IT楠老师SpringMVC教程

一、什么是SpringMVC

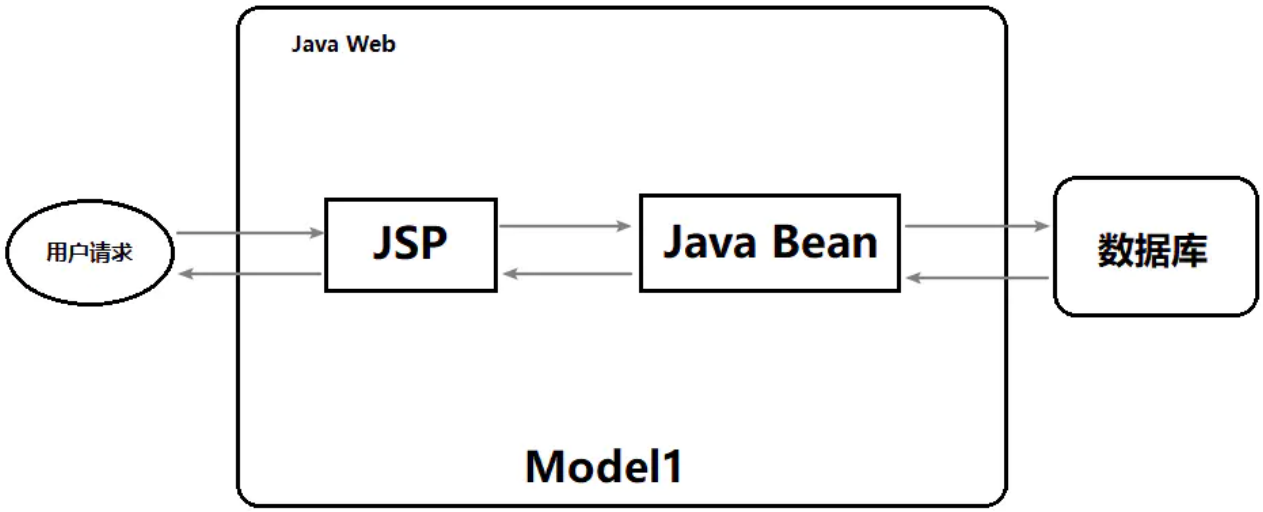
B站：IT楠老师 公众号：IT楠说jiava QQ群： 1083478826 新知大数据

制作不易、如果觉的好不妨打个赏：

Spring MVC是Spring Framework的一部分，是基于Java实现MVC的轻量级Web框架。

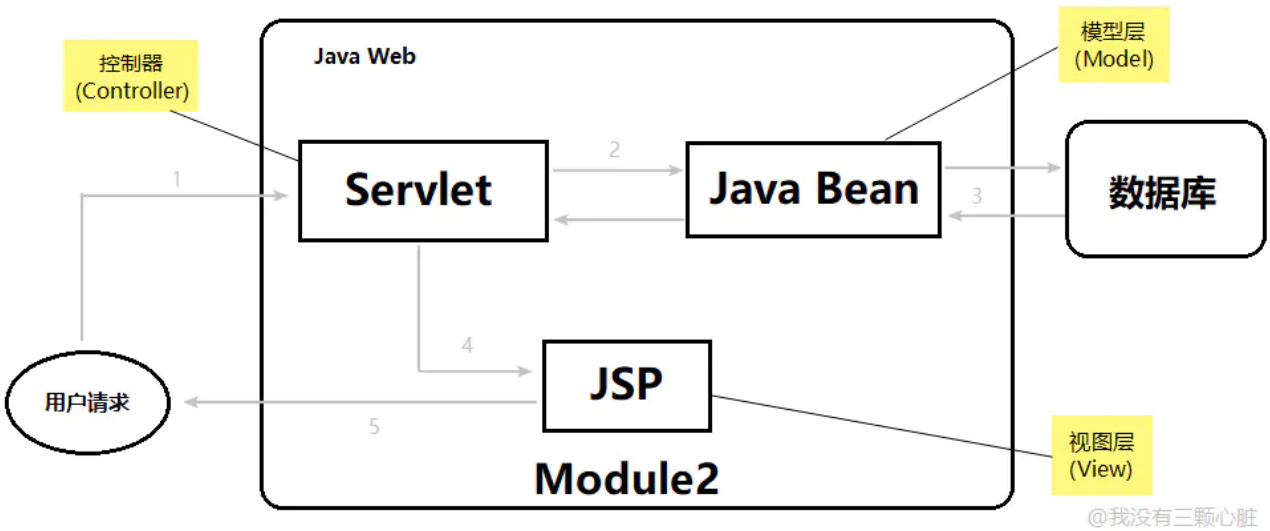
MVC 设计概述

在早期 Java Web 的开发中，统一把显示层、控制层、数据层的操作全部交给 JSP 或者 JavaBean 来进行处理，我们称之为 **Model1：**



- **出现的弊端：**
- JSP 和 Java Bean 之间严重耦合，Java 代码和 HTML 代码也耦合在了一起
- 要求开发者不仅要掌握 Java ， 还要有不错的前端水平
- 前端和后端相互依赖，前端需要等待后端完成，后端也依赖前端完成，才能进行有效的测试
- 代码难以复用

正因为上面的种种弊端，所以很快这种方式就被 Servlet + JSP + Java Bean 所替代了，早期的 MVC 模型（**Model2**）就像下图这样：



首先用户的请求会到达 Servlet，然后根据请求调用相应的 Java Bean，并把所有的显示结果交给 JSP 去完成，这样的模式我们就称为 MVC 模式。

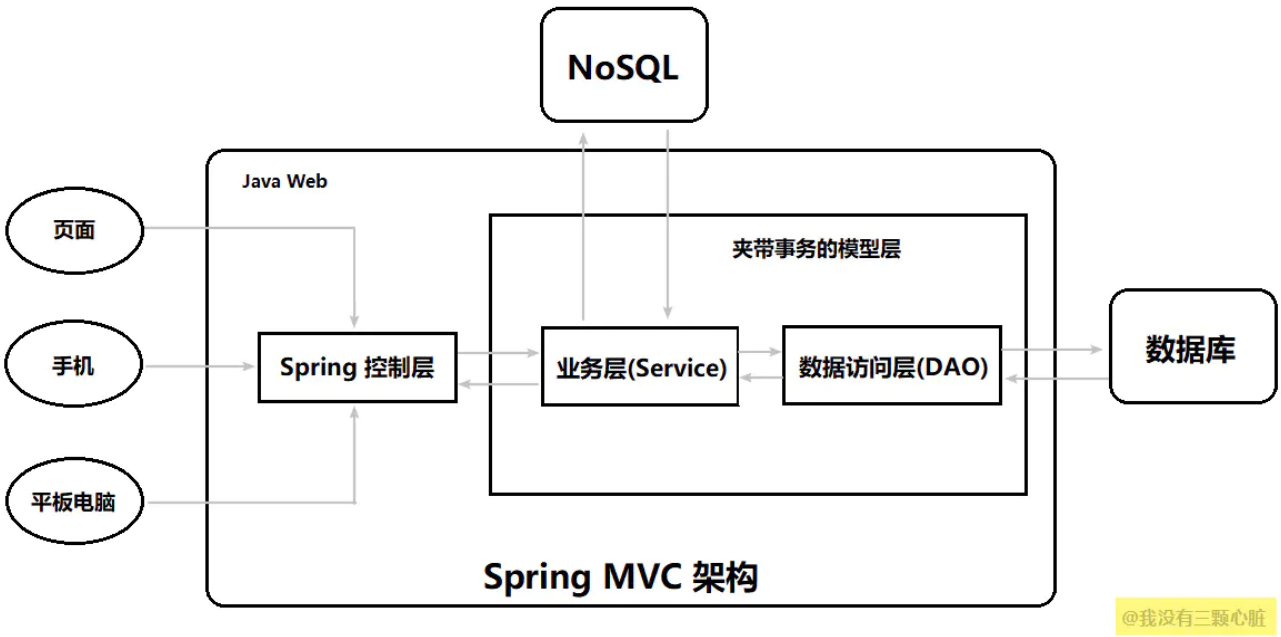
- **M 代表 模型（Model）** 模型是什么呢？ 模型就是数据，就是 dao,bean
- **V 代表 视图（View）** 视图是什么呢？ 就是网页, JSP，用来展示模型中的数据

- C 代表 控制器（controller）

控制器是什么？ 控制器的作用就是把不同的数据(Model)，显示在不同的视图(View)上，Servlet 扮演的就是这样的角色。

> **Spring MVC 的架构**

为解决持久层中一直未处理好的数据库事务的编程，又为了迎合 NoSQL 的强势崛起，Spring MVC 给出了方案：



传统的模型层被拆分为了业务层(Service)和数据访问层（DAO,Data Access Object）。 在 Service 下可以通过 **Spring 的声明式事务操作数据访问层**，而在业务层上还允许我们访问 NoSQL，这样就能够满足异军突起的 NoSQL 的使用了，它可以大大提高互联网系统的性能。

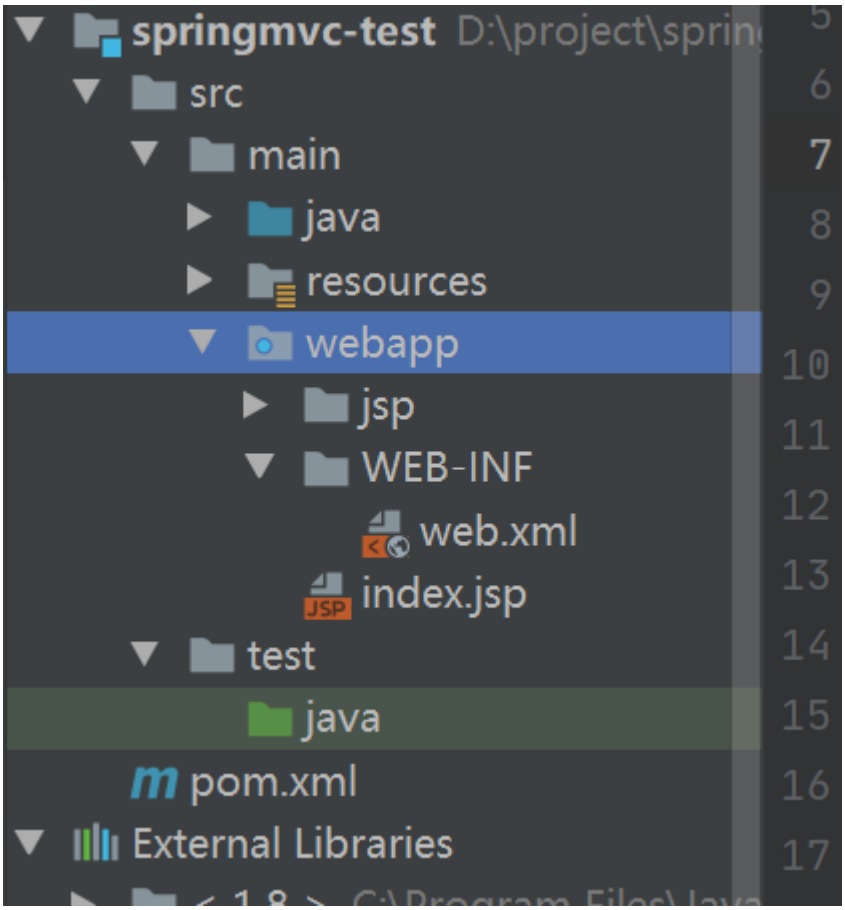
- **特点：** 结构松散，几乎可以在 Spring MVC 中使用各类视图 松耦合，各个模块分离 与 Spring 无缝集成

> **现在springmvc使用的公司越来越多，已经成为了霸主地位，基本上取代了早年的struts2，但是我们不能否仍依然有一些公司在使用老的框架，但是触类旁通，希望有机会自行了解。**

二、直接上代码

1、创建基本web工程

完善webapp工程必备目录



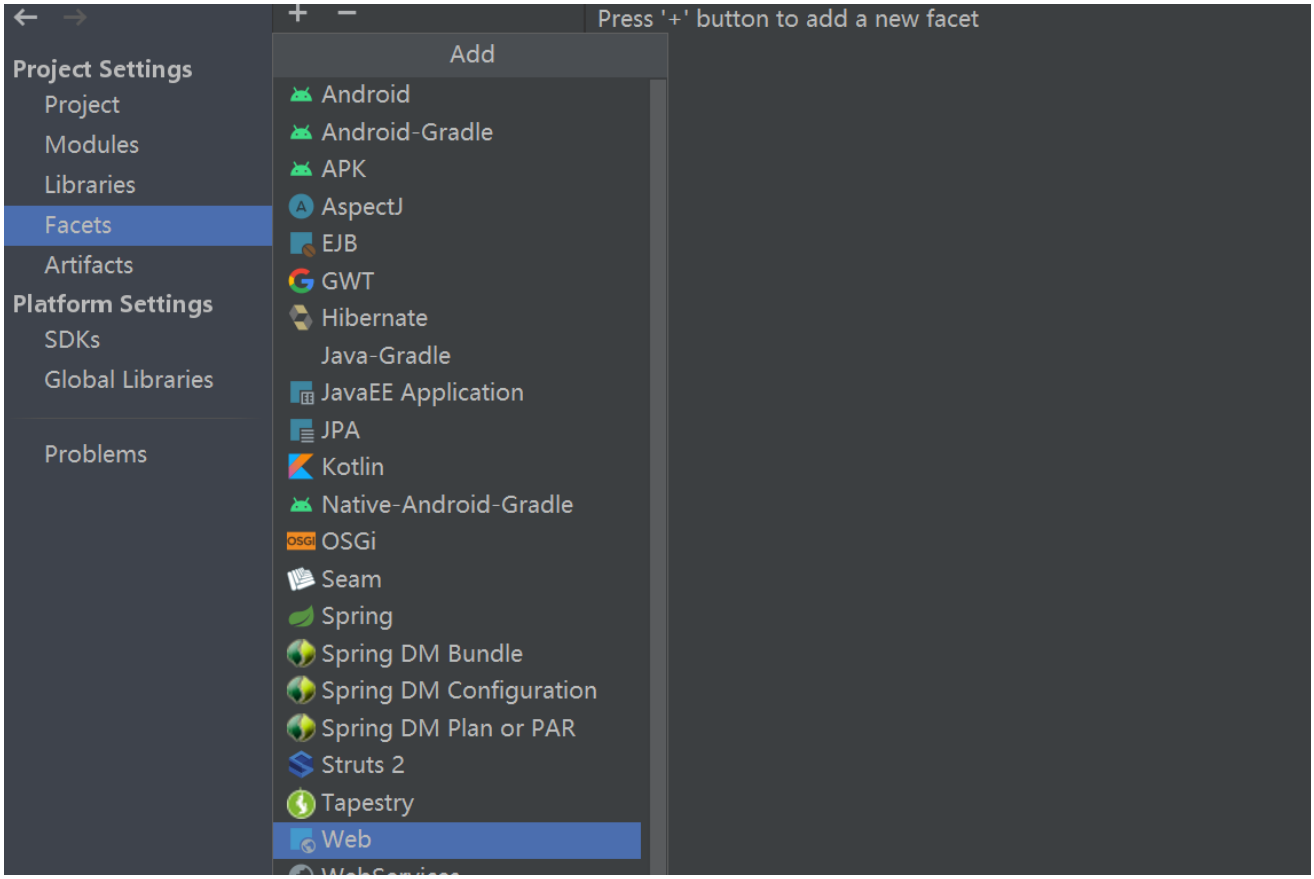
web.xml基本内容

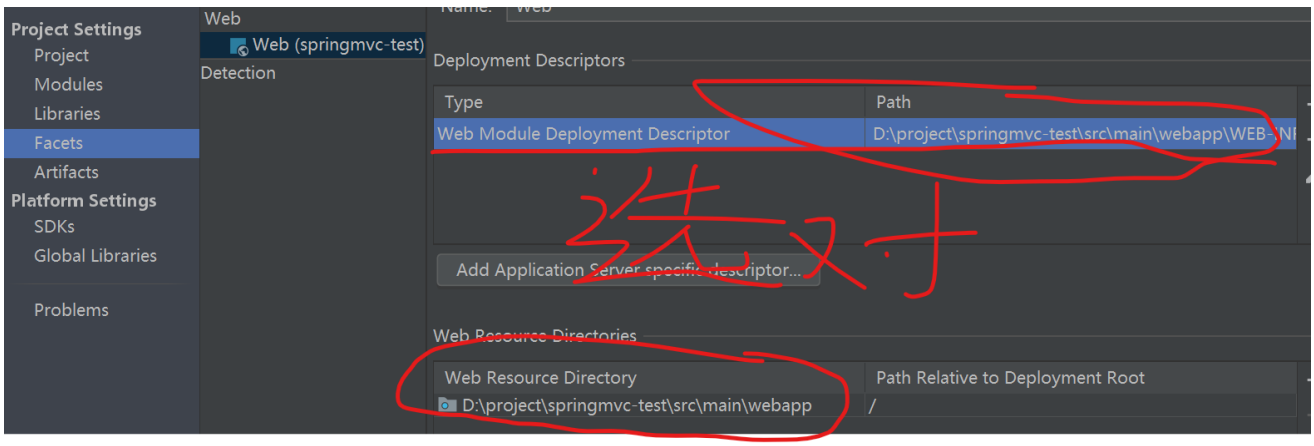
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
</web-app>
```

简单构建一下

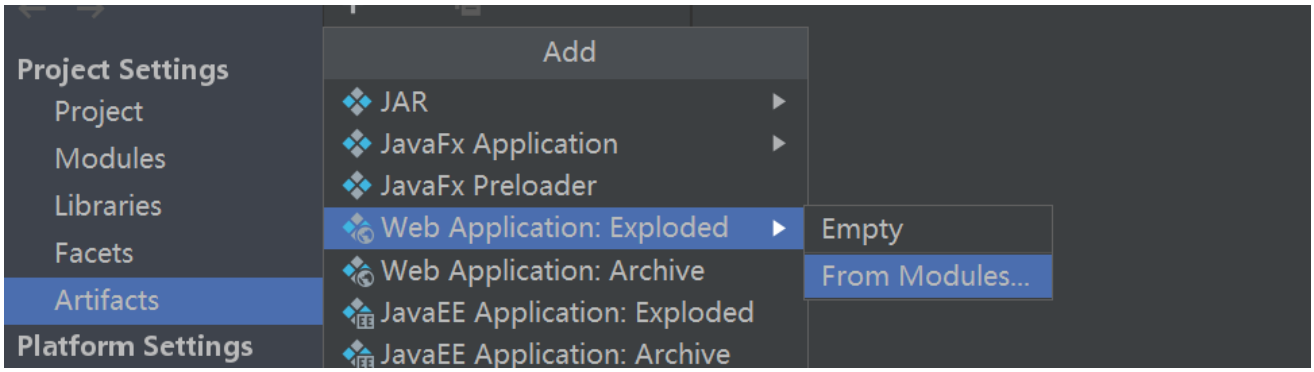
为的是idea能跑起来

Facets表述了在Module中使用的各种各样的框架、技术和语言。这些Facets让Intellij IDEA知道怎么对待module内容，并保证与相应的框架和语言保持一致。此处添加web项。注意配置。

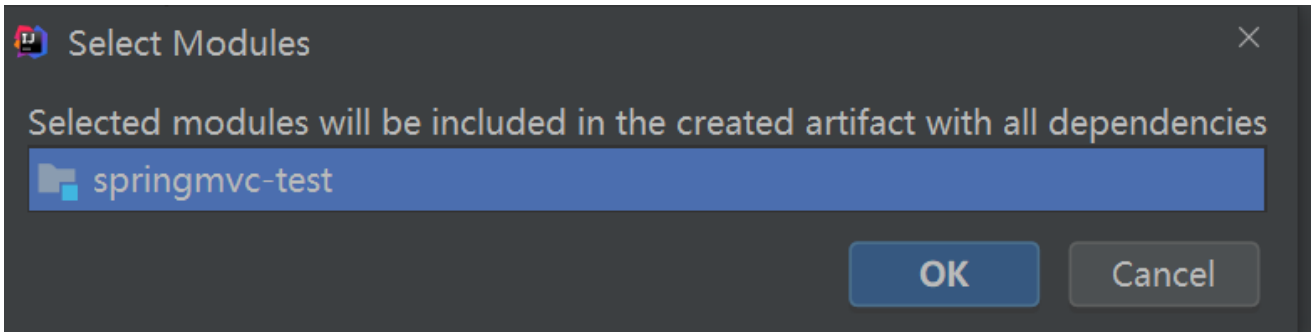




增加打包方式，Exploded是不压缩方式，打包后是文件夹形式，使用此方式部署，可以进行热启动，修改class文件和资源文件可以实时修改，但是Archive是打包后是真正的war包。

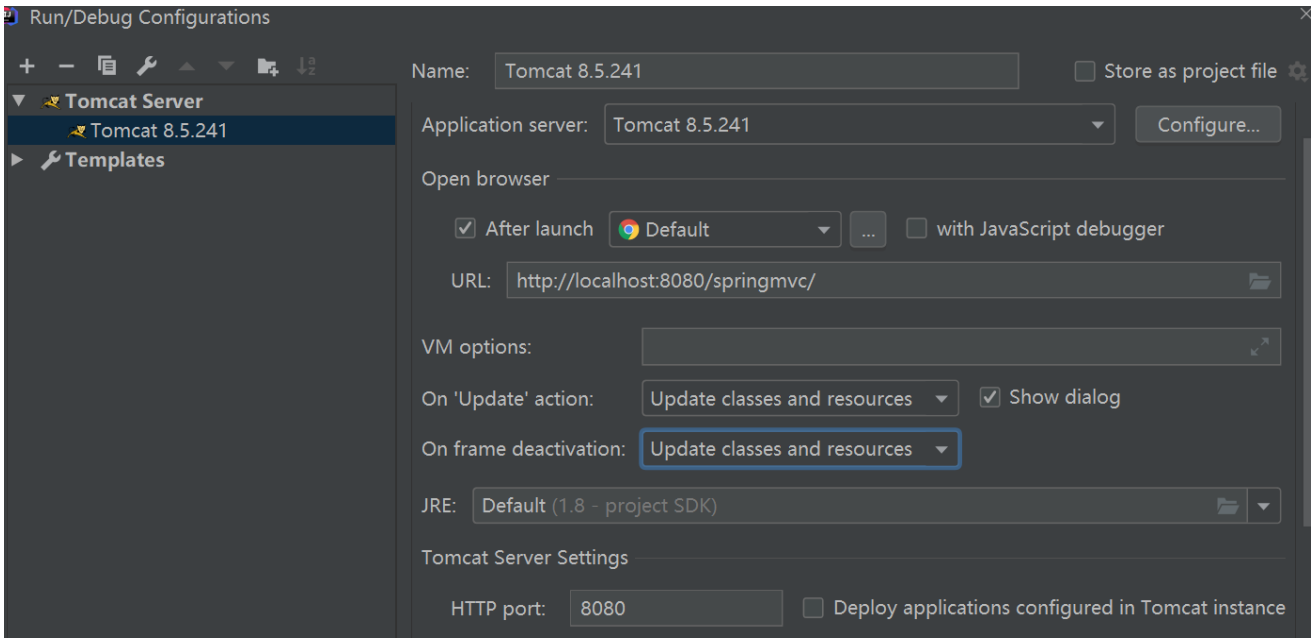


选择打包模块



配置tomcat

应该都会了



使用setvlet测试

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@WebServlet("/test")
public class TestServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.getWriter().println("hello servlet!!");
    }
}
```

可以再加一个index.jsp

启动tomcat,ok

2、搭建springmvc环境

（1）首先完整的pom

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.xinzi</groupId>
    <artifactId>test-spring-mvc</artifactId>
    <version>1.0.0</version>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.2.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.2</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
```

```
                <configuration>
                    <source>${java.version}</source> <!-- 源代码使用的JDK版本 -->
                    <target>${java.version}</target> <!-- 需要生成的目标class文件的编译版本
-->
                    <encoding>${project.build.sourceEncoding}</encoding><!-- 字符集编码 -
->
                </configuration>
            </plugin>
        </plugins>
        <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
        </resources>
    </build>

</project>
```

(2) 配置web.xml

> **注册DispatcherServlet**

```
<!--注册DispatcherServlet，这是springmvc的核心，就是个servlet-->
<servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--加载时先启动-->
    <load-on-startup>1</load-on-startup>
</servlet>
<!--/ 匹配所有的请求；（不包括.jsp）-->
<!--/* 匹配所有的请求；（包括.jsp）-->
<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

(3) 编写配置文件

> **名称：springmvc-servlet.xml (其实就是个spring的配置文件) 写在/WEB-INF/下**

> **在视图解析器中我们把所有的视图都存放在/WEB-INF/目录下，这样可以保证视图安全，因为这个目录下的文件，客户端不能直接访问。**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 处理映射器 -->
    <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
    <!-- 处理器适配器 -->
    <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>
    <!--视图解析器:DispatcherServlet给他的ModelAndView-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="InternalResourceViewResolver">
        <!--前缀-->
        <property name="prefix" value="/WEB-INF/page/"/>
        <!--后缀-->
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

(4) 编写Controller

- 方式一：实现Controller接口
- 方式二：增加注解；
- 注意：需要返回一个ModelAndView，这个对象封装了视图和模型；

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
public class FirstController implements Controller {

    public ModelAndView handleRequest(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse) throws Exception {
        //ModelAndView 封装了模型和视图
        ModelAndView mv = new ModelAndView();
        //模型里封装数据
        mv.addObject("hellomvc","Hello springMVC!");
        //封装跳转的视图
        mv.setViewName("hellomvc");
        //不是有个视图解析器吗
        //这玩意就是为了省事的，自动给你加个前缀后缀
        //就成了 /jsp/hellomvc.jsp  不就是拼串吗
        return mv;
    }
}
```

（4）controller注入容器

> **注意此时的id就成了你要访问的url了**

```
<bean id="/hellomvc" class="com.xinzhi.controller.FirstController"/>
```

（5）创建jsp页面

> **使用el表达式获取模型数据**

```
<body>
    ${hellomvc}
</body>
```

（6）测试抛异常

```
org.springframework.beans.factory.BeanDefinitionStoreException:
IOException parsing XML document from ServletContext resource [/WEB-INF/spring-mvc-
servlet.xml];
nested exception is java.io.FileNotFoundException:
Could not open ServletContext resource [/WEB-INF/spring-mvc-servlet.xml]
```

注意：springmvc默认回去找`/WEB-INF/spring-mvc-servlet.xml`这个文件，不去类路径找。

所以解决方法：

> **解决方案一：**

在`WEB-INF`建立`spring-mvc-servlet.xml`文件，名字不能错

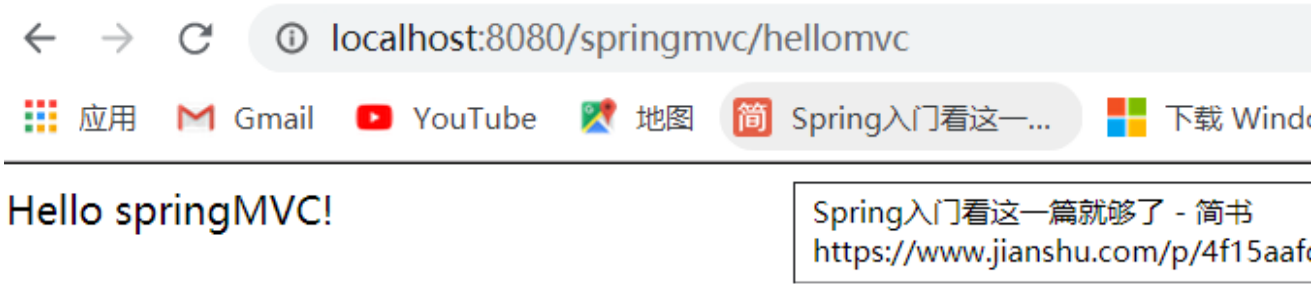
> **解决方案二：**

指定加载的配置文件

```
<!--注册DispatcherServlet，这是springmvc的核心，就是个servlet-->
<servlet>
    <servlet-name>spring-mvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<!--/ 匹配所有的请求；（不包括.jsp）-->
<!--/* 匹配所有的请求；（包括.jsp）-->
<servlet-mapping>
    <servlet-name>spring-mvc</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

(6) 配置Tomcat 启动测试

> 成功



3、使用注解来一波

> 记住一点，只要用注解就得扫包，要不然鬼知道，哪个类有注解

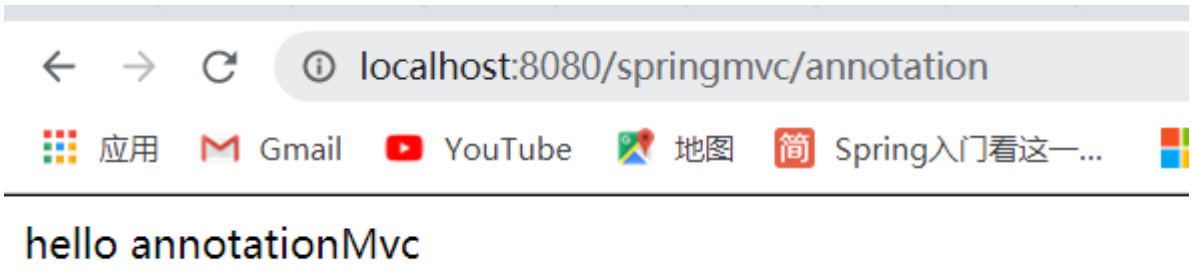
(1) 配置文件

```
<?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
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       https://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- 自动扫包 -->
    <context:component-scan base-package="com.xinzhi"/>
    <!-- 让Spring MVC不处理静态资源 -->
    <mvc:default-servlet-handler />
    <!--  让springmvc自带的注解生效  -->
    <mvc:annotation-driven />
     <!-- 处理映射器 -->
    <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
    <!-- 处理器适配器 -->
    <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>
    <!-- 视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    id="internalResourceViewResolver">
        <!-- 前缀 -->
        <property name="prefix" value="/WEB-INF/page/" />
        <!-- 后缀 -->
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

(2) 编写controller

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Controller
public class AnnotationController {

    @RequestMapping("/annotation")
    public ModelAndView testAnnotation(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("hello","hello annotationMvc");
        modelAndView.setViewName("annotation");
        return modelAndView;
    }

}
```

(3) 启动tomcat测试



> **注解用起来是不是贼爽，这也是企业里常用的。**

三、聊聊过程

1、中心控制器

> **Spring的web框架围绕DispatcherServlet设计。**

- DispatcherServlet的作用是将请求分发到不同的处理器。从Spring 2.5开始，使用Java 5或者以上版本的用户可以采用基于注解的controller声明方式。
- Spring MVC框架像许多其他MVC框架一样, **以请求为驱动** , **围绕一个中心Servlet分派请求及提供其他功能**，**DispatcherServlet是一个实际的Servlet (它继承自HttpServlet 基类)。**

> **注意一点——不要被各种器吓着哈，其实就是个方法或者类**

2、组件说明

Handler：处理器

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理，由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发Handler。

这玩意就是你写的controller，别把他想成啥高级玩意，你也能写个处理器。

View：视图

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。我们最常用的视图就是jsp。

这玩意就是我们写的jsp，牛逼不

DispatcherServlet：中央控制器

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，

由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

这玩意是核心，就是门卫传达室，一个请求进来先来传达室，然后一步步来处理问题。就是个servlet。

HandlerMapping：处理器映射器

HandlerMapping负责根据用户请求url找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，如配置文件方式，实现接口方式，注解方式等。

这玩意就是个map，放了一堆数据，key是url，value是你对应的处理器。一个请求来了，调用一下mao.get(url)就知道哪个类的哪个方法处理这个请求了。当然实际上会将这个url多对应的拦截器（马上学），处理器都拿到。

HandlAdapter：处理器适配器

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

这货会调用相应的方法，生成最终能够的modelAndView。

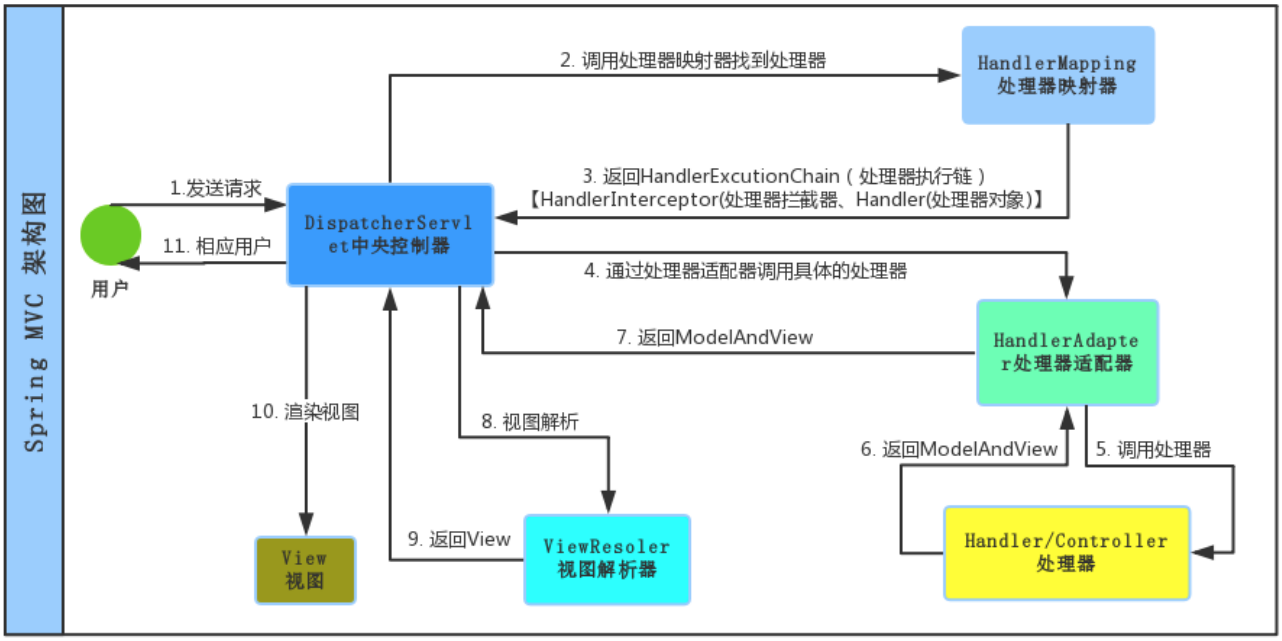
ViewResolver：视图解析器

View Resolver负责将处理结果生成View视图，View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

这货就是解析modelAndView的。有个常用最简单的功能就是拼接字符串，给你加个前缀后缀，让你方便了很多，当然他们解析很多类型的视图。

3、执行流程

1. 直接看图



四、深度学习

> **思考：所有的方法都返回ModelAndView是不是不太好?**

视图和模型是两个东西，耦合在一起当然不好了，比如有时候我只是跳转视图，有时候压根不需要跳转视图，是不是有更好的解决方案?

当然:

既然存在ModelAndView就应该存到Model和View。

> **springMvc的解决方案是:**

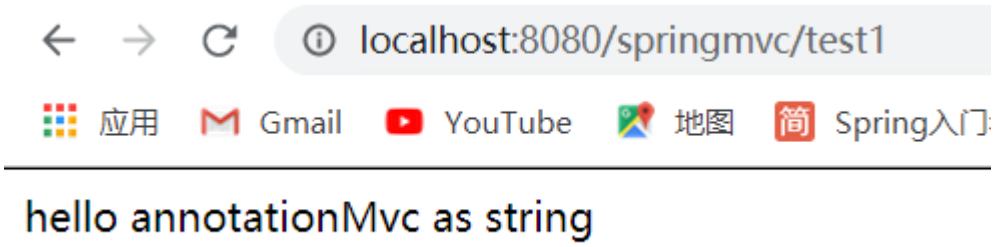
- Model会在调用时通过参数形式传入。
- View可以简化为字符串形式返回。

如此解决才是企业常用的最优解。

1、视图模型拆分案例

```
@RequestMapping("/test1")
public String testAnnotation(Model model){
    model.addAttribute("hello","hello annotationMvc as string");
    return "annotation";
}
```

结果:



> **这才是最好的方法**

- 如果只需要处理model，则返回null，当然一般都会有数据传入
- 如果只处理view，则不用传入参数model

2、@RequestMapping

- 这个注解很关键，可以方法类上也可以放在方法上。
- 如果放在类上，会给每个方法默认都加上，相当于本类全局前缀。

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Controller
@RequestMapping("/user/")
public class AnnotationController {

    @RequestMapping("register")
    public String register(Model model){
        ......
        return "register";
    }

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

> **好处**

- 一个类一般处理一类事物，可以统一加上前缀，好区分
- 简化书写复杂度

RequestMapping注解有六个属性。

1、`value`， `method`；

- value： 指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；
- method： 指定请求的method类型， GET、POST、PUT、DELETE等；

2、`consumes`，`produces`；

- consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;
- produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回,可以处理乱码

```
@GetMapping(value = "{id}",produces = {"content-type:text/json;charset=utf-8"})
```

3、`params`，`headers`；

- params： 指定request中必须包含某些参数值是，才让该方法处理。
- headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

3、内置统一的字符集处理

> **在`web.xml`中配置一个字符集过滤器即可**

```
<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>
```

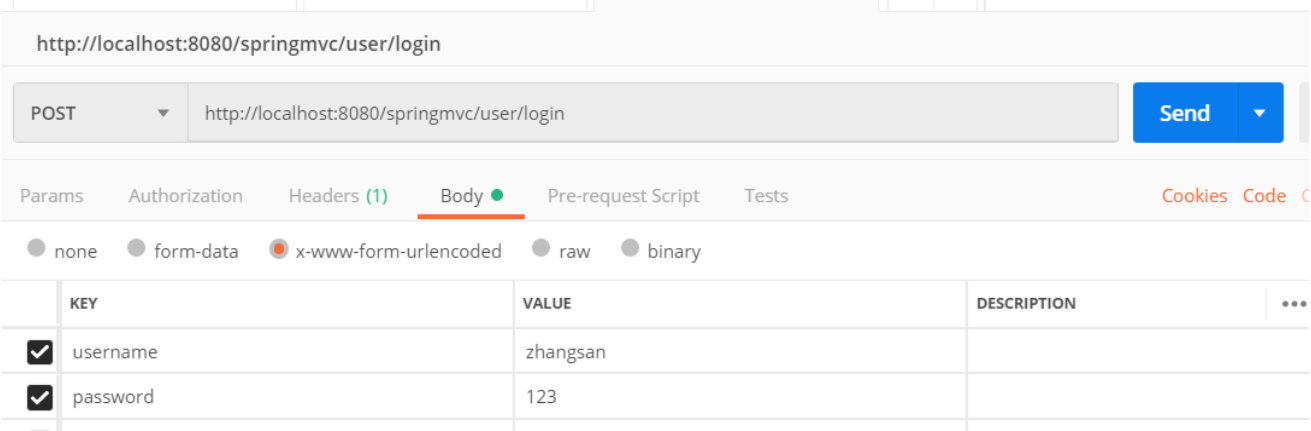
4、牛逼的传参

> **再也不用`getParamter`了**

使用postman模拟请求：

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Controller
@RequestMapping("/user/")
public class AnnotationController {

    @RequestMapping("login")
    public String login(String username,String password){
        System.out.println(username);
        System.out.println(password);
```

```
        return "login";
    }
}
```



结果

```
zhangsan
123
```

那么问题又来了

如果一个表单几十个参数怎么获取啊?

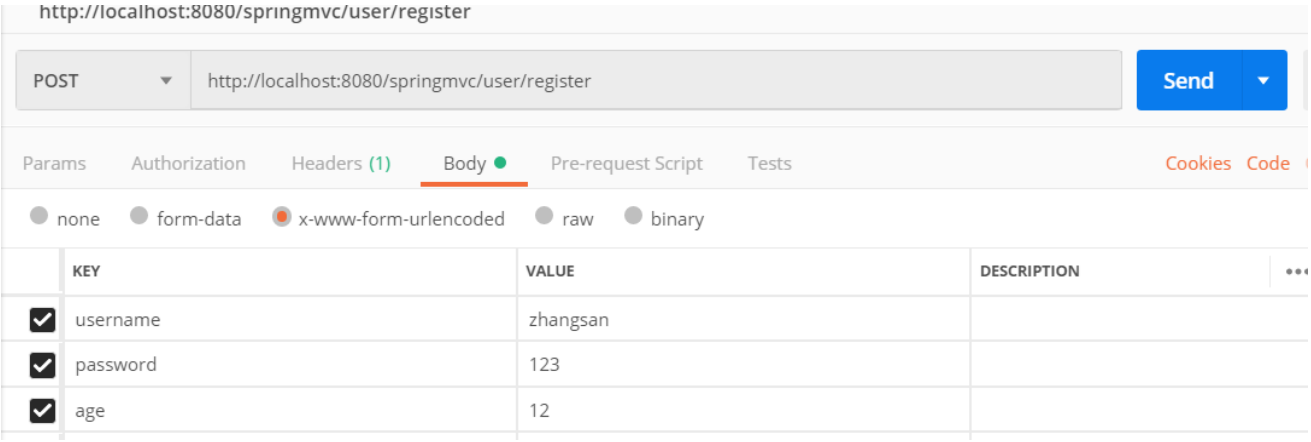
更牛的来了

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
public class User {

    private String username;
    private String password;
    private int age;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
```

```
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Controller
@RequestMapping("/user/")
public class AnnotationController {

    @RequestMapping("register")
    public String register(User user){
        System.out.println(user.getUsername());
        System.out.println(user.getPassword());
        System.out.println(user.getAge());
        return "register";
    }

    @RequestMapping("login")
    public String login(String username,String password){
        System.out.println(username);
        System.out.println(password);
        return "login";
    }
}
```

```
@RequestMapping("register")
public String register(User user){
    System.out.println(user.getUsername());
    System.out.println(user.getPassword());
    System.out.println(user.getAge());
    return "register";
}
```




```
zhangsan
123
12
```

哈哈，牛逼吧！

5、返回json数据

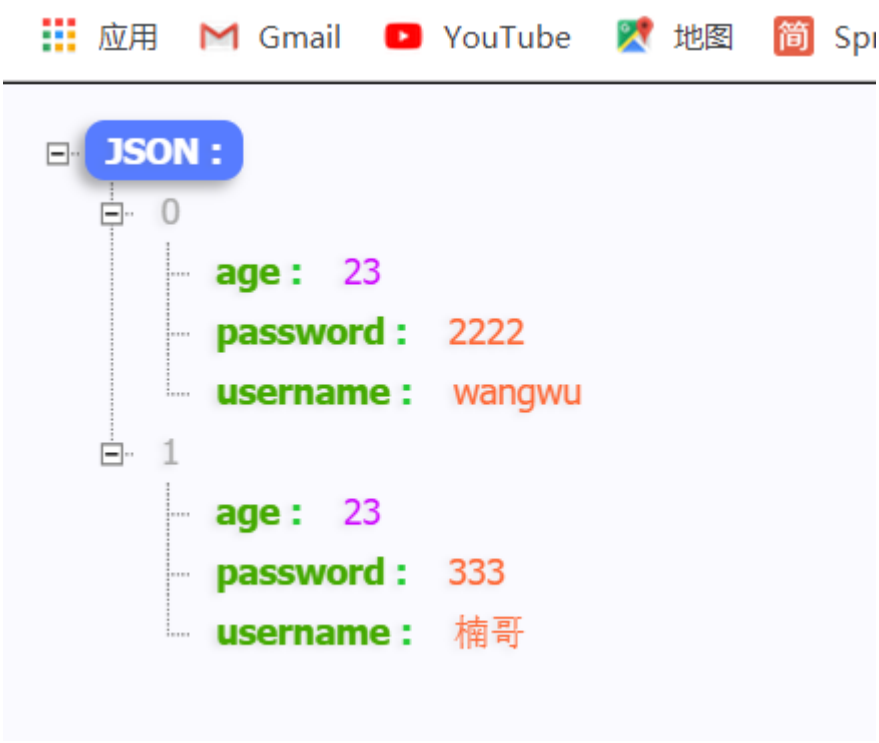
> **见得太多了，我们经常需要使用ajax请求后台获取参数**

> **最熟悉的fastjson**

```
<dependency>
   <groupId>com.alibaba</groupId>
   <artifactId>fastjson</artifactId>
   <version>1.2.68</version>
</dependency>
```

```
@RequestMapping(value = "getUsers",produces = {"text/json;charset=utf-8"})
@ResponseBody
public String getUsers(){
    List<User> users =  new ArrayList<User>(){{
        add(new User("wangwu","2222",23));
        add(new User("楠哥","333",23));
    }};
    return JSONArray.toJSONString(users);
}
```

测试：



成功!

@ResponseBody能将处理的结果放在响应体中，直接返回，不走视图解析器。

> **每次都需要自己处理数据麻烦**

当然能配置

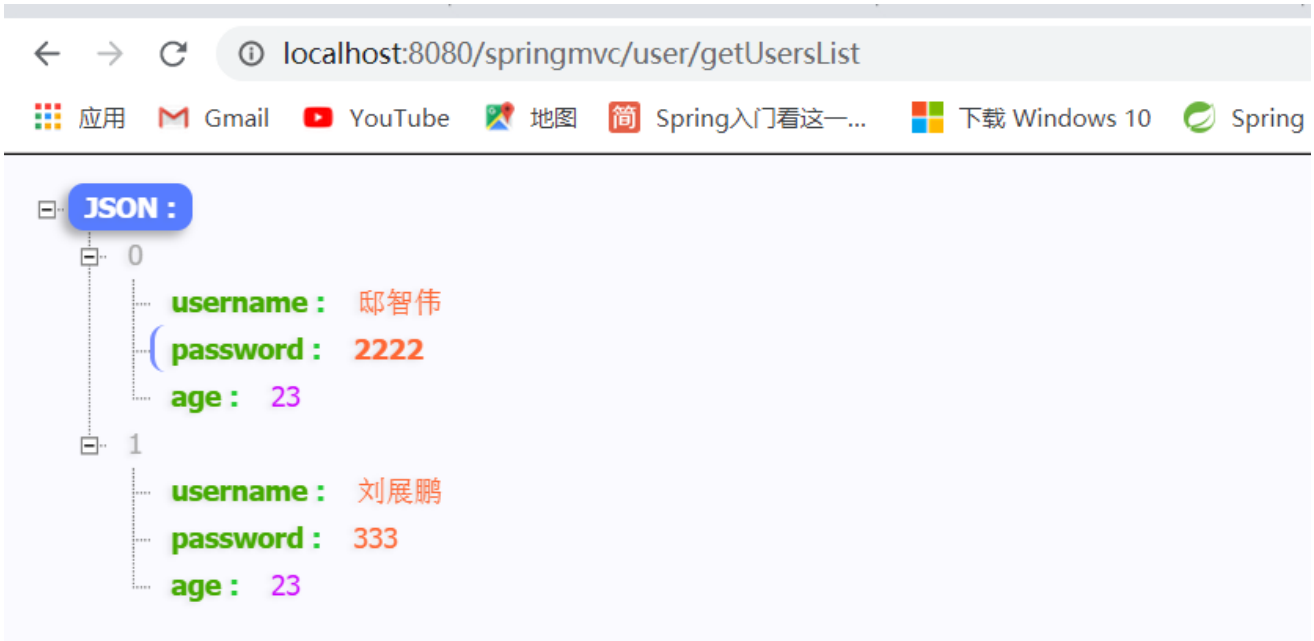
给容器注入一个消息转换的bean

```
<mvc:annotation-driven >
        <mvc:message-converters>
            <bean id="fastjson"
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">
                <property name="supportedMediaTypes">
                    <list>
                        <value>text/html;charset=UTF-8</value>
                        <value>application/json;charset=UTF-8</value>
                    </list>
                </property>
            </bean>
        </mvc:message-converters>
</mvc:annotation-driven>
```

controller

```
@RequestMapping(value = "getUsersList")
@ResponseBody
public List<User> getUsersList(){
    return   new ArrayList<User>(){{
        add(new User("邸智伟","2222",23));
        add(new User("刘展鹏","333",23));
    }};
}
```

测试，搞定



> **只要向容器注入一个用于转换的bean即可，除了fastjson还有jakson等**

6、获取请求体内的json数据

> **博客学习**

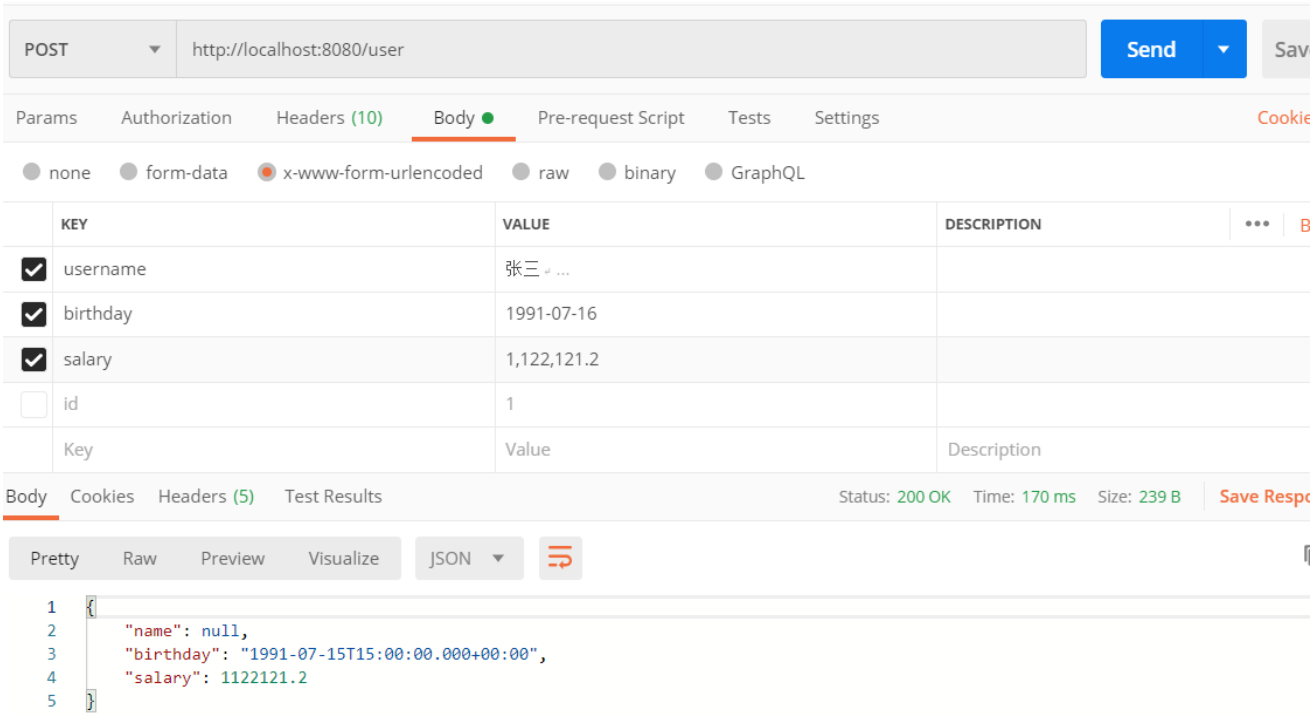
https://www.cnblogs.com/mmzuo-798/p/11634055.html

7、数据转化

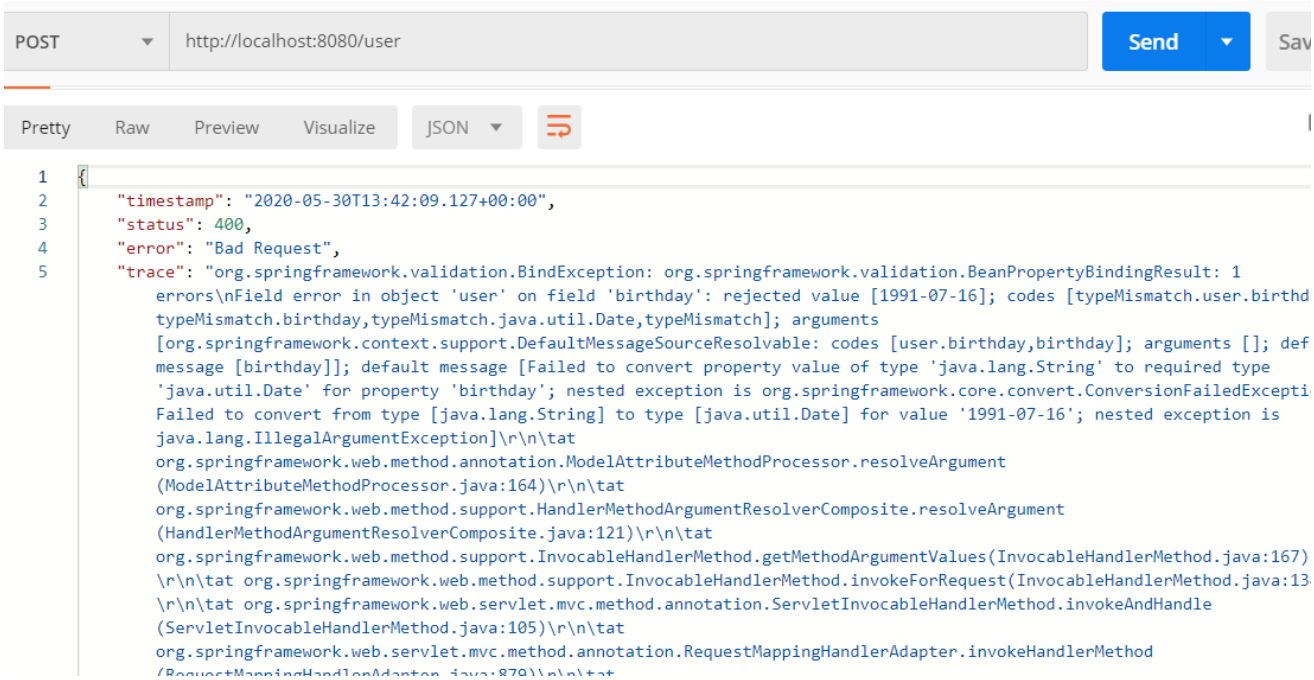
在我们传递表单参数时，表单输入的都是字符串，但是一些简单的转化他居然能自行完成?

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private String name;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date birthday;
    @NumberFormat(pattern = "#,###,###.#")
    private Double salary;
}
```

```java
/**
 * @author IT楠老师
 * @date 2020/5/30
 */
@RestController
public class HelloController {
    @PostMapping("/user")
    public User hello(User user){
        return  user;
    }
}
```



m



8、数据校验

JSR 303

JSR 303 是 Java 为 Bean 数据合法性校验提供的标准框架，它已经包含在 JavaEE 6.0 中 JSR 303 通过在 Bean 属性上标注类似于 @NotNull、@Max 等标准的注解指定校验规则，并通过标准的验证接口对 Bean 进行验证

Constraint	详细信息
`@Null`	被注释的元素必须为 `null`
`@NotNull`	被注释的元素必须不为 `null`
`@AssertTrue`	被注释的元素必须为 `true`
`@AssertFalse`	被注释的元素必须为 `false`
`@Min(value)`	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
`@Max(value)`	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
`@DecimalMin(value)`	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
`@DecimalMax(value)`	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
`@Size(max, min)`	被注释的元素的大小必须在指定的范围内
`@Digits (integer, fraction)`	被注释的元素必须是一个数字，其值必须在可接受的范围内
`@Past`	被注释的元素必须是一个过去的日期
`@Future`	被注释的元素必须是一个将来的日期
`@Pattern(value)`	被注释的元素必须符合指定的正则表达式

Hibernate Validator 扩展注解

Hibernate Validator 是 JSR 303 的一个参考实现，除支持所有标准的校验注解外，它还支持以下的扩展注解

Hibernate Validator 附加的 constraint

Constraint	详细信息
`@Email`	被注释的元素必须是电子邮箱地址
`@Length`	被注释的字符串的大小必须在指定的范围内
`@NotEmpty`	被注释的字符串的必须非空
`@Range`	被注释的元素必须在合适的范围内

Spring MVC 数据校验

Spring MVC 除了会将表单/命令对象的校验结果保存到对应的 BindingResult 或 Errors 对象中外，还会将所有校验结果保存到 “隐含模型” 即使处理方法的签名中没有对应于表单/命令对象的结果入参，校验结果也会保存在 “隐含对象” 中。 隐含模型中的所有数据最终将通过 HttpServletRequest 的属性列表暴露给 JSP 视图对象，因此在 JSP 中可以获取错误信息

实际操作

引入jar包

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>6.0.9.Final</version>
</dependency>
```

实体类加注解

```
public class User {
    @NotEmpty(message = "姓名不能为空")
    @Size(min = 4,max = 20,message = "姓名长度必须在{min}-{max}之间")
    private String name;

    @Min(value = 0, message = "年龄不能小于{value}")
    @Max(value = 120,message = "年龄不能大于{value}")
    private int age;

    @Pattern(regexp = "^1([358][0-9]|4[579]|66|7[0135678]|9[89])[0-9]{8}$", message =
"手机号码不正确")
    private String phone;
}
```

配置

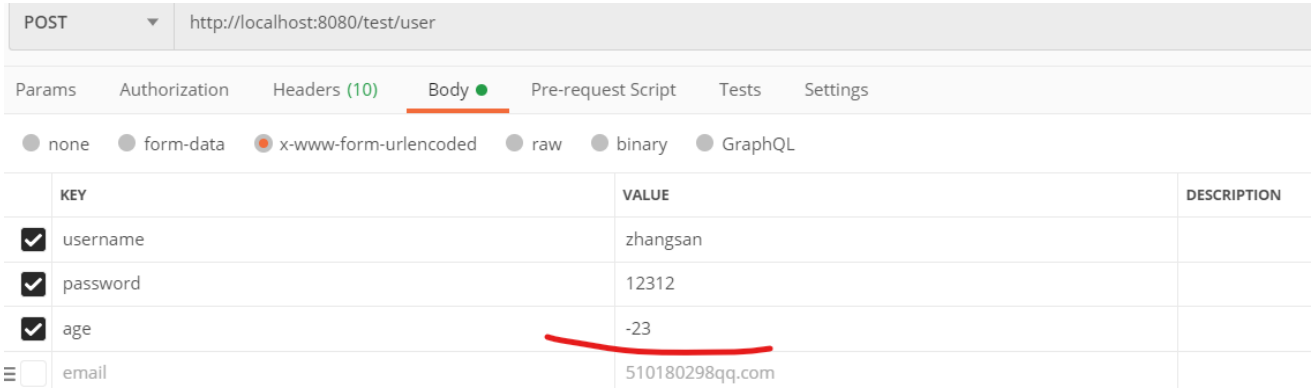
```
<bean id="beacon1024Validator"
class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <property name="providerClass" value="org.hibernate.validator.HibernateValidator"/>
</bean>
<!--注册注解驱动-->
<mvc:annotation-driven validator="beacon1024Validator"/>
```

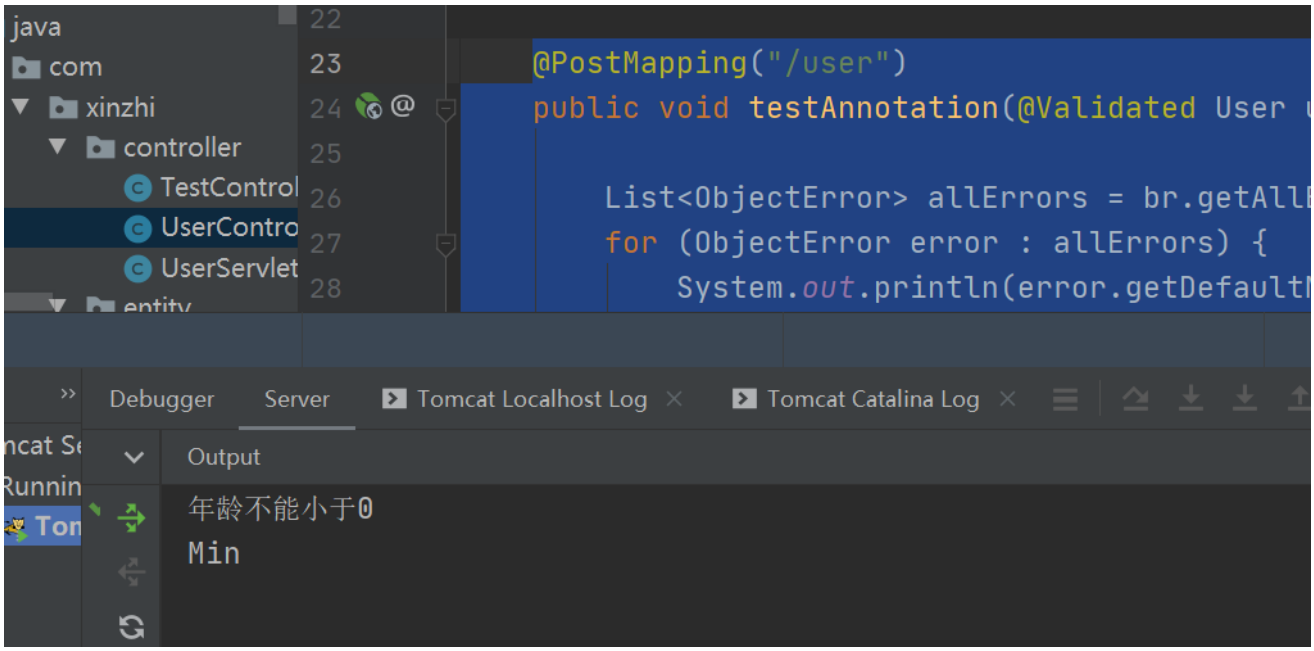
controller使用@Validated标识验证的对象，紧跟着的BindingResult获取错误信息

```
//两个参数之间不能有任何其他的参数
@PostMapping("/user")
public void testAnnotation(@Validated User user, BindingResult br){

    List<ObjectError> allErrors = br.getAllErrors();
    for (ObjectError error : allErrors) {
        System.out.println(error.getDefaultMessage());
        System.out.println(error.getCode());
```

```
        }
        //验证有错误直接返回
        if(allErrors.size() > 0){
            return;
        }

        System.out.print(user);
    }
```

测试





再举一个例子

```
public class User {
    @NotEmpty(message = "姓名不能为空")
    @Size(min = 4,max = 20,message = "姓名长度必须在{min}-{max}之间")
    private String name;

    @Min(value = 0, message = "年龄不能小于{value}")
    @Max(value = 120,message = "年龄不能大于{value}")
    private int age;

    @Pattern(regexp = "^1([358][0-9]|4[579]|66|7[0135678]|9[89])[0-9]{8}$", message =
"手机号码不正确")
    private String phone;
}
```

永远不要相信用户的输入，我们开发的系统凡是涉及到用户输入的地方，都要进行校验，这里的校验分为前台校验和后台校验，前台校验通常由javascript来完成，后台校验主要由java来负责，这里我们可以通过spring mvc+hibernate validator完成。

9、找个博客学习重定向和转发

https://blog.csdn.net/qq_28165595/article/details/76896354

五、restful

自从Roy Fielding博士在2000年他的博士论文中提出REST（Representational State Transfer）风格的软件架构模式后，REST就基本上迅速取代了复杂而笨重的SOAP，成为Web API的标准了。

阮一峰博客：http://www.ruanyifeng.com/blog/2011/09/restful.html

> **restful是一种风格，可以遵循，也可以不遵循，但是正在慢慢变成主流**

1、Rest架构的主要原则

- 网络上的所有事物都被抽象为资源
- 每个资源都有一个唯一的资源标识符
- 同一个资源具有多种表现形式(xml,json等
- 对资源的各种操作不会改变资源标识符
- 所有的操作都是无状态的
- 符合REST原则的架构方式即可称为RESTful

2、什么是Restful

Restful web service是一种常见的rest的应用,是遵守了rest风格的web服务;rest式的web服务是一种ROA(The Resource-Oriented Architecture)(面向资源的架构).

符合REST约束风格和原则的应用程序或设计就是RESTful

从此请求再也不止get和post了

请求url	请求方式	操作
/user/1	get	获取一个id为1的user
/user/1	delete	删除id为1的对象
/user/1	put	更新id为1的对象
/user	post	新增一个对象

> **Spring MVC 对 RESTful应用提供了以下支持**

- 利用@RequestMapping 指定要处理请求的URI模板和HTTP请求的动作类型
- 利用@PathVariable讲URI请求模板中的变量映射到处理方法参数上
- 利用Ajax,在客户端发出PUT、DELETE动作的请求

3、RequestMapping中的rest

```
@RequestMapping(value = "/{id}", method = RequestMethod.GET)
@RequestMapping(value = "/add", method = RequestMethod.POST)
@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
@RequestMapping(value = "/{id}", method = RequestMethod.PUT)
```

当然还有更好用的

```
@GetMapping("/user/{id}")
@PostMapping("/user")
@DeleteMapping("/user/{id}")
@PutMapping("/user/{id}")
```

获取请求url模板中的变量方法

> **定制统一返回类型**

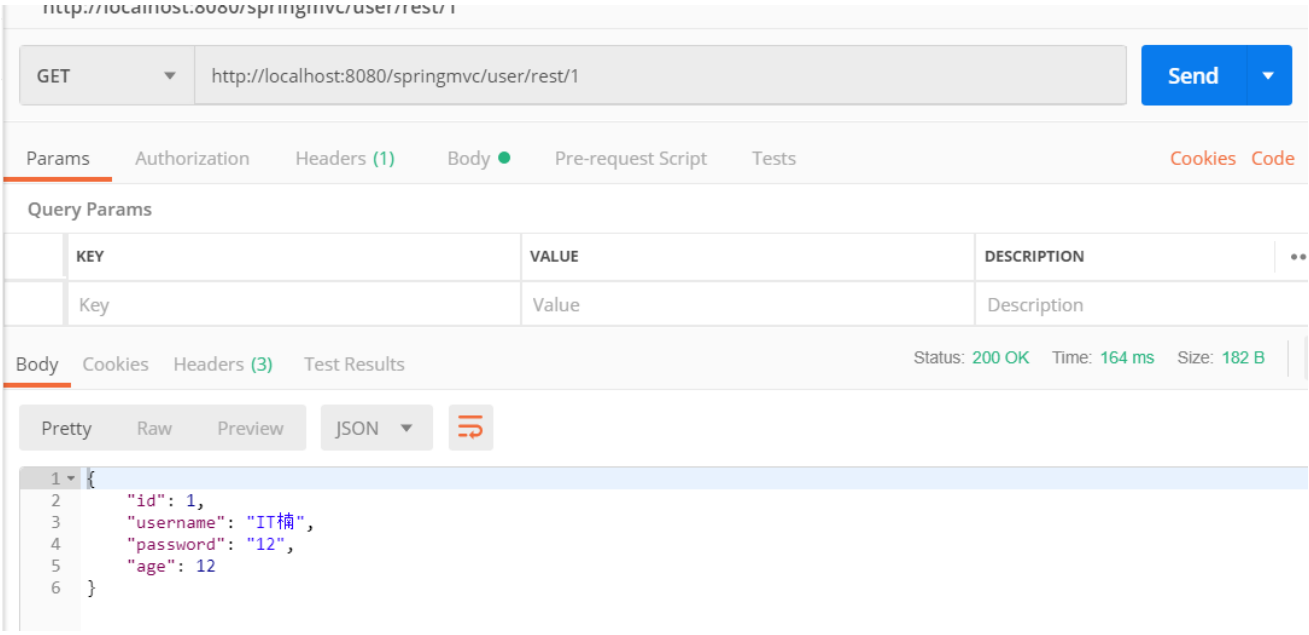
```
package com.xinzhi.util;

import lombok.Data;

import java.io.Serializable;
import java.util.HashMap;
```

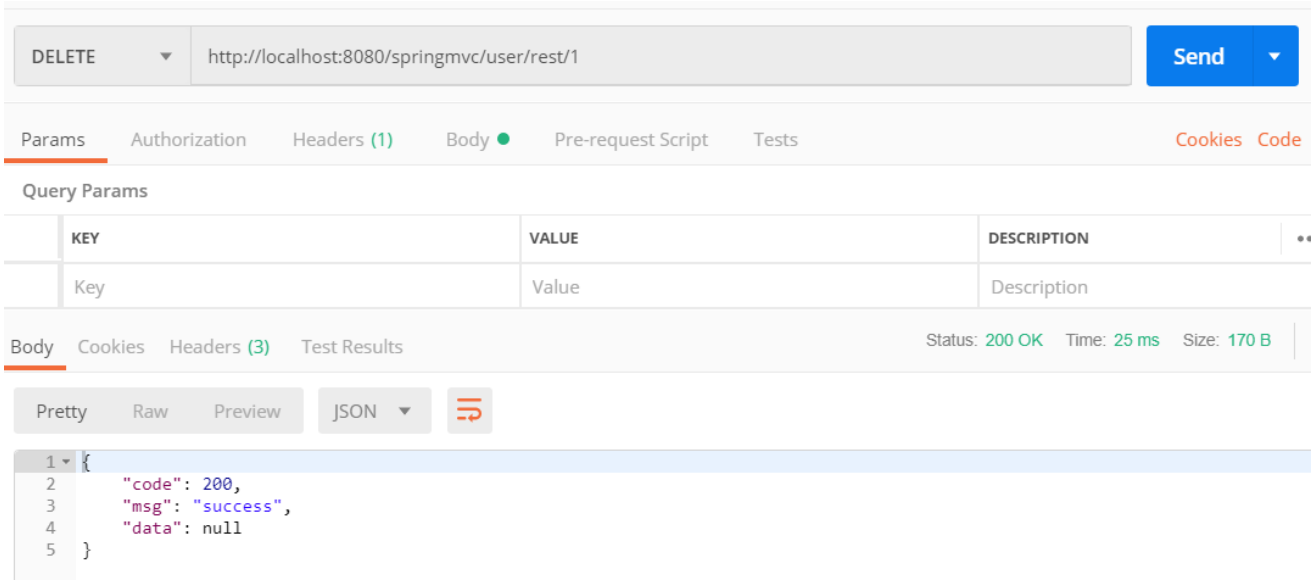
```
import java.util.Map;

/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Data
public class R implements Serializable {

    private int code;
    private String msg;
    private Map<String,Object> data;

    private R(int code, String msg, Map<String, Object> data) {
        this.code = code;
        this.msg = msg;
        this.data = data;
    }

    //成功的返回
    public static R success(){
        return new R(200,"操作成功",null);
    }

    //失败的返回
    public static R fail(){
        return new R(500,"操作失败",null);
    }

    //其他类型的返回
    public static R build(int code,String msg){
        return new R(500,msg,null);
    }


    public R put(String key,Object msg){

        if(this.getData() == null){
            this.setData(new HashMap<>(16));
        }

        this.getData().put(key,msg);
        return this;
    }
}
```

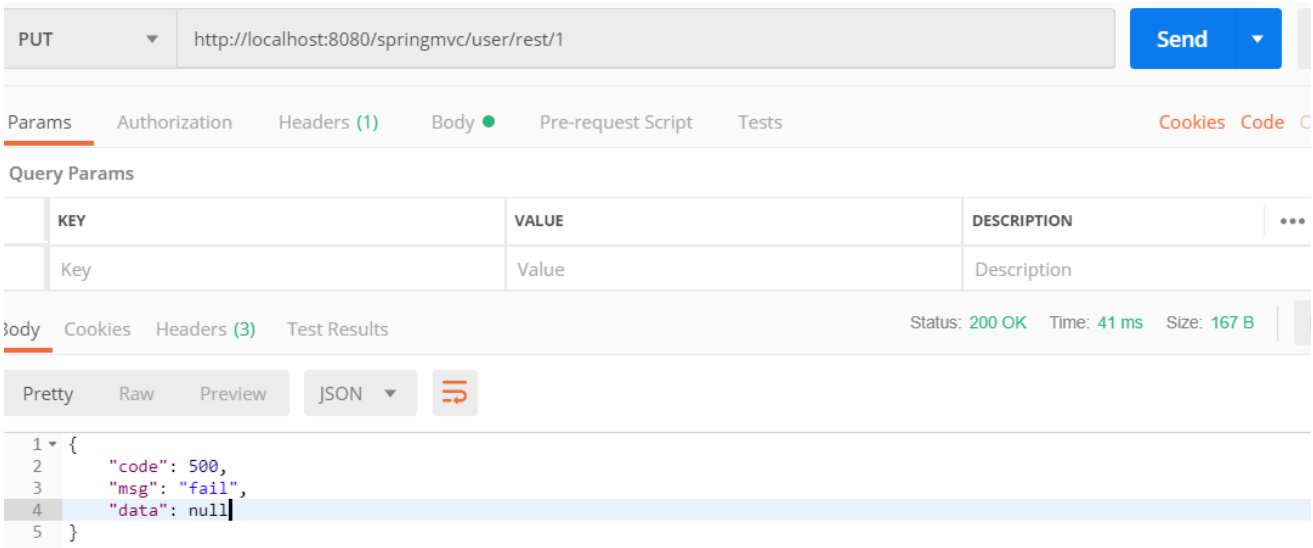
```
@GetMapping("/rest/{id}")
@ResponseBody
public User getUser(@PathVariable int id){
    return  new User(id,"IT楠","12",12);
}
```

```
@DeleteMapping("/rest/{id}")
@ResponseBody
public R deleteUser(@PathVariable int id){
    System.out.println(id);
    //使用service输出
    return  R.success();
}

@PutMapping("/rest/{id}")
@ResponseBody
public R updateUser(@PathVariable int id,User user){
    System.out.println(id);
    System.out.println(user);
    //使用service输出
    return  R.fail();
}
@PostMapping("/rest/add")
@ResponseBody
public R addUser(User user){
    System.out.println(user);
    //使用service输出
    return  R.build(304,"插入发生异常")
        .put("reason ","超时了！ ")
        .put("other","楠哥真帅");
}
```

测试：







4、ajax还能这么玩

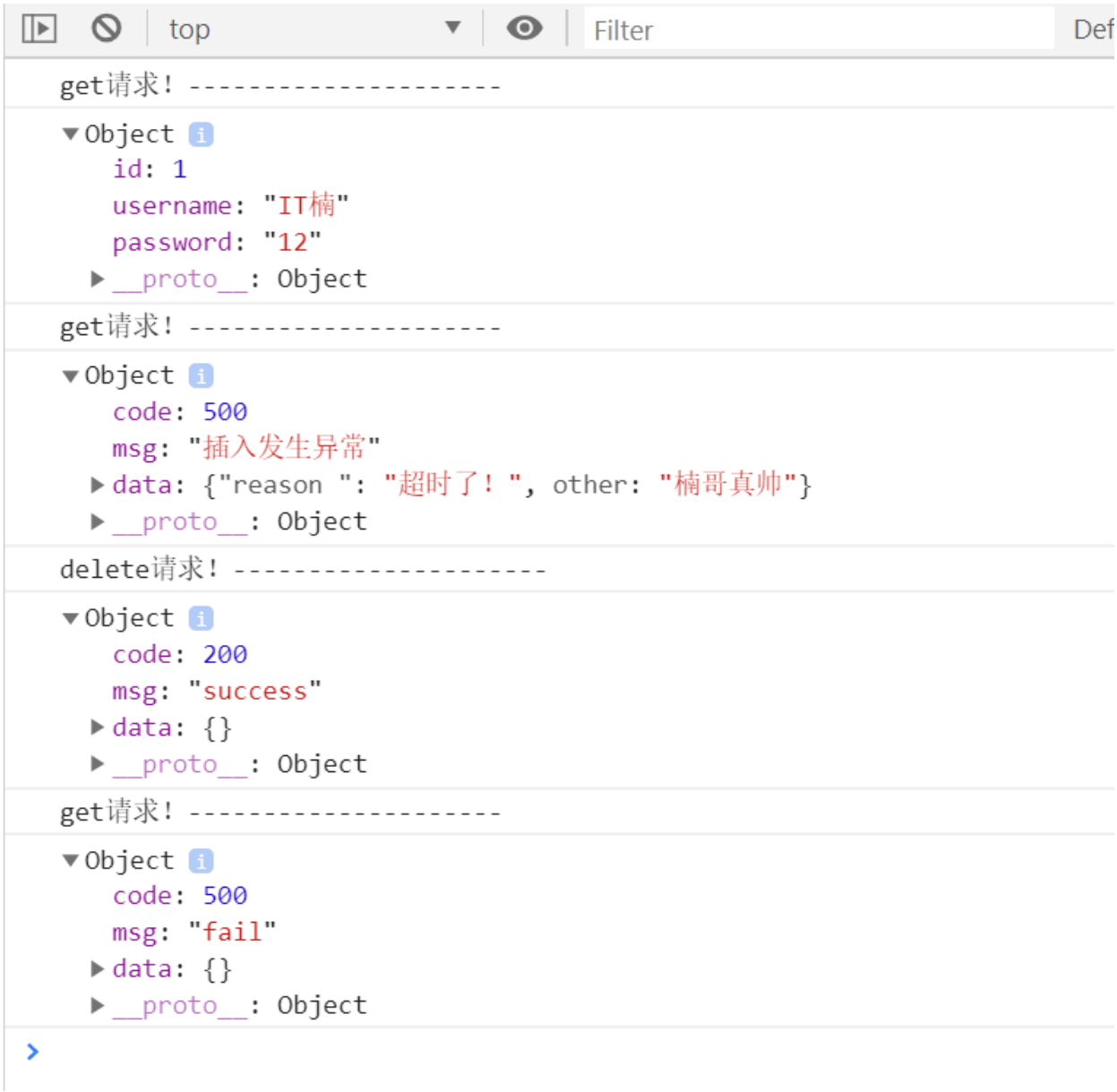
可以采用Ajax方式发送PUT和DELETE请求

深入玩转ajax

```
$.ajax( {
    type : "GET",
    url : "http://localhost:8080/springmvc/user/rest/1",
    dataType : "json",
    success : function(data) {
        console.log("get请求! ----------------------")
        console.log(data)
    }
});

$.ajax( {
```

```
    type : "DELETE",
    url : "http://localhost:8080/springmvc/user/rest/1",
    dataType : "json",
    success : function(data) {
        console.log("delete请求! ----------------------")
        console.log(data)
    }
});

$.ajax( {
    type : "put",
    url : "http://localhost:8080/springmvc/user/rest/1",
    dataType : "json",
    data: {id:12,username:"楠哥",password:"123"},
    success : function(data) {
        console.log("get请求! ----------------------")
        console.log(data)
    }
});

$.ajax( {
    type : "post",
    url : "http://localhost:8080/springmvc/user/rest",
    dataType : "json",
    data: {id:12,username:"楠哥",password:"123"},
    success : function(data) {
        console.log("get请求! ----------------------")
        console.log(data)
    }
});
```

结果测试，成功



六、拦截器

1. SpringMVC提供的拦截器类似于JavaWeb中的过滤器，只不过**SpringMVC拦截器只拦截被前端控制器拦截的请求**，而过滤器拦截从前端发送的任意请求。
2. 熟练掌握`SpringMVC`拦截器对于我们开发非常有帮助，在没使用权限框架(`shiro, spring security`)之前，一般使用拦截器进行认证和授权操作。
3. SpringMVC拦截器有许多应用场景，比如：登录认证拦截器，字符过滤拦截器，日志操作拦截器等等。

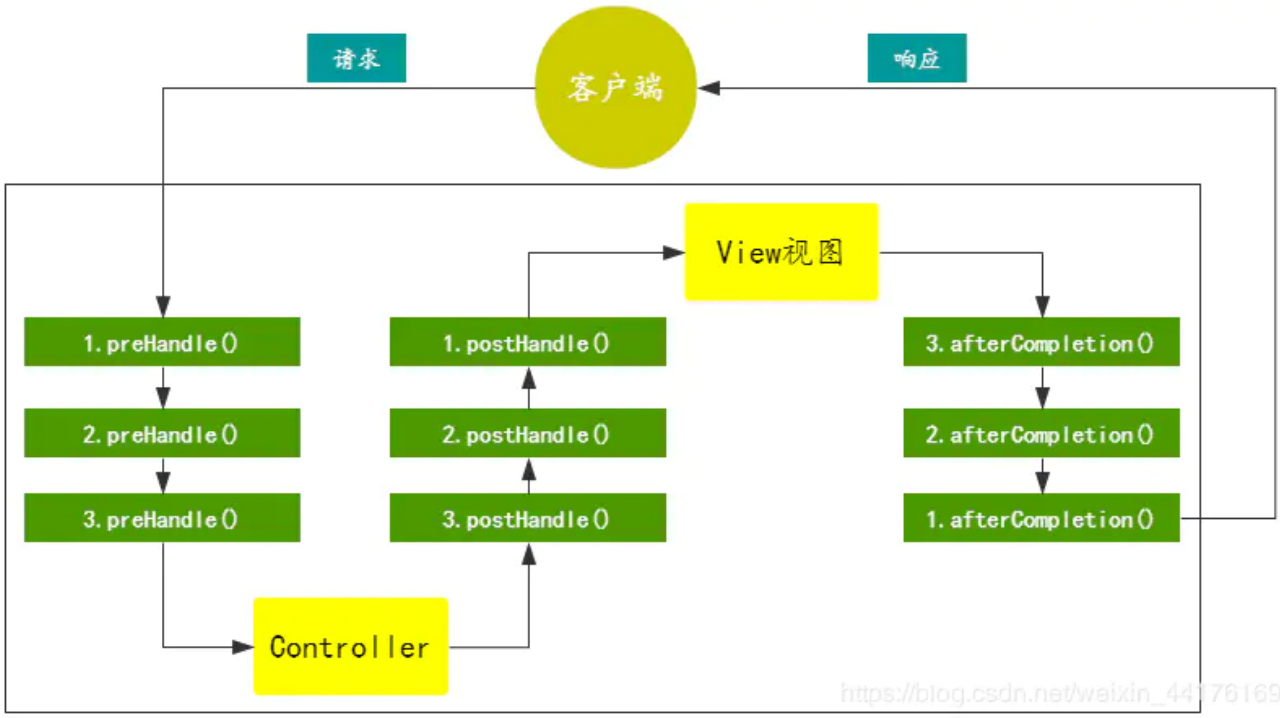
1、自定义拦截器

SpringMVC拦截器的实现一般有两种方式

1. 自定义的`Interceptor`类要实现了Spring的HandlerInterceptor接口。
2. 继承实现了`HandlerInterceptor`接口的类，比如Spring已经提供的实现了HandlerInterceptor接口的抽象类HandlerInterceptorAdapter。

```
public class LoginInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) {
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) {}

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) {}
}
```

2、拦截器拦截流程



在这里插入图片描述

3、拦截器规则

我们可以配置多个拦截器，每个拦截器中都有三个方法。下面将总结多个拦截器中的方法执行规律。

1. preHandle：Controller方法处理请求前执行，根据拦截器定义的顺序，正向执行。
2. postHandle：Controller方法处理请求后执行，根据拦截器定义的顺序，逆向执行。需要所有的preHandle方法都返回true时才会调用。
3. afterCompletion：View视图渲染后处理方法：根据拦截器定义的顺序，逆向执行。preHandle返回true就会调用。

4、登录拦截器

接下来编写一个登录拦截器，这个拦截器可以实现认证操作。就是当我们还没有登录的时候，如果发送请求访问我们系统资源时，拦截器不放行，请求失败。只有登录成功后，拦截器放行，请求成功。登录拦截器只要在preHandle()方法中编写认证逻辑即可，因为是在请求执行前拦截。代码实现如下：

```
/**
 *  登录拦截器
 */
public class LoginInterceptor implements HandlerInterceptor {

    /**
        在执行Controller方法前拦截，判断用户是否已经登录，
        登录了就放行，还没登录就重定向到登录页面
    */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) {
        HttpSession session = request.getSession();
        User user = session.getAttribute("user");
        if (user == null){
            //还没登录，重定向到登录页面
            response.sendRedirect("/toLogin");
        }else {
            //已经登录，放行
            return true;
        }
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) {}

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) {}
}
```

编写完SpringMVC拦截器，我们还需要在springmvc.xml配置文件中，配置我们编写的拦截器，配置代码如下：

1. 配置需要拦截的路径
2. 配置不需要拦截的路径
3. 配置我们自定义的拦截器类

```
<mvc:interceptors>
        <mvc:interceptor>
            <!--
            mvc:mapping：拦截的路径
            /**：是指所有文件夹及其子孙文件夹
            /*：是指所有文件夹，但不包含子孙文件夹
            /：Web项目的根目录
        -->
```

```
            <mvc:mapping path="/**"/>
            <!--
                mvc:exclude-mapping：不拦截的路径,不拦截登录路径
                /toLogin：跳转到登录页面
                /login：登录操作
            -->
            <mvc:exclude-mapping path="/toLogin"/>
            <mvc:exclude-mapping path="/login"/>
            <!--class属性就是我们自定义的拦截器-->
            <bean id="loginInterceptor"
class="com.xinzhi.interceptor.LoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>
```

编写SpringMVC配置类，将自定义拦截器添加到配置中:

> **在springboot中已经全面使用配置类进行配置了**

```
@Configuration
public class WebMvcConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //创建自定义的拦截器
        Interceptor interceptor = new LoginInterceptor();
        //添加拦截器
        registry.addInterceptor(interceptor)
                //添加需要拦截的路径
                .addPathPatterns("");
    }
}
```

七、文件上传

MultipartResolver 用于处理文件上传，当收到请求时 DispatcherServlet 的 checkMultipart() 方法会调用 MultipartResolver 的 isMultipart() 方法判断请求中是否包含文件。如果请求数据中包含文件，则调用 MultipartResolver 的 resolveMultipart() 方法对请求的数据进行解析，然后将文件数据解析成 MultipartFile 并封装在 MultipartHttpServletRequest (继承了 HttpServletRequest) 对象中，最后传递给 Controller。

MultipartResolver 默认不开启，需要手动开启。

前端表单要求：为了能上传文件，必须将表单的method设置为POST，并将enctype设置为multipart/form-data。只有在这样的情况下，浏览器才会把用户选择的文件以二进制数据发送给服务器；

对表单中的 enctype 属性做个详细的说明：

- application/x-www-form-urlencoded：默认方式，只处理表单域中的 value 属性值，采用这种编码方式的表单会将表单域中的值处理成 URL 编码方式。
- multipart/form-data：这种编码方式会以二进制流的方式来处理表单数据，这种编码方式会把文件域指定文件的内容也封装到请求参数中，不会对字符编码。

```
<form action="" enctype="multipart/form-data" method="post">
   <input type="file" name="file"/>
   <input type="submit">
</form>
```

一旦设置了enctype为multipart/form-data，浏览器即会采用二进制流的方式来处理表单数据，而对于文件上传的处理则涉及在服务器端解析原始的HTTP响应。在2003年，Apache Software Foundation发布了开源的Commons FileUpload组件，其很快成为Servlet/JSP程序员上传文件的最佳选择。

文件上传

1、导入文件上传的jar包，commons-fileupload ， Maven会自动帮我们导入他的依赖包 commons-io包；

```
<!--文件上传-->
<dependency>
   <groupId>commons-fileupload</groupId>
   <artifactId>commons-fileupload</artifactId>
   <version>1.3.3</version>
</dependency>
```

2、配置bean：multipartResolver

【注意！！！这个bena的id必须为：multipartResolver ， 否则上传文件会报400的错误！在这里栽过坑,教训！】

```
<!--文件上传配置-->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
   <!-- 请求的编码格式，必须和jSP的pageEncoding属性一致，以便正确读取表单的内容，默认为ISO-8859-1
-->
   <property name="defaultEncoding" value="utf-8"/>
   <!-- 上传文件大小上限，单位为字节（10485760=10M） -->
   <property name="maxUploadSize" value="10485760"/>
   <property name="maxInMemorySize" value="40960"/>
</bean>
```

CommonsMultipartFile 的 常用方法：

- **String getOriginalFilename()：获取上传文件的原名**
- **InputStream getInputStream()：获取文件流**
- **void transferTo(File dest)：将上传文件保存到一个目录文件中**

我们去实际测试一下

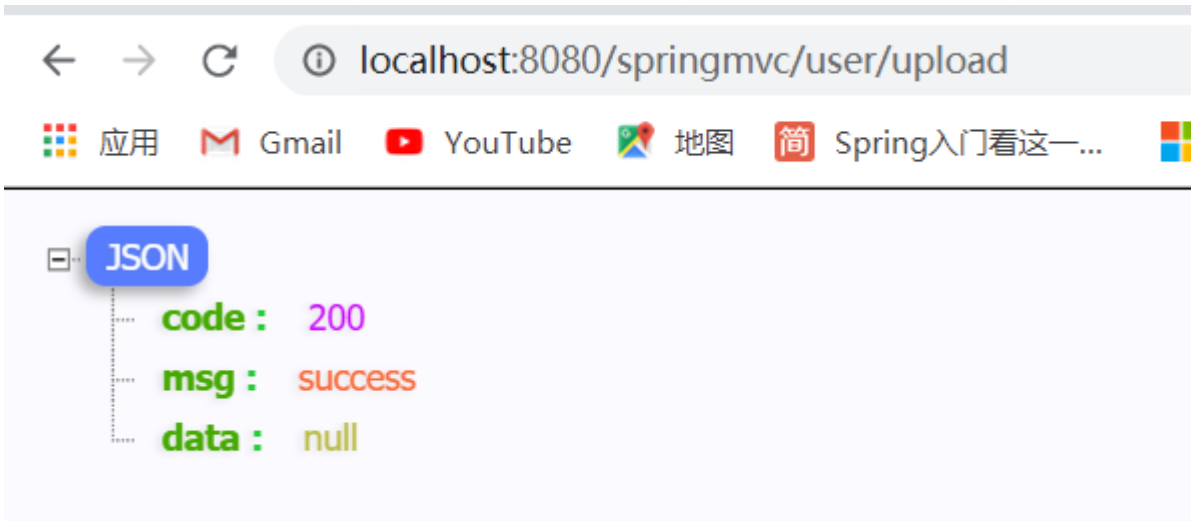
3、编写前端页面

```
<form action="/upload" enctype="multipart/form-data" method="post">
 <input type="file" name="file"/>
 <input type="submit" value="upload">
</form>
```

4、`Controller`

```
@PostMapping("/upload")
@ResponseBody
public R upload(@RequestParam("file") CommonsMultipartFile file, HttpServletRequest
request) throws Exception{
    //获取文件名 : file.getOriginalFilename();
    String uploadFileName = file.getOriginalFilename();
    System.out.println("上传文件名 : "+uploadFileName);

    //上传路径保存设置
    String path = "D:/upload";
    //如果路径不存在，创建一个
    File realPath = new File(path);
    if (!realPath.exists()){
        realPath.mkdir();
    }
    System.out.println("上传文件保存地址："+realPath);
    //就问香不香，就和你写读流一样
    file.transferTo(new File(path+"/"+uploadFileName));

    return R.success();
}
```

5、测试上传文件，OK!



文件下载

第一种可以直接向response的输出流中写入对应的文件流

第二种可以使用 ResponseEntity<byte[]>来向前端返回文件

一、传统方式

```
@GetMapping("/download1")
    @ResponseBody
    public R download1(HttpServletResponse response){
```

```
        FileInputStream fileInputStream = null;
        ServletOutputStream outputStream = null;
        try {
            //以文件形式下载

            String  fileName = "楠老师.jpg";

            //1、设置response 响应头，处理中文名字乱码问题
            response.reset(); //设置页面不缓存,清空buffer
            response.setCharacterEncoding("UTF-8"); //字符编码
            response.setContentType("multipart/form-data"); //二进制传输数据
            //设置响应头
            response.setHeader("Content-Disposition",
                    "attachment;fileName="+ URLEncoder.encode(fileName, "UTF-8"));


            File file = new File("D:/upload/"+fileName);
            fileInputStream = new FileInputStream(file);
            outputStream = response.getOutputStream();

            byte[] buffer = new byte[1024];
            int len;
            while ((len = fileInputStream.read(buffer)) != -1){
                outputStream.write(buffer,0,len);
                outputStream.flush();
            }

            return R.success();
        } catch (IOException e) {
            e.printStackTrace();
            return R.fail();
        }finally {
            if( fileInputStream != null ){
                try {
                    fileInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if( outputStream != null ){
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

推荐使用这种方式，这种方式可以以json形式给前台返回提示信息。

二、使用ResponseEntity

```
@GetMapping("/download2")
public ResponseEntity<byte[]> download2(){
    try {
        String fileName = "楠老师.jpg";
        byte[] bytes = FileUtils.readFileToByteArray(new File("D:/upload/"+fileName));
        HttpHeaders headers=new HttpHeaders();
        headers.set("Content-Disposition","attachment;filename="+
URLEncoder.encode(fileName, "UTF-8"));
        headers.set("charsetEncoding","utf-8");
        headers.set("content-type","multipart/form-data");
        ResponseEntity<byte[]> entity=new ResponseEntity<>(bytes,headers,
HttpStatus.OK);
        return entity;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

八、拓展学习

1、全局异常捕获

Spring MVC的Controller出现异常的默认处理是响应一个500状态码，再把错误信息显示在页面上，如果用户看到这样的页面，一定会觉得你这个网站太LOW了。 要解决Controller的异常问题，当然也不能在每个处理请求的方法中加上异常处理，那样太繁琐。

Spring MVC提供了一个`HandlerExceptionResolver`接口，可用于统一异常处理。

```
/**
 * @author IT楠老师
 * @date 2020/6/3
 */
@Component
public class MyExceptionResolver implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, Object o, Exception e) {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("error");
        return mv;
    }
}
```

2、作业、有兴趣的自行学习国际化

博客：https://www.cnblogs.com/ya-qiang/p/9388238.html

九、整合数据库，ssm结束

> **其实就是spring整合mybatis，咱们尽量使用注解完成工作**

1、完整的依赖

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

    <packaging>war</packaging>
    <groupId>org.example</groupId>
    <artifactId>springmvc-test</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- 测试相关 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <!-- springmvc -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.2.6.RELEASE</version>
        </dependency>
        <!-- servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.2</version>
            <scope>provided</scope>
        </dependency>

        <!--文件上传-->
        <dependency>
            <groupId>commons-fileupload</groupId>
            <artifactId>commons-fileupload</artifactId>
```

```
            <version>1.4</version>
        </dependency>
        <!-- fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.68</version>
        </dependency>

        <!-- mybatis 相关 -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.2</version>
        </dependency>
        <!-- 数据库连接驱动 相关 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>

        <!-- 提供了对JDBC操作的完整封装 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.1.10.RELEASE</version>
        </dependency>
        <!-- 织入 相关 -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.4</version>
        </dependency>
        <!-- spring, mybatis整合包 -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>2.0.2</version>
        </dependency>
        <!-- 集成德鲁伊使用 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.18</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- 配置jdk -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
        <!-- 资源路径 -->
        <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
        </resources>
    </build>
</project>
```

2、mybatis的配置文件

`mybatis-config.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

</configuration>
```

3、springmvc配置文件

`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
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        https://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- 自动扫包 -->
    <context:component-scan base-package="com.xinzhi"/>
    <!-- 让Spring MVC不处理静态资源 -->
    <mvc:default-servlet-handler />
    <!--  让springmvc自带的注解生效  -->
    <mvc:annotation-driven >


    </mvc:annotation-driven>
    <bean id="fastjson"
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">
        <property name="supportedMediaTypes">
            <list>
                <value>text/html;charset=UTF-8</value>
                <value>application/json;charset=UTF-8</value>
            </list>
        </property>
    </bean>

    <!--文件上传配置-->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <!-- 请求的编码格式，必须和jSP的pageEncoding属性一致，以便正确读取表单的内容，默认为ISO-
8859-1 -->
        <property name="defaultEncoding" value="utf-8"/>
        <!-- 上传文件大小上限，单位为字节（10485760=10M） -->
        <property name="maxUploadSize" value="10485760"/>
        <property name="maxInMemorySize" value="40960"/>
    </bean>

    <!-- 视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    id="internalResourceViewResolver">
        <!-- 前缀 -->
        <property name="prefix" value="/WEB-INF/page/" />
        <!-- 后缀 -->
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

4、数据源配置

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?
useSSL=true&useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

5、spring配置文件

application.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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 加载外部的数据库信息 classpath:不叫会报错具体原因下边解释-->
    <context:property-placeholder location="classpath:db.properties"/>
    <!-- 加入springmvc的配置 -->
    <import resource="classpath:springmvc-servlet.xml"/>

    <!-- Mapper 扫描器 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!-- 扫描 cn.wmyskxz.mapper 包下的组件 -->
        <property name="basePackage" value="com.xinzhi.dao"/>
    </bean>

    <!--配置数据源：数据源有非常多，可以使用第三方的，也可使使用Spring的-->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!--配置SqlSessionFactory-->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <!--关联Mybatis-->
        <property name="configLocation" value="classpath:mybatis-config.xml"/>
```

```
        <property name="mapperLocations" value="classpath:mappers/*.xml"/>
    </bean>

    <!--注册sqlSessionTemplate , 关联sqlSessionFactory-->
    <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
        <!--利用构造器注入-->
        <constructor-arg index="0" ref="sqlSessionFactory"/>
    </bean>

    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>


    <!--配置事务通知-->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!--配置哪些方法使用什么样的事务,配置事务的传播特性-->
            <tx:method name="add*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="up@date*" propagation="REQUIRED"/>
            <tx:method name="search*" propagation="REQUIRED"/>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>
    <!--配置aop织入事务-->
    <aop:config>
        <aop:pointcut id="txPointcut" expression="execution(*
com.xinzhi.service.impl.*.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
    </aop:config>

</beans>
```

6、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_4_0.xsd"
         version="4.0">

    <!--注册DispatcherServlet，这是springmvc的核心，就是个servlet-->
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:application.xml</param-value>
        </init-param>
        <!--加载时先启动-->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <!--/ 匹配所有的请求；（不包括.jsp）-->
    <!--/* 匹配所有的请求；（包括.jsp）-->
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Maven项目，application-context.xml、db.properties文件均放置在src/main/resources目录下，Tomcat部署项目，src/main/resources目录下的配置文件默认位置为：{项目名}/WEB-INF/classes，而Spring却在项目根目录下寻找，肯定找不到，因此，配置时指定classpath目录下寻找即可。

解决方案如下：

```
<context:property-placeholder location="**classpath:db.properties**" />
```

配置文件完

7、编写entity实体类

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable{

    private int id;
    private String username;
    private String password;
}
```

8、UserDao接口编写

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
```

```java
@Mapper
public interface UserMapper {
    /**
     * 根据id查找用户
     * @param id
     * @return
     */
    User findUserById(int id);

    /**
     * 获取所有的用户
     * @return
     */
    List<User> findAllUsers();
}
```

9、Mapper映射文件

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.xinzhi.dao.UserMapper">
    <select id="findUserById" resultType="com.xinzhi.entity.User" parameterType="int">
      select id,username,password from user where id = #{id}
    </select>

    <select id="findAllUsers" resultType="com.xinzhi.entity.User">
      select id,username,password from user
    </select>
</mapper>
```

10、编写service

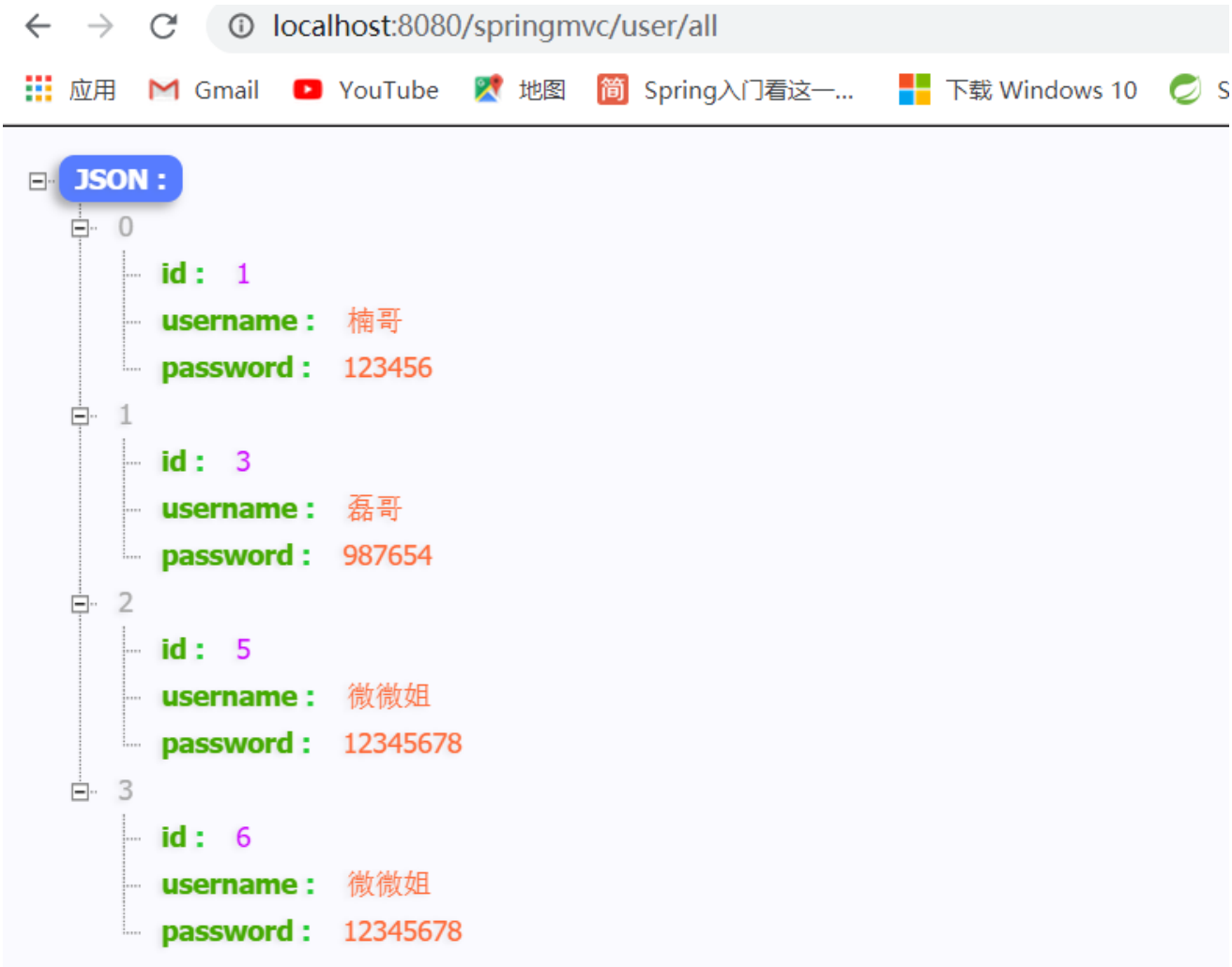
```java
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
public interface IUserService {
    /**
     * 获取一个用户的信息
     * @param id
     * @return
     */
    User getUserInfo(int id);

    /**
     * 获取所有用户信息
     * @return
     */
    List<User> getAllUsers();
```

```
}

/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Service
public class UserServiceImpl implements IUserService {

    @Resource
    private UserMapper userMapper;

    @Override
    public User getUserInfo(int id) {
        return userMapper.findUserById(id);
    }

    @Override
    public List<User> getAllUsers() {
        return userMapper.findAllUsers();
    }
}
```

11、编写controller

```
/**
 * @author IT楠老师
 * @date 2020/5/19
 */
@Controller
@RequestMapping("/user/")
public class UserController {

    @Resource
    IUserService userService;

    @GetMapping("{id}")
    @ResponseBody
    public User getUserInfo(@PathVariable int id){
        return userService.getUserInfo(id);
    }

    @GetMapping("/all")
    @ResponseBody
    public List<User> getUserInfo(){
        return userService.getAllUsers();
    }

}
```

12、测试



13、集成一个德鲁伊

(1) 更换数据源

```
<!--配置数据源：数据源有非常多，可以使用第三方的，也可使使用Spring的-->
```

```
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" destroy-
method="close">
    <property name="driverClassName" value="${jdbc.driver}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>

    <property name = "filters" value = "${filters}" />
    <!-- 最大并发连接数 -->
    <property name = "maxActive" value = "${maxActive}" />
    <!-- 初始化连接数量 -->
    <property name = "initialSize" value = "${initialSize}" />
    <!-- 配置获取连接等待超时的时间 -->
    <property name = "maxWait" value = "${maxWait}" />
    <!-- 最小空闲连接数 -->
    <property name = "minIdle" value = "${minIdle}" />
    <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
    <property name = "timeBetweenEvictionRunsMillis" value
="${timeBetweenEvictionRunsMillis}" />
    <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
    <property name = "minEvictableIdleTimeMillis" value
="${minEvictableIdleTimeMillis}" />
    <!--        <property name = "validationQuery" value = "${validationQuery}" />
-->
    <property name = "testWhileIdle" value = "${testWhileIdle}" />
    <property name = "testOnBorrow" value = "${testOnBorrow}" />
    <property name = "testOnReturn" value = "${testOnReturn}" />
    <property name = "maxOpenPreparedStatements" value ="${maxOpenPreparedStatements}"
/>
    <!-- 打开 removeAbandoned 功能 -->
    <property name = "removeAbandoned" value = "${removeAbandoned}" />
    <!-- 1800 秒，也就是 30 分钟 -->
    <property name = "removeAbandonedTimeout" value ="${removeAbandonedTimeout}" />
    <!-- 关闭 abanded 连接时输出错误日志 -->
    <property name = "logAbandoned" value = "${logAbandoned}" />
</bean>
```

(2) 增加`db.properties`内容

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?
useSSL=true&useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root

filters=wall,stat
maxActive=20
initialSize=3
maxWait=5000
minIdle=3
maxIdle=15
timeBetweenEvictionRunsMillis=60000
minEvictableIdleTimeMillis=300000
```

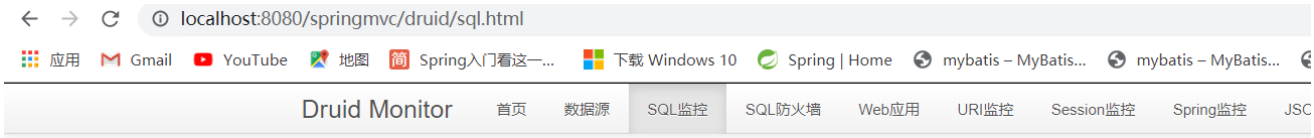
```
validationQuery=SELECT 'x'
testWhileIdle=true
testOnBorrow=false
testOnReturn=false
maxOpenPreparedStatements=20
removeAbandoned=true
removeAbandonedTimeout=1800
logAbandoned=true
```

(3) 开启web监控

在web.xml中启动web服务

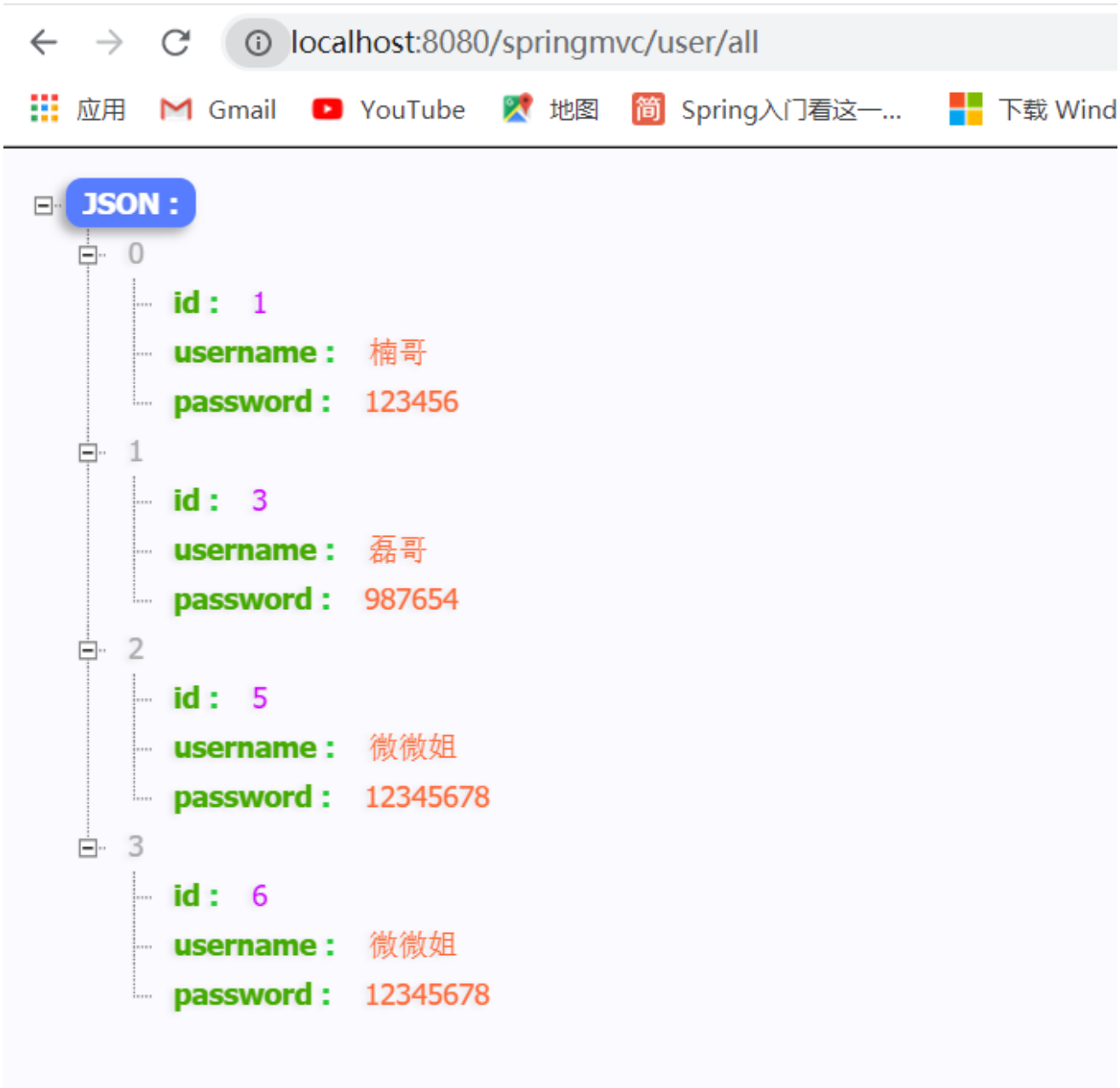
```
<!-- 连接池 启用 Web 监控统计功能    start-->
<filter>
    <filter-name>DruidWebStatFilter</filter-name>
    <filter-class>com.alibaba.druid.support.http.WebStatFilter</filter-class>
    <init-param>
        <param-name>exclusions</param-name>
        <param-value>*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>DruidWebStatFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<servlet>
    <servlet-name>DruidStatView </servlet-name>
    <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>
    <init-param>
        <!-- 用户名 -->
        <param-name>loginUsername</param-name>
        <param-value>xinzhi</param-value>
    </init-param>
    <init-param>
        <!-- 密码 -->
        <param-name>loginPassword</param-name>
        <param-value>123</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>DruidStatView</servlet-name>
    <url-pattern>/druid/*</url-pattern>
</servlet-mapping>
```

(4) 测试，成功。



SQL Stat View JSON API

N	SQL▼	执行数	执行时间	最慢	事务执行	错误数	更新行数	读取行数	执行中	最大并发
1	select id,username,passwo...	1	3	3	1			4		1



14、集成日志框架

（1）引入依赖

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
</dependency>
```

（2）新建配置文件

在类路径下（src/main/resources）新建一个logback.xml文件 这里贴出一个模板，下面会有解释

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="60 seconds" debug="false">
    <!-- 定义参数常量 -->
    <!-- 日志级别 TRACE<DEBUG<INFO<WARN<ERROR -->
    <!-- logger.trace("msg") logger.debug... -->
    <property name="log.level" value="debug" />
    <property name="log.maxHistory" value="30" />
    <property name="log.filePath" value="D:/log" />
    <property name="log.pattern"
              value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} -
%msg%n" />
    <!-- 控制台输出设置 -->
    <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
    </appender>
    <!-- DEBUG级别文件记录 -->
    <appender name="debugAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 文件路径 -->
        <file>${log.filePath}/debug.log</file>
        <!-- 滚动日志文件类型，就是每天都会有一个日志文件 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 文件名称 -->
            <fileNamePattern>${log.filePath}/debug/debug.%d{yyyy-MM-dd}.log.gz
            </fileNamePattern>
            <!-- 文件最大保存历史数量 -->
            <maxHistory>${log.maxHistory}</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>DEBUG</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>
    <!-- INFO -->
    <appender name="infoAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
```

```
        <!-- 文件路径 -->
        <file>${log.filePath}/info.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 文件名称 -->
            <fileNamePattern>${log.filePath}/info/info.%d{yyyy-MM-dd}.log.gz
            </fileNamePattern>
            <!-- 文件最大保存历史数量 -->
            <maxHistory>${log.maxHistory}</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>INFO</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!-- com.xinzhi开头的日志对应形式 -->
    <logger name="com.xinzhi" level="${log.level}" additivity="true">
        <appender-ref ref="debugAppender"/>
        <appender-ref ref="infoAppender"/>
    </logger>

    <!-- <root> 是必选节点，用来指定最基础的日志输出级别，只有一个level属性 -->
    <root level="info">
        <appender-ref ref="consoleAppender"/>
    </root>

    <!-- 捕捉sql开头的日志 -->
    <appender name="MyBatis" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.filePath}/sql_log/mybatis-sql.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <FileNamePattern>${log.filePath}/sql_log/mybatis-sql.log.%d{yyyy-MM-dd}
</FileNamePattern>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%thread|%d{yyyy-MM-dd
HH:mm:ss.SSS}|%level|%logger{36}|%m%n</pattern>
        </encoder>
    </appender>

    <logger name="sql" level="DEBUG">
        <appender-ref ref="MyBatis"/>
    </logger>

</configuration>
```

mybatis配置文件

```
<settings>
    <setting name="logPrefix" value="sql."/>
</settings>
```

（3）使用

> **注意引入的包必须是org.slf4j.Logger和org.slf4j.LoggerFactory**

必须定义一个log变量才能打log，参数就填本类的class，这样打印日志才能准确定位啊

```
 private final Logger log = LoggerFactory.getLogger(UserController.class);
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UserController {

    //定义一个log
    private final Logger log = LoggerFactory.getLogger(UserController.class);

    ....
    public void ....
}


//在方法中合理使用log，使用哪个级别看这个日志的重要性
@GetMapping(value = "{id}")
@ResponseBody
public User getUserInfo(@PathVariable Integer id){
    log.info("info");
    log.trace("trace");
    log.debug("debug");
    log.warn("warn");
    log.error("error");

    return userService.getUserInfo(id);
}
```

（4）使用lombok

在类上加注解：

```
@Slf4j
public class UserController {}
```

会在编译的时候自动加上

```
private final Logger log = LoggerFactory.getLogger(UserController.class);
```

所以这句话就不用写了。

结束

> **B站：IT楠老师 公众号：IT楠说jiava QQ群： 1083478826 新知大数据**

制作不易、如果觉的好不妨打个赏：

九、全部的配置文件：

1、pom文件

```
<?xml version="1.0" encoding="UTF-8"?>
<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>ssm</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>
```

```xml
<dependencies>
    <!-- 测试相关 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
    <!-- springmvc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.6.RELEASE</version>
    </dependency>
    <!-- servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.2</version>
        <scope>provided</scope>
    </dependency>

    <!--文件上传-->
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.4</version>
    </dependency>
    <!-- fastjson -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.68</version>
    </dependency>

    <!-- mybatis 相关 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.2</version>
    </dependency>
    <!-- 数据库连接驱动 相关 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.47</version>
    </dependency>
```

```
<!-- 提供了对JDBC操作的完整封装 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.1.10.RELEASE</version>
</dependency>
<!-- 织入 相关 -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.4</version>
</dependency>
<!-- spring, mybatis整合包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.2</version>
</dependency>
<!-- 集成德鲁伊使用 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.18</version>
</dependency>

<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.18</version>
</dependency>

<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>${java.version}</source> <!-- 源代码使用的JDK版本 -->
                    <target>${java.version}</target> <!-- 需要生成的目标class文件的编译版本
-->
                    <encoding>${project.build.sourceEncoding}</encoding><!-- 字符集编码 -
->
                </configuration>
            </plugin>
        </plugins>
        <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                </includes>
                <filtering>false</filtering>
            </resource>
        </resources>
    </build>
</project>
```

2、mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="lazyLoadingEnabled" value="true"/>
        <setting name="aggressiveLazyLoading" value="false"/>
        <!-- 下划线转驼峰式 -->
        <setting name="cacheEnabled" value="true"/>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
```

```
        <setting name="logPrefix" value="sql."/>
    </settings>


</configuration>
```

3、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
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- 让Spring MVC不处理静态资源 -->
    <mvc:default-servlet-handler />
    <!--  让springmvc自带的注解生效  -->
    <mvc:annotation-driven >
        <mvc:message-converters>
            <bean id="fastjson"
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">
                <property name="supportedMediaTypes">
                    <list>
                        <value>text/html;charset=UTF-8</value>
                        <value>application/json;charset=UTF-8</value>
                    </list>
                </property>
            </bean>
        </mvc:message-converters>
    </mvc:annotation-driven>


    <!--文件上传配置-->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <!-- 请求的编码格式，必须和jSP的pageEncoding属性一致，以便正确读取表单的内容，默认为ISO-
8859-1 -->
        <property name="defaultEncoding" value="utf-8"/>
        <!-- 上传文件大小上限，单位为字节（10485760=10M） -->
        <property name="maxUploadSize" value="10485760"/>
        <property name="maxInMemorySize" value="40960"/>
    </bean>

    <!-- 处理映射器 -->
    <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
    <!-- 处理器适配器 -->
```

```
    <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>
    <!--视图解析器:DispatcherServlet给他的ModelAndView-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="InternalResourceViewResolver">
        <!--前缀-->
        <property name="prefix" value="/WEB-INF/page/"/>
        <!--后缀-->
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

4、db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?
useSSL=true&useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root

filters=wall,stat
maxActive=20
initialSize=3
maxWait=5000
minIdle=3
maxIdle=15
timeBetweenEvictionRunsMillis=60000
minEvictableIdleTimeMillis=300000
validationQuery=SELECT 'x'
testWhileIdle=true
testOnBorrow=false
testOnReturn=false
maxOpenPreparedStatements=20
removeAbandoned=true
removeAbandonedTimeout=1800
logAbandoned=true
```

5、application.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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd
```

```
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 加载外部的数据库信息 classpath:不叫会报错具体原因下边解释-->
    <context:property-placeholder location="classpath:db.properties"/>
    <!-- 加入springmvc的配置 -->
    <import resource="classpath:springmvc-servlet.xml"/>

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

    <!-- Mapper 扫描器 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!-- 扫描 cn.wmyskxz.mapper 包下的组件 -->
        <property name="basePackage" value="com.xinzhi.dao"/>
    </bean>

    <!--配置数据源：数据源有非常多，可以使用第三方的，也可使使用Spring的-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>

        <property name = "filters" value = "${filters}" />
        <!-- 最大并发连接数 -->
        <property name = "maxActive" value = "${maxActive}" />
        <!-- 初始化连接数量 -->
        <property name = "initialSize" value = "${initialSize}" />
        <!-- 配置获取连接等待超时的时间 -->
        <property name = "maxWait" value = "${maxWait}" />
        <!-- 最小空闲连接数 -->
        <property name = "minIdle" value = "${minIdle}" />
        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name = "timeBetweenEvictionRunsMillis" value
="${timeBetweenEvictionRunsMillis}" />
        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name = "minEvictableIdleTimeMillis" value
="${minEvictableIdleTimeMillis}" />
        <!--        <property name = "validationQuery" value = "${validationQuery}" />
    -->
        <property name = "testWhileIdle" value = "${testWhileIdle}" />
        <property name = "testOnBorrow" value = "${testOnBorrow}" />
        <property name = "testOnReturn" value = "${testOnReturn}" />
        <property name = "maxOpenPreparedStatements" value
="${maxOpenPreparedStatements}" />
        <!-- 打开 removeAbandoned 功能 -->
        <property name = "removeAbandoned" value = "${removeAbandoned}" />
        <!-- 1800 秒，也就是 30 分钟 -->
        <property name = "removeAbandonedTimeout" value ="${removeAbandonedTimeout}" />
        <!-- 关闭 abanded 连接时输出错误日志 -->
        <property name = "logAbandoned" value = "${logAbandoned}" />
    </bean>
```

```
    <!--配置SqlSessionFactory-->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <!--关联Mybatis-->
        <property name="configLocation" value="classpath:mybatis-config.xml"/>
        <property name="mapperLocations" value="classpath:mappers/*.xml"/>
    </bean>

    <!--注册sqlSessionTemplate , 关联sqlSessionFactory-->
    <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
        <!--利用构造器注入-->
        <constructor-arg index="0" ref="sqlSessionFactory"/>
    </bean>

    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>


    <!--配置事务通知-->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!--配置哪些方法使用什么样的事务,配置事务的传播特性-->
            <tx:method name="add*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="search*" propagation="REQUIRED"/>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>
    <!--配置aop织入事务-->
    <aop:config>
        <aop:pointcut id="txPointcut" expression="execution(*
com.xinzhi.service.impl.*.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
    </aop:config>

</beans>
```

6、logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="60 seconds" debug="false">
    <!-- 定义参数常量 -->
    <!-- 日志级别 TRACE<DEBUG<INFO<WARN<ERROR -->
    <!-- logger.trace("msg") logger.debug... -->
    <property name="log.level" value="debug" />
    <property name="log.maxHistory" value="30" />
    <property name="log.filePath" value="D:/log" />
```

```
    <property name="log.pattern"
              value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} -
%msg%n" />
    <!-- 控制台输出设置 -->
    <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
    </appender>
    <!-- DEBUG级别文件记录 -->
    <appender name="debugAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 文件路径 -->
        <file>${log.filePath}/debug.log</file>
        <!-- 滚动日志文件类型，就是每天都会有一个日志文件 -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 文件名称 -->
            <fileNamePattern>${log.filePath}/debug/debug.%d{yyyy-MM-dd}.log.gz
            </fileNamePattern>
            <!-- 文件最大保存历史数量 -->
            <maxHistory>${log.maxHistory}</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>DEBUG</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>
    <!-- INFO -->
    <appender name="infoAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 文件路径 -->
        <file>${log.filePath}/info.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 文件名称 -->
            <fileNamePattern>${log.filePath}/info/info.%d{yyyy-MM-dd}.log.gz
            </fileNamePattern>
            <!-- 文件最大保存历史数量 -->
            <maxHistory>${log.maxHistory}</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>INFO</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>
```

```
    <!-- com.xinzhi开头的日志对应形式 -->
    <logger name="com.xinzhi" level="${log.level}" additivity="true">
        <appender-ref ref="debugAppender"/>
        <appender-ref ref="infoAppender"/>
    </logger>

    <!-- <root> 是必选节点，用来指定最基础的日志输出级别，只有一个level属性 -->
    <root level="info">
        <appender-ref ref="consoleAppender"/>
    </root>

    <!-- 捕捉sql开头的日志 -->
    <appender name="MyBatis" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.filePath}/sql_log/mybatis-sql.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <FileNamePattern>${log.filePath}/sql_log/mybatis-sql.log.%d{yyyy-MM-dd}
</FileNamePattern>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%thread|%d{yyyy-MM-dd
HH:mm:ss.SSS}|%level|%logger{36}|%m%n</pattern>
        </encoder>
    </appender>

    <logger name="sql" level="DEBUG">
        <appender-ref ref="MyBatis"/>
    </logger>

</configuration>
```

全剧终：

> **B站：IT楠老师 公众号：IT楠说jiava QQ群： 1083478826 新知大数据**

制作不易、如果觉的好不妨打个赏：

推荐使用**微信支付**

Alm张楠(*楠)

微信支付