SpringBoot 学习笔记

1. 使用netstat -ano | findstr 8080 查看8080端口被那个线程占用
2. 使用taskkill -pid 3168 -f 杀掉线程
3. 打开idea 对maven 进行配置 配置成自己的本地maven
4. 创建一个maven 项目 在pom.xml 中添加依赖或者直接查看官网

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
</parent>  
  
  
<!-- Add typical dependencies for a web application -->  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
</dependencies>  
  
<!-- Package as an executable jar -->  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

1. 编写一个main 函数作为srpingBoot 的入口

// 这是springBoot的主启动类  
@SpringBootApplication  
public class ApplicationMain {  
 public static void main(String[] args){  
  
 // 这个函数调用才是springBoot 项目的真正入口  
 SpringApplication.*run*(ApplicationMain.class, args);  
 }  
}

springBoot 支持properties 文件和yml 文件格式

Yml 文件的值必须有一个空格 map 形式 port: 80

数组有server:

-port

-323 或者 server: [port, 3]

**server**:  
 **port**: 8081  
 **servlet**:  
 **context-path**: /aa

将bean文件与yml 配置文件之间联调通

使用注解

@Component  
@ConfigurationProperties(prefix = "user")

其中prefix 指定前缀去识别 此时窗口上方有提示 然后点击进去添加一个依赖文件

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
 <optional>true</optional>  
</dependency>

当我们在yml 配置文件中输入的时候会给提示 但是要build -> Rebuild Project 文件一下才生效

案例:

编写一个userBean

package cn.ybzy.sbtdemo2.model;  
  
import org.springframework.boot.context.properties.ConfigurationProperties;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
import java.util.List;  
import java.util.Map;  
  
// 将bean 注入到springIOC容器中  
@Component  
// 这个注解将userBean 与yml 配置文件中的数据进行关联  
@ConfigurationProperties(prefix = "user")  
public class User {  
 private int id;  
  
 private String username;  
  
 private String password;  
  
 private Date birthday;  
  
 private List<String> list;  
  
 private Map<String, String> map;  
  
 private Address address;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public Date getBirthday() {  
 return birthday;  
 }  
  
 public void setBirthday(Date birthday) {  
 this.birthday = birthday;  
 }  
  
 public List<String> getList() {  
 return list;  
 }  
  
 public void setList(List<String> list) {  
 this.list = list;  
 }  
  
 public Map<String, String> getMap() {  
 return map;  
 }  
  
 public void setMap(Map<String, String> map) {  
 this.map = map;  
 }  
  
 public Address getAddress() {  
 return address;  
 }  
  
 public void setAddress(Address address) {  
 this.address = address;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 ", birthday=" + birthday +  
 ", list=" + list +  
 ", map=" + map +  
 ", address=" + address +  
 '}';  
 }  
}

编写一个Address 类

package cn.ybzy.sbtdemo2.model;  
  
import org.springframework.stereotype.Component;  
  
@Component  
public class Address {  
 private int id;  
  
 private String detail;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getDetail() {  
 return detail;  
 }  
  
 public void setDetail(String detail) {  
 this.detail = detail;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "id=" + id +  
 ", detail='" + detail + '\'' +  
 '}';  
 }  
}

此时的yml 配置文件如下

**user**:  
 **id**: 111  
 **username**: 张三  
 **password**: 32424  
 **birthday**: 2012/04/23  
 **list**: [aaa,bbb,ccc]  
 **map**: {**aaa**: 43535, **bbbb**: fdkfjd}  
 **address**:  
 **id**: 42432  
 **detail**: 上海市闵行区

@Value 注解

支持三种形式：

1. 运算符形式

@Value("#{11 + 12}")  
private int id;

1. 读取配置文件属性

@Value("${user.username}")  
private String username;

1. 常量字面量

@Value("true")  
private boolean sex;

ConfigurationProperties 与@Value 区别

@ConfiguratioProperties 支持JSR 303校验

@Validated  
public class User {  
 // @Value("#{11 + 12}")  
 private int id;  
  
 // @Value("${user.username}")  
 @NotEmpty  
 private String username;

@ConfiguratioProperties 支持 List,Map 等复杂的数据结构形式

使用建议:

当对整个配置文件进行配置的时候直接用@ConfiguratioProperties

当对单个值进行赋值的时候直接用@Value

@PropertySource

这个注解可以从非全局的配置文件中读取属性

// 这个注解将userBean 与yml 配置文件中的数据进行关联  
 @ConfigurationProperties(prefix = "user")  
 // @Validated  
@PropertySource(value={"classpath:user.properties"})

@ImportResource

这个配置文件可以导入配置文件, 在spring的web.xml 配置文件中引入的额外的配置文件

示例：

必须在入口函数中引入

@SpringBootApplication  
// 可以导入配置文件, 在spring的web.xml 配置文件中引入的额外的配置文件  
@ImportResource("classpath:spring.xml")  
public class Demo2Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Demo2Application.class, args);  
 }  
  
}

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 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">  
  
 <bean id="address" class="cn.ybzy.sbtdemo2.model.Address"></bean>  
</beans>

但是springBoot并不希望使用者这样做，于是就引入了配置类的概念

package cn.ybzy.sbtdemo2.config;  
  
import cn.ybzy.sbtdemo2.model.Address;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration // 标明这是配置类  
public class MyConfig {  
 @Bean // 注入bean  
 public Address address(){  
 return new Address();  
 }  
}

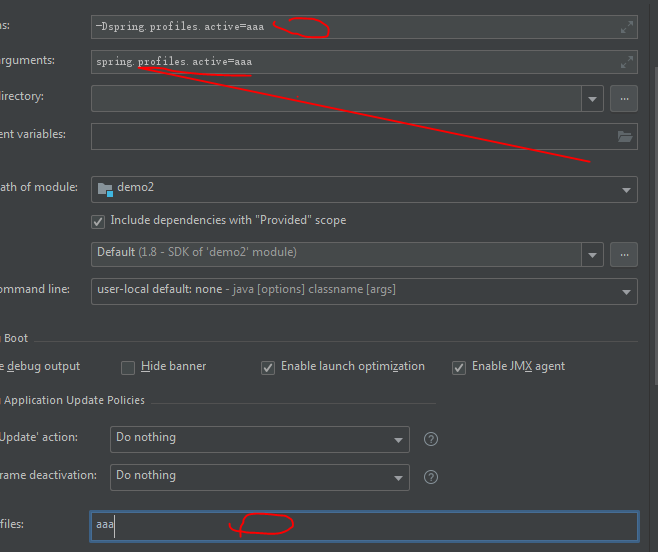
Properties 配置文件支持${} 用random 来获取随机值 或者引用变量的形式

user.id=${random.int} 随机整数  
user.username=${random.value} 随机值 和uuid的区别为这个值没有-

符号来隔开  
user.password=${random.uuid}  
user.birthday=2012/04/23  
user.list=aaa,bbb,ccc  
user.map.aaa=${user.username} 变量引用  
user.map.ggg=${user.usernames:'not found'} 引用变量，如果没有就给一个默认值  
user.address.id=${random.int(10)}  
user.address.detail=543543543

Spring.profiles用法： 表示激活哪些配置文件

第一种直接在编辑器配置中激活



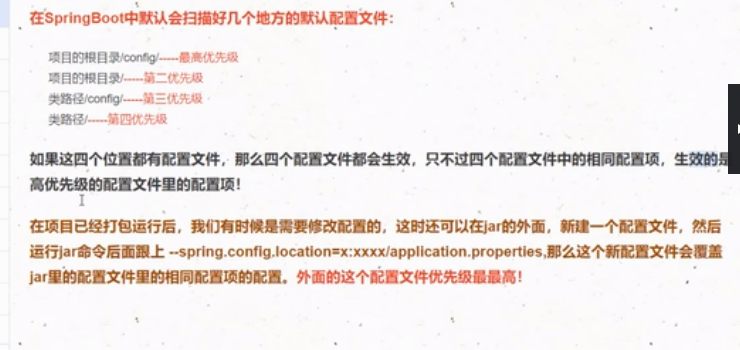
第二种直接在application.properties 中配置

server.port=8082  
spring.profiles.active=bbb

第三种直接在yml 配置文件中配置

**server**:  
 **port**: 8081  
**spring**:  
 **profiles**:  
 **active**: aaa  
  
  
  
---  
  
  
  
**server**:  
 **port**: 8083  
**spring**:  
 **profiles**: aaa  
  
  
---  
  
  
**server**:  
 **port**: 8086  
**spring**:  
 **profiles**: bbb

配置文件的优先级别



注意这种情况是在没有配置yml 文件的情况下 如果有yml 文件还是依然会覆盖

debug=**true**

会在运行的时候打出一份哪些类被使用的报告

springBoot 整合servelet 的方法有两种

1. 直接使用注解来注入

@WebServle 这个注解告诉容器这是servlet类

@WebServlet(urlPatterns = "/index.jsp")  
public class IndexServlet extends HttpServlet {  
  
 @Override  
 protected void doGet(HttpServletRequest req, HttpServletResponse resp){  
 System.*out*.println("this is servlet class");  
 }  
  
}

@SpringBootApplication  
@ServletComponentScan  
public class Demo2Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Demo2Application.class, args);  
 }  
  
}

@ServletComponentScan 这个注解在启动项目的时候回去扫描servlet并把它注入到容器中

第二种方法

@Bean  
public ServletRegistrationBean getBean(){  
 ServletRegistrationBean bean = new ServletRegistrationBean(new IndexServlet());  
 bean.addUrlMappings("/index"); // 这里可以加入数组  
 return bean;  
}

直接在入口启动函数中加入这个方法函数

SpringBoot 配置Filter 过滤器

也是两种方法和servlet 差不多

第一种

@WebFilter(urlPatterns = "/index")  
public class MyFilter implements Filter {  
  
 @Override  
 public void init(FilterConfig filterConfig) throws ServletException {  
 }  
  
 @Override  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chains) throws IOException, ServletException{  
 System.*out*.println("拦截器进来类");  
 chains.doFilter(request, response);  
 System.*out*.println("拦截器处理完了请求出去了");  
  
 }  
  
 @Override  
 public void destroy() {  
 }  
  
}

@ServletComponentScan  
public class Demo2Application {

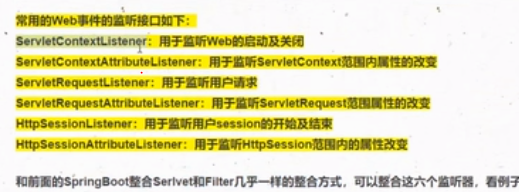
第二种方法：

@Bean  
public FilterRegistrationBean getFilterBean() {  
 FilterRegistrationBean bean = new FilterRegistrationBean(new MyFilter());  
 bean.addUrlPatterns("/\*");  
 return bean;  
}

这是Filter的处理部分:

@Override  
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chains) throws IOException, ServletException{  
 System.*out*.println("拦截器进来类");  
 HttpServletRequest req = (HttpServletRequest) request;  
 String url = req.getRequestURI();  
 if(url.contains("favicon.ico")){  
 return;  
 }  
 chains.doFilter(request, response);  
 System.*out*.println("拦截器处理完了请求出去了");  
  
}

SpringBoot 整合Listener



同样也有两种方法:

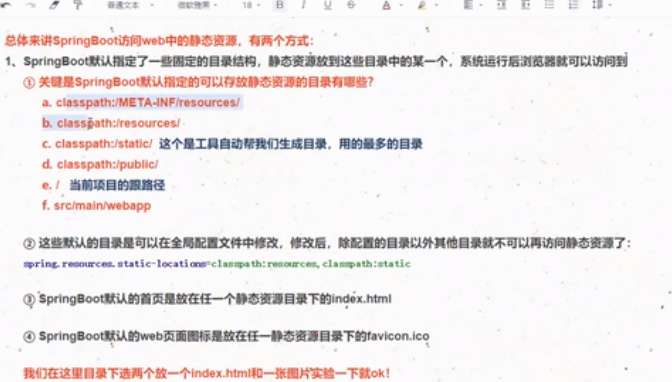
1.

@WebListener  
public class MyListener implements ServletContextListener {  
  
 @Override  
 public void contextInitialized(ServletContextEvent sce) {  
 System.*out*.println("系统被初始化了");  
 }  
  
 @Override  
 public void contextDestroyed(ServletContextEvent sce) {  
 }  
}

第二种方法:

@Bean  
public ServletListenerRegistrationBean getListenerBean() {  
 ServletListenerRegistrationBean bean = new ServletListenerRegistrationBean(new MyListener());  
 return bean;  
}

SpringBoot 访问静态资源



SpringBoot 整合jsp 页面

1. 先加载依赖

<!--jsp页面使用jstl标签-->  
<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
</dependency>  
<!--用于编译jsp-->  
<dependency>  
 <groupId>org.apache.tomcat.embed</groupId>  
 <artifactId>tomcat-embed-jasper</artifactId>  
 <scope>provided</scope>  
</dependency>

1. 修改全局配置文件

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

配置解析路径

1. 在main -> webapp -> WEB-INF -> jsp 新建 index.jsp 页面

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%@ **taglib** prefix="spring" uri="http://www.springframework.org/tags"%>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
 **${**11 + 12**}  
 ${** aaa **}**</body>  
</html>

编写controller 代码

@Controller  
public class indexController {  
  
 @Autowired  
 User user;  
  
 @Autowired  
 Address address;  
  
 @RequestMapping(value={"/index"},method= RequestMethod.*GET*)  
 public String index(Model model){  
  
 model.addAttribute("aaa", "这是springBooot 整合 jsp 页面的示例");  
 return "index";  
 }  
}

SpringBoot 扩展 mvc 的配置文件

/\* <mvc:view-controller path="//index.htmlmvc" view-name="index" />\*/

以前通过这个配置去配置一个页面，现在不需要，直接使用springBoot 的

扩展配置

代码示例:

@Configuration  
public class MyConfigAddMVC implements WebMvcConfigurer {  
  
 // WebMvcConfigurer 实现这个接口的方法  
 @Override  
 public void addViewControllers(ViewControllerRegistry registry) {  
 /\* <mvc:view-controller path="//index.htmlmvc" view-name="index" />\*/  
 registry.addViewController("/indexMvc.html").setViewName("index");  
 }  
  
}

springBoot 整合jdbc

1. 添加依赖

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

1. 在全局配置文件中添加jdbc连接属性

spring.datasource.username=root  
spring.datasource.data-password=wjx261357  
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdemo  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

1. 直接使用注解获取数据源连接

@Controller  
public class indexController {  
  
 @Autowired  
 private DataSource dataSource;  
  
 @Autowired  
 JdbcTemplate jdbcTemplate;  
  
 @ResponseBody  
 @RequestMapping(value={"/index"},method= RequestMethod.*GET*)  
 public String index(Model model) throws SQLException {  
 Connection connection = dataSource.getConnection();  
 List<Map<String,Object>> list = jdbcTemplate.queryForList("select \* from user");  
 System.*out*.println(list);  
 model.addAttribute("aaa", "这是springBooot 整合 jsp 页面的示例");  
 return "index";  
 }  
}

SpringBoot 整合Druid

1. 先加入依赖，因为druid 是第三方包 所以默认的springBoot 不支持

<!-- https://mvnrepository.com/artifact/com.alibaba/druid-spring-boot-starter -->  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid-spring-boot-starter</artifactId>  
 <version>1.1.17</version>  
</dependency>  
  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <!-- 可通过中央仓库获取最新的版本号，这里用我项目中的版本号 -->  
 <version>1.1.11</version>  
</dependency>

1. 在全局配置文件中加入配置属性

spring.datasource.username=root  
spring.datasource.data-password=wjx261357  
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdemo?useUnicode=true&characterEncoding=utf-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC   
spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
  
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  
  
spring.datasource.password=root  
spring.datasource.initialSize=5  
spring.datasource.minIdle=5  
spring.datasource.maxActive=10  
spring.datasource.maxWait=60000  
spring.datasource.timeBetweenEvictionRunsMillis=3600000  
spring.datasource.minEvictableIdleTimeMillis=3600000  
spring.datasource.validationQuery=select 'x'  
spring.datasource.testWhileIdle=true  
spring.datasource.testOnBorrow=true  
spring.datasource.testOnReturn=false  
spring.datasource.poolPreparedStatements=true  
spring.datasource.maxPoolPreparedStatementPerConnectionSize=20  
spring.datasource.filters=stat,wall,slf4j  
spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  
spring.datasource.useGlobalDataSourceStat=true

1. 使用注解将druidDataSource 注入到容器中

@Configuration  
public class DruidConfig {  
  
 // 配置Druid 数据源的参数信息  
 @ConfigurationProperties(prefix = "spring.datasource")  
 @Bean  
 public DataSource dataSource(){  
 return new DruidDataSource();  
 }  
  
 // 配置web 应用监听  
 @Bean  
 public ServletRegistrationBean getViewServlet(){  
 ServletRegistrationBean bean = new ServletRegistrationBean(new StatViewServlet());  
 // String urlArr = new String[]("/druid/\*"];  
 List<String> list = new ArrayList<>();  
 list.add("/druid/\*");  
 bean.setUrlMappings(list);  
 Map<String,String> maps = new HashMap<>();  
 maps.put("loginUsername", "admin");  
 maps.put("loginPassword", "123456");  
 maps.put("allow", "");  
 bean.setInitParameters(maps);  
 return bean;  
 }  
  
 // 配置druid 过滤器  
  
 @Bean  
 public FilterRegistrationBean getFilterBean(){  
 FilterRegistrationBean bean = new FilterRegistrationBean(new WebStatFilter());  
 List<String> list = new ArrayList<>();  
 list.add("/\*");  
 bean.setUrlPatterns(list);  
 Map<String, String> maps = new HashMap<>();  
 maps.put("exclusions","\*.js, \*.css, \*.jpg, /druid/\*");  
 bean.setInitParameters(maps);  
 return bean;  
 }  
}

1. 使用使用模板使用

@Controller  
public class indexController {  
  
 @Autowired  
 private DataSource dataSource;  
  
 @Autowired  
 JdbcTemplate jdbcTemplate;  
  
  
 @RequestMapping(value={"/index.html"},method= RequestMethod.*GET*)  
 public String index(Model model) throws SQLException {  
 // Connection connection = dataSource.getConnection();  
 List<Map<String,Object>> list = jdbcTemplate.queryForList("select \* from user");  
 // System.out.println(list);  
 // DruidDataSource data = new DruidDataSource();  
 model.addAttribute("user", list);  
 // System.out.println(dataSource);  
 model.addAttribute("aaa", "这是springBooot 整合 jsp 页面的示例");  
 return "index";  
 }  
}

SpringBoot 整合 mybatis

1. 添加依赖

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

1. **在resources 目录下创建mybatis 文件夹 并创建mybatis-config.xml 内容为:**

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE configuration  
 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">  
  
<configuration>  
 <settings>  
 <setting name="mapUnderscoreToCamelCase" value="true"/>  
 </settings>  
</configuration>

1. **在mybatis 文件夹下创建mapper 文件夹并创建UserDao.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="cn.ybcy.jdbc.jdbcdemo.dao.UserDao">  
   
 <insert id="add">  
 insert into user (id, username) values (#{id},#{username})  
 </insert>  
   
</mapper>

1. **在全局配置文件中添加配置**

mybatis.config-location=classpath:mybatis/mybatis-config.xml  
mybatis.mapper-locations=classpath:mybatis/mapper/\*.xml

**这两个文件分别指定mybatis 全局配置文件路径和对应的映射路径**

1. **在cn.ybcy.jdbc.jdbcdemo.model下穿件User模型文件**

package cn.ybcy.jdbc.jdbcdemo.model;  
  
public class User {  
  
 private int id;  
  
 private String username;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", username='" + username + '\'' +  
 '}';  
 }  
}

1. **在cn.ybcy.jdbc.jdbcdemo.dao下创建UserDao类文件，与mapper 文件一一对应**

package cn.ybcy.jdbc.jdbcdemo.dao;  
  
import cn.ybcy.jdbc.jdbcdemo.model.User;  
  
public interface UserDao {  
  
 public void add(User user);  
}

**7.在启动文件中加入**@MapperScan注解，告诉Mybatis在哪个文件下扫描对应的映射文件

@MapperScan({"cn.ybcy.jdbc.jdbcdemo.dao"})  
public class JdbcdemoApplication {  
  
 public static void main(String[] args) {  
  
 SpringApplication.*run*(JdbcdemoApplication.class, args);  
 }  
  
}

1. **使用**

@Autowired  
UserDao userDao;  
  
@RequestMapping(value={"/index.html"},method= RequestMethod.*GET*)  
public String index(Model model) throws SQLException {  
 User user = new User();  
 user.setId(34);  
 user.setUsername("xiaoming");  
 userDao.add(user);  
 return "index";  
}

**SpringCloud 学习部分**

1. **新建一个服务注册中心服务器 new -> module -> 选择eureka server**

**具体的pom文件如下:**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-eureka-my</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-eureka-my</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Application.yml 文件如下:**

**spring**:  
 **application**:  
 **name**: hello-spring-cloud-eureka-my  
**server**:  
 **port**: 8761  
**eureka**:  
 **instance**:  
 **hostname**: localhost  
 **client**:  
 **registerWithEureka**: **false  
 fetchRegistry**: **false  
 serviceUrl**:  
 **defaultZone**: http://${**eureka.instance.hostname**}:${**server.port**}/eureka/

**启动类的加入注解即可**

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

1. **新建服务提供者服务器 建module的流程一样**

**Pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-service-my</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-service-my</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
 <repositories>  
 <repository>  
 <id>spring-milestones</id>  
 <name>Spring Milestones</name>  
 <url>https://repo.spring.io/milestone</url>  
 </repository>  
 </repositories>  
  
</project>

**Application.yml**

**spring**:  
 **application**:  
 **name**: hello-spring-cloud-service-my  
**server**:  
 **port**: 8763  
**eureka**:  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://localhost:8761/eureka/

**启动类**

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

**Controller 这就是一个简单的服务**

@RestController  
public class AdminController {  
  
 @Value("${server.port}")  
 private int port;  
  
 @GetMapping(value="hi")  
 public String sayHi(String message){  
  
 return String.*format*("Hi your message is: %s port: %s", message, port);  
 }  
}

1. **新建服务消费者服务器 -- rabion**

**Pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-consumer-my</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-consumer-my</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-zipkin</artifactId>  
 </dependency>  
  
 <!--断路器-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
 </dependency>  
  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Application.yml**

**spring**:  
 **application**:  
 **name**: hello-spring-cloud-consumer-my  
**server**:  
 **port**: 8764  
**eureka**:  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://localhost:8761/eureka/

**启动类:**

@SpringBootApplication  
@EnableDiscoveryClient  
@EnableHystrix 启动断路器  
public class HelloSpringCloudConsumerMyApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(HelloSpringCloudConsumerMyApplication.class, args);  
 }  
  
}

**Service**

@Service  
public class AdminService {  
  
 @Autowired  
 RestTemplate restTemplate;  
  
 @HystrixCommand(fallbackMethod = "errorPage")  
 public String sayHi(String message){  
 return restTemplate.getForObject("http://hello-spring-cloud-service-my/hi?message=" + message, String.class);  
 }  
 // 断路器回调函数

public String errorPage(String message){  
 return String.*format*("Hi your message is: %s,but request is failure", message);  
 }  
}

**Controller**

@RestController  
public class AdminController {  
  
 @Autowired  
 AdminService adminService;  
  
 @GetMapping(value="hi")  
 public String sayHi(String message){  
 return adminService.sayHi(message);  
 }  
}

**restTemplate 配置类**

@Configuration  
public class RestTemplateConfiguration {  
  
 @Bean  
 @LoadBalanced  
 public RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
}

**第二种消费者 -- feign**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-feign-client</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-feign-client</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <!-- 断路器仪表盘-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
 <version>2.0.2.RELEASE</version>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Application.yml**

**spring**:  
 **cloud**:  
 **config**:  
 **uri**: http://localhost:8888  
 **name**: spring-cloud-feign  
 **label**: master  
 **profile**: prop 这里配置了统一配置

**启动类**

@SpringBootApplication  
@EnableFeignClients  
@EnableDiscoveryClient  
@EnableHystrixDashboard  
public class HelloSpringCloudFeignClientApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(HelloSpringCloudFeignClientApplication.class, args);  
 }  
  
}

**断路器service**

@FeignClient(value = "hello-spring-cloud-service-my", fallback = AdminServiceImpl.class)  
public interface FeignService {  
  
 @GetMapping(value="hi")  
 public String sayHi(@RequestParam(value = "message") String message);  
}

**Service 实现类**

@Component  
public class AdminServiceImpl implements FeignService {  
  
 @Override  
 public String sayHi(String message){  
 return String.*format*("Hi your message is: %s, but request is bad", message);  
 }  
}

**短路器配置类**

@Configuration  
public class HystrixServlet {  
  
 @Bean  
 public ServletRegistrationBean getHystrixBean(){  
 ServletRegistrationBean bean = new ServletRegistrationBean(new HystrixMetricsStreamServlet());  
 bean.setLoadOnStartup(1);  
 bean.addUrlMappings("/hystrix.stream");  
 bean.setName("HystrixServlet");  
 return bean;  
 }  
}

**Controller类**

@RestController  
public class AdminController {  
  
 @Autowired  
 private FeignService feignService;  
  
 @RequestMapping(value="hi", method = RequestMethod.*GET*)  
 public String sayHi(String message){  
 System.*out*.print("请求到这里了");  
 return feignService.sayHi(message);  
 }  
}

1. **新建统一网关用户访问接口**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-gateway</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-gateway</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-zipkin</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-zuul</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Application.yml 配置**

**spring**:  
 **application**:  
 **name**: hello-spring-cloud-gateway  
**server**:  
 **port**: 8769  
**eureka**:  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://localhost:8761/eureka/  
  
**zuul**:  
 **routes**:  
 **api-a**:  
 **path**: /api/a/\*\*  
 **serviceId**: hello-spring-cloud-feign-client  
 **api-b**:  
 **path**: /api/b/\*\*  
 **serviceId**: hello-spring-cloud-consumer-my

**启动类:**

package com.funtl.hellospringcloudgateway;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  
  
@SpringBootApplication  
@EnableEurekaClient  
@EnableZuulProxy  
public class HelloSpringCloudGatewayApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(HelloSpringCloudGatewayApplication.class, args);  
 }  
  
}

**Filter 过滤器**

@Component  
public class LoginFilter extends ZuulFilter {  
  
 @Override  
 public String filterType(){  
 return "pre";  
 }  
  
 @Override  
 public int filterOrder(){  
  
 return 0;  
 }  
  
 @Override  
 public boolean shouldFilter() {  
 return true;  
 }  
  
 @Override  
 public Object run() {  
 RequestContext currentContext = RequestContext.*getCurrentContext*();  
 HttpServletRequest request = currentContext.getRequest();  
 String token = request.getParameter("token");  
 if(null == token){  
 currentContext.setSendZuulResponse(false);  
 currentContext.setResponseStatusCode(401); // 无权限  
 try {  
 currentContext.getResponse().setContentType("text/html;charset=utf-8");  
 currentContext.getResponse().getWriter().write("非法请求");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return null;  
 }  
}

**回调函数提供类**

@Component  
public class WebAdminFeignFallbackProvider implements FallbackProvider {  
 @Override  
 public String getRoute(){  
 return "hello-spring-cloud-wsb-admin-feign";  
 }  
  
 @Override  
 public ClientHttpResponse fallbackResponse(String route, Throwable cause){  
 return new ClientHttpResponse() {  
 @Override  
 public HttpStatus getStatusCode() throws IOException {  
 return HttpStatus.*OK*;  
 }  
  
 @Override  
 public int getRawStatusCode() throws IOException {  
 return HttpStatus.*OK*.value();  
 }  
  
 @Override  
 public String getStatusText() throws IOException {  
 return HttpStatus.*OK*.getReasonPhrase();  
 }  
  
 @Override  
 public void close() {  
  
 }  
  
 @Override  
 public InputStream getBody() throws IOException {  
 ObjectMapper mapper = new ObjectMapper();  
 Map<String, Object> map = new HashMap<>();  
 map.put("status", 200);  
 map.put("message", "无法连接， 请检查您的网络");  
  
 return new ByteArrayInputStream(mapper.writeValueAsString(map).getBytes("UTF-8"));  
 }  
  
 @Override  
 public HttpHeaders getHeaders() {  
 HttpHeaders headers = new HttpHeaders();  
 headers.setContentType(MediaType.*APPLICATION\_JSON\_UTF8*);  
 return headers;  
 }  
 };  
 }  
}

1. **创建统一资源配置服务器**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.funtl</groupId>  
 <artifactId>hello-spring-cloud-config</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>hello-spring-cloud-config</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-zipkin</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**启动类:**

package com.funtl.hellospringcloudconfig;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.config.server.EnableConfigServer;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
@EnableConfigServer  
@EnableEurekaClient  
public class HelloSpringCloudConfigApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(HelloSpringCloudConfigApplication.class, args);  
 }  
  
}

**Application.yml:**

**server**:  
 **port**: 8888  
*#服务名字***spring**:  
 **application**:  
 **name**: hello-spring-cloud-config  
 **cloud**:  
 **config**:  
 **label**: master  
 **server**:  
 **git**:  
 *#git 仓库的地址* **uri**: https://github.com/jxwangGit/springCloud.git  
 *#git 仓库的账号密码* **username**: jxwangGit  
 **password**: wjx151364899419  
*#加入注册中心，实现高可用***eureka**:  
 **client**:  
 **serviceUrl**:  
 **defaultZone**: http://localhost:8761/eureka/  
  
**zipkin**:  
 **base-url**: http://localhost:9411