# SpringBoot进阶实践

## 第1节 工具的安装和使用

### 1.1开发工具

Spring Tool Suite：<https://spring.io/tools/sts/all>

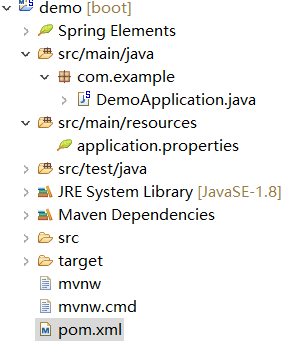
Eclipse：<http://www.eclipse.org/downloads/packages/all>

### 1.2插件

（1）maven 学习Spring Boot必备之一，setting.xml

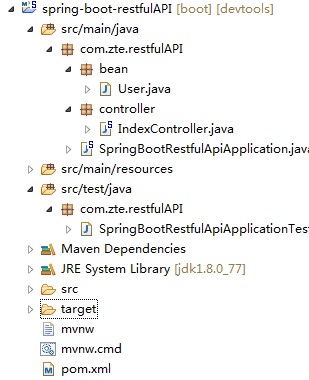
（2） Spring Tool Suite 利用它快速实现Spring Boot的项目搭建

### 1.3快速构建Spring Boot项目

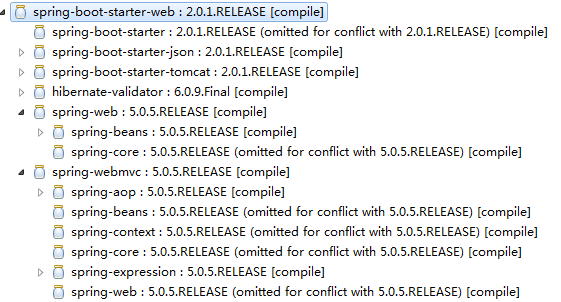


## RESTfull API简单项目的快速搭建

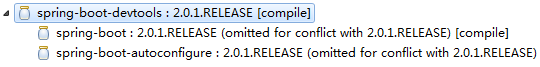
### 2.1新建SpringBoot项目



### 2.2spring-boot-starter-web的引入，其依赖包的学习



### 2.3spring-boot-devtools的引入，其依赖包的学习



### 2.4源码

1）SpringBootRestfulApiApplication

|  |
| --- |
| @SpringBootApplication  **public** **class** SpringBootRestfulApiApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootRestfulApiApplication.**class**, args);  }  } |

2）IndexController

|  |
| --- |
| @RestController  @RequestMapping(value="/index")  **public** **class** IndexController {  @RequestMapping  **public** String index(){  **return** "Hello World";  }    @RequestMapping(value="/get")  **public** Map<String, String> get(@RequestParam String name){  Map<String, String> map = **new** HashMap<>();  map.put("name", name);  map.put("value", "Hello World");  **return** map;  }    @RequestMapping(value = "find/{id}/{name}")  **public** User find(@PathVariable **int** id, @PathVariable String name){  User user = **new** User();  user.setId(id);  user.setName(name);  user.setDate(**new** Date());  **return** user;  }  } |

3）User

|  |
| --- |
| **public** **class** User {  **private** **int** id;  **private** String name;  **private** Date date;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Date getDate() {  **return** date;  }  **public** **void** setDate(Date date) {  **this**.date = date;  }      } |

### 2.5运行项目

直接运行main方法或者使用maven命令: spring-boot:run

测试： <http://localhost:8080/index>

带参数：<http://localhost:8080/index/get?name=Jack>

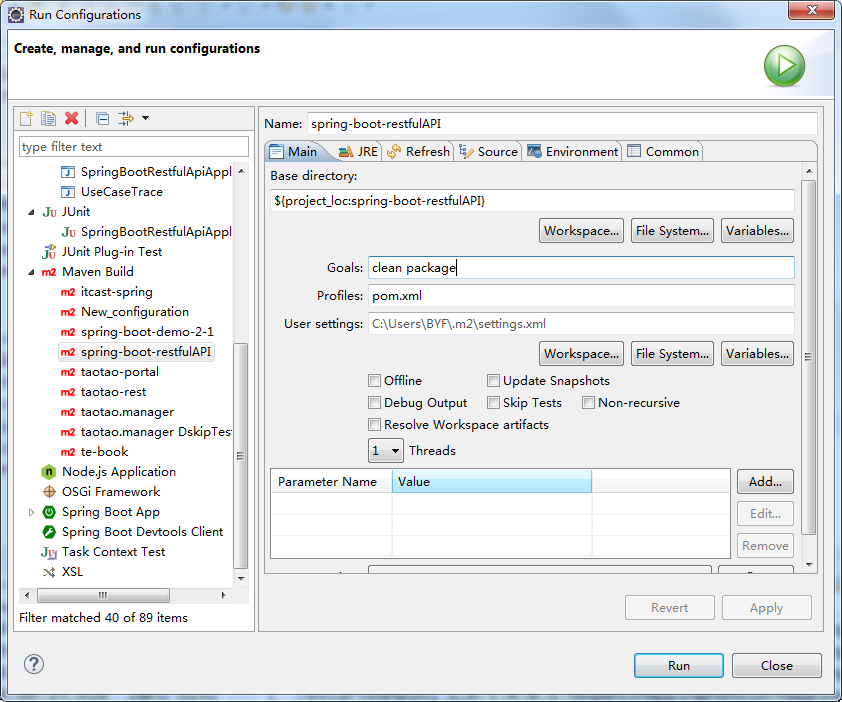
带参数有中文：<http://localhost:8080/index/get?name=杰克>

url测试：<http://localhost:8080/index/find/1/Jack>

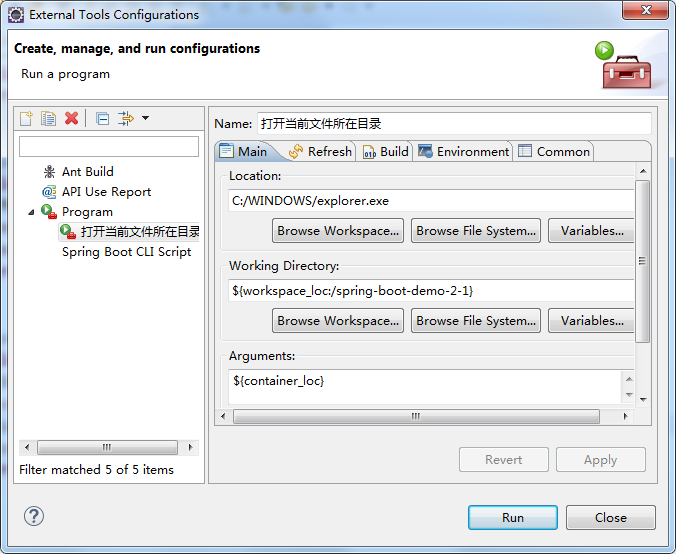
url测试：[http://localhost:8080/index/find/1/杰克](http://localhost:8080/index/get/1/杰克)

### 2.6打包

命令: clean package



快速打开Eclipse工程目录工具



在工程的target目录下找到打包好的jar，在当前目录打开命令窗口，可以使用命令运行java -jar restfulAPI-0.0.1-SNAPSHOT.jar。

## 配置文件详解：Properties和YAML

### 3.1配置文件的生效顺序，会对值进行覆盖

1. @TestPropertySource 注解

2. 命令行参数

3. Java系统属性（System.getProperties()）

4. 操作系统环境变量

5. 只有在random.\*里包含的属性会产生一个RandomValuePropertySource

6. 在打包的jar外的应用程序配置文件（application.properties，包含YAML和profile变量）

7. 在打包的jar内的应用程序配置文件（application.properties，包含YAML和profile变量）

8. 在@Configuration类上的@PropertySource注解

9. 默认属性（使用SpringApplication.setDefaultProperties指定）

### 3.2配置随机值

1）application.properties

|  |
| --- |
| #32位随机数  zte.secret=${random.value}  zte.number=${random.int}  zte.bignumber=${random.long}  #占位符  zte.name=www.zte.com  zte.desc=${zte.name} is a new domain name  #端口配置：  server.port=8090  #时间格式化  spring.jackson.date-format=yyyy-MM-dd HH:mm:ss  #时区设置  spring.jackson.time-zone=Asia/Chongqing |

#32为随机数（这里中文可能是ISO-8859-1编码，需要修改UTF-8）

zte.secret=${random.value}

zte.number=${random.int}

zte.bignumber=${random.long}

zte.number.less.than.ten=${random.int(10)}

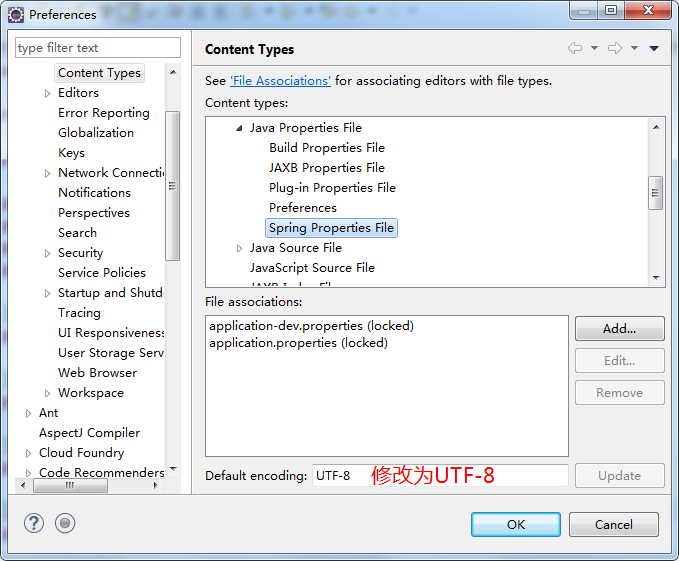
zte.number.in.range=${random.int[1024,65536]}

读取使用注解：@Value(value = "${zte.secret}")

注：出现黄点提示，是要提示配置元数据，可以不配置

设置application.properties文件的默认编码格式

注意：配置文件中打注释时，中文字符显示为\编码，是因为properties文件用的默认ISO-8859-1，需要改为中文编码或者UTF-8。



### 3.3属性占位符

当application.properties里的值被使用时，它们会被存在的Environment过滤，所以你能够引用先前定义的值（比如，系统属性）。

zte.name=www.zte.com

zte.desc=${zte.name} is a domain name

### 3.4Application属性文件，按优先级排序，位置高的将覆盖位置低的

1. 当前目录下的一个/config子目录

2. 当前目录

3. 一个classpath下的/config包

4. classpath根路径（root）

这个列表是按优先级排序的（列表中位置高的将覆盖位置低的）

### 3.5配置应用端口和其他配置的介绍

#端口配置：

server.port=8090

#时间格式化

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss

#时区设置

spring.jackson.time-zone=Asia/Chongqing

### 3.6使用YAML代替Properties

1）application.yaml

|  |
| --- |
| #32位随机数  zte:  secret : ${random.value}  number : ${random.int}  bignumber : ${random.long}  #占位符  name : www.zte.com  desc : ${zte.name} is a domain name  #端口配置：  server:  port : 8090  #时间格式化  spring:  jackson:  date-format : yyyy-MM-dd HH:mm:ss  time-zone : Asia/Chongqing |

注意写法：冒号后要加个空格

## 配置文件-多环境配置

### 4.1多环境配置的好处：

1.不同环境配置可以配置不同的参数

2.便于部署，提高效率，减少出错

### 4.2Properties多环境配置

1. 配置激活选项

spring.profiles.active=dev

2.添加其他配置文件



### 4.3YAML多环境配置

1.配置激活选项

spring:

profiles:

active: dev

2.在配置文件添加三个英文状态下的短横线即可区分

---

spring:

profiles: dev

3.只需要一个配置文件application.yaml

|  |
| --- |
| #32位随机数  zte:  secret : ${random.value}  number : ${random.int}  bignumber : ${random.long}  name : www.zte.com  desc : ${zte.name} is a domain name  spring:  profiles:  active : test    server:  port : 8080  ---  spring:  profiles: test  server:  port : 8081    ---  spring:  profiles: dev  server:  port : 8082    ---  spring:  profiles: pro  server:  port : 8083 |

### 4.4两种配置方式的比较

1. Properties配置多环境，需要添加多个配置文件，YAML只需要一个配件文件

2.书写格式的差异，yaml相对比较简洁，优雅

3. YAML的缺点：不能通过@PropertySource注解加载。如果需要使用@PropertySource注解的方式加载值，那就要使用properties文件。

### 4.5命令行运行不同环境变量方式

java -jar restfulAPI-0.0.1-SNAPSHOT.jar --spring.profiles.active=dev

运行到dev的8082端口。

## 日志配置logback和log4j2

**支持日志框架：**Java Util Logging, Log4J2 and Logback，默认是使用logback

**配置方式：**默认配置文件配置和引用外部配置文件配置

### 5.1默认配置文件配置(不建议使用：不够灵活，对log4j2等不够友好)

# 日志文件名，比如：zte.log，或者是 /var/log/zte.log

logging.file=zte.log

# 日志级别配置，比如： logging.level.org.springframework=DEBUG

logging.level.\*=info

logging.level.org.springframework=DEBUG

### 5.2logback引用外部配置文件

**logback配置方式：**

spring boot默认会加载classpath:logback-spring.xml或者classpath:logback-spring.groovy

**使用自定义配置文件，配置方式为：**

logging.config=classpath:logback-zte.xml

注意：不要使用logback这个来命名，否则spring boot将不能完全实例化

1）使用基于spring boot的配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <configuration>  <include resource=*"org/springframework/boot/logging/logback/base.xml"*/>  <logger name=*"org.springframework.web"* level=*"DEBUG"*/>  </configuration> |

1. 自定义配置logback-zte.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <configuration>  <!-- 文件输出格式 -->  <property name=*"PATTERN"* value=*"%-12(%d{yyyy-MM-dd HH:mm:ss.SSS}) |-%-5level [%thread] %c [%L] -| %msg%n"* />  <!-- test文件路径 -->  <property name=*"TEST\_FILE\_PATH"* value=*"c:/opt/zte/logs"* />  <!-- pro文件路径 -->  <property name=*"PRO\_FILE\_PATH"* value=*"d:/opt/zte/logs"* />  <!-- 开发环境 -->  <springProfile name=*"dev"*>  <appender name=*"CONSOLE"* class=*"ch.qos.logback.core.ConsoleAppender"*>  <encoder>  <pattern>${PATTERN}</pattern>  </encoder>  </appender>    <logger name=*"com.zte.restfulAPI"* level=*"debug"*/>  <root level=*"info"*>  <appender-ref ref=*"CONSOLE"* />  </root>  </springProfile>  <!-- 测试环境 -->  <springProfile name=*"test"*>  <!-- 每天产生一个文件 -->  <appender name=*"TEST-FILE"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>  <!-- 文件路径 -->  <file>${TEST\_FILE\_PATH}</file>  <rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>  <!-- 文件名称 -->  <fileNamePattern>${TEST\_FILE\_PATH}/info.%d{yyyy-MM-dd}.log</fileNamePattern>  <!-- 文件最大保存历史数量 -->  <MaxHistory>100</MaxHistory>  </rollingPolicy>    <layout class=*"ch.qos.logback.classic.PatternLayout"*>  <pattern>${PATTERN}</pattern>  </layout>  </appender>    <root level=*"info"*>  <appender-ref ref=*"TEST-FILE"* />  </root>  </springProfile>  <!-- 生产环境 -->  <springProfile name=*"pro"*>  <appender name=*"PROD\_FILE"* class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>  <file>${PRO\_FILE\_PATH}</file>  <rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>  <fileNamePattern>${PRO\_FILE\_PATH}/warn.%d{yyyy-MM-dd}.log</fileNamePattern>  <MaxHistory>100</MaxHistory>  </rollingPolicy>  <layout class=*"ch.qos.logback.classic.PatternLayout"*>  <pattern>${PATTERN}</pattern>  </layout>  </appender>    <root level=*"warn"*>  <appender-ref ref=*"PROD\_FILE"* />  </root>  </springProfile>  </configuration> |

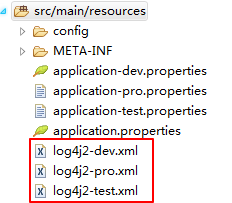
### 5.3 log4j配置

1）去除logback的依赖包，添加log4j2的依赖包

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-logging</artifactId>  </exclusion>  </exclusions>  </dependency>  <!-- 使用log4j2 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-log4j2</artifactId>  </dependency> |

2）在classpath添加log4j2.xml或者log4j2-spring.xml（spring boot 默认加载）

3）自定义配置文件



### 5.4比较

性能比较：Log4J2 和 Logback 都优于 log4j（不推荐使用）

配置方式：Logback最简洁，spring boot默认，推荐使用

## 第6节web应用开发-模板引擎

### 6.1spring boot的web应用开发

spring boot的web应用开发，是基于spring mvc

### 6.2自动配置属性

Spring boot 在spring默认基础上，自动配置添加了以下特性：

1.包含了ContentNegotiatingViewResolver和BeanNameViewResolver beans。

2.对静态资源的支持，包括对WebJars的支持。

3.自动注册Converter，GenericConverter，Formatter beans。

4.对HttpMessageConverters的支持。

5.自动注册MessageCodeResolver。

6.对静态index.html的支持。

7.对自定义Favicon的支持。

8.主动使用ConfigurableWebBindingInitializer bean

### 6.3模板引擎的选择

FreeMarker

Thymeleaf

Velocity (1.4版本之后弃用，Spring Framework 4.3版本之后弃用)

Groovy

Mustache

注：jsp应该尽量避免使用，原因如下：

1.jsp只能打包为：war格式，不支持jar格式，只能在标准的容器里面跑（tomcat，jetty都可以）

2.内嵌的Jetty目前不支持JSPs

3.Undertow不支持jsps

4.jsp自定义错误页面不能覆盖spring boot 默认的错误页面

### 6.4 FreeMarker Demo

1）引入FreeMarker 依赖

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

2）WebController

|  |
| --- |
| @Controller  @RequestMapping(value="/web")  **public** **class** WebController {  **private** Logger logger = LoggerFactory.*getLogger*(WebController.**class**);    @RequestMapping("index")  **public** String index(ModelMap map){  map.put("title", "freemarker test!");  logger.info("freemarker test!");  **return** "index";  }  } |

1. index.ftl，默认的模板文件位置：

/spring-boot-freemaker/src/main/resources/templates/index.ftl

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <title>Spring Boot Demo - FreeMarker</title>    </head>  <body>  首页  <h1 id="title">${title}</h1>  </body>  </html> |

### 6.5 Thymeleaf Demo

1）引入FreeMarker 依赖

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

2）WebController

|  |
| --- |
| @Controller  @RequestMapping(value="/web")  **public** **class** WebController {  **private** Logger logger = LoggerFactory.*getLogger*(WebController.**class**);    @RequestMapping("index")  **public** String index(ModelMap map){  map.put("title", "thymeleaf test!");  logger.info("thymeleaf test!");  **return** "index";  }  } |

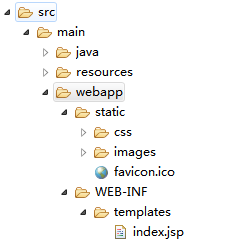
3）index.html，默认的模板文件位置：

/spring-boot-thymeleaf/src/main/resources/templates/index.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang=*"en"*>  <title>Spring Boot Demo - FreeMarker</title>    <link href=*"/css/index.css"* rel=*"stylesheet"* />    </head>  <body>  <center>  <img src=*"/images/logo.png"* />  <h1 id=*"title"* th:text=*"${title}"*></h1>  </center>    <script type=*"text/javascript"* src=*"/webjars/jquery/2.1.4/jquery.min.js"*></script>    <script>  $(**function**(){  $('#title').click(**function**(){  alert('点击了');  });  })  </script>  </body>  </html> |

### 6.6 Jsp Demp

注意：Jsp新建工程时，将项目类型改为war工程，在webapp目录下新建WEB-INF并在该目录下添加/templates/index.jsp，以及对应的静态资源文件。



1）引入FreeMarker 依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <!-- 指定maven依赖的生命周期，provided表明该包只在编译和测试的时候用 -->  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

2）添加配置参数

spring.mvc.view.prefix: /WEB-INF/templates/

spring.mvc.view.suffix: .jsp

3）WebController

|  |
| --- |
| @Controller  @RequestMapping(value="/web")  **public** **class** WebController {  **private** Logger logger = LoggerFactory.*getLogger*(WebController.**class**);    @RequestMapping("index")  **public** String index(ModelMap map){  map.put("title", "Jsp test!");  logger.info("Jsp test!这是一个Jsp模板");  **return** "index";  }  } |

1. index.jsp，默认的模板文件位置：

/spring-boot-jsp/src/main/webapp/WEB-INF/templates/index.jsp

|  |
| --- |
| <%@ taglib prefix=*"spring"* uri=*"http://www.springframework.org/tags"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>  <!DOCTYPE html>  <html>  <head lang=*"en"*>  <title>Spring Boot Demo - FreeMarker</title>    <link href=*"/static/css/index.css"* rel=*"stylesheet"* />    </head>  <body>  <img src=*"/static/images/logo.png"* alt=*"logo"*/>  <h1 id=*"title"*>${title}</h1>    <c:url value=*"http://www.jstl.com"* var=*"url"*/>  <spring:url value=*"http://www.spring.com"* htmlEscape=*"true"* var=*"springUrl"* />    Spring URL: ${springUrl}  <br>  JSTL URL: ${url}    <!-- <script type="text/javascript" src="/static/webjars/jquery/2.1.4/jquery.min.js"></script>  <script>  $(function(){  $('#title').click(function(){  alert('点击了');  });  })  </script> -->  </body>  </html> |

### 6.7错误处理

1）错误的处理

方法一：Spring Boot 将所有的错误默认映射到/error， 实现ErrorController

|  |
| --- |
| @Controller  @RequestMapping(value="/web")  **public** **class** WebController {  **private** Logger logger = LoggerFactory.*getLogger*(WebController.**class**);    @RequestMapping("index")  **public** String index(ModelMap map){  map.put("title", "freemarker test!");  logger.info("freemarker test!");  **return** "index";  }  } |

在templates下建error目录，新建error.ftl

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <title>Spring Boot Demo - FreeMarker</title>  <link href="/css/index.css" rel="stylesheet" />  </head>  <body>  <h1>系统错误，请联系管理员</h1>  </body>  </html> |

方法二：添加自定义的错误页面

2.1html静态页面：在resources/public/error/ 下定义

如添加404页面： resources/public/error/404.html页面，中文注意页面编码

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang=*"en"*>  <title>Spring Boot Demo - FreeMarker</title>  <link href=*"/css/index.css"* rel=*"stylesheet"* />  </head>  <body>  <h1>404-找不到页面</h1>  </body>  </html> |

2.2 模板引擎页面：在templates/error/下定义

如添加5xx页面： templates/error/5xx.ftl

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <title>Spring Boot Demo - FreeMarker</title>  </head>  <body>  <h1>5xx-系统错误</h1>  <h1>${exception}</h1>  </body>  </html> |

注意：templates/error/ 这个的优先级比较 resources/public/error/高

方法三：使用注解@ControllerAdvice

|  |
| --- |
| @ControllerAdvice  **public** **class** BizException {    **private** **static** Logger *logger* = LoggerFactory.*getLogger*(BizException.**class**);  /\*\*  \* 统一异常处理  \*  \* **@param** exception  \* exception  \* **@return**  \*/  @ExceptionHandler({ RuntimeException.**class** })  @ResponseStatus(HttpStatus.***OK***)  **public** ModelAndView processException(RuntimeException exception) {  *logger*.info("自定义异常处理-RuntimeException");  ModelAndView m = **new** ModelAndView();  m.addObject("zteException", exception.getMessage());  m.setViewName("error/500");  **return** m;  }  /\*\*  \* 统一异常处理  \*  \* **@param** exception  \* exception  \* **@return**  \*/  @ExceptionHandler({ Exception.**class** })  @ResponseStatus(HttpStatus.***OK***)  **public** ModelAndView processException(Exception exception) {  *logger*.info("自定义异常处理-Exception");  ModelAndView m = **new** ModelAndView();  m.addObject("zteException", exception.getMessage());  m.setViewName("error/500");  **return** m;  }  } |

访问：<http://localhost:8082/web/error>

5xx-系统错误

测试运行时异常

访问：<http://localhost:8082/web/error1>

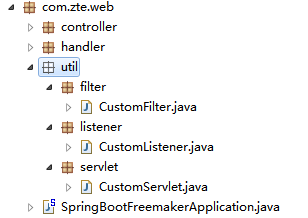
5xx-系统错误

测试普通异常

## 第7节web 应用开发-Servlets, Filters, listeners

### 7.1web常见组件定义

Web 开发使用 Controller 基本上可以完成大部分需求，但是我们还可能会用到 Servlet、 Filter、Listener 等等。



1）CustomServlet

|  |
| --- |
| **public** **class** CustomServlet **extends** HttpServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.***out***.println("servlet get method");  doPost(request, response);  }  @Override  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.***out***.println("servlet post method");  response.getWriter().write("hello world");  }  } |

2）CustomFilter

|  |
| --- |
| **public** **class** CustomFilter **implements** Filter {  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {  System.***out***.println("init filter");  }  @Override  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  **throws** IOException, ServletException {  System.***out***.println("do filter");  chain.doFilter(request, response);  }  @Override  **public** **void** destroy() {  System.***out***.println("destroy filter");  }  } |

3）CustomListener

|  |
| --- |
| **public** **class** CustomListener **implements** ServletContextListener {  @Override  **public** **void** contextInitialized(ServletContextEvent sce) {  System.***out***.println("contextInitialized");  }  @Override  **public** **void** contextDestroyed(ServletContextEvent sce) {  System.***out***.println("contextDestroyed");  }  } |

### 7.2在 spring boot 中的三种实现方式

#### 方法一：通过注册Bean获得控制

通过注册 ServletRegistrationBean、 FilterRegistrationBean 和ServletListenerRegistrationBean 获得控制

1）App

|  |
| --- |
| @SpringBootApplication  **public** **class** SpringBootFreemakerApplication {  @Bean  **public** ServletRegistrationBean servletRegistrationBean() {  **return** **new** ServletRegistrationBean(**new** CustomServlet(), "/webapp");  }  @Bean  **public** FilterRegistrationBean filterRegistrationBean() {  //只拦截ServletRegistrationBean  /\*return new FilterRegistrationBean(new CustomFilter(), servletRegistrationBean());\*/  //拦截全局  **return** **new** FilterRegistrationBean(**new** CustomFilter());  }  @Bean  **public** ServletListenerRegistrationBean<CustomListener> servletListenerRegistrationBean() {  **return** **new** ServletListenerRegistrationBean<CustomListener>(**new** CustomListener());  }  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

启动SpringBoot应用程序：

控制台打印启动过程包含

2018-05-11 23:59:49.194 |-INFO \*\*\*

contextInitialized

init filter

2018-05-11 23:59:49.194 |-INFO \*\*\*

访问：<http://localhost:8082/webapp>

控制台打印：

2018-05-11 23:59:49.194 |-INFO \*\*\*

do filter

servlet get method

servlet post method

关闭SpringBoot应用程序：

控制台打印关闭过程包含

2018-05-11 23:59:49.194 |-INFO \*\*\*

destroy filter

contextDestroyed

#### 方法二：通过实现 ServletContextInitializer 接口直接注册

1. App

|  |
| --- |
| @SpringBootApplication  **public** **class** SpringBootFreemakerApplication **implements** ServletContextInitializer {  @Override  **public** **void** onStartup(ServletContext servletContext) **throws** ServletException {  servletContext.addServlet("customServlet", **new** CustomServlet()).addMapping("/webapp");  servletContext.addFilter("customFilter", **new** CustomFilter())  .addMappingForServletNames(EnumSet.*of*(DispatcherType.***REQUEST***), **true**, "customServlet");  servletContext.addListener(**new** CustomListener());  }  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

启动SpringBoot应用程序：

控制台打印启动过程包含

2018-05-11 23:59:49.194 |-INFO \*\*\*

contextInitialized

init filter

2018-05-11 23:59:49.194 |-INFO \*\*\*

访问：<http://localhost:8082/webapp>

控制台打印：

2018-05-11 23:59:49.194 |-INFO \*\*\*

do filter

servlet get method

servlet post method

关闭SpringBoot应用程序：

控制台打印关闭过程包含

2018-05-11 23:59:49.194 |-INFO \*\*\*

destroy filter

contextDestroyed

#### 方法三：通过注解自动注册（推荐）

在 SpringBootApplication 上使用@ServletComponentScan 注解后，直接通过@WebServlet、@WebFilter、@WebListener 注解自动注册

1. App

|  |
| --- |
| @ServletComponentScan  @SpringBootApplication  **public** **class** SpringBootFreemakerApplication{  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

2）CustomServlet

|  |
| --- |
| @WebServlet(urlPatterns="/webapp", name="customServlet")  **public** **class** CustomServlet **extends** HttpServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.***out***.println("servlet get method");  doPost(request, response);  }  @Override  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.***out***.println("servlet post method");  response.getWriter().write("hello world");  }  } |

3）CustomFilter

|  |
| --- |
| @WebFilter(urlPatterns="/webapp")  **public** **class** CustomFilter **implements** Filter {  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {  System.***out***.println("init filter");  }  @Override  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  **throws** IOException, ServletException {  System.***out***.println("do filter");  chain.doFilter(request, response);  }  @Override  **public** **void** destroy() {  System.***out***.println("destroy filter");  }  } |

4）CustomListener

|  |
| --- |
| @WebListener  **public** **class** CustomListener **implements** ServletContextListener {  @Override  **public** **void** contextInitialized(ServletContextEvent sce) {  System.***out***.println("contextInitialized");  }  @Override  **public** **void** contextDestroyed(ServletContextEvent sce) {  System.***out***.println("contextDestroyed");  }  } |

## 第8节web应用开发-CORS 支持

### 8.1跨域解决方案

Web 开发经常会遇到跨域问题，解决方案有：jsonp，iframe,CORS 等等

CORS 与 JSONP 相比

1、 JSONP 只能实现 GET 请求，而 CORS 支持所有类型的 HTTP 请求。

2、 使用 CORS，开发者可以使用普通的 XMLHttpRequest 发起请求和获得数据，比起 JSONP 有更好的错误处理。

3、 JSONP 主要被老的浏览器支持，它们往往不支持 CORS，而绝大多数现代浏览器都已经支持了 CORS

浏览器支持情况

Chrome 3+

Firefox 3.5+

Opera 12+

Safari 4+

Internet Explorer 8+

### 8.2跨域请求及接口定义

1）运行在test配置的8081端口的ApiController

|  |
| --- |
| @RestController  @RequestMapping("/api")  **public** **class** ApiController {  @RequestMapping(value = "/get")  **public** HashMap<String, Object> get(@RequestParam String name) {  HashMap<String, Object> map = **new** HashMap<String, Object>();  map.put("title", "hello world");  map.put("name", name);  **return** map;  }  } |

2）运行在dev配置的8082端口的POST请求

|  |
| --- |
| $.ajax({  url: "http://localhost:8081/api/get", type: "POST",  data: {  name: "测试"  },  success: function(data, status, xhr) {  console.log(data);  alert(data.name);  }  }); |

访问：<http://localhost:8082/web/index>

页面上单击title，打开调试窗口F12：

Failed to load http://localhost:8081/api/get: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:8082' is therefore not allowed access.

### 8.2spring MVC跨域配置方法

在 spring MVC 中可以配置全局的规则，也可以使用@CrossOrigin 注解进行细粒度的配置。

#### 方法一：全局配置--bean

|  |
| --- |
| @Configuration  **public** **class** CustomCorsConfiguration {  // Springboot2.0使用的是Spring5.0,Spring5.0中WebMvcConfigurerAdapter算过时的，不再建议使用，那么我们可以使用WebMvcConfigurationSupport来代替。  @Bean  **public** WebMvcConfigurer corsConfigurer() {  **return** **new** ~~WebMvcConfigurerAdapter~~() {  @Override  **public** **void** ~~addCorsMappings~~(CorsRegistry registry) {  registry.addMapping("/api/\*\*").allowedOrigins("http://localhost:8082");  }  };  }  } |

#### 方法二：全局配置--继承

|  |
| --- |
| @Configuration  **public** **class** CustomCorsConfiguration2 **extends** WebMvcConfigurationSupport {  @Override  **public** **void** addCorsMappings(CorsRegistry registry) {  registry.addMapping("/api/\*\*").allowedOrigins("http://localhost:8082");  }  } |

#### 方法三：细粒度配置--注解

|  |
| --- |
| @RestController  @RequestMapping(value = "/api", method = RequestMethod.***POST***)  **public** **class** ApiController {  @CrossOrigin(origins = "http://localhost:8082")  @RequestMapping(value = "/get")  **public** HashMap<String, Object> get(@RequestParam String name) {  HashMap<String, Object> map = **new** HashMap<String, Object>();  map.put("title", "hello world");  map.put("name", name);  **return** map;  }  } |

## 第9节web 应用开发-文件上传

Spring Boot 默认使用 springMVC 包装好的解析器进行上传。

### 9.1上传文件表单

|  |
| --- |
| <form method="POST" enctype="multipart/form-data" action="/file/upload">  文件：<input type="file" name="zteFile" /> <input type="submit" value="上传" />  </form> |

### 9.2处理上传文件FileController

|  |
| --- |
| @Controller  @RequestMapping(value = "/file")  **public** **class** FileController {  **private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(FileController.**class**);  @RequestMapping(value = "upload")  @ResponseBody  **public** ModelMap upload(@RequestParam("zteFile") MultipartFile file) {  ModelMap map = **new** ModelMap();  **if** (file.isEmpty()) {  map.put("result", "文件为空");  **return** map;  }  // 获取文件名  String fileName = file.getOriginalFilename();  ***logger***.info("上传的文件名为：" + fileName);  // 获取文件的后缀名  String suffixName = fileName.substring(fileName.lastIndexOf("."));  ***logger***.info("上传的后缀名为：" + suffixName);  // 文件上传路径  String filePath = "d:/zte/test/";  // 解决中文问题，liunx 下中文路径，图片显示问题  //fileName = UUID.randomUUID() + suffixName;  File dest = **new** File(filePath + fileName);  // 检测是否存在目录  **if** (!dest.getParentFile().exists()) {  dest.getParentFile().mkdirs();  }  **try** {  file.transferTo(dest);  map.put("result", "上传成功");  **return** map;  } **catch** (IllegalStateException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  map.put("result", "上传失败");  **return** map;  }  } |

#### 注意：默认上传文件大小限制大约1M，超过则会抛异常：

5xx-系统错误

Maximum upload size exceeded; nested exception is java.lang.IllegalStateException: org.apache.tomcat.util.http.fileupload.FileUploadBase$FileSizeLimitExceededException: The field zteFile exceeds its maximum permitted size of 1048576 bytes.

### 9.3上传文件配置

|  |
| --- |
| #默认支持文件上传.  spring.servlet.multipart.enabled=true  #支持文件写入磁盘.  spring.servlet.multipart.file-size-threshold=0  # 上传文件的临时目录,目录需要已存在  spring.servlet.multipart.location= d:/tmp  # 最大支持文件大小  spring.servlet.multipart.max-file-size=10Mb  # 最大支持请求大小  spring.servlet.multipart.max-request-size=10Mb |

## 第10节使用 SQL 关系型数据库-JdbcTemplate

### 10.1配置数据源

嵌入式数据库的支持：Spring Boot 可以自动配置 H2, HSQL and Derby 数据库，不需要提供任何的链接 URLs，只需要加入相应的 jar 包，Spring boot 可以自动发现装配。

1. 引入数据库依赖

|  |
| --- |
| <!-- 数据库 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency> |

2）/spring-boot-jdbcTemplate/src/main/resources/application-dev.properties增加配置项

|  |
| --- |
| #数据库配置  spring.datasource.url=jdbc:mysql://localhost/spring\_boot\_demo?useUnicode=true&characterEncoding=utf-8  spring.datasource.username=root  spring.datasource.password=1234  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

3）建表sql

|  |
| --- |
| CREATE TABLE `zte\_user` (  `id` INT (11) NOT NULL AUTO\_INCREMENT,  `name` VARCHAR (255) DEFAULT NULL,  `create\_time` DATETIME DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE = INNODB AUTO\_INCREMENT = 1 DEFAULT CHARSET = utf8 COMMENT = '用户表' ; |

注意：

1. 可以不指定 driver-class-name，spring boot 会自动识别 url。
2. 数据连接池默认使用 tomcat-jdbc。

连接池的配置： spring.datasource.tomcat.\*

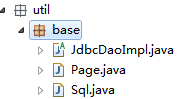
### 10.2JdbcTemplate 模板

// 自动注册

@Autowired

private JdbcTemplate jdbcTemplate;

### 10.3JdbcTemplate封装



1）spirng jdbc 基本支撑类AbstractJdbcDao

|  |
| --- |
| /\*\*  \* spirng jdbc 基本支撑类  \*  \*/  **public** **abstract** **class** AbstractJdbcDao {  @Autowired  **protected** JdbcTemplate jdbcTemplate;  /\*\*  \* 获取当前事务最后一次更新的主键值  \*/  **public** Long getLastId() {  **return** jdbcTemplate.queryForObject("select last\_insert\_id() as id", Long.**class**);  }  /\*\*  \* 获取对象信息  \*/  **public** <T> T queryForObject(String sql, Class<T> clazz, Object... args) {  Assert.*hasText*(sql, "sql 语句不能为空");  **return** jdbcTemplate.queryForObject(sql, **new** BeanPropertyRowMapper<T>(clazz), args);  }  /\*\*  \* 获取对象集合信息  \*/  **public** <T> List<T> queryForObjectList(String sql, Class<T> clazz, Object... args) {  Assert.*hasText*(sql, "sql 语句不能为空");  **return** jdbcTemplate.query(sql, args, **new** BeanPropertyRowMapper<T>(clazz));  }  /\*\*  \* 分页，jdbcTemplate 不支持like  \*/  **public** Page<Map<String, Object>> queryForPage(String sql, **int** pageCurrent, **int** pageSize, Object... args) {  Assert.*hasText*(sql, "sql 语句不能为空");  Assert.*isTrue*(pageCurrent >= 1, "pageNo 必须大于等于1");  String sqlCount = Sql.*countSql*(sql);  **int** count = jdbcTemplate.queryForObject(sqlCount, Integer.**class**, args);  pageCurrent = Sql.*checkPageCurrent*(count, pageSize, pageCurrent);  pageSize = Sql.*checkPageSize*(pageSize);  **int** totalPage = Sql.*countTotalPage*(count, pageSize);  String sqlList = sql + Sql.*limitSql*(count, pageCurrent, pageSize);  List<Map<String, Object>> list = jdbcTemplate.queryForList(sqlList, args);  **return** **new** Page<Map<String, Object>>(count, totalPage, pageCurrent, pageSize, list);  }  /\*\*  \* 分页，jdbcTemplate 不支持like  \*/  **public** <T> Page<T> queryForPage(String sql, **int** pageCurrent, **int** pageSize, Class<T> clazz, Object... args) {  Assert.*hasText*(sql, "sql 语句不能为空");  Assert.*isTrue*(pageCurrent >= 1, "pageNo 必须大于等于1");  Assert.*isTrue*(clazz != **null**, "clazz 不能为空");  String sqlCount = Sql.*countSql*(sql);  **int** count = jdbcTemplate.queryForObject(sqlCount, Integer.**class**, args);  pageCurrent = Sql.*checkPageCurrent*(count, pageSize, pageCurrent);  pageSize = Sql.*checkPageSize*(pageSize);  **int** totalPage = Sql.*countTotalPage*(count, pageSize);  String sqlList = sql + Sql.*limitSql*(count, pageCurrent, pageSize);  List<T> list = jdbcTemplate.query(sqlList, **new** BeanPropertyRowMapper<T>(clazz), args);  **return** **new** Page<T>(count, totalPage, pageCurrent, pageSize, list);  }  } |

2）数据分页组件Page<T>

|  |
| --- |
| /\*\*  \* 数据分页组件  \*/  **public** **class** Page<T> **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -5764853545343945831L;  /\*\*  \* 默认每页记录数(20)  \*/  **public** **static** **final** **int** ***DEFAULT\_PAGE\_SIZE*** = 20;  /\*\*  \* 最大每页记录数(1000)  \*/  **public** **static** **final** **int** ***MAX\_PAGE\_SIZE*** = 1000;  /\*\*  \* 当前分页的数据集  \*/  **private** List<T> list;  /\*\*  \* 总记录数  \*/  **private** **int** totalCount;  /\*\*  \* 总页数  \*/  **private** **int** totalPage;  /\*\*  \* 当前页  \*/  **private** **int** pageCurrent;  /\*\*  \* 每页记录数  \*/  **private** **int** pageSize;  /\*\*  \* 排序字段  \*/  **private** String orderField;  /\*\*  \* 排序方式：asc or desc  \*/  **private** String orderDirection;  /\*\*  \* 默认构造函数  \*/  **public** Page() {  }  /\*\*  \* 构造函数  \*  \* **@param** totalCount  \* 总记录数  \* **@param** totalPage  \* 总页数  \* **@param** pageCurrent  \* **@param** pageSize  \* **@param** list  \*/  **public** Page(**int** totalCount, **int** totalPage, **int** pageCurrent, **int** pageSize, List<T> list) {  **this**.totalCount = totalCount;  **this**.totalPage = totalPage;  **this**.pageCurrent = pageCurrent;  **this**.pageSize = pageSize;  **this**.list = list;  }  **public** List<T> getList() {  **return** list;  }  **public** **void** setList(List<T> list) {  **this**.list = list;  }  **public** **int** getTotalCount() {  **return** totalCount;  }  **public** **void** setTotalCount(**int** totalCount) {  **this**.totalCount = totalCount;  }  **public** **int** getTotalPage() {  **return** totalPage;  }  **public** **void** setTotalPage(**int** totalPage) {  **this**.totalPage = totalPage;  }  **public** **int** getPageCurrent() {  **return** pageCurrent;  }  **public** **void** setPageCurrent(**int** pageCurrent) {  **this**.pageCurrent = pageCurrent;  }  **public** **int** getPageSize() {  **return** pageSize;  }  **public** **void** setPageSize(**int** pageSize) {  **this**.pageSize = pageSize;  }  **public** String getOrderField() {  **return** orderField;  }  **public** **void** setOrderField(String orderField) {  **this**.orderField = orderField;  }  **public** String getOrderDirection() {  **return** orderDirection;  }  **public** **void** setOrderDirection(String orderDirection) {  **this**.orderDirection = orderDirection;  }  } |

3）sql工具类Sql

|  |
| --- |
| /\*\*  \* sql工具类  \*  \*/  **public** **class** Sql {  **private** Sql() {  }  /\*\*  \* 检测sql，防止sql注入  \*  \* **@param** sql  \* sql  \* **@return** 正常返回sql；异常返回""  \*/  **public** **static** String checkSql(String sql) {  String inj\_str = "'|and|exec|insert|select|delete|update|count|\*|%|chr|mid|master|truncate|char|declare|;|or|-|+|,";  String inj\_stra[] = inj\_str.split("\\|");  **for** (**int** i = 0; i < inj\_stra.length; i++) {  **if** (sql.indexOf(inj\_stra[i]) >= 0) {  **return** "";  }  }  **return** sql;  }  /\*\*  \* 计算总页数  \*  \* **@param** totalCount  \* 总记录数.  \* **@param** pageSize  \* 每页记录数.  \* **@return** totalPage 总页数.  \*/  **public** **static** **int** countTotalPage(**final** **int** totalCount, **final** **int** pageSize) {  **if** (totalCount % pageSize == 0) {  **return** totalCount / pageSize; // 刚好整除  } **else** {  **return** totalCount / pageSize + 1; // 不能整除则总页数为：商 + 1  }  }  /\*\*  \* 校验当前页数pageCurrent<br/>  \* 1、先根据总记录数totalCount和每页记录数pageSize，计算出总页数totalPage<br/>  \* 2、判断页面提交过来的当前页数pageCurrent是否大于总页数totalPage，大于则返回totalPage<br/>  \* 3、判断pageCurrent是否小于1，小于则返回1<br/>  \* 4、其它则直接返回pageCurrent  \*  \* **@param** totalCount  \* 要分页的总记录数  \* **@param** pageSize  \* 每页记录数大小  \* **@param** pageCurrent  \* 输入的当前页数  \* **@return** pageCurrent  \*/  **public** **static** **int** checkPageCurrent(**int** totalCount, **int** pageSize, **int** pageCurrent) {  **int** totalPage = *countTotalPage*(totalCount, pageSize); // 最大页数  **if** (pageCurrent > totalPage) {  // 如果页面提交过来的页数大于总页数，则将当前页设为总页数  // 此时要求totalPage要大于获等于1  **if** (totalPage < 1) {  **return** 1;  }  **return** totalPage;  } **else** **if** (pageCurrent < 1) {  **return** 1; // 当前页不能小于1（避免页面输入不正确值）  } **else** {  **return** pageCurrent;  }  }  /\*\*  \* 校验页面输入的每页记录数pageSize是否合法<br/>  \* 1、当页面输入的每页记录数pageSize大于允许的最大每页记录数MAX\_PAGE\_SIZE时，返回MAX\_PAGE\_SIZE  \* 2、如果pageSize小于1，则返回默认的每页记录数DEFAULT\_PAGE\_SIZE  \*  \* **@param** pageSize  \* 页面输入的每页记录数  \* **@return** checkPageSize  \*/  **public** **static** **int** checkPageSize(**int** pageSize) {  **if** (pageSize > Page.***MAX\_PAGE\_SIZE***) {  **return** Page.***MAX\_PAGE\_SIZE***;  } **else** **if** (pageSize < 1) {  **return** Page.***DEFAULT\_PAGE\_SIZE***;  } **else** {  **return** pageSize;  }  }  /\*\*  \* 计算当前分页的开始记录的索引  \*  \* **@param** pageCurrent  \* 当前第几页  \* **@param** pageSize  \* 每页记录数  \* **@return** 当前页开始记录号  \*/  **public** **static** **int** countOffset(**final** **int** pageCurrent, **final** **int** pageSize) {  **return** (pageCurrent - 1) \* pageSize;  }  /\*\*  \* 根据总记录数，对页面传来的分页参数进行校验，并返分页的SQL语句  \*  \* **@param** pageCurrent  \* 当前页  \* **@param** pageSize  \* 每页记录数  \* **@param** pageBean  \* DWZ分页查询参数  \* **@return** limitSql  \*/  **public** **static** String limitSql(**int** totalCount, **int** pageCurrent, **int** pageSize) {  // 校验当前页数  pageCurrent = *checkPageCurrent*(totalCount, pageSize, pageCurrent);  pageSize = *checkPageSize*(pageSize); // 校验每页记录数  **return** " limit " + *countOffset*(pageCurrent, pageSize) + "," + pageSize;  }  /\*\*  \* 根据分页查询的SQL语句，获取统计总记录数的语句  \*  \* **@param** sql  \* 分页查询的SQL  \* **@return** countSql  \*/  **public** **static** String countSql(String sql) {  String countSql = sql.substring(sql.toLowerCase().indexOf("from")); // 去除第一个from前的内容  **return** "select count(\*) " + *removeOrderBy*(countSql);  }  /\*\*  \* 移除SQL语句中的的order by子句（用于分页前获取总记录数，不需要排序）  \*  \* **@param** sql  \* 原始SQL  \* **@return** 去除order by子句后的内容  \*/  **private** **static** String removeOrderBy(String sql) {  Pattern pat = Pattern.*compile*("order\\s\*by[\\w|\\W|\\s|\\S]\*", Pattern.***CASE\_INSENSITIVE***);  Matcher mc = pat.matcher(sql);  StringBuffer strBuf = **new** StringBuffer();  **while** (mc.find()) {  mc.appendReplacement(strBuf, "");  }  mc.appendTail(strBuf);  **return** strBuf.toString();  }  } |

### 10.3实体类、接口、实现类及测试类代码

1）实体类ZteUser

|  |
| --- |
| **public** **class** ZteUser {  **private** **int** id;  **private** String name;  **private** Date createTime;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Date getCreateTime() {  **return** createTime;  }  **public** **void** setCreateTime(Date createTime) {  **this**.createTime = createTime;  }  @Override  **public** String toString() {  **return** "ZteUser [id=" + id + ", name=" + name + ", createTime=" + createTime  + "]";  }  } |

2）DAO接口IZteUserDao

|  |
| --- |
| **public** **interface** IZteUserDao {  **int** insert(ZteUser zteUser);  **int** deleteById(**int** id);  **int** updateById(ZteUser zteUser);  ZteUser selectById(**int** id);    Page<ZteUser> queryForPage(**int** pageCurrent, **int** pageSize, String name);  } |

3）实现类ZteUserDaoImpl

|  |
| --- |
| @Repository  **public** **class** ZteUserDaoImpl **extends** JdbcDaoImpl **implements** IZteUserDao {  @Autowired  **private** JdbcTemplate jdbcTemplate;  @Override  **public** **int** insert(ZteUser zteUser) {  String sql = "insert into zte\_user (name, create\_time) values (?, ?)";  **return** jdbcTemplate.update(sql, zteUser.getName(), zteUser.getCreateTime());  }  @Override  **public** **int** deleteById(**int** id) {  String sql = "delete from zte\_user where id=?";  **return** jdbcTemplate.update(sql, id);  }  @Override  **public** **int** updateById(ZteUser zteUser) {  String sql = "update zte\_user set name=?, create\_time=? where id=?";  **return** jdbcTemplate.update(sql, zteUser.getName(), zteUser.getCreateTime(), zteUser.getId());  }  @Override  **public** ZteUser selectById(**int** id) {  String sql = "select \* from zte\_user where id=?";    // 自己拼接  /\*return jdbcTemplate.queryForObject(sql, new RowMapper<ZteUser>() {  @Override  public ZteUser mapRow(ResultSet rs, int rowNum) throws SQLException {  ZteUser zteUser = new ZteUser();  zteUser.setId(rs.getInt("id"));  zteUser.setName(rs.getString("name"));  zteUser.setCreateTime(rs.getDate("create\_time"));  return zteUser;  }  }, id);\*/    // 使用封装方法拼接  **return** queryForObject(sql, ZteUser.**class**, id);  }  @Override  **public** Page<ZteUser> queryForPage(**int** pageCurrent, **int** pageSize, String name) {  // 若要like查询，如下  StringBuffer sql = **new** StringBuffer(" select \* from zte\_user where 1 ");  **if**(!StringUtils.*isNullOrEmpty*(name)){  // Sql.checkSql防止sql注入  sql.append(" and name like '%").append(Sql.*checkSql*(name)).append("%' ");  }  **return** queryForPage(sql.toString(), pageCurrent, pageSize, ZteUser.**class**);  }  } |

4）

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootFreemakerApplicationTests {  @Test  **public** **void** contextLoads() {  }  @Autowired  **private** ZteUserDaoImpl zteUserDao;  @Test  **public** **void** insert() {  ZteUser zteUser = **new** ZteUser();  zteUser.setName("测试");  zteUser.setCreateTime(**new** Date());  **int** result = zteUserDao.insert(zteUser);  System.***out***.println(result);  }  @Test  **public** **void** delete() {  **int** result = zteUserDao.deleteById(1);  System.***out***.println(result);  }  @Test  **public** **void** update() {  ZteUser zteUser = **new** ZteUser();  zteUser.setId(2);  zteUser.setName("测试 2");  zteUser.setCreateTime(**new** Date());  **int** result = zteUserDao.updateById(zteUser);  System.***out***.println(result);  }  @Test  **public** **void** select() {  ZteUser result = zteUserDao.selectById(2);  System.***out***.println(result);  }    @Test  **public** **void** queryForPage() {  Page<ZteUser> result = zteUserDao.queryForPage(1, 20, "测试");  System.***out***.println(result.getList());  }  } |

### 10.4日志打印 sql 语句

/spring-boot-jdbcTemplate/src/main/resources/logback-spring.xml添加如下：

<logger name="org.springframework.jdbc.core.JdbcTemplate" level="debug"/>

### 10.5测试结果

1. 测试insert

2018-05-12 23:35:57.138 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [848] -| Executing prepared SQL update

2018-05-12 23:35:57.139 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [insert into zte\_user (name, create\_time) values (?, ?)]

2018-05-12 23:35:57.145 |-INFO [main] com.zaxxer.hikari.HikariDataSource [110] -| HikariPool-1 - Starting...

2018-05-12 23:35:57.479 |-INFO [main] com.zaxxer.hikari.HikariDataSource [123] -| HikariPool-1 - Start completed.

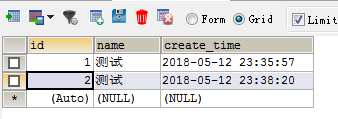
2018-05-12 23:35:57.530 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [857] -| SQL update affected 1 rows

1

2018-05-12 23:44:48.775 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [993] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@43f02ef2: startup date [Sat May 12 23:44:44 CST 2018]; root of context hierarchy

2018-05-12 23:44:48.779 |-INFO [Thread-3] com.zaxxer.hikari.HikariDataSource [381] -| HikariPool-1 - Shutdown initiated...

2018-05-12 23:44:48.785 |-INFO [Thread-3] com.zaxxer.hikari.HikariDataSource [383] -| HikariPool-1 - Shutdown completed.



1. 测试delete

2018-05-12 23:39:23.650 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [848] -| Executing prepared SQL update

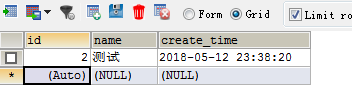
2018-05-12 23:39:23.651 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [delete from zte\_user where id=?]

2018-05-12 23:39:23.658 |-INFO [main] com.zaxxer.hikari.HikariDataSource [110] -| HikariPool-1 - Starting...

2018-05-12 23:39:24.020 |-INFO [main] com.zaxxer.hikari.HikariDataSource [123] -| HikariPool-1 - Start completed.

2018-05-12 23:39:24.058 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [857] -| SQL update affected 1 rows

1



3）测试update

2018-05-12 23:40:25.241 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [848] -| Executing prepared SQL update

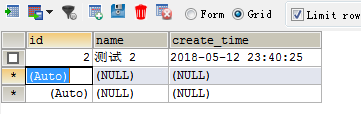
2018-05-12 23:40:25.243 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [update zte\_user set name=?, create\_time=? where id=?]

2018-05-12 23:40:25.251 |-INFO [main] com.zaxxer.hikari.HikariDataSource [110] -| HikariPool-1 - Starting...

2018-05-12 23:40:25.550 |-INFO [main] com.zaxxer.hikari.HikariDataSource [123] -| HikariPool-1 - Start completed.

2018-05-12 23:40:25.627 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [857] -| SQL update affected 1 rows

1



4）测试select

2018-05-12 23:41:23.523 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [655] -| Executing prepared SQL query

2018-05-12 23:41:23.525 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [select \* from zte\_user where id=?]

2018-05-12 23:41:23.532 |-INFO [main] com.zaxxer.hikari.HikariDataSource [110] -| HikariPool-1 - Starting...

2018-05-12 23:41:23.888 |-INFO [main] com.zaxxer.hikari.HikariDataSource [123] -| HikariPool-1 - Start completed.

ZteUser [id=2, name=测试 2, createTime=2018-05-12 23:40:25.0]

1. 测试分页查询

2018-05-12 23:42:21.967 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [655] -| Executing prepared SQL query

2018-05-12 23:42:21.968 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [select count(\*) from zte\_user where 1 and name like '%测试%' ]

2018-05-12 23:42:21.977 |-INFO [main] com.zaxxer.hikari.HikariDataSource [110] -| HikariPool-1 - Starting...

2018-05-12 23:42:22.323 |-INFO [main] com.zaxxer.hikari.HikariDataSource [123] -| HikariPool-1 - Start completed.

2018-05-12 23:42:22.380 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [655] -| Executing prepared SQL query

2018-05-12 23:42:22.381 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [597] -| Executing prepared SQL statement [ select \* from zte\_user where 1 and name like '%测试%' limit 0,20]

[ZteUser [id=2, name=测试 2, createTime=2018-05-12 23:40:25.0]]

### 10.6封装 spring jdbc，带分页

继承AbstractJdbcDao基本支撑类，覆写方法queryForPage，弥补JdbcTemplate不支持like查询，使用拼装查询sql增加like条件。

|  |
| --- |
| @Override  **public** Page<ZteUser> queryForPage(**int** pageCurrent, **int** pageSize, String name) {  // 若要like查询，如下  StringBuffer sql = **new** StringBuffer(" select \* from zte\_user where 1 ");  **if**(!StringUtils.*isNullOrEmpty*(name)){  // Sql.checkSql防止sql注入  sql.append(" and name like '%").append(Sql.*checkSql*(name)).append("%' ");  }  **return** queryForPage(sql.toString(), pageCurrent, pageSize, ZteUser.**class**);  } |

## 第11节使用 SQL 关系型数据库-spring-data-jpa

### 11.1配置数据源

1. Pom.xml使用SpringBoot1.4.1版本，在2.0中启动，即使在实体类加了注解@Entity，也会报无法找到entityManagerFactory错误。

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.4.0.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent> |

1. 替换JdbcTemplate的依赖为spring-boot-starter-data-jpa。

|  |
| --- |
| <!-- 数据库 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency> |

### 11.2 Jpa 配置

/spring-boot-jpa/src/main/resources/application-dev.properties增加jpa相关配置：

|  |
| --- |
| # JPA  spring.jpa.hibernate.ddl-auto= update  #显示 sql 语句  spring.jpa.show-sql=true |

### 11.3 实体类、接口及测试类

1）实体类ZteUserLog

|  |
| --- |
| @Entity  **public** **class** ZteUserLog {  @Id  @GeneratedValue  **private** Integer id;  @Column  **private** Date createTime;  @Column  **private** String userName;  @Column  **private** String userIp;  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** Date getCreateTime() {  **return** createTime;  }  **public** **void** setCreateTime(Date createTime) {  **this**.createTime = createTime;  }  **public** String getUserName() {  **return** userName;  }  **public** **void** setUserName(String userName) {  **this**.userName = userName;  }  **public** String getUserIp() {  **return** userIp;  }  **public** **void** setUserIp(String userIp) {  **this**.userIp = userIp;  }  @Override  **public** String toString() {  **return** "ZteUserLog [id=" + id + ", createTime=" + createTime + ", userName=" + userName + ", userIp=" + userIp + "]";  }  } |

2）ZteUserLogDao

|  |
| --- |
| **public** **interface** ZteUserLogDao **extends** JpaRepository<ZteUserLog, Integer> {  /\*\*  \* **@param** string  \* **@return**  \*/  @Query(value = "select u from ZteUserLog u where u.userName=?1")  List<ZteUserLog> findByUserName(String userName);  /\*\*  \* **@param** string  \* **@param** string2  \* **@return**  \*/  List<ZteUserLog> findByUserNameAndUserIp(String string, String string2);  /\*\*  \* **@param** exampl  \* **@param** pageable  \* **@return**  \*/  Page<ZteUserLog> findByUserName(String userName, Pageable pageable);  } |

3）

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootFreemakerApplicationTests {  @Autowired  **private** ZteUserLogDao zteUserLogDao;  @Test  **public** **void** insert() {  ZteUserLog entity = **new** ZteUserLog();  entity.setUserName("杰克");  entity.setUserIp("192.168.0.1");  entity.setCreateTime(**new** Date());  zteUserLogDao.save(entity);  }  @Test  **public** **void** delete() {  zteUserLogDao.delete(1);  }  @Test  **public** **void** update() {  ZteUserLog entity = **new** ZteUserLog();  entity.setId(2);  entity.setUserName("杰克2");  entity.setUserIp("192.168.0.1");  entity.setCreateTime(**new** Date());  zteUserLogDao.save(entity);  }  @Test  **public** **void** select() {  ZteUserLog result = zteUserLogDao.findOne(2);  System.***out***.println(result);  }  @Test  **public** **void** select2() {  List<ZteUserLog> result = zteUserLogDao.findByUserName("杰克2");  System.***out***.println(result);  }  @Test  **public** **void** select3() {  List<ZteUserLog> result = zteUserLogDao.findByUserNameAndUserIp("杰克2", "192.168.0.1");  System.***out***.println(result);  }  // 分页  @Test  **public** **void** queryForPage() {  Pageable pageable = **new** PageRequest(0, 20, **new** Sort(**new** Order(Direction.***DESC***, "id")));  Page<ZteUserLog> result = zteUserLogDao.findByUserName("杰克2", pageable);  System.***out***.println(result.getContent());  }  } |

### 11.4 测试结果

1）测试insert，此时不需要新建实体类ZteUserLog类对应的表，在测试insert时，jpa会根据实体类中的相关注解（@Id、@Column等）先建表，然后再向该表插入数据。

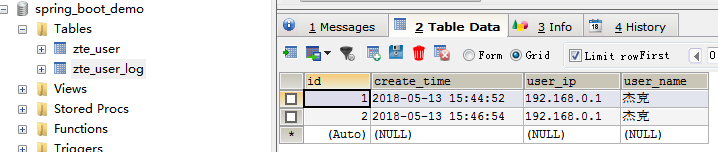
2018-05-13 15:44:51.944 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 9.561 seconds (JVM running for 10.468)

Hibernate: insert into zte\_user\_log (create\_time, user\_ip, user\_name) values (?, ?, ?)

2018-05-13 15:44:52.228 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@35841320: startup date [Sun May 13 15:44:43 CST 2018]; root of context hierarchy

2018-05-13 15:44:52.234 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

测试两次，第一次建表并插入记录，第二次仅插入数据。



2）测试delete

2018-05-13 15:47:20.719 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 9.598 seconds (JVM running for 10.484)

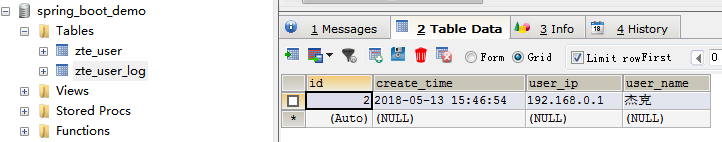
Hibernate: select zteuserlog0\_.id as id1\_0\_0\_, zteuserlog0\_.create\_time as create\_t2\_0\_0\_, zteuserlog0\_.user\_ip as user\_ip3\_0\_0\_, zteuserlog0\_.user\_name as user\_nam4\_0\_0\_ from zte\_user\_log zteuserlog0\_ where zteuserlog0\_.id=?

Hibernate: delete from zte\_user\_log where id=?

2018-05-13 15:47:20.981 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@35841320: startup date [Sun May 13 15:47:11 CST 2018]; root of context hierarchy

2018-05-13 15:47:20.986 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

删除id=1的记录



3）测试update

2018-05-13 15:50:50.480 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 9.597 seconds (JVM running for 10.48)

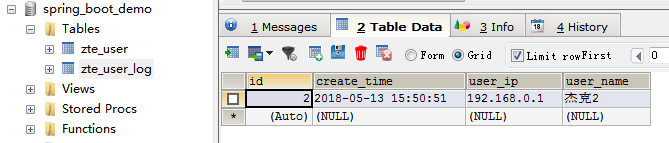
Hibernate: select zteuserlog0\_.id as id1\_0\_0\_, zteuserlog0\_.create\_time as create\_t2\_0\_0\_, zteuserlog0\_.user\_ip as user\_ip3\_0\_0\_, zteuserlog0\_.user\_name as user\_nam4\_0\_0\_ from zte\_user\_log zteuserlog0\_ where zteuserlog0\_.id=?

Hibernate: update zte\_user\_log set create\_time=?, user\_ip=?, user\_name=? where id=?

2018-05-13 15:50:50.745 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@35841320: startup date [Sun May 13 15:50:41 CST 2018]; root of context hierarchy

2018-05-13 15:50:50.752 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

更新id为2的记录



4）接口中增加内置的关检词查询，测试改方法

|  |
| --- |
| /\*\*  \* 查询在某个时间点之前的记录  \* **@param** date  \* **@return**  \*/  List<ZteUserLog> findByCreateTimeBefore(Date date);    /\*\*  \* 查询已某个字符开头的记录  \* **@param** prefix  \* **@return**  \*/  List<ZteUserLog> findByUserNameStartingWith(String prefix); |

|  |
| --- |
| @Test  **public** **void** select4() {  List<ZteUserLog> result = zteUserLogDao.findByCreateTimeBefore(**new** Date());  System.***out***.println(result);  }    @Test  **public** **void** select5() {  List<ZteUserLog> result = zteUserLogDao.findByUserNameStartingWith("杰");  System.***out***.println(result);  } |

### 11.5扩展进行自定义的查询

1）使用内置的关键词查询

<http://docs.spring.io/spring-data/jpa/docs/1.10.2.RELEASE/reference/html/>

2）使用自定义语句查询各种语法，具体查看

<http://docs.spring.io/spring-data/jpa/docs/1.10.2.RELEASE/reference/html/>

自定义优先级比内置的关检词高

|  |
| --- |
| /\*\*  \* **@param** string  \* **@return**  \*/  @Query(value = "select u from ZteUserLog u where u.userName=?1")  List<ZteUserLog> findByUserName(String userName); |

## 第12节使用SQL数据库-事务处理

### 12.1事务有四个特性：ACID

**原子性（Atomicity）**：事务是一个原子操作，由一系列动作组成。事务的原子性确保动作要么全部完成，要么完全不起作用。

**一致性（Consistency）**：一旦事务完成（不管成功还是失败），系统必须确保它所建模的业务处于一致的状态，而不会是部分完成部分失败。在现实中的数据不应该被破坏。

**隔离性（Isolation）**：可能有许多事务会同时处理相同的数据，因此每个事务都应该与其他事务隔离开来，防止数据损坏。

**持久性（Durability）**：一旦事务完成，无论发生什么系统错误，它的结果都不应该受到影响，这样就能从任何系统崩溃中恢复过来。通常情况下，事务的结果被写到持久化存储器中。

### 12.2传播行为

当事务方法被另一个事务方法调用时，必须指定事务应该如何传播。例如：方法可能继续在现有事务中运行，也可能开启一个新事务，并在自己的事务中运行。

Spring 定义了七种传播行为：

**PROPAGATION\_REQUIRED** 表示当前方法必须运行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务，Spring 默认使用；

**PROPAGATION\_SUPPORTS** 表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行；

**PROPAGATION\_MANDATORY**表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常；

**PROPAGATION\_REQUIRED\_NEW** 表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用 JTATransactionManager 的话，则需要访问 TransactionManager

**PROPAGATION\_NOT\_SUPPORTED** 表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用 JTATransactionManager 的话，则需要访问 TransactionManager

**PROPAGATION\_NEVER** 表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常；

**PROPAGATION\_NESTED** 表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION\_REQUIRED 一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务。

### 12.3隔离级别定义了一个事务可能受其他并发事务影响的程度。

**ISOLATION\_DEFAULT** 使用后端数据库默认的隔离级别，Spring 默认使用，mysql 默认的隔离级别为：Repeatable Read(可重复读)；

**ISOLATION\_READ\_UNCOMMITTED** 读未提交，最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读；

**ISOLATION\_READ\_COMMITTED** 读已提交，允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生；

**ISOLATION\_REPEATABLE\_READ** 可重复读，对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生；

**ISOLATION\_SERIALIZABLE** 可串行化，最高的隔离级别，完全服从 ACID 的隔离级别，确保阻止脏读、不可重复读以及幻读，也是最慢的事务隔离级别，因为它通常是通过完全锁定事务相关的数据库表来实现的；

**脏读（Dirty reads）**——脏读发生在一个事务读取了另一个事务改写但尚未提交的数据时。如果改写再稍后被回滚了，那么第一个事务获取的数据就是无效的。

**不可重复读（Nonrepeatable read）**——不可重复读发生在一个事务执行相同的查询两次或两次以上，但是每次都得到不同的数据时。这通常是因为另一个并发事务在两次查询期间进行了更新。

**幻读（Phantom read）**——幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录。

### 12.4属性说明 @Transactional

1. **isolation**：用于指定事务的隔离级别。默认为底层事务的隔离级别。
2. **noRollbackFor**：指定遇到指定异常时强制不回滚事务。
3. **noRollbackForClassName**：指定遇到指定多个异常时强制不回滚事务。该属性可以指定多个异常类名。
4. **propagation**:指定事务的传播属性。
5. **readOnly**：指定事务是否只读。表示这个事务只读取数据但不更新数据，这样可以帮助数据库引擎优化事务。若真的是一个只读取的数据库应设置 readOnly=true
6. **rollbackFor**：指定遇到指定异常时强制回滚事务。
7. **rollbackForClassName**：指定遇到指定多个异常时强制回滚事务。该属性可以指定多个异常类名。
8. **timeout**：指定事务的超时时长。

### 12.5@Transactional测试

1）新增service的包，添加UserService

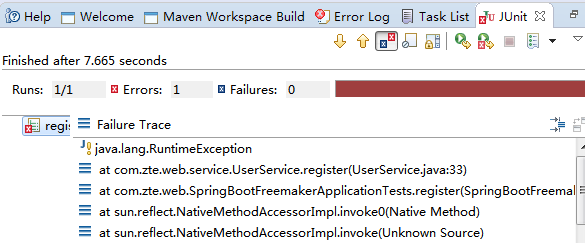
|  |
| --- |
| @Service  **public** **class** UserService {  @Autowired  **private** ZteUserDaoImpl zteUserDao;  @Autowired  **private** ZteUserLogDao zteUserLogDao;  //@Transactional  **public** String register(String name, String ip) {  // 1.添加用户  ZteUser zteUser = **new** ZteUser();  zteUser.setName(name);  zteUser.setCreateTime(**new** Date());  zteUserDao.insert(zteUser);  // 测试使用  **boolean** flag = **true**;  **if** (flag) {  **throw** **new** RuntimeException();  }  // 2.添加注册日志  ZteUserLog zteUserLog = **new** ZteUserLog();  zteUserLog.setUserName(name);  zteUserLog.setUserIp(ip);  zteUserLog.setCreateTime(**new** Date());  zteUserLogDao.save(zteUserLog);  **return** "success";  }  } |

2）测试代码

|  |
| --- |
| @Autowired  **private** UserService userService;  @Test  **public** **void** register() {  String result = userService.register("杰克", "192.168.1.1");  System.***out***.println(result);  } |

3）flag = **true**;测试结果

预期抛出运行时异常：



运行日志：

2018-05-14 22:34:37.610 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 7.483 seconds (JVM running for 8.34)

2018-05-14 22:34:37.666 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [869] -| Executing prepared SQL update

2018-05-14 22:34:37.667 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [616] -| Executing prepared SQL statement [insert into zte\_user (name, create\_time) values (?, ?)]

2018-05-14 22:34:37.701 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [879] -| SQL update affected 1 rows

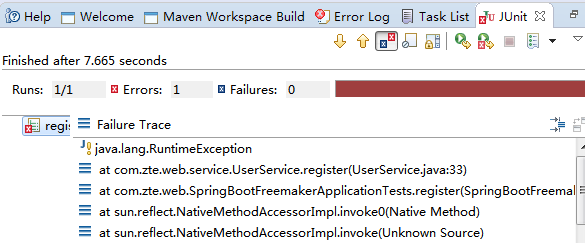
2018-05-14 22:34:37.730 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@451001e5: startup date [Mon May 14 22:34:30 CST 2018]; root of context hierarchy

2018-05-14 22:34:37.735 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

查看数据库发现两张表zte\_user和zte\_user\_log均未插入数据，说明事务回滚。

4）注释//@Transactional测试结果

预期抛出运行时异常：



运行日志：

2018-05-14 22:34:37.610 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 7.483 seconds (JVM running for 8.34)

2018-05-14 22:34:37.666 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [869] -| Executing prepared SQL update

2018-05-14 22:34:37.667 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [616] -| Executing prepared SQL statement [insert into zte\_user (name, create\_time) values (?, ?)]

2018-05-14 22:34:37.701 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [879] -| SQL update affected 1 rows

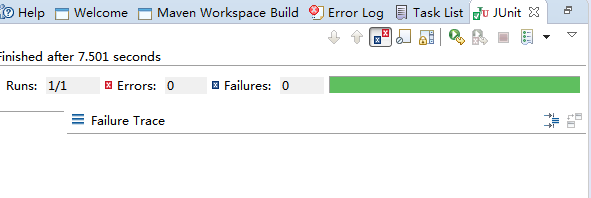
2018-05-14 22:34:37.730 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@451001e5: startup date [Mon May 14 22:34:30 CST 2018]; root of context hierarchy

2018-05-14 22:34:37.735 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

查看数据库发现两张表zte\_user插入数据，说明事务提交，zte\_user\_log没有记录，程序抛异常退出。

5）flag = **false**;测试结果

预期JUnit成功执行



运行日志

2018-05-14 22:38:50.797 |-INFO [main] com.zte.web.SpringBootFreemakerApplicationTests [57] -| Started SpringBootFreemakerApplicationTests in 7.223 seconds (JVM running for 8.11)

2018-05-14 22:38:50.857 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [869] -| Executing prepared SQL update

2018-05-14 22:38:50.859 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [616] -| Executing prepared SQL statement [insert into zte\_user (name, create\_time) values (?, ?)]

2018-05-14 22:38:50.887 |-DEBUG [main] org.springframework.jdbc.core.JdbcTemplate [879] -| SQL update affected 1 rows

success

2018-05-14 22:38:51.008 |-INFO [Thread-3] org.springframework.web.context.support.GenericWebApplicationContext [982] -| Closing org.springframework.web.context.support.GenericWebApplicationContext@451001e5: startup date [Mon May 14 22:38:44 CST 2018]; root of context hierarchy

2018-05-14 22:38:51.012 |-INFO [Thread-3] org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean [551] -| Closing JPA EntityManagerFactory for persistence unit 'default'

成功在两张表都插入了一条记录。

## 第13节使用SQL关系型数据库-h2 嵌入式数据库的使用

### 13.1添加依赖

删除mysql的依赖，新增h2的依赖

|  |
| --- |
| <dependency>  <groupId>com.h2database</groupId>  <artifactId>h2</artifactId>  <scope>runtime</scope>  </dependency> |

### 13.2配置

|  |
| --- |
| #数据库配置  #spring.datasource.url=jdbc:h2:~/test;AUTO\_SERVER=TRUE;DB\_CLOSE\_ON\_EXIT=FALSE  #指定数据库路径  spring.datasource.url=jdbc:h2:file:D:/zte\_h2/roncoo\_spring\_ boot;AUTO\_SERVER=TRUE;DB\_CLOSE\_ON\_EXIT=FALSE  spring.datasource.username=root  spring.datasource.password=1234 |

注：

1."~"这个符号代表的就是当前登录到操作系统的用户对应的用户目录

2.账号密码我们指定之后，就会自动创建

指定路径：

spring.datasource.url=jdbc:h2:file:D:/zte\_h2/roncoo\_spring\_ boot;AUTO\_SERVER=TRUE;DB\_CLOSE\_ON\_EXIT=FALSE

内存模式：

spring.datasource.url=jdbc:h2:mem:test

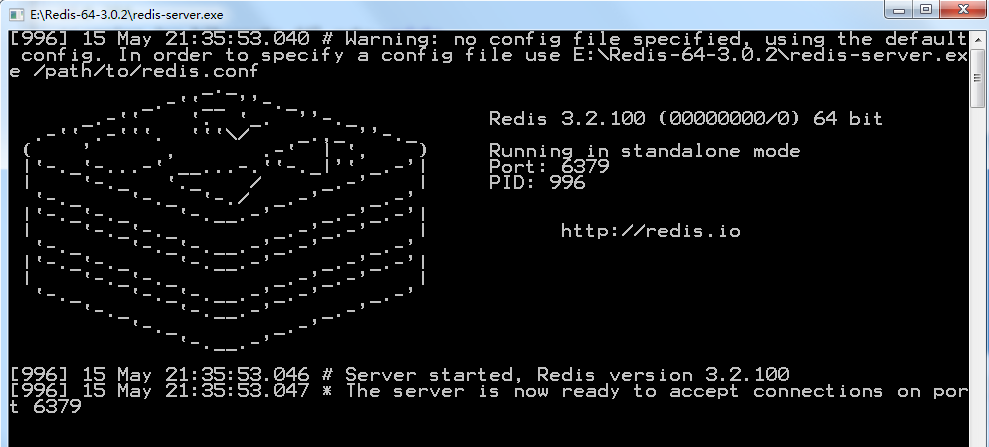
### 13.3进入h2嵌入式数据库控制台

[http://localhost:8082/h2-console/](http://localhost:8082/h2-console/.)

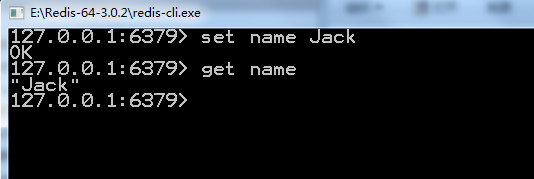
## 第14节使用NoSQL数据库-redis redis

### 14.1windows版本redis软件包及启动方式

启动redis-server：



启动redis-cli：



### 14.2添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency> |

### 14.3配置文件

|  |
| --- |
| #redis  spring.redis.host=localhost  spring.redis.port=6379  #spring.redis.password=123456  #spring.redis.database=0  #spring.redis.pool.max-active=8  #spring.redis.pool.max-idle=8  #spring.redis.pool.max-wait=-1  #spring.redis.pool.min-idle=0  #spring.redis.timeout=0 |

注意：生产环境下，如果外网可以访问，一定要设置密码！

### 14.4SpringBoot添加redis组件

1）RedisComponent

|  |
| --- |
| @Component  **public** **class** RedisComponent {  @Autowired  **private** StringRedisTemplate stringRedisTemplate;  **public** **void** set(String key, String value) {  ValueOperations<String, String> ops = **this**.stringRedisTemplate.opsForValue();  **if** (!**this**.stringRedisTemplate.hasKey(key)) {  ops.set(key, value);  System.***out***.println("set key success");  } **else** {  // 存在则打印之前的 value 值  System.***out***.println("this key = " + ops.get(key));  }  }  **public** String get(String key) {  **return** **this**.stringRedisTemplate.opsForValue().get(key);  }  **public** **void** del(String key) {  **this**.stringRedisTemplate.delete(key);  }  } |

### 14.5单元测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootFreemakerApplicationTests {  @Autowired  **private** RedisComponent redisComponent;  @Test  **public** **void** set() {  redisComponent.set("name", "hello world");  }  @Test  **public** **void** get() {  System.***out***.println(redisComponent.get("name"));  }  @Test  **public** **void** del() {  redisComponent.del("name");  }  } |

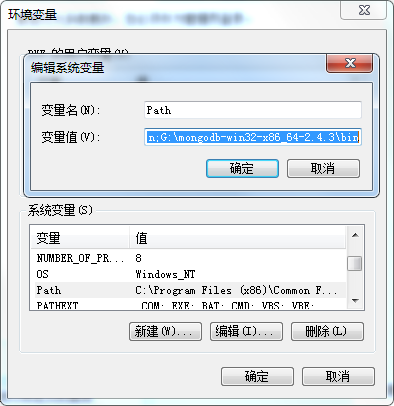
## 第15节使用NoSQL数据库-mongodb

### 15.1windows版本MongoDB软件包及启动方式

1. 新建MongoDB数据库的文件目录



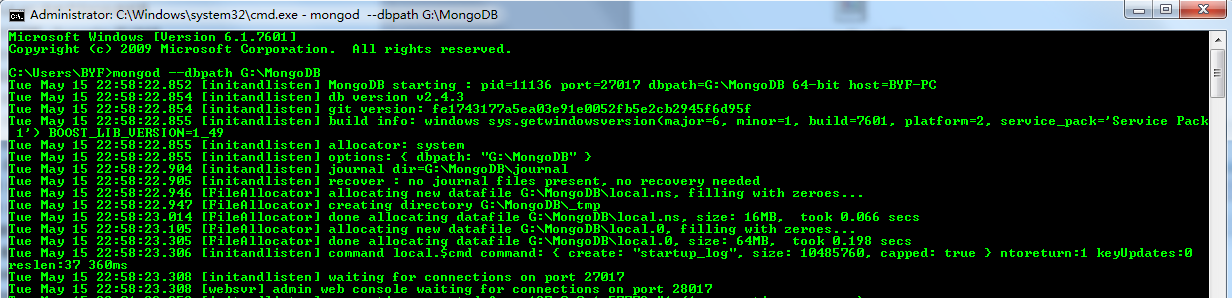
1. 设置环境变量



1. 使用命令行启动MongoDB

C:\Users\BYF>mongod --dbpath G:\MongoDB

注意：要先创建文件夹G:\MongoDB



1. 访问MongoDB的web端口，检测是否成功启动

<http://localhost:28017/>



### 15.2添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-mongodb</artifactId>  </dependency> |

### 15.3 配置文件

|  |
| --- |
| # MONGODB (MongoProperties)  spring.data.mongodb.uri=mongodb://localhost/test  spring.data.mongodb.port=27017  #spring.data.mongodb.authentication-database=  #spring.data.mongodb.database=test  #spring.data.mongodb.field-naming-strategy=  #spring.data.mongodb.grid-fs-database=  #spring.data.mongodb.host=localhost  #spring.data.mongodb.password=  #spring.data.mongodb.repositories.enabled=true  #spring.data.mongodb.username= |

### 15.4SpringBoot添加MongoDB组件

1）MongodbComponent

|  |
| --- |
| @Component  **public** **class** MongodbComponent {  @Autowired  **private** MongoTemplate mongoTemplate;  **public** **void** insert(ZteUser roncooUser) {  mongoTemplate.insert(roncooUser);  }  **public** **void** deleteById(**int** id) {  Criteria criteria = Criteria.*where*("id").in(id);  Query query = **new** Query(criteria);  mongoTemplate.remove(query, ZteUser.**class**);  }  **public** **void** updateById(ZteUser roncooUser) {  Criteria criteria = Criteria.*where*("id").in(roncooUser.getId());  Query query = **new** Query(criteria);  Update update = **new** Update();  update.set("name", roncooUser.getName());  update.set("createTime", roncooUser.getCreateTime());  mongoTemplate.updateMulti(query, update, ZteUser.**class**);  }  **public** ZteUser selectById(**int** id) {  Criteria criteria = Criteria.*where*("id").in(id);  Query query = **new** Query(criteria);  **return** mongoTemplate.findOne(query, ZteUser.**class**);  }  } |

### 15.5单元测试

|  |
| --- |
| @Autowired  **private** MongodbComponent mongodbComponent;  @Test  **public** **void** set() {  ZteUser zteUser = **new** ZteUser();  zteUser.setId(1);  zteUser.setName("杰克");  zteUser.setCreateTime(**new** Date());  mongodbComponent.insert(zteUser);  }  @Test  **public** **void** select() {  System.***out***.println(mongodbComponent.selectById(1));  }  @Test  **public** **void** update() {  ZteUser zteUser = **new** ZteUser();  zteUser.setId(1);  zteUser.setName("测试修改");  zteUser.setCreateTime(**new** Date());  mongodbComponent.updateById(zteUser);  System.***out***.println(mongodbComponent.selectById(1));  }  @Test  **public** **void** delete() {  mongodbComponent.deleteById(1);  } |

### 15.6SpringBoot封装的MongoDBRepository组件

|  |
| --- |
| **public** **interface** ZteUserLogMongoDao **extends** MongoRepository<ZteUserLog, Integer> {  ZteUserLog findByUserName(String string);  ZteUserLog findByUserNameAndUserIp(String string, String ip);  Page<ZteUserLog> findByUserName(String string, Pageable pageable);  } |

### 15.7单元测试

|  |
| --- |
| @Autowired  **private** ZteUserLogMongoDao zteUserLogMongoDao;  @Test  **public** **void** insert() {  ZteUserLog entity = **new** ZteUserLog();  entity.setId(1);  entity.setUserName("杰克");  entity.setUserIp("192.168.0.1");  entity.setCreateTime(**new** Date());  zteUserLogMongoDao.save(entity);  }  @Test  **public** **void** delete() {  zteUserLogMongoDao.delete(1);  }  @Test  **public** **void** update() {  ZteUserLog entity = **new** ZteUserLog();  entity.setId(1);  entity.setUserName("杰克2");  entity.setUserIp("192.168.0.1");  entity.setCreateTime(**new** Date());  zteUserLogMongoDao.save(entity);  }  @Test  **public** **void** select() {  ZteUserLog result = zteUserLogMongoDao.findOne(1);  System.***out***.println(result);  }  // 分页  @Test  **public** **void** queryForPage() {  Pageable pageable = **new** PageRequest(0, 20, **new** Sort(**new** Order(Direction.***DESC***, "id")));  //Page<ZteUserLog> result = zteUserLogMongoDao.findByUserName("杰克2", pageable);  Page<ZteUserLog> result = zteUserLogMongoDao.findAll(pageable);  System.***out***.println(result.getContent());  } |

### 15.8嵌入式MongoDB

1. 添加依赖

|  |
| --- |
| <dependency>  <groupId>de.flapdoodle.embed</groupId>  <artifactId>de.flapdoodle.embed.mongo</artifactId>  </dependency> |

1. 测试，加入嵌入式的 mongo 之后，首次启动会进行下载，时间会比较久，请耐心等待。

2018-05-15 23:37:05.585 |-INFO [main] org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo [54] -| Download 3.2.2:Windows:B64 : starting...

2018-05-15 23:37:07.591 |-INFO [main] org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo [59] -| Download 3.2.2:Windows:B64 : DownloadSize: 148359445

2018-05-15 23:37:07.595 |-INFO [main] org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo [42] -| Download 3.2.2:Windows:B64 : 0 %

注意：下载完成，启动之后，默认情况下数据会在内存里面，重启会丢失

## 第16节使用 Caching-EhCache

Spring boot 支持的缓存：

•Generic

•JCache (JSR-107)

•EhCache 2.x（一般在本地做缓存，也可以做集群）

•Hazelcast

•Infinispan

•Couchbase

•Redis（一般在集群做缓存）

•Caffeine

•Guava

•Simple

最常用的是 EhCache，文档多，资料全

### 16.1添加依赖

|  |
| --- |
| <!-- caching -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-cache</artifactId>  </dependency>  <dependency>  <groupId>net.sf.ehcache</groupId>  <artifactId>ehcache</artifactId>  </dependency> |

### 16.2配置文件

添加缓存配置文件：

/spring-boot-ehcache/src/main/resources/config/ehcache.xml

|  |
| --- |
| <ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"ehcache.xsd"*>  <cache name=*"userCache"* eternal=*"false"* maxEntriesLocalHeap=*"0"*  timeToIdleSeconds=*"200"*></cache>  <!-- eternal：true表示对象永不过期，此时会忽略timeToIdleSeconds和timeToLiveSeconds属性，默认为false -->  <!-- maxEntriesLocalHeap：堆内存中最大缓存对象数，0没有限制 -->  <!-- timeToIdleSeconds： 设定允许对象处于空闲状态的最长时间，以秒为单位。当对象自从最近一次被访问后，如果处于空闲状态的时间超过了  timeToIdleSeconds属性值，这个对象就会过期，EHCache将把它从缓存中清空。 只有当eternal属性为false，该属性才有效。如果该属性值为0，则表示对象可以无限期地处于空闲状态 -->  </ehcache> |

### 16.3启用注解支持

|  |
| --- |
| @EnableCaching  @ServletComponentScan  @SpringBootApplication  **public** **class** SpringBootFreemakerApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

### 16.4代码实现

1）UserLogCache

|  |
| --- |
| **public** **interface** UserLogCache {  ZteUserLog selectById(Integer id);  ZteUserLog updateById(ZteUserLog roncooUserLog);  String deleteById(Integer id);  } |

2）UserLogCacheImpl

|  |
| --- |
| @CacheConfig(cacheNames="userCache")  @Repository  **public** **class** UserLogCacheImpl **implements** UserLogCache {  @Autowired  **private** ZteUserLogDao zteUserLogDao;    @Autowired  **private** ZteUserLogMongoDao zteUserLogMongoDao;  @Cacheable(key="#p0")  @Override  **public** ZteUserLog selectById(Integer id) {  System.***out***.println("查询功能，缓存找不到，直接读库, id=" + id);  **return** zteUserLogMongoDao.findOne(id);  }  @CachePut(key="#p0.id")  @Override  **public** ZteUserLog updateById(ZteUserLog zteUserLog) {  System.***out***.println("更新功能，更新缓存，直接写库, id=" + zteUserLog);  **return** zteUserLogMongoDao.save(zteUserLog);  }  @CacheEvict(key="#p0")  @Override  **public** String deleteById(Integer id) {  System.***out***.println("删除功能，删除缓存，直接写库, id=" + id);  **return** "清空缓存成功";  }  } |

注解说明：

@CacheConfig：缓存配置，可标志一个缓存的名称

@Cacheable：应用到读取数据的方法上，即可缓存的方法，如查找方法：先从缓存中读取，如果没有再调用方法获取数据，然后把数据添加到缓存中。**适用于查找**

@CachePut：主要针对方法配置，能够根据方法的请求参数对其结果进行缓存，和 @Cacheable 不同的是，它每次都会触发真实方法的调用。**适用于更新和插入**

@CacheEvict：主要针对方法配置，能够根据一定的条件对缓存进行清空。**适用于删除**

### 16.5ApiController测试

|  |
| --- |
| @RestController  @RequestMapping("/api")  **public** **class** ApiController {  @Autowired  **private** UserLogCache userLogCache;    @RequestMapping(value = "/select", method = RequestMethod.***GET***)  **public** ZteUserLog get(@RequestParam(defaultValue = "1") Integer id) {  **return** userLogCache.selectById(id);  }  @RequestMapping(value = "/update", method = RequestMethod.***GET***)  **public** ZteUserLog update(@RequestParam(defaultValue = "1") Integer id) {  ZteUserLog bean = userLogCache.selectById(id);  bean.setUserName("测试");  bean.setCreateTime(**new** Date());  userLogCache.updateById(bean);  **return** bean;  }  @RequestMapping(value = "/del", method = RequestMethod.***GET***)  **public** String del(@RequestParam(defaultValue = "1") Integer id) {  **return** userLogCache.deleteById(id);  }  } |

## 第17节使用Caching-Redis

### 17.1添加依赖

|  |
| --- |
| <!-- redis -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-redis</artifactId>  </dependency> |

### 17.2 配置文件

|  |
| --- |
| # 默认按照常用缓存管理器的优先级进行查找，最先找到的缓存包，则使用；如果配置，则指定。  spring.cache.type=redis |

### 17.3缓存使用优先级问题

1.默认按照 spring boot 的加载顺序来实现

2.配置文件优先于默认

### 17.4自定义缓存管理器

1）RedisCacheConfiguration

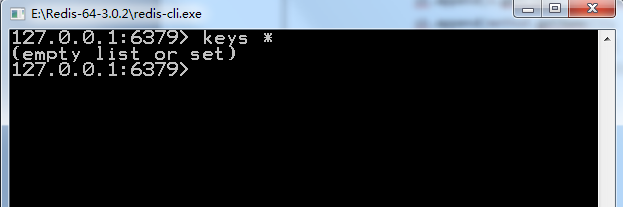
|  |
| --- |
| @Configuration  **public** **class** RedisCacheConfiguration **extends** CachingConfigurerSupport {  // 自定义缓存管理器  @Bean  **public** CacheManager cacheManager(RedisTemplate<?, ?> redisTemplate) {  RedisCacheManager cacheManager = **new** RedisCacheManager(redisTemplate); // 设置默认的过期时间  cacheManager.setDefaultExpiration(20);  Map<String, Long> expires = **new** HashMap<String, Long>();  // 单独设置  expires.put("userCache", 200L);  cacheManager.setExpires(expires);  **return** cacheManager;  }  /\*\*  \* 自定义 key. 此方法将会根据类名+方法名+所有参数的值生成唯一的一个 key,即使@Cacheable 中 的value 属性一样，key  \* 也会不一样。  \*/  @Override  **public** KeyGenerator keyGenerator() {  **return** **new** KeyGenerator() {  @Override  **public** Object generate(Object o, Method method, Object... objects) {  StringBuilder sb = **new** StringBuilder();  sb.append(o.getClass().getName());  sb.append(method.getName());  **for** (Object obj : objects) {  sb.append(obj.toString());  }  **return** sb.toString();  }  };  }  } |

2）StringRedisSerializer

|  |
| --- |
| /\*\*  \* 必须重写序列化器，否则@Cacheable注解的key会报类型转换错误  \*/  **public** **class** StringRedisSerializer **implements** RedisSerializer<Object> {  **private** **final** Charset charset;  **private** **final** String target = "\"";  **private** **final** String replacement = "";  **public** StringRedisSerializer() {  **this**(Charset.*forName*("UTF8"));  }  **public** StringRedisSerializer(Charset charset) {  Assert.*notNull*(charset, "Charset must not be null!");  **this**.charset = charset;  }  @Override  **public** String deserialize(**byte**[] bytes) {  **return** (bytes == **null** ? **null** : **new** String(bytes, charset));  }  @Override  **public** **byte**[] serialize(Object object) {  String string = JSON.*toJSONString*(object);  **if** (string == **null**) {  **return** **null**;  }  string = string.replace(target, replacement);  **return** string.getBytes(charset);  }  } |

### 17.5测试查询、更新、删除缓存

1）首先查看redis数据库，没有任何keys：



测试：<http://localhost:8082/api/select>

do filter

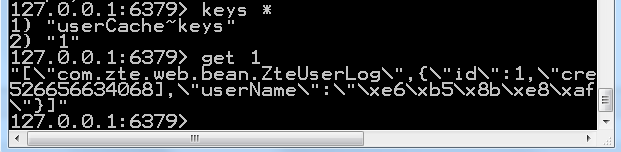
查询功能，缓存找不到，直接读库, id=1

2018-05-18 23:24:54.748 |-DEBUG [http-nio-8082-exec-1] org.springframework.data.mongodb.core.MongoTemplate [1708] -| findOne using query: { "id" : 1} fields: null for class: class com.zte.web.bean.ZteUserLog in collection: zteUserLog

2018-05-18 23:24:54.748 |-DEBUG [http-nio-8082-exec-1] org.springframework.data.mongodb.core.MongoTemplate [2140] -| findOne using query: { "\_id" : 1} in db.collection: test.zteUserLog

2018-05-18 23:24:54.753 |-INFO [http-nio-8082-exec-1] org.mongodb.driver.connection [71] -| Opened connection [connectionId{localValue:26, serverValue:119}] to localhost:27017

查看redis的keys，发现新增一条记录



再次访问<http://localhost:8082/api/select>



没有读数据库操作，控制台只有过滤器消息

do filter

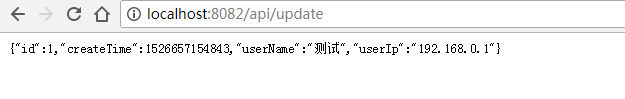
1. 测试更新：<http://localhost:8082/api/update>

do filter

更新功能，更新缓存，直接写库, id=ZteUserLog [id=1, createTime=Fri May 18 23:25:54 CST 2018, userName=测试, userIp=192.168.0.1]

2018-05-18 23:25:54.844 |-DEBUG [http-nio-8082-exec-3] org.springframework.data.mongodb.core.MongoTemplate [1077] -| Saving DBObject containing fields: [\_class, \_id, createTime, userName, userIp]

查看redis缓存的keys，还是两条记录，但key对应的值已经发生变化

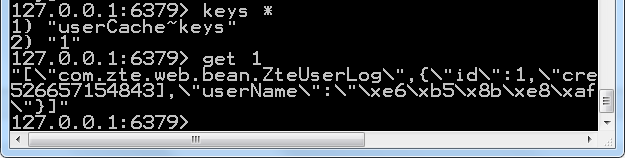


再次访问<http://localhost:8082/api/select>

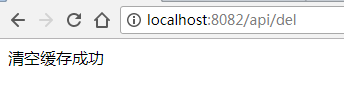
do filter【表示仅从缓存中读取】

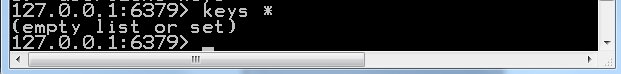


查看redis缓存中的记录，已经是更新过的createtime



1. 超期（200s）缓存全部清空或者直接删除





### 17.6异常处理

1）5xx-系统错误

Cannot get Jedis connection; nested exception is redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool

可能Redis-server没有启动，或者配置文件密码设置的不对

2）5xx-系统错误

Cannot serialize; nested exception is org.springframework.core.serializer.support.SerializationFailedException: Failed to serialize object using DefaultSerializer; nested exception is java.lang.IllegalArgumentException: DefaultSerializer requires a Serializable payload but received an object of type [com.zte.web.bean.ZteUserLog]

Bean对象使用redis做持久化需要进行序列化，实现**implements** Serializable接口，添加：

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

1. java.lang.ClassCastException: **XXX** cannot be cast to java.lang.String

需要重写RedisSerializer的序列化方法

1. [存入redis的key不是自己设置的指定内容@CachePut(key="#p0.id")](mailto:存入redis的key不是自己设置的指定内容@CachePut(key=\"#p0.id\"))

**启动注解**需要指定json对象的 id为key进行更新，否则会新增key，而不是更新select存入数据库的id ，再次select查询发现缓存还是原来的key对应的值。

## 第18节使用异步消息服务-JMS（ActiveMQ）

### 18.1添加依赖

|  |
| --- |
| <!-- jms -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency> |

### 18.2配置文件

|  |
| --- |
| # ACTIVEMQ (ActiveMQProperties)  spring.activemq.in-memory=true  #spring.activemq.broker-url=  #spring.activemq.password=  #spring.activemq.user=  #spring.activemq.packages.trust-all=false  #spring.activemq.packages.trusted=  #spring.activemq.pool.configuration.\*=  #spring.activemq.pool.enabled=false  #spring.activemq.pool.expiry-timeout=0  #spring.activemq.pool.idle-timeout=30000  #spring.activemq.pool.max-connections=1 |

### 18.3启动注解

|  |
| --- |
| @EnableJms  @EnableCaching  @ServletComponentScan  @SpringBootApplication  **public** **class** SpringBootFreemakerApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

### 18.4代码实现

1）配置队列JmsConfiguration

|  |
| --- |
| /\*\*  \* Jms队列配置  \* **@author** BYF  \*  \*/  @Configuration  **public** **class** JmsConfiguration {  @Bean  **public** Queue queue() {  **return** **new** ActiveMQQueue("zte.queue");  }  } |

2）生产消费队列

|  |
| --- |
| @Component  **public** **class** JmsComponent {  @Autowired  **private** JmsMessagingTemplate jmsMessagingTemplate;  @Autowired  **private** Queue queue;  **public** **void** send(String msg) {  **this**.jmsMessagingTemplate.convertAndSend(**this**.queue, msg);  }  @JmsListener(destination = "zte.queue")  **public** **void** receiveQueue(String text) {  System.***out***.println("接受到：" + text);  }  } |

### 18.5单元测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootFreemakerApplicationTests {  @Autowired  **private** JmsComponent jmsComponent;  @Test  **public** **void** send() {  **for** (**int** i = 0; i < 10; i++)  jmsComponent.send("hello world : " + i);  }  } |

### 18.6测试结果

接受到：hello world : 0

接受到：hello world : 1

接受到：hello world : 2

接受到：hello world : 3

接受到：hello world : 4

接受到：hello world : 5

接受到：hello world : 6

接受到：hello world : 7

接受到：hello world : 8

接受到：hello world : 9

## 第19节使用异步消息服务-AMQP（RabbitMQ）

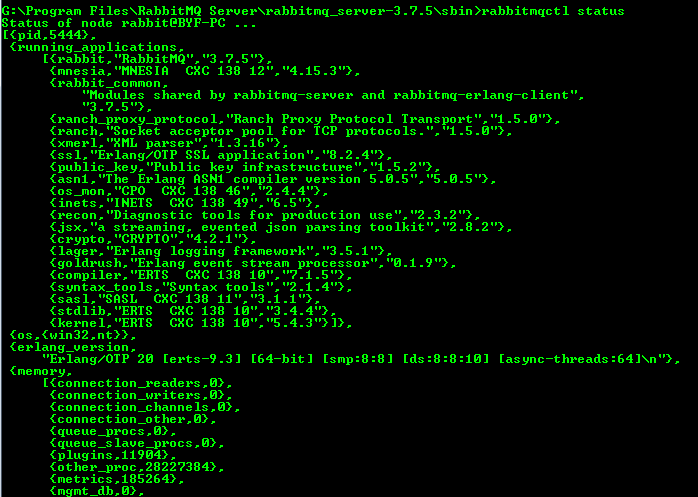
RabbitMQ下载地址：<http://www.rabbitmq.com/download.html>

erlang 下载地址：<http://www.erlang.org/downloads>

### 19.1RabbitMQ安装及启动

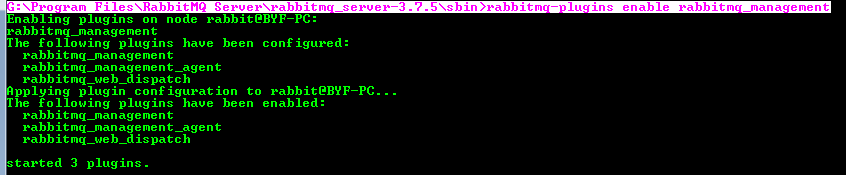
1）检测安装结果：

G:\Program Files\RabbitMQ Server\rabbitmq\_server-3.7.5\sbin>rabbitmqctl status

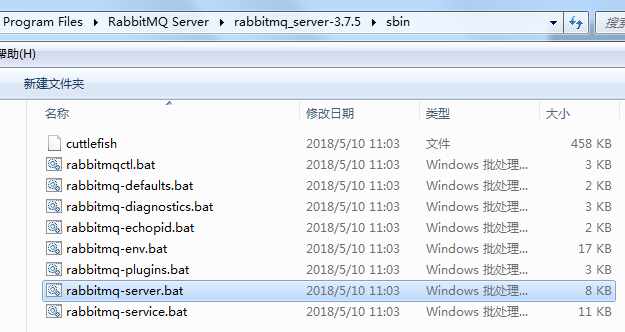


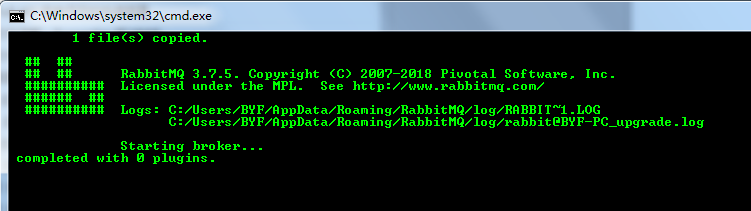
2）安装插件：

G:\Program Files\RabbitMQ Server\rabbitmq\_server-3.7.5\sbin>rabbitmqctl status

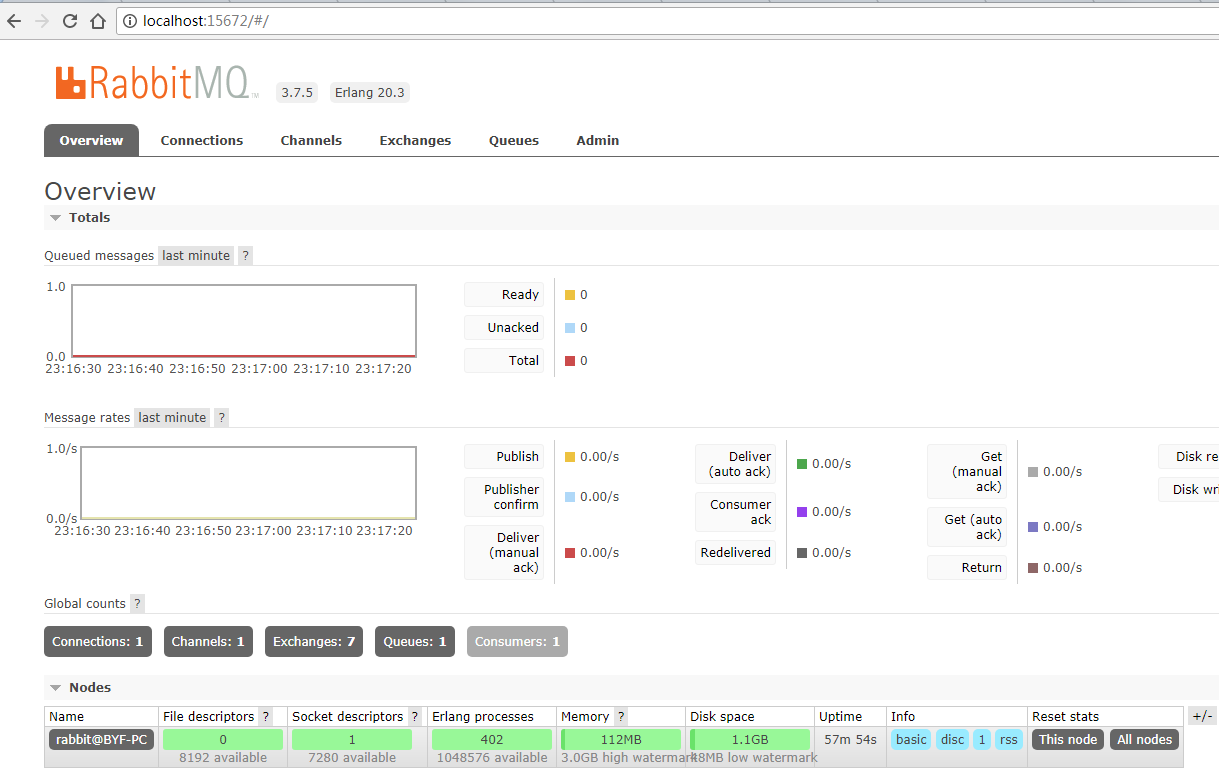


3）启动RabbitMQ服务：





4）登录RabbitMQ客户端，用户名guest，密码guest



### 19.2添加依赖

|  |
| --- |
| <!-- amqp -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-amqp</artifactId>  </dependency> |

### 19.3配置文件

|  |
| --- |
| # RABBIT (RabbitProperties)  #spring.rabbitmq.host=localhost  #spring.rabbitmq.port=5672  #spring.rabbitmq.password=  #spring.rabbitmq.username= |

### 19.4代码实现

1）启动@EnableRabbit注解

|  |
| --- |
| @EnableRabbit  @EnableCaching  @ServletComponentScan  @SpringBootApplication  **public** **class** SpringBootFreemakerApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootFreemakerApplication.**class**, args);  }  } |

2）amqp 队列配置AmqpConfiguration

|  |
| --- |
| **import** org.springframework.amqp.core.Queue;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  **public** **class** AmqpConfiguration {  @Bean  **public** Queue queue() {  **return** **new** Queue("zte.queue");  }  } |

3）AmqpComponent

|  |
| --- |
| **import** org.springframework.amqp.core.AmqpTemplate;  **import** org.springframework.amqp.rabbit.annotation.RabbitListener;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Component;  @Component  **public** **class** AmqpComponent {  @Autowired  **private** AmqpTemplate amqpTemplate;  **public** **void** send(String msg) {  **this**.amqpTemplate.convertAndSend("zte.queue", msg);  }  @RabbitListener(queues = "zte.queue")  **public** **void** receiveQueue(String text) {  System.***out***.println("接受到：" + text);  }  } |

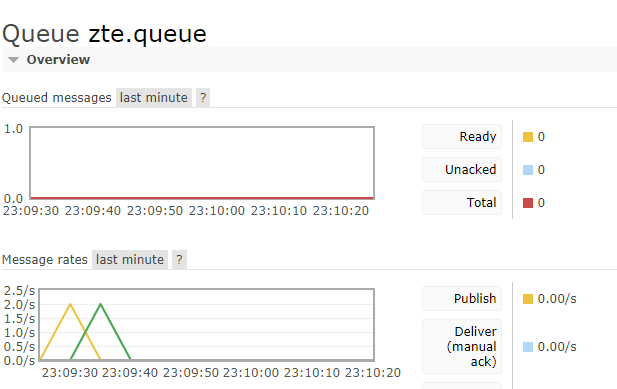
### 19.5单元测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootFreemakerApplicationTests {  @Autowired  **private** AmqpComponent amqpComponent;  @Test  **public** **void** send() {  **for**(**int** i=0;i<10;i++)  amqpComponent.send("AMQP : hello world--" +i );  }  } |

测试结果：

打开RabbitMQ Web客户端：

<http://localhost:15672/#/queues/%2F/zte.queue>



Console输出：

接受到：AMQP : hello world--0

接受到：AMQP : hello world--1

接受到：AMQP : hello world--2

接受到：AMQP : hello world--3

接受到：AMQP : hello world--4

接受到：AMQP : hello world--5

接受到：AMQP : hello world--6

接受到：AMQP : hello world--7

接受到：AMQP : hello world--8

接受到：AMQP : hello world--9

## 第20节调用REST服务-如何使用代理

### 20.1添加依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  </dependency> |

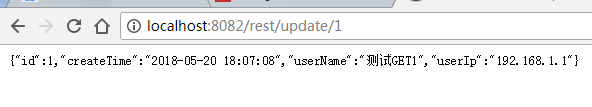
### 20.2构建待访问的restful接口

1）RestfulController

|  |
| --- |
| @RestController  @RequestMapping(value = "/rest", method = RequestMethod.***POST***)  **public** **class** RestfulController {  @Autowired  **private** UserLogCache userLogCache;  @RequestMapping(value = "/update/{id}", method = RequestMethod.***GET***)  **public** ZteUserLog update(@PathVariable(value = "id") **int** id) {  ZteUserLog bean = userLogCache.selectById(id);  **if**(bean == **null**){  bean = **new** ZteUserLog();  }  System.***out***.println("【URL】-->http:localhost:8082/rest/update/" + bean.getId());  bean.setUserName("测试GET" + id);  bean.setCreateTime(**new** Date());  bean.setUserIp("192.168.1.1");  userLogCache.updateById(bean);  **return** bean;  }    @RequestMapping(value = "/update")  **public** ZteUserLog update2(@RequestBody JsonNode jsonNode) {  System.***out***.println("【URL】-->http:localhost:8082/rest/update/" + "\njsonNode=" + jsonNode);  ZteUserLog bean = userLogCache.selectById(jsonNode.get("id").asInt(1));  **if**(bean == **null**){  bean = **new** ZteUserLog();  }  bean.setUserName("测试POST" + bean.getId());  bean.setCreateTime(**new** Date());  bean.setUserIp("192.168.1.1");  userLogCache.updateById(bean);  **return** bean;  }  } |

启动SpringBoot应用，对外提供rest服务。

访问：http://localhost:8082/rest/update/1测试GET请求



2）单元测试

|  |
| --- |
| @Autowired  **private** RestTemplateBuilder restTemplateBuilder;  @Test  **public** **void** getForObject() {  ZteUserLog bean = restTemplateBuilder.build().getForObject("http://localhost:8082/rest/update/{id}", ZteUserLog.**class**, 1);  System.***out***.println(bean);  Map<String,Object> map = **new** HashMap<String,Object>();  map.put("id", 2);  bean = restTemplateBuilder.build().postForObject("http://localhost:8082/rest/update", map, ZteUserLog.**class**);  System.***out***.println(bean);  }    @Test  **public** **void** getForObject1() {  String result = restTemplateBuilder.additionalCustomizers(**new** ProxyCustomizer()).build().getForObject("http://www.zte.com.cn", String.**class**);  System.***out***.println(result);  } |

### 20.3HttpClient代理实现

1）

|  |
| --- |
| **static** **class** ProxyCustomizer **implements** RestTemplateCustomizer {  @Override  **public** **void** customize(RestTemplate restTemplate) {  String proxyHost = "121.237.143.247";  **int** proxyPort = 18118;    HttpHost proxy = **new** HttpHost(proxyHost, proxyPort);  HttpClient httpClient = HttpClientBuilder.*create*().setRoutePlanner(**new** DefaultProxyRoutePlanner(proxy) {  @Override  **public** HttpHost determineProxy(HttpHost target, HttpRequest request, HttpContext context) **throws** HttpException {  System.***out***.println(target.getHostName());  **return** **super**.determineProxy(target, request, context);  }  }).build();  HttpComponentsClientHttpRequestFactory httpComponentsClientHttpRequestFactory = **new** HttpComponentsClientHttpRequestFactory(httpClient);  httpComponentsClientHttpRequestFactory.setConnectTimeout(10000);  httpComponentsClientHttpRequestFactory.setReadTimeout(60000);  restTemplate.setRequestFactory(httpComponentsClientHttpRequestFactory);  }  } |

2）单元测试

|  |
| --- |
| @Test  **public** **void** getForObject1() {  String result = restTemplateBuilder.additionalCustomizers(**new** ProxyCustomizer()).build().getForObject("http://www.zte.com.cn", String.**class**);  System.***out***.println(result);  } |

### 20.4在线代理

<http://ip.zdaye.com/>

## 第21节发送邮件-使用模板邮件并实现多账号轮询发送

### 21.1添加依赖

|  |
| --- |
| <!-- mail -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-mail</artifactId>  </dependency> |

### 21.2添加配置

|  |
| --- |
| # mail  spring.mail.host: smtp.163.com  #多账号轮询发送邮箱密码对应，都以逗号隔开  spring.mail.username:baiyifan@163.com  #需要开通客户端授权密码，否则会报javax.mail.AuthenticationFailedException: 535 Error: authentication failed  spring.mail.password:  spring.mail.properties.mail.smtp.auth: true |

### 21.3代码实现

1）邮箱发送配置ZteJavaMailSenderImpl

|  |
| --- |
| @Configuration  @EnableConfigurationProperties(MailProperties.**class**)  **public** **class** ZteJavaMailSenderImpl **extends** JavaMailSenderImpl **implements** JavaMailSender {  **private** ArrayList<String> usernameList;  **private** ArrayList<String> passwordList;  **private** **int** currentMailId = 0;  **private** **final** MailProperties properties;  **public** ZteJavaMailSenderImpl(MailProperties properties) {  **this**.properties = properties;  // 初始化账号  **if** (usernameList == **null**)  usernameList = **new** ArrayList<String>();  String[] userNames = **this**.properties.getUsername().split(",");  **if** (userNames != **null**) {  **for** (String user : userNames) {  usernameList.add(user);  }  }  // 初始化密码  **if** (passwordList == **null**)  passwordList = **new** ArrayList<String>();  String[] passwords = **this**.properties.getPassword().split(",");  **if** (passwords != **null**) {  **for** (String pw : passwords) {  passwordList.add(pw);  }  }  }  @Override  **protected** **void** doSend(MimeMessage[] mimeMessage, Object[] object) **throws** MailException {  **super**.setUsername(usernameList.get(currentMailId));  **super**.setPassword(passwordList.get(currentMailId));  // 设置编码和各种参数  **super**.setHost(**this**.properties.getHost());  **super**.setDefaultEncoding(**this**.properties.getDefaultEncoding().name());  **super**.setJavaMailProperties(asProperties(**this**.properties.getProperties()));  **super**.doSend(mimeMessage, object);  // 轮询  currentMailId = (currentMailId + 1) % usernameList.size();  }  **private** Properties asProperties(Map<String, String> source) {  Properties properties = **new** Properties();  properties.putAll(source);  **return** properties;  }  @Override  **public** String getUsername() {  **return** usernameList.get(currentMailId);  }  } |

2）多账号轮询发送组件JavaMailComponent

|  |
| --- |
| @Component  **public** **class** JavaMailComponent {  **private** **static** **final** String ***template*** = "mail/mail.ftl";  @Autowired  **private** FreeMarkerConfigurer freeMarkerConfigurer;  @Autowired  **private** ZteJavaMailSenderImpl javaMailSender;  **public** **void** sendMail(String email) {  Map<String, Object> map = **new** HashMap<String, Object>();  map.put("email", email);  **try** {  String text = getTextByTemplate(***template***, map);  send(email, text);  } **catch** (IOException | TemplateException e) {  e.printStackTrace();  } **catch** (MessagingException e) {  e.printStackTrace();  }  }  **private** String getTextByTemplate(String template, Map<String, Object> model) **throws** TemplateNotFoundException, MalformedTemplateNameException, ParseException, IOException, TemplateException {  **return** FreeMarkerTemplateUtils.*processTemplateIntoString*(freeMarkerConfigurer.getConfiguration().getTemplate(template), model);  }  **private** String send(String email, String text) **throws** MessagingException, UnsupportedEncodingException {  MimeMessage message = javaMailSender.createMimeMessage();  MimeMessageHelper helper = **new** MimeMessageHelper(message, **true**, "UTF-8");  InternetAddress from = **new** InternetAddress();  from.setAddress(javaMailSender.getUsername());  from.setPersonal("瓦坎达", "UTF-8");  helper.setFrom(from);  helper.setTo(email);  helper.setSubject("测试邮件");  helper.setText(text, **true**);  javaMailSender.send(message);  **return** text;  }  } |

1. 邮件发送信息模板

/spring-boot-mail/src/main/resources/templates/mail/mail.ftl

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  </head>  <body>  <div style="width: 600px; text-align: left; margin: 0 auto;">  <h1 style="color: #005da7;">瓦坎达</h1>  <div style="border-bottom: 5px solid #005da7; height: 2px; width: 100%;"></div>  <div style="border: 1px solid #005da7; font-size: 16px; line-height: 50px; padding: 20px;">  <div>${email}，您好！</div>  <div>  这是个测试  </div>    <div style="border-bottom: 2px solid #005da7; height: 2px; width: 100%;"></div>    <div>  <img src="http://res.www.zte.com.cn/mediares/zte/Global/logo/qr\_weixin.jpg?h=200&la=zh-CN&w=200" alt="二维码" />  </div>  <div>  想了解更多信息，请访问 <a href="http://www.zet.com.cn">http://www.zet.com.cn</a>  </div>  </div>  </div>  </body>  </html> |

1. 发送邮件页面/spring-boot-mail/src/main/resources/templates/index.ftl

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <title>Spring Boot Demo - FreeMarker</title>  <link href="/css/index.css" rel="stylesheet" />  </head>  <body>  <center>  <img src="/images/logo.png" />  <h1 id="title">${title}</h1>  </center>    <form method="POST" enctype="multipart/form-data" action="/file/upload">  文件：<input type="file" name="zteFile" /> <input type="submit" value="上传" />  </form>    <input type="text" name="email" id="email" />  <button id="send">发送邮件</button>    <script type="text/javascript" src="/webjars/jquery/2.1.4/jquery.min.js"></script>    <script>  $(function(){  $('#title').click(function(){  alert("点击了");  });  })    $(function(){  $('#send').click(function(){  var email = $('#email').val();  $.ajax({  url:'/api/mail',  type:'post',  data:{'email':email},  success:function(msg){  alert(msg);  }  });  });  })    </script>  </body>  </html> |

1. Controller接口JavaMailComponent

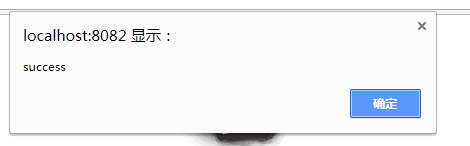
|  |
| --- |
| @Autowired  **private** JavaMailComponent component;  @RequestMapping(value = "mail")  **public** String mail(String email) {  component.sendMail(email);  **return** "success";  } |

### 21.4测试

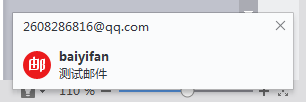
访问发邮件的页面：<http://localhost:8082/web/index>



单击发送，相应success表示发送成功



收到一封测试邮件



查看邮件内容



## 第22节使用Spring Session实现集群-redis

### 22.1session集群的解决方案

1）扩展指定server

利用Servlet容器提供的插件功能，自定义HttpSession的创建和管理策略，并通过配置的方式替换掉默认的策略。缺点：耦合Tomcat/Jetty等Servlet容器，不能随意更换容器。

2）利用Filter

利用HttpServletRequestWrapper，实现自己的 getSession()方法，接管创建和管理Session数据的工作。spring-session就是通过这样的思路实现的。

3）Spring Boot中spring session支持方式：

JDBC、MongoDB、Redis、Hazelcast、HashMap

### 22.2添加依赖

|  |
| --- |
| <!-- spring session -->  <dependency>  <groupId>org.springframework.session</groupId>  <artifactId>spring-session</artifactId>  </dependency>  <!-- redis -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-redis</artifactId>  </dependency> |

### 22.3添加配置

|  |
| --- |
| # spring session使用存储类型  spring.session.store-type=redis  # spring session刷新模式：默认on-save  #spring.session.redis.flush-mode=on-save  #spring.session.redis.namespace=  # session超时时间，单位秒  #server.session.timeout=30 |

### 22.4测试

1）在第一个SpringBoot应用8082端口中，添加web接口

|  |
| --- |
| @RequestMapping(value = "/index")  **public** String index(ModelMap map, HttpSession httpSession) {  map.put("title", "第一个应用：sessionID=" + httpSession.getId());  logger.info("sessionID=" + httpSession.getId());  **return** "index";  } |

2）复制第一个SpringBoot应用运行在8081端口，添加web接口

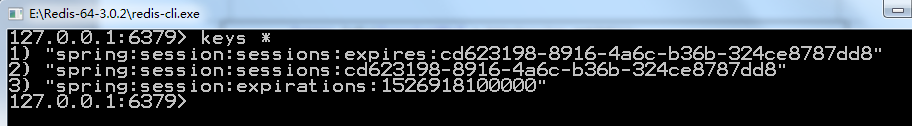
|  |
| --- |
| @RequestMapping(value = "/index")  **public** String index(ModelMap map, HttpSession httpSession) {  map.put("title", "第二个应用：sessionID=" + httpSession.getId());  logger.info("sessionID=" + httpSession.getId());  **return** "index";  } |

两个应用需要启动redis作为存储sessionID的key值，访问两个api。





两个sessionID一致，查看redis中的key没有增加：



## 第23节如何进行远程调试

### 23.1远程调试的概念

什么是远程调试：本地调用非本地的环境进行调试。

原理：两个VM之间通过socket协议进行通信，然后以达到远程调试的目的。

注意，如果 Java 源代码与目标应用程序不匹配，调试特性将不能正常工作。

### 23.2java启动命令

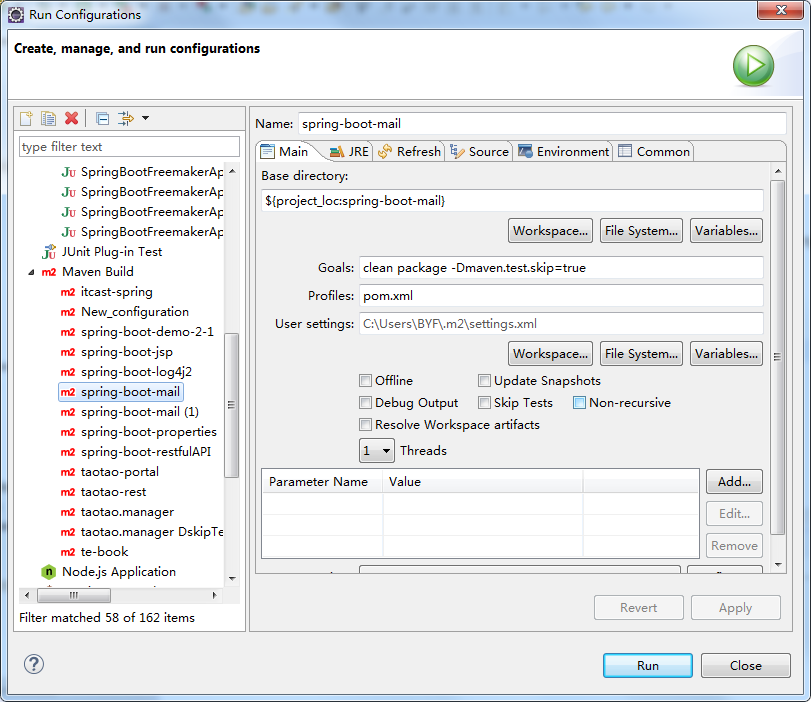
-Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n

比如：java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n -jar restfulAPI-0.0.1-SNAPSHOT.jar

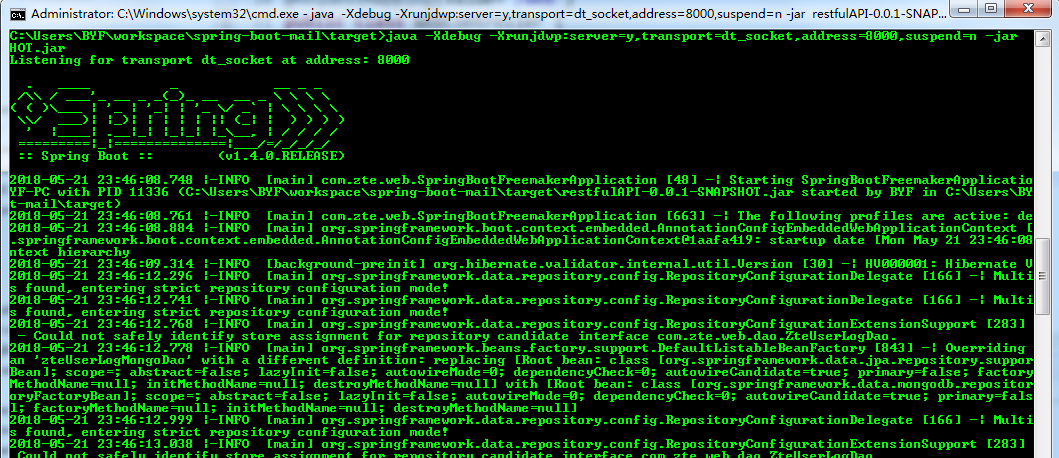
打包跳过测试：-Dmaven.test.skip=true

### 23.3演示

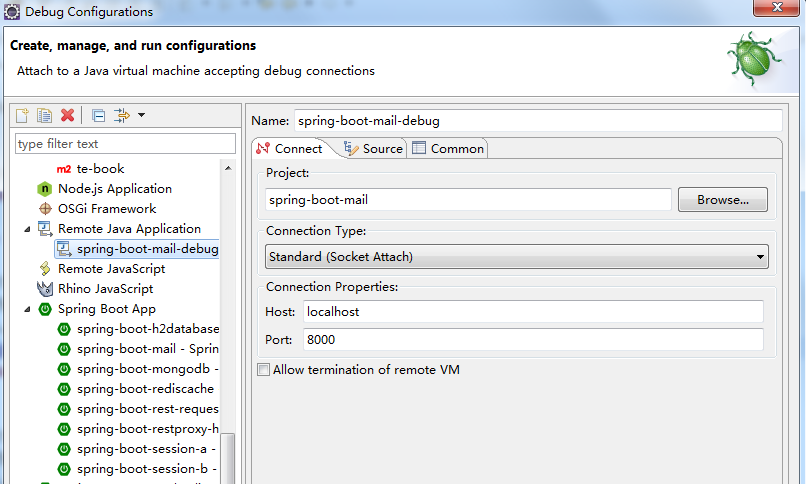
1. 打包



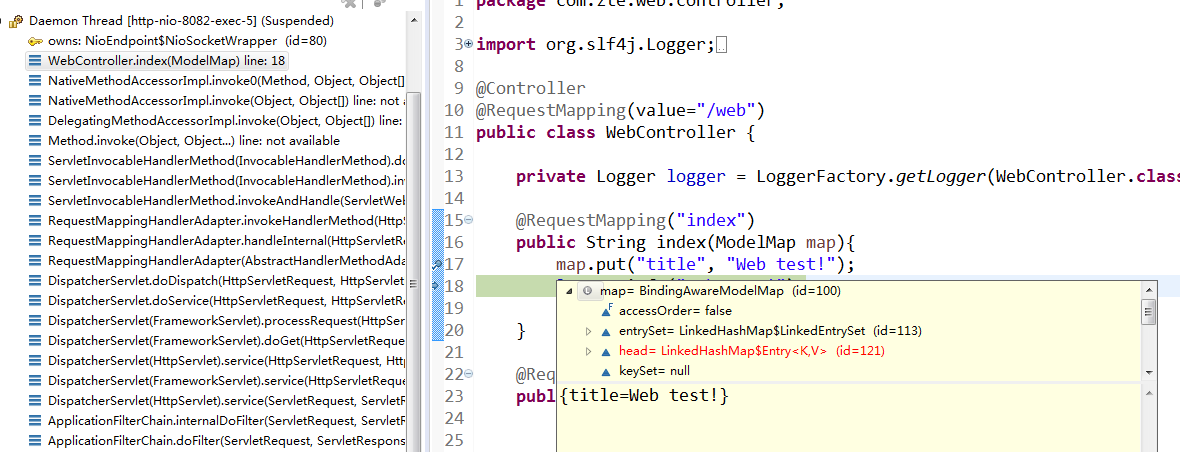
2）启动远程应用



3）打开本地调试



1. 在/web/api接口打断点，访问页面：<http://localhost:8082/web/index>



## 第24节生产准备-基于HTTP的监控

### 24.1利用Spring Boot的特性进行监控你的应用

1.通过HTTP（最简单方便）

2.通过JMX

3.通过远程shell

### 24.2添加依赖

1）http监控依赖

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

2）security鉴权依赖

|  |
| --- |
| <!-- security -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency> |

### 24.3端点（通过执行器端点可以监控应用及与应用进行交互）

1.端点暴露的方式取决于你采用的监控方式。如果使用HTTP监控，端点的ID映射到一个URL。例如，默认情况下，health端点将被映射到/health。

2.端点会默认有敏感度，根据不同的敏感度是否需要提供用户密码认证

3.如果没启用web安全，则敏感度高的会禁用

4.可以通过配置文件进行配置敏感度

5.默认情况下，除了shutdown外的所有端点都是启用的。

### 24.4配置

1）启用端点监控配置

|  |
| --- |
| #端点的配置  endpoints.sensitive=true  endpoints.shutdown.enabled=true |

2）鉴权配置

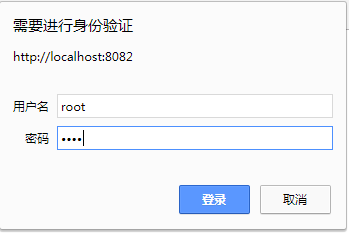
|  |
| --- |
| #保护端点  security.basic.enabled=true  security.user.name=root  security.user.password=root  management.security.roles=SUPERUSER |

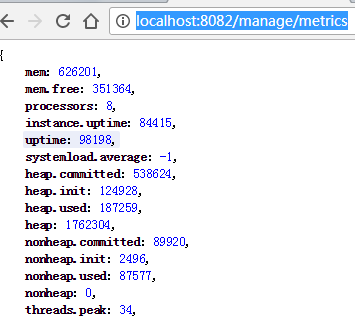
3）自定义端点映射路径配置

|  |
| --- |
| #自定义路径  security.basic.path=/manage  management.context-path=/manage |

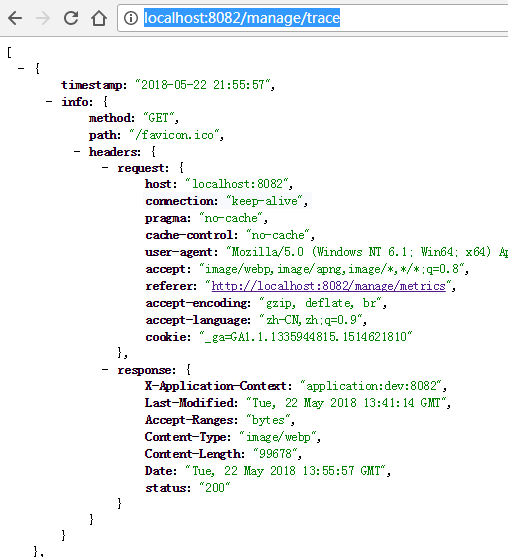
### 24.5测试

1. 访问：<http://localhost:8082/manage/metrics>

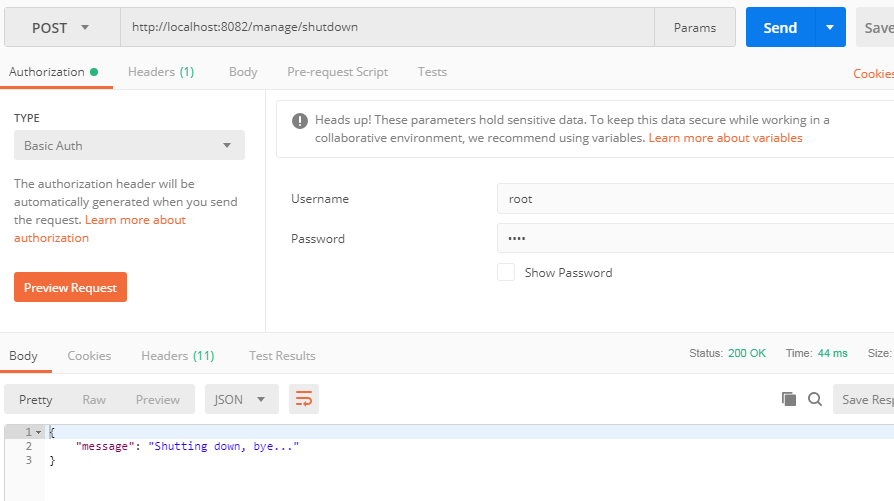




1. 访问：<http://localhost:8082/manage/trace>



1. 远程关闭SpringBoot应用



## 第25节Spring Boot 集成 mybatis

### 25.1添加依赖

|  |
| --- |
| <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.2</version>  </dependency> |

版本说明：最新 mybatis-spring-boot-starter 的版本为 1.2.0-SNAPSHOT，依赖的是 spring boot 的 1.4.1，但是还不是 released 版本。

GitHub: <https://github.com/mybatis/spring-boot-starter>

### 25.2基于mybatis注解的集成

1）配置mysql驱动

|  |
| --- |
| #mysql  spring.datasource.url=jdbc:mysql://localhost/spring\_boot\_demo?useUnicode=true&characterEncoding=utf-8  spring.datasource.username=root  spring.datasource.password=1234  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

2）定义bean对象ZteUser

|  |
| --- |
| **public** **class** ZteUser **implements** Serializable {  **private** Integer id;  **private** String name;  **private** Date createTime;  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name == **null** ? **null** : name.trim();  }  **public** Date getCreateTime() {  **return** createTime;  }  **public** **void** setCreateTime(Date createTime) {  **this**.createTime = createTime;  }  @Override  **public** String toString() {  StringBuilder sb = **new** StringBuilder();  sb.append(getClass().getSimpleName());  sb.append(" [");  sb.append("Hash = ").append(hashCode());  sb.append(", id=").append(id);  sb.append(", name=").append(name);  sb.append(", createTime=").append(createTime);  sb.append(", serialVersionUID=").append(***serialVersionUID***);  sb.append("]");  **return** sb.toString();  }  } |

3）定义Mapper接口

|  |
| --- |
| @Mapper  **public** **interface** ZteUserMapper {  @Insert(value = "insert into zte\_user (name, create\_time) values (#{name,jdbcType=VARCHAR}, #{createTime,jdbcType=TIMESTAMP})")  **int** insert(ZteUser record);  @Select(value = "select id, name, create\_time from zte\_user where id = #{id,jdbcType=INTEGER}")  @Results(value = { @Result(column = "create\_time", property = "createTime", jdbcType = JdbcType.***TIMESTAMP***) })  ZteUser selectByPrimaryKey(Integer id);  } |

4）测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** SpringBootMybatisApplicationTests {  @Autowired  **private** ZteUserMapper mapper;  @Test  **public** **void** insert() {  ZteUser roncooUser = **new** ZteUser();  roncooUser.setName("测试");  roncooUser.setCreateTime(**new** Date());  **int** result = mapper.insert(roncooUser);  System.***out***.println(result);  }  @Test  **public** **void** select() {  ZteUser result = mapper.selectByPrimaryKey(2);  System.***out***.println(result);  }  } |

### 25.3基于mybatis xml的集成

1. 配置

|  |
| --- |
| #mybatis  mybatis.mapper-locations: classpath:mybatis/\*.xml  #mybatis.type-aliases-package: com.zte.mybatis.bean |

2）

|  |
| --- |
| @Mapper  **public** **interface** ZteUserMapper {  **int** countByExample(ZteUserExample example);  **int** deleteByExample(ZteUserExample example);  **int** deleteByPrimaryKey(Integer id);  **int** insert(ZteUser record);  **int** insertSelective(ZteUser record);  List<ZteUser> selectByExample(ZteUserExample example);  ZteUser selectByPrimaryKey(Integer id);  **int** updateByExampleSelective(@Param("record") ZteUser record, @Param("example") ZteUserExample example);  **int** updateByExample(@Param("record") ZteUser record, @Param("example") ZteUserExample example);  **int** updateByPrimaryKeySelective(ZteUser record);  **int** updateByPrimaryKey(ZteUser record);  } |

3）/spring-boot-mybatis-xml/src/main/resources/mybatis/mapper.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  <mapper namespace=*"com.zte.mybatis.mapper.ZteUserMapper"* >  <resultMap id=*"BaseResultMap"* type=*"com.zte.mybatis.bean.ZteUser"* >  <id column=*"id"* property=*"id"* jdbcType=*"INTEGER"* />  <result column=*"name"* property=*"name"* jdbcType=*"VARCHAR"* />  <result column=*"create\_time"* property=*"createTime"* jdbcType=*"TIMESTAMP"* />  </resultMap>  <sql id=*"Example\_Where\_Clause"* >  <where >  <foreach collection=*"oredCriteria"* item=*"criteria"* separator=*"or"* >  <if test=*"criteria.valid"* >  <trim prefix=*"("* suffix=*")"* prefixOverrides=*"and"* >  <foreach collection=*"criteria.criteria"* item=*"criterion"* >  <choose >  <when test=*"criterion.noValue"* >  and ${criterion.condition}  </when>  <when test=*"criterion.singleValue"* >  and ${criterion.condition} #{criterion.value}  </when>  <when test=*"criterion.betweenValue"* >  and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}  </when>  <when test=*"criterion.listValue"* >  and ${criterion.condition}  <foreach collection=*"criterion.value"* item=*"listItem"* open=*"("* close=*")"* separator=*","* >  #{listItem}  </foreach>  </when>  </choose>  </foreach>  </trim>  </if>  </foreach>  </where>  </sql>  <sql id=*"Update\_By\_Example\_Where\_Clause"* >  <where >  <foreach collection=*"example.oredCriteria"* item=*"criteria"* separator=*"or"* >  <if test=*"criteria.valid"* >  <trim prefix=*"("* suffix=*")"* prefixOverrides=*"and"* >  <foreach collection=*"criteria.criteria"* item=*"criterion"* >  <choose >  <when test=*"criterion.noValue"* >  and ${criterion.condition}  </when>  <when test=*"criterion.singleValue"* >  and ${criterion.condition} #{criterion.value}  </when>  <when test=*"criterion.betweenValue"* >  and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}  </when>  <when test=*"criterion.listValue"* >  and ${criterion.condition}  <foreach collection=*"criterion.value"* item=*"listItem"* open=*"("* close=*")"* separator=*","* >  #{listItem}  </foreach>  </when>  </choose>  </foreach>  </trim>  </if>  </foreach>  </where>  </sql>  <sql id=*"Base\_Column\_List"* >  id, name, create\_time  </sql>  <select id=*"selectByExample"* resultMap=*"BaseResultMap"* parameterType=*"com.zte.mybatis.bean.ZteUserExample"* >  select  <if test=*"distinct"* >  distinct  </if>  <include refid=*"Base\_Column\_List"* />  from zte\_user  <if test=*"\_parameter != null"* >  <include refid=*"Example\_Where\_Clause"* />  </if>  <if test=*"orderByClause != null"* >  order by ${orderByClause}  </if>  <if test=*"limitStart >= 0"* >  limit ${limitStart} , ${pageSize}  </if>  </select>  <select id=*"selectByPrimaryKey"* resultMap=*"BaseResultMap"* parameterType=*"java.lang.Integer"* >  select  <include refid=*"Base\_Column\_List"* />  from zte\_user  where id = #{id,jdbcType=INTEGER}  </select>  <delete id=*"deleteByPrimaryKey"* parameterType=*"java.lang.Integer"* >  delete from zte\_user  where id = #{id,jdbcType=INTEGER}  </delete>  <delete id=*"deleteByExample"* parameterType=*"com.zte.mybatis.bean.ZteUserExample"* >  delete from zte\_user  <if test=*"\_parameter != null"* >  <include refid=*"Example\_Where\_Clause"* />  </if>  </delete>  <insert id=*"insert"* parameterType=*"com.zte.mybatis.bean.ZteUser"* >  insert into zte\_user (id, name, create\_time  )  values (#{id,jdbcType=INTEGER}, #{name,jdbcType=VARCHAR}, #{createTime,jdbcType=TIMESTAMP}  )  </insert>  <insert id=*"insertSelective"* parameterType=*"com.zte.mybatis.bean.ZteUser"* >  insert into zte\_user  <trim prefix=*"("* suffix=*")"* suffixOverrides=*","* >  <if test=*"id != null"* >  id,  </if>  <if test=*"name != null"* >  name,  </if>  <if test=*"createTime != null"* >  create\_time,  </if>  </trim>  <trim prefix=*"values ("* suffix=*")"* suffixOverrides=*","* >  <if test=*"id != null"* >  #{id,jdbcType=INTEGER},  </if>  <if test=*"name != null"* >  #{name,jdbcType=VARCHAR},  </if>  <if test=*"createTime != null"* >  #{createTime,jdbcType=TIMESTAMP},  </if>  </trim>  </insert>  <select id=*"countByExample"* parameterType=*"com.zte.mybatis.bean.ZteUserExample"* resultType=*"java.lang.Integer"* >  select count(\*) from zte\_user  <if test=*"\_parameter != null"* >  <include refid=*"Example\_Where\_Clause"* />  </if>  </select>  <update id=*"updateByExampleSelective"* parameterType=*"map"* >  update zte\_user  <set >  <if test=*"record.id != null"* >  id = #{record.id,jdbcType=INTEGER},  </if>  <if test=*"record.name != null"* >  name = #{record.name,jdbcType=VARCHAR},  </if>  <if test=*"record.createTime != null"* >  create\_time = #{record.createTime,jdbcType=TIMESTAMP},  </if>  </set>  <if test=*"\_parameter != null"* >  <include refid=*"Update\_By\_Example\_Where\_Clause"* />  </if>  </update>  <update id=*"updateByExample"* parameterType=*"map"* >  update zte\_user  set id = #{record.id,jdbcType=INTEGER},  name = #{record.name,jdbcType=VARCHAR},  create\_time = #{record.createTime,jdbcType=TIMESTAMP}  <if test=*"\_parameter != null"* >  <include refid=*"Update\_By\_Example\_Where\_Clause"* />  </if>  </update>  <update id=*"updateByPrimaryKeySelective"* parameterType=*"com.zte.mybatis.bean.ZteUser"* >  update zte\_user  <set >  <if test=*"name != null"* >  name = #{name,jdbcType=VARCHAR},  </if>  <if test=*"createTime != null"* >  create\_time = #{createTime,jdbcType=TIMESTAMP},  </if>  </set>  where id = #{id,jdbcType=INTEGER}  </update>  <update id=*"updateByPrimaryKey"* parameterType=*"com.zte.mybatis.bean.ZteUser"* >  update zte\_user  set name = #{name,jdbcType=VARCHAR},  create\_time = #{createTime,jdbcType=TIMESTAMP}  where id = #{id,jdbcType=INTEGER}  </update>  </mapper> |

### 25.4快速批量生成 bean，mapper，xml

使用 mybatis generator

1. 添加配置

|  |
| --- |
| #配置mysql驱动的位置  driver.path=C:/Users/BYF/.m2/repository/mysql/mysql-connector-java/5.1.39/mysql-connector-java-5.1.39.jar  #添加注解（若不需要，可以去除，把对应的xml里面的AnnotationPlugin禁用即可）  AnnotationPlugin.annotationClass=org.apache.ibatis.annotations.Mapper  AnnotationPlugin.annotationName=@Mapper  #JDBC的链接  jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://127.0.0.1:3306/spring\_boot\_demo?useUnicode=true&characterEncoding=utf-8  jdbc.username=root  jdbc.password=1234  #javaModelGenerator生成  javaModelGenerator.targetPackage=com.zte.mybatis.bean  javaModelGenerator.targetProject=C:/Users/BYF/workspace/spring-boot-mybatis-xml/src/main/java  #javaClientGenerator生成  javaClientGenerator.targetPackage=com.zte.mybatis.mapper  javaClientGenerator.targetProject=C:/Users/BYF/workspace/spring-boot-mybatis-xml/src/main/java  #sqlMapGenerator生成  sqlMapGenerator.targetPackage=mybatis  sqlMapGenerator.targetProject=C:/Users/BYF/workspace/spring-boot-mybatis-xml/src/main/resources |

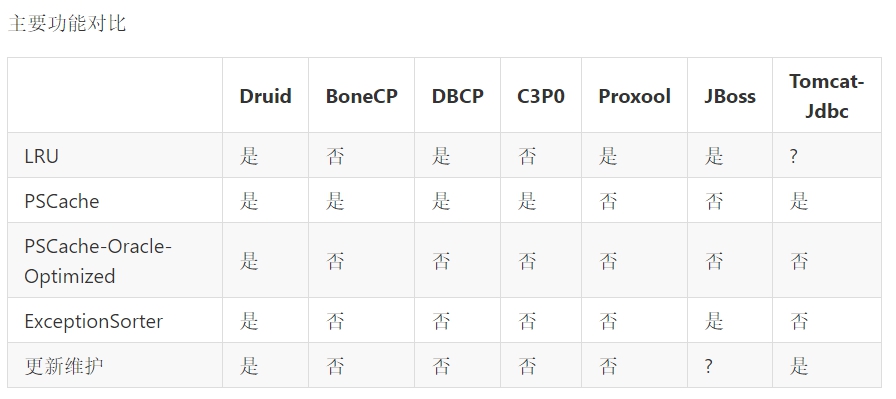
2）运行main自动生成bean，mapper，mapper.xml

|  |
| --- |
| **public** **class** SpringBoot {  **private** **static** Logger *logger* = Logger.*getLogger*(SpringBoot.**class**);  **public** **static** **void** main(String[] args) **throws** IOException, XMLParserException, InvalidConfigurationException, SQLException, InterruptedException {  *logger*.info("Start generator ...");  List<String> warnings = **new** ArrayList<String>();  **boolean** overwrite = **true**; // 是否覆盖原来的文件  File configFile = **new** File("C:/Users/BYF/workspace/mybatis-generator/src/test/resources/spring-boot.xml");  ConfigurationParser cp = **new** ConfigurationParser(warnings);  Configuration config = cp.parseConfiguration(configFile);  DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);  MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config, callback, warnings);  myBatisGenerator.generate(**null**);  *logger*.info("Finish generator");  }  } |

## 第26节Spring Boot 集成 Druid

### 26.1Druid 的介绍

<https://github.com/alibaba/druid/wiki/%E5%90%84%E7%A7%8D%E6%95%B0%E6%8D%AE%E5%BA%93%E8%BF%9E%E6%8E%A5%E6%B1%A0%E5%AF%B9%E6%AF%94>



中文文档：<https://github.com/alibaba/druid/wiki/%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98>

### 26.2添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.9</version>  </dependency> |

### 26.3添加配置

|  |
| --- |
| #Druid  #spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource |

注意：关于 spring.datasource.type 的说明

旧版本不支持这个属性，1.3.x 开始支持，但是 1.4.0 不支持，1.4.1 重新支持。

### 26.4添加 druid 的支持

1）添加数据源配置类DruidConfiguration

|  |
| --- |
| @Configuration  **public** **class** DruidConfiguration {  @ConditionalOnClass(DruidDataSource.**class**)  @ConditionalOnProperty(name = "spring.datasource.type", havingValue = "com.alibaba.druid.pool.DruidDataSource", matchIfMissing = **true**)  **static** **class** Druid **extends** DruidConfiguration {  @Bean  @ConfigurationProperties("spring.datasource.druid")  **public** DruidDataSource dataSource(DataSourceProperties properties) {  DruidDataSource druidDataSource = (DruidDataSource) properties.initializeDataSourceBuilder()  .type(DruidDataSource.**class**).build();  DatabaseDriver databaseDriver = DatabaseDriver.*fromJdbcUrl*(properties.determineUrl());  String validationQuery = databaseDriver.getValidationQuery();  **if** (validationQuery != **null**) {  druidDataSource.setValidationQuery(validationQuery);  }  **return** druidDataSource;  }  }  } |

2）添加数据库连接池配置

|  |
| --- |
| #初始化连接大小  spring.datasource.druid.initial-size=8  #最小空闲连接数  spring.datasource.druid.min-idle=5  #最大连接数  spring.datasource.druid.max-active=10  #查询超时时间  spring.datasource.druid.query-timeout=6000  #事务查询超时时间  spring.datasource.druid.transaction-query-timeout=6000  #关闭空闲连接超时时间  spring.datasource.druid.remove-abandoned-timeout=1800  #filter  #spring.datasource.druid.filter-class-names=stat  spring.datasource.druid.filters=stat,config |

### 26.5监控

1. 添加DruidStatViewServlet

|  |
| --- |
| @WebServlet(urlPatterns = { "/druid/\*" }, initParams = { @WebInitParam(name = "loginUsername", value = "root"),  @WebInitParam(name = "loginPassword", value = "root") })  **public** **class** DruidStatViewServlet **extends** StatViewServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  } |

1. 添加过滤器DruidWebStatFilter

|  |
| --- |
| @WebFilter(filterName = "druidWebStatFilter", urlPatterns = "/\*", initParams = {  @WebInitParam(name = "exclusions", value = "\*.js,\*.gif,\*.jpg,\*.bmp,\*.png,\*.css,\*.ico,/druid/\*") })  **public** **class** DruidWebStatFilter **extends** WebStatFilter {  } |

3）ServletInitializer对象初始化

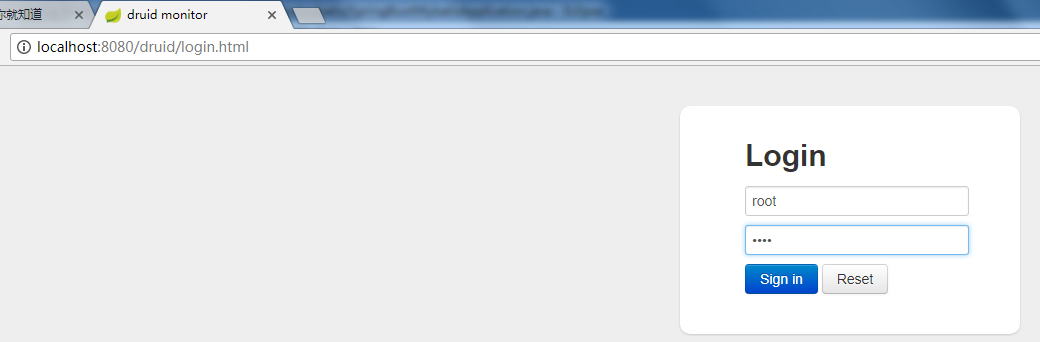
|  |
| --- |
| **public** **class** ServletInitializer **extends** SpringBootServletInitializer {  @Override  **protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {  **return** application.sources(SpringBootMybatisApplication.**class**);  }  } |

4）App

|  |
| --- |
| @ServletComponentScan  @SpringBootApplication  @ImportResource(locations = { "classpath:druid-bean.xml" })  **public** **class** SpringBootMybatisApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(SpringBootMybatisApplication.**class**, args);  }  } |

4）测试

启动SpringBoot应用，访问：<http://localhost:8080/druid/login.html>





5）sql 监控配置

|  |
| --- |
| #filter  #spring.datasource.druid.filter-class-names=stat  spring.datasource.druid.filters=stat,config |

1. spring 监控配置：

<https://github.com/alibaba/druid/wiki/%E9%85%8D%E7%BD%AE_Druid%E5%92%8CSpring%E5%85%B3%E8%81%94%E7%9B%91%E6%8E%A7%E9%85%8D%E7%BD%AE>

1. 按类型拦截配置；
2. 方法名正则匹配拦截配置；
3. 按照BeanId来拦截配置。

添加/spring-boot-druid-pool/src/main/resources/druid-bean.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd"*>  <!-- 配置\_Druid和Spring关联监控配置 -->  <bean id=*"druid-stat-interceptor"*  class=*"com.alibaba.druid.support.spring.stat.DruidStatInterceptor"*></bean>  <!-- 方法名正则匹配拦截配置 -->  <bean id=*"druid-stat-pointcut"* class=*"org.springframework.aop.support.JdkRegexpMethodPointcut"*  scope=*"prototype"*>  <property name=*"patterns"*>  <list>  <value>com.zte.mybatis.mapper.\*</value>  </list>  </property>  </bean>  <aop:config proxy-target-class=*"true"*>  <aop:advisor advice-ref=*"druid-stat-interceptor"*  pointcut-ref=*"druid-stat-pointcut"* />  </aop:config>  </beans> |

## 第27节Spring Boot 集成 Swagger

### 27.1 Swagger 是什么

Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

http://swagger.io/

Springfox 的前身是 swagger-springmvc，是一个开源的 API doc 框架，可以将我们的 Controller 的方法以文档的形式展现，基于 Swagger。

<http://springfox.github.io/springfox/>

### 27.2添加Swagger依赖

|  |
| --- |
| <!-- Swagger -->  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger-ui</artifactId>  <version>2.6.0</version>  </dependency>  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger2</artifactId>  <version>2.6.0</version>  </dependency> |

### 27.3添加配置类

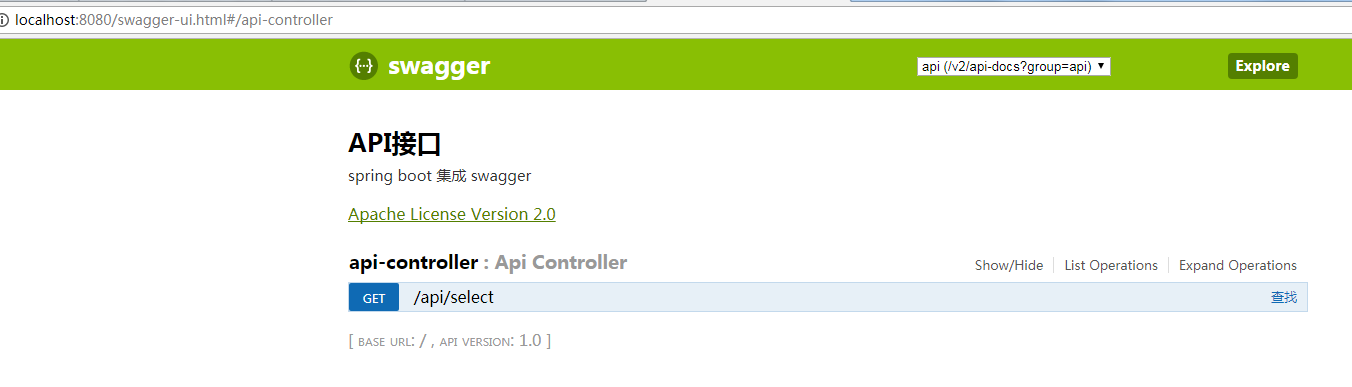
|  |
| --- |
| @Configuration  @EnableSwagger2  **public** **class** Swagger2Configuration {  @Bean  **public** Docket accessToken() {  **return** **new** Docket(DocumentationType.***SWAGGER\_2***).groupName("api")// 定义组  .select() // 选择那些路径和 api 会生成 document  .apis(RequestHandlerSelectors.*basePackage*("com.zte.mybatis")) // 拦截的包路径  .paths(*regex*("/api/.\*"))// 拦截的接口路径  .build() // 创建  .apiInfo(apiInfo()); // 配置说明  }  **private** ApiInfo apiInfo() {  **return** **new** ApiInfoBuilder()//  .title("API接口")// 标题  .description("spring boot 集成 swagger")// 描述  .termsOfServiceUrl("http://www.zte.com.cn")  .license("Apache License Version 2.0")// 开源协议  .licenseUrl("https://github.com/springfox/springfox/blob/master/LICENSE")// 地址  .version("1.0")// 版本  .build();  }  } |

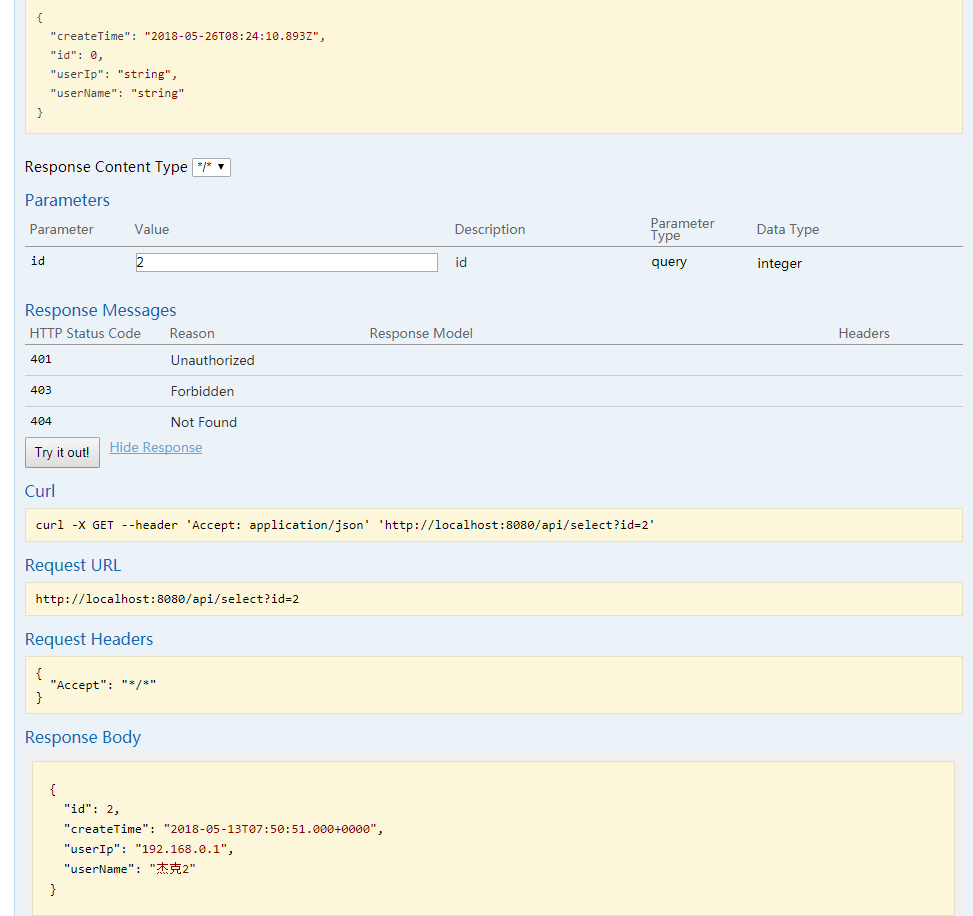
### 27.4测试

1）添加Api描述注解

|  |
| --- |
| @RestController  @RequestMapping("/api")  **public** **class** ApiController {  @Autowired  **private** ZteUserLogMapper zteUserLogMapper;  @ApiOperation(value = "查找", notes = "根据用户 ID 查找用户")  @RequestMapping(value = "/select", method = RequestMethod.***GET***)  **public** ZteUserLog get(@RequestParam(defaultValue = "2") Integer id) {  **return** zteUserLogMapper.selectByPrimaryKey(id);  }  } |

2）访问：http://localhost:8080/swagger-ui.html





1. 自定义(注解的使用)

| **Name** | **Description** |
| --- | --- |
| [@Api](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "api) | Marks a class as a Swagger resource. |
| [@ApiImplicitParam](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apiimplicitparam-apiimplicitparams) | Represents a single parameter in an API Operation. |
| [@ApiImplicitParams](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apiimplicitparam-apiimplicitparams) | A wrapper to allow a list of multiple ApiImplicitParam objects. |
| [@ApiModel](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apimodel) | Provides additional information about Swagger models. |
| [@ApiModelProperty](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apimodelproperty) | Adds and manipulates data of a model property. |
| [@ApiOperation](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apioperation) | Describes an operation or typically a HTTP method against a specific path. |
| [@ApiParam](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apiparam) | Adds additional meta-data for operation parameters. |
| [@ApiResponse](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apiresponses-apiresponse) | Describes a possible response of an operation. |
| [@ApiResponses](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "apiresponses-apiresponse) | A wrapper to allow a list of multiple ApiResponse objects. |
| [@Authorization](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "authorization-authorizationscope) | Declares an authorization scheme to be used on a resource or an operation. |
| [@AuthorizationScope](https://github.com/swagger-api/swagger-core/wiki/Annotations" \l "authorization-authorizationscope) | Describes an OAuth2 authorization scope. |

@ApiIgnore

忽略暴露的 api

@ApiOperation(value = "查找", notes = "根据用户 ID 查找用户")

添加说明

其他注解：

@Api：用在类上，说明该类的作用

@ApiImplicitParams：用在方法上包含一组参数说明

@ApiResponses：用于表示一组响应

## 第28节生产部署-注意事项和如何使用脚本

### 28.1注意事项

1. 去除不需要的 jar

开发工具 jar：spring-boot-devtools

1. 监控一定要做好权限控制或者去除

控制 jar：spring-boot-starter-actuator druid 的监控 swagger 的接口

1. 打包，跳过测试maven： clean package -Dmaven.test.skip=true

28.2脚本

|  |
| --- |
| #!/bin/sh  ## chang here  SERVICE\_DIR=/roncoo/spring-boot-demo  SERVICE\_NAME=spring-boot-demo-31-1-0.0.1-SNAPSHOT SPRING\_PROFILES\_ACTIVE=dev  ## java env  export JAVA\_HOME=/opt/jdk1.7.0\_79 export JRE\_HOME=$JAVA\_HOME/jre  case "$1" in  start)  procedure=`ps -ef | grep -w "$SERVICE\_NAME" |grep -w "java"| grep -v "grep" | awk '{print $2}'`  if test "$procedure" = "" ; then  echo "start ..." if [ "$2" != "" ]; then  SPRING\_PROFILES\_ACTIVE=$2  fi  echo "spring.profiles.active=$SPRING\_PROFILES\_ACTIVE" exec nohup $JRE\_HOME/bin/java -Xms128m -Xmx512m -jar  $SERVICE\_DIR/$SERVICE\_NAME\.jar -- spring.profiles.active=$SPRING\_PROFILES\_ACTIVE > /dev/null 2>&1 &  echo "start success"  else  echo "$SERVICE\_NAME is start"  fi  ;;  stop)  procedure=`ps -ef | grep -w "$SERVICE\_NAME" |grep -w "java"| grep -v "grep" | awk '{print $2}'`  if test "$procedure" = "" ; then  echo "$SERVICE\_NAME is stop"  else  kill -9 $procedure sleep 1  argprocedure=`ps -ef | grep -w "$SERVICE\_NAME" |grep -w "java"| grep -v "grep" | awk '{print $2}'`  if test "$argprocedure" = "" ; then  echo "$SERVICE\_NAME stop success"  else  kill -9 $argprocedure  echo "$SERVICE\_NAME stop error"  fi  fi  ;;  restart)  $0 stop sleep 1  $0 start $2  ;;  \*)  echo "usage: $0 [start|stop|restart] [dev|test|prod]"  ;;  esac |