

# 高可用和可伸缩架构

# 分布式架构

- 为什么我们需要分布式架构
  - 高可用 (单机房)
  - 可伸缩
  - 小组边界清晰
  - 提高服务质量 (多机房)
  - 高可用 (跨机房)

啥叫高可用?

- 狭义：单台机器当机后用户无感知
- 延伸：单台设备(主机，网卡，交换机，网线)失效后用户无感知、单个服务异常时用户无感知

啥叫可伸缩？

当业务量增长时，可以简单通过加机器布服务来解决，那么这个服务就是可伸缩的

你想讲啥？

# 想填一下这张表

	高可用	可伸缩
入口层		
业务层		
缓存层		
数据库		

# 入口层如何做高可用?

- 使用心跳技术: Keepalived 就提供这个功能
- 比如机器A IP是 1.2.3.4, 机器 B 的 IP 是 1.2.3.5, 那么再申请一个 IP 1.2.3.6(我们称之为心跳IP), 平时绑定在机器A上面, 如果 A 当机, 那么 IP 会自动绑定在机器 B 上边.
- DNS层面绑定到心跳IP即可

```
1 $ host www.example.com
2 www.example.com has address 1.2.3.6
3
```

**keepalived 有什么使用限制么？**

- 两台机器必须在同一个网段
- 服务监听必须监听所有IP，如果仅仅监听心跳IP，那么从机上的服务（即不持有心跳IP的机器）会启动失败
- 服务器利用率下降（混合部署可以改善这一点）

两台机器，两个公网IP，DNS上把域名同时定位到两个IP，这样算高可用么？

```
$ host www.example.com  
www.example.com has address 93.184.216.34  
www.example.com has address 93.184.216.35
```

不算，客户端（比如浏览器）解析完后会随机选一个IP来访问，而不是一个失败后就去另外一个。所以如果一台机器当机，那么就有一半左右的用户无法访问。

**业务层如何做高可用?**

业务层不要有状态，状态分散到缓存层和数据库。

只要没有状态，业务层的服务死掉后，前面的nginx会自动把流量打到剩下的服务。

友情提醒：不要因为想让服务端无状态就直接用 cookie session, 里边的坑有点大, 考察清楚后再用比较好。比如重放攻击。

**缓存层如何做高可用?**

缓存层分得细一点，保证单台缓存当机后数据库还能撑得住即可。

中小规模下缓存层和业务层可以混合部署，这样可以节省机器。

**你这也叫高可用，明明是祸水东引嘛**

也有办法，缓存机启用主从两台。

主服务活着的时候，主服务读，主从服务都写。

主服务当机后，从服务升级为主服务。主服务恢复后，变成新的从服务。

这个模式也有不少变种，但大部分都需要两倍的服务器，而且性能下降。

**数据库层有什么高可用手段么？**

MySQL 有主从模式，还有主主模式都能满足你的需求。

MongoDB 也有 ReplicaSet 的概念，基本都能满足大家的需求。

# 小结

	高可用	可伸缩
入口层	心跳	
业务层	服务无状态	
缓存层	减小粒度	
数据库	主从模式	

可伸缩

入口层如何提供伸缩性？这个我知道，直接铺机器，  
然后 DNS 加 IP 就可以了吧？

可以，但也有需要注意的地方：

尽管一个域名解析到几十个IP没有问题，但是很多浏览器客户端只会使用前面几个IP，部分域名供应商对此有优化（比如每次返回的IP顺序随机），但这个优化效果不稳定。

推荐的做法是使用少量的 nginx 机器作为入口，业务服务器隐藏在内网（HTTP类型的业务这种方式居多）。另外，也可以在客户端做一些调度（特别是非 HTTP 型的业务，比如直播）。

业务层伸缩性如何？

跟应付高可用一样，保证无状态是很好的手段。加机器继续水平部署即可。

缓存层伸缩性如何？

直接用 codis 或者 redis 3.0 即可。

太新的东西我不敢用，我准备自己搭如何？

如果低峰期间数据库能抗住，那么直接下线缓存然后上新缓存就是最简单有效的办法。

如果扛不住呢？

取决于缓存类型了，可以把缓存分为三类：

1. 强一致性缓存：无法接受从缓存拿到错误的数椐（比如用户余额，或者会被下游继续缓存这种情形）
2. 弱一致性缓存：能接受在一段时间内从缓存拿到错误的数椐（比如微博的转发数）
3. 不变型缓存：缓存key对应的value不会变更（比如从 SHA1 推出来的密码，或者其他复杂公式的计算结果）

**那什么缓存类型伸缩性比较好？**

弱一致性和不变型缓存的扩容很方便，用一致性 Hash 即可。

强一致性情况稍微复杂一些。

**为什么要用一致性Hash? 用简单 Hash 不行么?**

如果缓存从 9 台扩容到 10 台，简单 Hash 情况下 90% 的缓存会马上失效，而一致性Hash情况下，只有 10% 的缓存会失效。

**强一致性缓存有什么问题？**

1. 缓存客户端的配置更新时间会有微小的差异，在这个时间窗内有可能会拿到过期的数据。
2. 如果扩容之后裁撤节点，会拿到脏数据。比如 a 这个key 之前在机器1，扩容后在机器2，数据更新了，但裁撤节点后key回到机器1，这时候就会有脏数据的问题

有办法解决这个问题么？

要解决问题2比较简单，要么保持永不减少节点，要么节点调整间隔大于数据的有效时间

问题1可以用如下方法来解决:

- a. 两套hash配置都更新到客户端，但仍然使用旧配置
- b. 逐个客户端改为只有两套hash结果一致的情况下会使用缓存，其余情况从数据库读，但写入缓存
- c. 逐个客户端通知使用新配置

## 数据库层面如何伸缩?

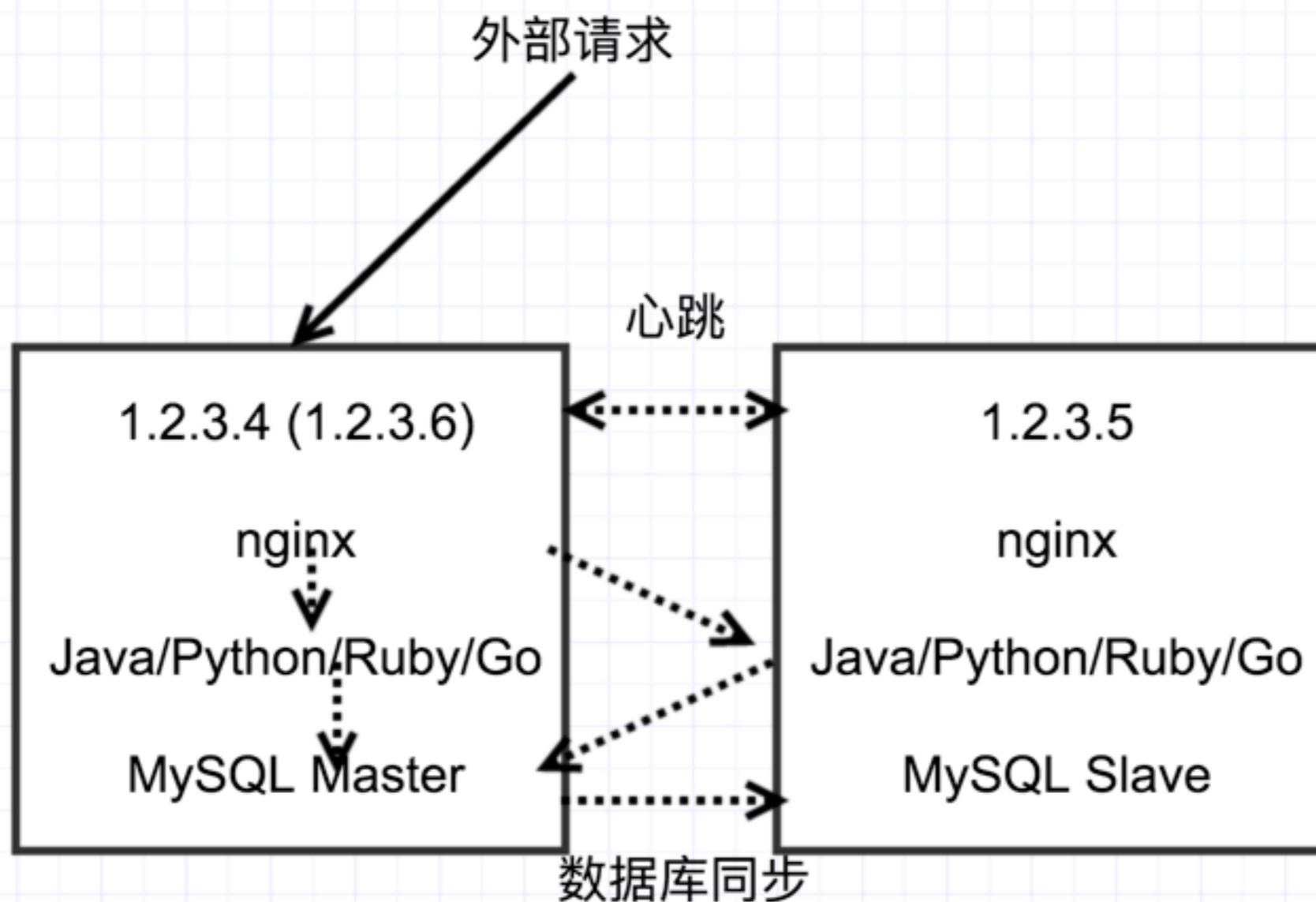
方法很多，文档也很多，就不细讲了。

1. 水平拆分
2. 垂直拆分
3. 定期滚动

# 总结

	高可用	可伸缩
入口层	心跳	平行部署
业务层	服务无状态	服务无状态
缓存层	减小粒度	一致性Hash
数据库	主从模式	拆分与滚动

# 最简单的双机高可用可伸缩部署



# 分布式架构的未来

- 技术产品化
  - 现在我们的分布式的架构的设计是基于我们对服务器操作系统和网络的理解，这种理解对于分布式也许是不必要的
  - 从 GAE 到 Heroku 和 Cloudfoundry, 这种努力不算太成功
  - 从 docker 到 Kubernetes 和 Mesos, 新的机会正在来临，但不能太乐观

Thanks for your  
attention

Q&A?