dubbo笔记（3）高可用

一、zookeeper宕机与dubbo直连

现象：zookeeper注册中心宕机，还可以消费dubbo暴露的服务。

原因：**健壮性**

- 监控中心宕掉不影响使用，只是丢失部分采样数据
- 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
- 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
- **注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯**
- 服务提供者无状态，任意一台宕掉后，不影响使用
- 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

高可用：通过设计，减少系统不能提供服务的时间；

dubbo直连：

```
    @Reference(url = "127.0.0.1:20882")
    UserService userService;
```

二、负载均衡机制

在集群负载均衡时，Dubbo 提供了多种均衡策略，缺省为 random 随机调用。

0. 配置

服务端服务级别

```
<dubbo:service interface="..." loadbalance="roundrobin" />
```

客户端服务级别

```
<dubbo:reference interface="..." loadbalance="roundrobin" />
```

服务端方法级别

```
<dubbo:service interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:service>
```

客户端方法级别

```
<dubbo:reference interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:reference>
```

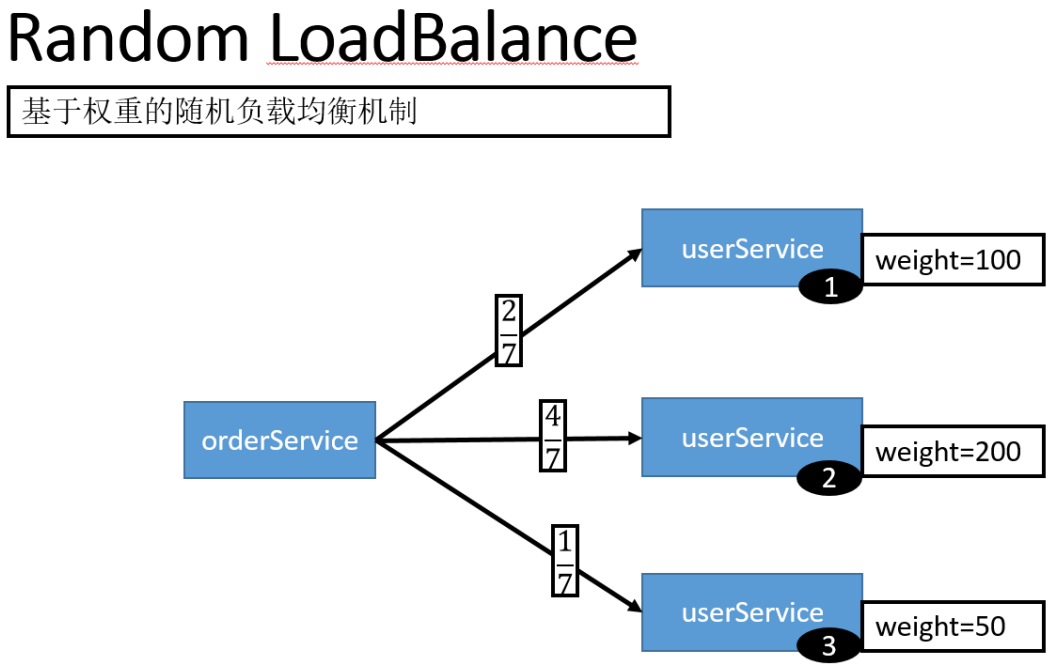
注解方式

```
    @Reference(loadbalance = "roundrobin")
    UserService userService;
```

1. Random LoadBalance

随机，按权重设置随机概率。

在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。



Dubbo控制台调整Provider权重：

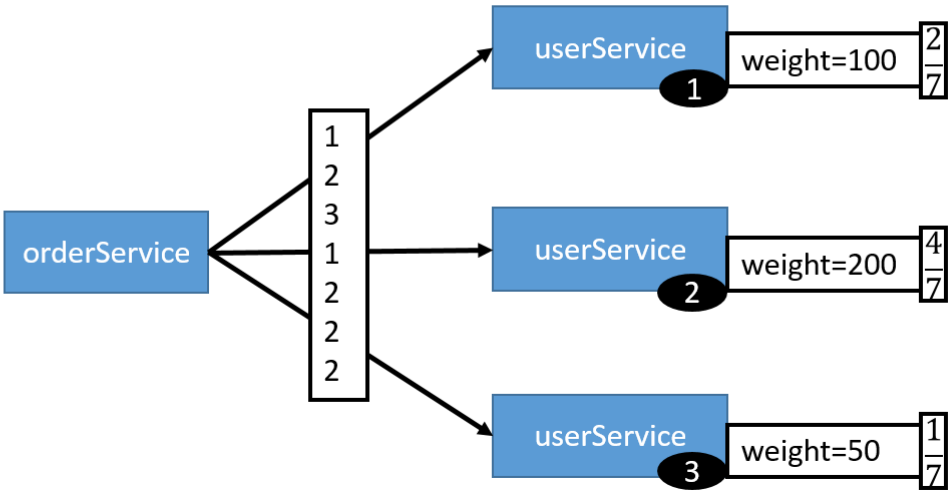
2. RoundRobin LoadBalance

轮循，按公约后的权重设置轮循比率。

存在慢的提供者累积请求的问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

RoundRobin LoadBalance

基于权重的轮询负载均衡机制



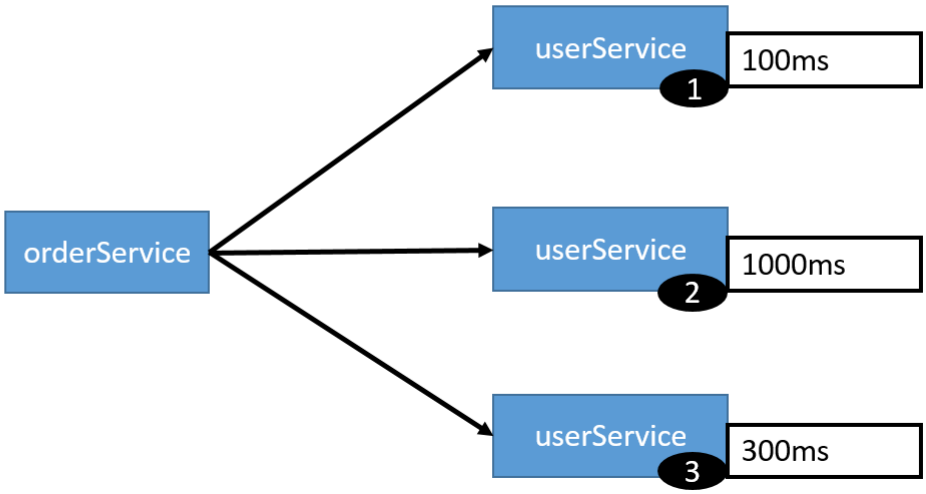
3. LeastActive LoadBalance

最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。

使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

LeastActive LoadBalance

最少活跃数-负载均衡机制



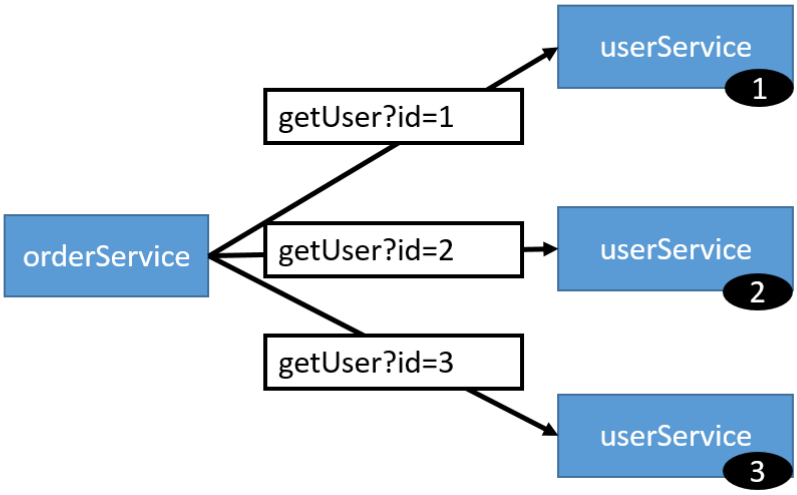
4. ConsistentHash LoadBalance

一致性 Hash，相同参数的请求总是发到同一提供者。

当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。

ConsistentHash LoadBalance

一致性hash-负载均衡机制



算法参见：http://en.wikipedia.org/wiki/Consistent_hashing

三、服务降级

什么是服务降级?

当服务器压力剧增的情况下，根据实际业务情况及流量，对一些服务和页面有策略的不处理或换种简单的方式处理，从而释放服务器资源以保证核心交易正常运作或高效运作。

可以通过服务降级功能临时屏蔽某个出错的非关键服务，并定义降级后的返回策略。

通过Dubbo控制台进行降级：

屏蔽后，将不发起远程调用，直接在客户端返回空对象。

容错后，当远程调用失败（比如超时）时，返回空对象。

四、集群容错&Hystrix

在集群调用失败时，Dubbo 提供了多种容错方案，缺省为 failover 重试。

1. 集群容错模式

- **Failover Cluster**

失败自动切换，当出现失败，重试其它服务器。通常用于读操作，但重试会带来更长延迟。可通过 retries="2" 来设置重试次数(不含第一次)。

重试次数配置如下：

```
<dubbo:service retries="2" />
```

或

```
<dubbo:reference retries="2" />
```

或

```
<dubbo:reference>
  <dubbo:method name="findFoo" retries="2" />
</dubbo:reference>
```

- **Failfast Cluster**

快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作，比如新增记录。

- **Failsafe Cluster**

失败安全，出现异常时，直接忽略。通常用于写入审计日志等操作。

- **Failback Cluster**

失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作。

- **Forking Cluster**

并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多服务资源。可通过 forks="2" 来设置最大并行数。

- **Broadcast Cluster**

广播调用所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地资源信息。

集群模式配置

按照以下示例在服务提供方和消费方配置集群模式

```
<dubbo:service cluster="failsafe" />
```

或

```
<dubbo:reference cluster="failsafe" />
```

2. 整合hystrix

Hystrix 旨在通过控制那些访问远程系统、服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力。Hystrix具备拥有回退机制和断路器功能的线程和信号隔离，请求缓存和请求打包，以及监控和配置等功能。

① 配置spring-cloud-starter-netflix-hystrix

spring boot官方提供了对hystrix的集成，直接在pom.xml里加入依赖：

```
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
            <version>2.2.3.RELEASE</version>
        </dependency>
```

然后在Application类上增加@EnableHystrix来启用hystrix starter：

```
@EnableDubbo
@SpringBootApplication
@EnableHystrix
public class ProviderApplication {
```

② 配置Provider端

在Dubbo的Provider实现方法上增加@HystrixCommand配置，这样子调用就会经过Hystrix代理。

```
    @HystrixCommand
    public List<UserAddress> getUserAddressList(String userId) {
```

③ 配置Consumer端

对于Consumer端，则可以增加一层method调用，并在method上配置@HystrixCommand。当调用出错时，会走到fallbackMethod = "reliable"的调用里。

```
    @Reference
    UserService userService;

    @HystrixCommand(fallbackMethod = "hello")
    public List<UserAddress> initOrder(String userId) {
        System.out.println("userId = " + userId);
        return userService.getUserAddressList(userId);
    }

    public List<UserAddress> hello(String userId) {
        System.out.println("userId = " + userId);
        return new ArrayList<>();
    }
```