高性能：负载均衡的常见算法有哪些？

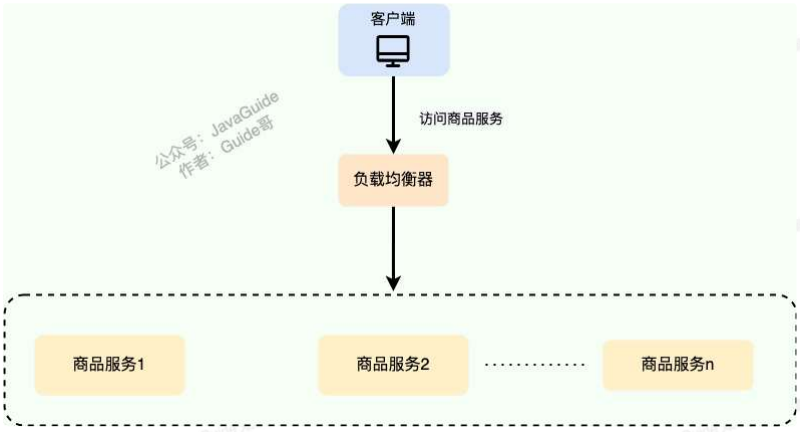
相关面试题：

- 服务端负载均衡一般怎么做？
- 四层负载均衡和七层负载均衡的区别？
- 负载均衡的常见算法有哪些？
- 七层负载均衡常见解决方案有哪些？
- 客户端负载均衡的常见解决方案有哪些？

什么是负载均衡？

负载均衡 指的是将用户请求分摊到不同的服务器上处理，以提高系统整体的并发处理能力以及可靠性。负载均衡服务可以有由专门的软件或者硬件来完成，一般情况下，硬件的性能更好，软件的价格更便宜（后文会详细介绍到）。

下图是《Java 面试指北》 <https://mp.weixin.qq.com/s?__biz=Mzg2OTA0Njk0OA==&mid=2247519384&idx=1&sn=bc7e71af75350b755f04ca4178395b1a&chksm=cea1c353f9d64a458f797696d4144b4d6e58639371a4612b8e4d106d83a66d2289e7b2cd7431&token=660789642&lang=zh_CN&scene=21#wechat_redirect> 「高并发篇」中的一篇文章的配图，从图中可以看出，系统的商品服务部署了多份在不同的服务器上，为了实现访问商品服务请求的分流，我们用到了负载均衡。



负载均衡是一种比较常用且实施起来较为简单的提高系统并发能力和可靠性的手段，不论是单体架构的系统还是微服务架构的系统几乎都会用到。

负载均衡通常分为哪两种？

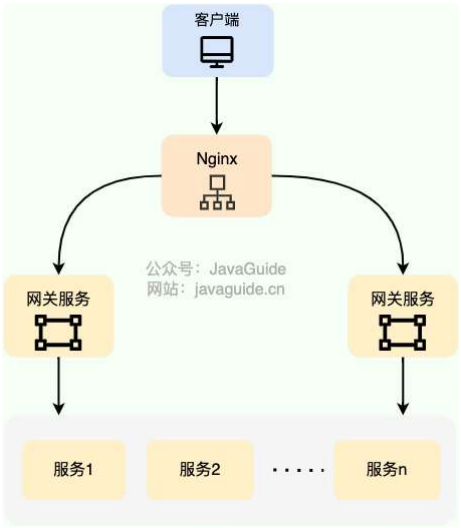
负载均衡可以简单分为 **服务端负载均衡** 和 **客户端负载均衡** 这两种。

服务端负载均衡涉及到的知识点更多，工作中遇到的也比较多，因为，我会花更多时间来介绍。

服务端负载均衡

服务端负载均衡 主要应用在 **系统外部请求** 和 **网关层** 之间，可以使用 **软件** 或者 **硬件** 实现。

下图是我画的一个简单的基于 Nginx 的服务端负载均衡示意图：



硬件负载均衡 通过专门的硬件设备（比如 **F5、A10、Array** ）实现负载均衡功能。

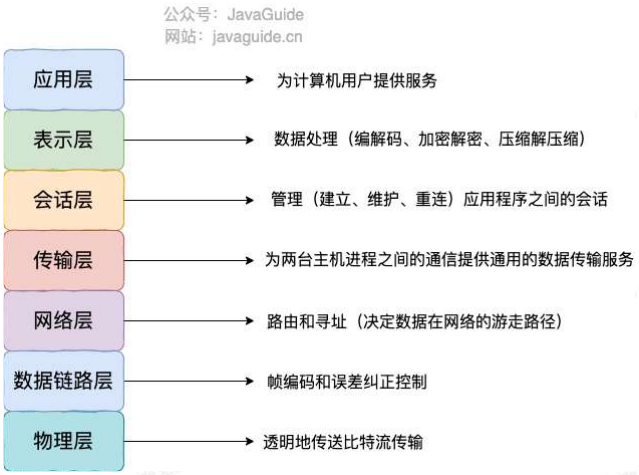
硬件负载均衡的优势是性能很强且稳定，缺点就是实在是太贵了。像基础款的 F5 最低也要 20 多万，绝大部分公司是根本负担不起的，业务量不大的话，真没必要非要去弄个硬件来做负载均衡，用软件负载均衡就足够了！

在我们日常开发中，一般很难接触到硬件负载均衡，接触的比较多的还是 **软件负载均衡** 。软件负载均衡通过软件（比如 **LVS、Nginx、HAproxy** ）实现负载均衡功能，性能虽然差一些，但价格便宜啊！像基础款的 Linux 服务器也就几千，性能好一点的 2~3 万的就很不错了。

根据 OSI 模型，服务端负载均衡还可以分为：

- 二层负载均衡
- 三层负载均衡
- 四层负载均衡
- 七层负载均衡

最常见的是四层和七层负载均衡，因此，本文也是重点介绍这两种负载均衡。



- **四层负载均衡** 工作在 OSI 模型第四层，也就是传输层，这一层的主要协议是 TCP/UDP，负载均衡器在这一层能够看到数据包里的源端口地址以及目的端口地址，会基于这些信息通过一定的负载均衡算法将数据包转发到后端真实服务器。
- **七层负载均衡** 工作在 OSI 模型第七层，也就是应用层，这一层的主要协议是 HTTP 。这一层的负载均衡比四层负载均衡路由网络请求的方式更加复杂，它会读取报文的数据部分（比如说我们的 HTTP 部分的报文），然后根据读取到的数据内容（如 URL、Cookie）做出负载均衡决策。

七层负载均衡比四层负载均衡会消耗更多的性能，不过，也相对更加灵活，能够更加智能地路由网络请求，比如说你可以根据请求的内容进行优化如缓存、压缩、加密。

简单来说，**四层负载均衡性能更强，七层负载均衡功能更强！**

在工作中，我们通常会使用 **Nginx** 来做七层负载均衡，LVS(Linux Virtual Server 虚拟服务器， Linux 内核的 4 层负载均衡)来做四层负载均衡。关于 Nginx 的常见知识点总结，《Java 面试指北》<https://mp.weixin.qq.com/s?__biz=Mzg2OTA0Njk0OA==&mid=2247519384&idx=1&sn=bc7e71af75350b755f04ca4178395b1a&chksm=cea1c353f9d64a458f797696d4144b4d6e58639371a4612b8e4d106d83a66d2289e7b2cd7431&token=660789642&lang=zh_CN&scene=21#wechat_redirect> 中「技术面试题篇」中已经有对应的内容了，感兴趣的小伙伴可以去看看。

不过，LVS 这个绝大部分公司真用不上，像阿里、百度、腾讯、eBay 等大厂才会使用到，用的最多的还是 Nginx。

客户端负载均衡

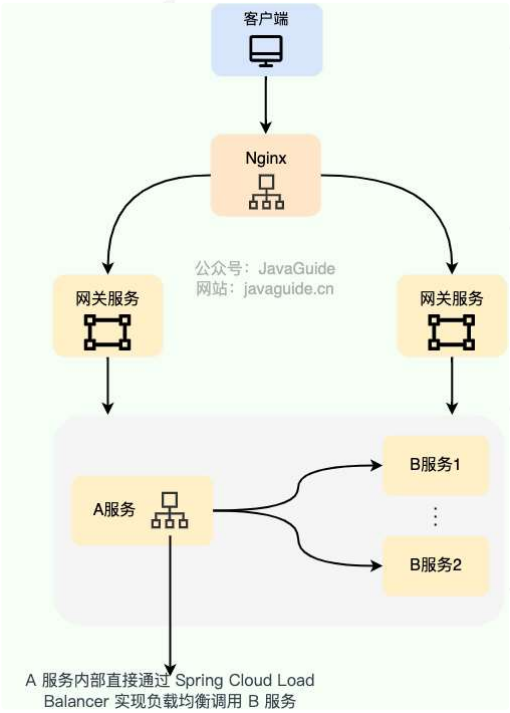
客户端负载均衡 主要应用于系统内部的不同的服务之间，可以使用现成的负载均衡组件来实现。

在客户端负载均衡中，客户端会自己维护一份服务器的地址列表，发送请求之前，客户端会根据对应的负载均衡算法来选择具体某一台服务器处理请求。

客户端负载均衡器和服务运行在同一个进程或者说 Java 程序里，不存在额外的网络开销。不过，客户端负载均衡的实现会受到编程语言的限制，比如说 Spring Cloud Load Balancer 就只能用于 Java 语言。

Java 领域主流的微服务框架 Dubbo、Spring Cloud 等都内置了开箱即用的客户端负载均衡实现。Dubbo 属于是默认自带了负载均衡功能，Spring Cloud 是通过组件的形式实现的负载均衡，属于可选项，比较常用的是 Spring Cloud Load Balancer（官方，推荐） 和 Ribbon（Netflix，已被启用）。

下图是我画的一个简单的基于 Spring Cloud Load Balancer（Ribbon 也类似） 的客户端负载均衡示意图：



负载均衡常见的算法有哪些?

随机法

随机法 是最简单粗暴的负载均衡算法。

如果没有配置权重的话，所有的服务器被访问到的概率都是相同的。如果配置权重的话，权重越高的服务器被访问的概率就越大。

未加权重的随机算法适合于服务器性能相近的集群，其中每个服务器承载相同的负载。加权随机算法适合于服务器性能不等的集群，权重的存在可以使请求分配更加合理化。

不过，随机算法有一个比较明显的缺陷：部分机器在一段时间之内无法被随机到，毕竟是概率算法，就算是大家权重一样， 也可能会出现这种情况。

于是，**轮询法** 来了！

轮询法

轮询法是挨个轮询服务器处理，也可以设置权重。

如果没有配置权重的话，每个请求按时间顺序逐一分配到不同的服务器处理。如果配置权重的话，权重越高的服务器被访问的次数就越多。

未加权重的轮询算法适合于服务器性能相近的集群，其中每个服务器承载相同的负载。加权轮询算法适合于服务器性能不等的集群，权重的存在可以使请求分配更加合理化。

一致性 Hash 法

相同参数的请求总是发到同一台服务器处理，比如同个 IP 的请求。

最小连接法

当有新的请求出现时，遍历服务器节点列表并选取其中活动连接数最小的一台服务器来响应当前请求。活动连接数可以理解为当前正在处理的请求数。

最小连接法可以尽可能最大地使请求分配更加合理化，提高服务器的利用率。不过，这种方法实现起来也最复杂，需要监控每一台服务器处理的请求连接数。

七层负载均衡可以怎么做?

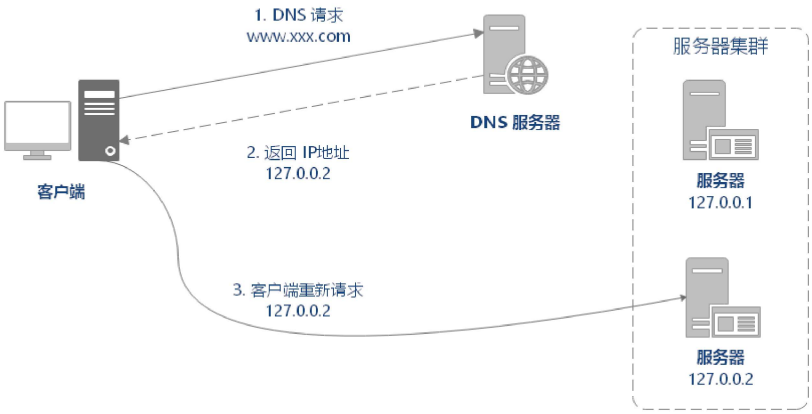
简单介绍两种项目中常用的七层负载均衡解决方案：DNS 解析和反向代理。

除了我介绍的这两种解决方案之外，HTTP 重定向等手段也可以用来实现负载均衡，不过，相对来说，还是 DNS 解析和反向代理用的更多一些，也更推荐一些。

DNS 解析

DNS 解析是比较早期的七层负载均衡实现方式，非常简单。

DNS 解析实现负载均衡的原理是这样的：在 DNS 服务器中为同一个主机记录配置多个 IP 地址，这些 IP 地址对应不同的服务器。当用户请求域名的时候，DNS 服务器采用轮询算法返回 IP 地址，这样就实现了轮询版负载均衡。



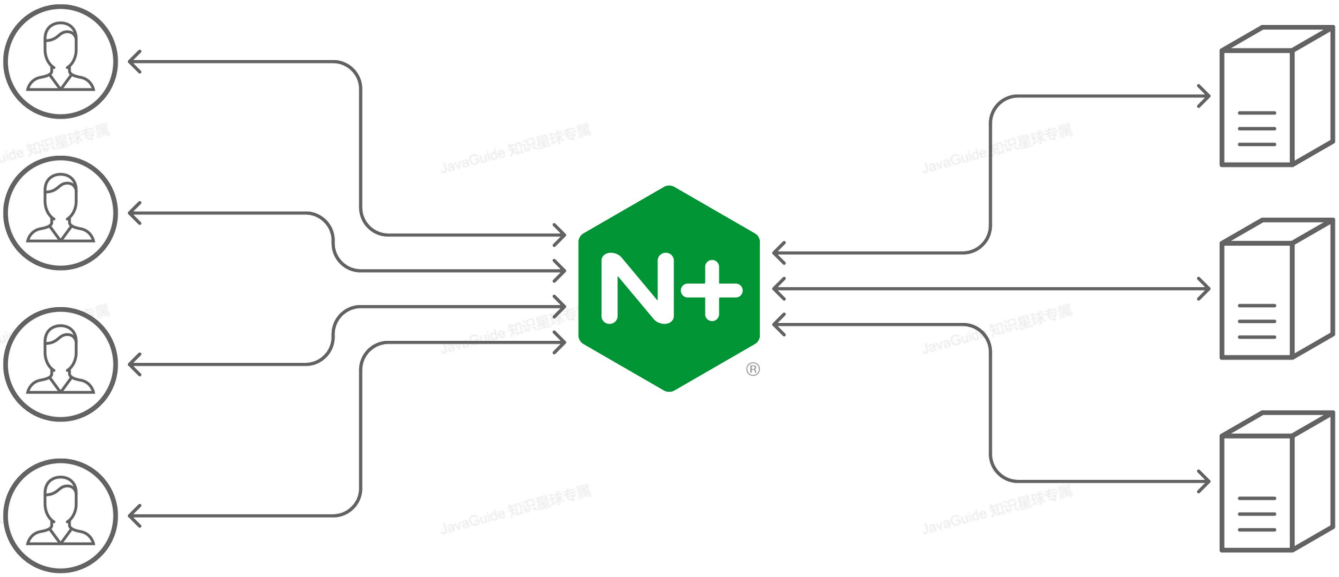
现在的 DNS 解析几乎都支持 IP 地址的权重配置，这样的话，在服务器性能不等的集群中请求分配会更加合理化。像我自己目前正在用的阿里云 DNS 就支持权重配置。

反向代理

客户端将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器，获取数据后再返回给客户端。对外暴露的是反向代理服务器地址，隐藏了真实服务器 IP 地址。反向代理“代理”的是目标服务器，这一个过程对于客户端而言是透明的。

Nginx 就是最常用的反向代理服务器，它可以将接收到的客户端请求以一定的规则（负载均衡策略）均匀地分配到这个服务器集群中所有的服务器上。

反向代理负载均衡同样属于七层负载均衡。



客户端负载均衡通常是怎么做的?

我们上面也说了，客户端负载均衡可以使用现成的负载均衡组件来实现。

Netflix Ribbon 和 **Spring Cloud Load Balancer** 就是目前 Java 生态最流行的两个负载均衡组件。

Ribbon 是老牌负载均衡组件，由 Netflix 开发，功能比较全面，支持的负载均衡策略也比较多。 Spring Cloud Load Balancer 是 Spring 官方为了取代 Ribbon 而推出的，功能相对更简单一些，支持的负载均衡也少一些。

Ribbon 支持的 7 种负载均衡策略：

- `RandomRule` ：随机策略。
- `RoundRobinRule` （默认） ：轮询策略
- `WeightedResponseTimeRule` ：权重（根据响应时间决定权重）策略
- `BestAvailableRule` ：最小连接数策略
- `RetryRule` ：重试策略（按照轮询策略来获取服务，如果获取的服务实例为 null 或已经失效，则在指定的时间之内不断地进行重试来获取服务，如果超过指定时间依然没获取到服务实例则返回 null）
- `AvailabilityFilteringRule` ：可用敏感性策略（先过滤掉非健康的服务实例，然后再选择连接数较小的服务实例）
- `ZoneAvoidanceRule` ：区域敏感性策略（根据服务所在区域的性能和服务的可用性来选择服务实例）

Spring Cloud Load Balancer 支持的 2 种负载均衡策略：

- `RandomLoadBalancer` ：随机策略
- `RoundRobinLoadBalancer` （默认） ：轮询策略

Java

```java
public class CustomLoadBalancerConfiguration {

    @Bean
    ReactorLoadBalancer<ServiceInstance> randomLoadBalancer(Environment environment,
            LoadBalancerClientFactory loadBalancerClientFactory) {
        String name = environment.getProperty(LoadBalancerClientFactory.PROPERTY_NAME);
        return new RandomLoadBalancer(loadBalancerClientFactory
                .getLazyProvider(name, ServiceInstanceListSupplier.class),
                name);
    }
}
```

不过，Spring Cloud Load Balancer 支持的负载均衡策略其实不止这两种， `ServiceInstanceListSupplier` 的实现类同样可以让其支持类似于 Ribbon 的负载均衡策略。这个应该是后续慢慢完善引入的，不看官方文档还真发现不了，所以说阅读官方文档真的很重要！

这里举两个官方的例子：

- `ZonePreferenceServiceInstanceListSupplier` ： 实现基于区域的负载平衡
- `HintBasedServiceInstanceListSupplier` ： 实现基于 hint 提示的负载均衡

```java
public class CustomLoadBalancerConfiguration {
    // 使用基于区域的负载平衡方法
    @Bean
    public ServiceInstanceListSupplier discoveryClientServiceInstanceListSupplier(
            ConfigurableApplicationContext context) {
        return ServiceInstanceListSupplier.builder()
                .withDiscoveryClient()
                .withZonePreference()
                .withCaching()
                .build(context);
    }
}
```

关于Spring Cloud Load Balancer更详细更新的介绍，推荐大家看看官方文档：https://docs.spring.io/spring-cloud-commons/docs/current/reference/html/#spring-cloud-loadbalancer <https://docs.spring.io/spring-cloud-commons/docs/current/reference/html/#spring-cloud-loadbalancer> ，一切以官方文档为主。

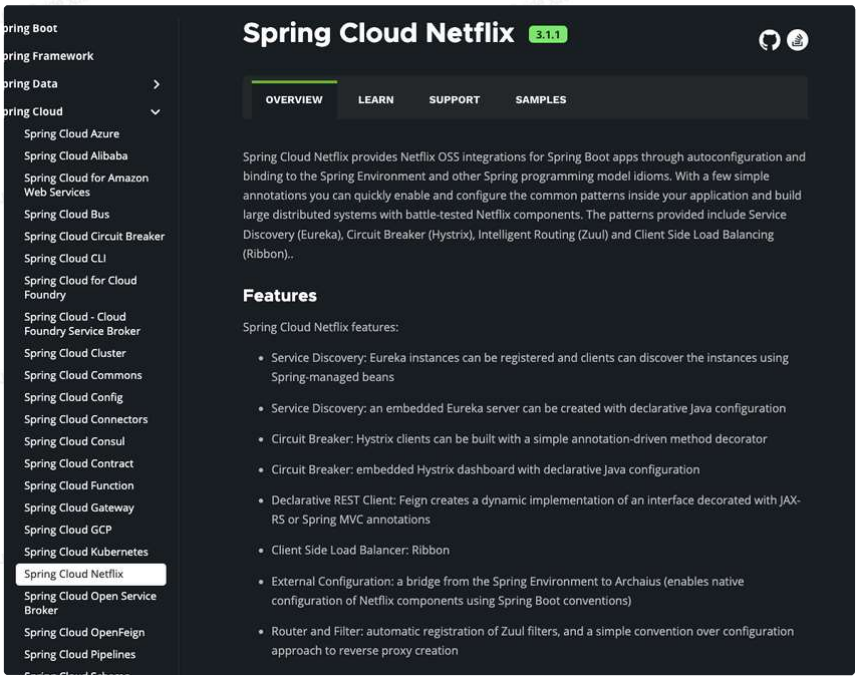
轮询策略基本可以满足绝大部分项目的需求，我们的实际项目中如果没有特殊需求的话，通常使用的就是默认的轮询策略。并且，Ribbon 和 Spring Cloud Load Balancer 都支持自定义负载均衡策略。

个人建议如非必需 Ribbon 某个特有的功能或者负载均衡策略的话，就优先选择 Spring 官方提供的 Spring Cloud Load Balancer。

最后再说说为什么我不太推荐使用 Ribbon 。

Spring Cloud 2020.0.0 版本移除了 Netflix 除 Eureka 外的所有组件。Spring Cloud Hoxton.M2 是第一个支持 Spring Cloud Load Balancer 来替代 Netfix Ribbon 的版本。

我们早期学习微服务，肯定接触过 Netflix 公司开源的 Feign、Ribbon、Zuul、Hystrix、Eureka 等知名的微服务系统构建所必须的组件，直到现在依然有非常非常多的公司在使用这些组件。不夸张地说，Netflix 公司引领了 Java 技术栈下的微服务发展。



那为什么 Spring Cloud 这么急着移除 Netflix 的组件呢？ 主要是因为在 2018 年的时候，Netflix 宣布其开源的核心组件 Hystrix、Ribbon、Zuul、Eureka 等进入维护状态，不再进行新特性开发，只修 BUG。于是，Spring 官方不得不考虑移除 Netflix 的组件。

Spring Cloud Alibaba 是一个不错的选择，尤其是对于国内的公司和个人开发者来说。

参考

- 干货 | eBay 的 4 层软件负载均衡实现：https://mp.weixin.qq.com/s/bZMxLTECOK3mjdgiLbHj-g <https://mp.weixin.qq.com/s/bZMxLTECOK3mjdgiLbHj-g>
- HTTP Load Balancing（Nginx 官方文档）：https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/ <https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>
- 深入浅出负载均衡 - vivo 互联网技术：https://www.cnblogs.com/vivotech/p/14859041.html <https://www.cnblogs.com/vivotech/p/14859041.html>