**架构是业务数据与业务逻辑之间的折衷关系**

分层：根据模块相似性，横向划分层次。

分割：根据子模块差异性，纵向划分单元。

集群：相同的模块部署在不同的机器上，构成一个集群，通过负载均衡向外服务。(冗余)

分布式：不同的模块部署在不同的机器上，远程调用协同。(分工合作)

自动化：固化而重复的工作，由机器自动完成。

Performance

1. 缓存(cache)

场景：读多写少、很少变化的数据，80%业务集中在20%数据上，空间换时间。

应用：本地缓存、CDN

1. IO多路复用(multiplexing)

场景：传输任务数 > 信道数

应用：线程池、HTTP/TCP链接

1. 压缩(compression)

场景：压缩开销 < 传输开销

应用：git

1. 异步任务队列

场景：不需实时响应

应用：小额批量支付

1. 大数据量分批次

场景：空间不足

应用：导入导出、数据库迁移

High Availability

1. 无状态化服务冗余，状态化数据备份，针对可靠性和容错性问题
2. 服务器：
3. 失效转移: 负载均衡设备通过心跳健康检测等手段，发现并剔除不可用的机器，使请求分发至可用的机器上，保证整个集群可用。
4. 分级管理: 着重核心和优先级高的服务。
5. 超时机制: 设定时限，制定后续处理规则。
6. 服务降级: 限流，关闭非必要的服务。
7. 幂等性设计: 重复调用，结果一致。
8. 灰度发布: 负载均衡下，关闭并部署个别机器，验证成功后，开启并同步代码至其他机器。
9. 数据库：
10. CAP原理: 以下要素相互制约。

consistency一致性: 同一个数据的多个副本是确定一致的

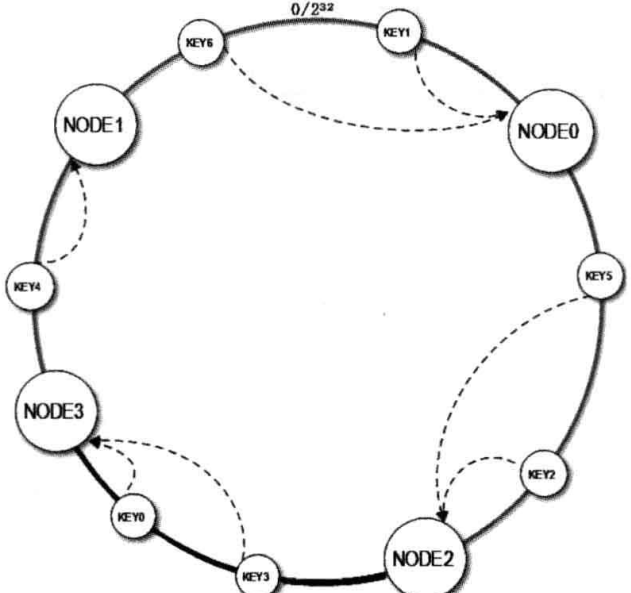
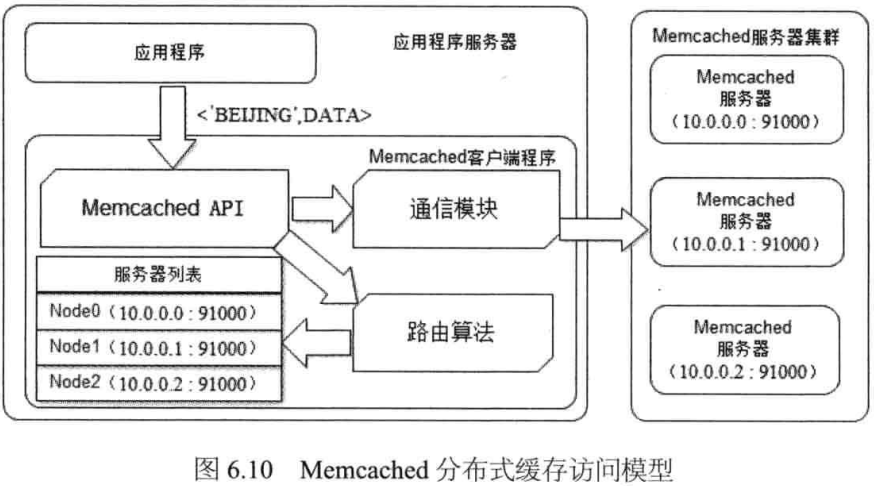
availability可用性: 随时随地可读写

patition tolerance分区耐受性(可伸缩性): 跨网络分区线性伸缩

1. 备份恢复: 写入时，数据同步复制到其他多个数据库，丢失后，从其他可用的数据库恢复，保证数据持久化。(e.g. master-slave读写分离的同步热备机制)
2. 监控monitor：
3. 日志 + 性能指标
4. 告警及处理

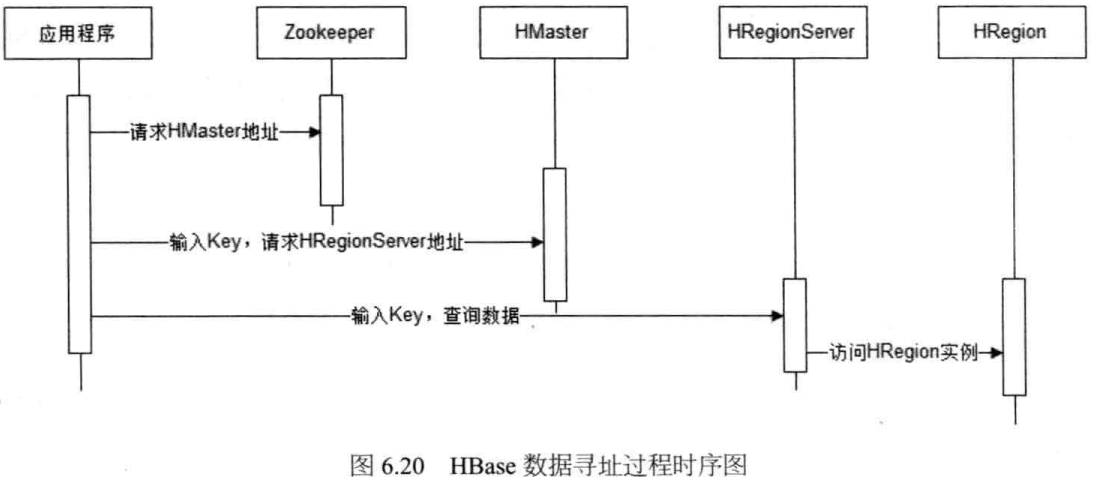
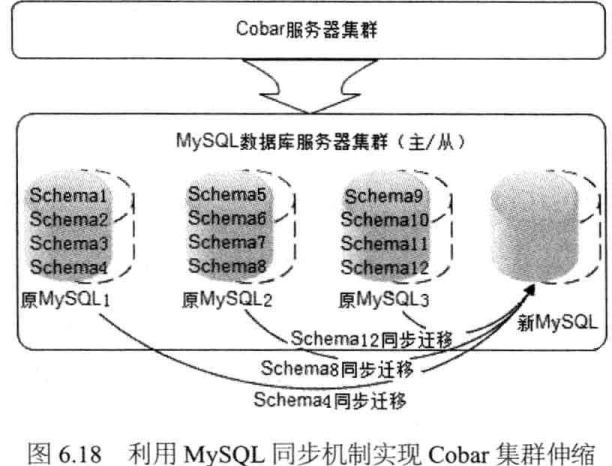
Scalability

1. 向集群中增加机器，针对业务量增长的问题
2. 服务器：(负载均衡的不同实现)
   1. HTTP重定向
   2. DNS域名解析
   3. 反向代理机器
   4. IP转发(SNAT)
   5. 数据链路层(保证虚拟IP一致，改mac实现转发)
   6. 均衡分发算法：round robin, weighted round robin, random, least connections, hash
3. 数据库：
   1. 缓存



一致性hash算法: 固定key的范围(环)，将缓存服务器置于环上，作为key的界线，后续的key顺时针寻找最近的节点保存。因为保持hash函数和值域的不变，细分扩容，保证以前的缓存基本不失效。

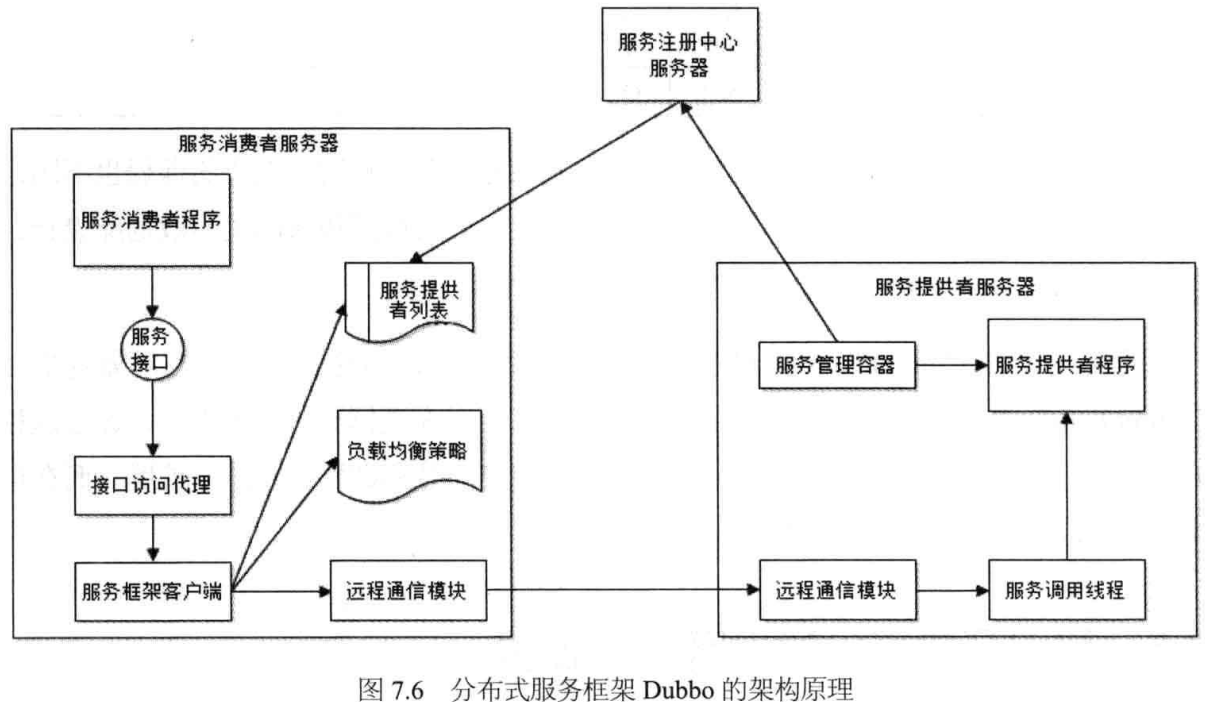
* 1. 持久化数据库(RDBMS/NoSQL)



迁移依据：一致性hash算法。缓存可以不迁移，因为有数据库兜底，但数据库就没兜底了。

Business Extensibility

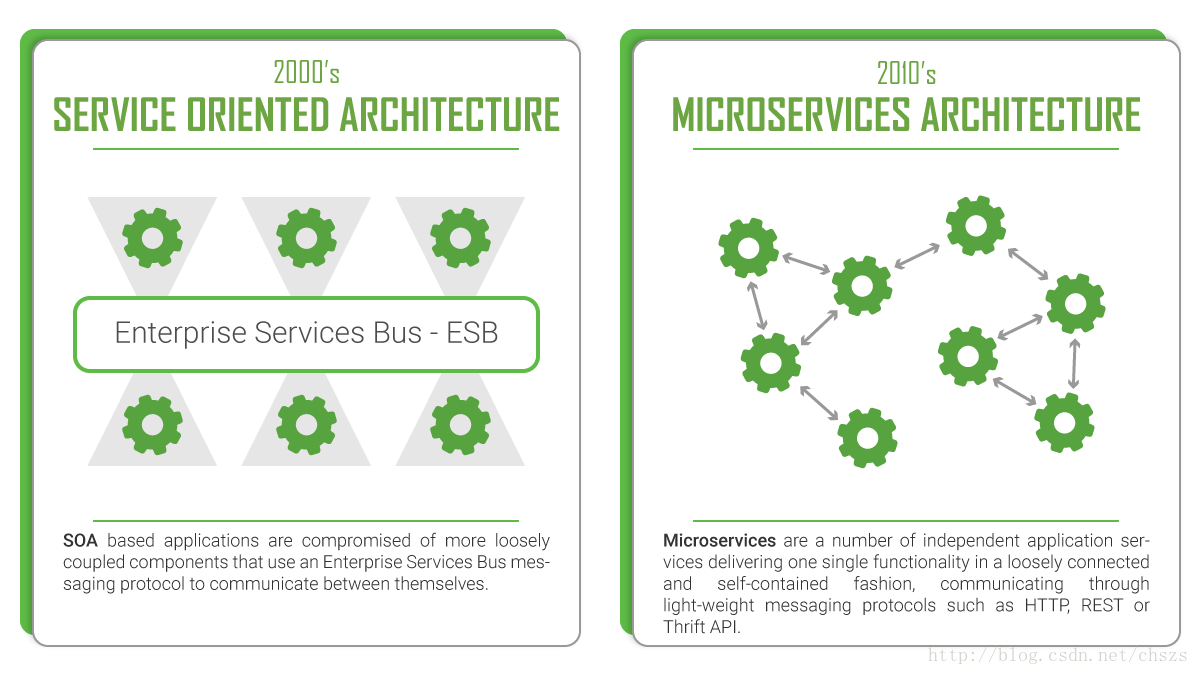
1. 业务模块化解耦，针对新增业务的问题
2. 消息队列/事件模型/订阅-推送机制
3. 注册中心



Security

1. 需拦截过滤的攻击：XSS/SQL注入/CSRF
2. 加密：单向哈希/对称加密/非对称加密/数字签名/数字信封
3. 垃圾信息过滤：规则匹配/分类算法/黑名单
4. 金融风控：规则引擎/统计模型

-----------------------------------------------------------------------------------



Micro Service

微服务开发流程：

1. 设计：接口、模型

2. 编码：代码版本库

3. 流水线自动化构建：编译打包、部署环境、用例验证

4. 发布版本

微服务集成：

服务容器化、路由转发、分层管理、容错机制