3、服务容错保护

# 1、准备的项目工程（1文档或者是是2中的工程,这里我终于找到源码的位置了，所以我使用的源码）

## 1、服务注册中心 ，端口为1111

## 2、服务提供者，端口为8080，8081

## 3、服务消费者 端口为9000

# 2、在服务消费者中引入依赖包

*<?*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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.didispace</groupId>  
 <artifactId>ribbon-consumer</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>ribbon-consumer</name>  
 <description>Demo project for Spring Boot</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.3.7.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-ribbon</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-hystrix</artifactId>  
 </dependency>  
  
   
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Brixton.SR5</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>

# 3、注解开启断路器功能@EnableCircuitBreaker

@EnableCircuitBreaker //开启断路器功能  
@EnableDiscoveryClient //开启服务发现客户端，  
@SpringBootApplication  
public class ConsumerApplication {  
  
 @Bean  
 @LoadBalanced  
 RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
 public static void main(String[] args) {  
 SpringApplication.run(ConsumerApplication.class, args);  
 }  
  
}

## 解释： 这里还可以使用 @ SpringCloudApplication 注解代替上面三个注解，由此也可以得到spring Cloud标准应用包含服务发下和断路器

@EnableCircuitBreaker //开启断路器功能  
@EnableDiscoveryClient //开启服务发现客户端，  
@SpringBootApplication  
//@SpringCloudApplication //可以取代上面三个  
public class ConsumerApplication {  
  
 @Bean  
 @LoadBalanced  
 RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
 public static void main(String[] args) {  
 SpringApplication.run(ConsumerApplication.class, args);  
 }  
  
}

# 4。@HystrixCommand新建service类，用来添加指定回调方法

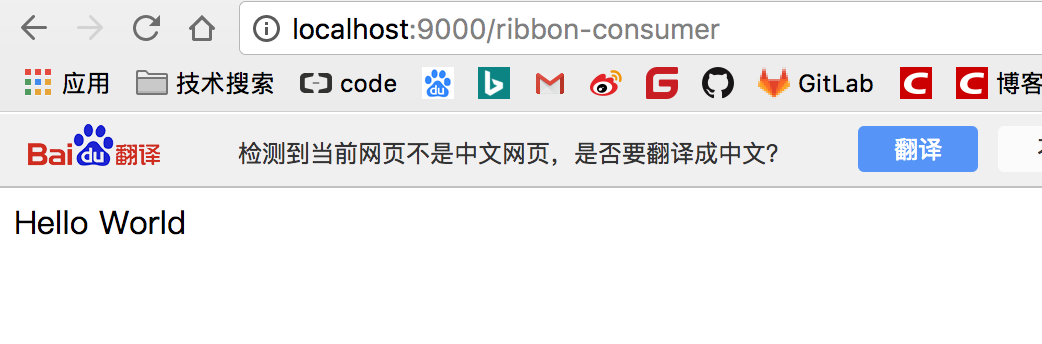
@Service  
public class HelloService {  
  
 @Autowired  
 RestTemplate restTemplate;  
  
 @HystrixCommand(fallbackMethod = "helloFallBack")  
 public String hello(){  
 return restTemplate.getForEntity("http://HELLO-SERVICE/hello",String.class).getBody();  
 }  
  
  
 public String helloFallBack(){  
 return "error";  
 }  
}

# 5、修改之前的服务消费者controller方法

@RestController  
public class ConsumerController {  
  
  
 @Autowired  
 HelloService helloService;  
  
 @ResponseBody  
 @RequestMapping(value = "/ribbon-consumer", method = RequestMethod.***GET***)  
 public String helloConsumer() {  
 return helloService.hello();  
 }  
  
  
}

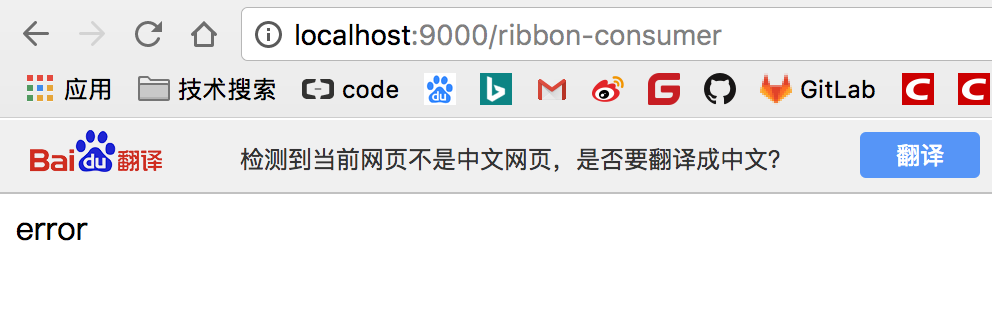
# 6、开始验证断路器的实现的回调逻辑，启动所有的服务，访问http://localhost:9000/ribbon-consumer

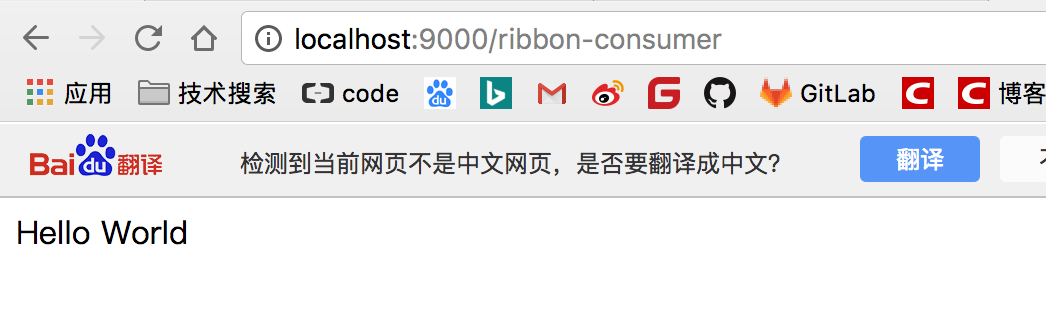
## 1、在8080 8081 这两个服务提供者工作的时候，返回的结果都是一样的



## 1、直接断开8081服务提供者

### 这个时候我们将8081的服务提供者挂掉，继续访问，发现一会正常，一会显示error，但是时间长了，就还是只会显示hello ，出现这种情况，应该是注册中心没有及时检测到挂掉了8081，还继续提供给消费者服务。但是时间长了，就肯定原型毕露





## 2、模拟服务阻塞，添加延迟(hystrix默认超时2000毫秒) （启动所有的服务）

### 结果：结果和上面的是一样的，也是有错误，有正常

/\*\*  
 \* 2、断路器，模拟服务阻塞  
 \*/  
@RequestMapping(value = "/hello", method = RequestMethod.***GET***)  
public String hello() throws InterruptedException {  
  
 ServiceInstance instance = client.getLocalServiceInstance();  
  
  
 int sleepTime = new Random().nextInt(3000);  
 logger.info("处理线程等待 "+sleepTime+" 秒");  
 Thread.sleep(sleepTime);  
  
 logger.info("/hello, host:" + instance.getHost() + ", service\_id:" + instance.getServiceId());  
 return "Hello World";  
}