Web 开发进阶

极客时间

扫码试看/订阅

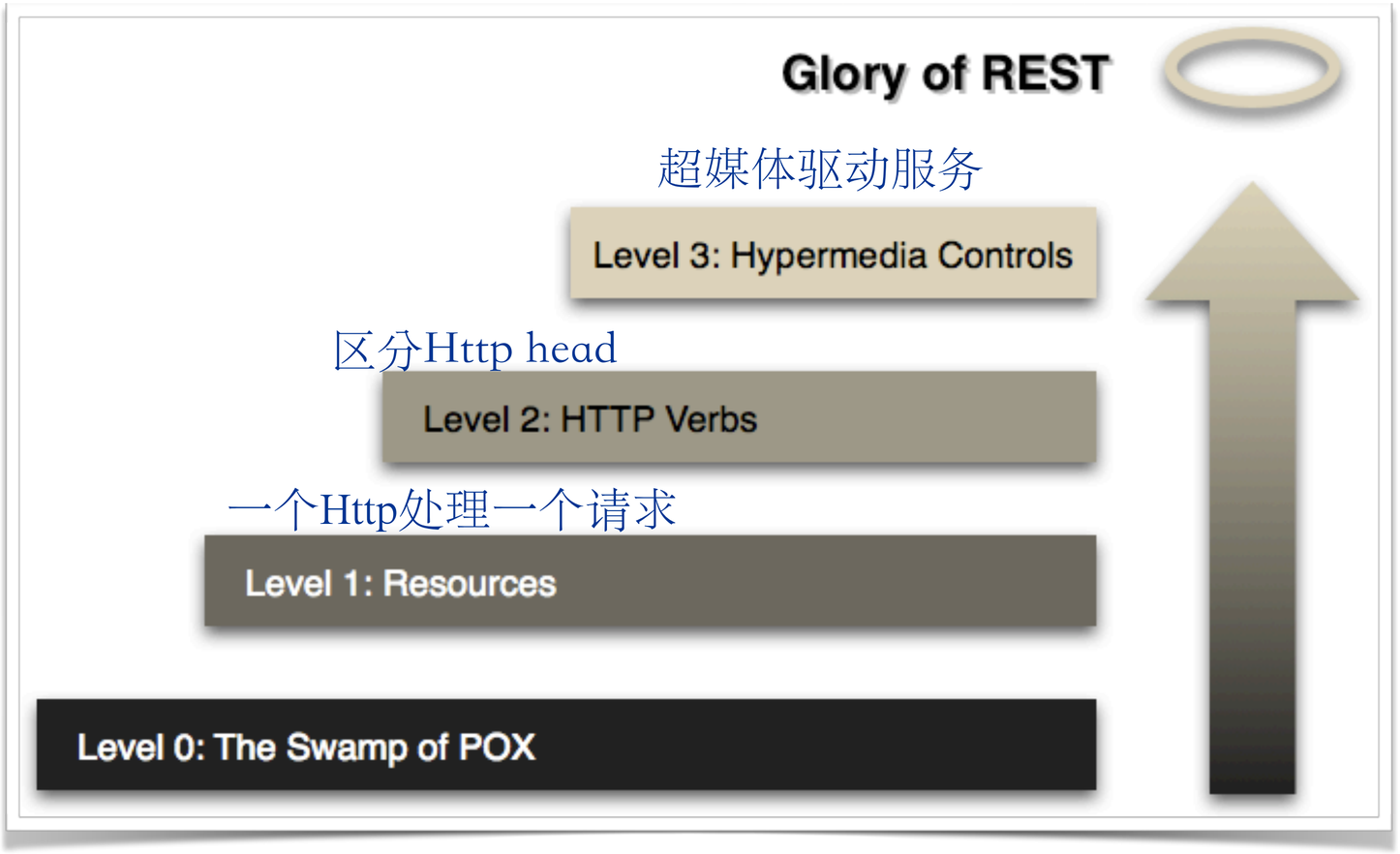
《玩转 Spring 全家桶》

设计好的 RESTful Web Service

“REST提供了一组架构约束，当作为一个整体来应用时，强调组件交互的可伸缩性、接口的通用性、组件的独立部署、以及用来减少交互延迟、增强安全性、封装遗留系统的中间组件。”

- Roy Thomas Fielding

Richardson 成熟度模型



全部都用一个接口，很混乱

如何实现 Restful Web Service

- 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　用JSON, xml还是web页面

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） 比如一个Coffee的集合，一个订单的集合。。。
- 将资源合并为复合资源 比如一个请求一起返回coffee+order的组合资源
- 计算或处理函数 比如上海到北京距离，那些无法找到领域名词的情况

资源的粒度

站在服务端的角度，要考虑

- 网络效率　一次提供尽可能多的信息
- 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

用户注册时间，用户名就可以放到缓存，不会变

构建更好的 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“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)

比如coffee/id

不是所有的库都用,或者；作为分隔符

不用驼峰规则

认识 HTTP 方法

安全：不会改变资源内容
幂等：每次同一个请求返回同一个值，效果一样

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

认识 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

2xx—表示成功
3xx—与缓存，重定向相关
4xx—客户端错误
5xx—服务端错误

选择合适的表述

JSON

- MappingJackson2HttpMessageConverter
- GsonHttpMessageConverter
- JsonbHttpMessageConverter

XML

- MappingJackson2XmlHttpMessageConverter
- Jaxb2RootElementHttpMessageConverter

HTML

ProtoBuf

- ProtobufHttpMessageConverter

什么是 HATEOAS

超媒体驱动

什么是 HATEOAS

Richardson 成熟度模型

- Level 3 - Hypermedia Controls

HATEOAS

- Hybermedia As The Engine Of Application State 超媒体视为应用程序状态引擎
- REST 统一接口的必要组成部分

HATEOAS v.s. WSDL

HATEOAS

- 表述中的超链接会提供服务所需的各种 REST 接口信息
- 无需事先约定如何访问服务 无需事先知道连接的URL是什么，该怎么调用

通过约定好的根路径访问列表，通过这个查找所有资源

传统的服务契约

- 必须事先约定服务的地址与格式

RPC:事先约定好服务接口

HATEOAS 示例

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
    <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
    <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
    <link rel="close" href="https://bank.example.com/accounts/12345/status" />
</account>
```

严格来说要带有一个self的link

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
</account>
```

现在只能充值：根据当前状态判断可以做什么操作

常用的超链接类型

REL	描述
self	指向当前资源本身的链接
edit	指向一个可以编辑当前资源的链接
collection	如果当前资源包含在某个集合中，指向该集合的链接
search	指向一个可以搜索当前资源与其相关资源的链接
related	指向一个与当前资源相关的链接
first	集合遍历相关的类型，指向第一个资源的链接
last	集合遍历相关的类型，指向最后一个资源的链接
previous	集合遍历相关的类型，指向上一个资源的链接
next	集合遍历相关的类型，指向下一个资源的链接

http://www.iana.org/assignments/link-relations/link-relations.xhtml

使用 Spring Data REST 实现简单的超媒体服务

认识 HAL

HAL

- Hypertext Application Language　　基于JSON的扩展
- HAL 是一种简单的格式，为 API 中的资源提供简单一致的链接

HAL 模型

- 链接
- 内嵌资源
- 状态　资源状态

Spring Data REST

Spring Boot 依赖

- spring-boot-starter-data-rest

常用注解与类 做了自动配置

- @RepositoryRestResource 如果有了Repository后自动变成资源
- Resource<T>
- PagedResource<T>

“Talk is cheap, show me the code.”

Chapter 8 / hateoas-waiter-service

如何访问 HATEOAS 服务 (客户端)

配置 Jackson JSON

- 注册 HAL 支持 增强，使之支持HAL

操作超链接

- 找到需要的 Link
- 访问超链接

“Talk is cheap, show me the code.”

Chapter 8 / hateoas-customer-service

分布式环境中如何解决 Session 的问题

常见的会话解决方案

通过Load balance实现。来自一个用户的会话尽可能落到同一台机器上——分布式变成单机。
服务器变化后，原先用户的会话就是失效，用户体验有所下降

- 粘性会话 Sticky Session
- 会话复制 Session Replication
- 集中会话 Centralized Session

每台机器的会话都复制一遍，整个集群所有机器几乎有着相同会话信息，这样请求不管落在哪都可以。 问题是访问量巨大时候，每台机器都复制不太现实；此外每台信息的副本信息可能不一样。不推荐

使用JDBC或者Redis来存储会话信息，只要有相同的JSessionID,就可以把session从集中会话中取出来

认识 Spring Session

Spring Session

- 简化集群中的用户会话管理
- 无需绑定容器特定解决方案

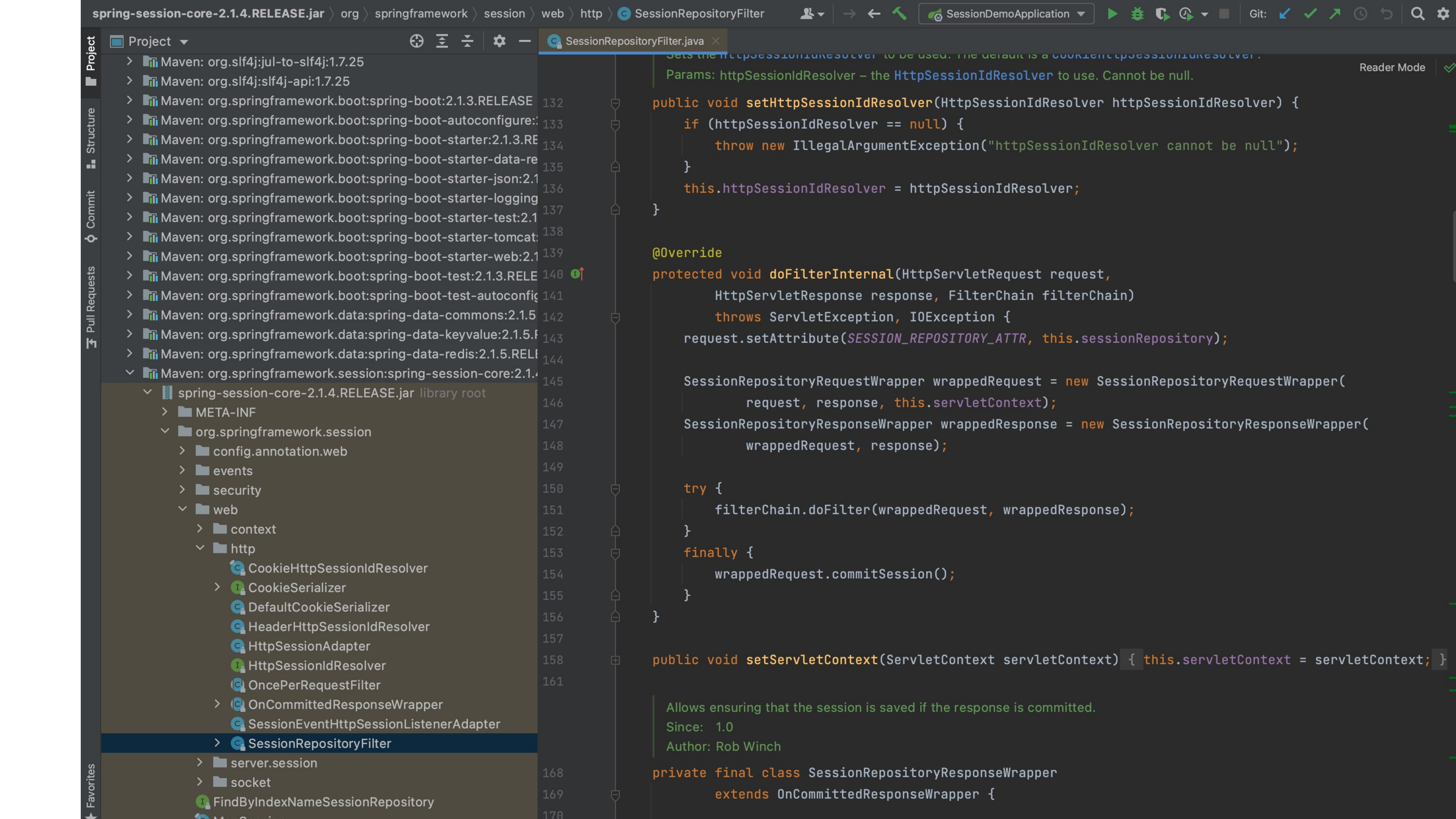
支持的存储

- Redis
- MongoDB
- JDBC
- Hazelcast

实现原理

定制 HttpSession

- 通过定制的 HttpServletRequest 返回定制的 HttpSession 屏蔽后端差异
  - SessionRepositoryRequestWrapper
  - SessionRepositoryFilter
  - DelegatingFilterProxy


spring-session-core-2.1.4.RELEASE.jar
org
springframework
session
web
http
SessionRepositoryFilter
SessionDemoApplication
Git:
Project
SessionRepositoryFilter.java
Reader Mode
Maven: org.slf4j:jul-to-slf4j:1.7.25
Maven: org.slf4j:slf4j-api:1.7.25
Maven: org.springframework.boot:spring-boot:2.1.3.RELEASE
Maven: org.springframework.boot:spring-boot-autoconfigure:
Maven: org.springframework.boot:spring-boot-starter:2.1.3.RE
Maven: org.springframework.boot:spring-boot-starter-data-re
Maven: org.springframework.boot:spring-boot-starter-json:2.1
Maven: org.springframework.boot:spring-boot-starter-logging
Maven: org.springframework.boot:spring-boot-starter-test:2.1
Maven: org.springframework.boot:spring-boot-starter-tomcat
Maven: org.springframework.boot:spring-boot-starter-web:2.1
Maven: org.springframework.boot:spring-boot-test:2.1.3.RELE
Maven: org.springframework.boot:spring-boot-test-autoconfig
Maven: org.springframework.data:spring-data-commons:2.1.5
Maven: org.springframework.data:spring-data-keyvalue:2.1.5.
Maven: org.springframework.data:spring-data-redis:2.1.5.RELE
Maven: org.springframework.session:spring-session-core:2.1.
spring-session-core-2.1.4.RELEASE.jar library root
META-INF
org.springframework.session
config.annotation.web
events
security
web
context
http
CookieHttpSessionIdResolver
CookieSerializer
DefaultCookieSerializer
HeaderHttpSessionIdResolver
HttpSessionAdapter
HttpSessionIdResolver
OncePerRequestFilter
OnCommittedResponseWrapper
SessionEventHttpSessionListenerAdapter
SessionRepositoryFilter
server.session
socket
FindByIndexNameSessionRepository
Params: httpSessionIdResolver – the HttpSessionIdResolver to use. Cannot be null.
132 public void setHttpSessionIdResolver(HttpSessionIdResolver httpSessionIdResolver) {
133 if (httpSessionIdResolver == null) {
134 throw new IllegalArgumentException("httpSessionIdResolver cannot be null");
135 }
136 this.httpSessionIdResolver = httpSessionIdResolver;
137 }
138
139 @Override
140 protected void doFilterInternal(HttpServletRequest request,
141 HttpServletResponse response, FilterChain filterChain)
142 throws ServletException, IOException {
143 request.setAttribute(SESSION_REPOSITORY_ATTR, this.sessionRepository);
144
145 SessionRepositoryRequestWrapper wrappedRequest = new SessionRepositoryRequestWrapper(
146 request, response, this.servletContext);
147 SessionRepositoryResponseWrapper wrappedResponse = new SessionRepositoryResponseWrapper(
148 wrappedRequest, response);
149
150 try {
151 filterChain.doFilter(wrappedRequest, wrappedResponse);
152 }
153 finally {
154 wrappedRequest.commitSession();
155 }
156 }
157
158 public void setServletContext(ServletContext servletContext) { this.servletContext = servletContext; }
161
Allows ensuring that the session is saved if the response is committed.
Since: 1.0
Author: Rob Winch
168 private final class SessionRepositoryResponseWrapper
169 extends OnCommittedResponseWrapper {

基于 Redis 的 HttpSession

引入依赖

- spring-session-data-redis

基本配置

- @EnableRedisHttpSession
- 提供 RedisConnectionFactory
- 实现 AbstractHttpSessionApplicationInitializer
  - 配置 DelegatingFilterProxy

Spring Boot 对 Spring Session 的支持

application.properties

- spring.session.store-type=redis　如果只有Redis的话，可以不写这句话
- spring.session.timeout=
  - server.servlet.session.timeout=
- spring.session.redis.flush-mode=on-save
- spring.session.redis.namespace=spring:session

“Talk is cheap, show me the code.”

Chapter 8 / session-demo

使用 WebFlux 代替 Spring MVC

认识 WebFlux

什么是 WebFlux

- 用于构建基于 Reactive 技术栈之上的 Web 应用程序
- 基于 Reactive Streams API，运行在非阻塞服务器上 Jetty或者Netty

为什么会有 WebFlux 默认使用Netty,大并发量时候希望通过非阻塞减少开销

- 对于非阻塞 Web 应用的需要
- 函数式编程

认识 WebFlux

关于 WebFlux 的性能

- 请求的耗时并不会有很大的改善
- 仅需少量固定数量的线程和较少的内存即可实现扩展

  支持同样的并发

WebMVC v.s. WebFlux

- 已有 Spring MVC 应用，运行正常，就别改了

比如MySQL数据库，即使有R2DBC,也不太建议

- 依赖了大量阻塞式持久化 API 和网络 API，建议使用 Spring MVC
- 已经使用了非阻塞技术栈，可以考虑使用 WebFlux
- 想要使用 Java 8 Lambda 结合轻量级函数式框架，可以考虑 WebFlux

WebFlux 中的编程模型

两种编程模型

- 基于注解的控制器　和SpringMVC几乎一样
- 函数式 Endpoints　　函数式路由

基于注解的控制器

常用注解

- @Controller
- @RequestMapping 及其等价注解
- @RequestBody / @ResponseBody

返回值

多个返回

- Mono<T> / Flux<T>

0/1个返回

“Talk is cheap, show me the code.”

Chapter 8 / webflux-waiter-service

SpringBucks 进度小结

本章小结

- 了解了如何让 API 更符合 REST 风格
- 了解了 HATEOAS 的基本概念
- 了解了 Spring Data REST 的基本用法
- 了解了分布式会话的相关知识
- 了解了 WebFlux 的相关知识

SpringBucks 进度小结

waiter-service

- 更换为 HATEOAS 风格的服务
- 使用 WebFlux 代替了 WebMVC

customer-service

- 更换为 HATEOAS 风格的服务调用方式

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》