核心知识①

# 负载均衡

负载均衡（load balancing）是用来在多台服务器（集群）、网络连接、CPU、磁盘驱动器或其他资源中分配负载，以达到最佳化资源使用、最大化吞吐率、最小化响应时间，同时避免过载的目的技术手段。从本质上来说，负载均衡也是一种程序代码。

负载均衡最重要的一个应用是利用多台服务器提供单一服务，这种服务可以是Web网站的服务，也有可能是数据库的服务，甚至是DNS服务。

以Web网站的服务为例，负载均衡会把来自于用户的请求分发到后台的服务器，从而保证用户请求最及时地得到响应，同时各台服务器的压力均衡。负载均衡的另一个好处是对用户屏蔽了后台服务器的（拓扑）结构，这样即使当有一部分后台服务器出现异常的情况下，其他工作正常的服务器仍旧可以为用户提供正常的服务。虽然通常响应时间会变长，但这样大大提高了网站的健壮性和容错能力。

下面来看看一般用户最常见的使用代理服务器达到负载均衡的机制：

当用户请求服务器资源时，请求并不会直接发送给服务器，而是先发送到代理服务器，再通过代理服务器发送给真实的响应服务器；返回时也是通过同样的方式，用户最终从代理服务器拿到真实的响应服务器上的资源。这么做的好处是，代理服务器可以把返回的资源缓存下来，这样当新的用户请求到达代理服务器时，代理服务器可以判断是否已经请求过相应的资源，如果资源存在，那么直接返回代理服务器缓存的资源，这样能大大提升响应速度；而如果没有请求过这个资源，则把这个请求发送给真实的响应服务器，从而获取并缓存新的资源。很多大公司为了提升网络访问速度，都在内网搭建了这样的代理服务器，当然规则比我们描述的要更具体。

## 核心知识② 防御式编程

防御式编程的核心思想是认为所有程序都会有问题，所以对于外部输入和依赖，都采取不可信的态度，在自身代码中针对外部条件可能出现的各种问题（绝大部分都是异常场景）进行处理，从而保证产品自身的健壮性和稳定性。

防御式编程要求程序员应该自始至终考虑各种各样的错误处理机制：在局部处理错误、使用错误码来传递错误、在日志文件中记录调试信息、关闭系统或其他一些方式等。

但是过度的防御式编程也会引起问题。因为就如同防御式编程所认为的那样：所有的代码/程序都不可能是完美的，那么应用防御式编程所编写的代码也是一样。试图通过防御式编程做到完美代码是不现实的，而且付出的成本（时间、人员、资源、代码复杂度）根据二八原则也是不可接受的，尤其是我们所面对的程序绝大多数是非航空航天类的民用产品。

所以如何明智地使用防御式编程在易错和关键环节添加基于业务价值或者业务风险的代码，就成为了软件开发过程中的一个难题。

## 拓展知识 非功能性需求

我们在软件开发过程中，最常见的是针对产品需要具备某一功能，从而为用户

提供对应的业务（价值）。因为这些需求和功能模块实现强相关，所以我们称之为功能性需求。而与之对应的是和功能模块实现弱相关的非功能性需求，通常这些需求会被用来衡量产品运作的状态，例如产品的安全性、性能、稳定性、用户体验等。

可以看出功能性需求和非功能性需求关注的纬度不同，而且非功能性需求通常着眼的角度会是针对整个产品，而非特定功能模块。一般来说非功能性需求要求至少某一功能模块的代码稳定之后才会执行对应的测试。