高可用架构

[一、 逻辑层 1](#_Toc29220)

[二、 缓存层 2](#_Toc21690)

# 逻辑层

1. 业务多
2. 逻辑复杂
3. 整体架构
4. ALL IN ONE
5. 问题:
6. 文件复杂
7. 耦合性严重
8. 开发代价高
9. 维护代价高
10. 牵一发而动全身
11. 适合:
12. 创业期
13. 业务不复杂
14. 如何设计

业务垂直划分

一个业务一个组件(目录和文件)

1. 优点
2. 业务独立
3. 耦合性降低
4. 业务键开发互不影响
5. 开发效率高
6. 缺点
7. 物理上是一个模块
8. 变异成本高
9. 一个业务修改重新上线
10. 重启影响所有业务

一个业务一个进程√

1. 设计
2. 无状态

不存储请求数据，随机请求服务器

1. 关键因素
2. 不保存请求状态
3. 不保存请求数据
4. 业务服务器完全对称

2. 负载均衡

3. 异步调用

1. 消息队列

4.超时

5.幂等设计

# **缓存层**

1. **缓存作用**
2. **加速请求响应时间,响应延迟小**
3. **减少数据库的压力**
4. **缓存适合场合**
5. **静态资源**
6. **较少改动的缓存**
7. **读多写少**
8. **缓存类型**
9. **本地缓存**

**静态不变数据**

1. **进程内缓存**
2. **分布式缓存**
3. **分布式缓存类型**

**Memcached:**

**简单的k-v**

**Redis:持久化**

**链表、集合、哈希，性能好 √**

1. **一致性**
2. **更新:先更新数据库,delete缓存**

**1秒后，再次失效一次缓存，双重失效**

1. **设置缓存有效期**
2. **事后补偿，记录日志，脚本定期修正**

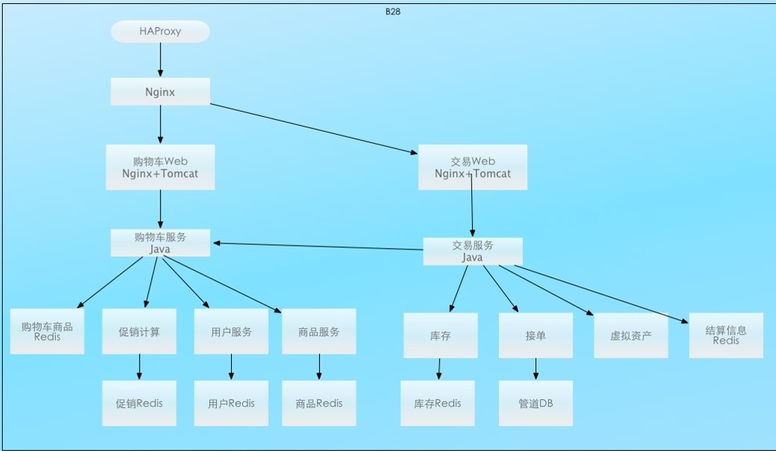
**6.高可用命中率**

1. **相对静态数据**
2. **缓存时间长**

# **十大原则**

**Nginx+Redis+SSDB**

1. **原则一:按照业务切分系统**



**Nginx:分流**

**购物车服务:**

**购物车商品(redis)**

**促销计算(redis)**

**用户服务(redis)**

**商品服务(redis)**

**交易服务:**

**库存**

**接单**

**虚拟资产**

**结算信息**

1. **原则二:灾备切换**
2. **多机房部署**

　　机房内部出现大面积故障和机房入口流量故障

　　1 入口流量切换策略

　　1)DNS域名：较慢，机房入口处问题+机房内部，有地域性

　　2)LVS+HAProxy 直接路由：较快，机房内部大面积故障

1. **机房内部切换**

　　调用者客户端切换

　　1 Java应用切换策略：底层服务框架，VIP，内部DNS

　　2 缓存切换策略

3 数据库切换策略：客户端，DB自身

1. **数据一致性保证：**

1 程序双写：灵活，集群间相互切换快。

　　a 同步双写

　　数据一致性高，影响性能，不适合多个集群。

　　b 异步双写

　　数据一致性差，性能高，适合多个集群。

　　举例：购物车系统。

　　2 底层存储数据复制

　　优点：数据一致性高

缺点：主从切换不灵活，复制节点出问题后，恢复慢。

1. **灾备集群**

　　1 空闲灾备集群

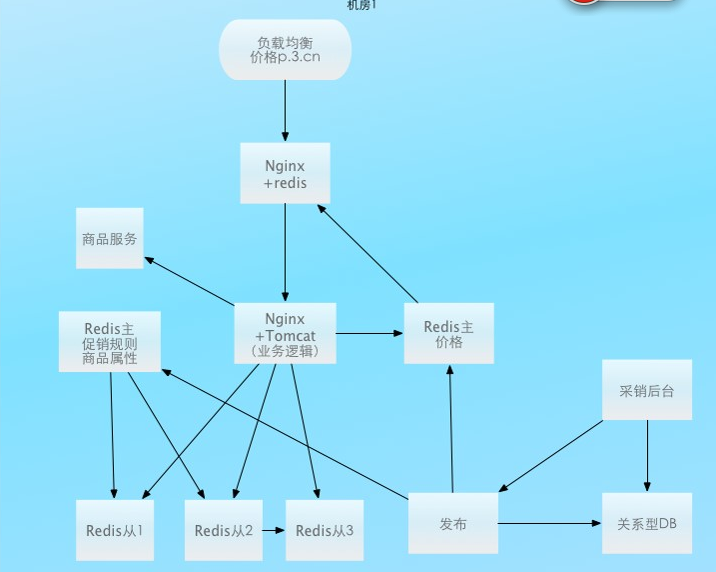
　　热点缓存，管理成本

　　2 小流量灾备集群

　　增加事故风险

　　3 全部切换-部分切换

1. **原则三:水平扩展 分流**



1. **原则四:超预期流量 限流**

**区分正常流量和超预期流量**

**限流策略:**

**Nginx层限流**

**Web应用层限流 (不用)**

**原则五:保护系统 降级**

**原则六:提高性能 读写分离**

**原则七:唯快不破 异步化**

**原则八:动态,静态流量分离**

**原则九:线上压力测试 发现瓶颈**

**读逻辑**

**写逻辑**

**Cpu,内存，硬盘,网络**

**吞吐量,并发量，响应时间**

**原则十:预案**