负载均衡组件开发

联系QQ：2816010068，加入会员群

二叉树出品

目录

- 场景分析
- 基本算法介绍
- 接口设计
- 功能开发

场景分析

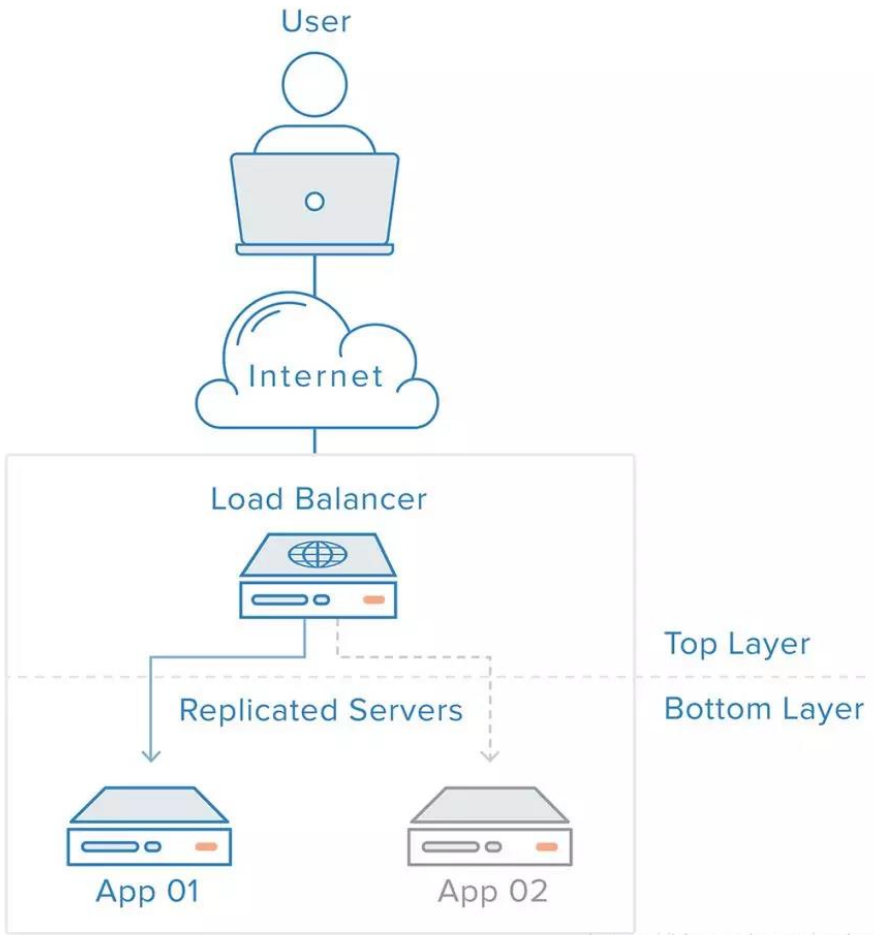
- 分布式系统
  - 每个服务都有多个实例
  - 请求如何路由？
- 传统解决方案
  - DNS+LVS
    - 集中式解决方案
    - 单点故障
  - 软件负载均衡
    - 通过提供负载均衡lib库，在调用方实现负载均衡
    - 结合服务发现，实现节点动态增删（扩容和缩容）
- 负载均衡的好处
  - 服务水平可扩展（解决性能问题）
  - 稳定性大大提升（解决单点故障问题）

场景分析

- 没有负载均衡的日子

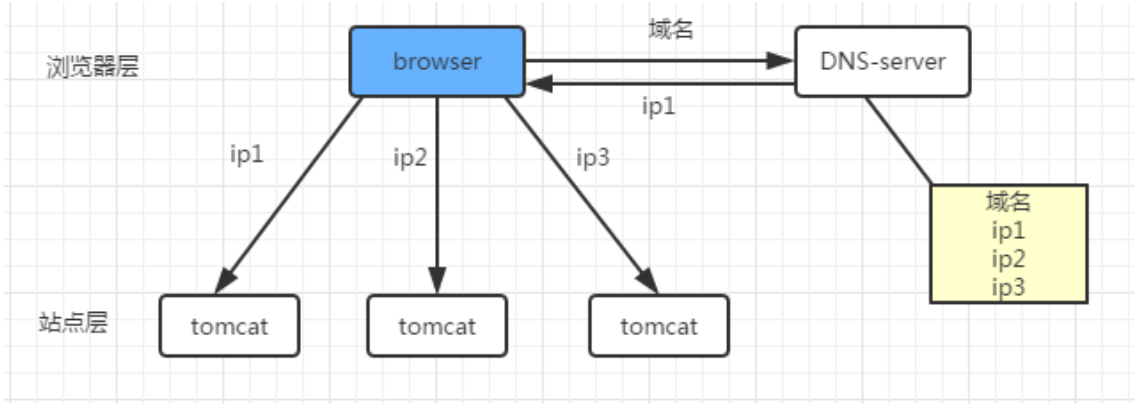
场景分析

- 负载均衡时代


User
Internet
Load Balancer
Top Layer
Replicated Servers
Bottom Layer
App 01
App 02
https://blog.csdn.net/moakun

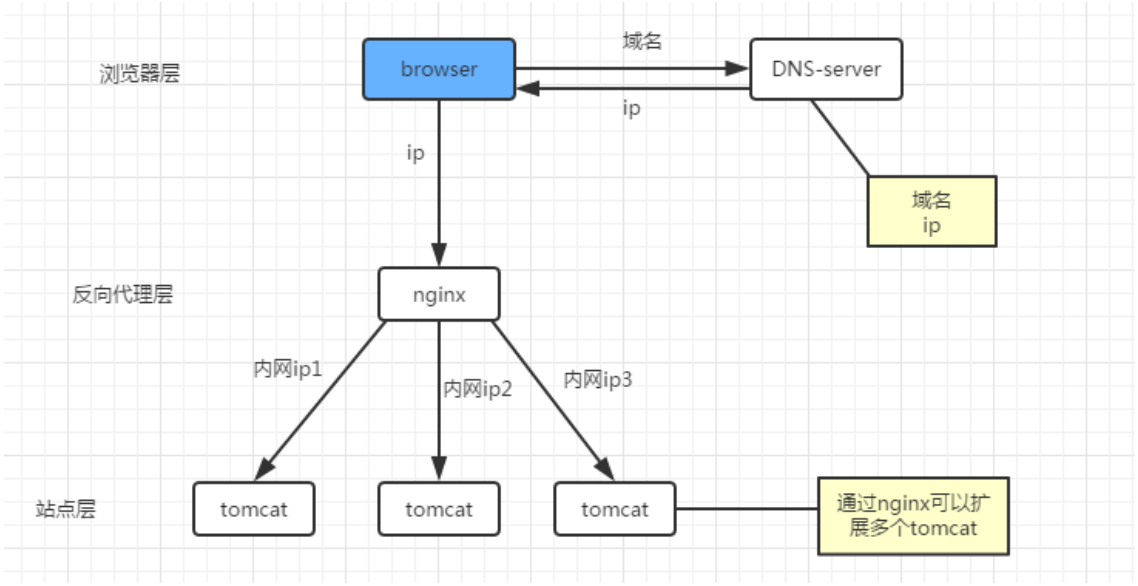
常见的负载均衡解决方案

- DNS解决方案
  - 把一个域名解析到多个IP上
  - 用户访问域名的之后，dns服务器通过一定策略返回一个IP
  - 具体策略：
    - 随机策略
    - 轮询策略
    - 加权轮询
  - 缺点
    - 其中一个IP宕机之后，有一定概率失败
- 动态DNS解决方案
  - 可以通过程序动态的修改dns中域名配置的ip
  - 监控程序发现后端ip宕机之后，通过dns进行删除



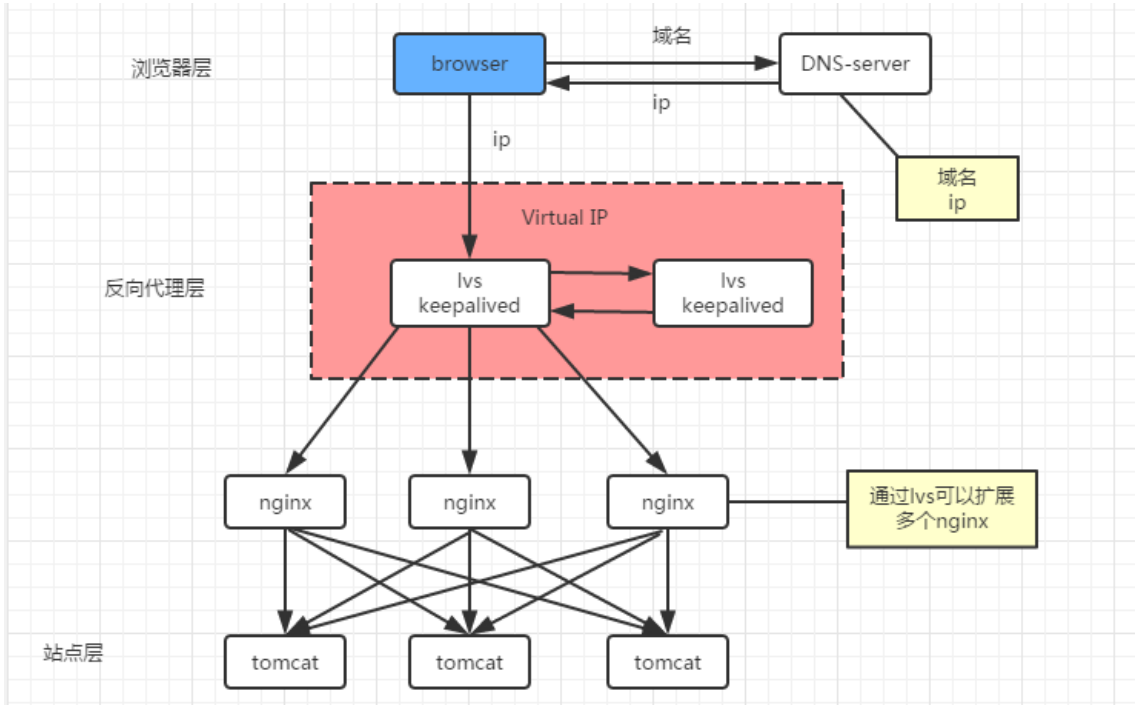
常见的负载均衡解决方案

- Nginx反向代理
  - Nginx负载均衡
  - 扩容实时，随时增加web server
  - Web server挂了，nginx实时摘除



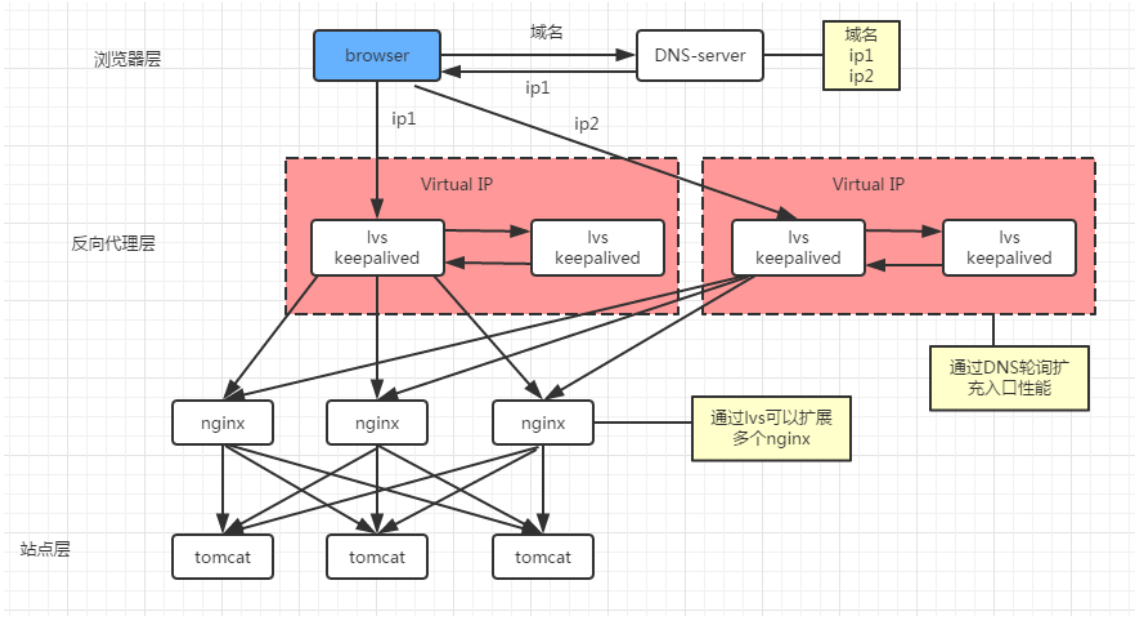
常见的负载均衡解决方案

- LVS负载均衡
  - Nginx实时扩容
  - Nginx挂了，实时摘除
  - Lvs通过virtual ip实现高可用



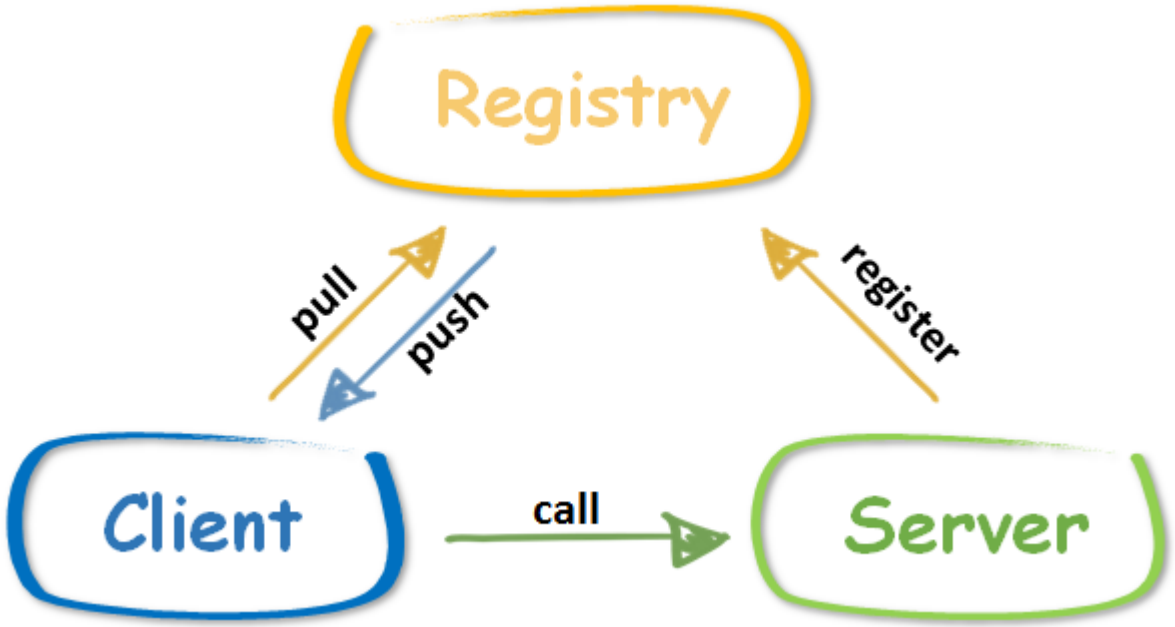
常见的负载均衡解决方案

- LVS负载均衡改进版
  - Lvs水平可扩展
  - 支撑无限流量



常见的负载均衡解决方案

- 客户端负载均衡
  - 服务的ip节点维护在客户端
  - 结合注册中心，实现节点动态增删



负载均衡算法

- 负载均衡算法的本质
  - 从一系列节点中，通过一定的策略，找到一个节点
  - 然后调用方使用该节点进行连接和调用

- 负载均衡算法
  - 随机算法
  - 轮询算法
  - 加权算法
    - 加权随机算法
    - 加权轮询算法
  - 一致性hash算法

随机算法

- 从一系列节点中，随机选择一个节点
  - 比如：节点列表：ip1，ip2，ip3，ipn
  - 通过随机算法，random.Int(n)，生成一个0到n之间的数I
  - 返回第i个下标对应的节点

轮询算法

- 从一系列节点中依次选择节点
  - 比如节点列表：ip1，ip2，ip3，ipn
  - 第一次选择ip1
  - 第二选择ip2
  - 依次类推

加权算法

- 由于后端机器性能不一致，每个节点处理能力不一样，应该需要设置一个权重，权重高的处理请求多，权重低的处理请求少
  - 比如对于三个节点的ip：[ip1, 3], [ip2, 2], [ip3, 1]
  - 先处理成一维列表： [ip1,ip1,ip1,ip2, ip2, ip3]
  - 然后结合轮询算法或随机算法，就形成了加权轮询或加权随机算法
    - 加权轮询算法
    - 加权随机算法

接口设计

- 接口定义

```
package loadbalance

import (
    "context"

    "github.com/ibinarytree/koala/registry"
)

type LoadBalance interface {
    Name() string
    Select(ctx context.Context, nodes []*registry.Node) (node *registry.Node, err error)
}
```

随机算法开发

轮询算法开发

加权随机算法

加权轮询算法