数据库

类型

关系型数据库

- 优点
  - 保持数据一致性
  - 可以进行join等复杂查询
  - 通用化，技术成熟
- 缺点
  - 读写经过Sql解析，大数据量、高并发下读写性能不足
  - 对数据读写或修改数据结构时需要加锁，影响并发操作
  - 无法适应非结构化存储
  - 扩展困难

非关系型数据库

- 优点
  - 高并发，在大数据下读写能力较强
  - 基本支持分布式，易于扩展，可伸缩
- 缺点
  - join等复杂操作功能较弱
  - 事务支持较弱
  - 通用性差
  - 无完整约束，复杂场景支持较差

使用nosql的原因

在超大规模和高并发的SNS（社交网络）类型的web2.0纯动态网站面临的问题

1. High performance - 对数据库高并发读写的需求
2. Huge Storage - 对海量数据的高效率存储和访问的需求
3. High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求

数据拆分

拆分模式

水平拆分

- 典型分片规则
  1. 按用户ID求模，将数据分散到不同数据库中，具有相同数据用户的数据都被分散到一个库中
  2. 按照日期，将不同月甚至日的数据分散到不同库中
  3. 按照某个特定的字段求模，或者根据特定范围段分散到不同的库中
- 优点
  - 拆分规则抽象化
  - 不存在单库大数据、高并发的性能瓶颈
  - 应用端改造较少
  - 提高了系统的稳定性及负载能力
- 缺点
  - 拆分规则难以抽象
  - 分片事务一致性难以解决
  - 数据多次扩展维度及维护量极大
  - 垮库join性能较差

垂直拆分

- 优点
  - 系统之间整合或扩展容易
  - 业务清晰，拆分规则明确
  - 数据维护简单
- 缺点
  - 部分业务表无法join，只能通过接口方式解决，提高了系统复杂度
  - 受业务限制存在单库性能瓶颈，不易数据扩展及性能提高
  - 事务处理复杂

拆分规则

- 能不拆分尽量不拆分
- 如果拆分则要选择好合适的拆分规则，提前规划好
- 数据拆分尽量通过数据冗余和表分组来降低跨库join的可能
  - 表分组（Table Group）是解决跨分片数据 join 的一种很好的思路，也是数据切分规划的重要一条规则
- 由于数据库中间件对数据join实现的优劣难以把握，而且实现高性能难度极大，业务读取尽量少用多表join

分库分表中间件

- cobar
  - 不支持
    - 存储过程
    - 读写分离
    - 垮库join
    - 分页
  - *proxy层*
- atlas
- mycat
  - 优点：对于各个项目透明，无耦合
  - 缺点：需要独立部署，保证高可用
  - *proxy层*
- sharding-jdbc
  - 优点：不用部署，运维成本低，不需要代理层的二次转发请求，性能较高
  - 缺点：遇到升级需要各个系统都重新升级版本再发布，各个系统都需要耦合 sharding-jdbc 的依赖
  - *client层*

分库分表面临的问题

- 垮库join
- 事务
- 跨节点分组、排序及聚合
- 数据迁移及扩容
- 分页查询
  - 如果是在前台应用提供分页，则限定用户只能看前面n页，这个限制在业务上也是合理的，一般看后面的分页意义不大（如果一定要看，可以要求用户缩小范围重新查询）
  - 如果是后台批处理任务要求分批获取数据，则可以加大page size，比如每次获取5000条记录，有效减少分页数（当然离线访问一般走备库，避免冲击主库）。