第五章

Spring MVC

内容提要

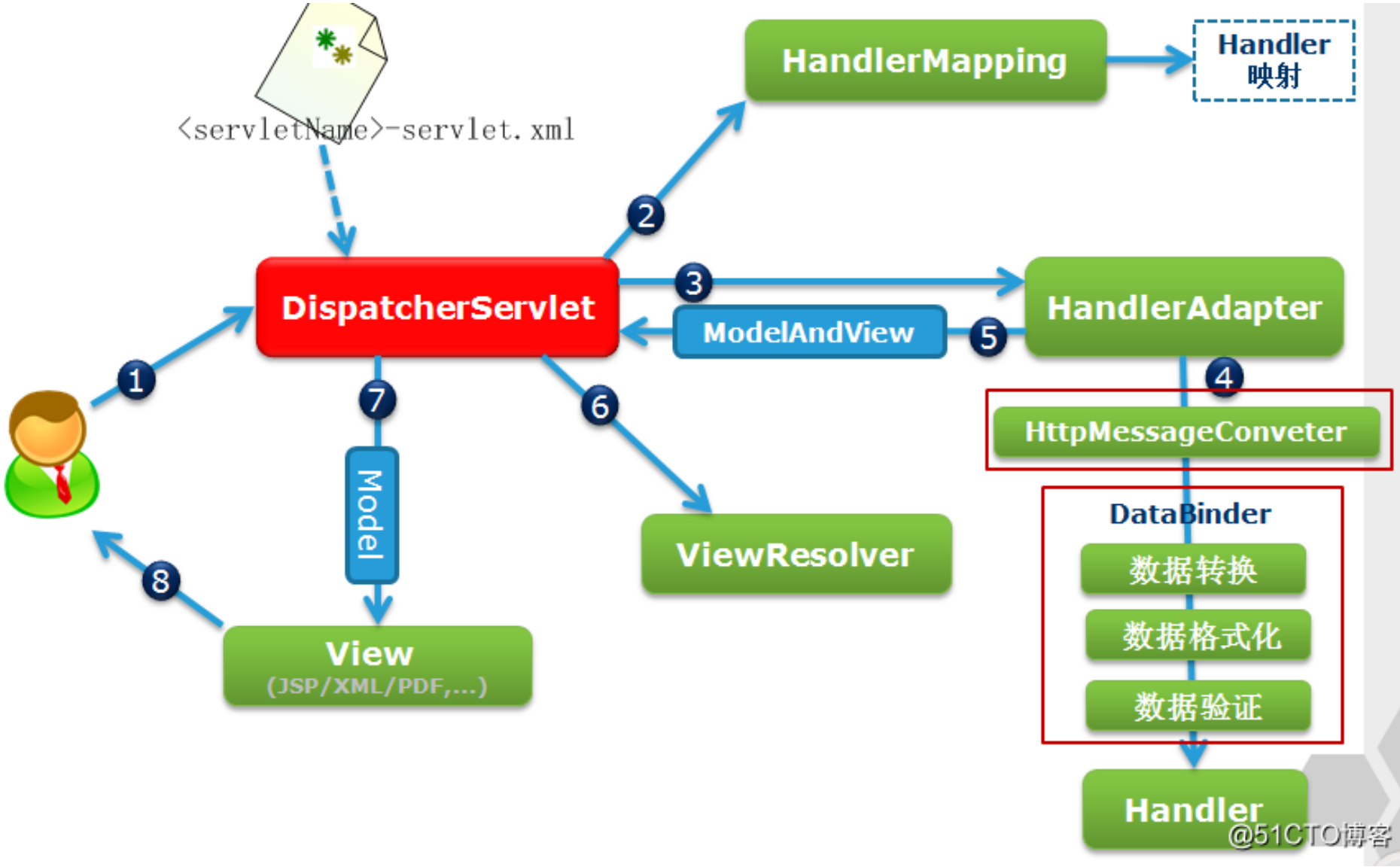
- **第一个Spring MVC程序**
- **设计Restful Web Service**
- **Spring 的上下文Context**
- **请求处理**
- **视图处理**
- **静态资源与缓存**
- **异常处理**
- **拦截器**

认识 Spring MVC

DispatcherServlet

- Controller
- xxxResolver
  - **ViewResolver**
  - **HandlerExceptionResolver**
  - **MultipartResolver**
- HandlerMapping

Spring MVC工作流程图



@51CTO博客

Spring MVC中的常用注解

simple-controller-demo

- @Controller
  - **@RestController**
- **@RequestMapping**
  - **@GetMapping / @PostMapping**
  - **@PutMapping / @DeleteMapping**
- @RequestBody / @ResponseBody / @ResponseStatus

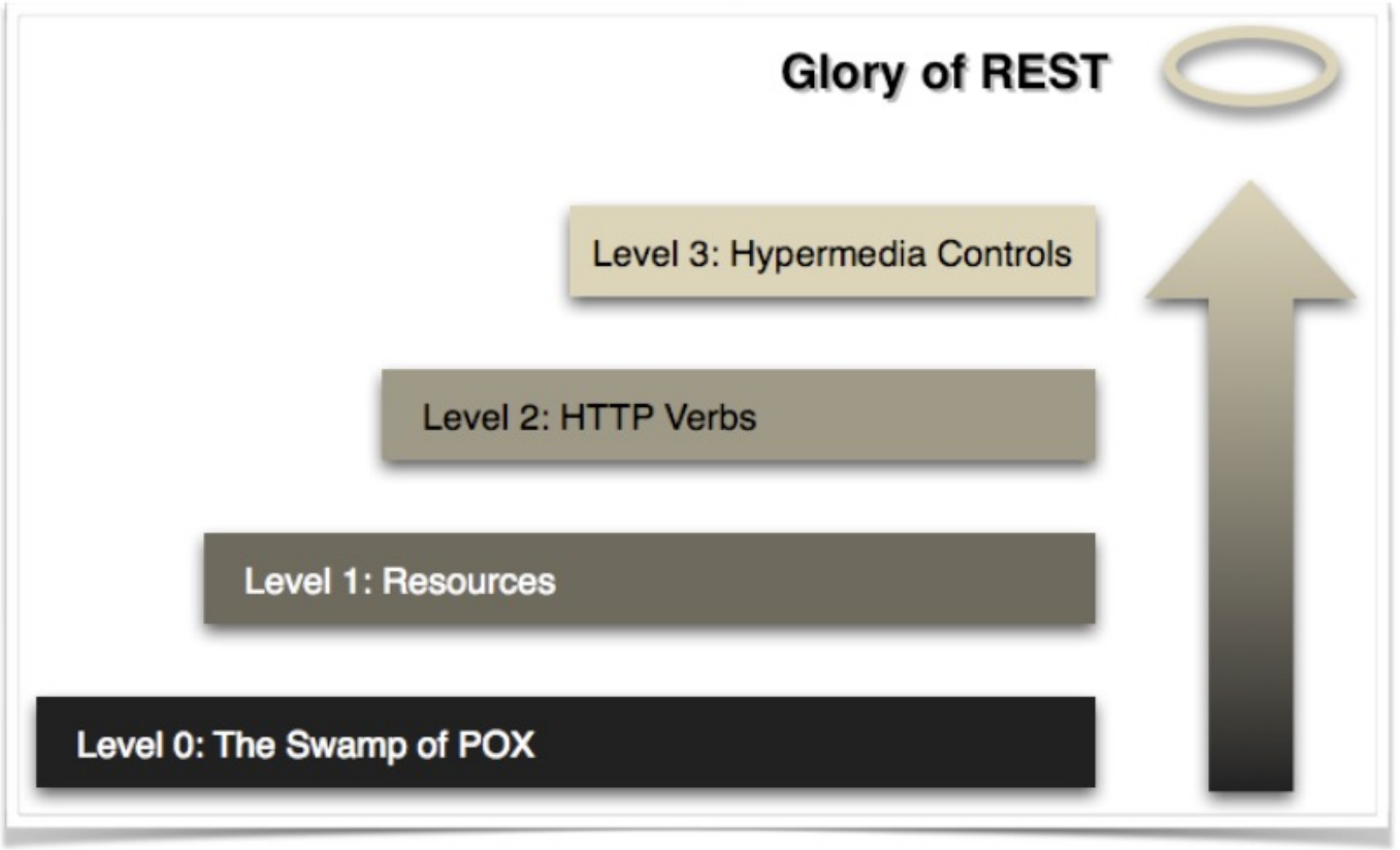
内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

什么是REST

- **REST -全称是 Resource REpresentational State Transfer**
- Roy Thomas Fielding在他2000年的博士论文中提出的
- **REST 是一组架构规范，并非协议或标准。API 开发人员可以采用各种方式实施 REST。**
- **RESTful API 最佳实践**

Richardson 成熟度模型



如何实现 Restful API

- 识别资源
- 选择合适的资源粒度
- 设计 URI
- 选择合适的 HTTP 方法和返回码
- 设计资源的表述

识别资源

- 找到领域名词
  - 能用 CRUD 操作的名词
- 将资源组织为集合(即集合资源)
- 将资源合并为复合资源
- 计算或处理理函数

选择合适的资源粒度

站在服务端的角度，要考虑

- 网络效率
- 表述的多少
- 客户端的易用程度

站在客户端的角度，要考虑

- 可缓存性
- 修改频率
- 可变性

设计 URI

- 使用域及子域对资源进行合理的分组或划分
- 在 URI 的路径部分使用斜杠分隔符 (/) 来表示资源之间的层次关系
- 在 URI 的路径部分使用逗号 (,) 和分号 (;) 来表示非层次元素
- 使用连字符 (-) 和下划线 (_) 来改善长路径中名称的可读性
- 在 URI 的查询部分使用用“与”符号 (&) 来分隔参数
- 在 URI 中避免出现文件扩展名 (例如 .php，.aspx 和 .jsp)

HTTP 方法

动作	安全/幂等	用途
GET	Y / Y	信息获取
POST	N / N	该方法用途广泛，可用于创建、更新或一次性对多个资源进行修改
DELETE	N / Y	删除资源
PUT	N / Y	更新或者完全替换一个资源
HEAD	Y / Y	获取与GET一样的HTTP头信息，但没有响应体
OPTIONS	Y / Y	获取资源支持的HTTP方法列表
TRACE	Y / Y	让服务器返回其收到的HTTP头

URI 与 HTTP 方法的组合

URI	HTTP方法	含义
/coffee/	GET	获取全部咖啡信息
/coffee/	POST	添加新的咖啡信息
/coffee/{id}	GET	获取特定咖啡信息
/coffee/{id}	DELETE	删除特定咖啡信息
/coffee/{id}	PUT	修改特定咖啡信息

HTTP 状态码

状态码	描述	状态码	描述
200	OK	400	Bad Request
201	Created	401	Unauthorized
202	Accepted	403	Forbidden
301	Moved Permanently	404	Not Found
303	See Other	410	Gone
304	Not Modified	500	Internal Server Error
307	Temporary Redirect	503	Service Unavailable

设计资源的表述

JSON

- MappingJackson2HttpMessageConverter
- GsonHttpMessageConverter
- JsonbHttpMessageConverter

XML

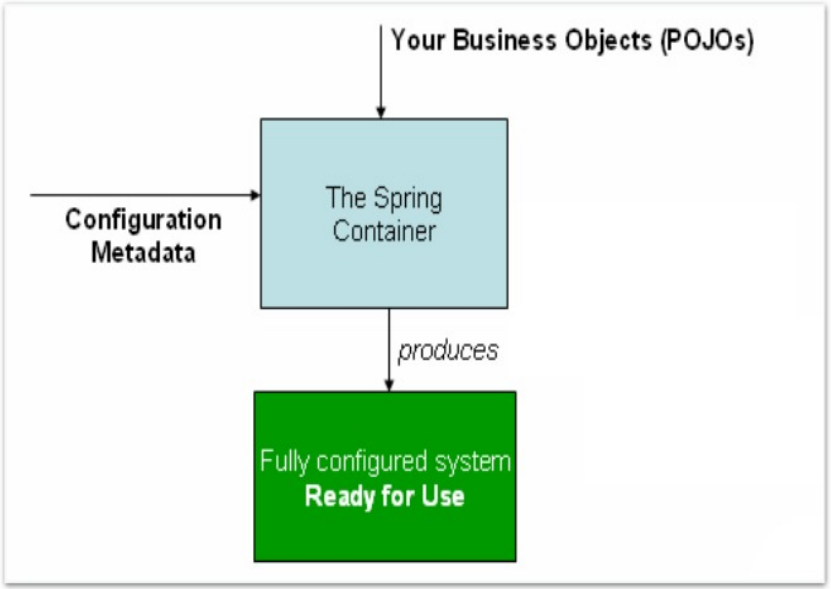
- MappingJackson2XmlHttpMessageConverter
- Jaxb2RootElementHttpMessageConverter

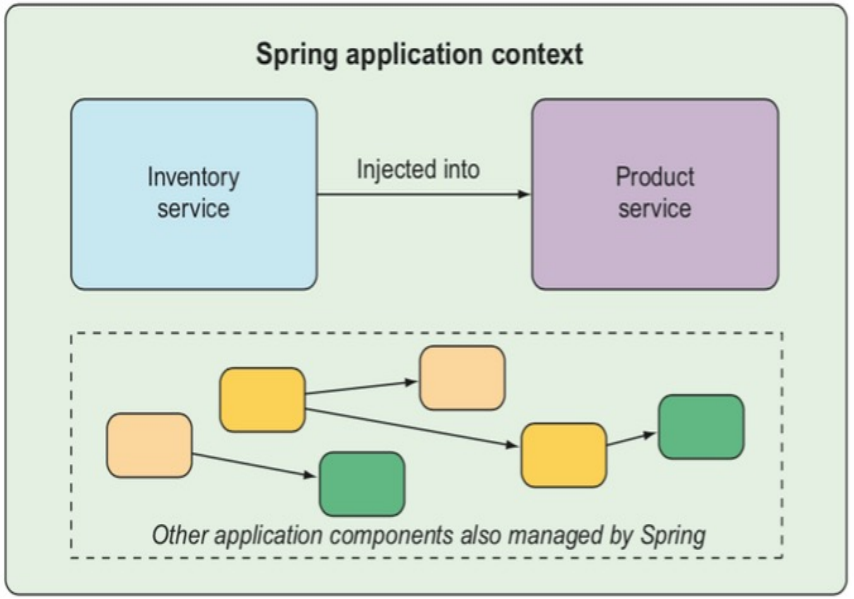
HTML

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

Spring 的应用程序上下文

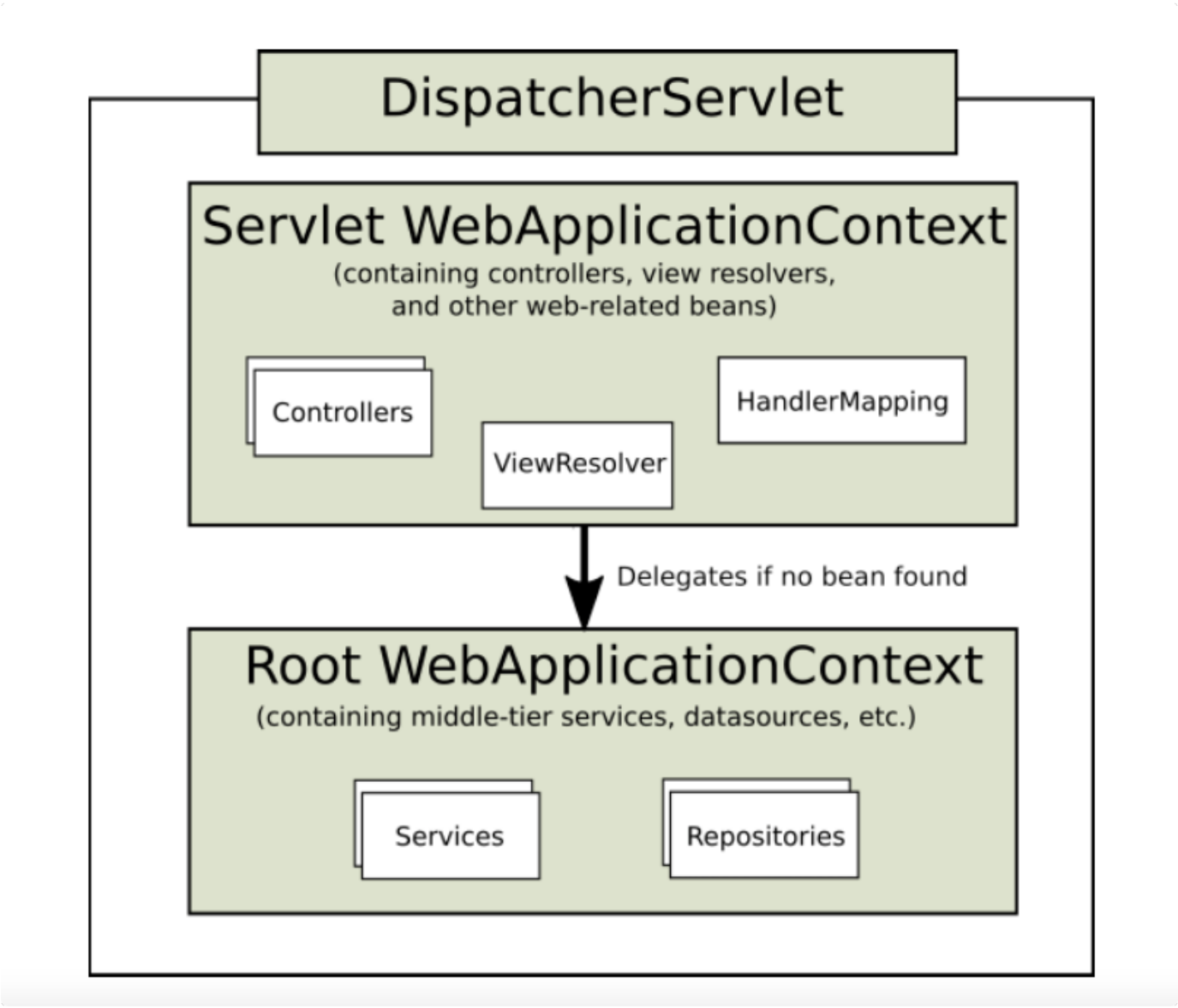




上下文Context常用的接口

- BeanFactory
  - DefaultListableBeanFactory
- ApplicationContext
  - ClassPathXmlApplicationContext
  - FileSystemXmlApplicationContext
  - AnnotationConfigApplicationContext
- WebApplicationContext

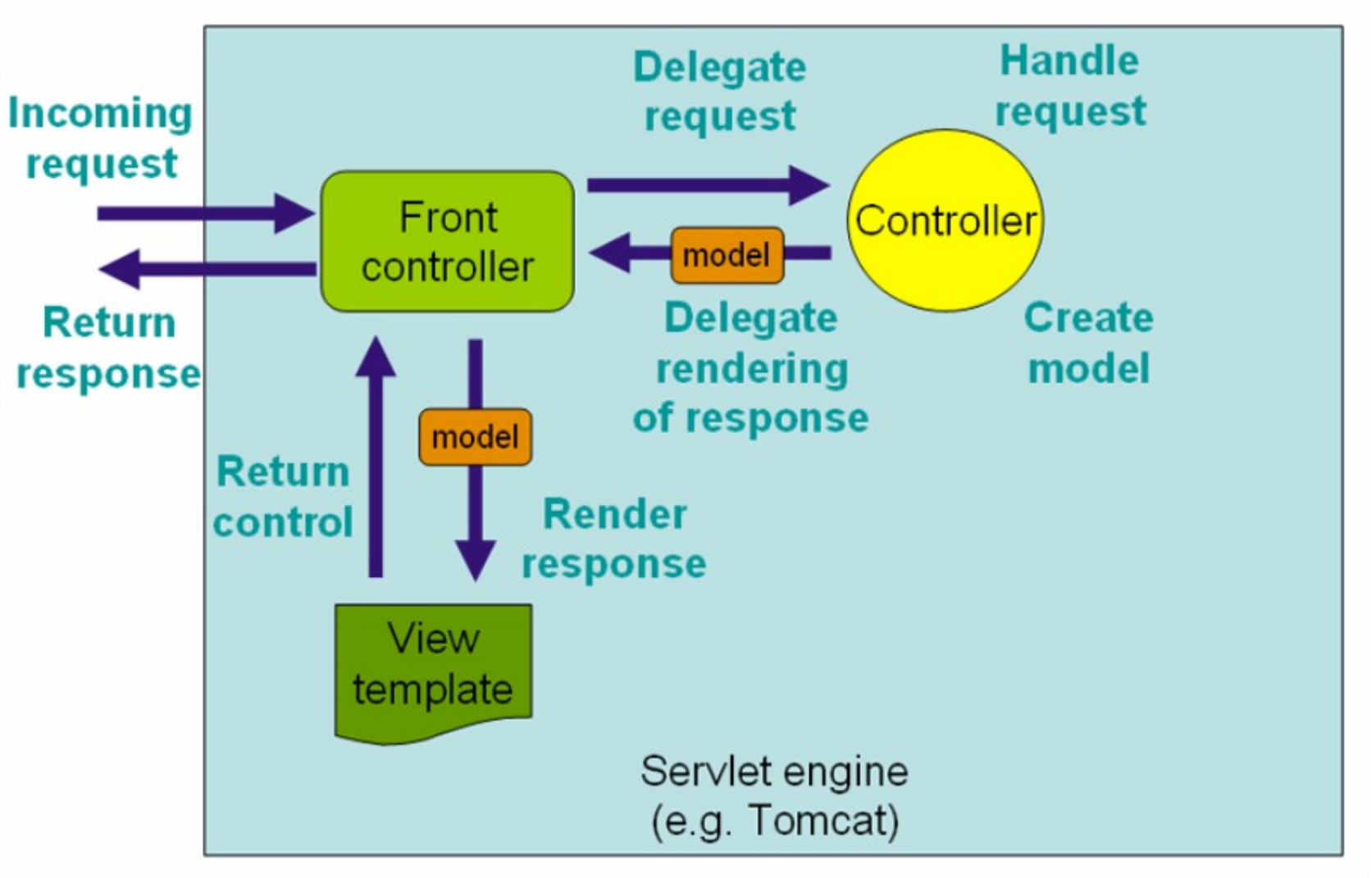
Web 上下文层次


DispatcherServlet
Servlet WebApplicationContext
(containing controllers, view resolvers,
and other web-related beans)
Controllers
ViewResolver
HandlerMapping
Delegates if no bean found
Root WebApplicationContext
(containing middle-tier services, datasources, etc.)
Services
Repositories

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

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

定义映射关系

@Controller

@RequestMapping

- path / method 指定映射路径与方法
- params / headers 限定映射范围
- consumes / produces 限定请求与响应格式

一些快捷方式

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

定义处理方法

- @RequestBody / @ResponseBody / @ResponseStatus
- @PathVariable / @RequestParam / @RequestHeader
- HttpEntity / ResponseEntity
- 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

方法示例一

```
@PostMapping(path = "/", consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
@ResponseStatus(HttpStatus.CREATED)
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);
}
```

方法示例二

complex-controller-demo

```
@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;
}

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

定义校验

- 通过 Validator 对绑定结果进行校验
  - **Hibernate Validator**
- @Valid 注解
- BindingResult

Multipart 上传

more-complex-controller-demo

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

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

视图解析的实现基础

- ViewResolver 与 View 接口
  - **AbstractCachingViewResolver**
  - **UrlBasedViewResolver**
  - **FreeMarkerViewResolver**
  - **ContentNegotiatingViewResolver**
  - **InternalResourceViewResolver**

重定向

- 两种不同的重定向前缀
  - **redirect:**
  - **forward:**

Spring MVC 支持的视图

- 支持的视图列表
  - https://docs.spring.io/spring-framework/docs/5.1.5.RELEASE/spring-framework-reference/web.html#mvc-view
  - **Jackson-based JSON / XML**
  - **Thymeleaf & FreeMarker**

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一篇就够了

Thymeleaf 的一些默认配置

thymeleaf-view-demo

- 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

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

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

controller-cache-demo

```
@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);
}
```

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

Spring MVC 的异常解析

核心接口

- HandlerExceptionResolver

实现类

- SimpleMappingExceptionResolver
- DefaultHandlerExceptionResolver
- ResponseStatusExceptionResolver
- ExceptionHandlerExceptionResolver

异常处理方法

exception-demo

处理方法

- @ExceptionHandler

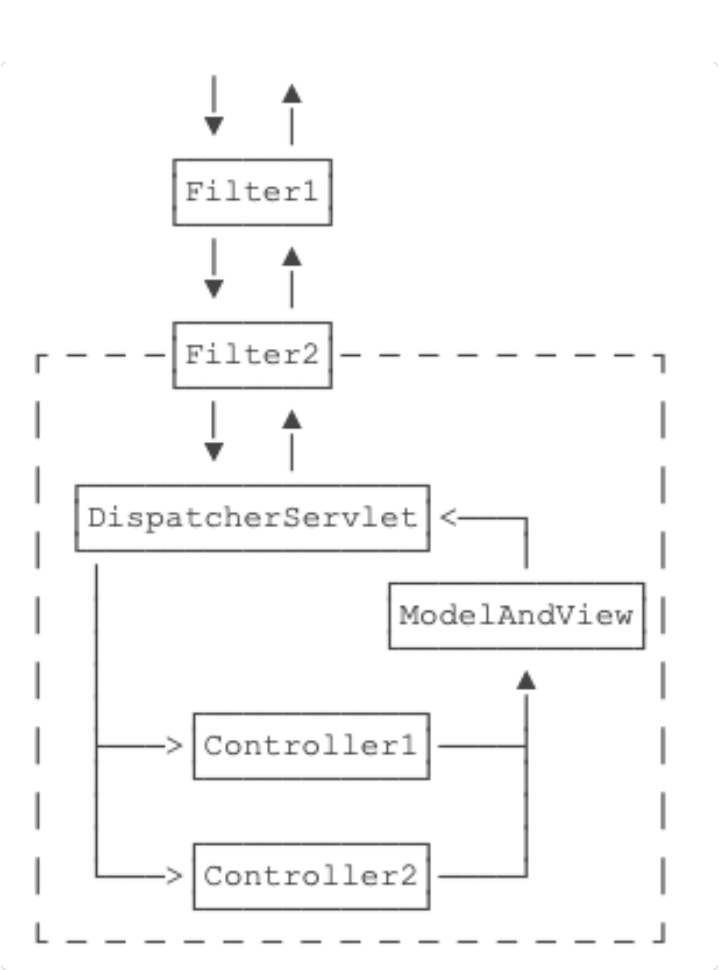
添加位置

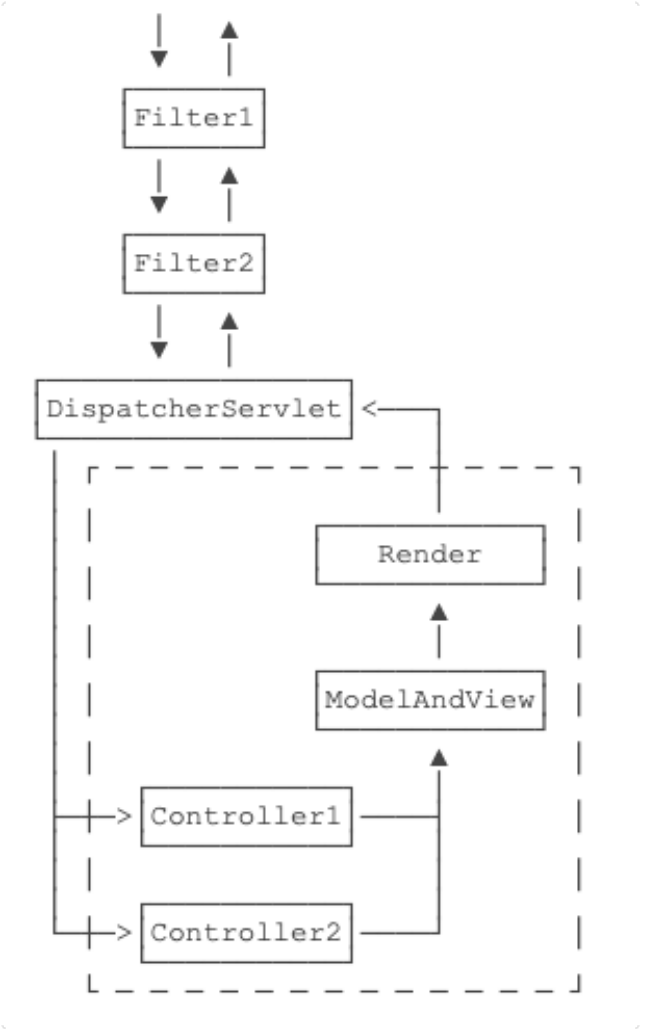
- @Controller / @RestController
- @ControllerAdvice / @RestControllerAdvice

内容提要

- 第一个Spring MVC程序
- 设计Restful Web Service
- Spring 的上下文Context
- 请求处理
- 视图处理
- 静态资源与缓存
- 异常处理
- 拦截器

拦截范围


Filter1
Filter2
DispatcherServlet
ModelAndView
Controller1
Controller2

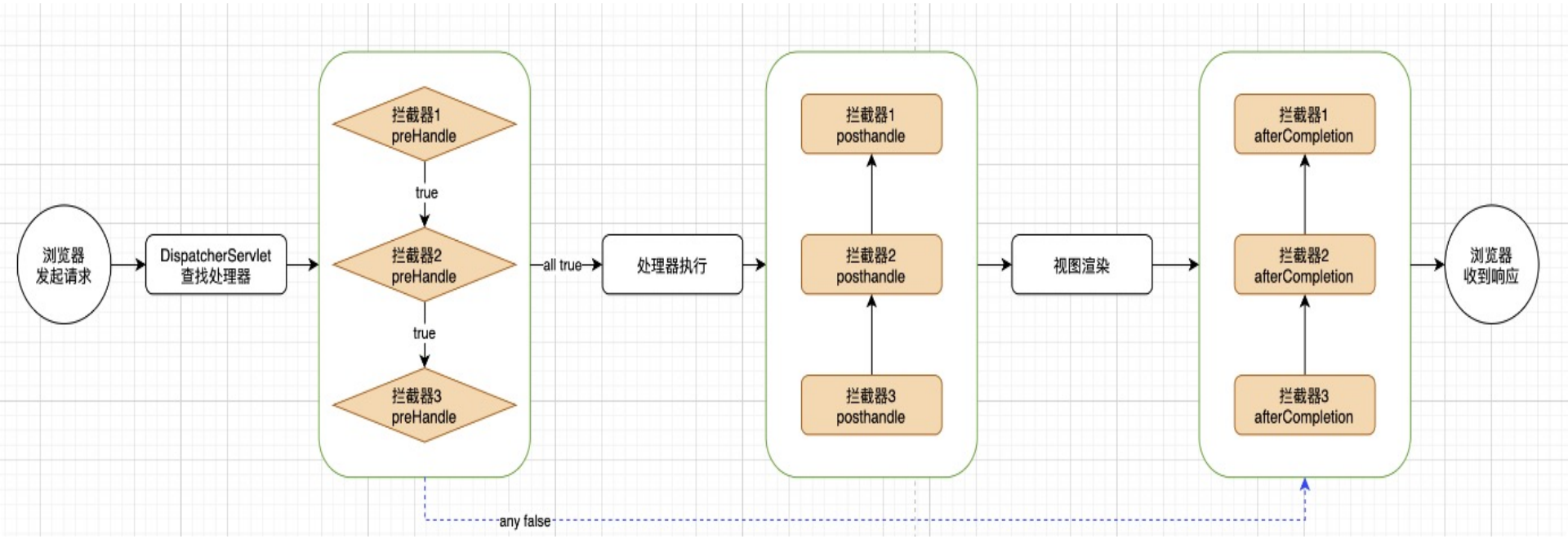

Filter1
Filter2
DispatcherServlet
Render
ModelAndView
Controller1
Controller2

Spring MVC 的拦截器

核心接口

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

拦截器执行顺序


浏览器
发起请求
DispatcherServlet
查找处理器
拦截器1
preHandle
true
拦截器2
preHandle
true
拦截器3
preHandle
all true
处理器执行
拦截器1
posthandle
拦截器2
posthandle
拦截器3
posthandle
视图渲染
拦截器1
afterCompletion
拦截器2
afterCompletion
拦截器3
afterCompletion
浏览器
收到响应
any false

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

interceptor-demo

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）