

SpringBoot2.0 笔记

1、零基础快速入门 SpringBoot2.0

1、SpringBoot2.x 课程全套介绍和高手系列知识点

简介：介绍 SpringBoot2.x 课程大纲章节
java 基础，jdk 环境，maven 基础

2、SpringBoot2.x 依赖环境和版本新特性说明

简介：讲解新版本依赖环境和 springboot2 新特性概述

- 1、依赖版本 jdk8 以上, Springboot2.x 用 JDK8, 因为底层是 Spring framework5,
- 2、安装 maven 最新版本, maven3.2 以上版本, 下载地址：
<https://maven.apache.org/download.cgi>
- 3、Eclipse 或者 IDE
- 4、新特性
- 5、翻译工具：<https://translate.google.cn/>
- 6、springbootGitHub 地址：<https://github.com/spring-projects/spring-boot>
- 7、springboot 官方文档：<https://spring.io/guides/gs/spring-boot/>

3、快速创建 SpringBoot2.x 应用之手工创建 web 应用

简介：使用 Maven 手工创建 SpringBoot2.x 应用

手工创建:<https://projects.spring.io/spring-boot/#quick-start>

官方推荐包命名接口，不要使用默认 defaultPackage

官方文档:

[https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#using-boot-using-the-default-package)

SNAPSHOT/reference/htmlsingle/#using-boot-using-the-default-package

例子:

```
com
+- example
  +- myapplication
    +- Application.java
    |
    +- customer
      +- Customer.java
      +- CustomerController.java
      +- CustomerService.java
```

```
|   +- CustomerRepository.java
|
+- order
    +- Order.java
    +- OrderController.java
    +- OrderService.java
    +- OrderRepository.java
```

4、快速创建 SpringBoot2.x 应用

简介：使用构建工具自动生成项目基本架构

1、工具自动创建:<http://start.spring.io/>

5、SpringBoot2.x 的依赖默认 Maven 版本

简介：讲解 SpringBoot2.x 的默认 Maven 依赖版本

1、官网地址

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#appendix-dependency-versions>

2、SpringBoot 接口 Http 协议开发实战

1、SpringBoot2.xHTTP 请求配置讲解

简介：SpringBoot2.xHTTP 请求注解讲解和简化注解配置技巧

1、@RestController and @RequestMapping 是 springMVC 的注解，不是 springboot 特有的

2、@RestController = @Controller+@ResponseBody

3@SpringBootApplication=@Configuration+@EnableAutoConfiguration+@ComponentScan

访问路径：localhost:8080

2、开发必备工具 PostMan 接口工具介绍和使用

简介：模拟 Http 接口测试工具 PostMan 安装和讲解

1、接口调试工具安装和基本使用

2、下载地址：<https://www.getpostman.com/>

3、SpringBoot 基础 HTTP 接口 GET 请求实战

简介:讲解 springboot 接口, http 的 get 请求, 各个注解使用

1、GET 请求

1、单一参数@RequestMapping(path =("/{id}", method = RequestMethod.GET)

1) public String getUser(@PathVariable String id) {}

2) @RequestMapping(path="/{depId}/{userId}", method=RequestMethod.GET) 可以同时指定多个提交方法

getUser(@PathVariable("depId")String departmentID, @PathVariable("userId") String userId)

3) 一个顶俩

@GetMapping = @RequestMapping(method = RequestMethod.GET)

@PostMapping = @RequestMapping(method = RequestMethod.POST)

@PutMapping = @RequestMapping(method = RequestMethod.PUT)

@DeleteMapping=@RequestMapping(method = RequestMethod.DELETE)

4) @RequestParam(value = "name", required = true)

可以设置默认值, 比如分页

5) @RequestBody 请求体映射实体类

需要指定 http 头为 content-type 为 application/json charset=utf-8

6) @RequestHeader 请求头, 比如鉴权

@RequestHeader("access_token") String accessToken

7) HttpServletRequest request 自动注入获取参数

4、SpringBoot 基础 HTTP 其他提交方法请求实战

简介: 讲解 http 请求 post, put, delete 提交方式

5、常用 json 框架介绍和 Jackson 返回结果处理

简介: 介绍常用 json 框架和注解的使用, 自定义返回 json 结构和格式

1、常用框架 阿里 fastjson, 谷歌 gson 等

JavaBean 序列化为 Json, 性能: Jackson > FastJson > Gson > Json-lib 同个结构

Jackson、FastJson、Gson 类库各有优点, 各有自己的专长

空间换时间, 时间换空间

2、jackson 处理相关自动

指定字段不返回: @JsonIgnore

指定日期格式:

@JsonFormat(pattern="yyyy-MM-dd hh:mm:ss", locale="zh", timezone="GMT+8")

空字段不返回: @JsonInclude(Include.NON_NULL)

指定别名: @JsonProperty

6、SpringBoot2.x 目录文件结构讲解

简介：讲解 SpringBoot 目录文件结构和官方推荐的目录规范

1、目录讲解

src/main/java：存放代码

src/main/resources

static: 存放静态文件，比如 css、js、image,
(访问方式 http://localhost:8080/js/main.js)

templates:存放静态页面 jsp,html,tpl

config:存放配置文件,application.properties

resources:

2、引入依赖 Thymeleaf

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-thymeleaf</artifactId>
```

```
</dependency>
```

注意：如果不引入这个依赖包，html 文件应该放在默认加载文件夹里面，
比如 resources、static、public 这几个文件夹，才可以访问

3、同个文件的加载顺序,静态资源文件

Spring Boot 默认会挨个从

META/resources > resources > static > public 里面找是否存在相应的资源，如果有
则直接返回。

4、默认配置

1) 官网地址：

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-static-content>

2:spring.resources.staticlocations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/

5、静态资源文件存储在 CDN

7、SpringBoot2.x 文件上传实战

简介：讲解 HTML 页面文件上传和后端处理实战

1、讲解 springboot 文件上传 MultipartFile file，源自 SpringMVC

1) 静态页面直接访问：localhost:8080/index.html

注意点：

如果想要直接访问 html 页面，则需要把 html 放在 springboot 默认
加载的文件夹下面

2) MultipartFile 对象的 transferTo 方法，用于文件保存（效率和操作比原先用
FileOutputStream 方便和高效）

访问路径：

http://localhost:8080/images/39020dbb-9253-41b9-8ff9-403309ff3f19.jpeg

8、jar 包方式运行 web 项目文件上传和访问（核心知识）

简介：讲解 SpringBoot2.x 使用 java -jar 运行方式的图片上传和访问处理

1、文件大小配置，启动类里面配置

```
@Bean
public MultipartConfigElement multipartConfigElement() {
    MultipartConfigFactory factory = new MultipartConfigFactory();
    //单个文件最大
    factory.setMaxFileSize("10240KB"); //KB,MB
    /// 设置总上传数据总大小
    factory.setMaxRequestSize("1024000KB");
    return factory.createMultipartConfig();
}
```

在这里需要注意 SpringBoot 的版本，不同版本的设置是不一样的，目前的学习环境是 2.1.0

2、打包成 jar 包，需要增加 maven 依赖

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

如果没加相关依赖，执行 maven 打包，运行后会报错:no main manifest attribute, in XXX.jar，意思是找不到 manifest 文件（指定启动类等配置文件）

GUI：反编译工具，作用就是用于把 class 文件转换成 java 文件

3、文件上传和访问需要指定磁盘路径

application.properties 中增加下面配置

```
1) web.images-path=/Users/jack/Desktop
2) spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/,classpath:/test/,file:${web.upload-path}
```

4、文件服务器：fastdfs，阿里云 oss，nginx 搭建一个简单的文件服务器

3、SpringBoot 热部署 devtool 和配置文件自动注入实战

1、SpringBoot2.x 使用 Dev-tool 热部署

简介:介绍什么是热部署，使用 springboot 结合 dev-tool 工具，快速加载启动应用
官方地址：

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD->

SNAPSHOT/reference/htmlsingle/#using-boot-devtools

核心依赖包:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <optional>true</optional>
</dependency>
```

添加依赖后, 在 ide 里面重启应用, 后续修改后马上可以生效
classloader

不被热部署的文件

1、/META-INF/maven, /META-INF/resources, /resources, /static, /public, or /templates

2、指定文件不进行热部署 spring.devtools.restart.exclude=static/**,public/**

3、手工触发重启 spring.devtools.restart.trigger-file=trigger.txt

改代码不重启, 通过一个文本去控制

[https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#using-boot-devtools-restart-exclude)

[SNAPSHOT/reference/htmlsingle/#using-boot-devtools-restart-exclude](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#using-boot-devtools-restart-exclude)

注意点: 生产环境不要开启这个功能, 如果用 java -jar 启动, springBoot 是不会进行热部署的

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <optional>true</optional>
</dependency>
```

2、SpringBoot2.x 配置文件讲解

简介: SpringBoot2.x 常见的配置文件 xml、yaml、properties 的区别和使用

xml、properties、json、yaml

1、常见的配置文件 xx.yaml, xx.properties,

1)YAML (Yet Another Markup Language)

写 YAML 要比写 XML 快得多(无需关注标签或引号)

使用空格 Space 缩进表示分层, 不同层次之间的缩进可以使用不同的空格数目

注意: key 后面的冒号, 后面一定要跟一个空格, 树状结构

application.properties 示例

```
server.port=8090
server.session-timeout=30
server.tomcat.max-threads=0
server.tomcat.uri-encoding=UTF-8
```

application.yaml 示例

```
server:
  port: 8090
  session-timeout: 30
```

```
tomcat.max-threads: 0
tomcat.uri-encoding: UTF-8
```

2、默认示例文件仅作为指导。不要将整个内容复制并粘贴到您的应用程序中，只挑选您需要的属性。

3、参考：

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD->

SNAPSHOT/reference/htmlsingle/#common-application-properties

如果需要修改，直接复制对应的配置文件，加到 application.properties 里面

3、SpringBoot 注解配置文件自动映射到属性和实体类实战

简介：讲解使用@value 注解配置文件自动映射到属性和实体类

1、配置文件加载

方式一

1、Controller 上面配置

```
@PropertySource({"classpath:resource.properties"})
```

2、增加属性

```
@Value("${test.name}")
private String name;
```

方式二：实体类配置文件

步骤：

- 1、添加 @Component 注解；
- 2、使用 @PropertySource 注解指定配置文件位置；
- 3、使用 @ConfigurationProperties 注解，设置相关属性；
- 4、必须 通过注入 IOC 对象 Resource 进来，才能在类中使用获取的配置文件值。

```
@Autowired
private ServerSettings serverSettings;
```

例子：

```
@Configuration
@ConfigurationProperties(prefix="test")
@PropertySource(value="classpath:resource.properties")
public class ServerConstant {
```

常见问题：

- 1、配置文件注入失败，Could not resolve placeholder
解决：根据 springboot 启动流程，会有自动扫描包没有扫描到相关注解，
默认 Spring 框架实现会从声明 @ComponentScan 所在的类的 package 进行扫描，来自动注入，
因此启动类最好放在根路径下面，或者指定扫描包范围
spring-boot 扫描启动类对应的目录和子目录
- 2、注入 bean 的方式，属性名称和配置文件里面的 key 一一对应，就用加 @Value 这个注解
如果不一样，就要加@value("\${XXX}")

3、当使用前缀的方式时，就不能再使用@Value("\${name}"), 而是直接使用字段名进行匹配

```
@Component
@PropertySource({"classpath:application.properties"})
@ConfigurationProperties(prefix = "test")
public class ServiceSetting {
    //@Value("${name}")
    private String name;
    //@Value("${domain}")
    private String domain;
    public ServiceSetting() {
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDomain() {
        return domain;
    }
    public void setDomain(String domain) {
        this.domain = domain;
    }
}
```

4、如果字段名和配置的 key 一致，那么可以不使用@Value，直接使用字段名进行匹配（前提是字段名不冲突）

5、如下所示，如果 application.properties 中也可以匹配到相应的属性，那么就不会使用 resource.properties

```
@Component
@PropertySource({"classpath:resource.properties"})
@ConfigurationProperties(prefix = "test")
public class ServiceSetting {
    //@Value("${name}")
    private String name;
    //@Value("${domain}")
    private String domain;
    public ServiceSetting() {
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```



```

    }
    public String getDomain() {
        return domain;
    }
    public void setDomain(String domain) {
        this.domain = domain;
    }
}

```

4、Springboot2.0 单元测试进阶实战和自定义异常处理

1、SpringBootTest 单元测试实战

简介：讲解 SpringBoot 的单元测试

1、引入相关依赖

<!--springboot 程序测试依赖，如果是自动创建项目默认添加-->

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

```

2、使用

```

@RunWith(SpringRunner.class) //底层用 junit  SpringJUnit4ClassRunner
@SpringBootTest(classes={XdclassApplication.class})//启动整个 springboot 工程
//XdclassApplication 这个是启动类
public class SpringBootTests {}
现在不指定 classes 也可以
@RunWith(SpringRunner.class)// 存在版本问题，版本不一致不需要指定 classes
@SpringBootTest

```

2、SpringBoot 测试进阶高级篇之 MockMvc 讲解

简介:讲解 MockMvc 类的使用和模拟 Http 请求实战

1、增加类注解 @AutoConfigureMockMvc

2、相关 API

```

perform: 执行一个 RequestBuilder 请求
andExpect: 添加 ResultMatcher->MockMvcResultMatchers 验证规则
andReturn: 最后返回相应的 MvcResult->Response

```

3、SpringBoot 个性化启动 banner 设置和 debug 日志

简介：自定义应用启动的趣味性日志图标和查看调试日志

- 1、启动获取更多信息 `java -jar xxx.jar --debug`
- 2、修改启动的 banner 信息
 - 1) 在类路径下增加一个 banner.txt，里面是启动要输出的信息
 - 2) 在 applicatoin.properties 增加 banner 文件的路径地址
`spring.banner.location=banner.txt`
 - 3) 官 网 地 址 <https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-banners>

4、SpringBoot2.x 配置全局异常实战

讲解：服务端异常讲解和 SpringBoot 配置全局异常实战

- 1、默认异常测试 `int i = 1/0`，不友好
- 2、异常注解介绍
 - @ControllerAdvice 如果是返回 json 数据 则用 RestControllerAdvice,就可以不加 @ResponseBody
 - //捕获全局异常,处理所有不可知的异常
 - @ExceptionHandler(value=Exception.class)

5、SpringBoot2.x 配置全局异常返回自定义页面

简介：使用 SpringBoot 自定义异常和错误页面跳转实战

- 1、返回自定义异常界面，需要引入 thymeleaf 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```
- 2、resource 目录下新建 templates,并新建 error.html

```
ModelAndView modelAndView = new ModelAndView();
modelAndView.setViewName("error.html");
modelAndView.addObject("msg", e.getMessage());
return modelAndView;
```

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-error-handling>

5、SpringBoot 部署 war 项目到 tomcat9 和启动原理讲解

1、SpringBoot 启动方式讲解和部署 war 项目到 tomcat9

简介：SpringBoot 常见启动方式讲解和部署 war 项目 Tomcat

1、ide 启动

2、jar 包方式启动

maven 插件:

```
<build>
```

```
<plugins>
```

```
<plugin>
```

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-maven-plugin</artifactId>
```

```
</plugin>
```

```
</plugins>
```

```
</build>
```

如果没有加，则执行 jar 包，报错如下

```
java -jar spring-boot-demo-0.0.1-SNAPSHOT.jar
```

no main manifest attribute, in spring-boot-demo-0.0.1-SNAPSHOT.jar

如果有安装 maven 用 mvn spring-boot:run

项目结构

example.jar

|

+ -META-INF

| + -MANIFEST.MF

+ -org

| + -springframework

| + -boot

| + -loader

| + -<spring boot loader classes>

+ -BOOT-INF

+ -classes

| + -mycompany

| + -project

| + -YourClasses.class

+ -lib

+ -dependency1.jar

+ -dependency2.jar

目录结构讲解

[https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#executable-jar-jar-file-structure)

SNAPSHOT/reference/htmlsingle/#executable-jar-jar-file-structure

3、war 包方式启动

1)在 pom.xml 中将打包形式 jar 修改为 war <packaging>war</packaging>

构建项目名称 <finalName>xdclass_springboot</finalName>

<!-- 打包必须要有的插件 -->

<build>

<finalName>start</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

2)tomcat 下载 <https://tomcat.apache.org/download-90.cgi>

3)修改启动类

```
public class XdclassApplication extends SpringBootServletInitializer {  
    @Override  
    protected SpringApplicationBuilder configure(SpringApplicationBuilder  
application) {  
        return application.sources(XdclassApplication.class);  
    }  
    public static void main(String[] args) throws Exception {  
        SpringApplication.run(XdclassApplication.class, args);  
    }  
}
```

4)打包项目，启动 tomcat

2、启动容器介绍和第三方测试数据讲解

使用 Jmeter 测试工具测试性能，QPS,TPS，RT

<https://examples.javacodegeeks.com/enterprise-java/spring/tomcat-vs-jetty-vs-undertow-comparison-of-spring-boot-embedded-servlet-containers/>

2、SpringBoot2.x 启动原理概述

简介：讲解 SpringBoot 启动流程概述和基本加载案例

6、SpringBoot 拦截器实战和 Servlet3.0 自定义 Filter、Listener

1、深入 SpringBoot 过滤器和 Servlet3.0 配置过滤器实战

简介:讲解 SpringBoot 里面 Filter 讲解和使用 Servlet3.0 配置自定义 Filter 实战

filter 简单理解：人--->检票员（filter）---> 景点

1、SpringBoot 启动默认加载的 Filter

characterEncodingFilter

hiddenHttpMethodFilter

httpPutFormContentFilter

requestContextFilter

2、Filter 优先级

Ordered.HIGHEST_PRECEDENCE

Ordered.LOWEST_PRECEDENCE

低位值意味着更高的优先级 Higher values are interpreted as lower priority

自定义 Filter，避免和默认的 Filter 优先级一样，不然会冲突

注册 Filter 的 bean FilterRegistrationBean

同模块里面有相关默认 Filter

web->servlet->filter

3、自定义 Filter

1) 使用 Servlet3.0 的注解进行配置

2) 启动类里面增加 @ServletComponentScan，进行扫描

3) 新建一个 Filter 类，implements Filter，并实现对应的接口

4) @WebFilter 标记一个类为 filter，被 spring 进行扫描

urlPatterns：拦截规则，支持正则

@WebFilter(urlPatterns = {"/login", ""})

6) 控制 chain.doFilter 的方法的调用，来实现是否通过放行

不放行，web 应用 resp.sendRedirect("/index.html");页面跳转使用转发和重定向

场景：权限控制、用户登录(适合非前端后端分离场景)等

```
package com.jian.start.filter;
```

```
import javax.servlet.*;
```

```
import javax.servlet.annotation.WebFilter;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.IOException;
```

```
@WebFilter(urlPatterns = {"/login", ""})
```

```
public class LoginFilter implements Filter {
```

```
    @Override
```

```
    public void init(FilterConfig filterConfig) throws ServletException {
```

```
        //容器初始化的时候执行
```

```
        System.out.println("LoginFilter init ...");
```

```
    }
```

```
    @Override
```

```
    public void doFilter(ServletRequest servletRequest, ServletResponse
```

```
servletResponse, FilterChain filterChain) throws IOException, ServletException {
```

```
        //当请求匹配到，被拦截时执行
```

```
        System.out.println("LoginFilter doing ...");
```

```
        filterChain.doFilter((HttpServletRequest)servletRequest,
```

```
(HttpServletResponse)servletResponse);
```

```
    }
```

```
    @Override
```

```
    public void destroy() {
```

```

        //当容器销毁之前执行
        System.out.println("LoginFilter destory ...");
    }
}

```

1、官网地址：<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-embedded-container-servlets-filters-listeners>

2、Servlet3.0 的注解原生 Servlet 实战

必须：启动类里面增加 @ServletComponentScan，进行扫描

讲解：使用 Servlet3.0 的注解自定义原生 Servlet 和 Listener

1、自定义原生 Servlet

```

@WebServlet(name = "LoginServlet", urlPatterns = {"/api/v1/login"})
public class UserServlet extends HttpServlet{
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        resp.getWriter().print("custom sevlet");
        resp.getWriter().flush();
        resp.getWriter().close();
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        this.doGet(req, resp);
    }
}

```

3、Servlet3.0 的注解原生 Listener 监听器实战

必须：启动类里面增加 @ServletComponentScan，进行扫描

简介：监听器介绍和 Servlet3.0 的注解自定义原生 Listener 监听器实战

1、自定义 Listener(常用的监听器 servletContextListener、httpSessionListener、servletRequestListener)

```

@WebListener
public class RequestListener implements ServletRequestListener {
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        // TODO Auto-generated method stub
        System.out.println("====requestDestroyed====");
    }
    @Override

```

```

public void requestInitialized(ServletRequestEvent sre) {
    System.out.println("=====requestInitialized=====");
}

package com.jian.start.servlet.listener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
@WebListener
public class CustomContextListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("ServletContextListener init");
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("ServletContextListener destory");
    }
}

```

4、SpringBoot2.X 拦截器实战及新旧配置对比

简介：讲解拦截器使用，Springboot2.x 新版本配置拦截器和旧版本 SpringBoot 配置拦截器区别讲解

1、@Configuration

继承 WebMvcConfigurationAdapter(SpringBoot2.X 之前旧版本)

SpringBoot2.X 新版本配置拦截器 implements WebMvcConfigurer

2、自定义拦截器 HandlerInterceptor

preHandle：调用 Controller 某个方法之前

postHandle：Controller 之后调用，视图渲染之前，如果控制器 Controller 出现了异常，则不会执行此方法

afterCompletion：不管有没有异常，这个 afterCompletion 都会被调用，用于资源清理

3、按照注册顺序进行拦截，先注册，先被拦截（拦截器的拦截先后顺序）

```
package com.jian.start.interceptor;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
```

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
```

```
import java.util.Arrays;
```

```
/**
```

```
 * 拦截器的注册中心
```

```
 */
```

```
@Configuration
```

```

public class HandlerInterceptor implements WebMvcConfigurer {
    @Autowired
    private LoginInterceptor loginInterceptor;
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        String[] list = {"", ""};
        registry.addInterceptor(loginInterceptor).addPathPatterns("/**");
        //registry.addInterceptor(loginInterceptor).addPathPatterns(Arrays.asList(list));
        WebMvcConfigurer.super.addInterceptors(registry);
    }
}

package com.jian.start.interceptor;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@Component
public class LoginInterceptor extends HandlerInterceptorAdapter {
    public LoginInterceptor() {
        super();
    }
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        System.out.println("调用拦截器之前");
        return super.preHandle(request, response, handler);
    }
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("调用拦截器之后，视图渲染之前");
        super.postHandle(request, response, handler, modelAndView);
    }
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        System.out.println("调用完成之后");
        super.afterCompletion(request, response, handler, ex);
    }
    @Override
    public void afterConcurrentHandlingStarted(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
        System.out.println("afterConcurrentHandlingStarted ...");
    }
}

```



```

        super.afterConcurrentHandlingStarted(request, response, handler);
    }
}

```

拦截器不生效常见问题:

- 1) 是否有加@Configuration
- 2) 拦截路径是否有问题 ** 和 *
- 3) 拦截器最后路径一定要 “/**”, 如果是目录的话则是 /*/ 示例: /api/*/**

Filter

是基于函数回调 doFilter(), 而 Interceptor 则是基于 AOP 思想

Filter 只在 Servlet 前后起作用, 而 Interceptor 够深入到方法前后、异常抛出前后等

依赖于 Servlet 容器即 web 应用中, 而 Interceptor 不依赖于 Servlet 容器所以可以运行在多种环境。

在接口调用的生命周期里, Interceptor 可以被多次调用, 而 Filter 只能在容器初始化时调用一次。

Filter 和 Interceptor 的执行顺序

过滤前->拦截前->action 执行->拦截后->过滤后

7、SpringBoot 整合模板引擎 Freemaker、thymeleaf

1、SpringBoot Starter 讲解

简介: 介绍什么是 SpringBoot Starter 和主要作用

1、官网地址: <https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#using-boot-starter>

2、starter 主要简化依赖用的

spring-boot-starter-web -> 里面包含多种依赖

3、几个常用的 starter

```

spring-boot-starter-activemq
spring-boot-starter-aop
spring-boot-starter-data-redis
spring-boot-starter-freemarker
spring-boot-starter-thymeleaf
spring-boot-starter-webflux

```

2、SpringBoot2.x 常见模板引擎讲解和官方推荐使用

简介: 介绍常用的 SpringBoot2.x 模板引擎和官方推荐案例

1、JSP (后端渲染, 消耗性能)

Java Server Pages 动态网页技术, 由应用服务器中的 JSP 引擎来编译和执行, 再将生成的整个页面返回给客户端

可以写 java 代码

持表达式语言 (el、jstl)

内建函数

JSP->Servlet(占用 JVM 内存)permSize

javaweb 官方推荐

springboot 不推荐 <https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-jsp-limitations>

2、Freemarker

FreeMarker Template Language (FTL) 文件一般保存为 xxx.ftl

严格依赖 MVC 模式, 不依赖 Servlet 容器 (不占用 JVM 内存)

内建函数

3、Thymeleaf (主推)

轻量级的模板引擎 (负责逻辑业务的不推荐, 解析 DOM 或者 XML 会占用多的内存)

可以直接在浏览器中打开且正确显示模板页面

直接是 html 结尾, 直接编辑

xdlcass.net/user/userinfo.html

社会工程学 伪装

3、SpringBoot 整合模板引擎 freemarker 实战

简介: SpringBoot2.x 整合模板引擎 freemarker 实战

1、Freemarker 相关 maven 依赖

<!-- 引入 freemarker 模板引擎的依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-freemarker</artifactId>

</dependency>

2、Freemarker 基础配置

是否开启 freemarker 缓存,本地为 false, 生产建议为 true

spring.freemarker.cache=false

spring.freemarker.charset=UTF-8

spring.freemarker.allow-request-override=false

spring.freemarker.check-template-location=true

#类型

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=true

spring.freemarker.expose-session-attributes=true

#文件后缀

spring.freemarker.suffix=.ftl

#路径

spring.freemarker.template-loader-path=classpath:/templates/

3、建立文件夹

1)src/main/resources/templates/fm/user/

2)建立一个 index.ftl

- 3)user 文件夹下面建立一个 user.html
- 4、简单测试代码编写和访问

4、SpringBoot2 整合模板引擎 thymeleaf 实战

讲解：SpringBoot2.x 整合模板引擎 thymeleaf 实战

官网地址：<https://www.thymeleaf.org/doc/articles/thymeleaf3migration.html>

1、thymeleaf 相关 maven 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2、thymeleaf 基础配置

#开发时关闭缓存,不然没法看到实时页面

spring.thymeleaf.cache=false

spring.thymeleaf.mode=HTML5

#前缀

spring.thymeleaf.prefix=classpath:/templates/

#编码

spring.thymeleaf.encoding=UTF-8

#类型

spring.thymeleaf.content-type=text/html

#名称的后缀

spring.thymeleaf.suffix=.html

3、建立文件夹

1)src/main/resources/templates/tl/

2)建立一个 index.html

4、简单测试代码编写和访问

注意：\$表达式只能写在 th 标签内部

快速入门：<https://www.thymeleaf.org/doc/articles/standarddialect5minutes.html>

8、数据库操作之整合 Mybatis 和事务讲解

1、SpringBoot2.x 持久化数据方式介绍

简介：介绍近几年常用的访问数据库的方式和优缺点

1、原始 java 访问数据库

开发流程麻烦

1、注册驱动/加载驱动

Class.forName("com.mysql.jdbc.Driver")

2、建立连接

- ```

 Connection con =
 DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname","root","root");

```
- 3、创建 Statement
  - 4、执行 SQL 语句
  - 5、处理结果集
  - 6、关闭连接，释放资源
- 2、apache dbutils 框架
    - 比上一步简单点
    - 官网:<https://commons.apache.org/proper/commons-dbutils/>
  - 3、jpa 框架
    - spring-data-jpa
    - jpa 在复杂查询的时候性能不是很好
  - 4、Hiberante 解释：ORM：对象关系映射 Object Relational Mapping
    - 企业大都喜欢使用 hibernate
  - 5、Mybatis 框架
    - 互联网行业通常使用 mybatis
    - 不提供对象和关系模型的直接映射,半 ORM

## 2、SpringBoot2.x 整合 Mybatis3.x 注解实战

简介：SpringBoot2.x 整合 Mybatis3.x 注解配置实战

- 1、使用 starter, maven 仓库地址：
  - <http://mvnrepository.com/artifact/org.mybatis.spring.boot/mybatis-spring-boot-starter>
- 2、加入依赖(可以用 <http://start.spring.io/> 下载)
 

```

<!-- 引入 starter -->
<dependency>
 <groupId>org.mybatis.spring.boot</groupId>
 <artifactId>mybatis-spring-boot-starter</artifactId>
 <version>1.3.2</version>
 <scope>runtime</scope>
</dependency>
<!-- MySQL 的 JDBC 驱动包 -->
<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <scope>runtime</scope>
</dependency>
<!-- 引入第三方数据源 -->
<dependency>
 <groupId>com.alibaba</groupId>
 <artifactId>druid</artifactId>
 <version>1.1.6</version>

```

</dependency>

### 3、加入配置文件

```
#mybatis.type-aliases-package=net.xdclass.base_project.domain
#可以自动识别
#spring.datasource.driver-class-name =com.mysql.jdbc.Driver
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/movie?useUnicode=true&characterEncoding=utf-8
spring.datasource.username =root
spring.datasource.password =password
#如果不使用默认的数据源 (com.zaxxer.hikari.HikariDataSource)
spring.datasource.type =com.alibaba.druid.pool.DruidDataSource
```

加载配置，注入到 sqlSessionSessionFactory 等都是 springBoot 帮我们完成

### 4、启动类增加 mapper 扫描

```
@MapperScan("net.xdclass.base_project.mapper")
或者在每一个 mapper 接口上添加@Mapper
技巧：保存对象，获取数据库自增 id
@Options(useGeneratedKeys=true, keyProperty="id", keyColumn="id")
```

### 4、开发 mapper

参考语法 <http://www.mybatis.org/mybatis-3/zh/java-api.html>

### 5、sql 脚本

```
CREATE TABLE `user` (
 `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
 `name` varchar(128) DEFAULT NULL COMMENT '名称',
 `phone` varchar(16) DEFAULT NULL COMMENT '用户手机号',
 `create_time` datetime DEFAULT NULL COMMENT '创建时间',
 `age` int(4) DEFAULT NULL COMMENT '年龄',
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;
```

相关资料：

<http://www.mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/#Configuration>

<https://github.com/mybatis/spring-boot-starter/tree/master/mybatis-spring-boot-samples>

整合问题集合：

<https://my.oschina.net/hxflar1314520/blog/1800035>

<https://blog.csdn.net/tingxuetaige/article/details/80179772>

## 3、SpringBoot 整合 Mybatis 实操和打印 SQL 语句

推荐使用#{ }进行 sql 赋值，不使用\${ }，因为存在 sql 注入的风险

讲解:SpringBoot2.x 整合 Mybatis3.x 增删改查实操，控制台打印 sql 语句

### 1、控制台打印 sql 语句

#增加打印 sql 语句，一般用于本地开发测试

```
mybatis.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl
```

## 2、增加 mapper 代码

```
@Select("SELECT * FROM user")
@Results({
 @Result(column = "create_time",property = "createTime") //javaType =
 java.util.Date.class
})
List<User> getAll();
@Select("SELECT * FROM user WHERE id = #{id}")
@Results({
 @Result(column = "create_time",property = "createTime")
})
User findById(Long id);
@Update("UPDATE user SET name=#{name} WHERE id =#{id}")
void update(User user);
@Delete("DELETE FROM user WHERE id =#{userId}")
void delete(Long userId);
```

## 3、增加 API

```
@GetMapping("find_all")
public Object findAll(){
 return JsonData.buildSuccess(userMapper.getAll());
}
@GetMapping("find_by_id")
public Object findById(long id){
 return JsonData.buildSuccess(userMapper.findById(id));
}
@GetMapping("del_by_id")
public Object delById(long id){
 userMapper.delete(id);
 return JsonData.buildSuccess();
}
@GetMapping("update")
public Object update(String name,int id){
 User user = new User();
 user.setName(name);
 user.setId(id);
 userMapper.update(user);
 return JsonData.buildSuccess();
}
```

# 4、事务介绍和常见的隔离级别，传播行为

简介：讲解什么是数据库事务，常见的隔离级别和传播行为

1、介绍什么是事务，单机事务，分布式事务处理等

## 2、讲解场景的隔离级别

Serializable: 最严格, 串行处理, 消耗资源大

Repeatable Read: 保证了一个事务不会修改已经由另一个事务读取但未提交 (回滚) 的数据

Read Committed: 大多数主流数据库的默认事务等级

Read Uncommitted: 保证了读取过程中不会读取到非法数据。

## 3、讲解常见的传播行为

PROPAGATION\_REQUIRED--支持当前事务, 如果当前没有事务, 就新建一个事务, 最常见的选择。

PROPAGATION\_SUPPORTS--支持当前事务, 如果当前没有事务, 就以非事务方式执行。

PROPAGATION\_MANDATORY--支持当前事务, 如果当前没有事务, 就抛出异常。

PROPAGATION\_REQUIRES\_NEW--新建事务, 如果当前存在事务, 把当前事务挂起, 两个事务之间没有关系, 一个异常, 一个提交, 不会同时回滚

PROPAGATION\_NOT\_SUPPORTED--以非事务方式执行操作, 如果当前存在事务, 就把当前事务挂起。

PROPAGATION\_NEVER--以非事务方式执行, 如果当前存在事务, 则抛出异常

# 5、SpringBoot 整合 mybatis 之事务处理实战

简介: SpringBoot 整合 Mybatis 之事务处理实战

1、service 逻辑引入事务 @Transactional(propagation=Propagation.REQUIRED)

2、service 代码

@Override

@Transactional(propagation=Propagation.REQUIRED)

```
public int addAccount() {
 User user = new User();
 user.setAge(9);
 user.setCreateTime(new Date());
 user.setName("事务测试");
 user.setPhone("000121212");

 userMapper.insert(user);
 int a = 1/0;
 return user.getId();
}
```

# 9、SpringBoot2.x 整合 Redis 实战

## 1、分布式缓存 Redis 介绍

简介:讲解为什么要用缓存和介绍什么是 Redis, 新手练习工具

- 1、redis 官网 <https://redis.io/download>
- 2、新手入门 redis 在线测试工具: <http://try.redis.io/>

## 2、源码编译安装 Redis4.x

简介: 使用源码安装 Redis4.x 和配置外网访问

- 1、快速安装 <https://redis.io/download#installation>

```
wget http://download.redis.io/releases/redis-4.0.9.tar.gz
tar xzf redis-4.0.9.tar.gz
cd redis-4.0.9
make
启动服务端: src/redis-server
启动客户端: src/redis-cli
```
- 2、默认是本地访问的, 需要开放外网访问
  - 1) 打开 redis.conf 文件在 NETWORK 部分修改  
注释掉 bind 127.0.0.1 可以使所有的 ip 访问 redis  
修改 protected-mode, 值改为 no 本地开发, 生产环境为 yes

## 3、SpringBoot2.x 整合 redis 实战讲解

简介: 使用 springboot-starter 整合 reids 实战

- 1、官网:  
<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-redis>  
集群文档:  
<https://docs.spring.io/spring-data/data-redis/docs/current/reference/html/#cluster>
- 2、springboot 整合 redis 相关依赖引入

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```
- 3、相关配置文件配置

```
#####redis 基础配置#####
spring.redis.database=0
spring.redis.host=127.0.0.1
spring.redis.port=6390
连接超时时间 单位 ms (毫秒)
spring.redis.timeout=3000
#####redis 线程池设置#####
连接池中的最大空闲连接, 默认值也是 8。
spring.redis.pool.max-idle=200
#连接池中的最小空闲连接, 默认值也是 0。
```



```
spring.redis.pool.min-idle=200
```

# 如果赋值为-1, 则表示不限制; pool 已经分配了 maxActive 个 jedis 实例, 则此时 pool 的状态为 exhausted(耗尽)。

```
spring.redis.pool.max-active=2000
```

# 等待可用连接的最大时间, 单位毫秒, 默认值为-1, 表示永不超时

```
spring.redis.pool.max-wait=1000
```

#### 4、常见 redistemplate 种类讲解和缓存实操(使用自动注入)

##### 1、注入模板

```
@Autowired
```

```
private StringRedisTemplate strTplRedis
```

##### 2、类型 String, List, Hash, Set, ZSet

对应的方法分别是 opsForValue()、opsForList()、opsForHash()、opsForSet()、opsForZSet()

## 4、Redis 工具类封装讲解和实战

简介: 高效开发方式 Redis 工具类封装讲解和实战

1、常用客户端 <https://redisdesktop.com/download>

2、封装 redis 工具类并操作

## 10、SpringBoot 整合定时任务和异步任务处理

### 1、SpringBoot 定时任务 schedule 讲解

简介: 讲解什么是定时任务和常见定时任务区别

1、常见定时任务 Java 自带的 java.util.Timer 类

timer:配置比较麻烦, 时间延后问题

timertask:不推荐

2、Quartz 框架

配置更简单

xml 或者注解

3、SpringBoot 使用注解方式开启定时任务

1) 启动类里面 @EnableScheduling 开启定时任务, 自动扫描

2) 定时任务业务类 加注解 @Component 被容器扫描

3) 定时执行的方法加上注解 @Scheduled(fixedRate=2000) 定期执行一次

### 2、SpringBoot 常用定时任务配置实战

简介: SpringBoot 常用定时任务表达式配置和在线生成器

1、cron 定时任务表达式 @Scheduled(cron="\*/1 \* \* \* \*") 表示每秒

1) crontab 工具 <https://tool.lu/crontab/>

- 2、fixedRate: 定时多久执行一次（上一次开始执行时间点后 xx 秒再次执行;）
- 3、fixedDelay: 上一次执行结束时间点后 xx 秒再次执行
- 4、fixedDelayString: 字符串形式，可以通过配置文件指定

### 3、SpringBoot2.x 异步任务实战（核心知识）

简介：讲解什么是异步任务，和使用 SpringBoot2.x 开发异步任务实战

- 1、什么是异步任务和使用场景：适用于处理 log、发送邮件、短信……等  
 下单接口->查库存 100  
                   余额校验 150  
                   风控用户 100  
                   ....
- 2、启动类里面使用@EnableAsync 注解开启功能，自动扫描
- 3、定义异步任务类并使用@Component 标记组件被容器扫描,异步方法加上 @Async  
 注意点：
  - 1) 要把异步任务封装到类里面，不能直接写到 Controller
  - 2) 增加 Future<String> 返回结果 AsyncResult<String>("task 执行完成");
  - 3) 如果需要拿到结果 需要判断全部的 task.isDone()
- 4、通过注入方式，注入到 controller 里面，如果测试同步和异步前后区别则改为同步则把 Async 注释掉

## 11、Logback 日志框架介绍和 SpringBoot 整合实战

### 1、新日志框架 LogBack 介绍

简介：日志介绍和新日志框架 Logback 讲解

- 1.常用处理 java 的日志组件 slf4j,log4j,logback,common-logging 等
- 2、logback 介绍：基于 Log4j 基础上大量改良，不能单独使用，推荐配合日志框架 SLF4J 来使用

logback 当前分成三个模块：logback-core,logback-classic 和 logback-access;  
 logback-core 是其它两个模块的基础模块

- 3、Logback 的核心对象：

Logger：日志记录器

Appender：指定日志输出的目的地，目的地可以是控制台，文件

Layout：日志布局 格式化日志信息的输出

- 4、日志级别：DEBUG < INFO < WARN < ERROR

=====log4j 示例=====

### 设置###

log4j.rootLogger = debug,stdout,D,E

### 输出信息到控制台 ###

```

log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target = System.out
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd
HH:mm:ss,SSS} method:%l%n%m%n
输出 DEBUG 级别以上的日志到=D://logs/error.log
log4j.appender.D = org.apache.log4j.DailyRollingFileAppender
log4j.appender.D.File = D://logs/log.log
log4j.appender.D.Append = true
log4j.appender.D.Threshold = DEBUG
log4j.appender.D.layout = org.apache.log4j.PatternLayout
log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [%t:%r]
- [%p] %m%n
输出 ERROR 级别以上的日志到=D://logs/error.log
log4j.appender.E = org.apache.log4j.DailyRollingFileAppender
log4j.appender.E.File = E://logs/error.log
log4j.appender.E.Append = true
log4j.appender.E.Threshold = ERROR
log4j.appender.E.layout = org.apache.log4j.PatternLayout
log4j.appender.E.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [%t:%r]
- [%p] %m%n
=====logback=====

```

4、Log4j 日志转换为 logback 在线工具（支持 log4j.properties 转换为 logback.xml,不支持 log4j.xml 转换为 logback.xml）

<https://logback.qos.ch/translator/>

## 2、SpringBoot2.x 日志讲解和 Logback 配置实战

简介：讲解 SpringBoot2.x 整合 Logback 配置实战

### 1、官网介绍：

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-logging>

各个组件案例：<https://logback.qos.ch/manual/index.html>

### 2、分析 SpringBoot 启动日志

1) 默认情况下，Spring Boot 将日志输出到控制台

### 3、整合 Logback 实战

1) 创建 日志文件 logback-spring.xml，官方推荐 -spring.xml 结尾

默认加载配置顺序 logback-spring.xml, logback-spring.groovy, logback.xml, or logback.groovy

注释：

```

<configuration> 子节点
<appender></appender>
<logger></logger>
<root></root>(要加在最后)

```

## 12 章 搜索框架 Elasticsearch 介绍和整合 SpringBoot

### 1、搜索知识和搜索框架 elasticsearch 介绍

简介：通过京东商城 介绍什么是搜索引擎，和开源搜索框架 Elasticsearch6.x 新特性介绍

前言：介绍 ES 的主要特点和使用场景，新特性讲解

mysql: like 模糊，性能问题，

solr: 针对企业，Lucene

elasticsearch: 针对数据量特别大，PB,TB

纯 java 开发，springboot 使用，5.6 版本

es 升级 4->5 版本，改动大，但是 5 版本后，改动不大

elasticsearch 主要特点

1、特点：全文检索，结构化检索，数据统计、分析，接近实时处理，分布式搜索 (可部署数百台服务器)，处理 PB 级别的数据

搜索纠错，自动完成

2、使用场景：日志搜索，数据聚合，数据监控，报表统计分析

3、国内外使用者：维基百科，Stack Overflow，GitHub

新特性讲解

1、6.2.x 版本基于 Lucene 7.x，更快，性能进一步提升,对应的序列化组件，升级到 Jackson 2.8

mysql: database table record

es : index type (只能存在一个) document

2、推荐使用 5.0 版本推出的 Java REST/HTTP 客户端，依赖少，比 Transport 使用更方便，在基准测试中，性能并不输于 Transport 客户端，

在 5.0 到 6.0 版本中，每次有对应的 API 更新，文档中也说明，推荐使用这种方式进行开发使用,所有可用节点间的负载均衡

在节点故障和特定响应代码的情况下进行故障转移,失败的连接处罚（失败的节点是否重试取决于失败的连续次数;失败的失败次数越多，客户端在再次尝试同一节点之前等待的时间越长）

3、(重要)不再支持一个索引库里面多个 type，6.x 版本已经禁止一个 index 里面多个 type，所以一个 index 索引库只能存在 1 个 type

官方文档：

1、6.0 更新特性

<https://www.elastic.co/guide/en/elasticsearch/reference/6.0/release-notes-6.0.0.html#breaking-java-6.0.0>

2、6.1 更新特性

<https://www.elastic.co/guide/en/elasticsearch/reference/6.1/release-notes-6.1.0.html>

## 2、快熟部署 Elasticsearch5.6.x

简介：讲解为什么不用 ES6.x 版本，及本地快速安装 ElasticSeach 和场景问题处理

配置 JDK1.8

使用 wget 下载 elasticsearch 安装包

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.8.tar.gz

解压

tar -zxvf elasticsearch-5.6.8.tar.gz

官网：<https://www.elastic.co/products/elasticsearch>

外网访问配置：

config 目录下面 elasticsearch.yml

修改为 network.host: 0.0.0.0

配置 es 出现相关问题处理（阿里云、腾讯云，亚马逊云安装问题集合）：

### 1、问题一

```
Java HotSpot(TM) 64-Bit Server VM warning: INFO:
os::commit_memory(0x00000000c5330000, 986513408, 0) failed;
error='Cannot allocate memory' (errno=12)
#
There is insufficient memory for the Java Runtime Environment to continue.
Native memory allocation (mmap) failed to map 986513408 bytes for
committing reserved memory.
An error report file with more information is saved as:
/usr/local/software/temp/elasticsearch-6.2.2/hs_err_pid1912.log
```

解决：内存不够，购买阿里云的机器可以动态增加内存

### 2、问题二

```
[root@iZwz95j86y235aroi85ht0Z bin]# ./elasticsearch
[2018-02-
22T20:14:04,870][WARN][o.e.b.ElasticsearchUncaughtExceptionHandler] []
uncaught exception in thread [main]
org.elasticsearch.bootstrap.StartupException: java.lang.RuntimeException: can
not run elasticsearch as root
at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:125)
~[elasticsearch-6.2.2.jar:6.2.2]
at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:112)
~[elasticsearch-6.2.2.jar:6.2.2]
at
org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAware
Command.java:86) ~[elasticsearch-6.2.2.jar:6.2.2]
at
org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:124)
~[elasticsearch-cli-6.2.2.jar:6.2.2]
```

解决：用非 root 用户

添加用户：useradd -m 用户名 然后设置密码 passwd 用户名

### 3、问题三

./elasticsearch

Exception in thread "main" java.nio.file.AccessDeniedException:  
/usr/local/software/temp/elasticsearch-6.2.2/config/jvm.options

解决: 权限不够 chmod 777 -R 当前 es 目录

常见配置问题资料: <https://www.jianshu.com/p/c5d6ec0f35e0>

## 3、ElasticSearch5.6 测试数据准备

简介: ElasticSearch5.6.x 简单测试

1、步骤 <https://www.elastic.co/guide/en/elasticsearch/reference/5.6/index.html>

2、使用 POSTMAN 工具

基础

查看集群状态: localhost:9200/\_cat/health?v

查看索引列表: localhost:9200/\_cat/indices?v

## 4、SpringBoot2.x 整合 elasticsearch5.6.x

简介: SpringBoot2.x 整合 elasticsearch5.6.8 实战

Spring Data Elasticsearch 文档地址

<https://docs.spring.io/spring->

[data/elasticsearch/docs/3.0.6.RELEASE/reference/html/](https://docs.spring.io/spring-data/elasticsearch/docs/3.0.6.RELEASE/reference/html/)

版本说明: SpringBoot 整合 elasticsearch

<https://github.com/spring-projects/spring-data-elasticsearch/wiki/Spring-Data-Elasticsearch---Spring-Boot---version-matrix>

1、添加 maven 依赖

```
<dependency>
```

```
 <groupId>org.springframework.boot</groupId>
```

```
 <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
```

```
</dependency>
```

2、接口继承 ElasticsearchRepository,里面有很多默认实现

注意点:

索引名称记得小写, 类属性名称也要小写

新建实体对象 article

加上类注解 @Document(indexName = "blog", type = "article")

3、配置文件:

```
ELASTICSEARCH (ElasticsearchProperties)
```

```
spring.data.elasticsearch.cluster-name=elasticsearch # Elasticsearch cluster name.
```

```
spring.data.elasticsearch.cluster-nodes=localhost:9300 # Comma-separated list of cluster node addresses.
```

```
spring.data.elasticsearch.repositories.enabled=true # Whether to enable Elasticsearch repositories.
```

#### 4、QueryBuilder 使用

<https://www.elastic.co/guide/en/elasticsearch/client/java-api/1.3/query-dsl-queries.html>

//单个匹配, 搜索 name 为 jack 的文档

```
QueryBuilder queryBuilder = QueryBuilders.matchQuery("title", "搜");
```

#### 4、查看 es 数据

查看索引信息: [http://localhost:9200/\\_cat/indices?v](http://localhost:9200/_cat/indices?v)

查看某个索引库结构: <http://localhost:9200/blog>

查看某个对象: <http://localhost:9200/blog/article/1>

## 13、SpringBoot2.x 整合 RocketMQ、ActiveMQ

### 1、JMS 介绍和使用场景及基础编程模型

简介: 讲解什么是小写队列, JMS 的基础知识和使用场景

1、什么是 JMS: Java 消息服务 (Java Message Service), Java 平台中关于面向消息中间件的接口

2、JMS 是一种与厂商无关的 API, 用来访问消息收发系统消息, 它类似于 JDBC(Java Database Connectivity)。这里, JDBC 是可以用来访问许多不同关系数据库的 API

3、使用场景:

- 1) 跨平台
- 2) 多语言
- 3) 多项目
- 4) 解耦
- 5) 分布式事务
- 6) 流量控制
- 7) 最终一致性
- 8) RPC 调用

上下游对接, 数据源变动->通知下属

4、概念

JMS 提供者: Apache ActiveMQ、RabbitMQ、Kafka、Notify、MetaQ、RocketMQ

JMS 生产者(Message Producer)

JMS 消费者(Message Consumer)

JMS 消息

JMS 队列

JMS 主题

JMS 消息通常有两种类型: 点对点(Point-to-Point)、发布/订阅(Publish/Subscribe)

5、编程模型

MQ 中需要用到的一些类

ConnectionFactory : 连接工厂, JMS 用它创建连接

Connection : JMS 客户端到 JMS Provider 的连接

Session: 一个发送或接收消息的线程

Destination : 消息的目的地;消息发送给谁.

MessageConsumer / MessageProducer: 消息接收者, 消费者

## 2、ActiveMQ5.x 消息队列基础介绍和安装

简介: 介绍 ActiveMQ5.x 消息队列基础特性和本地快速安装

特点:

- 1) 支持来自 Java, C, C++, C#, Ruby, Perl, Python, PHP 的各种跨语言客户端和协议
- 2) 支持许多高级功能, 如消息组, 虚拟目标, 通配符和复合目标
- 3) 完全支持 JMS 1.1 和 J2EE 1.4, 支持瞬态, 持久, 事务和 XA 消息
- 4) Spring 支持, ActiveMQ 可以轻松嵌入到 Spring 应用程序中, 并使用 Spring 的 XML 配置机制进行配置
- 5) 支持在流行的 J2EE 服务器 (如 TomEE, Geronimo, JBoss, GlassFish 和 WebLogic) 中进行测试
- 6) 使用 JDBC 和高性能日志支持非常快速的持久化

1、下载地址: <http://activemq.apache.org/activemq-5153-release.html>

2、快速开始: <http://activemq.apache.org/getting-started.html>

3、如果我们是 32 位的机器, 就双击 win32 目录下的 activemq.bat,如果是 64 位机器, 则双击 win64 目录下的 activemq.bat

4、bin 目录里面启动 选择对应的系统版本和位数, activeMQ start 启动

5、启动后访问路径 <http://127.0.0.1:8161/>

6、用户名和密码默认都是 admin

7、官方案例集合

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples>

面板:

Name: 队列名称。

Number Of Pending Messages: 等待消费的消息个数。

Number Of Consumers: 当前连接的消费者数目

Messages Enqueued: 进入队列的消息总个数, 包括出队列的和待消费的, 这个数量只增不减。

Messages Dequeued: 已经消费的消息数量。

## 3、SpringBoot2 整合 ActiveMQ 实战之点对点消息

简介: SpringBoot2.x 整合 ActiveMQ 实战之点对点消息

1、官网地址:

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-activemq>

2、加入依赖

```
<!-- 整合消息队列 ActiveMQ -->
```

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
```



```

 <artifactId>spring-boot-starter-activemq</artifactId>
 </dependency>
 <!-- 如果配置线程池则加入 -->
 <dependency>
 <groupId>org.apache.activemq</groupId>
 <artifactId>activemq-pool</artifactId>
 </dependency>

```

### 3、application.properties 配置文件配置

```

#整合 jms 测试，安装在别的机器，防火墙和端口号记得开放
spring.activemq.broker-url=tcp://127.0.0.1:61616
#集群配置
#spring.activemq.broker-url=failover:(tcp://localhost:61616,tcp://localhost:61617)
spring.activemq.user=admin
spring.activemq.password=admin
#下列配置要增加依赖
spring.activemq.pool.enabled=true
spring.activemq.pool.max-connections=100

```

### 4、springboot 启动类 @EnableJms，开启支持 jms

### 5、模拟请求

```
localhost:8080/api/v1/order?msg=12312321321312
```

### 6、消费者：实时监听对应的队列

```
@JmsListener(destination = "order.queue")
```

## 4、SpringBoot2 整合 ActiveMQ 实战之发布订阅模式

简介：SpringBoot 整合 ActiveMQ 实战之发布订阅模式(pub/sub),及同时支持点对点 and 发布订阅模型

### 1、需要加入配置文件，支持发布订阅模型，默认只支持点对点

```

#default point to point
spring.jms.pub-sub-domain=true

```

注意点：

### 1、默认消费者并不会消费订阅发布类型的消息，这是由于 springboot 默认采用的是 p2p 模式进行消息的监听

修改配置：spring.jms.pub-sub-domain=true

### 2、@JmsListener 如果不指定独立的 containerFactory 的话是只能消费 queue 消息

修改订阅者 container：containerFactory="jmsListenerContainerTopic"

//需要给 topic 定义独立的 JmsListenerContainer

@Bean

```

Public JmsListenerContainerFactory<?>
jmsListenerContainerTopic(ConnectionFactory
activeMQConnectionFactory) {
DefaultJmsListenerContainerFactory bean = new
DefaultJmsListenerContainerFactory();
bean.setPubSubDomain(true);

```

```
 bean.setConnectionFactory(activeMQConnectionFactory);
 return bean;
 }
}
```

在配置文件里面，注释掉 #spring.jms.pub-sub-domain=true

## 5、RocketMQ4.x 消息队列介绍

简介：阿里开源消息队列 RocketMQ4.x 介绍和新概念讲解

1、Apache RocketMQ 作为阿里开源的一款高性能、高吞吐量的分布式消息中间件

2、特点

- 1)在高压下 1 毫秒内响应延迟超过 99.6%。
- 2)适合金融类业务，高可用性跟踪和审计功能。
- 3)支持发布订阅模型，和点对点
- 4) 支持拉 pull 和推 push 两种消息模式
- 5)单一队列百万消息
- 6)支持单 master 节点，多 master 节点，多 master 多 slave 节点

...

3、概念

Producer:消息生产者

Producer Group:消息生产者组，发送同类消息的一个消息生产组

Consumer:消费者

Consumer Group:消费同个消息的多个实例

Tag:标签，子主题（二级分类），用于区分同一个主题下的不同业务的消息

Topic:主题

Message: 消息

Broker: MQ 程序，接收生产的消息，提供给消费者消费的程序

Name Server: 给生产和消费者提供路由信息，提供轻量级的服务发现和路由

3、官网地址：<http://rocketmq.apache.org/>

学习资源：

- 1) <http://jm.taobao.org/2017/01/12/rocketmq-quick-start-in-10-minutes/>
- 2) <https://www.jianshu.com/p/453c6e7ff81c>

## 6、RocketMQ4.x 本地快速部署

简介:RocketMQ4.x 本地快速部署

1、安装前提条件(推荐)

64bit OS, Linux/Unix/Mac

64bit JDK 1.8+;

2、快速开始 <http://rocketmq.apache.org/docs/quick-start/>

下载安装包：

<https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.2.0/rocketmq-all-4.2.0-bin-release.zip>

路径：/Users/jack/Desktop/person/springboot/资料/第 13 章/第 5 课/rocketmq-all-

4.2.0-bin-release/bin

### 3、解压压缩包

1) 进入 bin 目录, 启动 namesrv

```
nohup sh mqnamesrv &
```

2) 查看日志 tail -f nohup.out

结尾: The Name Server boot success. serializeType=JSON 表示启动成功

3、启动 broker

```
nohup sh mqbroker -n 127.0.0.1:9876 &
```

4)、关闭 nameserver broker 执行的命令

```
sh mqshutdown namesrv
```

```
sh mqshutdown broker
```

## 7、RoekerMQ4.x 可视化控制台讲解

简介: RoekerMQ4.x 可视化控制台讲解

1、下载 <https://github.com/apache/rocketmq-externals>

2、编译打包 mvn clean package -Dmaven.test.skip=true

3、target 目录 通过 java -jar 的方式运行

4、无法连接获取 broker 信息

1) 修改配置文件,名称路由地址为 namesrvAddr, 例如我本机为

2) src/main/resources/application.properties

```
rocketmq.config.namesrvAddr=192.168.0.101:9876
```

5、默认端口 localhost:8080

6、注意:

在阿里云, 腾讯云或者虚拟机, 记得检查端口号和防火墙是否启动

## 8、Springboot2 整合 RocketMQ4.x 实战上集

简介: Springboot2.x 整合 RocketMQ4.x 实战, 加入相关依赖, 开发生产者代码

启动 nameser 和 broker

1、加入相关依赖

```
<dependency>
```

```
<groupId>org.apache.rocketmq</groupId>
```

```
<artifactId>rocketmq-client</artifactId>
```

```
<version>${rocketmq.version}</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>org.apache.rocketmq</groupId>
```

```
<artifactId>rocketmq-common</artifactId>
```

```
<version>${rocketmq.version}</version>
```

```
</dependency>
```

2、application.properties 加入配置文件

```
消费者的组名
```

```

 apache.rocketmq.consumer.PushConsumer=orderConsumer
 # 生产者的组名
 apache.rocketmq.producer.producerGroup=Producer
 # NameServer 地址
 apache.rocketmq.namesrvAddr=127.0.0.1:9876

```

### 3、开发 MsgProducer

```

/**
 * 生产者的组名
 */
@Value("${apache.rocketmq.producer.producerGroup}")
private String producerGroup;

/**
 * NameServer 地址
 */
@Value("${apache.rocketmq.namesrvAddr}")
private String namesrvAddr;
private DefaultMQProducer producer ;

public DefaultMQProducer getProducer(){
 return this.producer;
}

@PostConstruct
public void defaultMQProducer() {
 //生产者的组名
 producer = new DefaultMQProducer(producerGroup);
 //指定 NameServer 地址， 多个地址以 ; 隔开
 //
 producer.setNamesrvAddr("192.168.100.141:9876;192.168.100.142:9876;192.168.100.149:9876");
 producer.setNamesrvAddr(namesrvAddr);
 producer.setVipChannelEnabled(false);
 try {
 /**
 * Producer 对象在使用之前必须要调用 start 初始化， 只能初始化一次
 */
 producer.start();
 } catch (Exception e) {
 e.printStackTrace();
 }
 // producer.shutdown(); 一般在应用上下文， 关闭的时候进行关闭， 用上下文
 监听器
}

```

## 9、Springboot2 整合 RocketMQ4.x 实战下集

简介：Springboot2.x 整合 RocketMQ4.x 实战，开发消费者代码，常见问题处理

### 1、创建消费者

问题：

1 Caused by: org.apache.rocketmq.remoting.exception.RemotingConnectException: connect to <172.17.42.1:10911> failed

2、com.alibaba.rocketmq.client.exception.MQClientException: Send [1] times, still failed, cost [1647]ms, Topic: TopicTest1, BrokersSent: [broker-a, null, null]

3、org.apache.rocketmq.client.exception.MQClientException: Send [3] times, still failed, cost [497]ms, Topic: TopicTest, BrokersSent: [chenyaowudeMacBook-Air.local, chenaowudeMacBook-Air.local, chenaowudeMacBook-Air.local]

解决：多网卡问题处理

1、设置 producer: producer.setVipChannelEnabled(false);

2、编辑 ROCKETMQ 配置文件：broker.conf（下列 ip 为自己的 ip）

namesrvAddr = 192.168.0.101:9876

brokerIP1 = 192.168.0.101

4、DESC: service not available now, maybe disk full, CL:

解决：修改启动脚本 runbroker.sh，在里面增加一句话即可：

JAVA\_OPT="{JAVA\_OPT} -Drocketmq.broker.diskSpaceWarningLevelRatio=0.98"

（磁盘保护的百分比设置成 98%，只有磁盘空间使用率达到 98%时才拒绝接收 producer 消息）

常见问题处理：

<https://blog.csdn.net/sqzhao/article/details/54834761>

<https://blog.csdn.net/mayifan0/article/details/67633729>

<https://blog.csdn.net/a906423355/article/details/78192828>

## 14 高级篇幅之 SpringBoot 多环境配置

### 1、SpringBoot 多环境配置介绍和项目实战（核心知识）

简介：SpringBoot 介绍多环境配置和使用场景

#### 1、不同环境使用不同配置

例如数据库配置，在开发的时候，我们一般用开发数据库，而在生产环境的时候，我们是用正式的数据

#### 2、配置文件存放路径

classpath 根目录的"/config"包下

classpath 的根目录下

3、spring boot 允许通过命名约定按照一定的格式(application-{profile}.properties)来定义多个配置文件

## 15、高级篇幅之 SpringBoot2.0 响应式编

### 1、SpringBoot2.x 响应式编程简介

简介:讲解什么是 reactive 响应式编程和使用的好处

#### 1、基础理解:

依赖于事件, 事件驱动(Event-driven)

一系列事件称为“流”

异步

非阻塞

观察者模式

网上的一个例子:

```
int b= 2;
```

```
int c=3
```

```
int a = b+c //命令式编程后续 b 和 c 变化, 都不影响 a
```

```
b=5;
```

```
int b= 2;
```

```
int c= 3
```

```
int a = b+c //响应式编程中, a 的变化, 会和 b、c 的变化而变化 (事件驱动)
```

```
b=5;
```

#### 2、官网:

[https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-webflux)

[SNAPSHOT/reference/htmlsingle/#boot-features-webflux](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-webflux)

SpringBoot2 底层是用 spring5,开始支持响应式编程, Spring 又是基于 Reactor 试下响应式。

学习资料

1、reactive-streams 学习资料: <http://www.reactive-streams.org/>

2、web-flux 相关资料: <https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#spring-webflux>

### 2、SpringBoot2.x 响应式编程 webflux 介绍

简介: 讲解 SpringBoot2.x 响应式编程介绍 Mono、Flux 对象和优缺点

1、Spring WebFlux 是 Spring Framework 5.0 中引入的新的反应式 Web 框架

与 Spring MVC 不同, 它不需要 Servlet API, 完全异步和非阻塞, 并通过 Reactor 项目实现 Reactive Streams 规范。

RxJava

2、Flux 和 Mono User List<User>

1) 简单业务而言: 和其他普通对象差别不大, 复杂请求业务, 就可以提升性能

2) 通俗理解:

Mono 表示的是包含 0 或者 1 个元素的异步序列

mono->单一对象 User      redis->用户 ID-》唯一的用户 Mono<User>

Flux 表示的是包含 0 到 N 个元素的异步序列

flux->数组列表对象 List<User>    redis->男性用户->Flux<User>

Flux 和 Mono 之间可以进行转换

3、Spring WebFlux 有两种风格：基于功能和基于注解的。基于注解非常接近 Spring MVC 模型，如以下示例所示：

第一种：

```
@RestController
@RequestMapping ("/ users")
public class MyRestController {
 @GetMapping ("/ {user}")
 public Mono <User> getUser (@PathVariable Long user) {
 // ...
 }
 @GetMapping ("/ {user} / customers")
 public Flux <Customer> getUserCustomers (@PathVariable Long user)
{
 // ...
}
 @DeleteMapping ("/ {user}")
 public Mono <User> deleteUser (@PathVariable Long user) {
 // ...
 }
}
```

第二种： 路由配置与请求的实际处理分开

```
@Configuration
public class RoutingConfiguration {
 @Bean
 public RouterFunction <ServerResponse> monoRouterFunction
(UserHandler userHandler) {
 return route(GET ("/ {user}") .and (accept (APPLICATION_JSON)),
userHandler :: getUser)
 .andRoute (GET ("/ {user} / customers") .and (accept
(APPLICATION_JSON)), userHandler :: getUserCustomers)
 .andRoute (DELETE ("/ {user}") .and (accept
(APPLICATION_JSON)), userHandler :: deleteUser) ;
 }
}

@Component
public class UserHandler {
 公共 Mono <ServerResponse> getUser (ServerRequest 请求) {
 // ...
 }
 public Mono <ServerResponse> getUserCustomers (ServerRequest
request) {
```

```

 // ...
 }
 公共 Mono <ServerResponse> deleteUser (ServerRequest 请求) {
 // ...
 }
}

```

4、Spring WebFlux 应用程序不严格依赖于 Servlet API，因此它们不能作为 war 文件部署，也不能使用 src/main/webapp 目录

5、可以整合多个模板引擎

除了 REST Web 服务外，您还可以使用 Spring WebFlux 提供动态 HTML 内容。

Spring WebFlux 支持各种模板技术，包括 Thymeleaf，FreeMarker

### 3、SpringBoot2.x webflux 实战

简介:webflux 响应式编程实战

1、WebFlux 中，请求和响应不再是 WebMVC 中的 ServletRequest 和 ServletResponse，而是 ServerRequest 和 ServerResponse

2、加入依赖，如果同时存在 spring-boot-starter-web，则会优先用 spring-boot-starter-web

```

<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>

```

测试

localhost:8080/api/v1/user/test

3、启动方式默认是 Netty,8080 端口

4、参考：<https://spring.io/blog/2016/04/19/understanding-reactive-types>

### 4、WebFlux 客户端 WebClient 讲解

简介：讲解 SpringBoot2.x WebFlux 客户端 WebClient 的介绍和使用

1、反应式客户端

官网地址：

[https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-](https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-webclient)

SNAPSHOT/reference/htmlsingle/#boot-features-webclient

## 16、高级篇幅之 SpringBoot2.0 服务器端主动推送 SSE

简介：服务端常用推送技术介绍，如 websocket，sse 轮询等

1、客户端轮询:ajax 定时拉取

2、服务端主动推送:WebSocket

全双工的，本质上是一个额外的 tcp 连接，建立和关闭时握手使用 http 协议，



其他数据传输不使用 http 协议

更加复杂一些，适用于需要进行复杂双向数据通讯的场景

### 3、服务端主动推送:SSE (Server Send Event)

html5 新标准，用来从服务端实时推送数据到浏览器端，

直接建立在当前 http 连接上，本质上是保持一个 http 长连接，轻量协议

简单的服务器数据推送的场景，使用服务器推送事件

学习资料：[http://www.w3school.com.cn/html5/html\\_5\\_serversentevents.asp](http://www.w3school.com.cn/html5/html_5_serversentevents.asp)

## 2、SpringBoot2.x 服务端主动推送 SSE（长连接）

简介：讲解 SpringBoot2.x 服务端主动推送 Sever-Send-Events

1、localhost:8080/index.html

2、需要把 response 的类型 改为 text/event-stream，才是 sse 的类型

这个还是由客户端不断发起请求，只是基于一个连接（类似于轮询，但是区别于轮询），没有重复的三次握手操作

例子：

服务端

```
package net.xdclass.base_project.controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/sse")
public class SSEController {
 @RequestMapping(value = "/get_data", produces = "text/event-stream;charset=UTF-8")
 public String push() {
 try {
 Thread.sleep(1000);
 //第三方数据源调用
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 return "data:xdclass 行情" + Math.random() + "\n\n";
 }
}
```

客户端：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript">
//需要判断浏览器支不支持，可以去 w3c 进行查看
var source = new EventSource('sse/get_data');
```

```

source.onmessage = function (event) {
 console.info(event.data);
 document.getElementById('result').innerText = event.data
};
</script>
</head>
<body>
模拟股票行情
 <div>xdclass test</div>
 <div id="result"></div>
</body>
</html>

```

## 17、高级篇幅之云服务器介绍和部署生产环境实战

### 1、阿里云服务器介绍和使用讲解

简介：阿里云服务器介绍和使用讲解

### 2、阿里云 Linux 服务器部署 JDK8 实战

简介：在阿里云服务器上安装 JDK8 和配置环境变量

Linux 下使用 wget 下载 jdk8:

进到目录 /usr/local/software

配置环境变量

vim /etc/profile

加入

export JAVA\_HOME=/usr/local/software/jdk8

export PATH=\$PATH:\$JAVA\_HOME/bin

export CLASSPATH=.:\$JAVA\_HOME/lib/dt.jar:\$JAVA\_HOME/lib/tools.jar

export JAVA\_HOME PATH CLASSPATH

使用 source /etc/profile 让配置立刻生效

### 3、阿里云服务器 SpringBoot2.x 生产环境部署实战

简介：讲解 SpringBoot 生产环境部署和常见注意事项

#### 1、去除相关生产环境没用的 jar

比如热部署 dev-tool

#### 2、本地 maven 打包成 jar 包

mvn clean package -Dmaven.test.skip=true 跳过测试

#### 3、服务器安装 jdk, 上传 Jar 包

上传工具:

windows:

winscp

securtyCRT

mac:

filezilla

ssh root@120.79.160.143

访问路径 <http://120.79.160.143:8080/api/v1/user/find>

java -jar xxxx.jar

守护进程、系统服务、shell 脚本

打包指定配置文件

- 1、使用 maven 的 profiles
- 2、使用 springboot 的 profile=active

访问不了

- 1、阿里云防火墙是否开启，可以选择关闭，关闭是不安全的，可以选择开放端口
- 2、阿里云的安全访问组，开启对应的端口，如果应用是以 80 端口启动，则默认可以访问

4、成熟的互联网公司应该有的架构

本地提交生产代码->gitlab 仓库->Jenkins 自动化构建->运维或者开发人员发布

## 4、SpringBoot2.x 监控 Actuator 实战上集

简介：讲解 SpringBoot 使用 actuator 监控配置和使用

可用性：100%，99.9%

1、介绍什么是 actuator

官方介绍：

Spring Boot 包含许多附加功能，可帮助您在将应用程序投入生产时监视和管理应用程序。 可以选择使用 HTTP 端点或 JMX 来管理和监控您的应用程序，自动应用于审计，健康和指标收集；

一句话：springboot 提供用于监控和管理生产环境的模块

官方文档：

<https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#production-ready>

2、加入依赖

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

3、加入上述依赖后，访问几个 url

```
/actuator/health
/actuator/info
/actuator
```

## 5、SpringBoot2 监控 Actuator 下集及生产环境建议

简介：SpringBoot2.x 监控 Actuator 实战下集及生产环境建议，SpringBoot 新旧版本区别

注意点：网上的资料大多数没有讲到访问的前缀

端点基础路径由 / 调整到 /actuator

如：/info 调整为/actuator/info  
/actuator/xxx

### 1、只能访问几个 url

#### 1) 需要在配置文件中加入下列配置

`management.endpoints.web.exposure.include=*`

#### 2) 官网说明:

[https://docs.spring.io/spring-boot/docs/current-](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-security-actuator)

[SNAPSHOT/reference/htmlsingle/#boot-features-security-actuator](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-security-actuator)

原因:

出于安全考虑, 除/ health 和/ info 之外的所有执行器默认都是禁用的。

`management.endpoints.web.exposure.include` 属性可用于启用执行器

### 2、建议

在设置 `management.endpoints.web.exposure.include` 之前, 请确保暴露的执行器不包含敏感信息和/

或通过将其放置在防火墙进行控制, 不对外进行使用

禁用的端点将从应用程序上下文中完全删除。如果您只想更改端点所暴露的技术, 请改用 `include` 和 `exclude` 属性。

例子:

开启全部: `management.endpoints.web.exposure.include=*`

开启某个: `management.endpoints.web.exposure.include=metrics`

关闭某个: `management.endpoints.web.exposure.exclude=metrics`

或者用 `springadmin` 进行管理

相关资料: <https://www.cnblogs.com/ityouknow/p/8440455.html>

或者用自己编写脚本监控

CPU、内存、磁盘、nginx 的 http 响应状态码 200,404,5xx

### 3、介绍常用的几个

/health 查看应用健康指标

/actuator/metrics 查看应用基本指标列表

/actuator/metrics/{name} 通过上述列表, 查看具体 查看具体指标

/actuator/env 显示来自 Spring 的 `ConfigurableEnvironment` 的属性