搭建 springboot+dubbo 的环境，建议自己搭建，出现问题再参考课堂的源码

负载均衡

负载均衡的背景

到目前为止，dubbo 集成 zookeeper 解决了服务注册以及服务动态感知的问题。那么当服务端存在多个节点的集群时，zookeeper 上会维护不同集群节点，对于客户端而言，他需要一种负载均衡机制来实现目标服务的请求负载。通过负载均衡，可以让每个服务器节点获得适合自己处理能力的负载。

负载均衡可以分为软件负载和硬件负载，在实际开发中，我们基础软件负载比较多，比如 nginx，硬件负载现在用得比较少而且有专门的人来维护。

Dubbo 里面默认就集成了负载均衡的算法和实现，默认提供了 4 中负载均衡实现。

Dubbo 中负载均衡的应用

配置的属性名称： roundrobin/random/ leastactive/ consistenthash

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

可以在服务端配置，也可以在客户端配置。

如果是基于注解，配置如下

```
@Service(loadbalance = "roundrobin")
public class HelloServiceImpl implements IHelloService{
```

或者

```
@Reference(loadbalance = "random")
IHelloService helloService;
```

演示方式

在 run configurations 中，配置多个 springboot application，添加 jvm 参数是的两个程序启动的端口不一样。然后客户端发起多次调用实现请求的负载均衡

-Ddubbo.protocol.port=20881

Dubbo 负载均衡算法

RandomLoadBalance

权重随机算法，根据权重值进行随机负载

它的算法思想很简单。假设我们有一组服务器 servers = [A, B, C]，他们对应的权重为 weights = [5, 3, 2]，权重总和为 10。现在把这些权重值平铺在一维坐标值上，[0, 5) 区间属于服务器 A，[5, 8) 区间属于服务器 B，[8, 10) 区间属于服务器 C。接下来通过随机数生成器生成一个范围在 [0, 10) 之间的随机数，然后计算这个随机数会落到哪个区间上。比如数字 3 会落到服务器 A 对应的区间上，此时返回服务器 A 即可。权重越大的机器，在坐标轴上对应的区间范围就越大，因此随机数生成器生成的数字就会有更大的概率落到此区间内。只要随机数生成器产生的随机数分布性很好，在经过多次选择后，每个服务器被选中的次数比例接近其权重比例

LeastActiveLoadBalance

最少活跃调用数算法，活跃调用数越小，表明该服务提供者效率越高，单位时间内可

处理更多的请求这个是比较科学的负载均衡算法。

每个服务提供者对应一个活跃数 active。初始情况下，所有服务提供者活跃数均为 0。每收到一个请求，活跃数加 1，完成请求后则将活跃数减 1。在服务运行一段时间后，性能好的服务提供者处理请求的速度更快，因此活跃数下降的也越快，此时这样的服务提供者能够优先获取到新的服务请求

ConsistentHashLoadBalance

hash 一致性算法，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

RoundRobinLoadBalance

加权轮询算法

所谓轮询是指将请求轮流分配给每台服务器。举个例子，我们有三台服务器 A、B、C。我们将第一个请求分配给服务器 A，第二个请求分配给服务器 B，第三个请求分配给服务器 C，第四个请求再次分配给服务器 A。这个过程就叫做轮询。轮询是一种无状态负载均衡算法，实现简单，适用于每台服务器性能相近的场景下。但现实情况下，我们并不能保证每台服务器性能均相近。如果我们将等量的请求分配给性能较差的服务器，这显然是不合理的。因此，这个时候我们需要对轮询过程进行加权，以调控每台服务器的负载。经过加权后，每台服务器能够得到的请求数比例，接近或等于他们的权重比。比如服务器 A、B、C 权重比为 5:2:1。那么在 8 次请求中，服务器 A 将收到其中的 5 次请求，服务器 B 会收到其中的 2 次请求，服务器 C 则收到其中的 1

次请求

一致性 hash 算法原理

集群容错

在分布式网络通信中，容错能力是必须要具备的，什么叫容错呢？ 从字面意思来看：容：是容忍， 错：是错误。 就是容忍错误的能力。

我们知道网络通信会有很多不确定因素，比如网络延迟、网络中断、服务异常等，会造成当前这次请求出现失败。当服务通信出现这个问题时，需要采取一定的措施应对。而 dubbo 中提供了容错机制来优雅处理这种错误

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

```
@Service(loadbalance = "random", cluster = "failsafe")
```

Failover Cluster

失败自动切换，当出现失败，重试其它服务器。(缺省)

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)。

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。(2.1.0 开始支持)
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。
在实际应用中 查询语句容错策略建议使用默认 Failover Cluster ，而增删改 建议使用 Failfast Cluster 或者 使用 Failover Cluster（retries="0"） 策略 防止出现数据 重复添加等等其它问题！建议在设计接口时候把查询接口方法单独做一个接口提供查询。

服务降级

降级的概念

当某个非关键服务出现错误时，可以通过降级功能来临时屏蔽这个服务。降级可以有几个层面的分类： 自动降级和人工降级； 按照功能可以分为：读服务降级和写服务降级；

1. 对一些非核心服务进行人工降级，在大促之前通过降级开关关闭哪些推荐内容、评价等对主流程没有影响的功能
2. 故障降级，比如调用的远程服务挂了，网络故障、或者 RPC 服务返回异常。 那么可以直接降级，降级的方案比如设置默认值、采用兜底数据（系统推荐的行为广告挂了，可以提前准备静态页面做返回）等等
3. 限流降级，在秒杀这种流量比较集中并且流量特别大的情况下，因为突发访问量特别大可能会导致系统支撑不了。这个时候可以采用限流来限制访问量。当达到阀值时，后续的请求被降级，比如进入排队页面，比如跳转到错误页（活动太火爆，稍后重试等）

那么，Dubbo 中如何实现服务降级呢？Dubbo 中提供了一个 mock 的配置，可以通过 mock 来实现当服务提供方出现网络异常或者挂掉以后，客户端不抛出异常，而是通过 Mock 数据返回自定义的数据

Dubbo 实现服务降级

在 dubbo-client 端创建一个 mock 类，当出现服务降级时，会被调用

```
public class MockSayHelloService implements IHelloService {
    @Override
    public String sayHello() {
        return "Sorry, 服务端发生异常，被降级啦！";
    }
}
```

修改客户端的注解，增加 mock 配置，以及修改 timeout=1， 表示本次调用的超时时间是 1 毫秒，这样可以模拟出失败的场景

需要配置 cluster=failfast，否则因为默认是 failover 导致客户端会发起 3 次重试，等待的时间比较长

```
@Reference(
        loadbalance = "random",
        mock =
"com.springboot.practice.springbootdubboclient.MockSayHelloService",
        timeout =1000,
        cluster = "failfast")
IHelloService helloService;
```

启动时检查

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题，默认 check="true"。

可以通过 check="false" 关闭检查，比如，测试时，有些服务不关心，或者出现了循环

依赖，必须有一方先启动。

➢ registry、reference、consumer 都可以配置 check 这个属性.

```
@Reference(
        loadbalance = "random",
        mock =
"com.springboot.practice.springbootdubboclient.MockSayHelloService",
        timeout =1000,
        cluster = "failfast",
        check=false)
IHelloService helloService;
```

多版本支持

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。

可以按照以下的步骤进行版本迁移：

1. 在低压力时间段，先升级一半提供者为新版本
2. 再将所有消费者升级为新版本
3. 然后将剩下的一半提供者升级为新版本

主机绑定

默认的主机绑定方式

1. 通过 LocalHost.getLocalHost()获取本机地址。

2. 如果是 127.*等 loopback（环路地址）地址，则扫描各网卡，获取网卡 IP。
    1. 如果是 springboot，修改配置：dubbo.protocol.host=""
    2. 如果注册地址获取不正确，可以通过在 dubbo.xml 中加入主机地址的配置

```
<dubbo:protocol host="205.182.23.201">
```

缺省主机端口

dubbo: 20880

rmi: 1099

http: 80

hessian: 80

webservice: 80

memcached: 11211

redis: 6379

Dubbo 新的功能

动态配置规则

动态配置是 Dubbo2.7 版本引入的一个新的功能，简单来说，就是把 dubbo.properties 中的属性进行集中式存储，存储在其他的服务器上。
那么如果需要用到集中式存储，那么还需要一些配置中心的组件来支撑。
目前 Dubbo 能支持的配置中心有：apollo、nacos、zookeeper

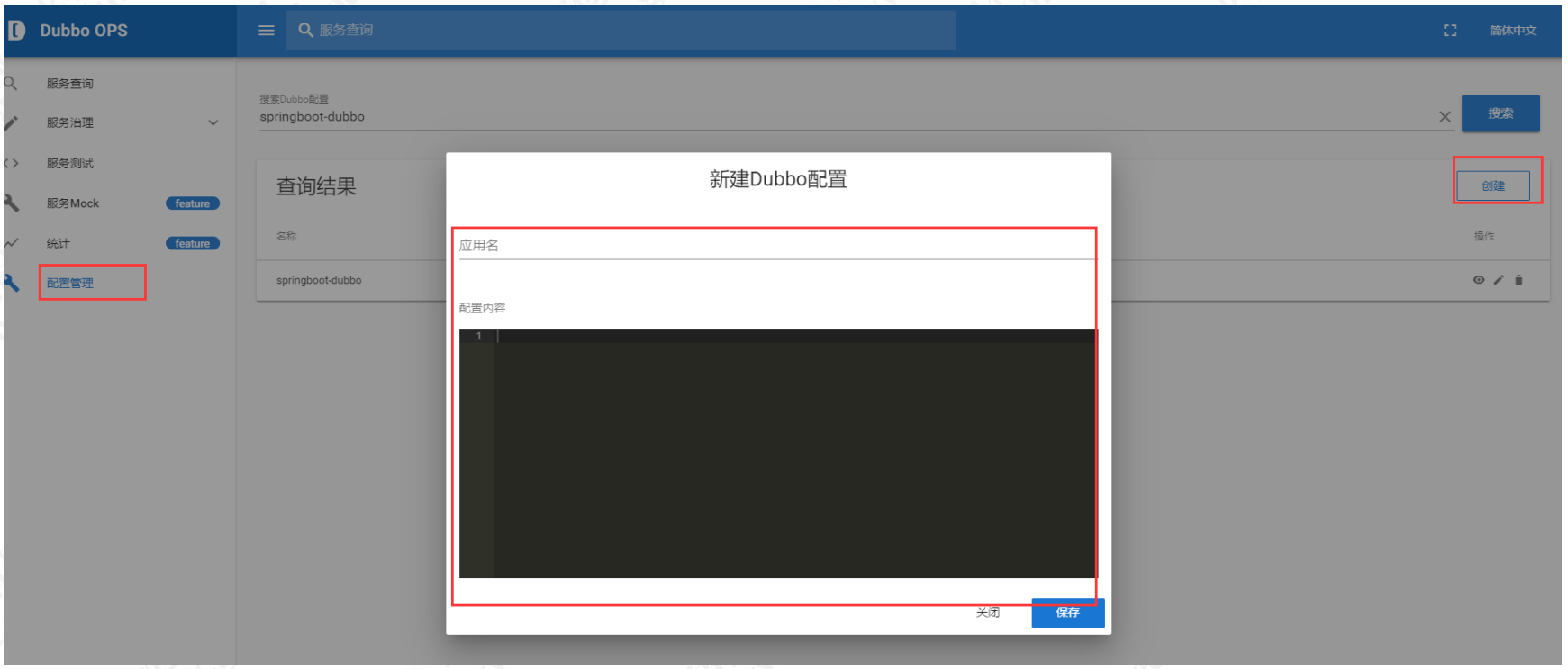
其实，从另外一个角度来看，我们之前用 zookeeper 实现服务注册和发现，本质上就

是使用 zookeeper 实现了配置中心，这个配置中心只是维护了服务注册和服务感知的功能。在 2.7 版本中，dubbo 对配置中心做了延展，出了服务注册之外，还可以把其他的数据存储在 zookeeper 上，从而更好的进行维护

在 dubboadmin 添加配置

应用名称可以是 global，或者对应当前服务的应用名，如果是 global 表示全局配置，针对所有应用可见

配置的内容，实际就是 dubbo.properties 中配置的基本信息。只是同意存储在了 zookeeper 上



本地的配置文件添加配置中心

在 application.properties 中添加配置中心的配置项，app-name 对应的是上一步创建的配置项中的应用名.

dubbo.config-center.address=zookeeper://192.168.13.106:2181

dubbo.config-center.app-name=spring-boot-provider

需要注意的是，存在于配置中心上的配置项，本地仍然需要配置一份。所以下面这些配置一定要加上。否则启动不了。这样做的目的是保证可靠性

dubbo.application.name=springboot-dubbo

dubbo.protocol.port=20880

dubbo.protocol.name=dubbo

dubbo.registry.address=zookeeper://192.168.13.102:2181?backup=192.168.13.103:2181,192.168.13.104:2181

- 课堂演示不成功,是因为 dubbo.application.name 和 dubbo.config-center.app-name 两个名字不一致导致的。我猜测读取配置是根据 dubbo.application.name 来实现的

配置的优先级

引入配置中心后，配置的优先级就需要关注了，默认情况下，外部配置的优先级最高，也就是意味着配置中心上的配置会覆盖本地的配置。当然我们也可以调整优先级

dubbo.config-center.highest-priority=false

配置中心的原理

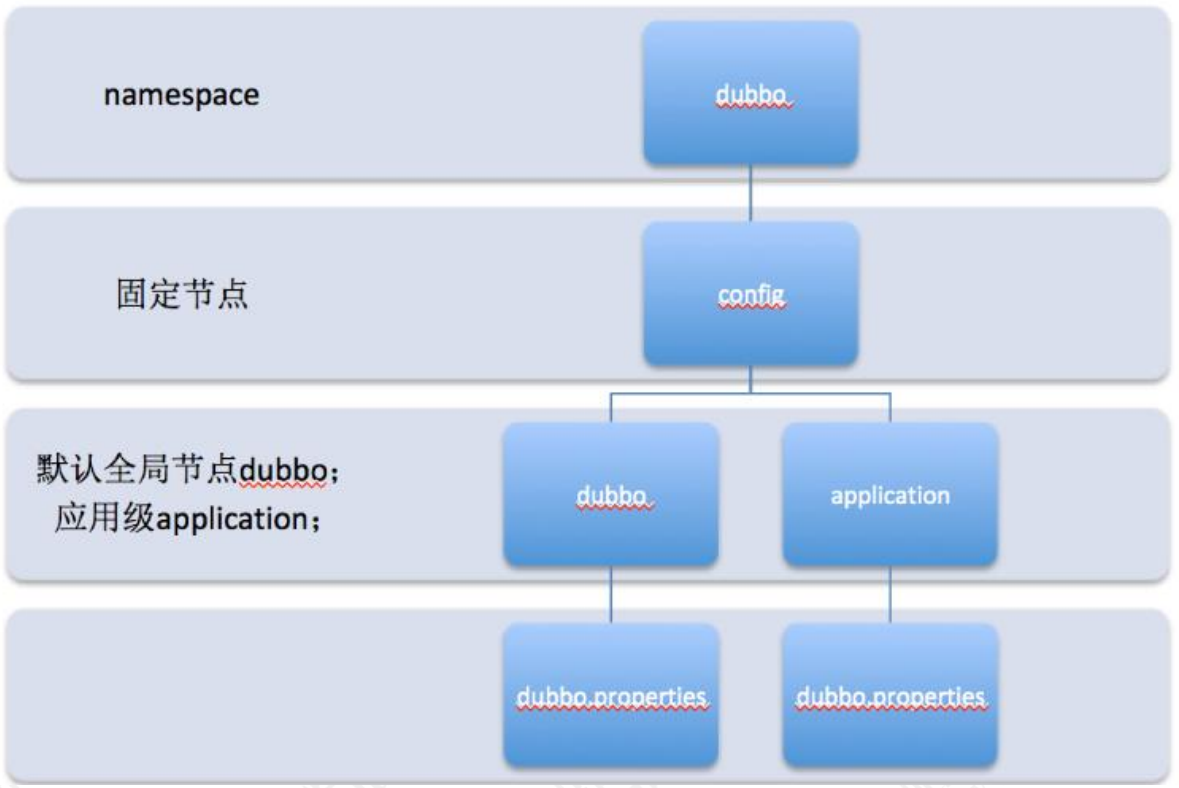
默认所有的配置都存储在/dubbo/config 节点，具体节点结构图如下。

namespace，用于不同配置的环境隔离。

config，Dubbo 约定的固定节点，不可更改，所有配置和服务治理规则都存储在此节点下。

dubbo/application，分别用来隔离全局配置、应用级别配置：dubbo 是默认 group 值，application 对应应用名

dubbo.properties，此节点的 node value 存储具体配置内容



元数据中心

Dubbo2.7 的另外一个新的功能，就是增加了元数据的配置。

在 Dubbo2.7 之前，所有的配置信息，比如服务接口名称、重试次数、版本号、负载策略、容错策略等等，所有参数都是基于 url 形式配置在 zookeeper 上的。这种方式会造成一些问题

1. url 内容过多，导致数据存储空间增大
2. url 需要涉及到网络传输，数据量过大会造成网络传输过慢
3. 网络传输慢，会造成服务地址感知的延迟变大，影响服务的正常响应

服务提供者这边的配置参数有 30 多个，有一半是不需要作为注册中心进行存储和粗暗地的。而消费者这边可配置的参数有 25 个以上，只有个别是需要传递到注册中心的。所以，在 Dubbo2.7 中对元数据进行了改造，简单来说，就是把属于服务治理的数据发布到注册中心，其他的配置数据统一发布到元数据中心。这样一来大大降低了注册中心的负载。

元数据中心配置

元数据中心目前支持 redis 和 zookeeper。官方推荐是采用 redis。毕竟 redis 本身对于非结构化存储的数据读写性能比较高。当然，也可以使用 zookeeper 来实现。

在配置文件中添加元数据中心的地址

```
dubbo.metadata-report.address=zookeeper://192.168.13.106:2181
dubbo.registry.simplified=true  //注册到注册中心的 URL 是否采用精简模式的（与低版本兼容）
```