项目课程系列之 Atcrowdfunding

尚硅谷 JavaEE 教研组

版本：V1.0

第一章 SpringCloud 介绍

1.1 概念

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring 并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

“微服务架构”在这几年非常的火热，以至于关于微服务架构相关的开源产品被反复的提及（比如：netflix、dubbo），Spring Cloud 也因 Spring 社区的强大知名度和影响力也被广大架构师与开发者备受关注。

那么什么是“微服务架构”呢？简单的说，微服务架构就是将一个完整的应用从数据存储开始垂直拆分成多个不同的服务，每个服务都能独立部署、独立维护、独立扩展，服务与服务间通过诸如 RESTful API 的方式互相调用。

1.2 为什么使用 Spring Cloud？

Spring Cloud 对于中小型互联网公司来说是一种福音，因为这类公司往往没有实力或者没有足够的资金投入去开发自己的分布式系统基础设施，使用 Spring Cloud 一站式解决方案能在从容应对业务发展的同时大大减少开发成本。同时，随着近几年微服务架构和 Docker 容器概念的火爆，也会让 Spring Cloud 在未来越来越“云”化的软件开发风格中立有一席之地，尤其是在目前五花八门的分布式解决

方案中提供了标准化的、全站式的技术方案，意义可能会堪比当前 Servlet 规范的诞生，有效推进服务端软件系统技术水平的进步。

1.3 应用 Spring Cloud

Spring Cloud Netflix 项目是 Spring Cloud 的子项目之一，主要内容是对 Netflix 公司一系列开源产品的包装，它为 Spring Boot 应用提供了自配置的 Netflix OSS 整合。通过一些简单的注解，开发者就可以快速的在应用中配置一下常用模块并构建庞大的分布式系统。它主要提供的模块包括：服务发现（Eureka），断路器（Hystrix），智能路由（Zuul），客户端负载均衡（Ribbon）等。

1.4 Spring Cloud 是分布式系统的整体解决方案

- SpringBoot&Spring 什么关系？

SpringBoot 底层就是 Spring，简化使用 Spring 的方式而已，多加了好多的自动配置；

- Spring Cloud&SpringBoot 什么关系？

Spring Cloud 是分布式系统的整体解决方案，底层用的 SpringBoot 来构建项目，Cloud 新增很多的分布式的 starter，包括这些 starter 的自动配置；

1.5 官方网站

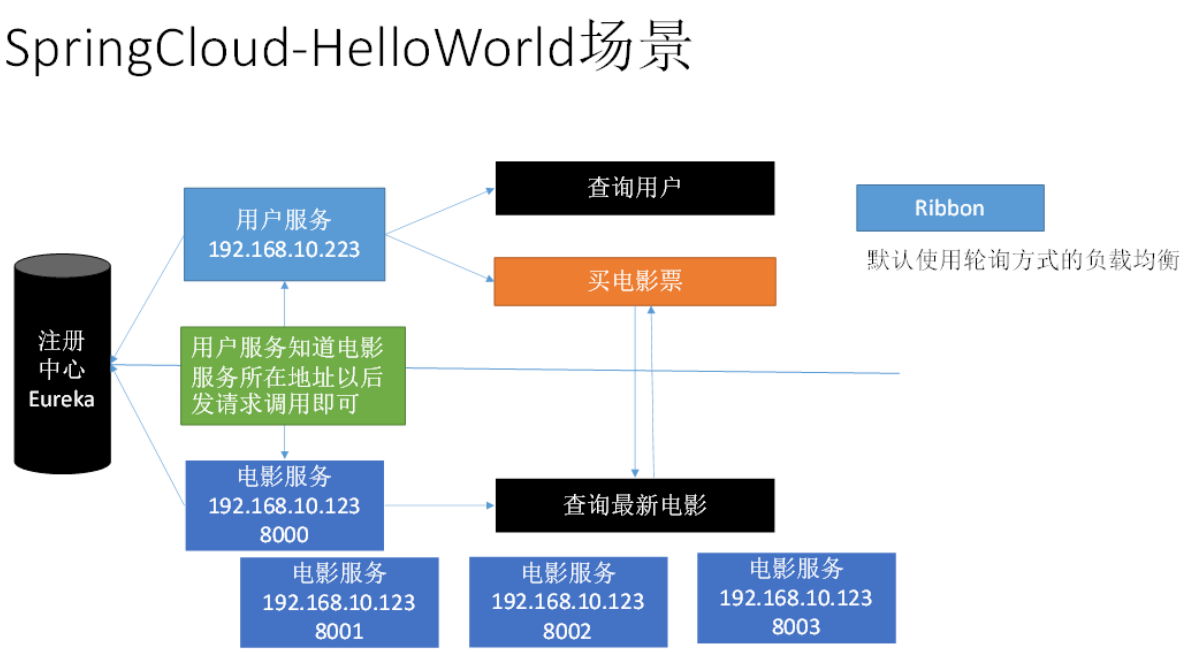
http://spring.io/projects

https://projects.spring.io/spring-cloud/#quick-start

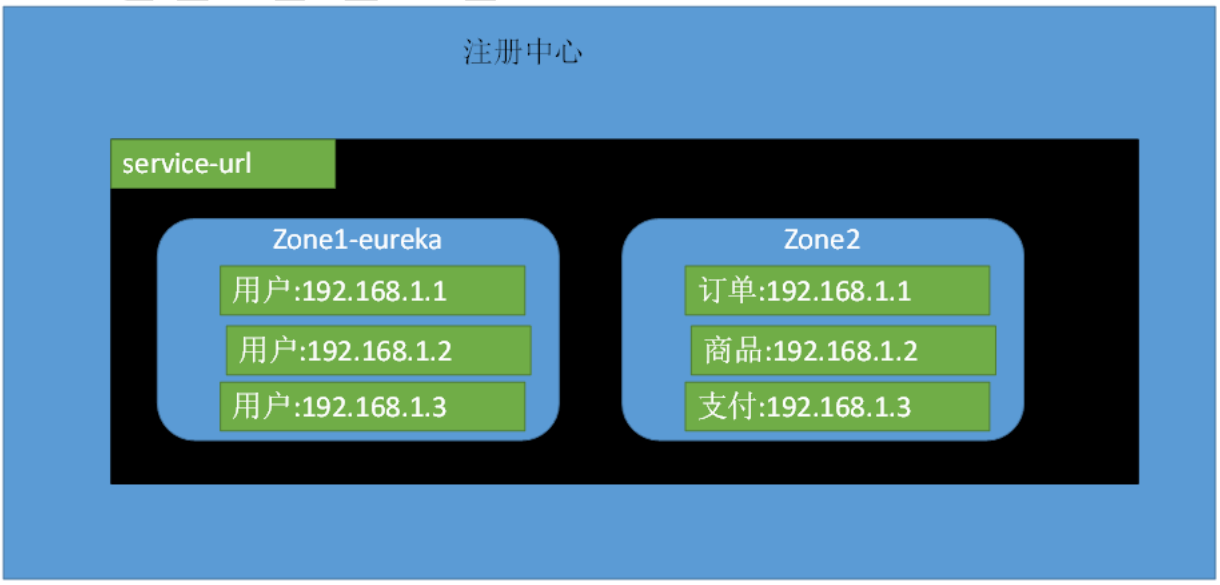
https://springcloud.cc/

第二章 SpringCloud-HelloWorld 架构图

2.1 案例图解



2.2 注册中心



配置 Region 和 Zone

```
eureka:
  instance:
    prefer-ip-address: true
    lease-renewal-interval-in-seconds: 30
    lease-expiration-duration-in-seconds: 90
  client:
    region: cq          # 定义region
    availability-zones:
      cq: cs            # 定义cq下的zone
    fetch-registry: true
    register-with-eureka: true
    service-url:
      cs: http://localhost:8080/eureka/, http://localhost:8079/eureka/
```

Region 和 Zone 就相当于大区和机房，一个 Region（大区）可以有很多的 Zone（机房）。在 Spring Cloud 中，服务消费者会优先查找在同一个 Zone 的服务，之后在去查找其他的服务。如果该项配置使用的好，那么项目请求的响应时间将大大缩短！

第三章 SpringCloud-HelloWorld 案例开发-注册中心

3.1 注册中心

（Eureka , jar 工程）-cloud-eureka-registry-center

3.1.1 引入 eureka-server

3.1.2 编写 application.yml

```
spring:
  application:
```

```
    name: cloud-eureka-registry-center

server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false #自己就是注册中心，不用注册自己
    fetch-registry: false #要不要去注册中心获取其他服务的地址
    service-url:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

3.1.3 开启 Eureka 注册中心功能；@EnableEurekaServer

3.1.4 测试

Boot Dashboard 视图启动

访问 http://localhost:8761

第四章 SpringCloud-HelloWorld 案例开发-电影服务

4.1 电影服务

（jar 工程，提供查询电影功能）cloud-provider-movie

4.1.1 引入 eureka-Discovery、web 模块

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

4.1.2 创建 Movie 实体类,增加 id,movieName 属性

4.1.3 创建 MovieDao

```
@Repository
public class MovieDao {
    public Movie getNewMovie(){
        Movie movie = new Movie();
        movie.setId(1);
        movie.setMovieName("战狼");
        return movie;
    }
}
```

4.1.4 创建 MovieService

```
@Service
```

```
public class MovieService {
    @Autowired
    MovieDao movieDao;
    public Movie getNewMovie(){
        return movieDao.getNewMovie();
    }
}
```

4.1.5 创建 MovieController

```
@RestController
public class MovieController {

    @Autowired
    MovieService movieService;

    /**
     * 获取最新电影
     */
    @GetMapping("/movie")
    public Movie getNewMovie(){
        return movieService.getNewMovie();
    }

}
```

4.1.6 启动测试

http://localhost:8080，与注册中心无关，注册中心无此服务

4.1.7 编写 application.yml

```
spring:
  application:
    name: cloud-provider-movie

server:
  port: 8000

# 指定注册到哪个注册中心
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true #注册自己服务使用 ip 的方式
```

4.1.8 将自己自动注册到注册中心@EnableDiscoveryClient

4.1.9 启动注册中心和服务

查看注册中心，访问服务 http://localhost:8000

第五章 SpringCloud-HelloWorld 案例开发-用户服务

5.1 用户服务

（jar 工程，提供查询用户，买电影票功能）cloud-consumer-user

5.1.1 引入 eureka-Discovery、web 模块

5.1.2 创建 User 实体类,增加 id,userName 属性

5.1.3 创建 UserDao

```
@Repository
public class UserDao {
    public User getUser(Integer id){
        User user = new User();
        user.setId(id);
        user.setUserName("张三");
        return user;
    }
}
```

5.1.4 创建 UserService

拷贝 Movie 类

```
@Service
```

```
public class UserService {


    @Autowired
    UserDao userDao;


    public User getUserById(Integer id){
        User user = userDao.getUser(id);
        return user;
    }


    /**
     * 购买最新的电影票
     * 传入用户 id
     */
    @HystrixCommand(fallbackMethod="hystrix")
    public Map<String, Object> buyMovie(Integer id){
        Map<String, Object>    result = new HashMap<>();
        //1、查询用户信息
        User user = getUserById(id);
        //2、查到最新电影票
        result.put("user", user);
        result.put("movie", null);//暂时为 null
        return result;
    }
}
```

5.1.5 创建 UserController

```
@RestController
public class UserController {


    @Autowired
    UserService userService;


    @GetMapping("/user")
    public User getUser(@RequestParam("id")Integer id){
        User user = userService.getUserById(id);
        return user;
    }


    @GetMapping("/buyMovie")
    public Map<String, Object> buyMovie(@RequestParam("id")Integer userid){
        Map<String, Object> map = userService.buyMovie(id);
        return map;
    }


}
```

5.1.6 编写 application.yml

```
spring:
    application:
        name: cloud-consumer-user
```

```
server:
  port: 9000

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true #注册中心保存我的 ip
```

5.1.7 将自己自动注册到注册中心@EnableDiscoveryClient

5.1.8 启动注册中心和服务，查看注册中心，访问服务

http://localhost:9000/user?id=1

http://localhost:9000/buyMovie?id=1

第六章 SpringCloud-HelloWorld 案例开发-Ribbon-RestTemplate

6.1 Ribbon 负载均衡，可以用于远程调用（用户服务 调用 电影服务 项目）

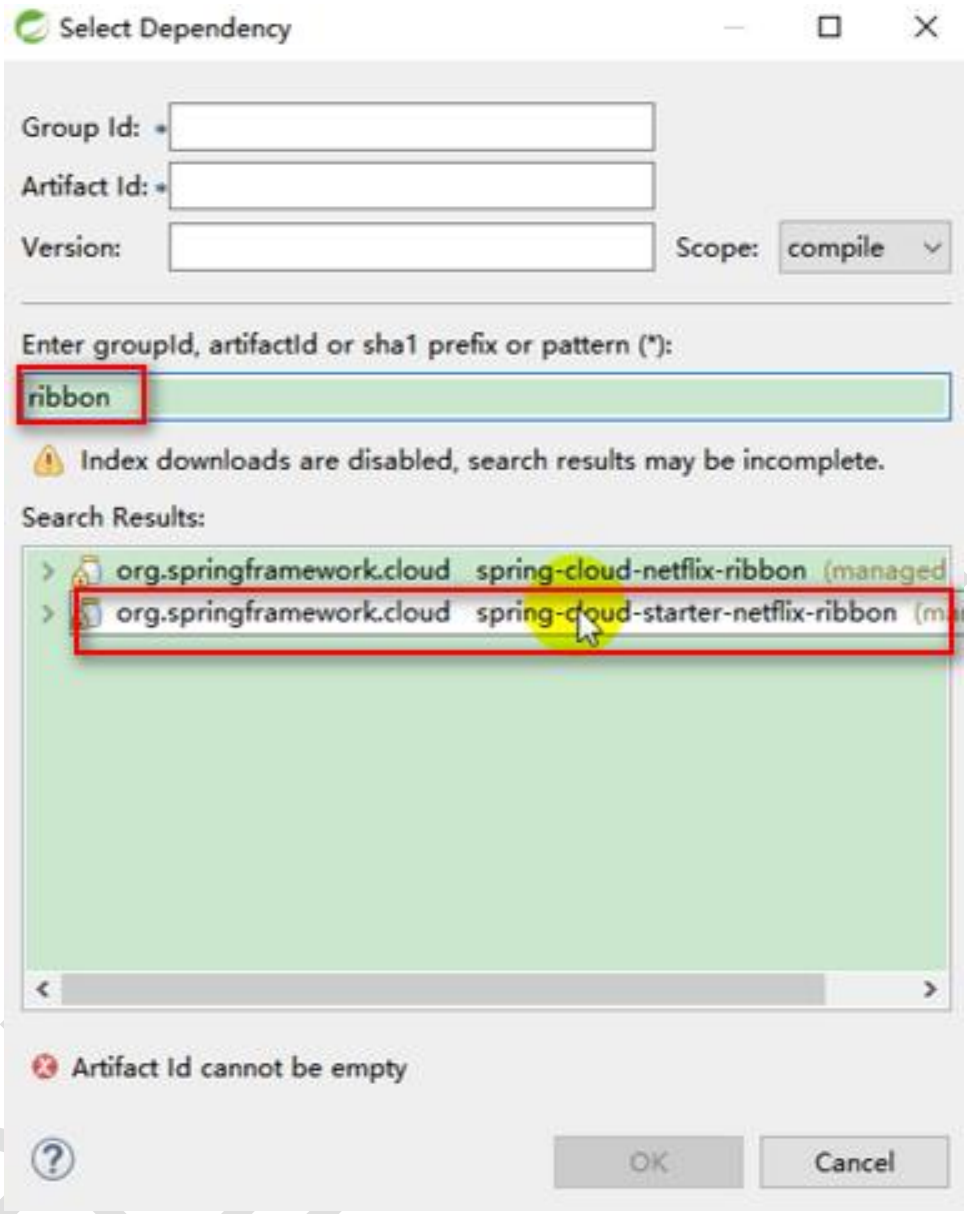
如何使用 Ribbon

1）、引入 Ribbon 的 Starter

2）、配置使用 Ribbon 功能；底层使用 RestTemplate 的工具来给远程发送请求

6.1.1 用户服务项目中引入 Ribbon

```
<!-- 引入 ribbon 实现远程调用和负载均衡功能 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
```



6.1.2 RestTemplate

给容器中注入一个 RestTemplate 并使用 Ribbon 进行负载均衡调用

```
@LoadBalanced      //负载均衡
@Bean
public RestTemplate restTemplate(){
    return new RestTemplate();
```

```
}
```

6.1.3 使用 RestTemplate 远程调用（UserService 中完善代码）

```
@Autowired
RestTemplate restTemplate;
/**
 * 购买最新的电影票
 * 传入用户 id
 */
public Map<String, Object> buyMovie(Integer id){
    Map<String, Object>    result = new HashMap<>();
    //1、查询用户信息
    User user = getUserById(id);
    //2、查到最新电影票    restTemplate 使用 java 代码来模拟发请求
    Movie movie = restTemplate.getForObject("http://CLOUD-PROVIDER-MOVIE/movie", Movie.class);

    result.put("user", user);
    result.put("movie", movie);
    return result;
}
```

6.1.4 启动注册中心和服务，访问服务

http://localhost:9000/buyMovie?id=1　显示用户和电影信息

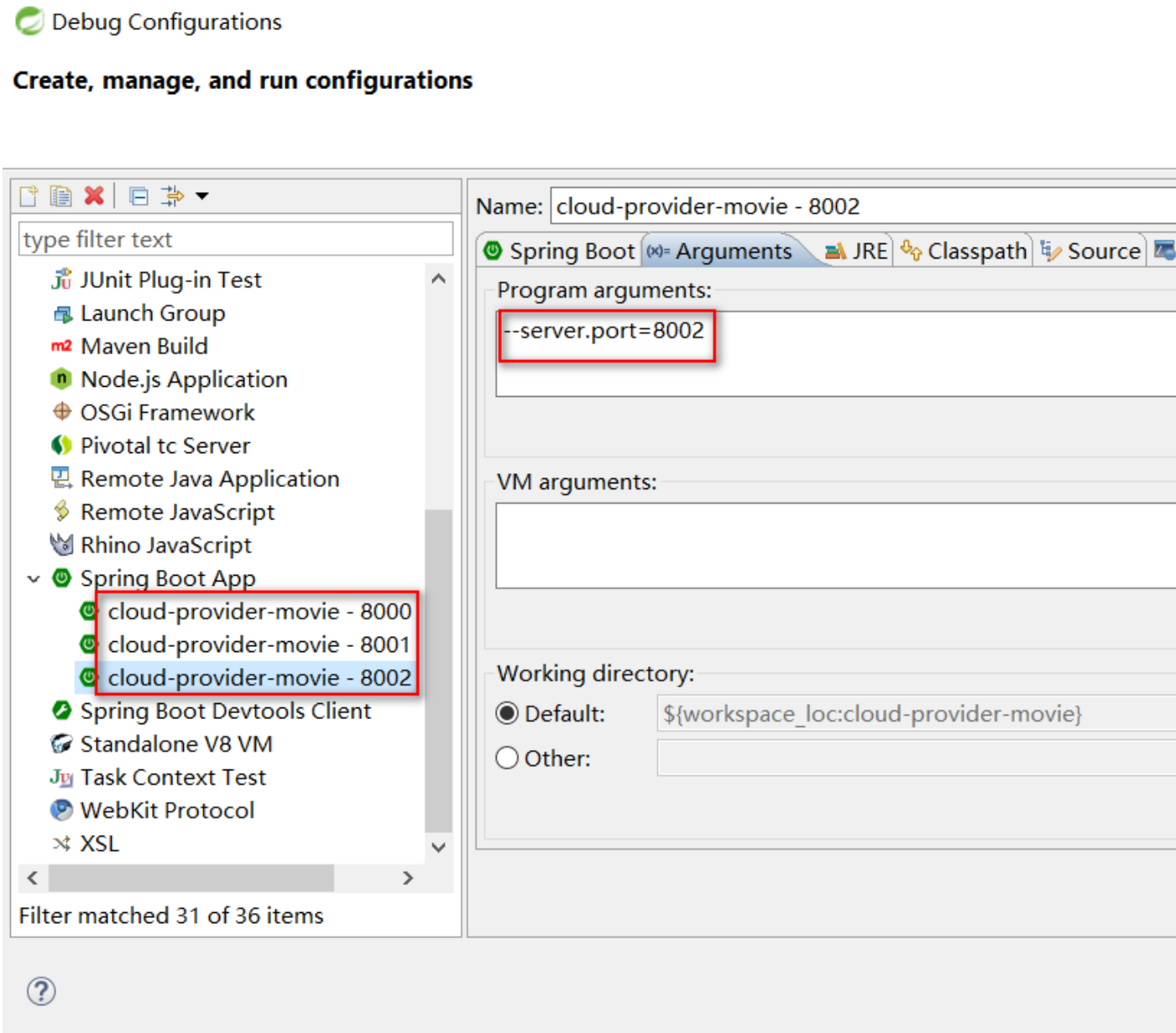
6.1.5 在电影服务（MovieService 类）中打印服务端口，方便监控服务执行情况。

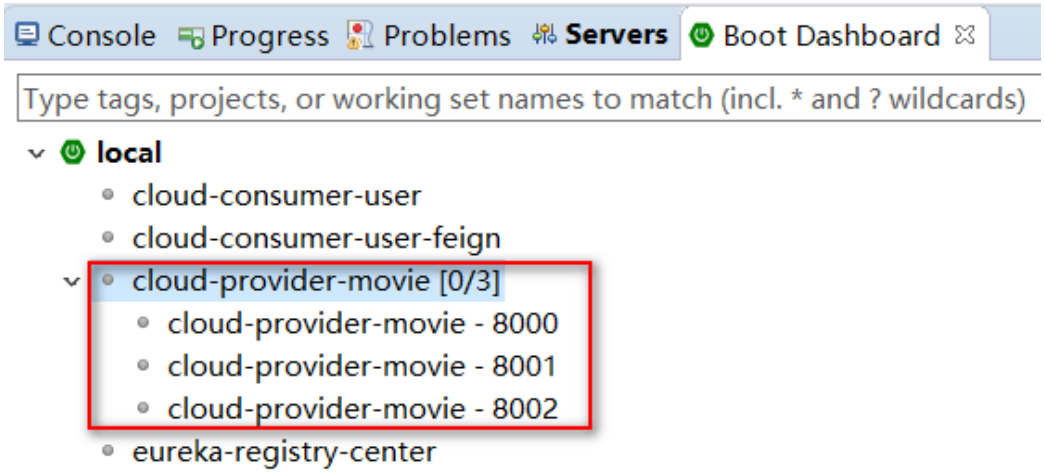
```
@Value("${server.port}")
private String port;


public Movie getNewMovie(){
    System.out.println("当前电影服务的端口："+port);
    return movieDao.getNewMovie();
}
```

6.1.6 将电影服务项目启动多次，测试负载均衡

http://localhost:9000/buyMovie?id=1

- 打 jar 包方式运行
  - java -jar xxx.jar --server.port=8000
  - java -jar xxx.jar --server.port=8001
  - java -jar xxx.jar --server.port=8002
- IDE 工具中方式运行


Debug Configurations
Create, manage, and run configurations
type filter text
JUnit Plug-in Test
Launch Group
Maven Build
Node.js Application
OSGi Framework
Pivotal tc Server
Remote Java Application
Remote JavaScript
Rhino JavaScript
Spring Boot App
cloud-provider-movie - 8000
cloud-provider-movie - 8001
cloud-provider-movie - 8002
Spring Boot Devtools Client
Standalone V8 VM
Task Context Test
WebKit Protocol
XSL
Filter matched 31 of 36 items
Name: cloud-provider-movie - 8002
Spring Boot
Arguments
JRE
Classpath
Source
Program arguments:
--server.port=8002
VM arguments:
Working directory:
Default:
${workspace_loc:cloud-provider-movie}
Other:



第七章 SpringCloud-HelloWorld 案例开发-Feign-声明式调用

7.1 Feign 声明式调用

7.2 创建新 User 项目 cloud-consumer-user-feign，引入 eureka-Discovery、web、Feign 模块

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

7.2.1 开启@EnableDiscoveryClient 服务发现

```
/**
 * 1、引入 feign 的 starter
```

```
 * 2、写一个接口，和被调用的服务关联起来
 * 3、开启 Feign 功能;@EnableFeignClients
 */
@EnableFeignClients
@EnableDiscoveryClient
@SpringBootApplication
public class CloudConsumerUserFeignApplication {
    public static void main(String[] args) {
        SpringApplication.run(CloudConsumerUserFeignApplication.class, args);
    }
}
```

7.2.2 编写 application.yml

```
spring:
  application:
    name: cloud-consumer-user-feign

server:
  port: 7000

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true #注册中心保存我的 ip
```

7.2.3 复制用户服务项目的代码：实体类，Dao，Service，Controller

7.2.4 将 RestTemplate 方式改成 Feign 方式进行远程调用

```
package com.atguigu.feign;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

import com.atguigu.bean.Movie;
import com.atguigu.feign.exception.MovieFeignExceptionHandlerService;

/**
 * 调用指定服务名称 “CLOUD-PROVIDER-MOVIE” 的 @GetMapping("/movie") 映射方法
 * 这个方法声明与电影服务端 Controller 映射的方法声明一致即可。
 */
@FeignClient(value="CLOUD-PROVIDER-MOVIE") //与被调用端的服务名称一致
public interface MovieServiceFeign {
   @GetMapping("/movie")
   public Movie getNewMovie(); //与被调用服务端的映射方法一致
}
```

7.2.5 修改 UserService 代码

```
@Service
public class UserService {

   @Autowired
```

```
    UserDao userDao;

    //面向接口编程
    @Autowired
    MovieServiceFeign movieServiceFeign; //调用 Feign 接口；其实就是调用远程服务

    /**
     * 购买最新的电影票
     * 传入用户 id
     */
    public Map<String, Object> buyMovie(Integer id){
        Map<String, Object>    result = new HashMap<>();
        //1、查询用户信息
        User userById = getUserById(id);
        //2、查到最新电影票   Feign 方式发起远程调用
        Movie movie = movieServiceFeign.getNewMovie();

        result.put("user", userById);
        result.put("movie", movie);
        return result;
    }
}
```

7.2.6 开启 Feign 功能@EnableFeignClients

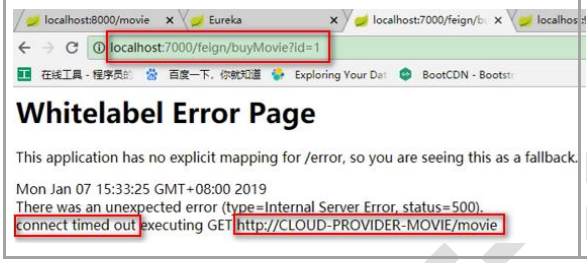
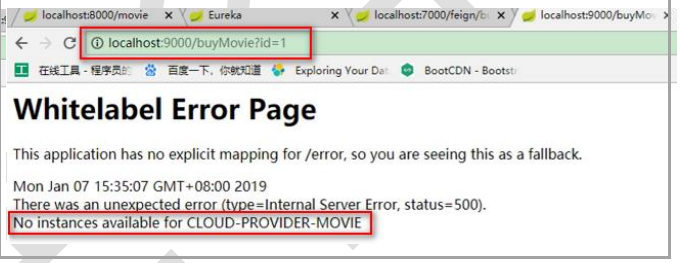
7.2.7 测试调用与负载均衡效果

http://localhost:7000/buyMovie?id=1

第八章 SpringCloud-HelloWorld 案例开发-Ribbon+Hystrix 组合

8.1 Hystrix 服务熔断

目前情况：如果集群的三个电影服务都挂了，会怎样？

服务刚刚都挂掉	服务挂掉有一会了
	

解决：	返回默认数据，会提示错误消息

8.2 使用 Ribbon+Hystrix 组合：cloud-consumer-user

8.2.1 引入 Hystrix

```
<!--  引入 hystrix 进行服务熔断  -->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

8.2.2 开启断路保护功能@EnableCircuitBreaker

8.2.3 在方法上标注@HystrixCommand(fallbackMethod="xxx")来指定出错时调用 xx 方法

```
/**
 * 购买最新的电影票
 * 传入用户 id
 */
@HystrixCommand(fallbackMethod="hystrix")
public Map<String, Object> buyMovie(Integer id){
    Map<String, Object>    result = new HashMap<>();
    //1、查询用户信息
    User userById = getUserById(id);
    //2、查到最新电影票    restTemplate 使用 java 代码来模拟发请求
    Movie movie = restTemplate.getForObject("http://CLOUD-PROVIDER-MOVIE/movie", Movie.class);

    result.put("user", userById);
    result.put("movie", movie);
    return result;
}
```

8.2.4 在本类编写 xxx 方法，方法的参数和原来一样即可

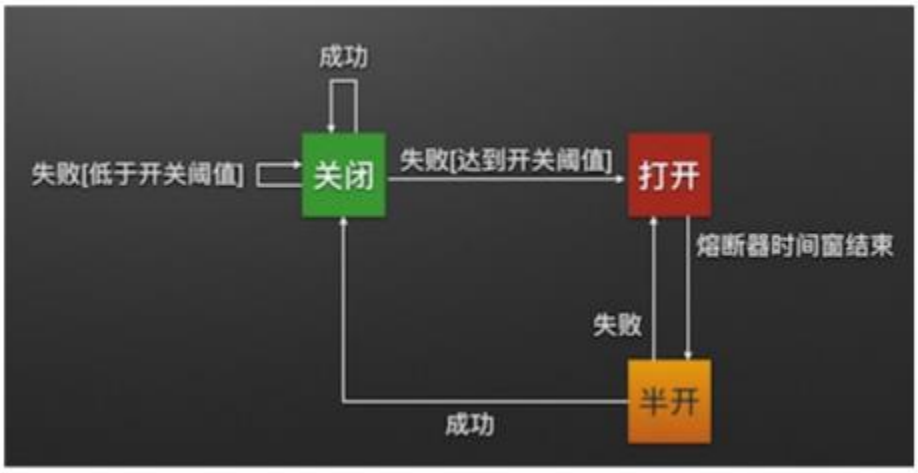
```
public Map<String, Object> hystrix(Integer id){
    User user = new User();
    user.setId(-1);
```

```
    user.setUserName("未知用户");


    Movie movie = new Movie();
    movie.setId(-100);
    movie.setMovieName("无此电影");


    Map<String, Object>    result = new HashMap<>();
    result.put("user", user);
    result.put("movie", movie);
    return result;
}
```

8.2.5 测试正常调用&停止 user 服务，测试异常调用&启动 user 服务过一段时间测试是否正常

http://localhost:9000/buyMovie?id=1

8.2.6 熔断器原理



8.2.7 我们悄悄的启动一个服务，刷刷浏览器会怎样?

让子弹飞一分钟再看结果

8.3 使用 Feign+Hystrix 组合：cloud-consumer-user-feign

8.3.1 引入 Hystrix

```
<!--  引入 hystrix 进行服务熔断  -->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

8.3.2 开启断路保护功能@EnableCircuitBreaker

8.3.3 开启 Feign 对 Hystrix 支持

```
feign:
  hystrix:
    enabled: true #默认 false
```

8.3.4 Feign 已经集成了 Hystrix，使用起来非常简单

@FeignClient(name="provider-user",fallback="异常处理类")

```
/*使用 Hystrix 进行服务的熔断
* 1）、引入 Hystrix 的 starter
* 2）、开启 xxx 功能 ： @EnableCircuitBreaker
* 3）、@FeignClient(value="CLOUD-PROVIDER-MOVIE",fallback=指定这个接口的异常处理类（异常处理类必须实现这个接口）)
*/
@FeignClient(value="CLOUD-PROVIDER-MOVIE",fallback=MovieFeignExceptionHandlerService.class)
public interface MovieServiceFeign {
  //  未来这个接口就会调用很多方法，定制每一个方法远程出错如何返回兜底 mock 数据；
  @GetMapping("/movie")
  public Movie getNewMovie();
}
```

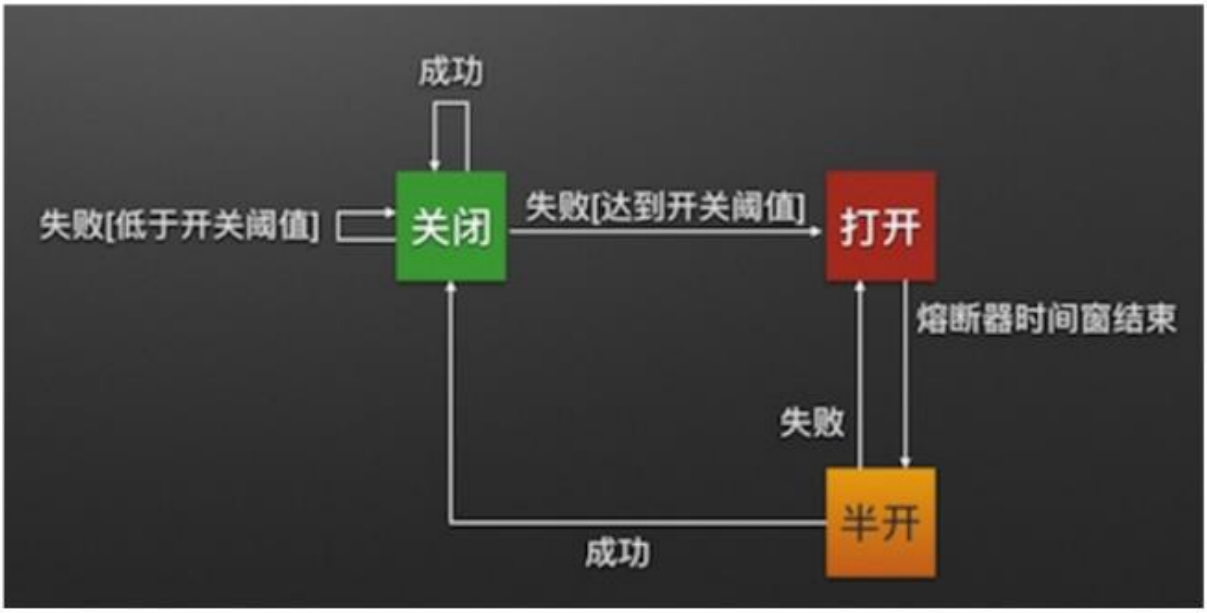
8.3.5 fallback="异常处理类"指定的异常处理类实现这个类的接口即可，并且放在容器中

```
package com.atguigu.feign.exception;


import org.springframework.stereotype.Component;


import com.atguigu.bean.Movie;
import com.atguigu.feign.MovieServiceFeign;


@Component
public class MovieFeignExceptionHandlerService implements MovieServiceFeign{


    /**
     * 远程这个方法调用出问题就会调用此方法
     */
    @Override
    public Movie getNewMovie() {
        Movie movie = new Movie();
        movie.setId(-100);
        movie.setMovieName("无此电影呀...");
        return movie;
    }
}
```

8.3.6 测试正常调用&停止 user 服务，测试异常调用&启动 user 服务过一段时间测试是否正常

http://localhost:7000/buyMovie?id=1

第九章 SpringCloud-HelloWorld 案例开发-Hystrix Dashboard

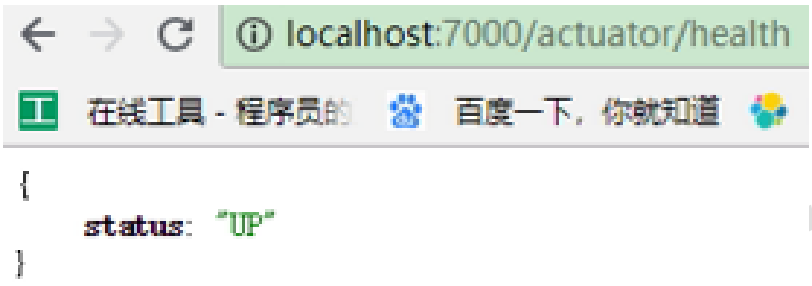


除了隔离依赖服务的调用以外，Hystrix 还提供了近实时的监控，Hystrix 会实时、累加地记录所有关于 HystrixCommand 的执行信息，包括每秒执行多少请求，多少成功，多少失败等。Netflix 通过 **hystrix-metrics-event-stream** 项目实现了对以上指标的监控

9.1 引入 actuator（cloud-consumer-user-feign）

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

9.1.1 actuator 是用来监控 SpringBoot 服务的，注意路径问题，具体的版本可能不一样，参考启动日志



9.1.2 可提供的监控服务为

HTTP方法	路径	描述	鉴权
GET	/autoconfig	查看自动配置的使用情况	true
GET	/configprops	查看配置属性，包括默认配置	true
GET	/beans	查看bean及其关系列表	true
GET	/dump	打印线程栈	true
GET	/env	查看所有环境变量	true
GET	/env/{name}	查看具体变量值	true
GET	/health	查看应用健康指标	false
GET	/info	查看应用信息	false
GET	/mappings	查看所有url映射	true
GET	/metrics	查看应用基本指标	true
GET	/metrics/{name}	查看具体指标	true
POST	/shutdown	关闭应用	true
GET	/trace	查看基本追踪信息	true

9.2 修改配置文件，暴露数据监控流

```
management:
  endpoints:
    web:
      exposure:
        include: hystrix.stream # 访问/actuator/hystrix.stream 能看到不断更新的监控流
```

访问/actuator/hystrix.stream 可以看到打印

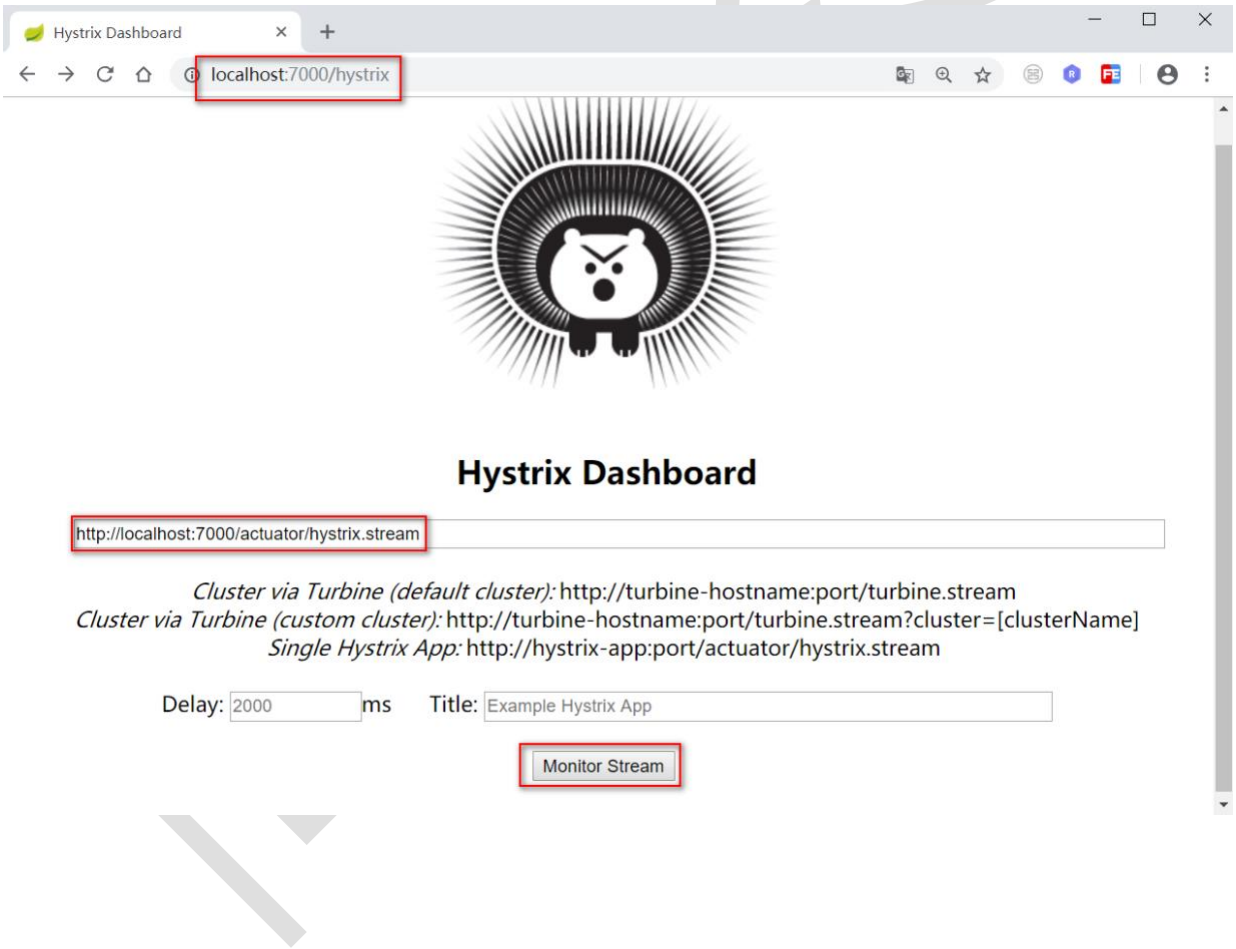
9.3 引入 HystrixDashboard

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
```

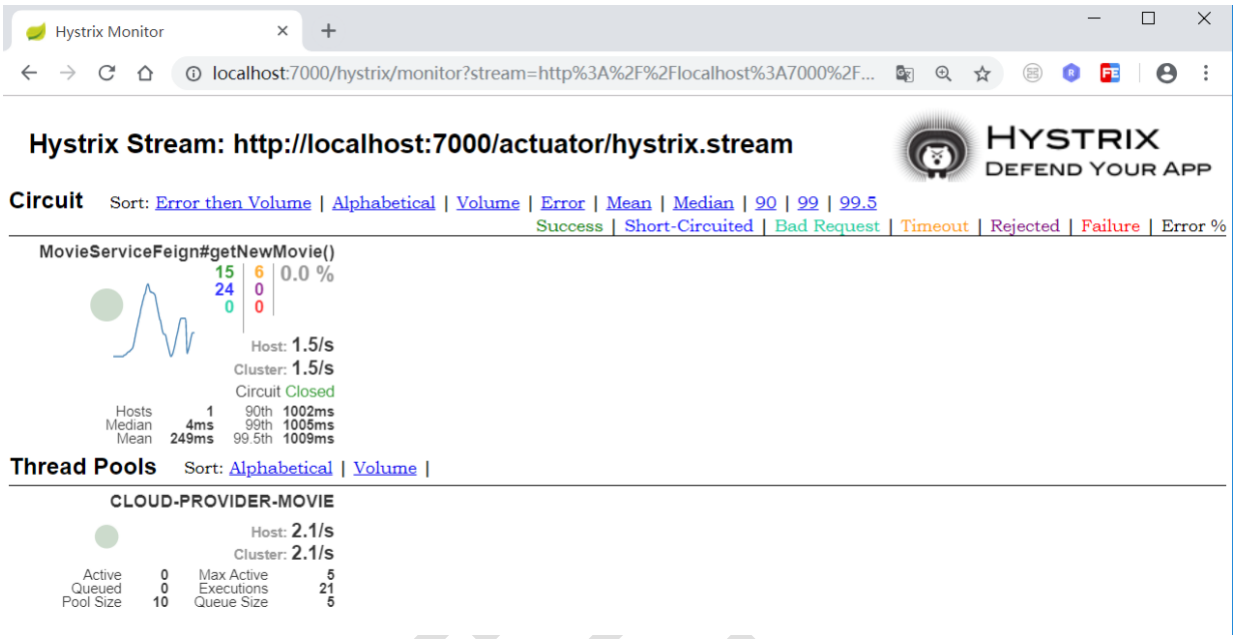
9.4 开启可视化监控功能

9.4.1 @EnableHystrixDashboard

9.4.2 访问当前项目/hystrix，将 Hystrix 的 stream 地址输入(http://localhost:7000/actuator/hystrix.stream)，点击 monitor 按钮即可监控



9.4.3 监控中，服务器正常



9.4.4 监控中，服务器重新启动，断路器打开