

谈谈 Web 那些事



扫码试看/订阅
《玩转 Spring 全家桶》

Spring MVC 实践

编写第一个Spring MVC Controller

认识 Spring MVC

DispatcherServlet

- Controller
- xxxResolver
 - ViewResolver
 - HandlerExceptionResolver
 - MultipartResolver
- HandlerMapping 请求映射处理

Spring MVC 中的常用注解

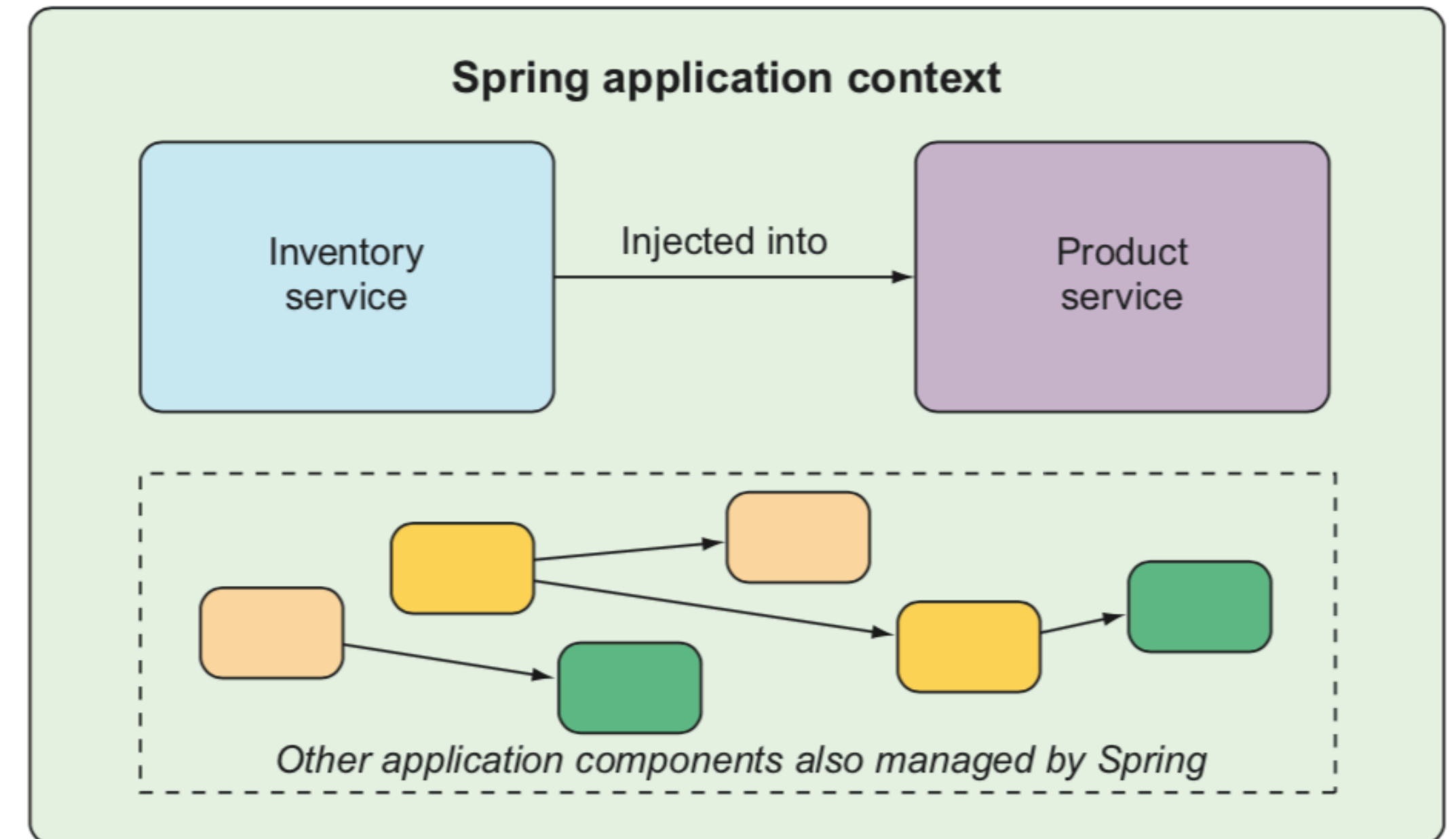
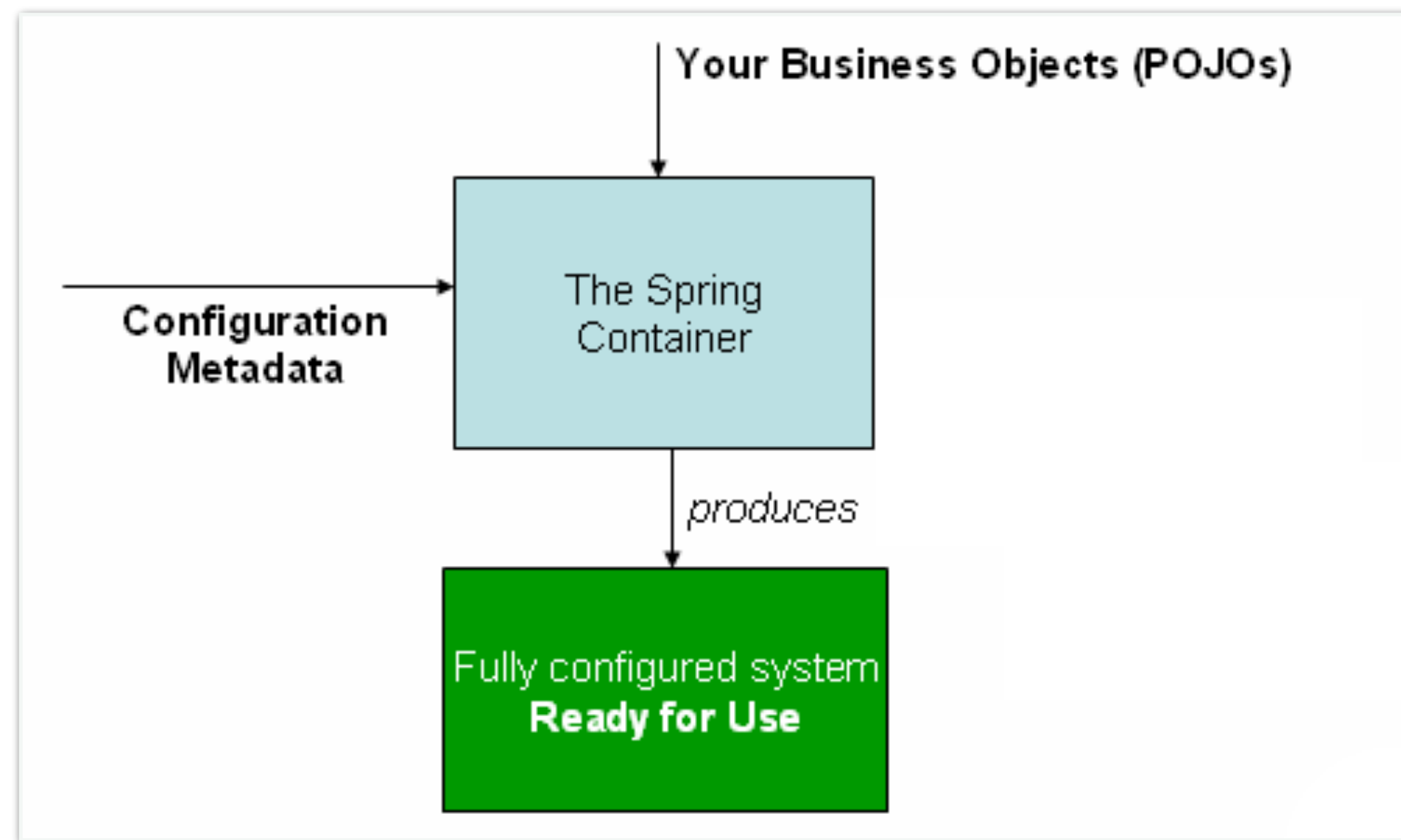
- @Controller
 - @RestController
- @RequestMapping controller处理什么请求
定义在类上面的就是定义在方法上的路径的前缀
 - @GetMapping / @PostMapping
 - @PutMapping / @DeleteMapping
- @RequestBody / @ResponseBody / @ResponseStatus
 - 请求报文体 响应报文体 响应Http代码

“Talk is cheap, show me the code.”

Chapter 6 / simple-controller-demo

理解 Spring 的应用上下文

Spring 的应用程序上下文



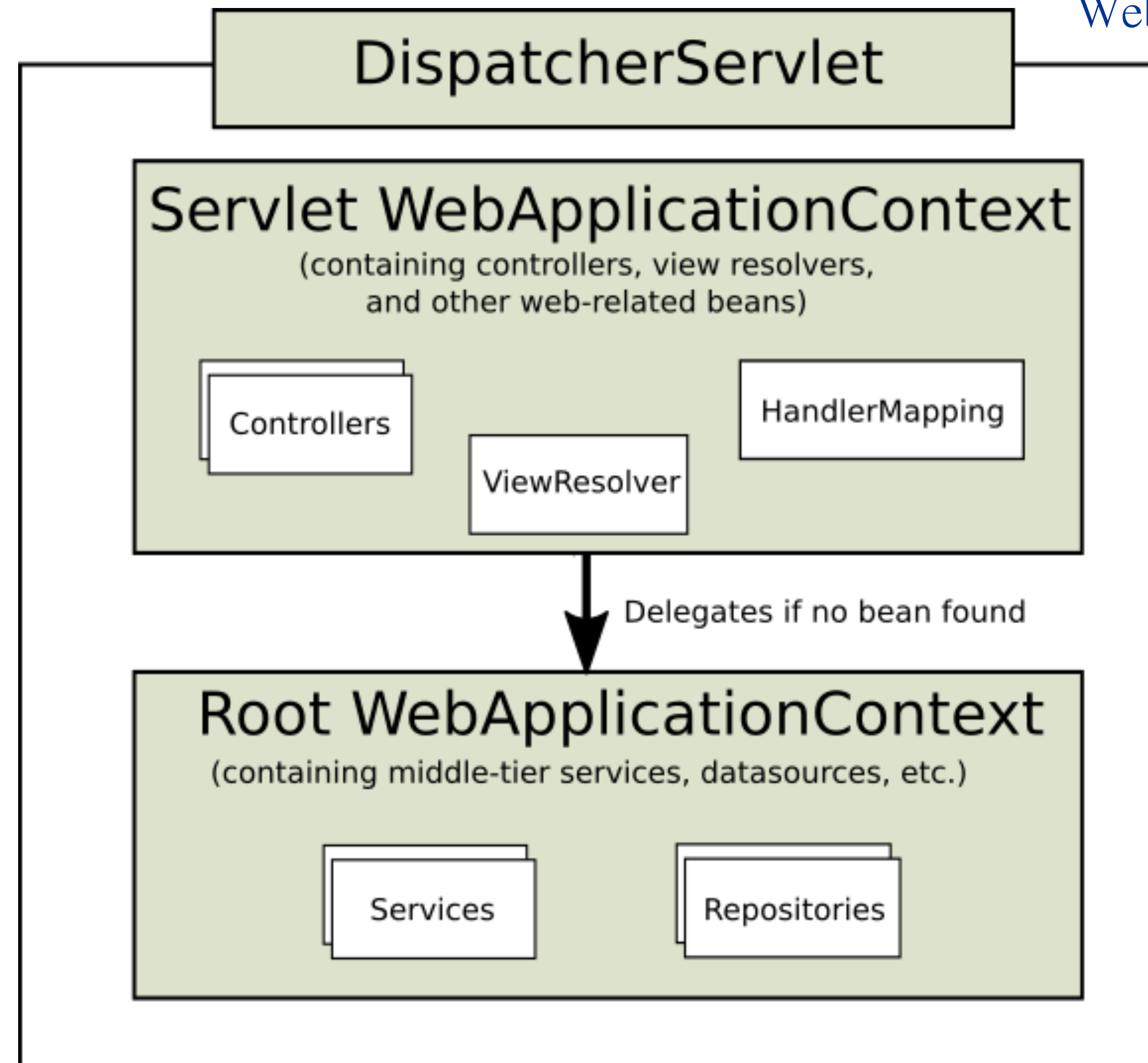
Spring 的应用程序上下文

关于上下文常用的接口

- BeanFactory 一般不用这个，没有特殊原因请使用ApplicationContext
 - DefaultListableBeanFactory
- ApplicationContext
 - ClassPathXmlApplicationContext
 - FileSystemXmlApplicationContext
 - AnnotationConfigApplicationContext
- WebApplicationContext

Web 上下文层次

如果AOP拦截在servlet web
ApplicationContext里面，但这个时候
并不会拦截到root
WebApplicationContext



Web 上下文层次

基于注解

```
<web-app>

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

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/app-context.xml</param-value>
  </context-param>

  <servlet>
    Servlet config
    <servlet-name>app</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value></param-value>应用service,dao由contextLoaderListener加载
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>app</servlet-name>
    <url-pattern>/app/*</url-pattern>
  </servlet-mapping>

</web-app>
```

```
public class MyWebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

  @Override
  protected Class<?>[] getRootConfigClasses() {
    return new Class<?>[] { RootConfig.class };
  }

  @Override
  protected Class<?>[] getServletConfigClasses() {
    return new Class<?>[] { App1Config.class };
  }

  @Override
  protected String[] getServletMappings() {
    return new String[] { "/app1/*" };
  }
}
```

基于xml的

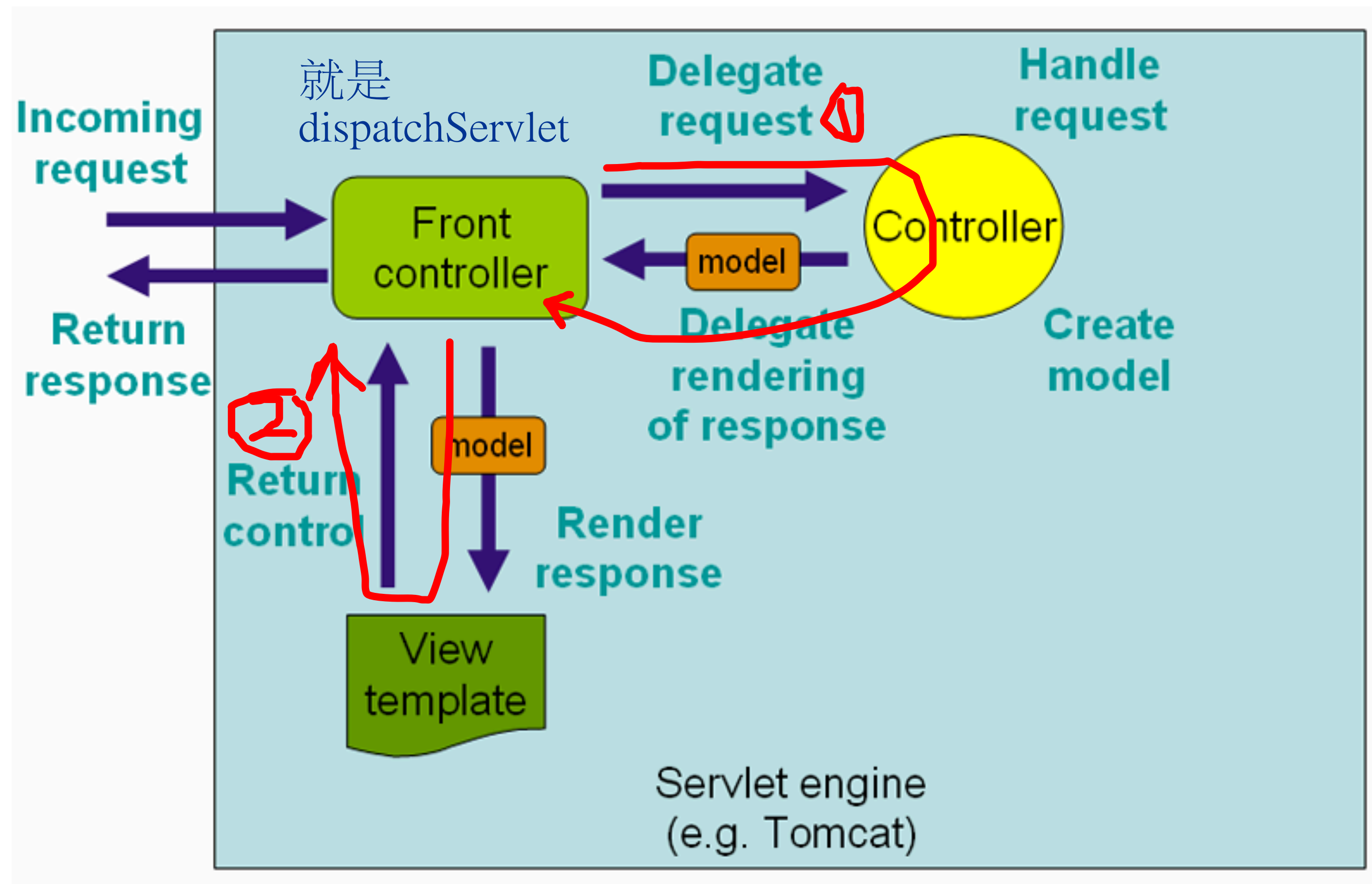
“Talk is cheap, show me the code.”

Chapter 6 / context-hierarchy-demo

Spring MVC 中的各种机制

请求处理

Spring MVC 的请求处理流程



一个请求的大致处理流程

绑定一些 Attribute

- WebApplicationContext / LocaleResolver / ThemeResolver

处理 Multipart

- 如果是，则将请求转为 MultipartHttpServletRequest

Handler 处理

- 如果找到对应 Handler，执行 Controller 及前后置处理器逻辑

处理返回的 Model ， 呈现视图

render

什么是multipart/form-data请求

根据http/1.1 rfc 2616的协议规定，我们的请求方式只有OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE等，那为为何我们还会有multipart/form-data请求之说呢？这就要从头来说了。

http协议规定以ASCII码传输，建立在tcp，ip协议智商的引用规范，规范内容把http请求分成3个部分，状态行，请求头，请求体。所有的方法，实现都是围绕如何使用和组织这三部分来完成了，万变不离其宗，http的知识大家可以问度娘。

既然上面请求方式里面没有multipart/form-data那这个请求又是怎么回事呢，其实是一回事，multipart/form-data也是在post基础上演变而来的，具体如下：

- 1.multipart/form-data的基础方式是post，也就是说通过post组合方式来实现的。
- 2.multipart/form-data于post方法的不同之处在于请求头和请求体。
- 3.multipart/form-data的请求头必须包含一个特殊的头信息：Content-Type，其值也必须为multipart/form-data，同时还需要规定一个内容分割用于分割请求提中多个post的内容，如文件内容和文本内容是需要分隔开来的，不然接收方就无法解析和还原这个文件了，具体的头信息如下：

Content-Type: multipart/form-data; boundary=\${bound}

其中\${bound} 是一个占位符，代表我们规定的分割符，可以自己任意规定，但为了避免和正常文本重复了，尽量要使用复杂一点的内容。如：-----56423498738365

multipart/form-data的请求体也是一个字符串，不过和post的请求体不同的是它的构造方式，post是简单的name=value键值连接，而multipart/form-data是添加了分隔符等内容的构造体，具体如下：

```
--${bound}
Content-Disposition: form-data; name="Filename"

HTTP.pdf
--${bound}
Content-Disposition: form-data; name="file000"; filename="HTTP协议详解.pdf"
Content-Type: application/octet-stream

%PDF-1.5
file content
%%EOF

--${bound}
Content-Disposition: form-data; name="Upload"

Submit Query
--${bound}--
```

其中\${bound}是之前头信息中的分隔符，如果头信息中规定是123，那这里也要是123；可以很容易看到，这个请求体是多个相同部分组成的：每一部分都是以--加分隔符开始的，然后是该部分内容的描述信息，然后一个回车，然后是描述信息的具体内容；如果传送的内容是一个文件的话，那么还会包含文件名信息以及文件内容类型。上面第二部分是一个文件体的结构，最后以--分隔符--结尾，表示请求体结束。

如何定义处理方法

定义映射关系

@Controller

@RequestMapping

- path / method 指定映射路径与方法

带了什么头可以映射

- params / headers 限定映射范围

带了什么参数可以映射

- consumes / produces 限定请求与响应格式

包含特定Content-type请求

请求只会响应特定的响应形式

一些快捷方式

- @RestController response body+controller

- @GetMapping / @PostMapping / @PutMapping / @DeleteMapping / @PatchMapping

特定http method

定义处理方法

- @RequestBody / @ResponseBody / @ResponseStatus Http状态码
- @PathVariable / @RequestParam / @RequestHeader
url路径变量 请求参数 请求Http头
- HttpEntity / ResponseEntity

定义处理方法

详细参数

- <https://docs.spring.io/spring/docs/5.1.5.RELEASE/spring-framework-reference/web.html#mvc-ann-arguments>

详细返回

- <https://docs.spring.io/spring/docs/5.1.5.RELEASE/spring-framework-reference/web.html#mvc-ann-return-types>

方法示例

只能处理Application_JSON的请求

```
@PostMapping(path = "/", consumes = MediaType.APPLICATION_JSON_VALUE,  
             produces = MediaType.APPLICATION_JSON_UTF8_VALUE) 只返回Application_JSON,并且字符  
@ResponseStatus(HttpStatus.CREATED)                           集为UTF8  
public CoffeeOrder create(@RequestBody NewOrderRequest newOrder) {  
    log.info("Receive new Order {}", newOrder);  
    Coffee[] coffeeList = coffeeService.getCoffeeByName(newOrder.getItems())  
        .toArray(new Coffee[] {});  
    return orderService.createOrder(newOrder.getCustomer(), coffeeList);  
}
```

方法示例

```
@RequestMapping(path =("/{id}", method = RequestMethod.GET,  
    produces = MediaType.APPLICATION_JSON_UTF8_VALUE)  
@ResponseBody  
public Coffee getById(@PathVariable Long id) {  
    Coffee coffee = coffeeService.getCoffee(id);  
    return coffee;  
}
```

也可以指定url中哪个名字作为参数

必须有Name的http RequestParam才会匹配

```
@GetMapping(path = "/", params = "name")  
@ResponseBody  
public Coffee getByName(@RequestParam String name) {  
    return coffeeService.getCoffee(name);  
}
```


“Talk is cheap, show me the code.”

Chapter 6 / complex-controller-demo

定义类型转换

自己实现 WebMvcConfigurer

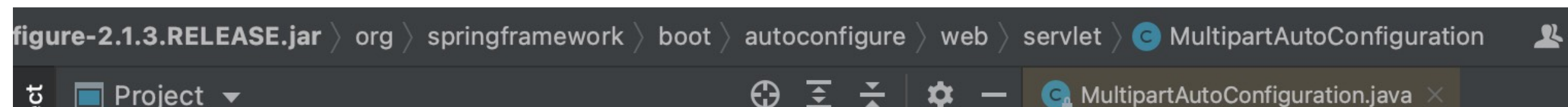
- Spring Boot 在 WebMvcAutoConfiguration 中实现了一个
- 添加自定义的 Converter
- 添加自定义的 Formatter

定义校验

- 通过 Validator 对绑定结果进行校验
 - Hibernate Validator
- @Valid 注解
- BindingResult 校验结果通过这个传递进来

controller控制层写参数接收的入口，需要注意的是@Valid 和 BindingResult 是一一对应的，如果有多个@Valid，那么每个@Valid后面都需要添加BindingResult用于接收bean中的校验信息

Multipart 上传



- 配置 `MultipartResolver`
 - Spring Boot 自动配置 `MultipartAutoConfiguration`
- 支持类型 `multipart/form-data`
- `MultipartFile` 类型

```
web > servlet > MultipartProperties > location | WaiverServiceApplication
MultipartAutoConfiguration.java x MultipartProperties.java x
Since: 1.1.0
Author: Josh Long, Toshiaki Maki, Stephane Nicoll
48 @ConfigurationProperties(prefix = "spring.servlet.multipart", ignoreUnknownFields = true)
49 public class MultipartProperties {
50
51     | Whether to enable support of multipart uploads.
54     private boolean enabled = true;
55
56     | Intermediate location of uploaded files.
59     private String location;
60
61     | Max file size.
64     private DataSize maxFileSize = DataSize.ofMegabytes(1);
65
66     | Max request size.
69     private DataSize maxRequestSize = DataSize.ofMegabytes(10);
70
71     | Threshold after which files are written to disk.
74     private DataSize fileSizeThreshold = DataSize.ofBytes(0);
75
```

比如这个MaxRequestSize就可以在properties中配置：
spring.servlet.multipart.maxRequestSize = 10

form-data 例子

POST

http://localhost:8080/coffee/

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	file	coffee.txt			
	Key	Value	Description		

Body

Cookies

Headers (3)

Test Results

Status: 201 Created

Time: 274 ms

Size: 1.16 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
7  "price": {
8    "currencyUnit": {
9      "code": "CNY",
10     "numericCode": 156,
11     "decimalPlaces": 2,
12     "symbol": "CN¥",
13     "numeric3Code": "156",
14     "countryCodes": [
15       "CN"
16     ],
17     "pseudoCurrency": false
18   },
19   "amount": 25.00,
20   "negative": false,
21   "zero": false,
```

Bootcamp

Panner

Trash

“Talk is cheap, show me the code.”

Chapter 6 / more-complex-controller-demo

Spring MVC 中的各种机制

视图处理

视图解析的实现基础

ViewResolver 与 View 接口

- AbstractCachingViewResolver
- UrlBasedViewResolver
- FreeMarkerViewResolver
- ContentNegotiatingViewResolver
- InternalResourceViewResolver

DispatcherServlet 中的视图解析逻辑

- `initStrategies()`
 - `initViewResolvers()` 初始化了对应 `ViewResolver`
- `doDispatch()`
 - `processDispatchResult()`
 - 没有返回视图的话, 尝试 `RequestToViewNameTranslator`
 - `resolveViewName()` 解析 `View` 对象

DispatcherServlet 中的视图解析逻辑

使用 @ResponseBody 的情况

- 在 `HandlerAdapter.handle()` 的中完成了 Response 输出
- `RequestMappingHandlerAdapter.invokeHandlerMethod()`
- `HandlerMethodReturnValueHandlerComposite.handleReturnValue()`
- `RequestResponseBodyMethodProcessor.handleReturnValue()`

重定向

两种不同的重定向前缀

- redirect:
- forward:

Spring MVC 中的常用视图

Spring MVC 支持的视图

支持的视图列表

- <https://docs.spring.io/spring/docs/5.1.5.RELEASE/spring-framework-reference/web.html#mvc-view>
- Jackson-based JSON / XML
- Thymeleaf & FreeMarker

Template Engines

- ☐ Thymeleaf
Thymeleaf templating engine
- ☐ Freemarker
FreeMarker templating engine
- ☐ Mustache
Mustache templating engine
- ☐ Groovy Templates
Groovy templating engine

配置 MessageConverter

- 通过 WebMvcConfigurer 的 configureMessageConverters()
- Spring Boot 自动查找 HttpMessageConverters 进行注册

```
public class WebConfiguration implements WebMvcConfigurer {  
  
    @Override  
    public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {  
        Jackson2ObjectMapperBuilder builder = new Jackson2ObjectMapperBuilder()  
            .indentOutput(true)  
            .dateFormat(new SimpleDateFormat("yyyy-MM-dd"))  
            .modulesToInstall(new ParameterNamesModule());  
        converters.add(new MappingJackson2HttpMessageConverter(builder.build()));  
        converters.add(new  
MappingJackson2XmlHttpMessageConverter(builder.createXmlMapper(true).build()));  
    }  
}
```

Spring Boot 对 Jackson 的支持

- JacksonAutoConfiguration
 - Spring Boot 通过 @JsonComponent 注册 JSON 序列化组件
 - Jackson2ObjectMapperBuilderCustomizer
- JacksonHttpMessageConvertersConfiguration
 - 增加 jackson-dataformat-xml 以支持 XML 序列化

“Talk is cheap, show me the code.”

Chapter 6 / json-view-demo

“Thymeleaf is a modern server-side Java template engine for both web and standalone environments.”

– <https://www.thymeleaf.org/>

使用 Thymeleaf

添加 Thymeleaf 依赖

- `org.springframework.boot:spring-boot-starter-thymeleaf`

Spring Boot 的自动配置

- `ThymeleafAutoConfiguration`
 - `ThymeleafViewResolver`

Thymeleaf 的一些默认配置

- `spring.thymeleaf.cache=true`
- `spring.thymeleaf.check-template=true`
- `spring.thymeleaf.check-template-location=true`
- `spring.thymeleaf.enabled=true`
- `spring.thymeleaf.encoding=UTF-8`
- `spring.thymeleaf.mode=HTML`
- `spring.thymeleaf.servlet.content-type=text/html`
- `spring.thymeleaf.prefix=classpath:/templates/`
- `spring.thymeleaf.suffix=.html`

“Talk is cheap, show me the code.”

Chapter 6 / thymeleaf-view-demo

静态资源与缓存

Spring Boot 中的静态资源配置

核心逻辑

- `WebMvcConfigurer.addResourceHandlers()`

常用配置

- `spring.mvc.static-path-pattern=/**`
- `spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/`

Spring Boot 中的缓存配置

常用配置（默认时间单位都是秒）

- `ResourceProperties.Cache`
- `spring.resources.cache.cachecontrol.max-age=时间`
- `spring.resources.cache.cachecontrol.no-cache=true/false`
- `spring.resources.cache.cachecontrol.s-max-age=时间`

Controller 中手工设置缓存

```
@GetMapping("/book/{id}")
public ResponseEntity<Book> showBook(@PathVariable Long id) {

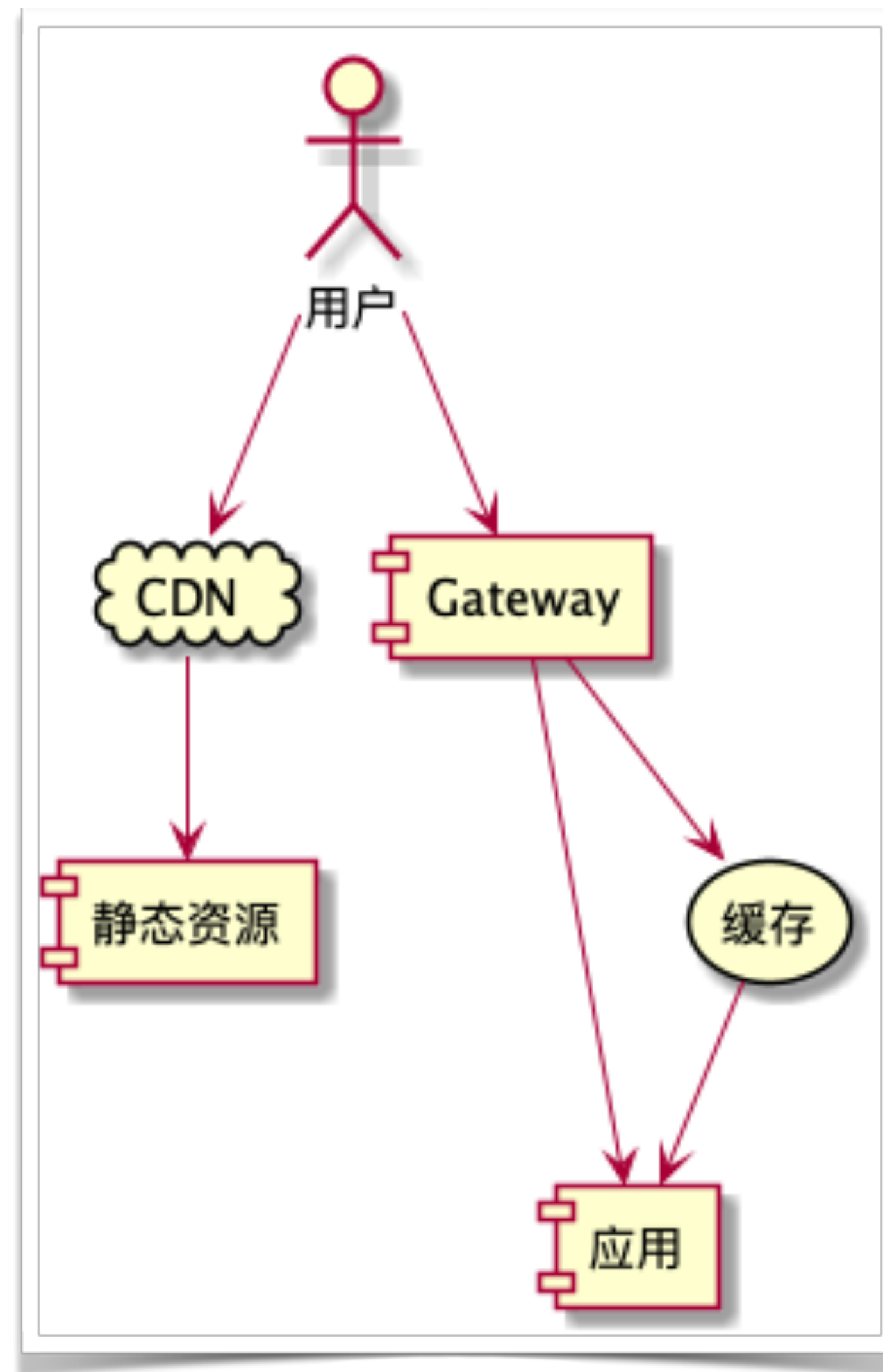
    Book book = findBook(id);
    String version = book.getVersion();

    return ResponseEntity
        .ok()
        .cacheControl(CacheControl.maxAge(30, TimeUnit.DAYS))
        .eTag(version) // lastModified is also available
        .body(book);
}
```

“Talk is cheap, show me the code.”

Chapter 6 / cache-demo

建议的资源访问方式



Spring MVC 中的各种机制

异常处理

Spring MVC 的异常解析

核心接口

- `HandlerExceptionResolver`

实现类

- `SimpleMappingExceptionHandlerResolver`
- `DefaultHandlerExceptionHandlerResolver`
- `ResponseStatusExceptionHandlerResolver`
- `ExceptionHandlerExceptionHandlerResolver`

异常处理方法

处理方法

- `@ExceptionHandler`

添加位置

- `@Controller` / `@RestController`
- `@ControllerAdvice` / `@RestControllerAdvice`

“Talk is cheap, show me the code.”

Chapter 6 / exception-demo

了解 Spring MVC 的切入点

Spring MVC 的拦截器

核心接口

- HandlerInteceptor
 - `boolean preHandle()`
 - `void postHandle()`
 - `void afterCompletion()`

Spring MVC 的拦截器

针对 **@ResponseBody** 和 **ResponseEntity** 的情况

- `ResponseBodyAdvice`

针对异步请求的接口

- `AsyncHandlerInterceptor`
 - `void afterConcurrentHandlingStarted()`

拦截器的配置方式

常规方法

- `WebMvcConfigurer.addInterceptors()`

Spring Boot 中的配置

- 创建一个带 `@Configuration` 的 `WebMvcConfigurer` 配置类
- 不能带 `@EnableWebMvc` (想彻底自己控制 MVC 配置除外)

“Talk is cheap, show me the code.”

Chapter 6 / springbucks

SpringBucks 进度小结

本章小结

- 解释了什么是 Spring 的 ApplicationContext
- 了解了 Spring MVC 的基本使用
- 理解 Spring MVC 的多种机制

SpringBucks 进度小结

- 拆分了 waiter-service
- 增加了更多 REST 方法
- 增加了缓存、性能日志与异常处理



扫码试看/订阅
《玩转 Spring 全家桶》