JAVA 达摩班

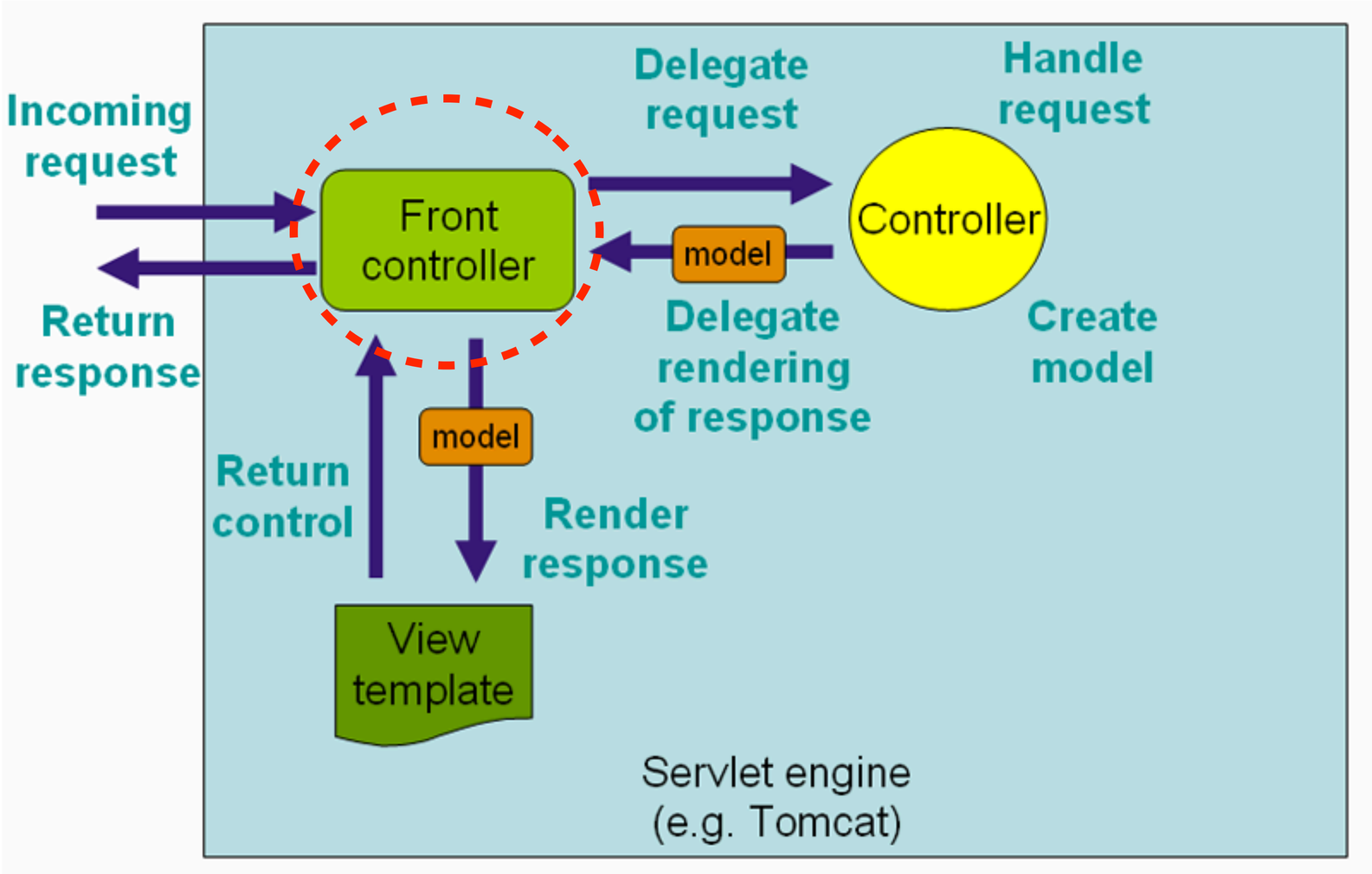
Spring MVC

Spring MVC 核心概念 1

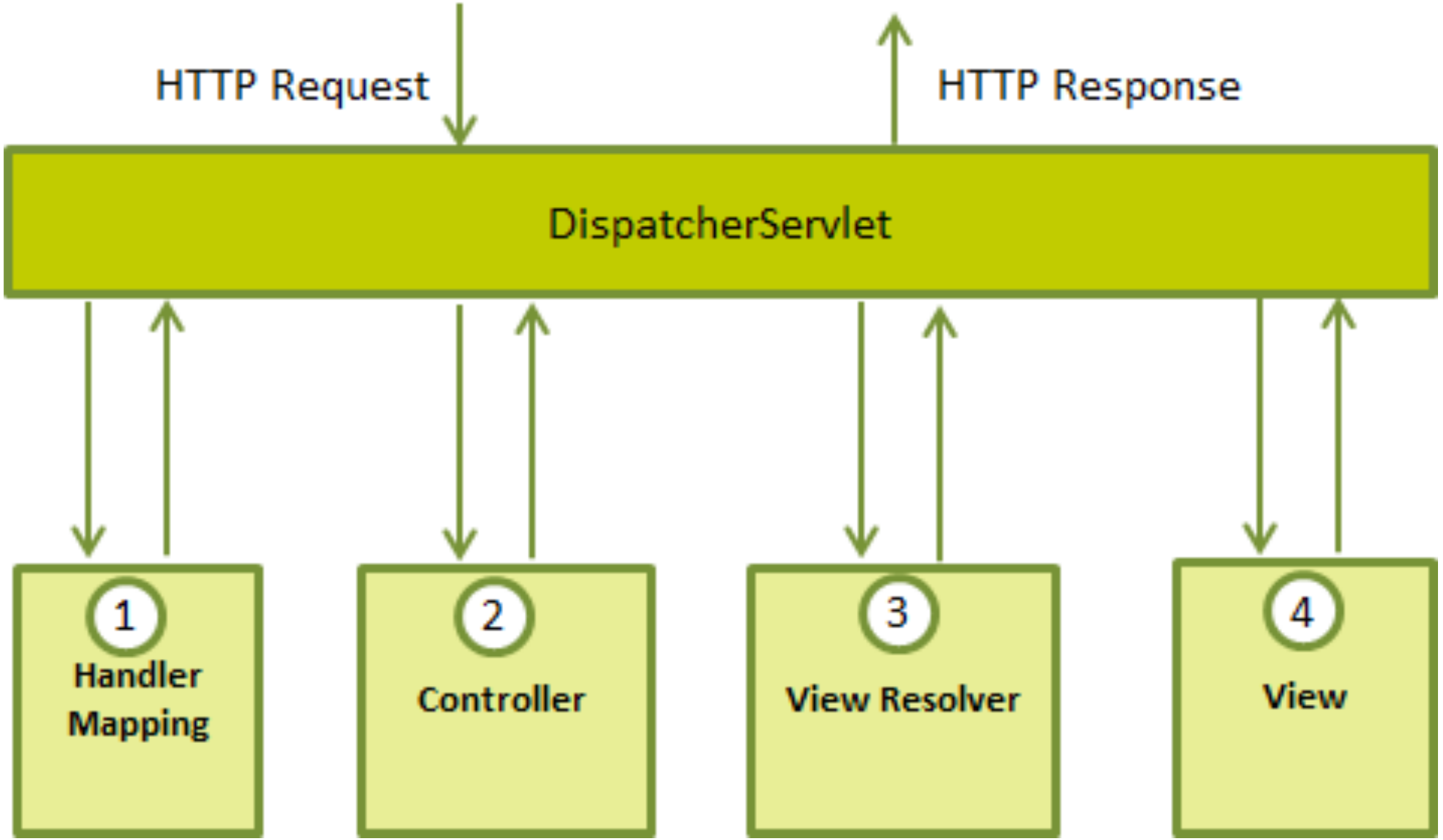
基于Spring MVC 4.0+

Spring MVC

Spring MVC是Model-View-Controller(MVC)的web框架，基于中央前端控制servlet（**DispatherServlet**），它用于分发请求到对应的处理方法，解析视图并返回结果。



Spring MVC


HTTP Request
HTTP Response
DispatcherServlet
1
Handler Mapping
2
Controller
3
View Resolver
4
View

Spring MVC

1.创建Intellij maven工程，选择maven-archetype-webapp原型

2.创建java文件夹，标示为sources root

3.添加pom文件依赖：spring-webmvc，javax.servlet-api，javax.servlet.jsp-api，jstl

4.添加controller，service和repository类

5.添加视图文件：webapp文件下jsp

6.添加servlet xml配置文件：注册bean，注释和视图解析器（基于xml配置）

或@Configuration类（基于注释配置）

7.配置web.xml文件：servlet映射到spring mvc（基于xml配置）

或实现WebApplicationInitializer接口类（基于注释配置）

Spring MVC

@Controller：处理HTTP请求的Spring Bean注释
@RequestMapping：Web请求映射到指定的类或方法，controller类是默认的处理器。包含value，method，params等属性。返回值是视图文件前缀（视图文件前后缀以及文件位置定义在xml文件中）

ModelMap是LinkedHashMap的子类，增加了用于controller的特性
addAttribute方法可以不传key，只传value，默认key为value的类型
addAttribute方法返回ModelMap，可以链式操作
ModelMap的泛型类型是Map<String, Object>，适合于视图模型
还有一个**Model**接口也可以用于同样操作，ExtendedModelMap是继承于Model，并且实现ModelMap的增强型ModelMap

<mvc:annotation-driven /> 允许通过注释注入类
<context:component-scan base-package="com.dharma.springmvc" /> 指定扫描目录，检查目录下文件是否有注释（e.g. @Controller, @Service, @Repository, @Component），然后将带有注释的类注册到类工厂
通过xml文件声明view resolver可以实现“代理”请求到对应的视图（jsp，模版文件等）

Spring MVC

@EnableWebMvc = mvc:annotation-driven：支持使用@Controller+@RequestMapping组合处理请求
@ComponentScan = context:component-scan base-package="..."：Spring寻找beans/class的目录

addResourceHandlers：配置静态资源处理器，如CSS,JavaScript,images等，以/static/开始的请求会被映射到webapp下的/static/文件夹。addResourceHandlers是抽象类WebMvcConfigurerAdapter的方法

messageSource方法用于配置国家化消息。假设参数是messages，那么它会寻找class path下的messages.properties文件

@ModelAttribute注释用于绑定方法参数或者方法返回值到model属性，然后暴露给视图（jsp）

@Valid通知spring去验证请求数据，验证结果存放到BindingResult，或者生成error

Spring MVC REST 2

基于Spring MVC 4.0+

Spring MVC REST

实现Restful API的基本方式

= Spring MVC + Restful style + JSON

@PathVariable：表示在RUI中定义的变量参数

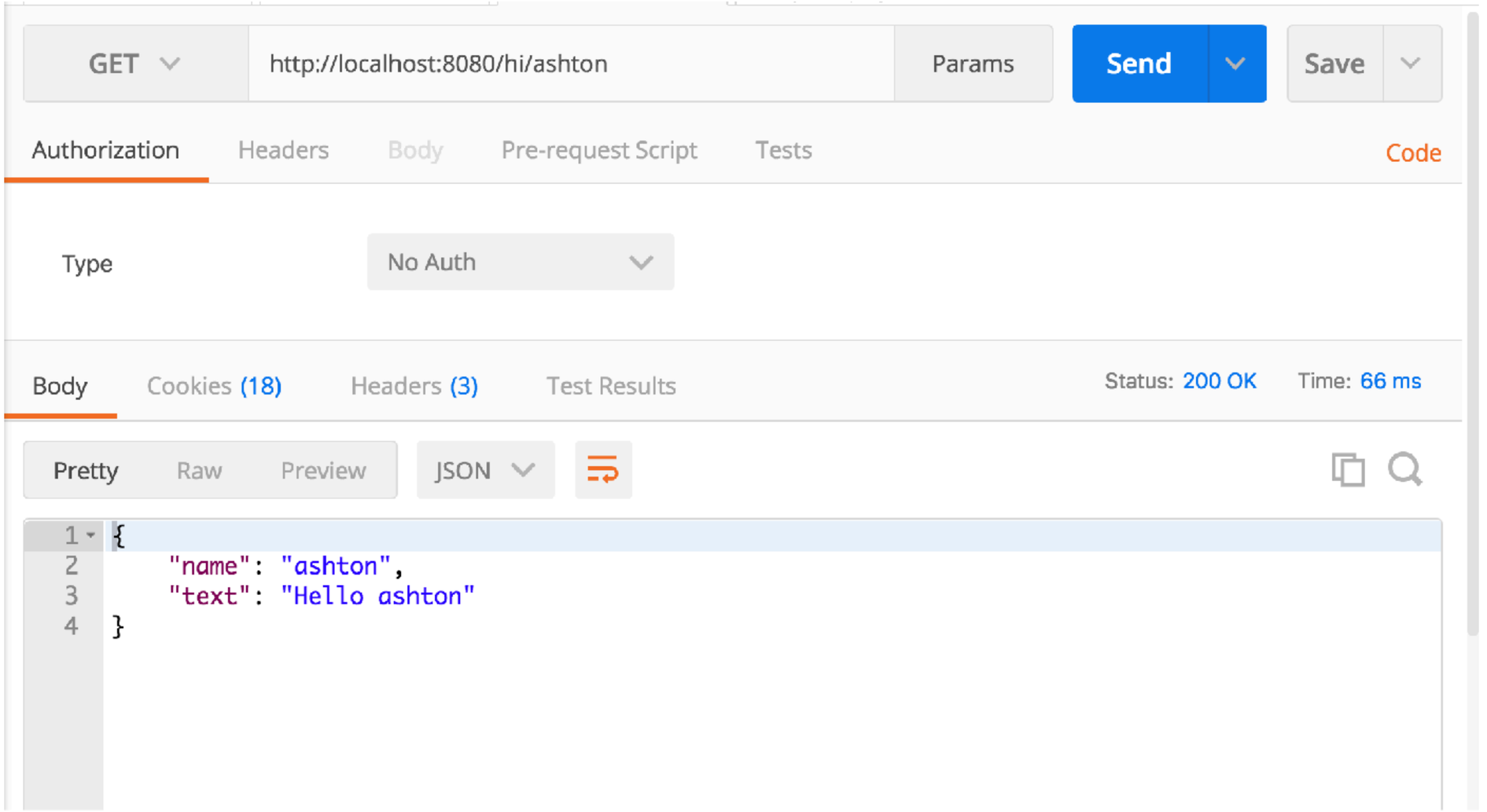
@RestController：表示该类是一个controller，但每个方法返回领域对象或POJO对象，而不是视图名称。这意味着我们不在需要视图解析器（view-resolvers），响应将以更方便用户理解的JSON等格式返回，而不是传统式html代码。

Jackson library(jackson-databind)用于将response转为JSON字符串，Jackson XML Extension library(jackson-dataformat-xml)用于response转为XML字符串

1. 引入jackson-databind，POJO不变 —> 默认JSON格式
2. 引入jackson-dataformat-xml，POJO不变 —> XML格式
3. JAXB（@XmlRootElement， @XmlElement），没有jackson-dataformat-xml：返回XML格式

Spring MVC REST

API测试：1. 浏览器；2. postman(https://www.getpostman.com/ + Chrome插件)



跨域问题

同源策略（Same Origin Policy）是一种安全策略，用于限制不同源之间的信息访问

= 协议相同 + 域名相同 + 端口相同

http://www.dharma-mall.com/	http://www.dharma-mall.com/product/	
http://www.dharma-mall.com/	http://www.dharma-mall.com:8888/product/	端口不同
https://www.dharma-mall.com/	http://www.dharma-mall.com/product/	协议不同
http://www.dharma-mall.com/	http://www.tmall.com/product/	域名不同
http://www.dharma-mall.com/	http://dharma-mall.com/product/	子域名不同

常见API访问错误：

- "No ‘Access-Control-Allow-Origin’ header is present on the requested resource. Origin ‘http://127.0.0.1:8080’ is therefore not allowed access."
- "XMLHttpRequest cannot load http://dharma-mall.com/product. Origin http://localhost:8080 is not allowed by Access-Control-Allow-Origin."

解决方案：

CORS（Cross-Origin Resource Sharing），API响应带上有CORS访问控制的header，赋予user agent访问指定资源的能力

跨域问题

```
@Override
protected Filter[] getServletFilters() {
    Filter [] singleton = { new CORSFilter()};
    return singleton;
}

public class CORSFilter implements Filter {
        public void doFilter(
            ServletRequest req, ServletResponse res, FilterChain chain)
            throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) res;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, PUT, OPTIONS, DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "x-requested-with");
        chain.doFilter(req, res);
    }
}
```

RequestBody & ResponseBody

@RequestBody和@ResponseBody注释用于实现HTTP request/response和领域对象（model）之间的转换，前者用于请求参数，后者用于返回值。

@RequestBody
Spring使用HTTP Message converters，根据accept header将HTTP请求体转换为领域对象（反序列化）
浏览器发送的请求带有Accept header，它用于告诉服务器接受的内容类型（content type）
服务器返回的响应也包含Accept header，它用于告诉浏览器实际的内容类型（content type）
主要用在POST和PUT请求，包含请求体和内容类型

Spring提供很多默认的HttpMessageConverters, 它依赖于project classpath定义的类库
例如，请求Header中Content-Type是application/json或application/xml，并且引入了Jackson类库，那么spring将转换代理给MappingJackson2HttpMessageConverter用于json，或MappingJackson2XmlHttpMessageConverter用于xml
ResponseEntity代表整个HTTP response，包含status code, headers和body

@ResponseBody
Spring会使用HTTP Message converters，根据Content-Type类型，将返回值转换为HTTP响应体（序列化）

RequestBody & ResponseBody

```
@RequestMapping(value="/user/create", method=RequestMethod.POST)
public ResponseEntity<Void> createUser(@RequestBody User user, UriComponentsBuilder ucBuilder){

    if(userService.isUserExist(user)){
        return new ResponseEntity<Void>(HttpStatus.CONFLICT);
    }

    userService.saveUser(user);

    HttpHeaders headers = new HttpHeaders();
    headers.setLocation(ucBuilder.path("/user/{id}").buildAndExpand(user.getId()).toUri());
    return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
}

@RequestMapping(value = "/user/all", method = RequestMethod.GET)
public @ResponseBody List<User> listAllUsers() {
    return userService.findAllUsers();
}
```

NOTE: 从Spring 4开始推荐使用@RestController，@RestController = @Controller + @ResponseBody

RequestBody & ResponseBody

Spring提供了默认的Http message converters，他们都实现了HttpMessageConverter interface

- **StringHttpMessageConverter** - 支持所有文本类型(text/*)，返回类型为text/plain，用于处理字符串
- **FormHttpMessageConverter** - 支持application/x-www-form-urlencoded，返回MultiValueMap，用于处理表单数据
- **ByteArrayHttpMessageConverter** - 支持所有媒体类型(*/*)，返回application/octet-stream，用于处理字节数组
- **MarshallingHttpMessageConverter** - 支持XML(text/xml)和(application/xml)，用于处理XML数据，使用Marshaller
- **MappingJackson2HttpMessageConverter** - 支持(application/json)，用于处理JSON数据，使用Jackson ObjectMapper
- **MappingJackson2XmlHttpMessageConverter** 支持支持(application/xml)，用于处理XML数据，使用Jackson XmlMapper
- **SourceHttpMessageConverter** - 支持(text/xml)和(application/xml)，用于处理javax.xml.transform.Source
- **BufferedImageHttpMessageConverter** - 支持Java I/O API支持的媒体类型，用于处理java.awt.image.BufferedImage

定制HttpMessageConverters

```
@Bean
   public MappingJackson2HttpMessageConverter mappingJackson2HttpMessageConverter() {
       MappingJackson2HttpMessageConverter converter = new MappingJackson2HttpMessageConverter();
       converter.setObjectMapper(new ObjectMapper().configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false));
       return converter;
   }
```

Thanks!

Any questions?