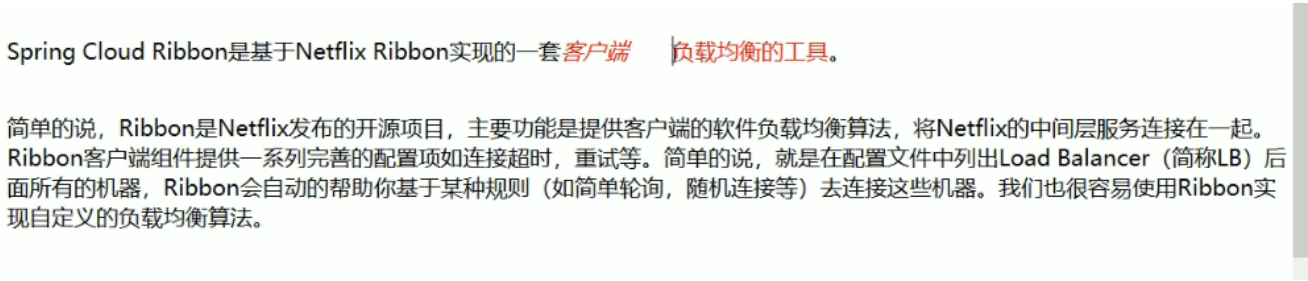
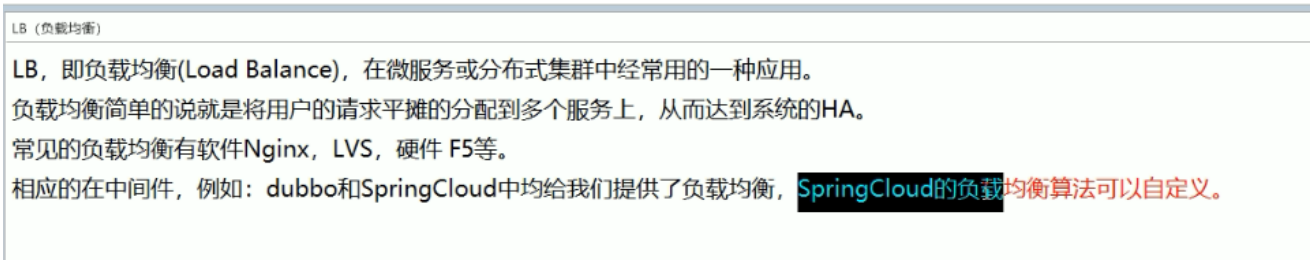
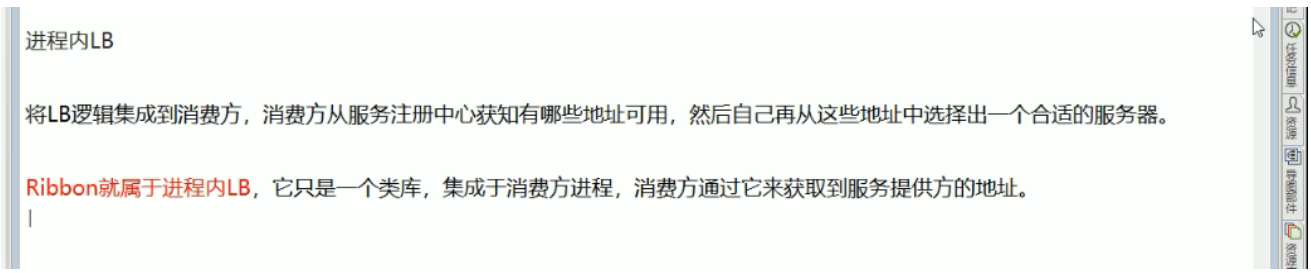
一、ribbon负载均衡

1.ribbon简介







2. 基本配置

修改consumer 80项目，因为ribbon是客户端负载均衡，所以我们修改consumer项目。

1. POM依赖

comsumer 80项目添加如下依赖

```
<!-- Ribbon相关 ，负载均衡-->
        <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.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
```

2. yaml文件

添加配置

```yaml
eureka:
  client:
    register-with-eureka: false #自己不能注册
    service-url:
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

3.修改ConfigBean

getRestTemplete方法上添加@LoadBalanced注解,这个注解模式使用轮询算法

```java
@Configuration //@Configuration配置   ConfigBean = applicationContext.xml
public class ConfigBean {
    @Bean
    @LoadBalanced//Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端       负载均衡的工具。
    public RestTemplate getRestTemplete() {
        return new RestTemplate();
    }
}
```

4. 修改主启动类

添加@EnableEurekaClient注解

```
@SpringBootApplication
@EnableEurekaClient//ribbon做负载均衡时添加的
public class DeptConsumer80_App {

    public static void main(String[] args) {

SpringApplication.run(DeptConsumer80_App.class,
args);
    }

}
```

5.修改访问客户端

使用微服务名称进行远程调用

```
@RestController
public class DeptController_Consumer {
//  private static final String REST_URL_PREFIX
= "http://localhost:8001";
    private static final String REST_URL_PREFIX
= "http://springcloud-dept";

    @Autowired
    private RestTemplate template;
@SuppressWarnings("unchecked")
```

```
9      @RequestMapping(value =
"/consumer/dept/list")
10     public List<Dept> list() {
11
12         return
template.getForObject(REST_URL_PREFIX +
"/dept/list", List.class);
13     }
14 }
```

6.测试

首先启动7001,7002,7003三个服务端项目，然后启动8001项目。最后启动80 项目。测试地址http://localhost/consumer/dept/list。

实现通过微服务名称访问微服务

3. 负载均衡配置

1.仿照8001项目创建两个provider

8002项目yaml配置

```
1 server:
2   port: 8002     # 项目的端口号
3
4 mybatis:
```

```yaml
  config-location: classpath:mybatis/mybatis.cfg.xml        # mybatis配置文件所在路径
  type-aliases-package: com.atguigu.springcloud.entities    # 所有Entity别名类所在包
  mapper-locations:
  - classpath:mybatis/mapper/**/*.xml             # mapper映射文件

spring:
   profiles:
    active:
    - dev
   application:
    name: springcloud-dept  # 当前微服务向外暴露的微服务名称
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource            # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver              # mysql驱动包
    url: jdbc:mysql://localhost:3306/cloudDB02              # 数据库名称
    username: root
    password: 123456
    dbcp2:
```

```
      min-idle: 5
            # 数据库连接池的最小维持连接数
      initial-size: 5
            # 初始化连接数
      max-total: 5
           # 最大连接数
      max-wait-millis: 200
           # 等待连接获取的最大超时时间
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka  #单机版
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
  instance:
    instance-id: deptService8002      #对当前服务起的别名
    prefer-ip-address: true      #我们在eureka服务端查看服务名称的时候：访问路径可以显示IP地址
info:
  app.name: Deptservicecloud
  company.name: www.haoge.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$

```

8003项目yaml配置

```yaml
server:
  port: 8003    # 项目的端口号

mybatis:
  config-location:
classpath:mybatis/mybatis.cfg.xml        #
mybatis配置文件所在路径
  type-aliases-package:
com.atguigu.springcloud.entities    # 所有Entity
别名类所在包
  mapper-locations:
  - classpath:mybatis/mapper/**/*.xml
            # mapper映射文件

spring:
   profiles:
    active:
    - dev
   application:
    name: springcloud-dept  # 当前微服务向外暴露的
微服务名称
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource
          # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver
          # mysql驱动包
```

```
    url: jdbc:mysql://localhost:3306/cloudDB03
          # 数据库名称
    username: root
    password: 123456
    dbcp2:
      min-idle: 5
           # 数据库连接池的最小维持连接数
      initial-size: 5
           # 初始化连接数
      max-total: 5
          # 最大连接数
      max-wait-millis: 200
           # 等待连接获取的最大超时时间
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka
  #单机版
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka
7002.com:7002/eureka/,http://eureka7003.com:7003
/eureka/
  instance:
    instance-id: deptService8003      #对当前服务
起的别名
    prefer-ip-address: true      #我们在eureka服务
端查看服务名称的时候：访问路径可以显示IP地址
info:
```

```
36    app.name: Deptservicecloud
37    company.name: www.haoge.com
38    build.artifactId: $project.artifactId$
39    build.version: $project.version$
40
```

2.测试

分别启动7001,7002,7003,8001,8002,8003,80 项目

链接：http://localhost/consumer/dept/list

效果：ribbon默认使用轮询算法。我们看到依次访问的是8001,8002,8003提供的微服务

4. 切换负载均衡策略

1. ribbon默认提供的负载均衡策略

2. 代码示例

切换ribbon默认的负载均衡策略.ribbon默认使用轮询算法，同时提供了上述7种策略。如果我们需要切换默认负载均衡算法。显式的申明我们相用的算法，并且使用@Bean注解将其加在容器中即可。如下的myRule

```
@Configuration //@Configuration配置   ConfigBean = applicationContext.xml
public class ConfigBean {
    @Bean
    @LoadBalanced//Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端       负载均衡的工具。
    //@LoadBalanced内置7种不同的负载均衡的算法，如果我们不显示的申明我们想要的算法，就使用默认的轮训算法。
    //如果我们显示的声明我们需要的算法，则会替代默认的轮训算法
    public RestTemplate getRestTemplete() {
        return new RestTemplate();
    }
//如果我们要显式的指定自己想要的算法，则改变返回算法的名字即可。例子如下
    @Bean
    public IRule myRule(){
        //return new  RetryRule();//如果服务提供者全部可用，则和轮训算法一样。当某一个服务不可用的时候
```

```
14                                          //查询该服务不可用
几次之后，自动的不会再次查找该服务。在剩下的服务中进
行轮训
15
16              return new RandomRule();//随机算法。
达到的目的，用我们重新选择的随机算法替代默认的轮询。
17      }
18  }
```

5. 自定义负责均衡算法

1. @RibbonClient

主启动类加@RibbonClient(name="springcloud-dept",configuration=MySelfRule.class)注解。表示对springcloud-dept的微服务使用MySelfRule定义的负载均衡算法

```
@RibbonClient(name="springcloud-dept",configuration=MySelfRule.class)
public class DeptConsumer80_App {

    public static void main(String[] args) {

SpringApplication.run(DeptConsumer80_App.class, args);
    }

}
```

2.MySelfRule.java

```
@Configuration
public class MySelfRule {
    @Bean
    public IRule mySelfRuler() {
        return new RandomRule_ldh();
    }
}
```

3. RandomRule_ldh

效果：当前服务器调用五次之后重新随机一个服务器再次调用五次

```

```

```java
public class RandomRule_ldh extends AbstractLoadBalancerRule
{
    Random rand=new Random();
    // total = 0 // 当total==5以后，我们指针才能往下走，
    // index = 0 // 当前对外提供服务的服务器地址，
    // total需要重新置为零，但是已经达到过一个5次，我们的index = 1
    // 分析：我们5次，但是微服务只有8001 8002 8003 三台，OK？
    //
    private int total = 0;            // 总共被调用的次数，目前要求每台被调用5次
    private int currentIndex = 0;    // 当前提供服务的机器号

    public Server choose(ILoadBalancer lb, Object key)
    {
        if (lb == null) {
            return null;
        }
        Server server = null;

        while (server == null) {
            if (Thread.interrupted()) {
                return null;
```

```
            }
            //获得可用的服务器列表
            List<Server> upList =
lb.getReachableServers();
            //获得所有的服务器列表
            List<Server> allList =
lb.getAllServers();

            int serverCount = allList.size();
            if (serverCount == 0) {
                /*
                 * No servers. End regardless of
pass, because subsequent passes only get more
                 * restrictive.
                 */
                return null;
            }

//          int index =
rand.nextInt(serverCount);//
java.util.Random().nextInt(3);
//          server = upList.get(index);


//          private int total = 0;          //
总共被调用的次数，目前要求每台被调用4次
//          private int currentIndex = 0;   //
当前提供服务的机器号
```

```
43              if(total < 4)//如果total小于4，则继续访问这个服务器。否则重新进行随机
44              {
45  //                  currentIndex = rand.nextInt(serverCount);// java.util.Random().nextInt(3);
46                  server = upList.get(currentIndex);//获取将返回的服务器
47                  total++;
48              }else {
49                  total = 0;
50                  currentIndex = rand.nextInt(serverCount);//重新随机
51                  server = upList.get(currentIndex);//获取将返回的服务器
52  //                  if(currentIndex >= upList.size())
53  //                  {
54  //                    currentIndex = 0;
55  //                  }
56              }
57
58
59              if (server == null) {
60                  /*
61                   * The only time this should happen is if the server list were somehow trimmed.
```

```
62                  * This is a transient
condition. Retry after yielding.
63                  */
64                 Thread.yield();
65                 continue;
66             }
67
68             if (server.isAlive()) {//如果服务器可
用，返回服务器地址
69                 return (server);
70             }
71
72             // Shouldn't actually happen.. but
must be transient or a bug.
73             server = null;
74             Thread.yield();
75         }
76
77         return server;
78
79     }
80
81     @Override
82     public Server choose(Object key)
83     {
84         return choose(getLoadBalancer(), key);
85     }
86     @Override
```

```
87      public void initWithNiwsConfig(IClientConfig
    clientConfig)
88      {
89          // TODO Auto-generated method stub
90
91      }
92  }
```