

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

# SpringBoot概述

Spring 框架对于很多 Java 开发人员来说都不陌生。自从 2002 年发布以来，Spring 框架已经成为企业应用开发领域非常流行的基础框架。有大量的企业应用基于 Spring 框架来开发。Spring 框架包含几十个不同的子项目，涵盖应用开发的不同方面。如此多的子项目和组件，一方面方便了开发人员的使用，另外一个方面也带来了使用方面的问题。每个子项目都有一定的学习曲线。开发人员需要了解这些子项目和组件的具体细节，才能知道如何把这些子项目整合起来形成一个完整的解决方案。在如何使用这些组件上，并没有相关的最佳实践提供指导。对于新接触 Spring 框架的开发人员来说，并不知道如何更好的使用这些组件。Spring 框架的另外一个常见问题是要快速创建一个可以运行的应用比较麻烦。**Spring Boot 是 Spring 框架的一个新的子项目，用于创建 Spring 4.0 项目。**它的开发始于 2013 年。2014 年 4 月发布 1.0.0 版本。它可以自动配置 Spring 的各种组件，并不依赖代码生成和 XML 配置文件。Spring Boot 也提供了对于常见场景的推荐组件配置。Spring Boot 可以大大提升使用 Spring 框架时的开发效率

## Spring介绍

## SpringBoot简介

在您第1次接触和学习Spring框架的时候，是否因为其繁杂的配置而退却了？在你第n次使用Spring框架的时候，是否觉得一堆反复黏贴的配置有一些厌烦？那么您就不妨来试试使用Spring Boot来让你更易上手，更简单快捷地构建Spring应用！

Spring Boot让我们的Spring应用变的更轻量化。比如：你可以仅仅依靠一个Java类来运行一个Spring引用。你也可以打包你的应用为jar并通过使用java -jar来运行你的Spring Web应用。

## 系统要求：

Java 7及以上

Spring Framework 4.1.5及以上

**本文采用**Java 1.8.0\_73**、**Spring Boot 1.3.2**调试通过。**

## 简介：

优点：

快速构建项目。

对主流开发框架的无配置集成。

项目可独立运行，无须外部依赖Servlet容器。

提供运行时的应用监控。

极大的提高了开发、部署效率。

与云计算的天然集成。

轻松的**创建独立运行的程序**，非常容易构建独立的服务组件，是实现分布式架构、微服务架构利器。

缺点：

如果你不认同spring框架，也许这就是缺点。

## Spring boot的优点

1. 创建独立的Spring应用程序

2. 嵌入的Tomcat，无需部署WAR文件

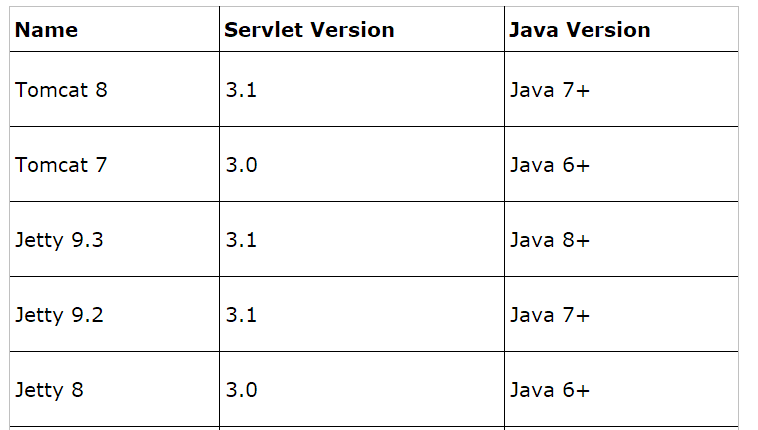
3.提供一系列的“starter” 来简化的Maven配置，不需要添加很多依赖。

4. 自动配置Spring

5. 提供生产就绪型功能，如指标，健康检查和外部配置

6. 绝对没有代码生成和对XML没有要求配置

## 内置Servlet容器

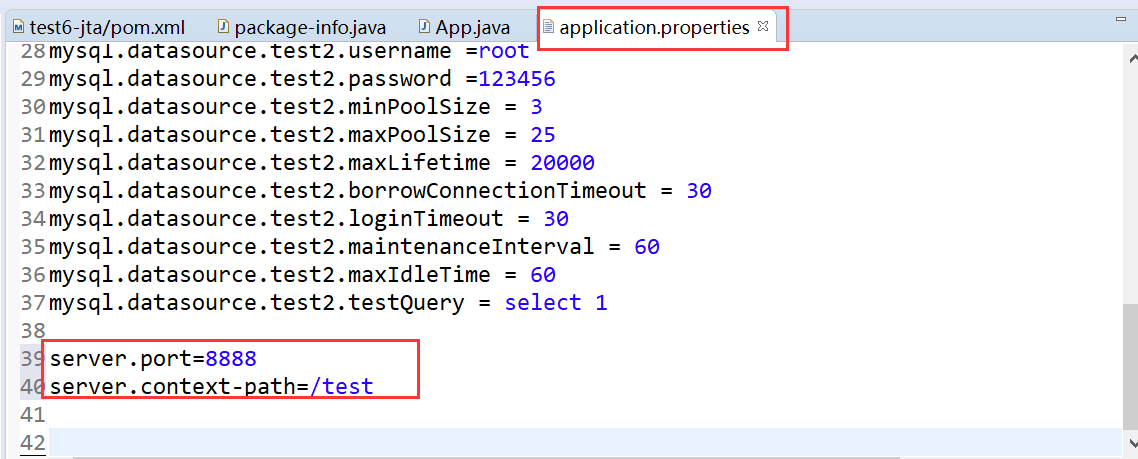


## 修改端口号

### 在application.properties中

server.port=8888

server.context-path=/test



### 在application.yml中

注意冒号后的只能用空格,不能用tab

|  |
| --- |
| server:  port: 8090  context-path: /test-yml |

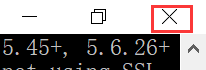
## Springboot 打包部署

1. 先打成war包或者jar包
2. 使用java -jar test3-0.0.1-SNAPSHOT.jar 运行即可

打包时添加下面依赖代码

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>utf-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <mainClass>com.gyf.app.App</mainClass>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

在终端要停止服务器时，使用Ctrl + C,直接点x，会端口没有关掉



# 快速入门

## SpringBoot入门案例

### Step01:

解压一个maven3.5到本地

### Step02:

修改maven的conf\setting的本地仓库存储路径

### Step03:

修改maven的conf\setting 中远程仓库为阿里云的

### Step04:

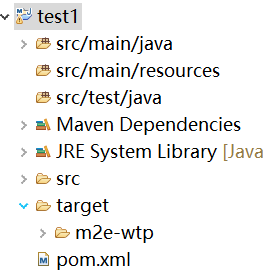
把Eclipse中的本地和全局的仓库文件都改成conf\setting

### Step05:

使用neon版本的eclipse创建一个 maven war项目(test1)

使用neon版本的eclipse创建一个 maven jar项目(test2)

这两种都可以，但一般都使 用jar,因为spring是用于服务，不建议与jsp使用



### Step06:

在pom.xml添加spring-boot-starter-web依赖，如图

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |
| spring-boot-starter-parent作用  在pom.xml中引入spring-boot-start-parent,它可以提供dependency management,也就是说依赖管理，引入以后在申明其它dependency的时候就不需要version了，后面可以看到。  spring-boot-starter-web作用  springweb 核心组件 |

### Step07:

写一个控制器如下，然后写一个main方法，把程序跑起来：

|  |
| --- |
| **package** com.gyf.web.controller;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.EnableAutoConfiguration;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** org.springframework.web.bind.annotation.RestController;  @RestController//相当于声明Controller - 提共restful 风格  @EnableAutoConfiguration//自动配置，不需要写spring的配置文件  **class** HelloController {  @RequestMapping("/hello")//映射路径  @ResponseBody//响应体  **public** String hello() {  **return** "Hello World";  }    **public** **static** **void** main(String[] args) {  //启动程序  SpringApplication.*run*(HelloController.**class**, args);  }  } |

### Step08:

在浏览器中访问http://localhost:8080/hello

|  |
| --- |
|  |

**注：Maven项目上总有一个小红叉**

解决决思路：

1、查看problems选项卡，里面有对问题的描述

2、工程右键->maven->update project(勾选Force Update)

## 创建一个Maven工程

**名为”springboot-helloworld” 类型为Jar工程项目**



## pom文件引入依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.3.RELEASE</version>  </parent>  <dependencies>  <!—SpringBoot web 组件 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |
| **spring-boot-starter-parent作用**  **在pom.xml中引入spring-boot-start-parent,spring官方的解释叫什么stater poms,它可以提供dependency management,也就是说依赖管理，引入以后在申明其它dependency的时候就不需要version了，后面可以看到。**  **spring-boot-starter-web作用**  **springweb 核心组件**  **spring-boot-maven-plugin作用**  **如果我们要直接Main启动spring，那么以下plugin必须要添加，否则是无法启动的。如果使用maven 的spring-boot:run的话是不需要此配置的。（我在测试的时候，如果不配置下面的plugin也是直接在Main中运行的。）** |

## Spring Boot依赖介绍

|  |  |
| --- | --- |
| **spring-boot-starter** | 核心 POM，包含自动配置支持、日志库和对 YAML 配置文件的支持。 |
| spring-boot-starter-amqp | 通过 spring-rabbit 支持 AMQP |
| spring-boot-starter-aop | 包含 spring-aop 和 AspectJ 来支持面向切面编程（AOP）。 |
| spring-boot-starter-batch | 支持 Spring Batch，包含 HSQLDB。 |
| spring-boot-starter-data-jpa | 包含 spring-data-jpa、spring-orm 和 Hibernate 来支持 JPA。 |
| spring-boot-starter-data-mongodb | 包含 spring-data-mongodb 来支持 MongoDB。 |
| spring-boot-starter-data-rest | 通过 spring-data-rest-webmvc 支持以 REST 方式暴露 Spring Data 仓库。 |
| spring-boot-starter-jdbc | 支持使用 JDBC 访问数据库 |
| spring-boot-starter-security | 包含 spring-security。 |
| spring-boot-starter-test | 包含常用的测试所需的依赖，如 JUnit、Hamcrest、Mockito 和 spring-test 等。 |
| spring-boot-starter-velocity | 支持使用 Velocity 作为模板引擎。 |
| **spring-boot-starter-web** | 支持 Web 应用开发，包含 Tomcat 和 spring-mvc。 |
| spring-boot-starter-websocket | 支持使用 Tomcat 开发 WebSocket 应用。 |
| spring-boot-starter-ws | 支持 Spring Web Services |
| spring-boot-starter-actuator | 添加适用于生产环境的功能，如性能指标和监测等功能。 |
| spring-boot-starter-remote-shell | 添加远程 SSH 支持 |
| spring-boot-starter-jetty | 使用 Jetty 而不是默认的 Tomcat 作为应用服务器。 |
| spring-boot-starter-log4j | 添加 Log4j 的支持 |
| spring-boot-starter-logging | 使用 Spring Boot 默认的日志框架 Logback |
| spring-boot-starter-tomcat | 使用 Spring Boot 默认的 Tomcat 作为应用服务器。 |
|  |  |

### spring-boot-starter-web

* POM 文件中可以看到，应用所声明的依赖很少
* 只有一个“org.springframework.boot:spring-boot-starter-web”
* 而不是像其他 Spring 项目一样需要声明很多的依赖。
* 当使用 Maven 命令“mvn dependency:tree”来查看项目实际的依赖时
* 发现其中包含SpringMVC框架、SLF4J、Jackson、Hibernate Validator 和 Tomcat 等依赖。
* 这实际上 Spring 推荐的 Web 应用中使用的开源库的组合。

### EnableAutoConfiguration

* EnableAutoConfiguration”注解的作用在于让 Spring Boot 根据应用所声明的依赖来对 Spring 框架进行自动配置，这就减少了开发人员的工作量。
* Spring Boot 推荐采用基于 Java 注解的配置方式，而不是传统的 XML。只需要在主配置 Java 类上添加“@EnableAutoConfiguration”注解就可以启用自动配置。
* 注解“@RestController”和”@RequestMapping”由 Spring MVC 提供，用来创建 REST 服务。这两个注解和 Spring Boot 本身并没有关系。

## 编写HelloWorld服务

创建package命名为com.ysy.controller（根据实际情况修改）

创建HelloController类，内容如下

|  |
| --- |
| @RestController  @EnableAutoConfiguration  public class HelloController {  @RequestMapping("/hello")  public String index() {  return "Hello World";  }  public static void main(String[] args) {  SpringApplication.*run*(HelloController.class, args);  }  } |

## @RestController

**在上加上RestController 表示修饰该Controller所有的方法返回JSON格式,直接可以编写**

**Restful接口**

## @EnableAutoConfiguration

注解:作用在于让 Spring Boot   根据应用所声明的依赖来对 Spring 框架进行自动配置  
   这个注解告诉Spring Boot根据添加的jar依赖猜测你想如何配置Spring。由于spring-boot-starter-web添加了Tomcat和Spring MVC，所以auto-configuration将假定你正在开发一个web应用并相应地对Spring进行设置。

## SpringApplication.*run*(HelloController.class, args);

标识为启动类

## 启动方式

### SpringBoot启动方式1

在控制器配置EnableAutoConfiguration并使用SpringApplication启动程序

Springboot默认端口号为8080

|  |
| --- |
| @RestController  @EnableAutoConfiguration  public class HelloController {  @RequestMapping("/hello")  public String index() {  return "Hello World";  }  public static void main(String[] args) {  SpringApplication.*run*(HelloController.class, args);  }  } |

启动主程序，打开浏览器访问http://localhost:8080/index，可以看到页面输出Hello World

### SpringBoot启动方式2【常用】

@ComponentScan(basePackages = "com.ysy.controller")---控制器扫包范围

|  |
| --- |
| @ComponentScan(basePackages = "com.ysy.controller")  @EnableAutoConfiguration  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

创建一个App类，在App类中配置EnableAutoConfiguration和组件扫描ComponentScan,

然后使用SpringApplication启动程序,这样就可以访问多个Controller了.

|  |
| --- |
| 创建一个User控制器 |
| 创建App类 |

# Web开发

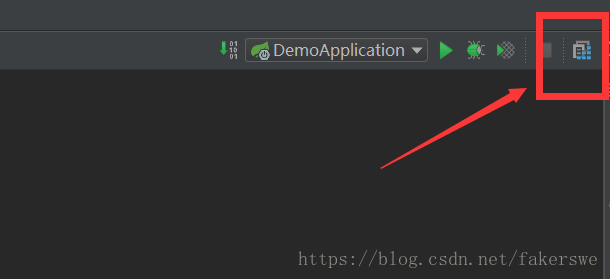
## idea配置

**一、文章简述**

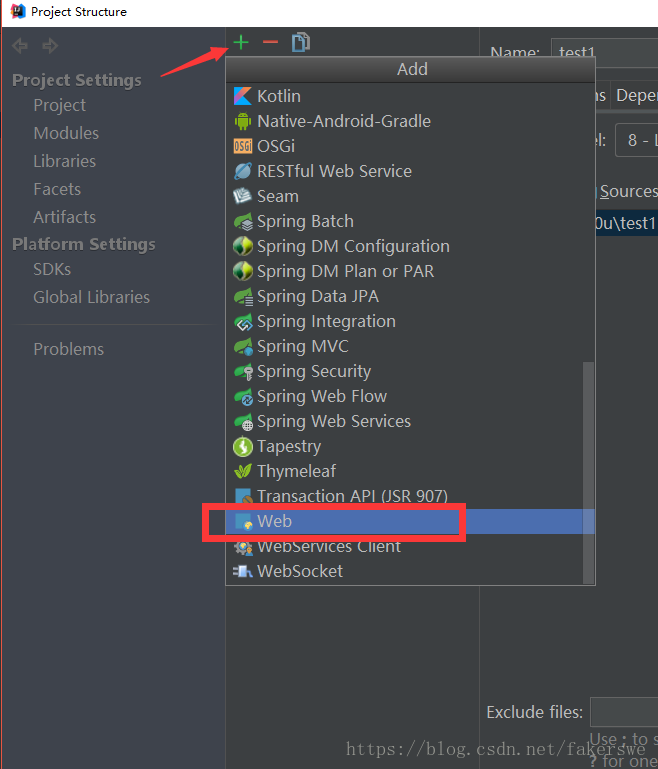
使用IDEA工具创建的SpringBoot项目本身是没有webapp目录的。如果我们想要添加webapp目录的话，可以手动添加。

**二、操作步骤**

**1）点击IDEA右上角的Project Structure**

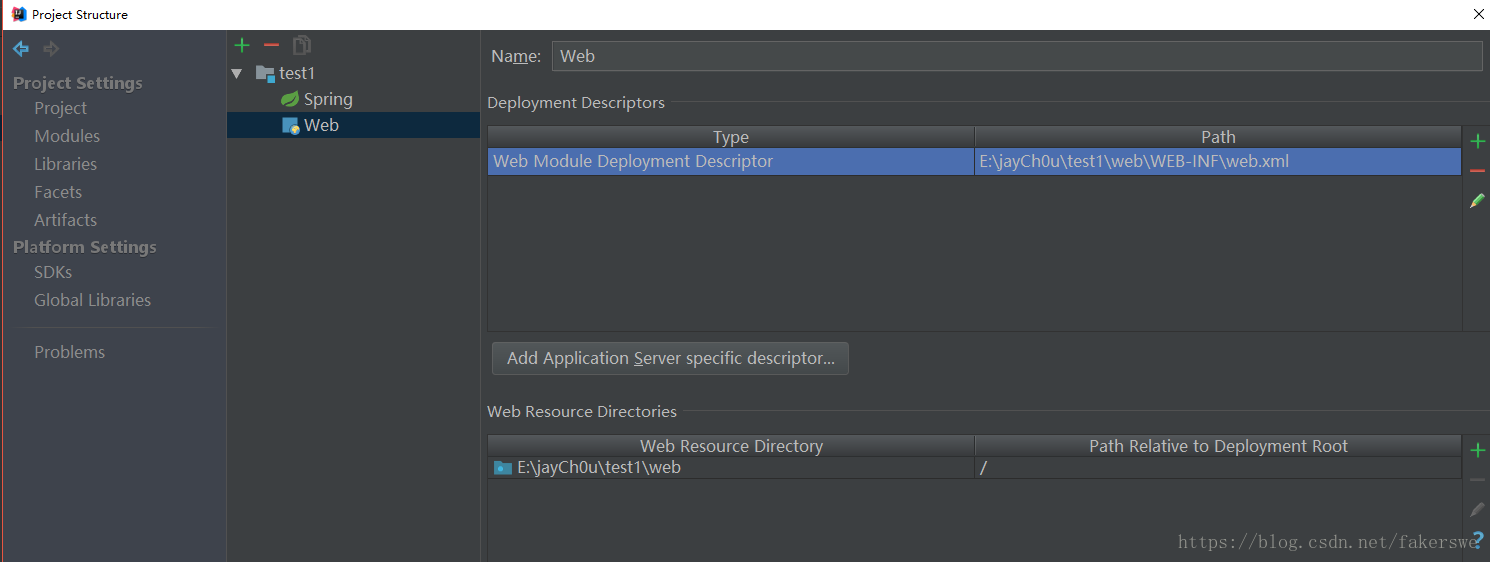


**2）先点击下图中的+号，再点击Web**

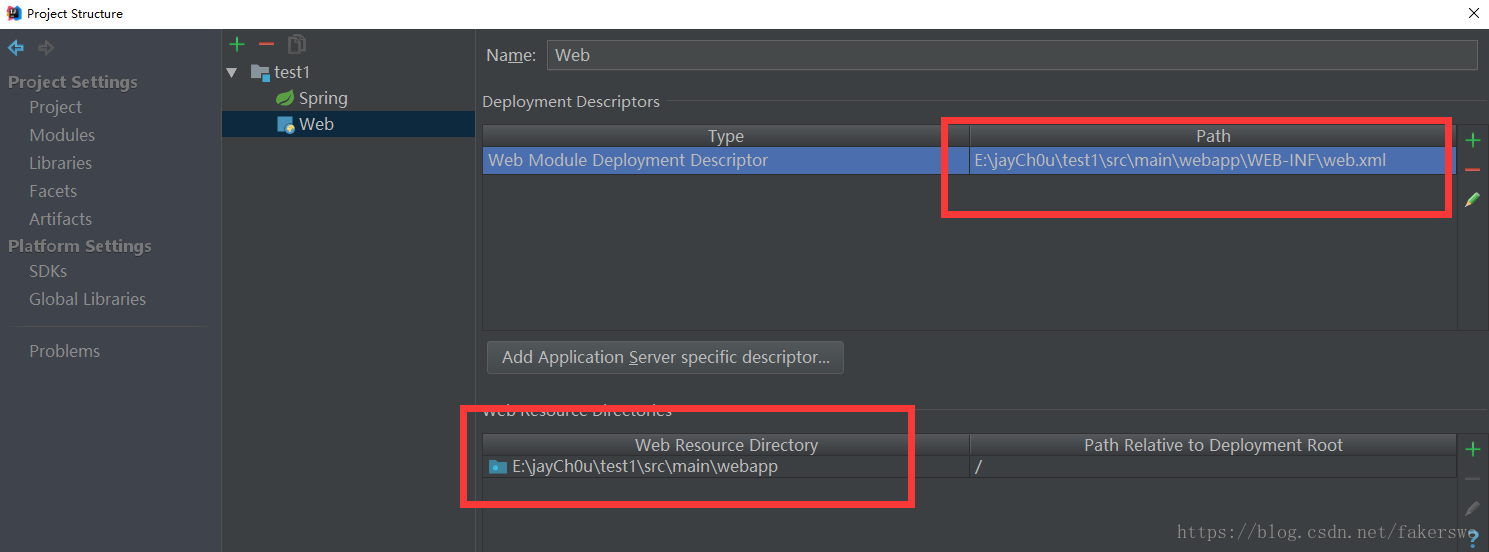


**3）修改Web模板的位置**

 ①web模板未修改前的位置如下：



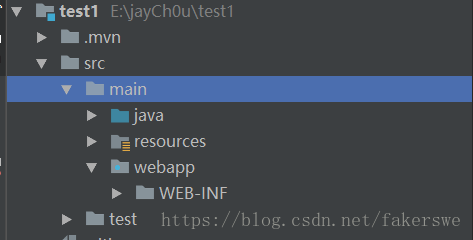
②先修改Path路径（点击上图中path框右边的小铅笔图标即可编辑），再修改Web Resource Directory



注意：本项目是把webapp放置在main目录底下。大家可以根据自己的需求进行修改webapp的位置。

PS：上图中的test1是项目名称。

**③**最后点击OK，附上效果图：



## 静态资源访问

在我们开发Web应用的时候，需要引用大量的js、css、图片等静态资源。

默认配置

Spring Boot默认提供静态资源目录位置需置于classpath下，目录名需符合如下规则：

/static

/public

/resources

/META-INF/resources

举例：我们可以在src/main/resources/目录下创建static，在该位置放置一个图片文件。启动程序后，尝试访问http://localhost:8080/D.jpg。如能显示图片，配置成功。

访问的时候不需要加static，

默认以static为根目录

## 渲染Web页面

渲染Web页面

在之前的示例中，我们都是通过@RestController来处理请求，所以返回的内容为json对象。那么如果需要渲染html页面的时候，要如何实现呢？

模板引擎

在动态HTML实现上Spring Boot依然可以完美胜任，并且提供了多种模板引擎的默认配置支持，所以在推荐的模板引擎下，我们可以很快的上手开发动态网站。

Spring Boot提供了默认配置的模板引擎主要有以下几种：

* Thymeleaf
* FreeMarker
* Velocity
* Groovy
* Mustache

Spring Boot建议使用这些模板引擎，避免使用JSP，若一定要使用JSP将无法实现Spring Boot的多种特性，具体可见后文：支持JSP的配置

当你使用上述模板引擎中的任何一个，它们默认的模板配置路径为：src/main/resources/templates。当然也可以修改这个路径，具体如何修改，可在后续各模板引擎的配置属性中查询并修改。

## 使用JSP渲染Web视图

### pom文件引入以下依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.3.RELEASE</version>  </parent>  <dependencies>  <!-- SpringBoot 核心组件 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </dependency>  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  </dependencies> |

### 在application.properties创建以下配置

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp |

### 后台代码

|  |
| --- |
| @Controller  public class IndexController {  @RequestMapping("/index")  public String index() {  return "index";  }  } |

## 静态资源的访问

在我们开发Web应用的时候，需要引用大量的js、css、图片等静态资源。

Spring Boot默认提供静态资源目录位置**需置于classpath下**，目录名需符合如下规则：

**/static**

**/public**

**/resources**

**/META-INF/resources**

举例：我们可以在src/main/resources/目录下创建static，在该位置放置一个图片文件。启动程序后，尝试访问http://localhost:8080/imgs/d.jpg。如能显示图片，配置成功。

|  |
| --- |
|  |

## 自动返回json格式数据

|  |
| --- |
|  |
|  |

## 全局捕获异常

@ExceptionHandler 表示拦截异常

@ControllerAdvice

controller 的一个辅助类，最常用的就是作为全局异常处理的切面类

可以指定扫描范围

约定了几种可行的返回值，如果是直接返回 model 类的话，需要使用

@ResponseBody 进行 json 转换

案例：

|  |
| --- |
| **在com.gyf.web.exception包中定义一个全局异常类** |
| @ControllerAdvice//切面  **public** **class** GlobalExceptionHandler {  @ExceptionHandler(RuntimeException.**class**)//捕获运行时异常  @ResponseBody  **public** Map<String,Object> exceptionHander(){  Map<String, Object> map = **new** HashMap<String, Object>();  map.put("errorCode", "101");  map.put("errorMsg", "系統错误!");  **return** map;  }  } |
| **在启动spring中，配置扫描包为com.gyf.web** |
|  |
| **在某个映射的方法中添加个int i = 10/0的算术异常** |
|  |
| 访问上的个路径结果为 |
|  |

## 全局捕获异常

@ExceptionHandler 表示拦截异常

@ControllerAdvice 是 controller 的一个辅助类，最常用的就是作为全局异常处理的切面类

@ControllerAdvice 可以指定扫描范围

@ControllerAdvice 约定了几种可行的返回值，如果是直接返回 model 类的话，需要使用 @ResponseBody 进行 json 转换

返回 String，表示跳到某个 view

返回 modelAndView

返回 model + @ResponseBody

|  |
| --- |
| @ControllerAdvice  **public** **class** GlobalExceptionHandler {  @ExceptionHandler(RuntimeException.**class**)  @ResponseBody  **public** Map<String, Object> exceptionHandler() {  Map<String, Object> map = **new** HashMap<String, Object>();  map.put("errorCode", "101");  map.put("errorMsg", "系統错误!");  **return** map;  }  } |

## 渲染Web页面

**模板引擎**

在动态HTML实现上Spring Boot依然可以完美胜任，并且提供了多种模板引擎的默认配置支持，所以在推荐的模板引擎下，我们可以很快的上手开发动态网站。

Spring Boot提供了默认配置的模板引擎主要有以下几种：

* Thymeleaf
* FreeMarker
* Velocity
* Groovy
* Mustache

Springboot+freemarker

**Spring Boot建议使用这些模板引擎，避免使用JSP，若一定要使用JSP将无法实现Spring Boot的多种特性**，具体可见后文：支持JSP的配置

当你使用上述模板引擎中的任何一个，它们默认的模板配置路径为：src/main/resources/templates。当然也可以修改这个路径，具体如何修改，可在后续各模板引擎的配置属性中查询并修改。

## [使用Freemarker模板引擎渲染web视图](http://blog.didispace.com/springbootweb/)

### pom文件引入:

|  |
| --- |
| <!-- 引入freeMarker的依赖包. -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency> |

### 3.4.2、后台代码

在src/main/resources/创建一个templates文件夹,后缀为\*.ftl

|  |
| --- |
| @RequestMapping("/index")  public String index(Map<String, Object> map) {  map.put("name","美丽的天使...");  return "index";  } |

### 3.4.3、前台代码

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <meta charset="UTF-8" />  <title></title>  </head>  <body>  ${name}  </body>  </html> |

### Freemarker其他用法

|  |
| --- |
| @RequestMapping("/index")  public String index(Map<String, Object> map) {  map.put("name","###蚂蚁课堂###");  map.put("sex",1);  List<String> userlist=new ArrayList<String>();  userlist.add("YaoSiyuan");  userlist.add("张三");  userlist.add("李四");  map.put("userlist",userlist);  return "index";  }  <!DOCTYPE html>  <html>  <head lang="en">  <meta charset="UTF-8" />  <title>首页</title>  </head>  <body>  ${name}  <#if sex==1>  男  <#elseif sex==2>  女  <#else>  其他    </#if>    <#list userlist as user>  ${user}  </#list>  </body>  </html> |

### 3.4.5、Freemarker配置

新建application.properties文件

|  |
| --- |
| ########################################################  ###FREEMARKER (FreeMarkerAutoConfiguration)  ########################################################  spring.freemarker.allow-request-override=false  spring.freemarker.cache=true  spring.freemarker.check-template-location=true  spring.freemarker.charset=UTF-8  spring.freemarker.content-type=text/html  spring.freemarker.expose-request-attributes=false  spring.freemarker.expose-session-attributes=false  spring.freemarker.expose-spring-macro-helpers=false  #spring.freemarker.prefix=  #spring.freemarker.request-context-attribute=  #spring.freemarker.settings.\*=  spring.freemarker.suffix=.ftl  spring.freemarker.template-loader-path=classpath:/templates/  #comma-separated list  #spring.freemarker.view-names= # whitelist of view names that can be resolved |

## Freemarker的使用

### 第一步：

|  |
| --- |
| <!-- 引入freeMarker的依赖包. -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency> |

### 第二步：

写个控制器，并返回些数据

|  |
| --- |
|  |

### 第三步：创建.ftl模板文件

* 在src/main/resources/创建一个templates文件夹,后缀为\*.ftl
* **掌握如何取值和判断**

|  |
| --- |
| <!DOCTYPE html>  <html>  <head lang="en">  <meta charset="UTF-8" />  <title></title>  </head>  <body>  欢迎${loginname}  <#if age <= 17>小哥  <#elseif age <= 30>先生  <#else>大叔  </#if>登录  <table border="1">  <tr>  <td>ID</td>  <td>名字</td>  <td>性别</td>  </tr>  <#list stuList?sort\_by("id")?reverse as stu>  <tr>  <td> ${stu.id}</td>  <td> ${stu.name}</td>  <td> ${stu.gender}</td>  </tr>  </#list>  </table>  </body>  </html> |

# 数据访问

### springboot整合[使用JdbcTemplate](http://blog.didispace.com/springbootdata1/)

#### pom文件引入

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.21</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

#### 4.1.2 application.properties新增配置

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/test  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

#### 4.1.3 UserService类

|  |
| --- |
| @Service  public class UserServiceImpl implements UserService {  @Autowired  private JdbcTemplate jdbcTemplate;  public void createUser(String name, Integer age) {  System.*out*.println("ssss");  jdbcTemplate.update("insert into users values(null,?,?);", name, age);  }  } |

#### 4.1.4 App类

|  |
| --- |
| @ComponentScan(basePackages = "com.ysy")  @EnableAutoConfiguration  public class App {  public static void main(String[] args) {  SpringApplication.*run*(App.class, args);  }  } |

**注意: spring-boot-starter-parent要在1.5以上**

### springboot整合[使用](http://blog.didispace.com/springbootdata1/)mybatis

#### 4.2.1、pom文件引入

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.21</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

#### 配置文件引入

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/test  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

#### Mapper代码

|  |
| --- |
| public interface UserMapper {  @Select("SELECT \* FROM USERS WHERE NAME = #{name}")  User findByName(@Param("name") String name);  @Insert("INSERT INTO USERS(NAME, AGE) VALUES(#{name}, #{age})")  int insert(@Param("name") String name, @Param("age") Integer age);  } |

实体类

|  |
| --- |
| package com.ysy;  /\*\*  \* @ClassName UserEntity  \* @Description TODO  \* @Author yaosiyuan  \* @Date 2019/3/8 21:35  \* @Version 1.0  \*\*/  public class UserEntity {  private Long id;  private String name;  private Integer age;  public Long getId() {  return id;  }  public void setId(Long id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  } |

#### 启动方式

|  |
| --- |
| @ComponentScan(basePackages = "com.ysy")  @MapperScan(basePackages = "com.ysy.mapper")  @SpringBootApplication  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

### springboot整合[使用](http://blog.didispace.com/springbootdata1/)springjpa

#### pom文件引入依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.4.2.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.21</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

#### 数据源

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/test  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

#### 创建User实体类

|  |
| --- |
| import javax.persistence.Entity;  @Entity(name = "users")  **public** **class** User {  @Id  @GeneratedValue  **private** Integer id;  @Column  **private** String name;  @Column  **private** Integer age;  // ..get/set方法  } |

#### 创建UserDao

|  |
| --- |
| **public** **interface** UserDao **extends** JpaRepository<User, Integer> {  // Integer为主键类型  } |

#### 创建IndexController

|  |
| --- |
| @RestController  public class IndexController {  @Autowired  private UserDao userDao;  @RequestMapping("/index")  public String index(Integer id) {  User findUser = userDao.findOne(id);  System.*out*.println(findUser.getName());  return "success";  }  } |

#### 启动项目

|  |
| --- |
| @ComponentScan(basePackages = { "com.ysy" })  @EnableJpaRepositories(basePackages = "com.ysy.dao")  @EnableAutoConfiguration  @~~EntityScan~~(~~basePackages~~ = "com.ysy.entity")  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

### springboot整合多数据源

同学们思考下，你们在项目中有使用到多数据源吗？

#### 配置文件中新增两个数据源

|  |
| --- |
| spring.datasource.test1.driverClassName = com.mysql.jdbc.Driver  spring.datasource.test1.url = jdbc:mysql://localhost:3306/test01?useUnicode=true&characterEncoding=utf-8  spring.datasource.test1.username = root  spring.datasource.test1.password = root  spring.datasource.test2.driverClassName = com.mysql.jdbc.Driver  spring.datasource.test2.url = jdbc:mysql://localhost:3306/test02?useUnicode=true&characterEncoding=utf-8  spring.datasource.test2.username = root  spring.datasource.test2.password = root |

#### 配置文件中新增两个数据源

|  |
| --- |
| @Configuration // 注册到springboot容器中  @MapperScan(basePackages = "com.ysy.user1", sqlSessionFactoryRef = "test1SqlSessionFactory")  public class DataSource1Config {  /\*\*  \*  \* @methodDesc: 功能描述:(配置test1数据库)  \* @author: YaoSiyuan  \* @param: @return  \* @createTime:2017年9月17日 下午3:16:44  \* @returnType:@return DataSource  \* @copyright:昭阳科技  \* @QQ:871664215  \*/  @Bean(name = "test1DataSource")  @Primary  @ConfigurationProperties(prefix = "spring.datasource.test1")  public DataSource testDataSource() {  return DataSourceBuilder.*create*().build();  }  /\*\*  \*  \* @methodDesc: 功能描述:(test1 sql会话工厂)  \* @author: YaoSiyuan  \* @param: @param  \* dataSource  \* @param: @return  \* @param: @throws  \* Exception  \* @createTime:2017年9月17日 下午3:17:08  \* @returnType:@param dataSource  \* @returnType:@return  \* @returnType:@throws Exception SqlSessionFactory  \* @copyright:昭阳科技  \* @QQ:871664215  \*/  @Bean(name = "test1SqlSessionFactory")  @Primary  public SqlSessionFactory testSqlSessionFactory(@Qualifier("test1DataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  // bean.setMapperLocations(  // new PathMatchingResourcePatternResolver().getResources("classpath:mybatis/mapper/test1/\*.xml"));  return bean.getObject();  }  /\*\*  \*  \* @methodDesc: 功能描述:(test1 事物管理)  \* @author: YaoSiyuan  \* @param: @param  \* dataSource  \* @param: @return  \* @param: @throws  \* Exception  \* @createTime:2017年9月17日 下午3:17:08  \* @returnType:@param dataSource  \* @returnType:@return  \* @returnType:@throws Exception SqlSessionFactory  \* @copyright:昭阳科技  \* @QQ:871664215  \*/  @Bean(name = "test1TransactionManager")  @Primary  public DataSourceTransactionManager testTransactionManager(@Qualifier("test1DataSource") DataSource dataSource) {  return new DataSourceTransactionManager(dataSource);  }  @Bean(name = "test1SqlSessionTemplate")  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test1SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |

#### 创建分包Mapper

|  |
| --- |
| **public** **interface** User1Mapper {  @Insert("insert into users values(null,#{name},#{age});")  **public** **int** addUser(@Param("name") String name, @Param("age") Integer age);  } |

#### 启动项目

|  |
| --- |
| @ComponentScan(basePackages = "com.ysy")  @EnableAutoConfiguration  public class App {  public static void main(String[] args) {  SpringApplication.*run*(App.class, args);  }  } |

No qualifying bean of type [javax.sql.DataSource] is defined: expected single matching bean but found 2: test1DataSource,test2DataSource

## 数据库访问

### SpringBoot使用JDBC

#### Step1:添加依赖

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

#### Step2:application.propert**i**es新增配置

|  |
| --- |
| #视图配置  spring.mvc.view.prefix=/WEB-INF/view/  spring.mvc.view.suffix=.jsp  #数据库配置  spring.datasource.url=jdbc:mysql://localhost:3306/day12  spring.datasource.username=root  spring.datasource.password=123456  spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

#### Step3:Service

|  |
| --- |
|  |

#### Step4:Controller

|  |
| --- |
|  |

#### Step5:App

|  |
| --- |
|  |

### 使用Mybatis

#### 创建一个maven-jar新项目来讲解

#### pom导入

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!-- mybaties -->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  <!-- mysql驱动 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

#### 数据库配置文件

|  |
| --- |
|  |

#### Mapper

|  |
| --- |
|  |
| 换成xml形式,将这个放入Maper包中 |
| *<?***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.gyf.mapper.UserMapper"** >   <**insert id="save"**>  insert into t\_user (username,password) VALUES(#{0},#{1})  </**insert**>  <**select id="findByUsername" resultType="com.gyf.model.User" parameterType="string"**>  select *\** from t\_user where username = #{username,jdbcType=VARCHAR}  </**select**> </**mapper**> |
| 注意这里需要在pom中添加下面代码  <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  <**include**>\*\*/\*.xml</**include**>  </**includes**>  </**resource**>  </**resources**> </**build**> |

#### Controller

|  |
| --- |
| **package** com.gyf.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.gyf.mapper.UserMapper;  **import** com.gyf.model.User;  @Controller  @RequestMapping("/user")  **public** **class** UserController {  @Autowired  **private** UserMapper userMapper;  @RequestMapping("/find")  @ResponseBody  **public** User find(String name){  **return** userMapper.findByName(name);  }    @ResponseBody  @RequestMapping("/add")  **public** **int** add(String name){  **return** userMapper.insert(name,"e10adc3949ba59abbe56e057f20f883e");  }  } |

#### App

|  |
| --- |
| package com.gyf.app;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.EnableAutoConfiguration;  import org.springframework.context.annotation.ComponentScan;  @ComponentScan(basePackages={"com.gyf.controller"})  **@MapperScan(basePackages={"com.gyf.mapper"})//扫描Mapper**  @EnableAutoConfiguration  public class App {  public static void main(String[] args) {  SpringApplication.run(App.class, args);  }  } |

### spring中使用的事务

只需要加个Transactional注解即可

|  |
| --- |
|  |

### 配置多数据源

以前是在applicationContext.xml中配置数据源

通过注解来配置数据源

|  |
| --- |
|  |

#### 配置文件中新增两个数据源

|  |
| --- |
| spring.datasource.test1.driverClassName=com.mysql.jdbc.Driver  spring.datasource.test1.url=jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncoding=utf-8  spring.datasource.test1.username=root  spring.datasource.test1.password=123456  spring.datasource.test2.driverClassName=com.mysql.jdbc.Driver  spring.datasource.test2.url=jdbc:mysql://localhost:3306/test2?useUnicode=true&characterEncoding=utf-8  spring.datasource.test2.username=root  spring.datasource.test2.password=123456 |
|  |
|  |
| use test1;  CREATE table user(  id int PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR(50),  password VARCHAR(50),  email VARCHAR(50),  birthday TIMESTAMP  );  use test2;  CREATE table customer(  id int PRIMARY KEY AUTO\_INCREMENT,  name VARCHAR(50),  tel VARCHAR(50)  ); |

#### 写两个数据源配置

|  |
| --- |
| package com.gyf.datasource;  @Configuration//注解到springboot容器中  @MapperScan(basePackages="com.gyf.test1.mapper",sqlSessionFactoryRef="test1SqlSessionFactory")  public class DataSource01 {  /\*\*  \* @return 返回test1数据库的数据源  \*/  @Bean(name="test1DataSource")  @Primary//主数据源  @ConfigurationProperties(prefix="spring.datasource.test1")  public DataSource dateSource(){  return DataSourceBuilder.create().build();  }    /\*\*  \* @return 返回test1数据库的会话工厂  \*/  @Bean(name = "test1SqlSessionFactory")  public SqlSessionFactory sqlSessionFactory(@Qualifier("test1DataSource") DataSource ds) throws Exception{  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(ds);    return bean.getObject();  }    /\*\*  \* @return 返回test1数据库的事务  \*/  @Bean(name = "test1TransactionManager")  @Primary  public DataSourceTransactionManager transactionManager(@Qualifier("test1DataSource") DataSource dataSource) {  return new DataSourceTransactionManager(dataSource);  }  /\*\*  \* @return 返回test1数据库的会话模版  \*/  @Bean(name = "test1SqlSessionTemplate")  public SqlSessionTemplate sqlSessionTemplate(  @Qualifier("test1SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |
| package com.gyf.datasource;  @Configuration//注解到springboot容器中  @MapperScan(basePackages="com.gyf.test2.mapper",sqlSessionFactoryRef="test2SqlSessionFactory")  public class DataSource02 {  /\*\*  \* @return 返回test2数据库的数据源  \*/  @Bean(name="test2DataSource")  @ConfigurationProperties(prefix="spring.datasource.test2")  public DataSource dateSource(){  return DataSourceBuilder.create().build();  }    /\*\*  \* @return 返回test2数据库的会话工厂  \*/  @Bean(name = "test2SqlSessionFactory")  public SqlSessionFactory sqlSessionFactory(@Qualifier("test2DataSource") DataSource ds) throws Exception{  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(ds);    return bean.getObject();  }    /\*\*  \* @return 返回test2数据库的事务  \*/  @Bean(name = "test2TransactionManager")  public DataSourceTransactionManager transactionManager(@Qualifier("test2DataSource") DataSource dataSource) {  return new DataSourceTransactionManager(dataSource);  }  /\*\*  \* @return 返回test2数据库的会话模版  \*/  @Bean(name = "test2SqlSessionTemplate")  public SqlSessionTemplate sqlSessionTemplate(  @Qualifier("test2SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |

#### test1 Mapper&service

|  |
| --- |
|  |
|  |

#### test2 mapper & service

|  |
| --- |
|  |
|  |

#### Controller

|  |
| --- |
|  |

#### App

|  |
| --- |
|  |

### 多数据源问题

|  |
| --- |
|  |
|  |
| 最后一个可以插入一条数据  原是是Transactoinal的事务只针对userMapper有效 |

### springboot中的多事务管理

* 使用springboot+jta+atomikos 分布式事物管理解决方案

#### 添加jta事务依赖

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

#### 修改数据库连接配置数据

|  |
| --- |
| # Mysql 1  mysql.datasource.test1.url = jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncoding=utf-8  mysql.datasource.test1.username = root  mysql.datasource.test1.password = 123456  mysql.datasource.test1.minPoolSize = 3  mysql.datasource.test1.maxPoolSize = 25  mysql.datasource.test1.maxLifetime = 20000  mysql.datasource.test1.borrowConnectionTimeout = 30  mysql.datasource.test1.loginTimeout = 30  mysql.datasource.test1.maintenanceInterval = 60  mysql.datasource.test1.maxIdleTime = 60  mysql.datasource.test1.testQuery = select 1  # Mysql 2  mysql.datasource.test2.url =jdbc:mysql://localhost:3306/test2?useUnicode=true&characterEncoding=utf-8  mysql.datasource.test2.username =root  mysql.datasource.test2.password =123456  mysql.datasource.test2.minPoolSize = 3  mysql.datasource.test2.maxPoolSize = 25  mysql.datasource.test2.maxLifetime = 20000  mysql.datasource.test2.borrowConnectionTimeout = 30  mysql.datasource.test2.loginTimeout = 30  mysql.datasource.test2.maintenanceInterval = 60  mysql.datasource.test2.maxIdleTime = 60  mysql.datasource.test2.testQuery = select 1 |

#### 添加2个配置模型

* **模型放在com.gyf.dbconfig包**

|  |
| --- |
| @ConfigurationProperties("mysql.datasource.test1")  **public** **class** DBConfig1 {  **private** String url;  **private** String username;  **private** String password;  **private** **int** minPoolSize;  **private** **int** maxPoolSize;  **private** **int** maxLifetime;  **private** **int** borrowConnectionTimeout;  **private** **int** loginTimeout;  **private** **int** maintenanceInterval;  **private** **int** maxIdleTime;  **private** String testQuery; |
| @ConfigurationProperties("mysql.datasource.test2")  **public** **class** DBConfig2 {  **private** String url;  **private** String username;  **private** String password;  **private** **int** minPoolSize;  **private** **int** maxPoolSize;  **private** **int** maxLifetime;  **private** **int** borrowConnectionTimeout;  **private** **int** loginTimeout;  **private** **int** maintenanceInterval;  **private** **int** maxIdleTime;  **private** String testQuery; |

#### 重定两个数据源配置

|  |
| --- |
| package com.gyf.datasource;  import java.sql.SQLException;  import javax.sql.DataSource;  import org.apache.ibatis.session.SqlSessionFactory;  import org.mybatis.spring.SqlSessionFactoryBean;  import org.mybatis.spring.SqlSessionTemplate;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Primary;  import org.springframework.jdbc.datasource.DataSourceTransactionManager;  import com.atomikos.jdbc.AtomikosDataSourceBean;  import com.gyf.dbconfig.DBConfig1;  import com.mysql.jdbc.jdbc2.optional.MysqlXADataSource;  @Configuration//注解到springboot容器中  @MapperScan(basePackages="com.gyf.test1.mapper",sqlSessionFactoryRef="test1SqlSessionFactory")  public class DataSource01 {  // 配置数据源  @Primary  @Bean(name = "test1DataSource")  public DataSource testDataSource(DBConfig1 testConfig) throws SQLException {  MysqlXADataSource mysqlXaDataSource = new MysqlXADataSource();  mysqlXaDataSource.setUrl(testConfig.getUrl());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  mysqlXaDataSource.setPassword(testConfig.getPassword());  mysqlXaDataSource.setUser(testConfig.getUsername());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  AtomikosDataSourceBean xaDataSource = new AtomikosDataSourceBean();  xaDataSource.setXaDataSource(mysqlXaDataSource);  xaDataSource.setUniqueResourceName("test1DataSource");  xaDataSource.setMinPoolSize(testConfig.getMinPoolSize());  xaDataSource.setMaxPoolSize(testConfig.getMaxPoolSize());  xaDataSource.setMaxLifetime(testConfig.getMaxLifetime());  xaDataSource.setBorrowConnectionTimeout(testConfig.getBorrowConnectionTimeout());  xaDataSource.setLoginTimeout(testConfig.getLoginTimeout());  xaDataSource.setMaintenanceInterval(testConfig.getMaintenanceInterval());  xaDataSource.setMaxIdleTime(testConfig.getMaxIdleTime());  xaDataSource.setTestQuery(testConfig.getTestQuery());  return xaDataSource;  }  @Bean(name = "test1SqlSessionFactory")  public SqlSessionFactory testSqlSessionFactory(@Qualifier("test1DataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  return bean.getObject();  }  @Bean(name = "test1SqlSessionTemplate")  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test1SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |
| **package** com.gyf.datasource;  **import** java.sql.SQLException;  **import** javax.sql.DataSource;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.mybatis.spring.SqlSessionFactoryBean;  **import** org.mybatis.spring.SqlSessionTemplate;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** com.atomikos.jdbc.AtomikosDataSourceBean;  **import** com.gyf.dbconfig.DBConfig2;  **import** com.mysql.jdbc.jdbc2.optional.MysqlXADataSource;  @Configuration//注解到springboot容器中  @MapperScan(basePackages="com.gyf.test2.mapper",sqlSessionFactoryRef="test2SqlSessionFactory")  **public** **class** DataSource02 {  // 配置数据源  @Bean(name = "test2DataSource")  **public** DataSource testDataSource(DBConfig2 testConfig) **throws** SQLException {  MysqlXADataSource mysqlXaDataSource = **new** MysqlXADataSource();  mysqlXaDataSource.setUrl(testConfig.getUrl());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(**true**);  mysqlXaDataSource.setPassword(testConfig.getPassword());  mysqlXaDataSource.setUser(testConfig.getUsername());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(**true**);  AtomikosDataSourceBean xaDataSource = **new** AtomikosDataSourceBean();  xaDataSource.setXaDataSource(mysqlXaDataSource);  xaDataSource.setUniqueResourceName("test2DataSource");  xaDataSource.setMinPoolSize(testConfig.getMinPoolSize());  xaDataSource.setMaxPoolSize(testConfig.getMaxPoolSize());  xaDataSource.setMaxLifetime(testConfig.getMaxLifetime());  xaDataSource.setBorrowConnectionTimeout(testConfig.getBorrowConnectionTimeout());  xaDataSource.setLoginTimeout(testConfig.getLoginTimeout());  xaDataSource.setMaintenanceInterval(testConfig.getMaintenanceInterval());  xaDataSource.setMaxIdleTime(testConfig.getMaxIdleTime());  xaDataSource.setTestQuery(testConfig.getTestQuery());  **return** xaDataSource;  }  @Bean(name = "test2SqlSessionFactory")  **public** SqlSessionFactory testSqlSessionFactory(@Qualifier("test2DataSource") DataSource dataSource)  **throws** Exception {  SqlSessionFactoryBean bean = **new** SqlSessionFactoryBean();  bean.setDataSource(dataSource);  **return** bean.getObject();  }  @Bean(name = "test2SqlSessionTemplate")  **public** SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test2SqlSessionFactory") SqlSessionFactory sqlSessionFactory) **throws** Exception {  **return** **new** SqlSessionTemplate(sqlSessionFactory);  }  } |

#### App

|  |
| --- |
|  |

# 事物管理

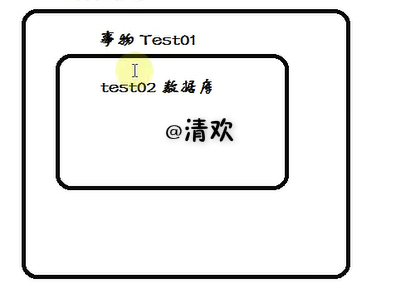
### springboot整合事物管理

springboot默认集成事物,只主要在方法上加上@Transactional即可

|  |
| --- |
| package com.ysy.service;  import com.ysy.mapper.UserMapper;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  /\*\*  \* @ClassName UserService  \* @Description TODO  \* @Author yaosiyuan  \* @Date 2019/3/9 13:24  \* @Version 1.0  \*\*/  @Service  public class UserService {  @Autowired  private UserMapper userMapper;  @Transactional  public int addUser(String name, Integer age) {  int insertUser = userMapper.insertUser(name, age);  int i =1/0;  return insertUser;  }  } |

### SpringBoot分布式事物管理

多个数据源只配置了一个数据库，导致没法对另外一个做回滚，



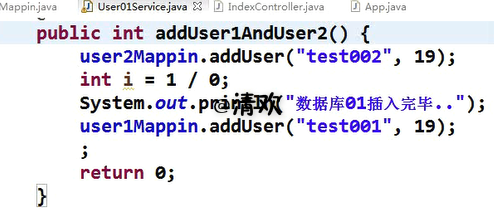
可用方案：

1. jta+atomikos
2. 两端提交协议
3. mq推送

使用springboot+jta+atomikos分布式事物管理



结果插入0条数据



因为只配置了数据库一，所以结果插入1条数据库，只有数据库1回滚了，

#### 新增配置文件信息

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

#### 新增配置文件信息

|  |
| --- |
| # Mysql 1  mysql.datasource.test.url = jdbc:mysql://localhost:3306/test01?useUnicode=true&characterEncoding=utf-8  mysql.datasource.test.username = root  mysql.datasource.test.password = root  mysql.datasource.test.minPoolSize = 3  mysql.datasource.test.maxPoolSize = 25  mysql.datasource.test.maxLifetime = 20000  mysql.datasource.test.borrowConnectionTimeout = 30  mysql.datasource.test.loginTimeout = 30  mysql.datasource.test.maintenanceInterval = 60  mysql.datasource.test.maxIdleTime = 60  mysql.datasource.test.testQuery = select 1  # Mysql 2  mysql.datasource.test2.url =jdbc:mysql://localhost:3306/test02?useUnicode=true&characterEncoding=utf-8  mysql.datasource.test2.username =root  mysql.datasource.test2.password =root  mysql.datasource.test2.minPoolSize = 3  mysql.datasource.test2.maxPoolSize = 25  mysql.datasource.test2.maxLifetime = 20000  mysql.datasource.test2.borrowConnectionTimeout = 30  mysql.datasource.test2.loginTimeout = 30  mysql.datasource.test2.maintenanceInterval = 60  mysql.datasource.test2.maxIdleTime = 60  mysql.datasource.test2.testQuery = select 1 |

#### 读取配置文件信息

|  |
| --- |
| package com.ysy.config;  import org.springframework.boot.context.properties.ConfigurationProperties;  @ConfigurationProperties(prefix = "mysql.datasource.test")  public class DBConfig1 {  private String url;  private String username;  private String password;  private int minPoolSize;  private int maxPoolSize;  private int maxLifetime;  private int borrowConnectionTimeout;  private int loginTimeout;  private int maintenanceInterval;  private int maxIdleTime;  private String testQuery;  getandeset  。getandeset  。getandeset  } |
| package com.ysy.config;  import org.springframework.boot.context.properties.ConfigurationProperties;  @ConfigurationProperties(prefix = "mysql.datasource.test1")  public class DBConfig2 {  private String url;  private String username;  private String password;  private int minPoolSize;  private int maxPoolSize;  private int maxLifetime;  private int borrowConnectionTimeout;  private int loginTimeout;  private int maintenanceInterval;  private int maxIdleTime;  private String testQuery;  } |

#### 创建多数据源

|  |
| --- |
| @Configuration  // basePackages 最好分开配置 如果放在同一个文件夹可能会报错  @MapperScan(basePackages = "com.ysy.test01", sqlSessionTemplateRef = "testSqlSessionTemplate")  public class TestMyBatisConfig1 {  // 配置数据源  @Primary  @Bean(name = "testDataSource")  public DataSource testDataSource(DBConfig1 testConfig) throws SQLException {  MysqlXADataSource mysqlXaDataSource = new MysqlXADataSource();  mysqlXaDataSource.setUrl(testConfig.getUrl());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  mysqlXaDataSource.setPassword(testConfig.getPassword());  mysqlXaDataSource.setUser(testConfig.getUsername());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  AtomikosDataSourceBean xaDataSource = new AtomikosDataSourceBean();  xaDataSource.setXaDataSource(mysqlXaDataSource);  xaDataSource.setUniqueResourceName("testDataSource");  xaDataSource.setMinPoolSize(testConfig.getMinPoolSize());  xaDataSource.setMaxPoolSize(testConfig.getMaxPoolSize());  xaDataSource.setMaxLifetime(testConfig.getMaxLifetime());  xaDataSource.setBorrowConnectionTimeout(testConfig.getBorrowConnectionTimeout());  xaDataSource.setLoginTimeout(testConfig.getLoginTimeout());  xaDataSource.setMaintenanceInterval(testConfig.getMaintenanceInterval());  xaDataSource.setMaxIdleTime(testConfig.getMaxIdleTime());  xaDataSource.setTestQuery(testConfig.getTestQuery());  return xaDataSource;  }  @Bean(name = "testSqlSessionFactory")  public SqlSessionFactory testSqlSessionFactory(@Qualifier("testDataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  return bean.getObject();  }  @Bean(name = "testSqlSessionTemplate")  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("testSqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |
| package com.ysy.datasource;  import java.sql.SQLException;  import javax.sql.DataSource;  import org.apache.ibatis.session.SqlSessionFactory;  import org.mybatis.spring.SqlSessionFactoryBean;  import org.mybatis.spring.SqlSessionTemplate;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import com.atomikos.jdbc.AtomikosDataSourceBean;  import com.ysy.config.DBConfig1;  import com.mysql.jdbc.jdbc2.optional.MysqlXADataSource;  @Configuration  // basePackages 最好分开配置 如果放在同一个文件夹可能会报错  @MapperScan(basePackages = "com.ysy.test02", sqlSessionTemplateRef = "test2SqlSessionTemplate")  public class TestMyBatisConfig2 {  // 配置数据源  @Bean(name = "test2DataSource")  public DataSource testDataSource(DBConfig1 testConfig) throws SQLException {  MysqlXADataSource mysqlXaDataSource = new MysqlXADataSource();  mysqlXaDataSource.setUrl(testConfig.getUrl());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  mysqlXaDataSource.setPassword(testConfig.getPassword());  mysqlXaDataSource.setUser(testConfig.getUsername());  mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);  AtomikosDataSourceBean xaDataSource = new AtomikosDataSourceBean();  xaDataSource.setXaDataSource(mysqlXaDataSource);  xaDataSource.setUniqueResourceName("test2DataSource");  xaDataSource.setMinPoolSize(testConfig.getMinPoolSize());  xaDataSource.setMaxPoolSize(testConfig.getMaxPoolSize());  xaDataSource.setMaxLifetime(testConfig.getMaxLifetime());  xaDataSource.setBorrowConnectionTimeout(testConfig.getBorrowConnectionTimeout());  xaDataSource.setLoginTimeout(testConfig.getLoginTimeout());  xaDataSource.setMaintenanceInterval(testConfig.getMaintenanceInterval());  xaDataSource.setMaxIdleTime(testConfig.getMaxIdleTime());  xaDataSource.setTestQuery(testConfig.getTestQuery());  return xaDataSource;  }  @Bean(name = "test2SqlSessionFactory")  public SqlSessionFactory testSqlSessionFactory(@Qualifier("test2DataSource") DataSource dataSource)  throws Exception {  SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(dataSource);  return bean.getObject();  }  @Bean(name = "test2SqlSessionTemplate")  public SqlSessionTemplate testSqlSessionTemplate(  @Qualifier("test2SqlSessionFactory") SqlSessionFactory sqlSessionFactory) throws Exception {  return new SqlSessionTemplate(sqlSessionFactory);  }  } |

#### 启动加载配置

@EnableConfigurationProperties(value = { DBConfig1.class, DBConfig2.class })

# 日志管理

## 整合Log4j

### 7.1 导入Log4j属性文件

#### log4j.properties

注意，复制时要把每一行后面的空格去除

|  |
| --- |
| log4j.rootLogger=INFO,Console,File  log4j.appender.Console=org.apache.log4j.ConsoleAppender  log4j.appender.Console.Target=System.out  log4j.appender.Console.layout = org.apache.log4j.PatternLayout  log4j.appender.Console.layout.ConversionPattern=[%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c - %L]%m%n    log4j.appender.File = org.apache.log4j.RollingFileAppender  log4j.appender.File.File = C:/Users/10301/Desktop/test/logs/info/info.log  log4j.appender.File.MaxFileSize = 10MB    log4j.appender.File.Threshold = ALL  log4j.appender.File.layout = org.apache.log4j.PatternLayout  log4j.appender.File.layout.ConversionPattern =[%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c - %L]%m%n |

### 7.2 pom.xml

* 去除springboot的logging，添加log4j，因为自带的logging不启效果
* springboot下的Log4j的版本最新1.3.8，如果你的springboot的parent版本过高，那在在添加log4j自己版本

|  |
| --- |
|  |
|  |
|  |

### 7.3 测试

|  |
| --- |
|  |

## [使用log4j记录日志](http://blog.didispace.com/springbootlog4j/)

### 新建log4j配置文件

|  |
| --- |
| #log4j.rootLogger=CONSOLE,info,error,DEBUG  log4j.rootLogger=info,error,CONSOLE,DEBUG  log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  log4j.appender.CONSOLE.layout.ConversionPattern=%d{yyyy-MM-dd-HH-mm} [%t] [%c] [%p] - %m%n  log4j.logger.info=info  log4j.appender.info=org.apache.log4j.DailyRollingFileAppender  log4j.appender.info.layout=org.apache.log4j.PatternLayout  log4j.appender.info.layout.ConversionPattern=%d{yyyy-MM-dd-HH-mm} [%t] [%c] [%p] - %m%n  log4j.appender.info.datePattern='.'yyyy-MM-dd  log4j.appender.info.Threshold = info  log4j.appender.info.append=true  #log4j.appender.info.File=/home/admin/pms-api-services/logs/info/api\_services\_info  log4j.appender.info.File=/Users/dddd/Documents/testspace/pms-api-services/logs/info/api\_services\_info  log4j.logger.error=error  log4j.appender.error=org.apache.log4j.DailyRollingFileAppender  log4j.appender.error.layout=org.apache.log4j.PatternLayout  log4j.appender.error.layout.ConversionPattern=%d{yyyy-MM-dd-HH-mm} [%t] [%c] [%p] - %m%n  log4j.appender.error.datePattern='.'yyyy-MM-dd  log4j.appender.error.Threshold = error  log4j.appender.error.append=true  #log4j.appender.error.File=/home/admin/pms-api-services/logs/error/api\_services\_error  log4j.appender.error.File=/Users/dddd/Documents/testspace/pms-api-services/logs/error/api\_services\_error  log4j.logger.DEBUG=DEBUG  log4j.appender.DEBUG=org.apache.log4j.DailyRollingFileAppender  log4j.appender.DEBUG.layout=org.apache.log4j.PatternLayout  log4j.appender.DEBUG.layout.ConversionPattern=%d{yyyy-MM-dd-HH-mm} [%t] [%c] [%p] - %m%n  log4j.appender.DEBUG.datePattern='.'yyyy-MM-dd  log4j.appender.DEBUG.Threshold = DEBUG  log4j.appender.DEBUG.append=true  #log4j.appender.DEBUG.File=/home/admin/pms-api-services/logs/debug/api\_services\_debug  log4j.appender.DEBUG.File=/Users/dddd/Documents/testspace/pms-api-services/logs/debug/api\_services\_debug |

## 6.2[使用AOP统一处理Web请求日志](http://blog.didispace.com/springbootaoplog/)

### 6.2.1POM文件新增依赖

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

### 6.2.2POM文件新增依赖

|  |
| --- |
| @Aspect  @Component  public class WebLogAspect {  private Logger logger = LoggerFactory.*getLogger*(getClass());  @Pointcut("execution(public \* com.ysy.controller..\*.\*(..))")  public void webLog() {  }  @Before("webLog()")  public void doBefore(JoinPoint joinPoint) throws Throwable {  // 接收到请求，记录请求内容  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = attributes.getRequest();  // 记录下请求内容  logger.info("URL : " + request.getRequestURL().toString());  logger.info("HTTP\_METHOD : " + request.getMethod());  logger.info("IP : " + request.getRemoteAddr());  Enumeration<String> enu = request.getParameterNames();  while (enu.hasMoreElements()) {  String name = (String) enu.nextElement();  logger.info("name:{},value:{}", name, request.getParameter(name));  }  }  @AfterReturning(returning = "ret", pointcut = "webLog()")  public void doAfterReturning(Object ret) throws Throwable {  // 处理完请求，返回内容  logger.info("RESPONSE : " + ret);  }  } |

# 使用AOP统一处理Web请求日志

## Step01

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

## Step02:写个切面类

|  |
| --- |
| **package** com.gyf.aop;  **import** java.util.Enumeration;  **import** javax.servlet.http.HttpServletRequest;  **import** org.apache.log4j.Logger;  **import** org.aspectj.lang.JoinPoint;  **import** org.aspectj.lang.annotation.AfterReturning;  **import** org.aspectj.lang.annotation.Aspect;  **import** org.aspectj.lang.annotation.Before;  **import** org.aspectj.lang.annotation.Pointcut;  **import** org.springframework.stereotype.Component;  **import** org.springframework.web.context.request.RequestContextHolder;  **import** org.springframework.web.context.request.ServletRequestAttributes;  @Aspect  @Component  **public** **class** WebLogAspect {  **private** Logger logger = Logger.*getLogger*(getClass());    @Pointcut("execution(public \* com.gyf.controller..\*.\*(..))")  **public** **void** webLog() {    }    @Before("webLog()")  **public** **void** doBefore(JoinPoint joinPoint) **throws** Throwable {  // 接收到请求，记录请求内容  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = attributes.getRequest();  // 记录下请求内容  logger.info("---------------request----------------");  logger.info("URL : " + request.getRequestURL().toString());  logger.info("HTTP\_METHOD : " + request.getMethod());  logger.info("IP : " + request.getRemoteAddr());  Enumeration<String> enu = request.getParameterNames();  **while** (enu.hasMoreElements()) {  String name = (String) enu.nextElement();  logger.info("name:" + name + "value" + request.getParameter(name));  }  }  @AfterReturning(returning = "ret", pointcut = "webLog()")  **public** **void** doAfterReturning(Object ret) **throws** Throwable {  logger.info("---------------response----------------");  // 处理完请求，返回内容  logger.info("RESPONSE : " + ret);  }  } |

## Step3:App配置注解

|  |
| --- |
|  |

# 缓存支持

## [注解配置与EhCache使用](http://blog.didispace.com/springbootcache1/)

### pom文件引入

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

### 新建ehcache.xml 文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ehcache.org/ehcache.xsd"*  updateCheck=*"false"*>  <diskStore path=*"java.io.tmpdir/Tmp\_EhCache"* />  <!-- 默认配置 -->  <defaultCache maxElementsInMemory=*"5000"* eternal=*"false"*  timeToIdleSeconds=*"120"* timeToLiveSeconds=*"120"*  memoryStoreEvictionPolicy=*"LRU"* overflowToDisk=*"false"* />  <cache name=*"baseCache"* maxElementsInMemory=*"10000"*  maxElementsOnDisk=*"100000"* />  </ehcache> |

**配置信息介绍**

1. <!--
2. name:缓存名称。
3. maxElementsInMemory：缓存最大个数。
4. eternal:对象是否永久有效，一但设置了，timeout将不起作用。
5. timeToIdleSeconds：设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=**false**对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。
6. timeToLiveSeconds：设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=**false**对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。
7. overflowToDisk：当内存中对象数量达到maxElementsInMemory时，Ehcache将会对象写到磁盘中。
8. diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。
9. maxElementsOnDisk：硬盘最大缓存个数。
10. diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The **default** value is **false**.
11. diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。
12. memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。
13. clearOnFlush：内存数量最大时是否清除。
14. -->

### 代码使用Cacheable

|  |
| --- |
| @CacheConfig(cacheNames = "baseCache")  public interface UserMapper {  @Select("select \* from users where name=#{name}")  @Cacheable  UserEntity findName(@Param("name") String name);  } |

### 清除缓存

|  |
| --- |
| @Autowired  private CacheManager cacheManager;  @RequestMapping("/remoKey")  public void remoKey() {  cacheManager.getCache("baseCache").clear();  } |

### [使用Redis做集中式缓存](http://blog.didispace.com/springbootcache2/)

# 其他内容

## [使用@Scheduled创建定时任务](http://blog.didispace.com/springbootscheduled/)

在Spring Boot的主类中加入@EnableScheduling注解，启用定时任务的配置

|  |
| --- |
| @Component  public class ScheduledTasks {  private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");  @Scheduled(fixedRate = 5000)  public void reportCurrentTime() {  System.out.println("现在时间：" + dateFormat.format(new Date()));  }  } |

## [使用@Async实现异步调用](http://blog.didispace.com/springbootasync/)

启动加上@EnableAsync ,需要执行异步方法上加入 @Async

## 自定义参数

配置文件值

|  |
| --- |
| name=ysy.com |

配置文件值

|  |
| --- |
| @Value("${name}")  private String name;  @ResponseBody  @RequestMapping("/getValue")  public String getValue() {  return name;  } |

## 多环境配置

|  |
| --- |
| spring.profiles.active=pre |

|  |
| --- |
| application-dev.properties：开发环境  application-test.properties：测试环境  application-prod.properties：生产环境 |

## 修改端口号

server.port=8888

server.context-path=/ysy

## SpringBoot yml 使用

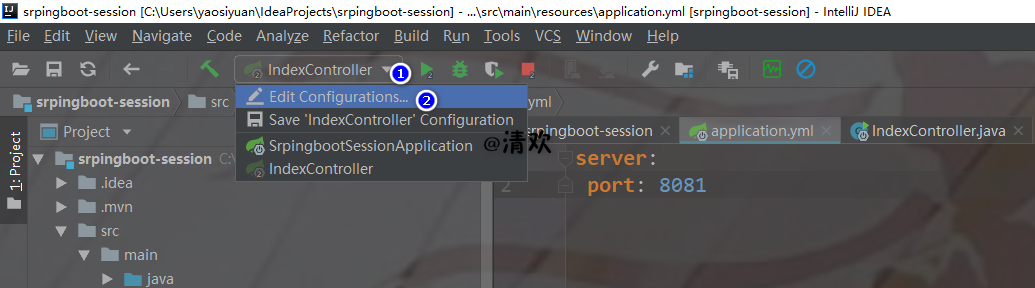
创建application.yml

|  |
| --- |
| server:  port: 8090  context-path: /ysy |

## idea启动多个springboot

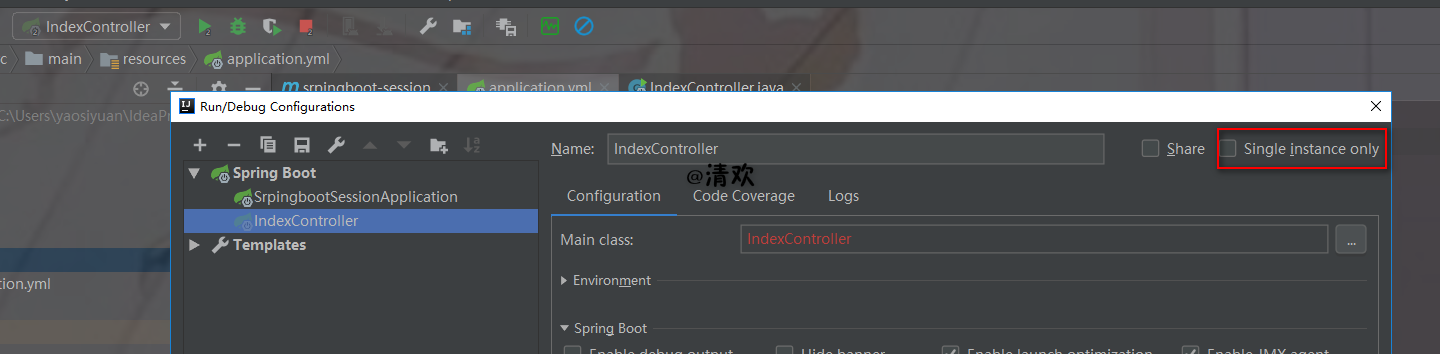
### step 1

在IDEA上点击Application右边的下三角  
,弹出选项后，点击Edit Configuration



### step 2

打开配置后，将默认的Single instance only(单实例)的钩去掉。



### step 3

通过修改application文件的server.port的端口，启动。多个实例，需要多个端口，分别启动。

## 发布打包

使用mvn package 打包

使用java –jar 包名

如果报错没有主清单,在pom文件中新增

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <maimClass>com.ysy.app.App</maimClass>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |