.2)
    ▶ Dependencies
```

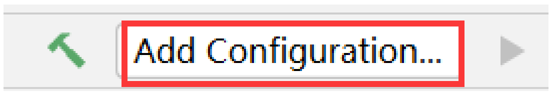
启动成功的控制台信息：

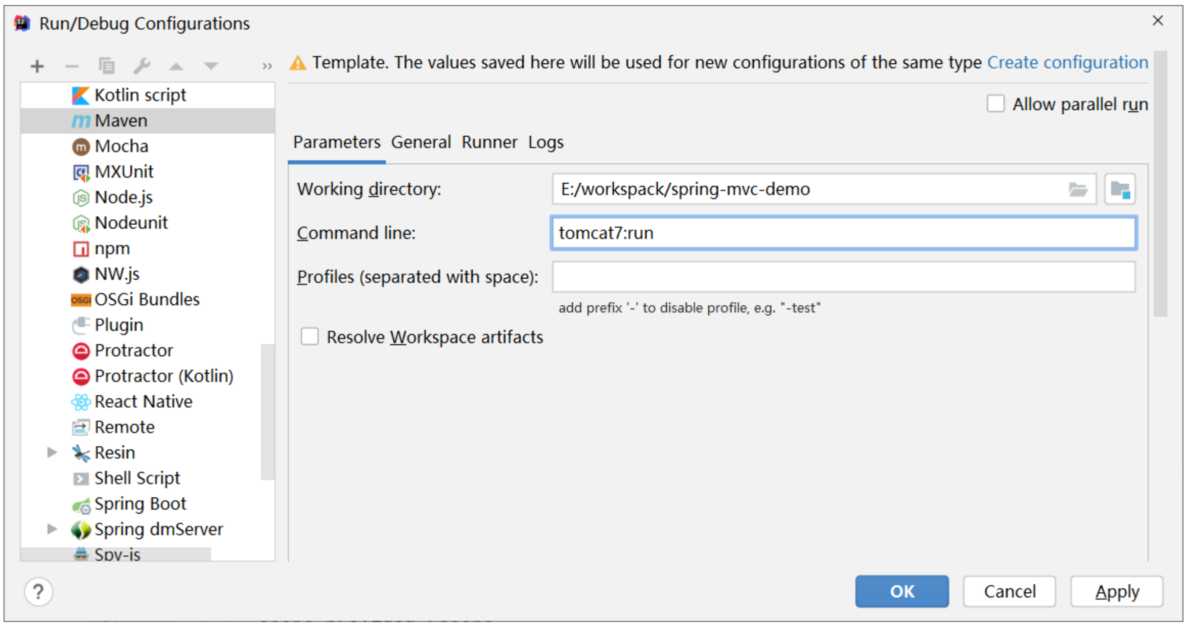


```
[INFO] Running war on http://localhost:8080/spring-mvc-demo
[INFO] Creating Tomcat server configuration at E:\workspack\spring-mvc-demo\spring-mvc-demo\spring-mvc-demo\target\tomcat
[INFO] create webapp with contextPath: /spring-mvc-demo
七月 14, 2020 2:50:29 下午 org.apache.coyote.AbstractProtocol init
信息: Initializing ProtocolHandler ["http-bio-8080"]
七月 14, 2020 2:50:29 下午 org.apache.catalina.core.StandardService startInternal
信息: Starting service Tomcat
七月 14, 2020 2:50:29 下午 org.apache.catalina.core.StandardEngine startInternal
信息: Starting Servlet Engine: Apache Tomcat/7.0.47
七月 14, 2020 2:50:31 下午 org.apache.catalina.core.ApplicationContext log
信息: No Spring WebApplicationInitializer types detected on classpath
[七月 14, 2020 2:50:31 下午 org.apache.catalina.core.ApplicationContext log
INFO信息: Initializing Spring DispatcherServlet 'springmvc'
] Initializing Servlet 'springmvc'
[INFO] Completed initialization in 456 ms
七月 14, 2020 2:50:31 下午 org.apache.coyote.AbstractProtocol start
信息: Starting ProtocolHandler ["http-bio-8080"]
[WARNING] No mapping for GET /spring-mvc-demo
```

可以看到部署的 **项目地址、服务配置、项目部署名** 以及 **端口号**等信息。

4.2 自定义配置-运行项目：




Run/Debug Configurations

Template. The values saved here will be used for new configurations of the same type Create configuration

Allow parallel run

Parameters General Runner Logs

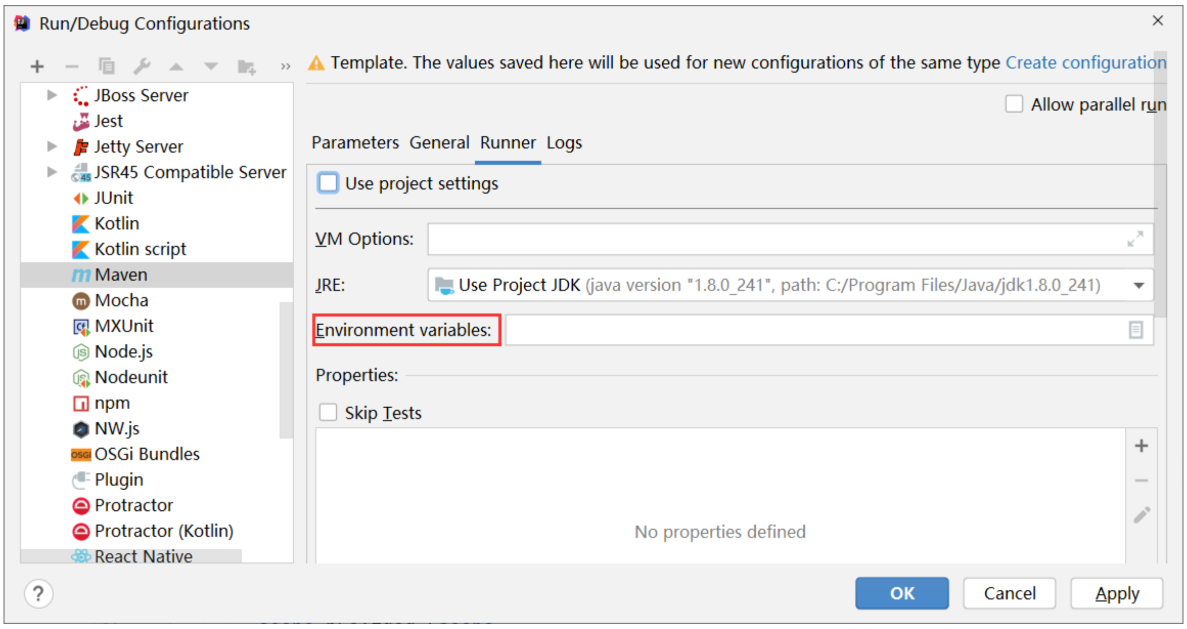
Working directory: E:/workspack/spring-mvc-demo

Command line: tomcat7:run

Profiles (separated with space):

add prefix '-' to disable profile, e.g. "-test"

Resolve Workspace artifacts


Run/Debug Configurations

Template. The values saved here will be used for new configurations of the same type Create configuration

Allow parallel run

Parameters General Runner Logs

Use project settings

VM Options:

JRE: Use Project JDK (java version "1.8.0_241", path: C:/Program Files/Java/jdk1.8.0_241)

Environment variables:

Properties:

Skip Tests

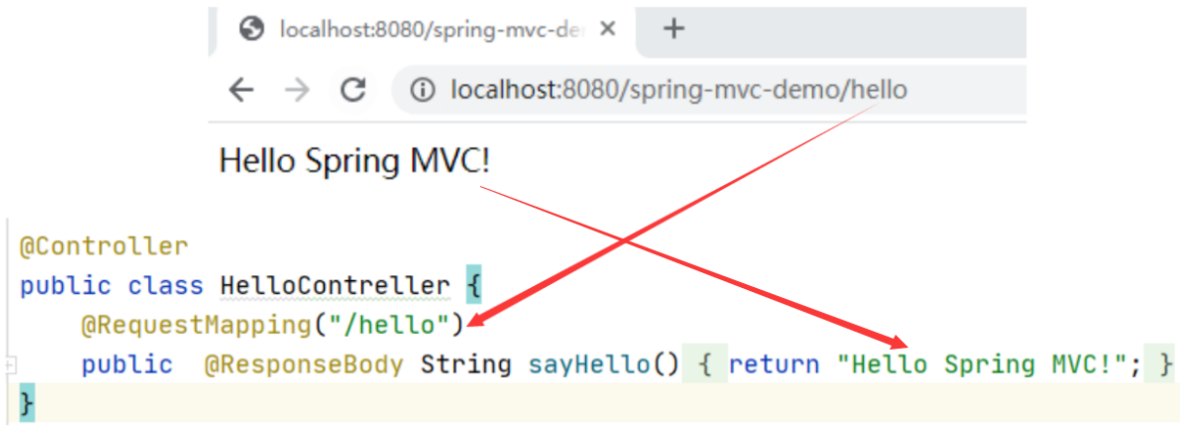
No properties defined

手动配置的好处是可以设置操作系统运行环境的一些参数，例如：当tomcat内存不足导致内存溢出时可以修改环境变量设置，调整启动的内存让程序正常运行。

5. 打开浏览器

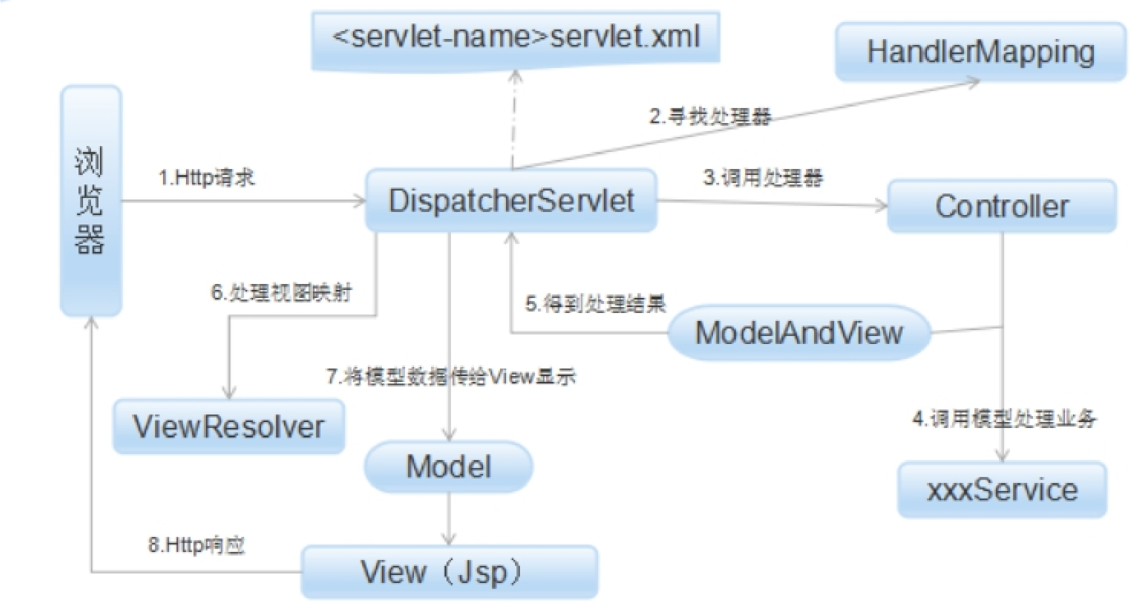
访问： http://localhost:8081/springmvc/hello

浏览器显示：


localhost:8080/spring-mvc-de
localhost:8080/spring-mvc-demo/hello
Hello Spring MVC!

```
@Controller
public class HelloContreller {
    @RequestMapping("/hello")
    public  @ResponseBody String sayHello() { return "Hello Spring MVC!"; }
}
```

6. Spring MVC 运行原理



(1) Http请求：客户端请求提交到DispatcherServlet。
(2) 寻找处理器：由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller。
(3) 调用处理器：DispatcherServlet将请求提交到Controller。
(4)(5)调用业务处理和返回结果：Controller调用业务逻辑处理后，返回ModelAndView。
(6)(7)处理视图映射并返回模型： DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图。
(8) Http响应：视图负责将结果显示到客户端。

7. Spring MVC 接口解释

（1）DispatcherServlet接口： Spring提供的前端控制器，所有的请求都有经过它来统一分发。在DispatcherServlet将请求分发给Spring Controller之前，需要借助于Spring提供的HandlerMapping定位到具体的Controller。
（2）HandlerMapping接口： 能够完成客户请求到Controller映射。
（3）Controller接口：
需要为并发用户处理上述请求，因此实现Controller接口时，必须保证线程安全并且可重用。
Controller将处理用户请求，这和Struts Action扮演的角色是一致的。一旦Controller处理完用户请求，则返回ModelAndView对象给DispatcherServlet前端控制器，ModelAndView中包含了模型

（Model）和视图（View）。
从宏观角度考虑，DispatcherServlet是整个Web应用的控制器；从微观考虑，Controller是单个Http请求处理过程中的控制器，而ModelAndView是Http请求过程中返回的模型（Model）和视图（View）。
（4）ViewResolver接口：
Spring提供的视图解析器（ViewResolver）在Web应用中查找View对象，从而将相应结果渲染给客户。

二. 使用 Jquery ui搭建项目前端

---

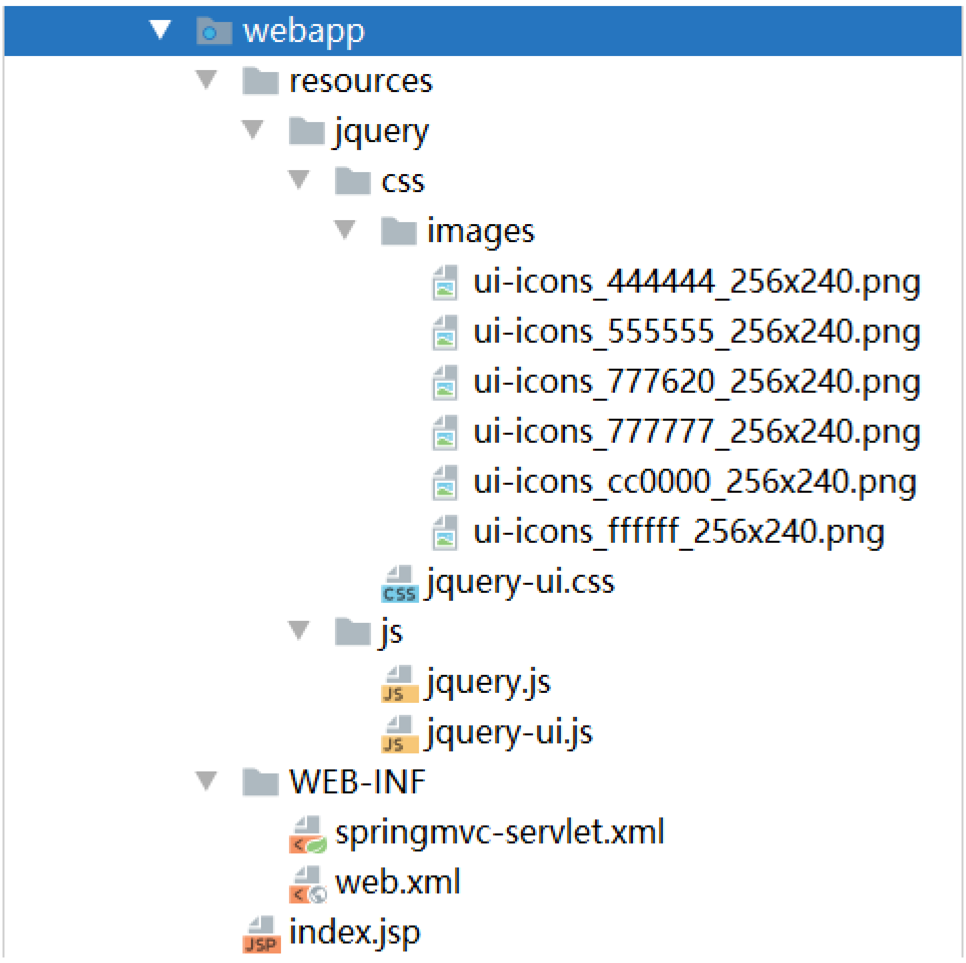
1. 下载jquery-ui库

https://jqueryui.com/download/

快速下载连接： Stable (Themes) (1.12.1: *for jQuery1.7+*)

下载完成后解压缩，可打开index.html页面学习ui库的常用组件。

2. 复制文件到webapp目录

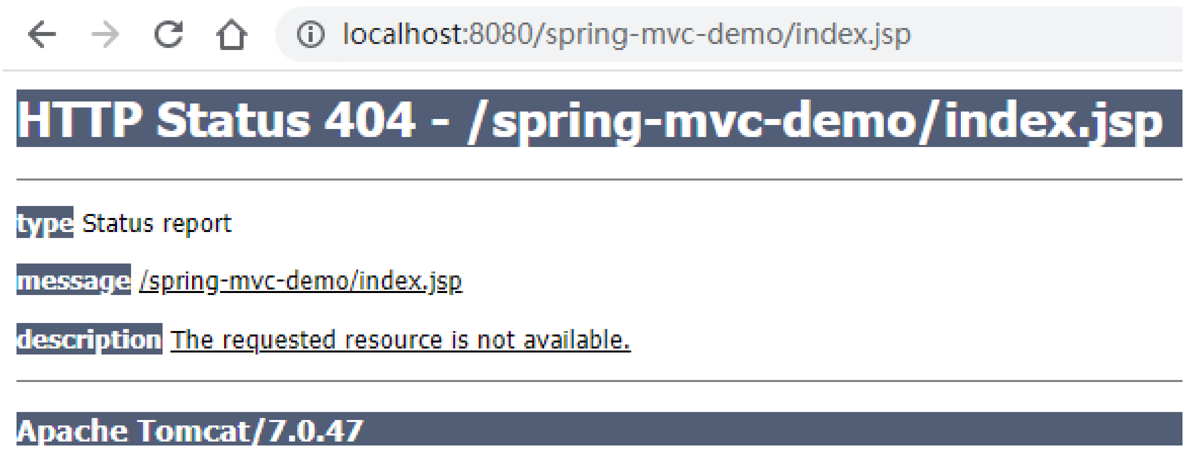


```
webapp
  resources
    jquery
      css
        images
          ui-icons_444444_256x240.png
          ui-icons_555555_256x240.png
          ui-icons_777620_256x240.png
          ui-icons_777777_256x240.png
          ui-icons_cc0000_256x240.png
          ui-icons_ffffff_256x240.png
        jquery-ui.css
      js
        jquery.js
        jquery-ui.js
  WEB-INF
    springmvc-servlet.xml
    web.xml
  index.jsp
```

3. 呈现视图 Rendering Views

3.1 直接访问index.jsp页面

发现404错误：


localhost:8080/spring-mvc-demo/index.jsp

HTTP Status 404 - /spring-mvc-demo/index.jsp

type Status report

message /spring-mvc-demo/index.jsp

description The requested resource is not available.

Apache Tomcat/7.0.47

WEB-INF目录作用

WEB-INF是Java的WEB应用的安全目录。所谓安全就是客户端无法直接访问，只有服务端可以访问的目录。如果想在浏览器中直接访问其中的文件，必须通过web.xml文件对要访问的文件进行相应映射才能访问。

3.2 设置系统默认首页(视图)

修改`springmvc-servlet.xml`配置

```
<!-- 将“/”请求映射到“index”视图 -->
<mvc:view-controller path="/" view-name="index"/>
<!-- 内部资源视图解析程序 将@Controllers的返回值 或 view-controller解析为/WEB-INF/views目录中的.jsp视图资源 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

上述配置就是访问/时返回index视图，然后被**内部资源视图解析程序**匹配前后缀后响应`/WEB-INF/views/index.jsp`主页。

注意：必须把`index.jsp`首页移动到`/WEB-INF/views/`目录下。

```
<mvc:view-controller path="/" view-name="index"/>
```

等效于：

```
package com.springmvc.demo.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class ViewController {
    @GetMapping("/")
    public String index(){
        return "index";
    }
}
```

3.3 修改web.xml

因为上面的配置需要访问`path="/"`,所以需要把DispatcherServlet的拦截`/*`修改为`/`，否则无法拦截`/`。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">

  <display-name>spring-mvc-demo</display-name>

  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

重新部署后，访问超链接`http://localhost:8081/springmvc/hello`发现404请求错误：

在web.xml中servlet-mapping的url-pattern设置的是/，而不是如.do。表示将所有的文件，包含静态资源文件都交给spring mvc处理。就需要用到`<mvc:annotation-driven />`了。如果不加，DispatcherServlet则无法区分请求是资源文件还是mvc的注解，而导致controller的请求报404错误。

修改springmvc-servlet.xml配置文件,增加`<mvc:annotation-driven />`配置：

```
<?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:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="
```

```
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">


    <!--扫描web层的@Controller-->
    <context:component-scan base-package="com.springmvc.demo.web"/>

    <!-- 将“/”请求映射到“index”视图 -->
    <mvc:view-controller path="/" view-name="index"/>
    <!-- 将@Controllers的返回值用于呈现的视图解析为/WEB-INF/views目录中的.jsp资源 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!--启用注解驱动:自动将扫描到的@Component，@Controller，@Service，@Repository等注
解标记的组件注册到工厂中，来处理我们的请求。-->
    <mvc:annotation-driven/>
</beans>
```

3.4 首页实现

3.4.1 创建样式文件

`src\main\webapp\resources\messages\messages.css` 。

用于ajax请求的提示信息(字体颜色、背景颜色、背景图片等)样式。

注意：图片部分在案例源码中提供。

```
div.info,div.success,div.warning,div.error {
    border: 1px solid;
    margin: 10px 0px;
    padding: 15px 10px 15px 50px;
    background-repeat: no-repeat;
    background-position: 10px center;
}

div.info {
    color: #00529B;
    background-color: #BDE5F8;
    background-image: url('info.png');
}

div.success {
    color: #4F8A10;
    background-color: #DFF2BF;
    background-image: url('success.png');
}

div.warning {
    color: #9F6000;
    background-color: #FEEFB3;
    background-image: url('warning.png');
}

div.error {
    color: #D8000C;
    background-color: #FFBABA;
    background-image: url('error.png');
}

span {
    margin: 5px 5px 5px 5px;
}

span.success {
    color: #4F8A10;
}

span.error {
    color: #D8000C;
}

body {
    font-family: Lucida Grande, sans-serif;
    font-size: .75em;
    margin: 1em auto;
}

ul li {
    padding: 5px;
}
```

3.4.2创建响应信息工具类

`resources/util/mvc-util.js`

提供显示响应信息(带样式)、xml字符实体转换 等方法 并把信息绑定到请求元素的后面。

```
MvcUtil = {};
//显示成功响应信息  响应文本内容,页面请求元素（A、FROM）
MvcUtil.showSuccessResponse = function (text, element) {
    MvcUtil.showResponse("success", text, element);
};
//显示错误响应信息  响应文本内容,页面请求元素（A、FROM）
MvcUtil.showErrorResponse = function showErrorResponse(text, element) {
    MvcUtil.showResponse("error", text, element);
};

//显示响应信息  响应类型（"success"/"error"）,响应文本内容,页面请求元素（A、FROM）
//根据type参数 绑定显示样式 ： src\main\webapp\resources\messages\messages.css
//span.success {
//    color: #4F8A10;
//}
MvcUtil.showResponse = function(type, text, element) {
    //1.获取 页面元素的 id属性值 拼接 带新的id页面元素
    var responseElementId = element.attr("id") + "Response";
    //2.判断页面元素的 响应元素 是否存在?
    var responseElement = $("#" + responseElementId);

    //3.响应元素不存在 - 创建  一个新的响应元素 并且绑定到 请求的页面元素的后面。
    if (responseElement.length == 0) {
        responseElement = $('<span id="' + responseElementId + '" class="' +
type + '" style="display:none">' + text + '</span>').insertAfter(element);
    } else {
        //3.响应的页面元素存在  替换  为最新的元素。
        responseElement.replaceWith('<span id="' + responseElementId + '"
class="' + type + '" style="display:none">' + text + '</span>');
        responseElement = $("#" + responseElementId);
    }
    responseElement.fadeIn("slow");
};

//XML数据的转换的函数 - 把html的特殊字符 转换为字符实体
MvcUtil.xmlencode = function(xml) {
    //for IE
    var text;
    if (window.ActiveXObject) {
        text = xml.xml;
    }
    // for Mozilla, Firefox, Opera, etc.
    else {
        text = (new XMLSerializer()).serializeToString(xml);
    }
    return text.replace(/\&/g,'&'+'amp;').replace(/</g,'&'+'lt;')

.replace(/>/g,'&'+'gt;').replace(/\'/g,'&'+'apos;').replace(/\"/g,'&'+'quot;');
};
```

3.4.3 创建核心脚本文件index.js

`resources/index.js`

提供Jquery就绪函数: 用于 初始化主视图的tabs组件， 并且给视图绑定ajax请求事件。

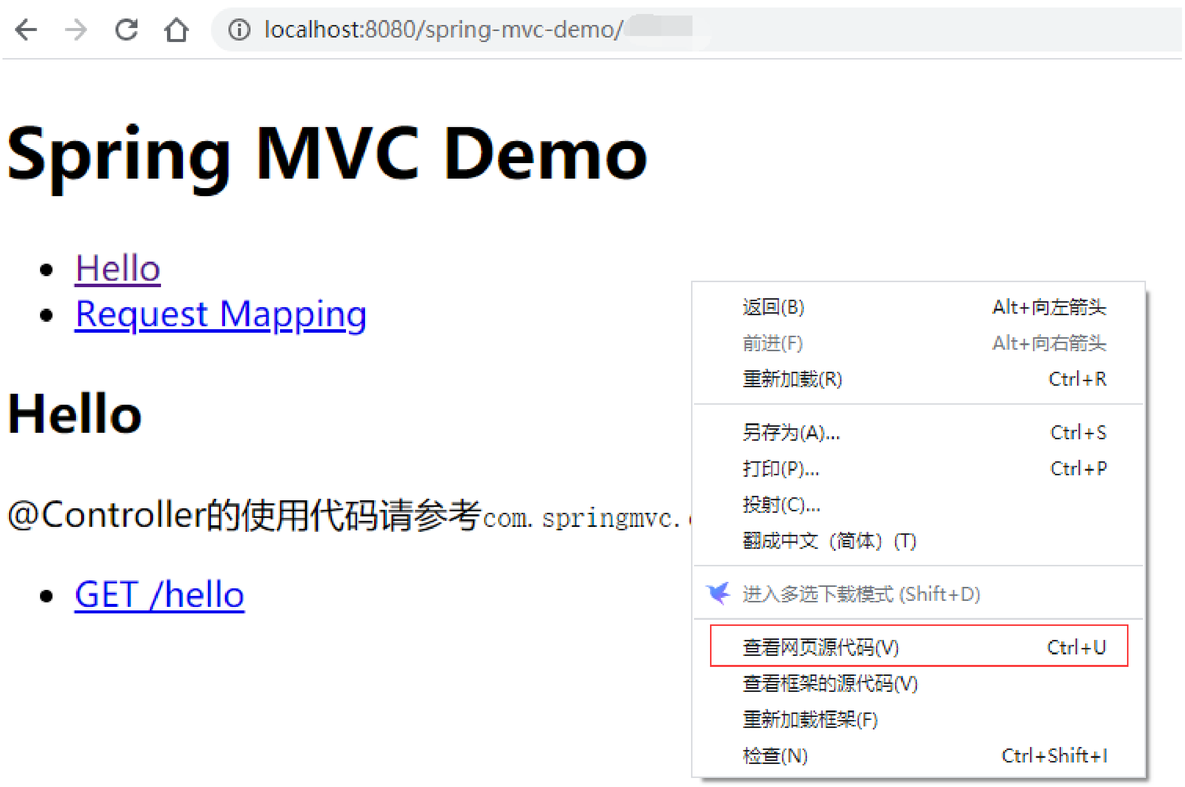
```
$(document).ready(function(){
    console.log("jquery and jquery-ui ready!");
    <!-- tabs组件JS部分代码 -->
    $("#tabs").tabs();

    $("a.textLink").click(function(){
        var link = $(this);
        $.ajax({
            url: link.attr("href"),
            dataType: "text",
            success: function(text) {
                MvcUtil.showSuccessResponse(text, link);
            },
            error: function(xhr) {
                MvcUtil.showErrorResponse(xhr.responseText, link);
            }
        });
        return false;
    });
});
```

注意：后续需要编写更多前后台交互的Ajax方法...

3.4.4 index.jsp首页实现：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<!DOCTYPE html>
<html lang="cn">
<head>
    <meta charset="UTF-8">
    <title>Spring MVC Demo</title>
    <!--CSS样式-->
    <link href="resources/jquery/css/jquery-ui.css" rel="stylesheet">
    <link href="resources/messages/messages.css" rel="stylesheet">

    <!--jquery核心库 和 ui组件库 -->
    <script type="text/javascript" src="resources/jquery/js/jquery.js"></script>
    <script type="text/javascript" src="resources/jquery/js/jquery-ui.js">
</script>

    <!--MvcUtil:提供显示响应信息(带样式)、xml字符实体转换 等方法。 -->
    <script type="text/javascript" src="resources/util/mvc-util.js"></script>
    <!-- Jquery就绪函数: 初始化tabs组件 , 并且给视图绑定ajax请求事件 -->
    <script type="text/javascript" src="resources/index.js"></script>

</head>
<body>
<h1>Spring MVC Demo</h1>

<!-- tabs组件HTML部分代码 -->
<div id="tabs">
    <ul>
        <li><a href="#hello">Hello</a></li>
        <li><a href="#mapping">Request Mapping</a></li>
    </ul>
    <div id="hello">
        <h2>Hello</h2>
        <p>
            @Controller的使用代码请参考
<code>com.springmvc.demo.web.ex01.HelloContreller</code>。
        </p>
        <ul>
            <li>
                <a id="helloLink" class="textLink" href="<c:url value="/hello"
/>">GET /hello</a>
            </li>
        </ul>
    </div>
    <div id="mapping">

    </div>
</div>
</body>
</html>
```

启动项目访问首页发现项目并没有达到预期的效果：



localhost:8080/spring-mvc-demo/

Spring MVC Demo

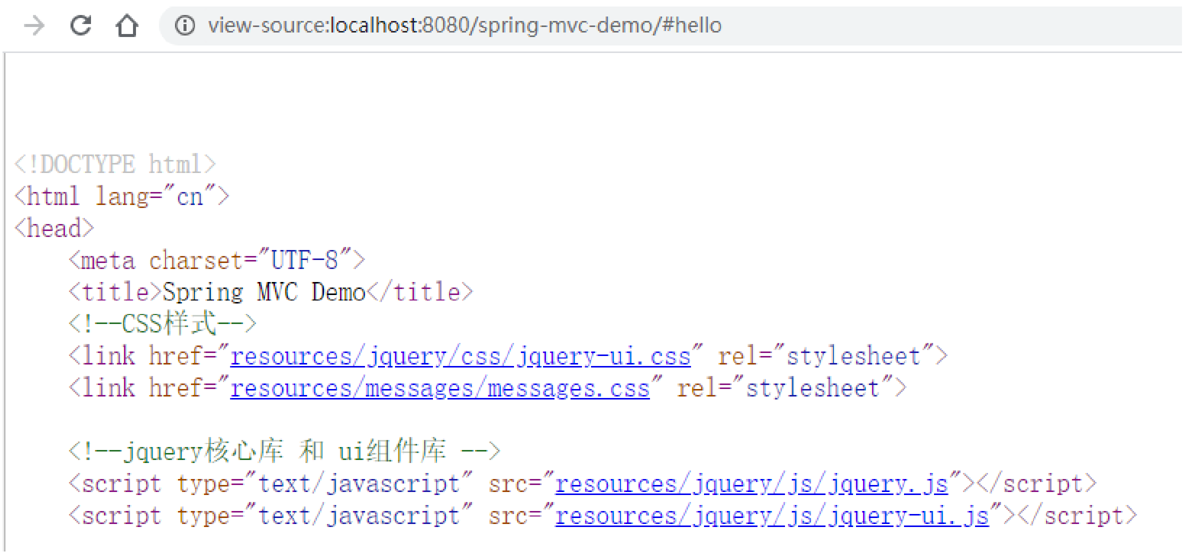
- Hello
- Request Mapping

Hello

@Controller的使用代码请参考com. springmvc.

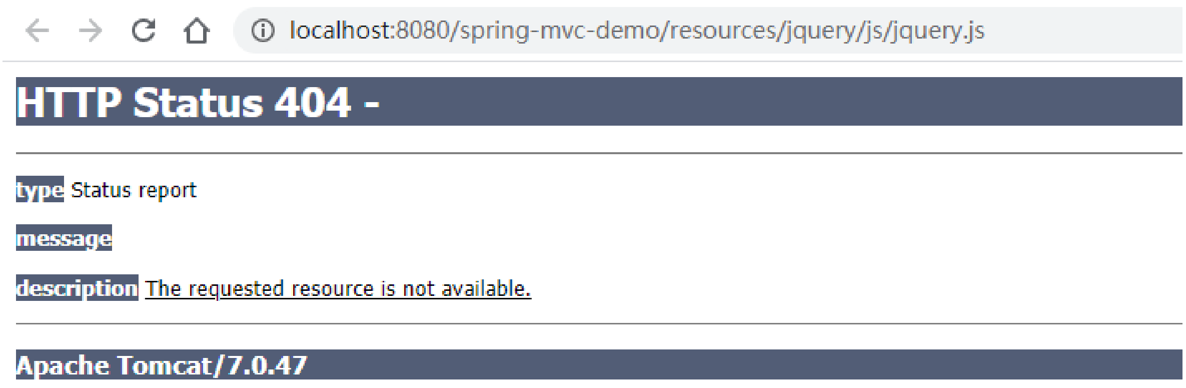
- GET /hello

检查资源文件方法1：**查看网页源代码 并 点击资源文件路径**



view-source:localhost:8080/spring-mvc-demo/#hello

```
<!DOCTYPE html>
<html lang="cn">
<head>
    <meta charset="UTF-8">
    <title>Spring MVC Demo</title>
    <!--CSS样式-->
    <link href="resources/jquery/css/jquery-ui.css" rel="stylesheet">
    <link href="resources/messages/messages.css" rel="stylesheet">

    <!--jquery核心库 和 ui组件库 -->
    <script type="text/javascript" src="resources/jquery/js/jquery.js"></script>
    <script type="text/javascript" src="resources/jquery/js/jquery-ui.js"></script>
```



localhost:8080/spring-mvc-demo/resources/jquery/js/jquery.js

HTTP Status 404 -

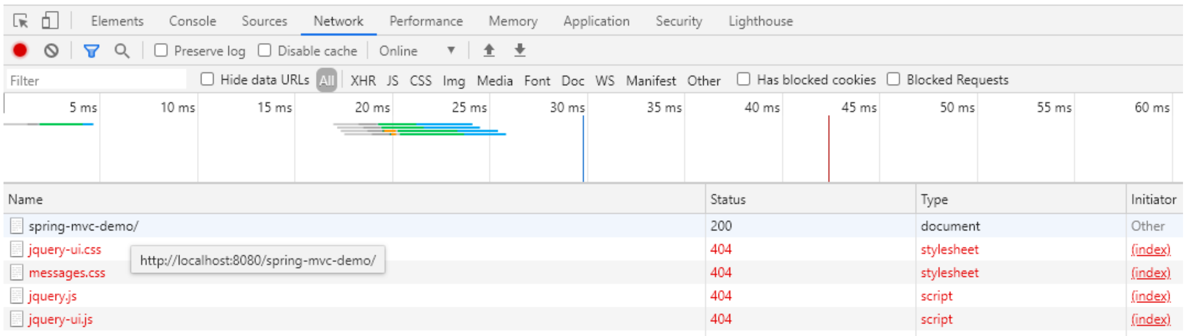
type Status report

message

description The requested resource is not available.

Apache Tomcat/7.0.47

检查资源文件方法2：**F12打开开发者工具查看**Network**网络(F5刷新)**：发现所有资源文件都无法访问：**



Name	Status	Type	Initiator
spring-mvc-demo/	200	document	Other
jquery-ui.css	404	stylesheet	(index)
messages.css	404	stylesheet	(index)
jquery.js	404	script	(index)
jquery-ui.js	404	script	(index)

4. Spring MVC 静态资源访问

在SpringMVC中常用的就是Controller与View。但是我们常常会需要访问静态资源，如html、js、css、image等。默认的访问的URL都会被DispatcherServlet所拦截，所以jsp或者html中则无法访问静态资源文件（js、css、image）。

解决方案1：直接在spring mvc配置文件中spirngmvc-servlet.xml中添加资源映射。

```
<mvc:resources mapping="/resources/**" location="/resources/" />
<mvc:resources mapping="/images/**" location="/images/" />
<mvc:resources mapping="/js/**" location="/js/" />
```

mapping：映射,请求访问的url地址。

location：本地资源路径，注意必须是webapp根目录下的路径。

两个`**`，它表示映射resources/下所有的URL，包括子路径（即接多个/）。

WEB-INF目录作用

WEB-INF是Java的WEB应用的安全目录。所谓安全就是客户端无法直接访问，只有服务端可以访问的目录。如果想在页面中直接访问其中的文件，必须通过web.xml文件对要访问的文件进行相应映射才能访问。

当然，你非要放在WEB-INF中，则必须修改resources映射，如：

```
<mvc:resources mapping="/js/**" location="/WEB-INF/js/" />
```

解决方案2: 使用Servlet 容器默认控制器

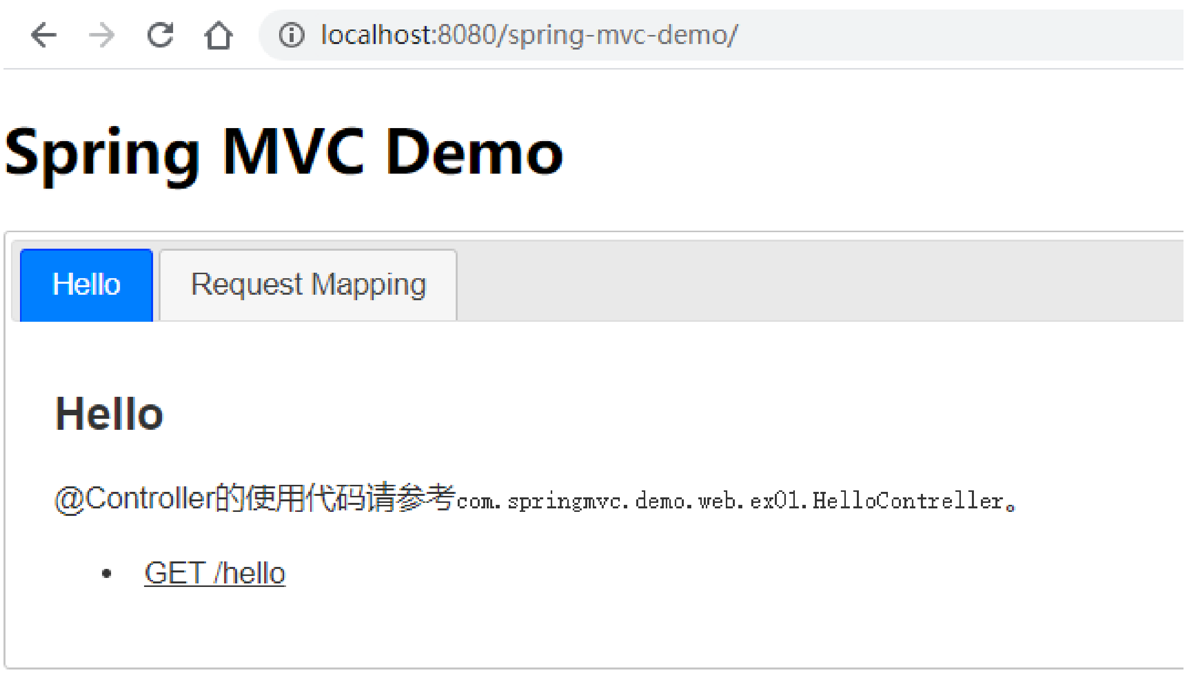
```
<mvc:default-servlet-handler/>
```

会把 "/**" url,注册到 SimpleUrlHandlerMapping 的 urlMap 中,把对静态资源的访问由 HandlerMapping 转到 org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler 处理并返回.

DefaultServletHttpRequestHandler 使用就是各个 Servlet 容器自己的默认 Servlet.

SpringMVC访问静态资源的三种方式

配置完成后，重新部署项目发现成功显示jqueryui组件了：



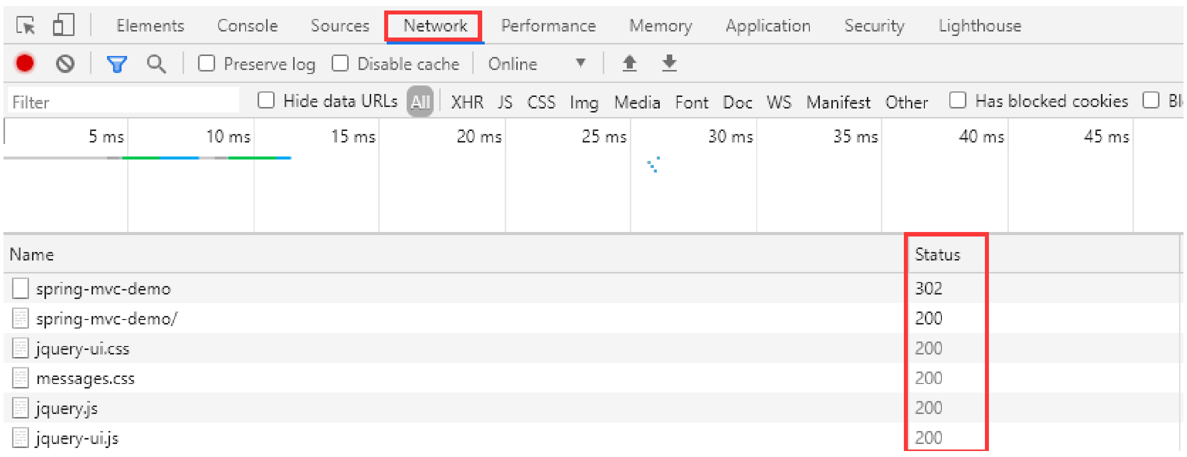
Spring MVC Demo

Hello

@Controller的使用代码请参考com.springmvc.demo.web.ex01.HelloContreller。

- GET /hello

网络请求状态从404变成200：



Name	Status
spring-mvc-demo	302
spring-mvc-demo/	200
jquery-ui.css	200
messages.css	200
jquery.js	200
jquery-ui.js	200

三、Mapping Requests(请求映射)

1. Entity层实现代码

```
package com.springmvc.demo.web.ex02;

import javax.xml.bind.annotation.XmlRootElement;

public class JavaBean {
   private String foo = "bar";

   private String fruit = "apple";

   public String getFoo() {
      return foo;
   }

   public void setFoo(String foo) {
      this.foo = foo;
   }

   public String getFruit() {
      return fruit;
   }

   public void setFruit(String fruit) {
      this.fruit = fruit;
   }

   @Override
   public String toString() {
      return "JavaBean {foo=[" + foo + "], fruit=[" + fruit + "]}";
   }
}
```

2. Web层实现代码

```
package com.springmvc.demo.web.ex02;

import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;

@Controller
public class MappingController {

    @RequestMapping("/mapping/path")
    public @ResponseBody
    String byPath() {
        return "Mapped by path!";
    }

    @RequestMapping(value="/mapping/path/*", method= RequestMethod.GET)
    public @ResponseBody
    String byPathPattern(HttpServletRequest request) {
        return "Mapped by path pattern ('" + request.getRequestURI() + "')";
    }

    @RequestMapping(value="/mapping/method", method= RequestMethod.GET)
    public @ResponseBody
    String byMethod() {
        return "Mapped by path + method";
    }

    @RequestMapping(value="/mapping/parameter", method= RequestMethod.GET,
params="foo")
    public @ResponseBody
    String byParameter() {
        return "Mapped by path + method + presence of query parameter!";
    }

    @RequestMapping(value="/mapping/parameter", method= RequestMethod.GET,
params="!foo")
    public @ResponseBody
    String byParameterNegation() {
        return "Mapped by path + method + not presence of query parameter!";
    }

    @RequestMapping(value="/mapping/header", method= RequestMethod.GET,
headers="FooHeader=foo")
    public @ResponseBody
    String byHeader() {
        return "Mapped by path + method + presence of header!";
    }

    @RequestMapping(value="/mapping/header", method= RequestMethod.GET,
headers="!FooHeader")
    public @ResponseBody
    String byHeaderNegation() {
        return "Mapped by path + method + absence of header!";
    }

    @RequestMapping(value="/mapping/consumes", method= RequestMethod.POST,
consumes= MediaType.APPLICATION_JSON_VALUE)
    public @ResponseBody
    String byConsumesJson(@RequestBody JavaBean javaBean) {
        return "Mapped by path + method + consumable media type (javaBean '" +
javaBean + "')";
    }

    @RequestMapping(value="/mapping/produces", method= RequestMethod.GET,
produces= MediaType.APPLICATION_JSON_VALUE)
    public @ResponseBody
    JavaBean byProducesJson() {
        return new JavaBean();
    }

    @RequestMapping(value="/mapping/produces", method= RequestMethod.GET,
produces= MediaType.APPLICATION_XML_VALUE)
    public @ResponseBody
    JavaBean byProducesXml() {
        return new JavaBean();
    }

}
```

注解说明:

注解及属性	用途
@Controller	于标识处理器类;
@RequestMapping	请求到处理器功能方法的**映射规则**;
value="/url/`*`"	请求url没有其他属性可以不声明,*代表任意字符串。
method=RequestMethod.GET	指定请求方式。例如POST、GET、PUT、DELETE等。
params="foo"	代表请求**必须带**名为foo的参数。params="!foo"代表请求必须**不带**名为foo的参数。
headers="FooHeader=foo"	请求中**必须包含**名字为FooHeader，值为foo的请求头。headers="!FooHeader" 请求中**不包含**名字为FooHeader的请求头。
consumes	**设置请求接收的数据类型**。例如：`consumes=MediaType.APPLICATION_JSON_VALUE`为JSON。配合@RequestBody把json转换为参数列表中的Bean对象。
produces	**设置响应的数据格式**。例如：`produces=MediaType.APPLICATION_JSON_VALUE`设置响应JSON数据格式。配合@ResponseBody把返回类型Bean转换为JSON。

3. View视图实现代码

修改index.jsp,增加body视图部分代码:

```
<div id="tabs">
    <ul>
        <li><a href="#hello">Hello</a></li>
        <li><a href="#mapping">Request Mapping</a></li>
    </ul>
    <div id="hello">
        <h2>Hello</h2>
        <p>
            @Controller的使用代码请参考
<code>com.springmvc.demo.web.ex01.HelloContreller</code>。
        </p>
        <ul>
            <li>
                <a id="helloLink" class="textLink" href="<c:url value="/hello"
/>">GET /hello</a>
            </li>
        </ul>
    </div>
    <div id="mapping">
        <h2>Request Mapping</h2>
        <p>
            See the <code>org.springframework.samples.mvc.mapping</code> package
for the @Controller code
        </p>
        <ul>
            <li>
                <a id="byPath" class="textLink" href="<c:url
value="/mapping/path" />">By path</a>
            </li>
            <li>
                <a id="byPathPattern" class="textLink" href="<c:url
value="/mapping/path/wildcard" />">By path pattern</a>
            </li>
            <li>
```

```
                <a id="byMethod" class="textLink" href="<c:url
value="/mapping/method" />">By path and method</a>
            </li>
            <li>
                <a id="byParameter" class="textLink" href="<c:url
value="/mapping/parameter?foo=bar" />">By path, method, and presence of
parameter</a>
            </li>
            <li>
                <a id="byNotParameter" class="textLink" href="<c:url
value="/mapping/parameter" />">By path, method, and not presence of
parameter</a>
            </li>
            <li>
                <a id="byHeader" href="<c:url value="/mapping/header" />">By
presence of header</a>
            </li>
            <li>
                <a id="byHeaderNegation" class="textLink" href="<c:url
value="/mapping/header" />">By absence of header</a>
            </li>
            <li>
                <form id="byConsumes" class="readJsonForm" action="<c:url
value="/mapping/consumes" />" method="post">
                    <input id="byConsumesSubmit" type="submit" value="By
consumes" />
                </form>
            </li>
            <li>
                <a id="byProducesAcceptJson" class="writeJsonLink" href="<c:url
value="/mapping/produces" />">By produces via Accept=application/json</a>
            </li>
            <li>
                <a id="byProducesAcceptXml" class="writeXmlLink" href="<c:url
value="/mapping/produces" />">By produces via Accept=appilcation/xml</a>
            </li>
            <li>
                <a id="byProducesJsonExt" class="writeJsonLink" href="<c:url
value="/mapping/produces.json" />">By produces via ".json"</a>
            </li>
            <li>
                <a id="byProducesXmlExt" class="writeXmlLink" href="<c:url
value="/mapping/produces.xml" />">By produces via ".xml"</a>
            </li>
        </ul>
    </div>
</div>
```

扩展阅读:

两种常用的数据交换格式：XML和JSON

FastJSON、Gson和Jackson性能对比注意：数据会随着版本变更，导致测试结果也不大相同！

SpringMVC返回json数据的三种方式

3. 请求后台返回JSON数据实现步骤

3.1 导入Jackson支持库

```
<!-- jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.0</version>
</dependency>
```

注意：版本不可以太新，导致版本兼容问题。

3.2 接收客户端提交的JSON数据

转换为JavaBean

后台：MappingController.java

```
@RequestMapping(value="/mapping/consumes", method=RequestMethod.POST,
consumes=MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody String byConsumesJson(@RequestBody JavaBean javaBean) {
    return "Mapped by path + method + consumable media type (javaBean '" +
javaBean + "')";
}
```

使用@RequestBody注解自动转换JSON为JavaBean。

前端：index.html

```
<li>
    <form id="byConsumesJson" class="readJsonForm" action="<c:url
value="/mapping/consumes" />" method="post">
        <input id="byConsumesJsonSubmit" type="submit" value="By consumes" />
    </form>
</li>
```

脚本：index.js

```
$("#byHeader").click(function(){
    var link = $(this);
    $.ajax({
        url: this.href,
        dataType: "text",
        beforeSend: function(req) {
            req.setRequestHeader("FooHeader", "foo");
        },
        success: function(form) {
            MvcUtil.showSuccessResponse(form, link);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});

$("form.readJsonForm").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    var data = form.hasClass("invalid") ?
        "{ \"foo\": \"bar\" }" :
    "{ \"foo\": \"bar\", \"fruit\": \"apple\" }";
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: data, //发送的JSON格式数据
        contentType: "application/json",//请求数据类型为 json格式
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});
```

3.3 把JavaBean转换为JSON

数据响应给客户端

后台：MappingController.java

```
@RequestMapping(value="/mapping/produces", method=RequestMethod.GET,
produces=MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody JavaBean byProducesJson() {
    return new JavaBean();
}
```

使用@ResponseBody 注解自动转换JavaBean为JSON。

前端：index.html

```
<li>
    <a id="byProducesAcceptJson" class="writeJsonLink" href="<c:url
value="/mapping/produces" />">By produces via Accept=application/json</a>
</li>
<li>
    <a id="byProducesJsonExt" class="writeJsonLink" href="<c:url
value="/mapping/produces.json" />">By produces via ".json"</a>
</li>
```

url后带.json后缀相当于发送`req.setRequestHeader("Accept", "application/json");`。

默认可以省略。

脚本：index.js

```
$("a.writeJsonLink").click(function() {
    var link = $(this);
    $.ajax({ url: this.href,
            beforeSend: function(req) {
                if (!this.url.match(/\.json$/)) {
                    req.setRequestHeader("Accept", "application/json");
                }
            },
            success: function(json) {
                MvcUtil.showSuccessResponse(JSON.stringify(json), link);
            },
            error: function(xhr) {
                MvcUtil.showErrorResponse(xhr.responseText, link);
            }});
    return false;
});
```

Ajax请求可以设置请求头：

```
req.setRequestHeader("Accept", "application/json");
```

4.请求后台返回XML数据实现步骤

4.1 修改Entity对象

增加@XmlRootElement注解：

```
package com.springmvc.demo.web.ex02;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class JavaBean {
    //...
}
```

4.2 接收客户端提交的XML数据

转换为JavaBean：

Spring MVC接收XML及返回XML

4.3 把JavaBean转换为XML

后台：MappingController.java

```
@RequestMapping(value="/mapping/produces", method= RequestMethod.GET, produces=
MediaType.APPLICATION_XML_VALUE)
public @ResponseBody
    JavaBean byProducesXml() {
    return new JavaBean();
}
```

使用@ResponseBody 注解自动转换JavaBean为JSON。

前端：index.html

```
<li>
    <a id="byProducesAcceptXml" class="writeXmlLink" href="<c:url
value="/mapping/produces" />">By produces via Accept=appilcation/xml</a>
</li>

<li>
    <a id="byProducesXmlExt" class="writeXmlLink" href="<c:url
value="/mapping/produces.xml" />">By produces via ".xml"</a>
</li>
```

url后带.xml后缀相当于发送`req.setRequestHeader("Accept", "application/xml");`，默认可以省略。

脚本：index.js

```
$("a.writeXmlLink").click(function() {
    var link = $(this);
    $.ajax({
        url: link.attr("href"),
        beforeSend: function(req) {
            if (!this.url.match(/\.xml$/)) {
                req.setRequestHeader("Accept", "application/xml");
            }
        },
        success: function(xml) {
            MvcUtil.showSuccessResponse(MvcUtil.xmlencode(xml), link);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});
```

使用MvcUtil.xmlencode(xml)转换字符实体后显示 xml。

Ajax请求可以设置请求头：

```
req.setRequestHeader("Accept", "application/xml");
```

XML和JSON同时使用?

不同URI地址，不会产生冲突，传请求头可以省略，后缀结尾也可以省略。（.json / .xml）

相同URI地址，直接请求URI,因为请求头Accept包含`application/xml`，所以默认返回XML格式的数据。

如果希望返回JSON请在URI加上.json后缀即可。

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.9
```

5.简化请求映射路径

```java
@Controller
public class MappingController {

    @RequestMapping("/mapping/path")
    public @ResponseBody String byPath() {
        return "Mapped by path!";
    }
}
```

在Controller的上方添加@RequestMapping，作为全局映射路径，所有类中方法映射地址自动添加全局路径。实现简化映射路径目的。

```java
@Controller
@RequestMapping("/mapping")
public class MappingController {

    @RequestMapping("/path")
    public @ResponseBody String byPath() {
        return "Mapped by path!";
    }

    //...
}
```

真实案例写法：

```java
@Controller
@RequestMapping("/user")
public class UserController {
    //注入Service
    @Autowired
    private UserService userService;

    @RequestMapping("/save")
    public @ResponseBody String save(User user) {
        userService.save();
        return "success";
    }

    //...
}
```

四、Obtaining Request Data(获取请求数据)

1. Entity层实现代码

```java
package com.springmvc.demo.web.ex03;

public class JavaBean {

    private String param1;
    private String param2;
    private String param3;

    public String getParam1() {
        return param1;
    }

    public void setParam1(String param1) {
        this.param1 = param1;
    }

    public String getParam2() {
        return param2;
    }

    public void setParam2(String param2) {
        this.param2 = param2;
    }

    public String getParam3() {
        return param3;
    }

    public void setParam3(String param3) {
        this.param3 = param3;
    }

    @Override
    public String toString() {
        return "JavaBean{" +
                "param1='" + param1 + '\'' +
                ", param2='" + param2 + '\'' +
                ", param3='" + param3 + '\'' +
                '}';
    }
}
```

2. Web层实现代码

```java
package com.springmvc.demo.web.ex03;

import org.springframework.http.HttpEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/data")
public class RequestDataController {

    @RequestMapping(value="param", method= RequestMethod.GET)
    public @ResponseBody
    String withParam(@RequestParam String foo) {
        return "Obtained 'foo' query parameter value '" + foo + "'";
    }

    @RequestMapping(value="group", method= RequestMethod.GET)
    public @ResponseBody
    String withParamGroup(JavaBean bean) {
        return "Obtained parameter group " + bean;
    }

    @RequestMapping(value="path/{var}", method= RequestMethod.GET)
    public @ResponseBody
    String withPathVariable(@PathVariable String var) {
        return "Obtained 'var' path variable value '" + var + "'";
    }

    @RequestMapping(value="{path}/simple", method= RequestMethod.GET)
    public @ResponseBody
    String withMatrixVariable(@PathVariable String path, @MatrixVariable String
foo) {
        return "Obtained matrix variable 'foo=" + foo + "' from path segment '"
+ path + "'";
    }

    @RequestMapping(value="{path1}/{path2}", method= RequestMethod.GET)
    public @ResponseBody
    String withMatrixVariablesMultiple (
            @PathVariable String path1, @MatrixVariable(value="foo",
pathVar="path1") String foo1,
            @PathVariable String path2, @MatrixVariable(value="foo",
pathVar="path2") String foo2) {

        return "Obtained matrix variable foo=" + foo1 + " from path segment '" +
path1
                + "' and variable 'foo=" + foo2 + " from path segment '" + path2
+ "'";
    }

    @RequestMapping(value="header", method= RequestMethod.GET)
    public @ResponseBody
    String withHeader(@RequestHeader String Accept) {
        return "Obtained 'Accept' header '" + Accept + "'";
    }

    @RequestMapping(value="cookie", method= RequestMethod.GET)
    public @ResponseBody
    String withCookie(@CookieValue String openid_provider) {
        return "Obtained 'openid_provider' cookie '" + openid_provider + "'";
    }

    @RequestMapping(value="body", method= RequestMethod.POST)
    public @ResponseBody
    String withBody(@RequestBody String body) {
        return "Posted request body '" + body + "'";
    }

    @RequestMapping(value="entity", method= RequestMethod.POST)
    public @ResponseBody
    String withEntity(HttpEntity<String> entity) {
        return "Posted request body '" + entity.getBody() + "'; headers = " +
entity.getHeaders();
    }

}
```

注解说明:

注解及属性	用途
@RequestParam	**接收请求参数**绑定到处理器的**处理方法**的**方法参数上**; name：指定接收参数的名字。 required：是否必须提交属性。 defaultValue：设置不传值时的默认值
@PathVariable	请求URI中的模板变量部分到处理器功能处理方法的方法参数上的绑定，从而支持RESTful架构风格的URI;
@MatrixVariable	矩阵变量可以出现在任何路径片段中，每一个矩阵变量都用分号（;）隔开。比如 “`/cars;color=red;year=2012`”。多个值可以用逗号隔开，比如 “`color=red,green,blue`”，或者分开写 “`color=red;color=green;color=blue`”。如果你希望一个 URL 包含矩阵变量，那么请求映射模式必须用 URI 模板来表示这些矩阵变量。这样的话，不管矩阵变量顺序如何，都能够保证请求可以正确的匹配。
@RequestBody	请求的body体的绑定（通过HttpMessageConverter进行类型转换）；

3. View视图实现代码

修改index.jsp,增加body视图部分代码:

```
<!--......-->
<ul>
    <li><a href="#hello">Hello</a></li>
    <li><a href="#mapping">Request Mapping</a></li>

    <li><a href="#data">Request Data</a></li>

</ul>
<!--......-->
<div id="data">
    <h2>Request Data</h2>
    <p>
        接收请求参数 请参考
<code>com.springmvc.demo.web.ex03.RequestDataController</code>。
    </p>
    <ul>
        <li>
            <a id="param" class="textLink" href="<c:url value="/data/param?
foo=bar" />">Query parameter</a>
        </li>
        <li>
            <a id="group" class="textLink" href="<c:url value="/data/group?
param1=foo&param2=bar&param3=baz" />">Group of query parameters</a>
        </li>
        <li>
            <a id="var" class="textLink" href="<c:url value="/data/path/foo"
/>">Path variable</a>
        </li>
        <li>
            <a id="matrixVar" class="textLink" href="<c:url
value="/data/matrixvars;foo=bar/simple" />">Matrix variable</a>
        </li>
        <li>
            <a id="matrixVarMultiple" class="textLink" href="<c:url
value="/data/matrixvars;foo=bar1/multiple;foo=bar2" />">Matrix variables
(multiple)</a>
        </li>
        <li>
            <a id="header" class="textLink" href="<c:url value="/data/header"
/>">Header</a>
        </li>
        <li>
            <form id="requestBody" class="textForm" action="<c:url
value="/data/body" />" method="post">
                <input id="requestBodySubmit" type="submit" value="Request Body"
/>
            </form>
        </li>
        <li>
            <form id="requestBodyAndHeaders" class="textForm" action="<c:url
value="/data/entity" />" method="post">
                <input id="requestBodyAndHeadersSubmit" type="submit"
value="Request Body and Headers" />
            </form>
        </li>
    </ul>
</div>
```

4.修改核心脚本index.js

```
$("form.textForm").submit(function(event) {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: "foo",
        contentType: "text/plain",
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});
```

5.详讲接收请求参数

5.1 参数列表接收请求参数

1)同名的参数（请求参数与处理方法的参数名一致）。自动赋值。

2）使用@RequestParam指定处理方法的参数 从 请求参数中获取。

```
/data/param?foo=bar

@RequestMapping(value="param", method= RequestMethod.GET)
public @ResponseBody String withParam(@RequestParam String foo) {
    return "Obtained 'foo' query parameter value '" + foo + "'";
}
```

@RequestParam(name="foo",required=**true**,defaultValue="guest") String foo

Name：指定接收参数的名字

Required：是否必须提交属性

defaultValue：设置不传值时的默认值

5.2 请求参数封装到JavaBean

1. 提交属性字段与JavaBean字段一致
2. JavaBean必须为接收字段提供对应Setter函数。

1）GET/POST直接把数据字段提交,后台自动把数据填充到拥有一样字段的JavaBean

```
/data/group?param1=foo&param2=bar&param3=baz

public class JavaBean {
    private String param1;
    private String param2;
    private String param3;

    //setter
}

@RequestMapping(value="group", method= RequestMethod.GET)
public @ResponseBody
    String withParamGroup(JavaBean bean) {
```

```
    return "Obtained parameter group " + bean;
}
```

2）提交JSON/XML数据，JavaBean参数必须加上@RequestBody注解实现自动转换。

```
@RequestMapping(value="/mapping/consumes", method= RequestMethod.POST, consumes=
MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody
    String byConsumesJson(@RequestBody JavaBean javaBean) {
    return "Mapped by path + method + consumable media type (javaBean '" +
javaBean + "')";
}
```

5.3 从请求Url中获取数据

```
@RequestMapping(value="path/{var}")
public @ResponseBody String withPathVariable(@PathVariable String var) {}
```

5.4 从url解析矩阵变量

修改：src\main\webapp\WEB-INF\springmvc-servlet.xml

1）启用矩阵变量配置

```
<mvc:annotation-driven enable-matrix-variables="true"/>
```

2）在Spring3.2 后，一个@MatrixVariable出现了，这个注解的出现拓展了URL请求地址的功能。

矩阵变量1：多个变量可以使用“;”（分号）分隔，例如：

```
/cars;color=red;year=2012
```

矩阵变量2：如果是一个变量的多个值那么可以使用“,”（逗号）分隔

```
color=red,green,blue
```

矩阵变量3：可以使用重复的变量名：

```
color=red;color=green;color=blue
```

3) 请求url带matrixvars;关键字,传多个矩阵变量使用multiple关键字。

```
//  /data/matrixvars;foo=bar/simple   {path}  =  matrixvars;foo=bar
@RequestMapping(value="{path}/simple", method= RequestMethod.GET)
public @ResponseBody String withMatrixVariable(@PathVariable String path,
@MatrixVariable String foo) {
    return "Obtained matrix variable 'foo=" + foo + "' from path segment '" +
path + "'";
}
```

```
//  /data/matrixvars;foo=bar1/multiple;foo=bar2"  path1 = matrixvars;foo=bar1
path2 = multiple;foo=bar2
@RequestMapping(value="{path1}/{path2}", method= RequestMethod.GET)
public @ResponseBody String withMatrixVariablesMultiple (
    @PathVariable String path1, @MatrixVariable(value="foo", pathVar="path1")
String foo1,
    @PathVariable String path2, @MatrixVariable(value="foo", pathVar="path2")
String foo2) {

    return "Obtained matrix variable foo=" + foo1 + " from path segment '" +
path1
        + "' and variable 'foo=" + foo2 + " from path segment '" + path2 + "'";
}
```

5.5 从servlet不同作用域获取参数

1）获取请求Accept的值

使用@RequestHeader注解获取请求头的值。对应参数名必须为请求头的Name。

```
@RequestMapping(value="header", method= RequestMethod.GET)
public @ResponseBody String withHeader(@RequestHeader String Accept) {
    return "Obtained 'Accept' header '" + Accept + "'";
}
```

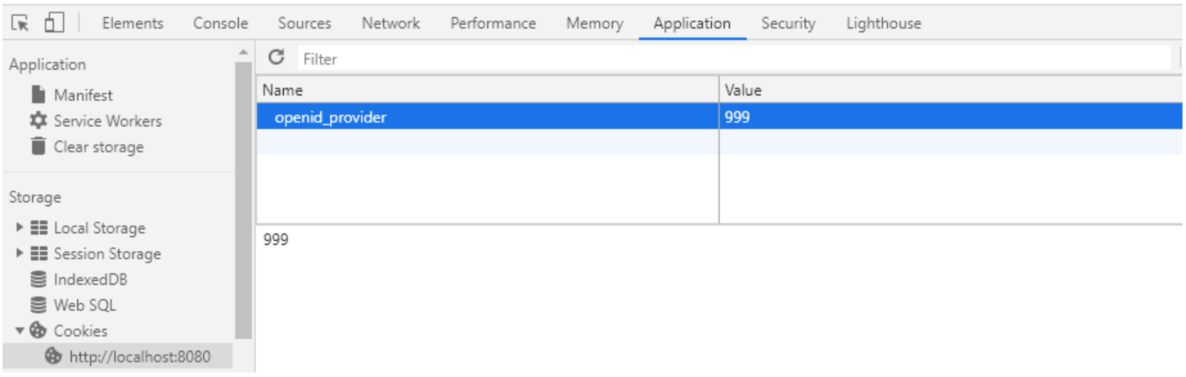
```
Accept: text/plain, */*; q=0.01
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: keep-alive
Host: localhost:8080
Referer: http://localhost:8080/spring-mvc-demo/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/83.0.4103.116 Safari/537.36
X-Requested-With: XMLHttpRequest
```

2）获取Cookie 的值

使用@CookieValue注解获取参数名对应的Cookie的值。

```
@RequestMapping(value="cookie", method= RequestMethod.GET)
public @ResponseBody
String withCookie(@CookieValue String openid_provider) {
    return "Obtained 'openid_provider' cookie '" + openid_provider + "'";
}
```

可以手工添加一个名为openid_provider的Cookie：



Name	Value
openid_provider	999

999

3）获取请求body的内容

@RequestBody 获取请求body的内容（获取POST方式提交的数据）。

```
@RequestMapping(value="body", method= RequestMethod.POST)
public @ResponseBody
String withBody(@RequestBody String body) {
    return "Posted request body '" + body + "'";
}
```

4）获取请求body和所有的头信息

使用HttpEntity对象获取请求的body和头信息集合。

```
@RequestMapping(value="entity", method= RequestMethod.POST)
public @ResponseBody String withEntity(HttpEntity<String> entity) {
return "Posted request body '" + entity.getBody() + "'; headers = " +
entity.getHeaders();
}
```

6.Servlet标准参数

6.1 Spring MVC使用 Servlet API

spring mvc与servlet松耦合，对Servlet是**可插拔**的。没有具体耦合。

1.需要使用时才声明在处理器的处理方法的参数列表中。

2.Spring MVC 自动注入 Servlet API的实例（标准Servlet api的参数自动初始化）。

```
public @ResponseBody String byPathPattern(HttpServletRequest request) {}
public @ResponseBody String response(HttpServletResponse response) {}
public @ResponseBody String session(HttpSession session) {}

public @ResponseBody String write(Writer responseWriter) {}
public @ResponseBody String read(Reader requestBodyReader) {}
public @ResponseBody String output(OutputStream os) {}
public @ResponseBody String input(InputStream is) {}
```

6.2 Web层实现代码

```
package com.springmvc.demo.web.ex03;

import org.springframework.stereotype.Controller;
import org.springframework.util.FileCopyUtils;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.*;
import java.security.Principal;
import java.util.Locale;

@Controller
public class StandardArgumentsController {

   // request related

   @RequestMapping(value="/data/standard/request", method= RequestMethod.GET)
   public @ResponseBody
    String standardRequestArgs(HttpServletRequest request, Principal user,
Locale locale) {
      StringBuilder buffer = new StringBuilder();
      buffer.append("request = ").append(request).append(", ");
      buffer.append("userPrincipal = ").append(user).append(", ");
      buffer.append("requestLocale = ").append(locale);
      return buffer.toString();
   }

   @RequestMapping(value="/data/standard/request/reader", method=
RequestMethod.POST)
   public @ResponseBody
    String requestReader(Reader requestBodyReader) throws IOException {
      return "Read char request body = " +
FileCopyUtils.copyToString(requestBodyReader);
   }

   @RequestMapping(value="/data/standard/request/is", method=
RequestMethod.POST)
   public @ResponseBody
    String requestReader(InputStream requestBodyIs) throws IOException {
      return "Read binary request body = " + new
String(FileCopyUtils.copyToByteArray(requestBodyIs));
   }

   // response related

   @RequestMapping("/data/standard/response")
   public @ResponseBody
    String response(HttpServletResponse response) {
      return "response = " + response;
   }

   @RequestMapping("/data/standard/response/writer")
   public void availableStandardResponseArguments(Writer responseWriter) throws
IOException {
        responseWriter.write("Wrote char response using Writer");
    }

    @RequestMapping("/data/standard/response/os")
    public void availableStandardResponseArguments(OutputStream os) throws
IOException {
        os.write("Wrote binary response using OutputStream".getBytes());
    }

    // HttpSession

    @RequestMapping("/data/standard/session")
    public @ResponseBody
     String session(HttpSession session) {
        StringBuilder buffer = new StringBuilder();
        buffer.append("session=").append(session);
        return buffer.toString();
    }

}
```

6.3 View视图实现代码

前端：index.jsp增加代码，增加到`<div id="data"></div>`内：

```
<div id="standardArgs">
    <h3>Standard Resolvable Web Arguments</h3>
    <ul>
        <li>
            <a id="request" class="textLink" href="<c:url
value="/data/standard/request" />">Request arguments</a>
        </li>
        <li>
            <form id="requestReader" class="textForm" action="<c:url
value="/data/standard/request/reader" />" method="post">
                <input id="requestReaderSubmit" type="submit" value="Request
Reader" />
            </form>
        </li>
        <li>
            <form id="requestIs" class="textForm" action="<c:url
value="/data/standard/request/is" />" method="post">
                <input id="requestIsSubmit" type="submit" value="Request
InputStream" />
            </form>
        </li>
        <li>
            <a id="response" class="textLink" href="<c:url
value="/data/standard/response" />">Response arguments</a>
        </li>
        <li>
            <a id="writer" class="textLink" href="<c:url
value="/data/standard/response/writer" />">Response Writer</a>
        </li>
        <li>
            <a id="os" class="textLink" href="<c:url
value="/data/standard/response/os" />">Response OutputStream</a>
```

```
        </li>
        <li>
            <a id="session" class="textLink" href="<c:url
value="/data/standard/session" />">Session</a>
        </li>
    </ul>
</div>
```

7.自定义请求参数注解

自定义@RequestAttribute注解实现从request作用域中获取参数的值，赋值给处理方法的变量。

```
 request.setAttribute("foo", "bar");

@RequestMapping(value="/data/custom", method= RequestMethod.GET)
public @ResponseBody String custom(@RequestAttribute("foo") String foo) {
    return "Got 'foo' request attribute value '" + foo + "'";
}
```

7.1 创建请求参数注解

```
package com.springmvc.demo.web.ex03;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface RequestAttribute {
    String value();
}
```

@Target、@Retention、@Documented注解简介

7.2 自定义参数解析程序

```
package com.springmvc.demo.web.ex03;

import org.springframework.core.MethodParameter;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.method.support.ModelAndViewContainer;
```

```java
public class CustomArgumentResolver implements HandlerMethodArgumentResolver {

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        return parameter.getParameterAnnotation(RequestAttribute.class) != null;
    }

    @Override
    public Object resolveArgument(MethodParameter parameter,
                                  ModelAndViewContainer mavContainer,
                                  NativeWebRequest webRequest,
                                  WebDataBinderFactory binderFactory) throws
Exception
    {
        RequestAttribute attr =
parameter.getParameterAnnotation(RequestAttribute.class);
        return webRequest.getAttribute(attr.value(), WebRequest.SCOPE_REQUEST);
    }

}
```

1.实现**处理程序方法参数解析程序** HandlerMethodArgumentResolver接口

2.重写 支持参数supportsParameter 和 解析参数resolveArgument 两个方法。

7.3 配置参数解析程序

修改：src\main\webapp\WEB-INF\springmvc-servlet.xml

```xml
<mvc:annotation-driven enable-matrix-variables="true">
    <mvc:argument-resolvers>
        <bean class="com.springmvc.demo.web.ex03.CustomArgumentResolver"/>
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

7.4 使用请求参数注解

```java
package com.springmvc.demo.web.ex03;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;

@Controller
public class CustomArgumentController {

    @ModelAttribute
    void beforeInvokingHandlerMethod(HttpServletRequest request) {
        request.setAttribute("foo", "bar");
    }
    @RequestMapping(value="/data/custom", method= RequestMethod.GET)
    public @ResponseBody
    String custom(@RequestAttribute("foo") String foo) {
        return "Got 'foo' request attribute value '" + foo + "'";
    }
}
```

注解	用途
`@ModelAttribute`	注解注释的方法会在此controller每个方法执行前被执行，因此对于一个controller映射多个URL的用法来说，要谨慎使用。

7.5 View视图实现代码

前端：index.jsp增加代码，增加到`<div id="data"></div>`内：

```
<div id="customArgs">
    <h3>Custom Resolvable Web Arguments</h3>
    <ul>
        <li>
            <a id="customArg" class="textLink" href="<c:url value="/data/custom"
/>">Custom</a>
        </li>
    </ul>
</div>
```

五、Generating Responses (生成响应)

主要讲解@ResponseBody的使用。

注解	描述
@ResponseBody	处理器功能处理方法的返回值作为响应体（通过HttpMessageConverter进行类型转换）； 返回对象自动转换JSON/XML。 响应文本内容和跳转 设置响应字符编码 设置响应状态编码

1. Web层实现代码

```
package com.springmvc.demo.web.ex04;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping(value="/response", method= RequestMethod.GET)
public class ResponseController {

    @RequestMapping("/annotation")
    public @ResponseBody
    String responseBody() {
        return "The String ResponseBody";
    }

    @RequestMapping("/charset/accept")
    public @ResponseBody
    String responseAcceptHeaderCharset() {
        return "\u4f60\u597d\u4e16\u754c\u0021 (\"Hello world!\" in Chinese)";
    }

    @RequestMapping(value="/charset/produce", produces="text/plain;charset=UTF-
8")
    public @ResponseBody
    String responseProducesConditionCharset() {
        return "\u4f60\u597d\u4e16\u754c\u0021 (\"Hello world!\" in Chinese)";
    }

    @RequestMapping("/entity/status")
    public ResponseEntity<String> responseEntityStatusCode() {
        return new ResponseEntity<String>("The String ResponseBody with custom
status code (403 Forbidden)",
                HttpStatus.FORBIDDEN);
    }

    @RequestMapping("/entity/headers")
    public ResponseEntity<String> responseEntityCustomHeaders() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.TEXT_PLAIN);
        return new ResponseEntity<String>("The String ResponseBody with custom
header Content-Type=text/plain",
                headers, HttpStatus.OK);
    }

}
```

2. View视图实现代码

修改index.jsp,增加body视图部分代码：

```
<!--......-->
<ul>
    <li><a href="#hello">Hello</a></li>
    <li><a href="#mapping">Request Mapping</a></li>
    <li><a href="#data">Request Data</a></li>
    <li><a href="#responses">Generating Responses</a></li>

</ul>
<!--......-->
<div id="responses">
    <h2>Response Writing</h2>
    <p>
        生成响应主体及状态码、响应头 请参考
<code>com.springmvc.demo.web.ex04.ResponseController</code>。
    </p>
    <ul>
        <li>
            <a id="responseBody" class="textLink" href="<c:url
value="/response/annotation" />">@ResponseBody</a>
        </li>
        <li>
            <a id="responseCharsetAccept" class="utf8TextLink" href="<c:url
value="/response/charset/accept" />">@ResponseBody (UTF-8 charset requested)</a>
        </li>
        <li>
            <a id="responseCharsetProduce" class="textLink" href="<c:url
value="/response/charset/produce" />">@ResponseBody (UTF-8 charset produced)</a>
        </li>
        <li>
            <a id="responseEntityStatus" class="textLink" href="<c:url
value="/response/entity/status" />">ResponseEntity (custom status)</a>
        </li>
        <li>
            <a id="responseEntityHeaders" class="textLink" href="<c:url
value="/response/entity/headers" />">ResponseEntity (custom headers)</a>
        </li>
    </ul>
</div>
```

3.修改核心脚本index.js

```
$("a.utf8TextLink").click(function(){
    var link = $(this);
    $.ajax({
        url: link.attr("href"),
        dataType: "text",
        beforeSend: function(req) {
            req.setRequestHeader("Accept", "text/plain;charset=UTF-8");
        },
        success: function(text) {
            MvcUtil.showSuccessResponse(text, link);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});
```

4. 响应文本内容和跳转

@Controller中的方法，如果返回类型为String。

- 带@ResponseBody，则返回文本(字符串/JSON/XML等)内容。

```
@RequestMapping("/login")
public @ResponseBody String login() {
    return "login";
}
```

响应："login"字符串。

- 不带@ResponseBody，则进行页面跳转。

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

例如：访问/login地址则自动重定向到**/WEB-INF/views/login.jsp**/

```
@RequestMapping("/login")
public String login() {
    return "login";
}
```

5. 设置响应字符编码

扩展阅读-Unicode转换工具

- 前端设置请求响应字符编码：

```
beforeSend: function(req) {
    req.setRequestHeader("Accept", "text/plain;charset=UTF-8");
},


@RequestMapping("/charset/accept")
public @ResponseBody
    String responseAcceptHeaderCharset() {
    return "\u4f60\u597d\u4e16\u754c\u0021 (\"Hello world!\" in Chinese)";
}
```

- 后台设置请求响应字符编码：

```
@RequestMapping(value="/charset/produce", produces="text/plain;charset=UTF-
8")
public @ResponseBody
    String responseProducesConditionCharset() {
    return "\u4f60\u597d\u4e16\u754c\u0021 (\"Hello world!\" in Chinese)";
}
```

等效于：

```
@RequestMapping(value="/charset/produce")
public @ResponseBody String setCharset(HttpServletRequest request) {
    request.setRequestHeader("Accept", "text/plain;charset=UTF-8");
    return "\u4f60\u597d\u4e16\u754c\u0021 (\"Hello world!\" in Chinese)";
}
```

- 全局设置-响应字符编码：

  修改Spring mvc配置文件springmvc-servlet.xml,在mvc:annotation-driven配置中添加**StringHttpMessageConverter。**

```
<mvc:annotation-driven enable-matrix-variables="true">
    <mvc:message-converters register-defaults="true">
        <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
            <constructor-arg value="UTF-8"/>
        </bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```

6. 设置响应状态编码

@ResponseBody：加在请求处理方法上，能够处理方法结果值作为http响应体。
@ResponseStatus：加在方法上、返回自定义http状态码。

ResponseEntity类可以定义返回 **响应体内容**、**头部信息**（请求头和响应头） 和 **状态码** 。

扩展阅读-HttpStatus响应状态编码

扩展阅读-HttpHeaders头部信息

1.使用ResponseEntity返回 响应体内容 和 状态码：

```
@RequestMapping("/entity/status")
public ResponseEntity<String> responseEntityStatusCode() {
   return new ResponseEntity<String>(
         "The String ResponseBody with custom status code (403 Forbidden)",
         HttpStatus.FORBIDDEN);
}
```

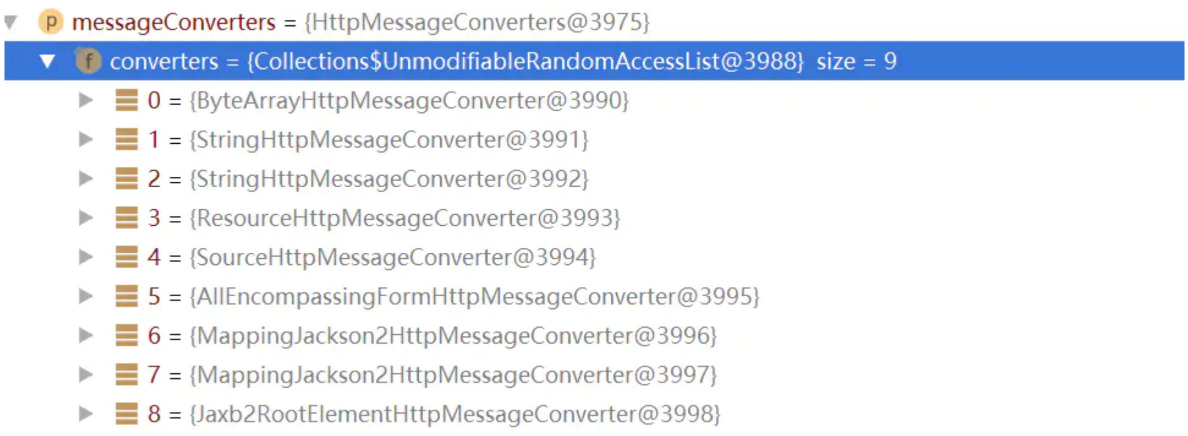
2.使用ResponseEntity返回 响应体内容 、头信息 和 状态码：

```java
@RequestMapping("/entity/headers")
public ResponseEntity<String> responseEntityCustomHeaders() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.TEXT_PLAIN);

    return new ResponseEntity<String>(
        "The String ResponseBody with custom header Content-Type=text/plain",
        headers,
        HttpStatus.OK);
}
```

六、 Message Converters(消息转换器)

spring MVC为我们提供了一系列默认的消息转换器。

接收和响应**String、Form表单数据、json**、xml、atom、rss消息数据转换。


```
▼ p messageConverters = {HttpMessageConverters@3975}
  ▼ f converters = {Collections$UnmodifiableRandomAccessList@3988}  size = 9
    ▶ 0 = {ByteArrayHttpMessageConverter@3990}
    ▶ 1 = {StringHttpMessageConverter@3991}
    ▶ 2 = {StringHttpMessageConverter@3992}
    ▶ 3 = {ResourceHttpMessageConverter@3993}
    ▶ 4 = {SourceHttpMessageConverter@3994}
    ▶ 5 = {AllEncompassingFormHttpMessageConverter@3995}
    ▶ 6 = {MappingJackson2HttpMessageConverter@3996}
    ▶ 7 = {MappingJackson2HttpMessageConverter@3997}
    ▶ 8 = {Jaxb2RootElementHttpMessageConverter@3998}
```

对于消息转换器的调用，都是在RequestResponseBodyMethodProcessor类中完成的。它实现了HandlerMethodArgumentResolver和HandlerMethodReturnValueHandler两个接口，分别实现了处理参数和处理返回值的方法。

而要动用这些消息转换器，需要在特定的位置加上

@RequestBody 接收(声明于方法参数列表) 和@ResponseBody 响应(声明于方法返回类型 或者 方法的上方) 。

AJAX ：配合 contentType: "application/xml"， 和 请求头   req.setRequestHeader("Accept", "application/xxx");

后台：consumes= MediaType.APPLICATION_JSON_VALUE 和 produces= MediaType.APPLICATION_JSON_VALUE

扩展阅读-消息转换器

1. Entity层实现代码

```java
package com.springmvc.demo.web.ex05;
```

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class JavaBean {


    private String foo;
    private String fruit;

    public JavaBean() {
    }

    public JavaBean(String foo, String fruit) {
        this.foo = foo;
        this.fruit = fruit;
    }

    public String getFoo() {
        return foo;
    }

    public void setFoo(String foo) {
        this.foo = foo;
    }

    public String getFruit() {
        return fruit;
    }

    public void setFruit(String fruit) {
        this.fruit = fruit;
    }

    @Override
    public String toString() {
        return "JavaBean {foo=[" + foo + "], fruit=[" + fruit + "]}";
    }

}
```

2. Web层实现代码

```
package com.springmvc.demo.web.ex05;

import com.rometools.rome.feed.atom.Feed;
import com.rometools.rome.feed.rss.Channel;
import com.springmvc.demo.web.ex05.JavaBean;
import org.springframework.stereotype.Controller;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/messageconverters")
public class MessageConvertersController {
    // StringHttpMessageConverter
    @RequestMapping(value="/string", method= RequestMethod.POST)
    public @ResponseBody
    String readString(@RequestBody String string) {
        return "Read string '" + string + "'";
    }

    @RequestMapping(value="/string", method= RequestMethod.GET)
    public @ResponseBody
    String writeString() {
        return "Wrote a string";
    }

    // Form encoded data (application/x-www-form-urlencoded)

    @RequestMapping(value="/form", method= RequestMethod.POST)
    public @ResponseBody
    String readForm(@ModelAttribute JavaBean bean) {
        return "Read x-www-form-urlencoded: " + bean;
    }

    @RequestMapping(value="/form", method= RequestMethod.GET)
    public @ResponseBody
    MultiValueMap<String, String> writeForm() {
        MultiValueMap<String, String> map = new LinkedMultiValueMap<String,
String>();
        map.add("foo", "bar");
        map.add("fruit", "apple");
        return map;
    }

    // Jaxb2RootElementHttpMessageConverter (requires JAXB2 on the classpath -
useful for serving clients that expect to work with XML)

    @RequestMapping(value="/xml", method= RequestMethod.POST)
    public @ResponseBody
    String readXml(@RequestBody JavaBean bean) {
        return "Read from XML: " + bean;
    }

    @RequestMapping(value="/xml", method= RequestMethod.GET)
    public @ResponseBody
    JavaBean writeXml() {
        return new JavaBean("bar", "apple");
    }

    // MappingJacksonHttpMessageConverter (requires Jackson on the classpath -
particularly useful for serving JavaScript clients that expect to work with
JSON)

    @RequestMapping(value="/json", method= RequestMethod.POST)
    public @ResponseBody
    String readJson(@RequestBody JavaBean bean) {
        return "Read from JSON: " + bean;
    }

    @RequestMapping(value="/json", method= RequestMethod.GET)
    public @ResponseBody
    JavaBean writeJson() {
        return new JavaBean("bar", "apple");
    }

    // AtomFeedHttpMessageConverter (requires Rome on the classpath - useful for
serving Atom feeds)

    @RequestMapping(value="/atom", method= RequestMethod.POST)
    public @ResponseBody
    String readFeed(@RequestBody Feed feed) {
        return "Read " + feed.getTitle();
    }

    @RequestMapping(value="/atom", method= RequestMethod.GET)
    public @ResponseBody
    Feed writeFeed() {
        Feed feed = new Feed();
        feed.setFeedType("atom_1.0");
        feed.setTitle("My Atom feed");
        return feed;
    }

    // RssChannelHttpMessageConverter (requires Rome on the classpath - useful
for serving RSS feeds)

    @RequestMapping(value="/rss", method= RequestMethod.POST)
    public @ResponseBody
    String readChannel(@RequestBody Channel channel) {
        return "Read " + channel.getTitle();
    }

    @RequestMapping(value="/rss", method= RequestMethod.GET)
    public @ResponseBody
    Channel writeChannel() {
        Channel channel = new Channel();
        channel.setFeedType("rss_2.0");
        channel.setTitle("My RSS feed");
        channel.setDescription("Description");
        channel.setLink("http://localhost:8080/mvc-showcase/rss");
        return channel;
    }

}
```

3. View视图实现代码

修改index.jsp,增加body视图部分代码：

```
<!--......-->
<ul>
    <li><a href="#hello">Hello</a></li>
    <li><a href="#mapping">Request Mapping</a></li>
    <li><a href="#data">Request Data</a></li>
    <li><a href="#responses">Generating Responses</a></li>

    <li><a href="#messageconverters">Http Message Converters</a></li>

</ul>
<!--......-->
<div id="messageconverters">
    <h2>Http Message Converters</h2>
    <p>
        消息转换器 请参考
<code>com.springmvc.demo.web.ex05.MessageConvertersController</code>。
    </p>
    <div id="stringMessageConverter">
        <h3>StringHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readString" class="textForm" action="<c:url
value="/messageconverters/string" />" method="post">
                    <input id="readStringSubmit" type="submit" value="Read a
String" />
                </form>
            </li>
            <li>
                <a id="writeString" class="textLink" href="<c:url
value="/messageconverters/string" />">Write a String</a>
            </li>
        </ul>
        <h3>FormHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readForm" action="<c:url
value="/messageconverters/form" />" method="post">
                    <input id="readFormSubmit" type="submit" value="Read Form
Data" />
                </form>
            </li>
            <li>
                <a id="writeForm" href="<c:url value="/messageconverters/form"
/>">Write Form Data</a>
            </li>
        </ul>
        <h3>Jaxb2RootElementHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readXml" class="readXmlForm" action="<c:url
value="/messageconverters/xml" />" method="post">
                    <input id="readXmlSubmit" type="submit" value="Read XML" />
                </form>
            </li>
            <li>
                <a id="writeXmlAccept" class="writeXmlLink" href="<c:url
value="/messageconverters/xml" />">Write XML via Accept=application/xml</a>
            </li>
            <li>
                <a id="writeXmlExt" class="writeXmlLink" href="<c:url
value="/messageconverters/xml.xml" />">Write XML via ".xml"</a>
            </li>
        </ul>
        <h3>MappingJacksonHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readJson" class="readJsonForm" action="<c:url
value="/messageconverters/json" />" method="post">
                    <input id="readJsonSubmit" type="submit" value="Read JSON"
/>
                </form>
            </li>
            <li>
                <a id="writeJsonAccept" class="writeJsonLink" href="<c:url
value="/messageconverters/json" />">Write JSON via Accept=application/json</a>
            </li>
            <li>
                <a id="writeJsonExt" class="writeJsonLink" href="<c:url
value="/messageconverters/json.json" />">Write JSON via ".json"</a>
            </li>
        </ul>
        <h3>AtomFeedHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readAtom" action="<c:url
value="/messageconverters/atom" />" method="post">
                    <input id="readAtomSubmit" type="submit" value="Read Atom"
/>
                </form>
            </li>
            <li>
                <a id="writeAtom" href="<c:url value="/messageconverters/atom"
/>">Write Atom</a>
            </li>
        </ul>
        <h3>RssChannelHttpMessageConverter</h3>
        <ul>
            <li>
                <form id="readRss" action="<c:url value="/messageconverters/rss"
/>" method="post">
                    <input id="readRssSubmit" type="submit" value="Read Rss" />
                </form>
            </li>
            <li>
                <a id="writeRss" href="<c:url value="/messageconverters/rss"
/>">Write Rss</a>
            </li>
        </ul>
    </div>
</div>
```

4.修改核心脚本index.js

```
$("#readForm").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: "foo=bar&fruit=apple",
        contentType: "application/x-www-form-urlencoded",
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeForm").click(function() {
    var link = $(this);
    $.ajax({
        url: this.href,
        dataType: "text",
        beforeSend: function(req) {
            req.setRequestHeader("Accept", "application/x-www-form-urlencoded");
        },
        success: function(form) {
            MvcUtil.showSuccessResponse(form, link);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});

$("form.readXmlForm").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>
<javaBean><foo>bar</foo><fruit>apple</fruit></javaBean>",
        contentType: "application/xml",
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#readAtom").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: '<?xml version="1.0" encoding="UTF-8"?> <feed
xmlns="http://www.w3.org/2005/Atom"><title>My Atom feed</title></feed>',
        contentType: "application/atom+xml",
        dataType: "text", success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeAtom").click(function() {
    var link = $(this);
    $.ajax({ url: link.attr("href"),
            beforeSend: function(req) {
                req.setRequestHeader("Accept", "application/atom+xml");
            },
            success: function(feed) {
                MvcUtil.showSuccessResponse(MvcUtil.xmlencode(feed), link);
            },
            error: function(xhr) {
                MvcUtil.showErrorResponse(xhr.responseText, link);
            }
           });
    return false;
});

$("#readRss").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: '<?xml version="1.0" encoding="UTF-8"?> <rss version="2.0">
<channel><title>My RSS feed</title></channel></rss>',
        contentType: "application/rss+xml",
        dataType: "text", success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeRss").click(function() {
    var link = $(this);
    $.ajax({ url: link.attr("href"),
        beforeSend: function(req) {
req.setRequestHeader("Accept", "application/rss+xml");
        },
        success: function(feed) {
                MvcUtil.showSuccessResponse(MvcUtil.xmlencode(feed), link);
        },
        error: function(xhr) {
```

```
MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});
```

5.StringHttpMessageConverter

```
@RequestMapping(value="/string", method= RequestMethod.POST)
public @ResponseBody
    String readString(@RequestBody String string) {
    return "Read string '" + string + "'";
}

@RequestMapping(value="/string", method= RequestMethod.GET)
public @ResponseBody
    String writeString() {
    return "Wrote a string";
}
```

6.FormHttpMessageConverter

Form encoded data (application/x-www-form-urlencoded)

```
@RequestMapping(value="/form", method= RequestMethod.POST)
public @ResponseBody
    String readForm(@ModelAttribute JavaBean bean) {
    return "Read x-www-form-urlencoded: " + bean;
}

@RequestMapping(value="/form", method= RequestMethod.GET)
public @ResponseBody
    MultiValueMap<String, String> writeForm() {
    MultiValueMap<String, String> map = new LinkedMultiValueMap<String, String>
();
    map.add("foo", "bar");
    map.add("fruit", "apple");
    return map;
}
```

修改核心脚本index.js

```
$("#readForm").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: "foo=bar&fruit=apple",
        contentType: "application/x-www-form-urlencoded",
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeForm").click(function() {
    var link = $(this);
    $.ajax({
        url: this.href,
        dataType: "text",
        beforeSend: function(req) {
            req.setRequestHeader("Accept", "application/x-www-form-urlencoded");
        },
        success: function(form) {
            MvcUtil.showSuccessResponse(form, link);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});
```

7.Jaxb2RootElementHttpMessageConverter

```
@RequestMapping(value="/xml", method= RequestMethod.POST)
public @ResponseBody
    String readXml(@RequestBody JavaBean bean) {
    return "Read from XML: " + bean;
}

@RequestMapping(value="/xml", method= RequestMethod.GET)
public @ResponseBody
    JavaBean writeXml() {
    return new JavaBean("bar", "apple");
}
```

修改核心脚本index.js

```
$("form.readXmlForm").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>
<javaBean><foo>bar</foo><fruit>apple</fruit></javaBean>",
        contentType: "application/xml",
        dataType: "text",
        success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});
```

8.MappingJacksonHttpMessageConverter

```
@RequestMapping(value="/json", method= RequestMethod.POST)
public @ResponseBody
    String readJson(@RequestBody JavaBean bean) {
    return "Read from JSON: " + bean;
}

@RequestMapping(value="/json", method= RequestMethod.GET)
public @ResponseBody
    JavaBean writeJson() {
    return new JavaBean("bar", "apple");
}
```

需要实现Atom+RSS需要引人：

```
<!-- Rome Atom + RSS -->
<dependency>
    <groupId>com.rometools</groupId>
    <artifactId>rome</artifactId>
    <version>1.5.0</version>
</dependency>

<!-- ROME is a set of Atom/RSS Java utilities -->
<dependency>
    <groupId>com.rometools</groupId>
    <artifactId>rome</artifactId>
    <version>1.14.1</version>
</dependency>
```

9.AtomFeedHttpMessageConverter

```
// AtomFeedHttpMessageConverter (requires Rome on the classpath - useful for
serving Atom feeds)

@RequestMapping(value="/atom", method= RequestMethod.POST)
public @ResponseBody
   String readFeed(@RequestBody Feed feed) {
   return "Read " + feed.getTitle();
}
```

```
@RequestMapping(value="/atom", method= RequestMethod.GET)
public @ResponseBody
   Feed writeFeed() {
   Feed feed = new Feed();
   feed.setFeedType("atom_1.0");
   feed.setTitle("My Atom feed");
   return feed;
}
```

修改核心脚本index.js

```
$("#readAtom").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: '<?xml version="1.0" encoding="UTF-8"?> <feed
xmlns="http://www.w3.org/2005/Atom"><title>My Atom feed</title></feed>',
        contentType: "application/atom+xml",
        dataType: "text", success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeAtom").click(function() {
    var link = $(this);
    $.ajax({ url: link.attr("href"),
            beforeSend: function(req) {
                req.setRequestHeader("Accept", "application/atom+xml");
            },
            success: function(feed) {
                MvcUtil.showSuccessResponse(MvcUtil.xmlencode(feed), link);
            },
            error: function(xhr) {
                MvcUtil.showErrorResponse(xhr.responseText, link);
            }
           });
    return false;
});
```

10.RssChannelHttpMessageConverter

```
// RssChannelHttpMessageConverter (requires Rome on the classpath - useful for
serving RSS feeds)

@RequestMapping(value="/rss", method= RequestMethod.POST)
public @ResponseBody
```

```
    String readChannel(@RequestBody Channel channel) {
    return "Read " + channel.getTitle();
}

@RequestMapping(value="/rss", method= RequestMethod.GET)
public @ResponseBody
    Channel writeChannel() {
    Channel channel = new Channel();
    channel.setFeedType("rss_2.0");
    channel.setTitle("My RSS feed");
    channel.setDescription("Description");
    channel.setLink("http://localhost:8080/mvc-showcase/rss");
    return channel;
}
```

修改核心脚本index.js

```
$("#readRss").submit(function() {
    var form = $(this);
    var button = form.children(":first");
    $.ajax({
        type: "POST",
        url: form.attr("action"),
        data: '<?xml version="1.0" encoding="UTF-8"?> <rss version="2.0">
<channel><title>My RSS feed</title></channel></rss>',
        contentType: "application/rss+xml",
        dataType: "text", success: function(text) {
            MvcUtil.showSuccessResponse(text, button);
        },
        error: function(xhr) {
            MvcUtil.showErrorResponse(xhr.responseText, button);
        }
    });
    return false;
});

$("#writeRss").click(function() {
    var link = $(this);
    $.ajax({ url: link.attr("href"),
        beforeSend: function(req) {
req.setRequestHeader("Accept", "application/rss+xml");
        },
        success: function(feed) {
                MvcUtil.showSuccessResponse(MvcUtil.xmlencode(feed), link);
        },
        error: function(xhr) {
MvcUtil.showErrorResponse(xhr.responseText, link);
        }
    });
    return false;
});
```

<mvc:annotation-driven>快速配置

七、View Rendering(视图渲染)

1. Entity层实现代码

```
package com.springmvc.demo.web.ex06;

public class JavaBean {

    private String foo;
    private String fruit;

    public String getFoo() {
        return foo;
    }

    public void setFoo(String foo) {
        this.foo = foo;
    }

    public String getFruit() {
        return fruit;
    }

    public void setFruit(String fruit) {
        this.fruit = fruit;
    }

}
```

2. Web层实现代码

```
package com.springmvc.demo.web.ex06;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/views/*")
public class ViewsController {

    @RequestMapping(value="html", method= RequestMethod.GET)
    public String prepare(Model model) {
        model.addAttribute("foo", "bar");
        model.addAttribute("fruit", "apple");
        return "page/html";
    }

    @RequestMapping(value="/viewName", method= RequestMethod.GET)
    public void usingRequestToViewNameTranslator(Model model) {
        model.addAttribute("foo", "bar");
        model.addAttribute("fruit", "apple");
    }


    @RequestMapping(value="pathVariables/{foo}/{fruit}", method=
RequestMethod.GET)
    public String pathVars(@PathVariable String foo, @PathVariable String fruit)
{
        // No need to add @PathVariables "foo" and "fruit" to the model
        // They will be merged in the model before rendering
        return "page/html";
    }

    @RequestMapping(value="dataBinding/{foo}/{fruit}", method=
RequestMethod.GET)
    public String dataBinding(JavaBean javaBean, Model model) {
        // JavaBean "foo" and "fruit" properties populated from URI variables
        return "page/dataBinding";
    }

}
```

3. View视图实现代码

创建响应数据的页面：

src\main\webapp\WEB-INF\views\views\html.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
    <title>My HTML View</title>
    <link href="<c:url value="/resources/messages/messages.css" />"
rel="stylesheet"  type="text/css" />
</head>
<body>
<div class="success">
    <h3>foo: "${foo}"</h3>
    <h3>fruit: "${fruit}"</h3>
</div>
</body>
</html>
```

src\main\webapp\WEB-INF\views\views\viewName.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
    <title>My HTML View</title>
    <link href="<c:url value="/resources/messages/messages.css" />"
rel="stylesheet"  type="text/css" />
</head>
<body>
<div class="success">
    <h3>foo: "${foo}"</h3>
    <h3>fruit: "${fruit}"</h3>
</div>
</body>
```

```
</html>
```

src\main\webapp\WEB-INF\views\views\dataBinding.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
    <title>Data Binding with URI Template Variables</title>
    <link href="<c:url value="/resources/messages/messages.css" />"
rel="stylesheet"  type="text/css" />
</head>
<body>
<div class="success">
    <h3>javaBean.foo: ${javaBean.foo}</h3>
    <h3>javaBean.fruit: ${javaBean.fruit}</h3>
</div>
</body>
</html>
```

[SpringMVC 中ModelAndView用法]

修改index.jsp,增加body视图部分代码：

```
<!--......-->
<ul>
    <li><a href="#hello">Hello</a></li>
    <li><a href="#mapping">Request Mapping</a></li>
    <li><a href="#data">Request Data</a></li>
    <li><a href="#responses">Generating Responses</a></li>
    <li><a href="#messageconverters">Http Message Converters</a></li>

    <li><a href="#views">View Rendering</a></li>

</ul>
<!--......-->
<div id="views">
    <h2>View Rendering</h2>
    <p>
        实现代码请参考<code>com.springmvc.demo.web.ex06.ViewsController</code>.
    </p>
    <ul>
        <li>
            <a href="<c:url value="/views/html" />">JSP模板生成的HTML</a>
        </li>
    </ul>
    <ul>
        <li>
            <a href="<c:url value="/views/viewName" />">默认请求按视图名称转换约定
</a>
        </li>
    </ul>
    <ul>
        <li>
```

```
            <a href="<c:url value="/views/pathVariables/bar/apple" />">在视图模板
中使用路径变量</a>
        </li>
    </ul>
    <ul>
        <li>
            <a href="<c:url value="/views/dataBinding/bar/apple" />">使用URI变量进
行数据绑定</a>
        </li>
    </ul>
</div>
```

八、Type Conversion

1.自动数据类型转换

String自动转换为其他8种封装类型。

2.日期格式转换

导入joda-time的依赖

```
<!-- Joda Time Library -->
<dependency>
    <groupId>joda-time</groupId>
    <artifactId>joda-time</artifactId>
    <version>2.3</version>
</dependency>
```

接收日期数据：

String转换为java.util.Date

字段的上方或者参数列表前声明@DateTimeFormat

```
@DateTimeFormat(iso=ISO.DATE_TIME)
@DateTimeFormat(iso=ISO.DATE)
@DateTimeFormat(iso=ISO.TIME)
@DateTimeFormat(pattern = "yyyy/MM/dd HH:mm:ss")
```

响应日期数据：

```
2010/07/04（推荐）
2010-07-04（不推荐）->NaN-NaN-NaN
```

少8小时，设置日期为东8区。

解决：返回字段的getter方法上声明：

```
@JsonFormat(pattern = "yyyy/MM/dd HH:mm:ss",timezone = "GMT+8")
```

东八区设置数据连接的时候解决!

九、Validation

[使用Hibernate-Validator优雅的验证参数]

导入依赖包:

```
<!-- JSR 303 with Hibernate Validator -->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.3.Final</version>
</dependency>
```

修改pom.xml

```
<dependency>
    <groupId>javax.servlet.jsp.jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>${jstl.version}</version>
    <scope>provided</scope>
    <exclusions>
        <exclusion>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

包冲突需要排除servlet-api依赖。

1. Entity层实现代码

```
package com.springmvc.demo.web.ex06;

import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.DateTimeFormat.ISO;

import javax.validation.constraints.Future;
import javax.validation.constraints.Max;
import javax.validation.constraints.NotNull;
import java.util.Date;

public class JavaBean {

    @NotNull    //不可以为null
    @Max(5)     //最大值为5
    private Integer number;

    @NotNull
    @Future //未来的时间
    @DateTimeFormat(iso= ISO.DATE)//接收数据 并进行ISO格式化
    private Date date;

    public Integer getNumber() {
        return number;
    }

    public void setNumber(Integer number) {
        this.number = number;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

}
```

2. Web层实现代码

```
package com.springmvc.demo.web.ex06;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.validation.Valid;

@Controller
public class ValidationController {

    // enforcement of constraints on the JavaBean arg require a JSR-303 provider
on the classpath

    @RequestMapping("/validate")
    public @ResponseBody
    String validate(@Valid JavaBean bean, BindingResult result) {
        if (result.hasErrors()) {
            return "Object has validation errors";
        } else {
            return "No errors";
        }
    }

}
```

使用 @Valid 来进行后台校验

3. View视图实现代码

修改index.jsp,增加body视图部分代码：

```
<div id="validation">
    <h2>Validation</h2>
    <p>
        See the <code>org.springframework.samples.mvc.validation</code> package
for the @Controller code
    </p>
    <ul>
        <li>
            <a id="validateNoErrors" class="textLink" href="<c:url
value="/validate?number=3&date=2029-07-04" />">Validate, no errors</a>
        </li>
        <li>
            <a id="validateErrors" class="textLink" href="<c:url
value="/validate?number=3&date=2010-07-01" />">Validate, errors</a>
        </li>
    </ul>
</div>
```

4.修改核心脚本index.js

```

```

1. 在Entity中声明hibernate-validator校验注解

@NotNull

@Max(5)

@NotNull

@Future

```

```

2. 接收数据时使用@Valid启用校验

```
@RequestMapping("/validate")
public @ResponseBody String validate(@Valid JavaBean bean, BindingResult result)
{
    if(result.hasErrors()) {
        return"Object has validation errors";
    }else {
        return "No errors";
    }
}
```

十、 Forms

参考前面接收和响应表单数据。可以结合Validation一起做表单校验

增加样式：src\main\webapp\resources\form.css

```
/* CLEAN FORM
///////////////////////////////*/

/* General */

.cleanform {
    font-size:1em;
    width:40em;
    color:#1b1b1b;
    text-align:left;
    margin:1em auto
}

/* Elements */

.cleanform  label,.cleanform legend {
    padding:0;
    margin:0.3em 0
}

.cleanform fieldset {
    padding:0.7em;
    border:1px solid #ddd;
    margin:0 0 0.5em 0
}

.cleanform label {
    font-weight:bold
}

.cleanform fieldset input {
    width:70%;
    line-height:1.5em;
    padding:0.15em
}

.cleanform .radio input,
.cleanform .checkbox input {
    width:auto;
    border:none;
    margin:0 1.5em 0 0
}

.cleanform input, .cleanform textarea, .cleanform select {
    display:block;
    margin-bottom:1em;
    font-size:1em;
    border:1px solid #bbb;
    padding:0.15em;
    margin-right:1em
}

.cleanform .radio label, .cleanform .radio input,
.cleanform .checkbox label, .cleanform .checkbox input {
    display:inline;
    margin:0 1.5em 0 0
}

.cleanform .radio input, .cleanform .checkbox input {
    margin:0 0.3em 0 0
}

.cleanform .multiple label{
    float:left;
    width:29%;
    overflow:hidden;
    padding-left:1px
}

.cleanform .multiple input {
    cursor:pointer
}


/* information */

.cleanform .formInfo {
    margin-bottom:1em;
    padding-bottom:0.5em;
    border-bottom:0.1em solid #ddd
}

.cleanform .formInfo h2 {
    color:#00889e;
    font-weight:bold;
    font-size:1.2em;
    margin-bottom:1em
}

.cleanform .formInfo p{
    text-align:justify
}

.cleanform .required {
    color:#ff3838;
    font-weight:bold;
    font-size:0.8em
}
```

十一、File Upload

1.导入commons-fileupload和commons-io支持

```
<!-- File Upload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.2.2</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.0.1</version>
</dependency>
```

2.修改springmvc配置文件，增加上传配置(扩展设置上传限制，例如文件大小)

```
<!-- Only needed because we require fileupload in the
org.springframework.samples.mvc.fileupload package -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />
```

3.修后台接收上传文件（多文件上传）

判断Ajax请求的工具类：

```
package com.springmvc.demo.util;

import org.springframework.web.context.request.WebRequest;

public class AjaxUtils {

    public static boolean isAjaxRequest(WebRequest webRequest) {
        String requestedWith = webRequest.getHeader("X-Requested-With");
        return requestedWith != null ? "XMLHttpRequest".equals(requestedWith) :
false;
    }

    public static boolean isAjaxUploadRequest(WebRequest webRequest) {
        return webRequest.getParameter("ajaxUpload") != null;
    }

    private AjaxUtils() {}
}
```

上传实现代码：

```
package com.springmvc.demo.web.ex08;

import com.springmvc.demo.util.AjaxUtils;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

@Controller
@RequestMapping("/fileupload")
public class FileUploadController {


    //跳转到表单  views/fileupload.jsp
    @RequestMapping(value = "form" , method=RequestMethod.GET)
    public String fileUploadForm() {
        return "fileupload";
    }
    //@RequestParam MultipartFile file   对应 前端 input type="file"  name="file"
    @RequestMapping(method=RequestMethod.POST)
    public void processUpload(@RequestParam MultipartFile file, Model model)
throws IOException {
        System.out.println(  "File:" +file.getOriginalFilename());
        //InputStream  file -> path
    }

}
```

接收@RequestParam MultipartFile file 对象。

4.前端

添加表单工具 src\main\webapp\resources\jquery\js\jquery.form.js

页面代码：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
    <title>fileupload | mvc-showcase</title>
    <script type="text/javascript" src="<c:url
value="/resources/jquery/js/jquery.js" />"></script>
```

```
    <script type="text/javascript" src="<c:url
value="/resources/jquery/js/jquery.form.js" />"></script>
</head>
<body>
    <div id="fileuploadContent">
        <h2>File Upload</h2>
        <p>
            See the <code>org.springframework.samples.mvc.fileupload</code>
package for the @Controller code
        </p>
        <!--
            File Uploads must include CSRF in the URL.
            See http://docs.spring.io/spring-
security/site/docs/3.2.x/reference/htmlsingle/#csrf-multipart
        -->
        <form id="fileuploadForm" action="<c:url value="/fileupload"/>"
method="POST" enctype="multipart/form-data" class="cleanform">
            <div class="header">
                <h2>Form</h2>
            </div>
            <label for="file">File</label>
            <input id="file" type="file" name="file" />
            <p><button type="submit">Upload</button></p>
        </form>
        <script type="text/javascript">
            $(document).ready(function() {
                $('<input type="hidden" name="ajaxUpload" value="true"
/>').insertAfter($("#file"));
                $("#fileuploadForm").ajaxForm({
                    success: function(html) {
                        $("#fileuploadContent").replaceWith(html);
                    }
                });
            });
        </script>
    </div>

</body>
</html>
```

核心代码：

```
@RequestParam MultipartFile file


enctype="multipart/form-data"
<input id="file" type="file" name="file" />
```

十二、Exception Handling

[Spring MVC 异常处理（Exception Handling）](#)

[SPRING MVC异常处理详解 EXCEPTIONHANDLER GOOD](#)

十三、Redirect

```
@RequestMapping(value="/uriTemplate", method=RequestMethod.GET)
public String uriTemplate(RedirectAttributes redirectAttrs) {
   redirectAttrs.addAttribute("account", "a123");  // Used as URI template
variable
   redirectAttrs.addAttribute("date", new LocalDate(2011, 12, 31));  // Appended
as a query parameter
   return "redirect:/redirect/{account}";
}
```

views/redirect/a123.jsp

十四、Async Requests处理

[spring mvc对异步请求的处理](#)

十五、Spring MVC CSRF

[SpringMVC如何防御CSRF](#)

十五、 解决Spring MVC项目中文乱码问题

1. 运行环境的编码设置

1.1 服务器编码：

tomcat/jetty/websphere..

maven插件:

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <port>8080</port>
        <uriEncoding>UTF-8</uriEncoding>
    </configuration>
</plugin>
```

Tomcat配置文件：

apache-tomcat\conf\server.xml

```
<Connector URIEcoding="UTF-8" connectionTimeout="20000" port="8080"
protocol="HTTP/1.1" redirectPort="8443"/>

<Connector URIEcoding="UTF-8" port="8009" protocol="AJP/1.3"
redirectPort="8443"/>
```

1.2 数据库编码设置

- 创建数据库选中UTF-8编码

```
CREATE SCHEMA `new_schemaa` DEFAULT CHARACTER SET utf8 ;
```

- 连接数据库设置URL：

```
jdbc:mysql://localhost:3306/dbname?useUnicode=true&characterEncoding=UTF-8
```

1.3 IDE工具工作区间编码设置

```
Preferences->General->Workspace->Test file encoding->UTF-8
```

1.4 Jsp编码设置：

```
<%@ page language="java" contentType="text/html;charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Index Page</title>
</head>
<body>

</body>
</html>
```

1.5 Spring Web项目配置

1. 修改项目web.xml，增加编码过滤器，接收请求数据时过滤

```
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. Spring mvc设置全局响应编码(配合@ResponseBody使用):

```
<mvc:annotation-driven enable-matrix-variables="true">
    <mvc:message-converters register-defaults="true">
        <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
            <constructor-arg value="UTF-8" />
        </bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```

1.6 响应JSON编码设置（可选）

```
response.setContentType("application/json;charset=UTF-8");
```

1.7 接收请求数据为乱码：

```
request.setCharacterEncoding("UTF-8");
String str = new String((request.getParameter("pa")).getBytes("iso-8859-
1"),"utf-8");
//可以打印当前编码:
System.out.println(request.getCharacterEncoding());
```

附录：Spring MVC Annotation汇总

Spring2.5引入注解式处理器支持，通过@Controller 和 @RequestMapping注解定义我们的处理器类。并且提供了一组强大的注解：

需要通过处理器映射DefaultAnnotationHandlerMapping和处理器适配器AnnotationMethodHandlerAdapter来开启支持@Controller 和 @RequestMapping注解的处理器。

@Controller：用于标识是处理器类；

@RequestMapping：请求到处理器功能方法的**映射规则**；

value="/simple/revisited"：请求url，没有其他属性可以不声明value=。

method=RequestMethod.GET,：指定请求方式。

headers="Accept=text/plain"

@PostMapping

@GetMapping

@RequestParam：请求参数到处理器功能处理方法的方法参数上的绑定；

@ModelAttribute：请求参数到命令对象的绑定；

@SessionAttributes：用于声明session级别存储的属性，放置在处理器类上，通常列出模型属性（如@ModelAttribute）对应的名称，则这些属性会透明的保存到session中；

@InitBinder：自定义数据绑定注册支持，用于将请求参数转换到命令对象属性的对应类型；

三、Spring3.0引入RESTful架构风格支持(通过@PathVariable注解和一些其他特性支持),且又引入了更多的注解支持：

@CookieValue：cookie数据到处理器功能处理方法的方法参数上的绑定；

@RequestHeader：请求头（header）数据到处理器功能处理方法的方法参数上的绑定；

@RequestBody：请求的body体的绑定（通过HttpMessageConverter进行类型转换）；

@ResponseBody：处理器功能处理方法的返回值作为响应体（通过HttpMessageConverter进行类型转换）；

@ResponseStatus：定义处理器功能处理方法/异常处理器返回的状态码和原因；

@ExceptionHandler：注解式声明异常处理器；

@PathVariable：请求URI中的模板变量部分到处理器功能处理方法的方法参数上的绑定，从而支持RESTful架构风格的URI；

Spring mvc4.x 注解概述

四、还有比如：

JSR-303验证框架的无缝支持（通过@Valid注解定义验证元数据）；

使用Spring 3开始的ConversionService进行类型转换（PropertyEditor依然有效），支持使用@NumberFormat 和 @DateTimeFormat来进行数字和日期的格式化；

HttpMessageConverter（Http输入/输出转换器，比如JSON、XML等的数据输出转换器）；