软件架构与中间件

涂志莹
tzy_hit@hit.edu.cn

哈尔滨工业大学

软件架构与中间件

Software Architecture and Middleware

第4章
数据层的软件架构技术

第4章 数据层的软件架构技术

4.1 数据驱动的软件架构演化

4.2 数据读写与主从分离

4.3 数据分库分表

4.4 数据缓存

4.1 数据驱动的软件架构演化

1、数据与软件

2、数据带来的架构变化

4.1.1 数据与软件

数据的定义

- 一般而言，数据是指对客观事件进行记录并可以鉴别的符号，是对客观事物的性质、状态以及相互关系等进行记载的物理符号或这些物理符号的组合。它是可识别的、抽象的符号。
- 表示形态上，数据可以是狭义上的数字，可以是具有一定意义的文字、字母、数字符号的组合、图形、图像、视频、音频等，也可以是客观事物的属性、数量、位置及其相互关系的抽象表示。
- 在计算机科学中，数据是指所有能输入到计算机并被计算机程序处理的符号的介质的总称，是用于输入电子计算机进行处理，具有一定意义的数字、字母、符号和模拟量等的通称。

数据的解释

- 数据与信息
  - 信息与数据既有联系，又有区别。数据是信息的表达、载体，信息是数据的内涵，是形与质的关系。
  - 数据本身没有意义，数据只有对实体行为产生影响时才成为信息。
- 数据与语义
  - 数据的表现形式还不能完全表达其内容，需要经过解释，数据和关于数据的解释是不可分的。
  - 数据的解释是指对数据含义的说明，数据的含义称为数据的语义，数据与其语义是不可分的。

数据的软件价值

数据+语义+逻辑=业务

代码+业务=软件应用系统

4.1.2 数据带来的架构变化

数据带来的变化

1. Stand alone->Interoperability->Transboundary
2. Customized user data
3. IoT data

数据+语义+逻辑=业务

海量、离散

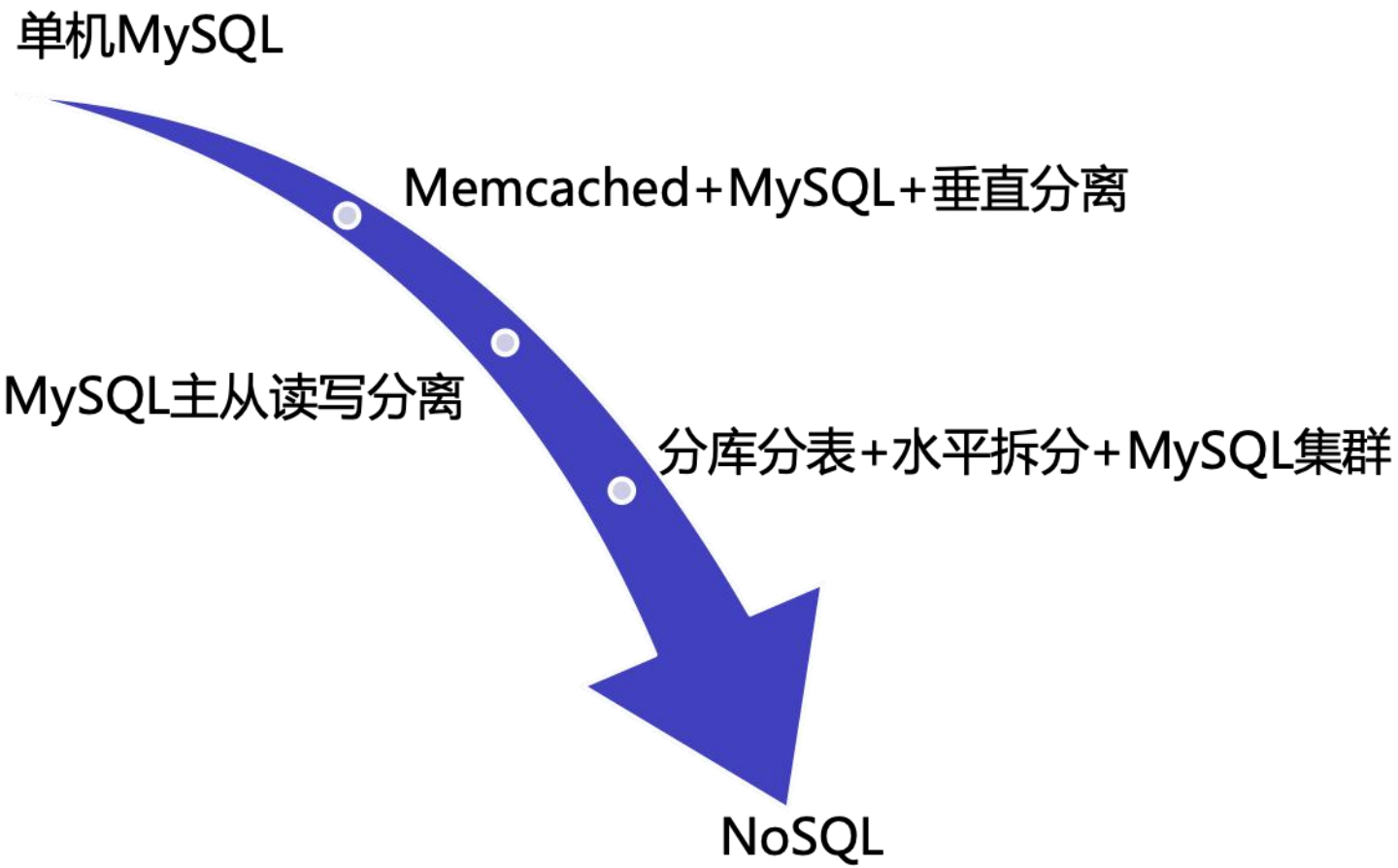
异构

复杂、多用户、多片段

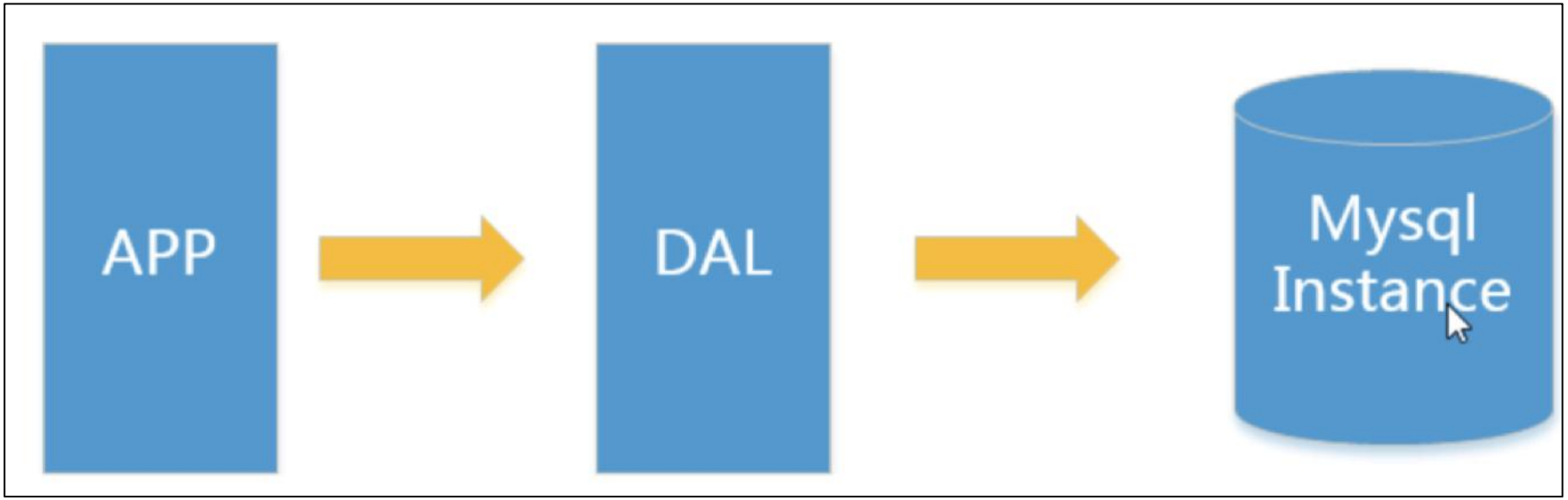
代码+业务=软件应用系统

面向海量数据的高并发的复杂分布式异构系统

数据架构的演化


单机MySQL
Memcached+MySQL+垂直分离
MySQL主从读写分离
分库分表+水平拆分+MySQL集群
NoSQL

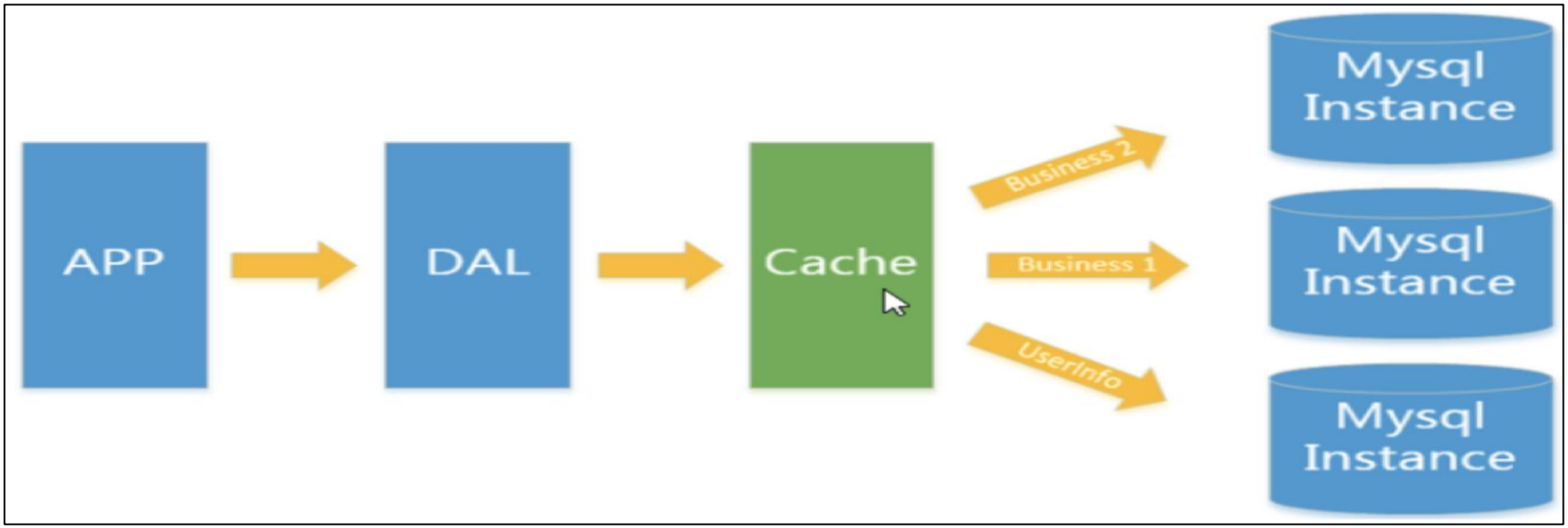
单机MySQL



问题：

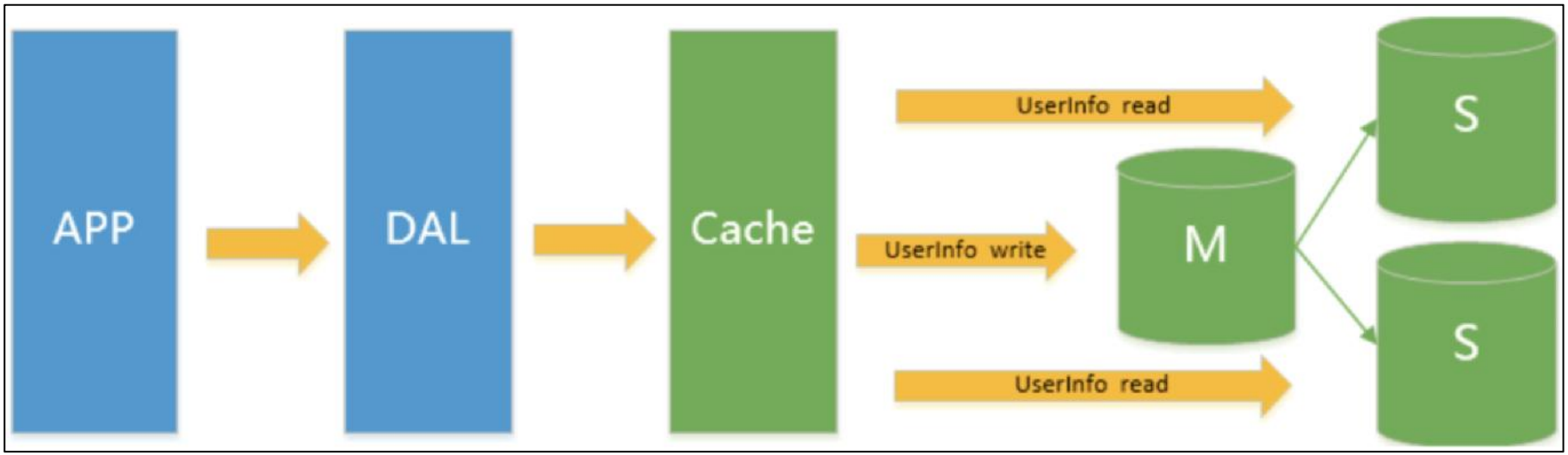
- 1、数据量过大，单机存储不下
  - MySQL5.7单表极限存储值500w数量，但是300w时已需要优化。
- 2、数据量过大，单机内存存不下数据索引
  - 索引是加速数据库访问效率的一种机制，存储于内存
  - 数据量越大，索引越大，内存消耗越多
  - 每当加入数据，都需要维持索引，不做优化，数据访问将非常缓慢。
- 3、访问量过大，在读写混合的情况下，一个实例不能承受。

Memcached+MySQL+垂直分离



- 随着访问量的上升，几乎大部分使用MySQL架构的网站在数据库上都出现性能问题，程序员们开始大量的使用缓存技术来缓解数据库的压力。
- 最开始比较流行的是通过文件缓存来缓解数据库的压力，但是当访问量继续增大的时候，多台web服务器通过文件缓存不能共享，大量的小文件缓存也带来了比较高的IO压力。
- Memcached(缓存)蕴意而生，管理缓存。
- 缓存原则：频繁被访问的数据可以被放置于缓存当中，以供频繁访问。
- 垂直分离指按业务划分将不同的数据放在不同的数据库里。

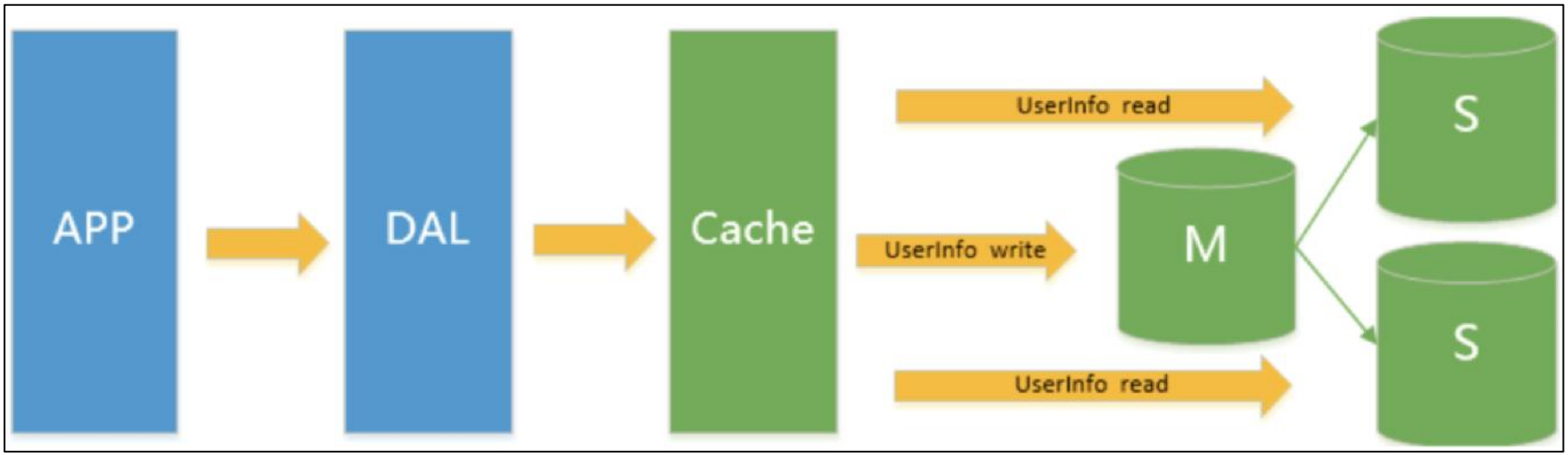
MySQL主从读写分离



- 主从复制：容灾备份，缓存备份，保证数据的完整性
- 读写分离：
  - 增删改是写，查为读。
  - “读”的职能允许从被查询的数据库服务器读数据。
  - “写”的职能允许向写数据的数据库服务器写数据。
  - 分工明确，结合缓存能实现性能的一大提升。

分库分表+水平拆分+MySQL集群

更多



原因：

•数据库中的数据量不一定是可控的，在未进行分库分表的情况下，随着时间和业务的发展，库中的表会越来越多，表中的数据量也会越来越大，相应地，数据操作，增删改查的开销也会越来越大

•另外，由于无法进行分布式部署，而一台服务器的资源（CPU、磁盘、内存、IO等）是有限的，最终数据库所能承载的数据量、数据处理能力都将遭遇瓶颈

NoSQL=Not Only SQL

云时代对DB的新需求：

- 高性能 - 对数据库高并发读写的需求
  - 数据库并发负载非常高，往往要达到每秒上万次读写请求，硬盘IO无法承受。
- 海量存储 - 对海量数据的高效率存储和访问的需求
  - 对海量数据进行SQL查询，效率极其低下不可忍受。
- 高伸缩性与高可用性 - 对数据库的高可扩展性和高可用性的需求
  - 无法通过简单添加更多的硬件和服务节点来扩展性能和负载能力，需要停机维护和数据迁移。

数据层非功能需求与架构技术

- 存储高性能
  - 读写分离、数据缓存、分库分表、NoSQL
- 存储高可用
  - 主从、CAP理论
- 存储高扩展
  - 分库分表、NoSQL

4.2 数据读写与主从分离

1、读写分离

2、主备与主从复制

3、主备倒换与主从倒换

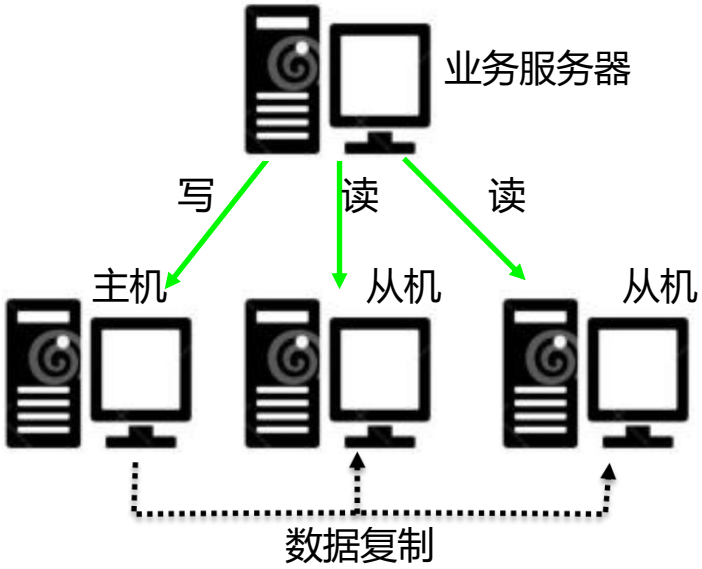
4、主主复制

5、数据集群

4.2.1 读写分离

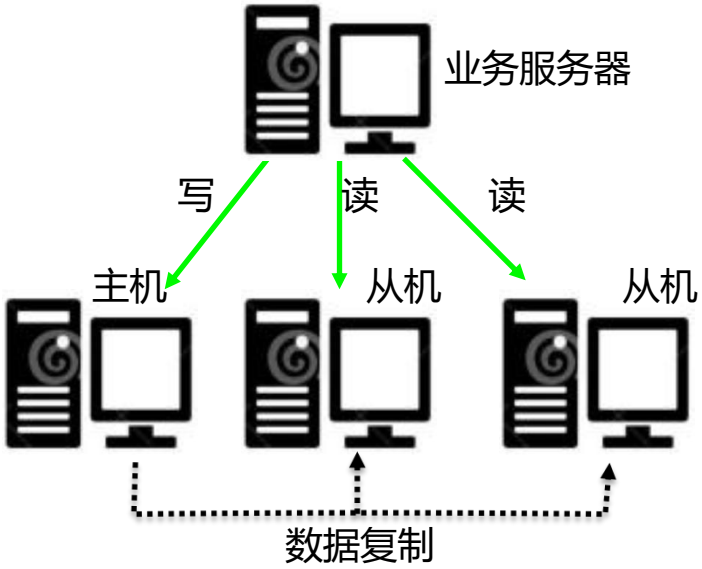
基本原理

- 将数据库读写操作分散到不同的节点上
  - 数据库服务器搭建主从集群，一主一从、一主多从都可以。
  - 数据库主机负责读写操作，从机只负责读操作。
  - 数据库主机通过复制将数据同步到从机，每台数据库服务器都存储了所有的业务数据。
  - 业务服务器将写操作发给数据库主机，将读操作发给数据库从机。



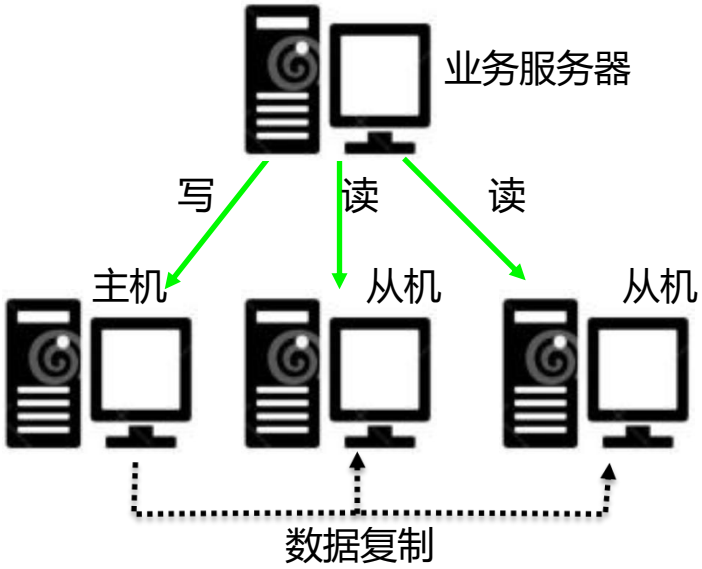
复制延迟

- 主从复制延迟问题
  - 如果业务服务器将数据写入到数据库主服务器后立刻进行读取，此时读操作访问的是从机，主机还没有将数据复制过来，到从机读取数据是读不到最新数据的，业务上就可能出现问题。



复制延迟

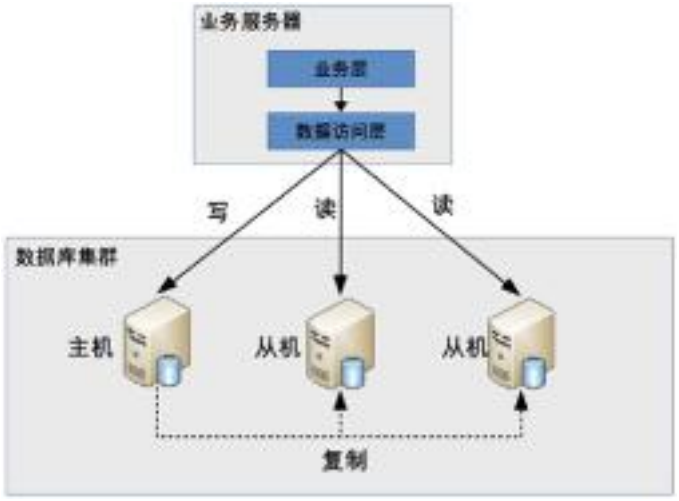
- 应对复制延迟的方案
  1. 写操作后的读操作指定发给数据库主服务器
     - 例如，注册账号完成后，登录时读取账号的读操作也发给数据库主服务器
  2. 读从机失败后再读一次主机
  3. 关键业务读写操作全部指向主机，非关键业务采用读写分离



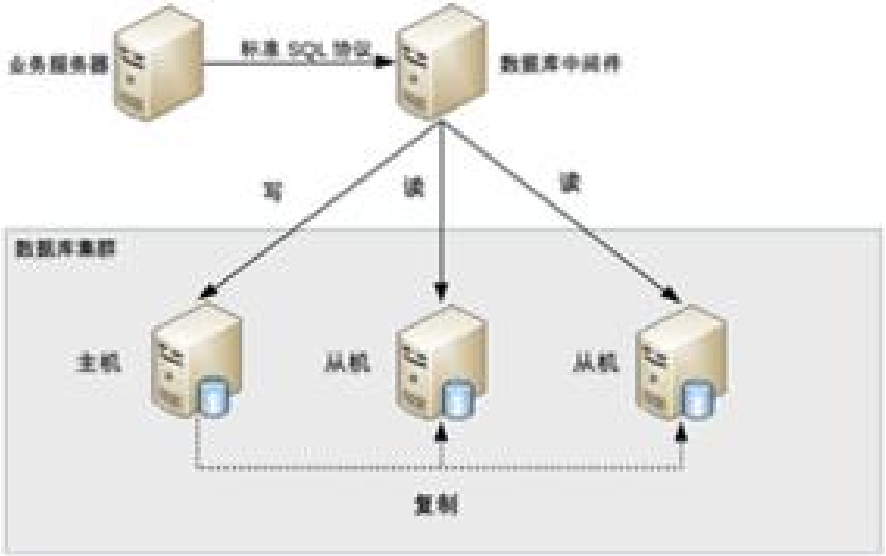
分配机制

- 一般有两种方式

程序代码封装



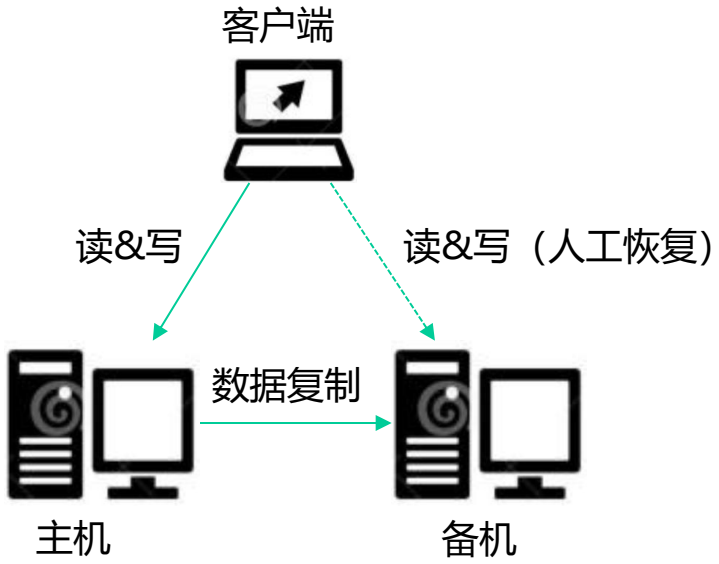
中间件封装



4.2.2 主备与主从复制

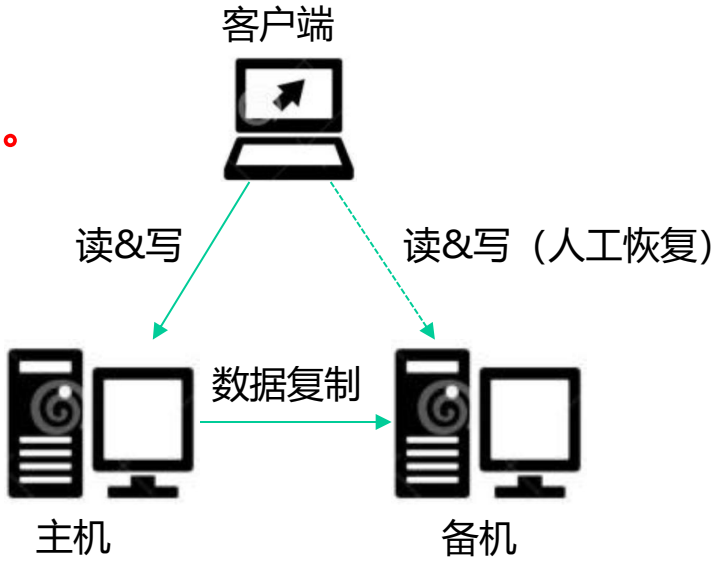
基本实现逻辑

- 1、主机存储数据，通过复制通道将数据复制到备机。
- 2、正常情况下，客户端无论读写操作，都发送给主机，备机不对外提供任何读写服务。
- 3、主机故障情况下，客户端不会自动将请求发给备机，此时整个系统处于不可用状态，不能读写数据，但数据并没有全部丢失，因为备机上有数据。
- 4、如果主机能够恢复（人工或自动），客户端继续访问主机，主机继续将数据复制给备机。
- 5、如果主机不能恢复，则需要人工升级备机为主机，增加新备机，切换访问链路。



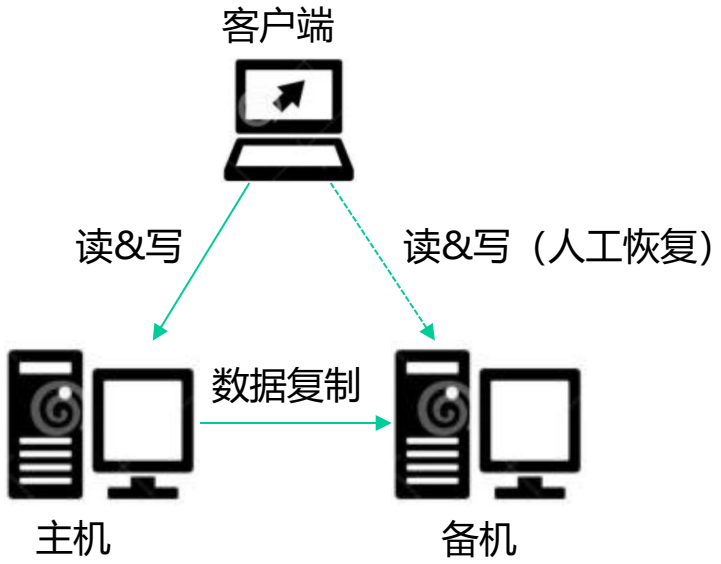
基本实现逻辑

- 6、主机不能恢复的情况下，成功写入主机但还没有复制到备机的数据会丢失，需要人工进行排查和恢复，也许有的数据就永远丢失了，业务上需要考虑如何应对此类风险。
- 7、如果主备间数据复制延迟，由于备机并不对外提供读写操作，因此对业务没有影响，但如果延迟较多，恰好此时主机又宕机了，则可能丢失较多数据，因此对于复制延迟也不能掉以轻心。一般的做法是做复制延迟的监控措施，当延迟数据量较大时及时预警，由人工干预处理。
- 注意，在此类方案中，备机只起到备份作用。



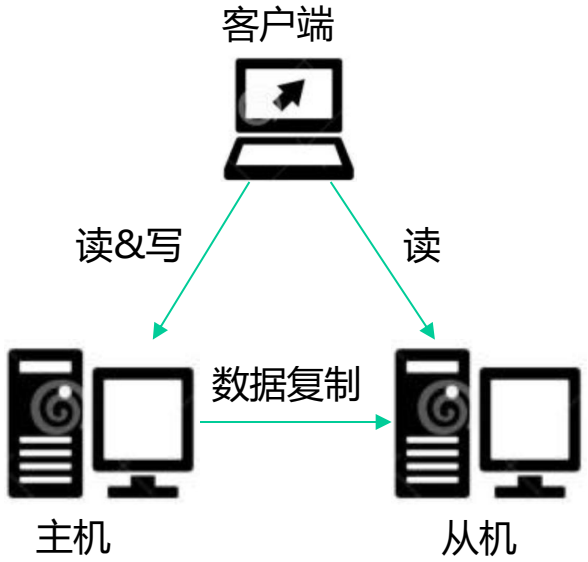
优缺点分析

- 优点：
  - 对于客户端来说，不需要感知备机的存在，即使灾难恢复后，原来的备机被人工修改为主机后，对于客户端来说，只是认为主机的地址换了，无需知道是原来的备机升级为主机了。
  - 对于主机和备机来说，双方只需要进行数据复制即可，无须进行状态判断和主备倒换等复杂操作。
- 缺点：
  - 备机仅是备份，不提供读写操作，硬件浪费。
  - 故障后需人工干预，无法自动恢复。



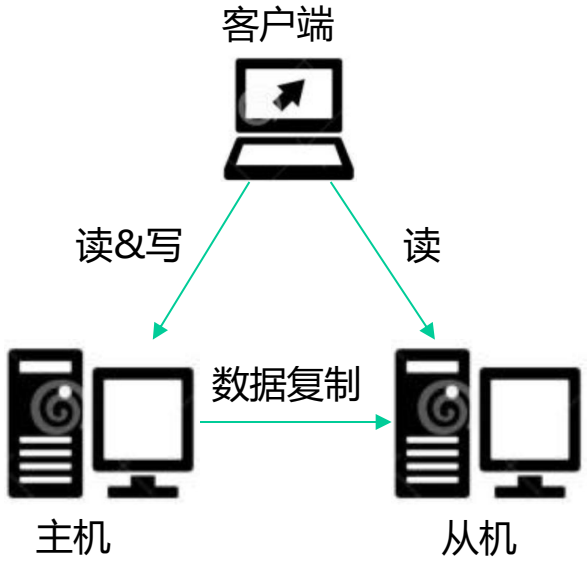
基本实现逻辑

- 1、主机存储数据，通过复制通道将数据复制到从机。
- 2、正常情况下，客户端写操作发送给主机，读操作可发送给主机也可以发送给从机(可以随机读，轮询读，只读主机)。
- 3、主机故障情况下，客户端无法进行写操作，但可以将读操作发送给从机，从机继续响应读操作，此时和写操作相关的业务不可用，但是读操作相关的不受影响。
- 4、如果主机能够恢复（人工或自动），客户端继续将写操作请求发送给主机，主机继续将数据复制给从机。
- 5、如果主机不能恢复，则需要人工升级从机为主机，然后让客户端访问新主机，同时，为了继续保持主从结构，人工增加新机器作为从机。



基本实现逻辑

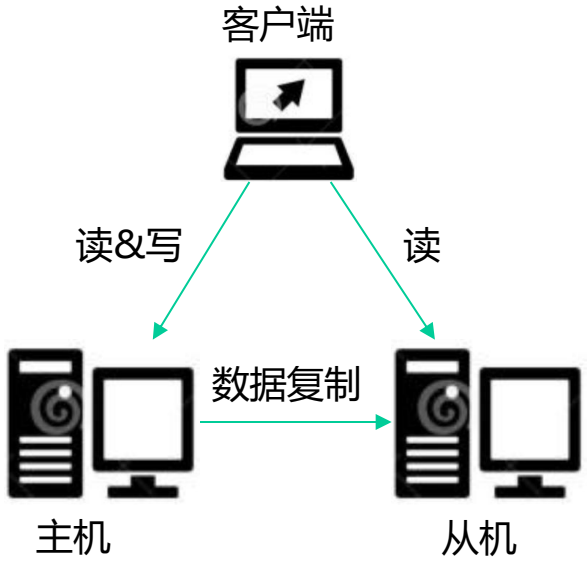
- 6、主机不能恢复的情况下，成功写入了主机但是还没有复制到从机的数据会丢失，需要人工进行排查和恢复，也许有的数据就永远丢失了，业务上需要考虑如何应对此类风险。
- 7、如果主从间数据复制延迟，则会出现主从读取数据不一致问题。
- 8、如果主从间延迟较多，恰好此时主机又宕机了，则可能丢失较多数据，因此对于复制延迟也不能掉以轻心。一般的做法是做复制延迟的监控措施，当延迟数据量较大时及时预警，由人工干预处理



优缺点分析

- 优缺点（相对主备复制而言）：
  - 主从复制在主机故障时，读操作相关的业务不受影响。
  - 主从复制架构的从机提供读操作，发挥了硬件的性能。
  - 主从复制要比主备复制复杂，主要体现在客户端需要感知主从关系，并将不同的操作发给不同的机器进行处理。

同样的缺点，需要人工的干预处理故障，效率低。



4.2.3 主从倒换与主备倒换

设计关键

- 主备和主从复制的共性问题：
  - 主机故障后，无法进行写操作。
  - 如果主机无法恢复，需要人工指定新的主机。

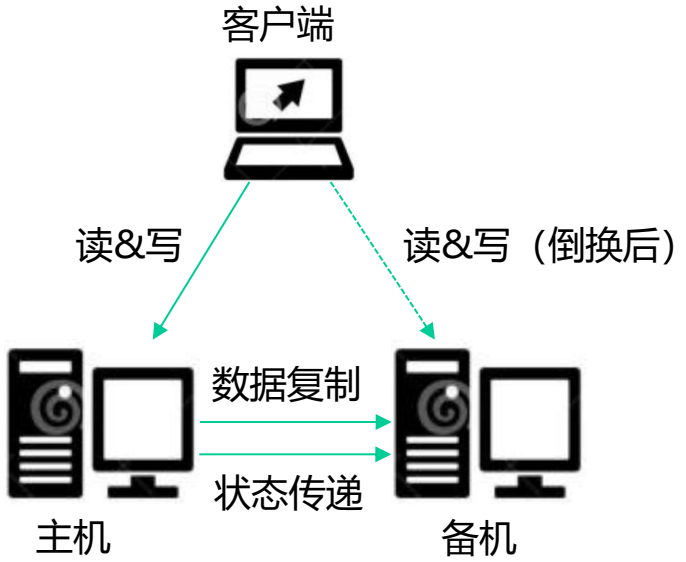
关键的设计点：

- 主备间状态判断
  - 状态传递渠道：P2P，或者第三方仲裁？
  - 状态检测内容：如，机器掉电？进程是否存在？响应缓慢？等
- 倒换决策
  - 倒换时机：即时，短时，长时的决策？
  - 倒换策略：是否让原主机做主机还是切换？
  - 自动程度：自动or半自动？
- 数据冲突

常见架构-互连式

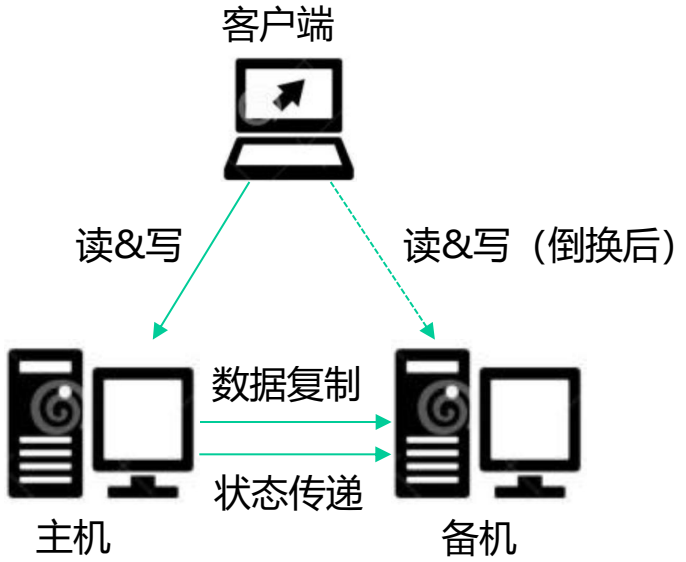
- 互连式：主备机直接建立状态传递的渠道。
  - 可以是网络连接（如，各开一个端口），也可以是非网络连接（用串口线连接）。
  - 可以是主机发送状态给备机，也可以是备机拉取主机的状态。
  - 可以和数据复制通道共用，也可以独立一条通道。
  - 状态传递通道可以是一条，也可以是多条，还可以是不同类型的通道混合（如，网络+串口）。

- 为了充分利用主备自动倒换方案自动决定主机的优势，方案有：
  - 1、主备机共享一个对于客户端来说唯一的地址（如，虚拟IP）。
  - 2、客户端记录主备机各自的IP，备机具有拒绝服务的能力。



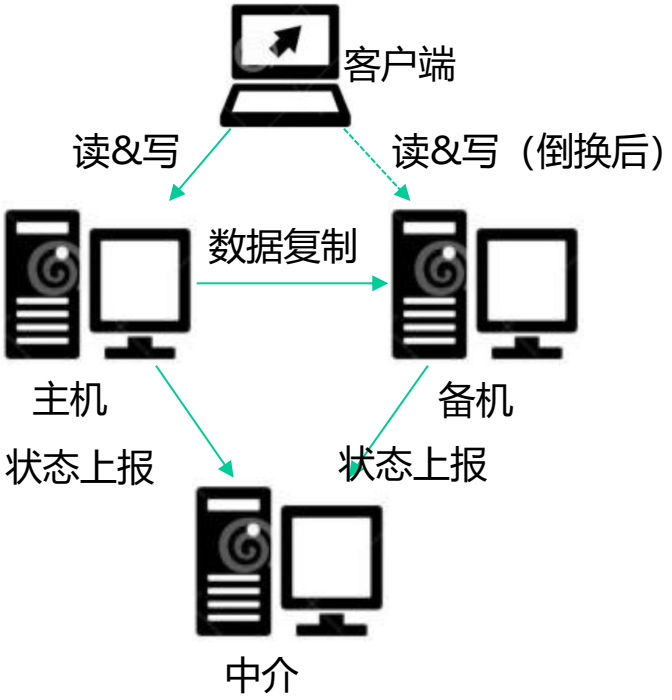
常见架构-互连式

- 缺点：
  - 状态传递通道本身故障了，则备机会主动升级为主机
  - 虽然可以通过多通道来降低通道故障的机率，但是通道越多，后续的状态决策越复杂，特别是容易收到多种矛盾的信息。



常见架构-中介式

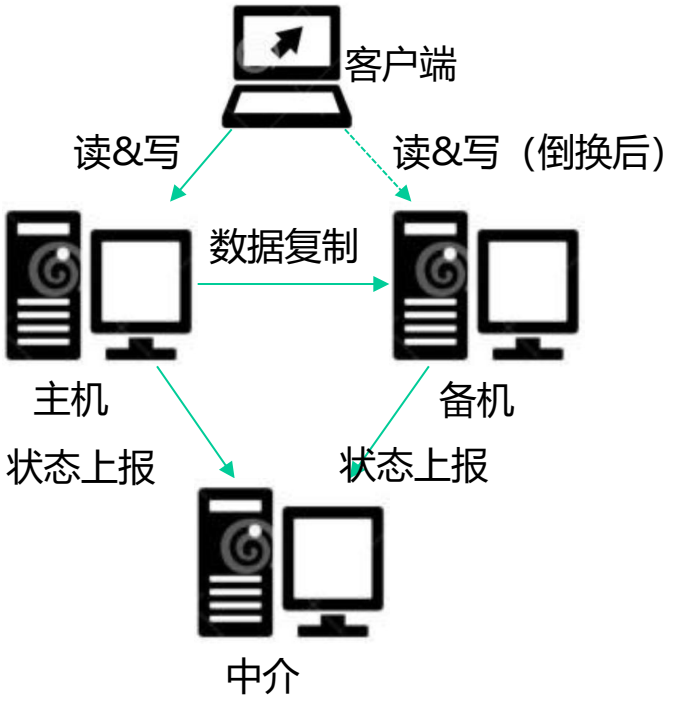
- 中介式：在主备机之间引入第三方中介，主备机之间不直接连接，而都去连接中介，并且通过中介来传递状态信息。结构复杂了，但是从状态传递和决策上更加简单：
  - 1、连接管理更简单：主备机无须再建立和管理多种类型的状态传递连接通道，只要连接到中介即可，实际上降低了主备机的连接复杂度。
  - 2、状态决策更简单：无须考虑多种类型的连接通道获取状态信息如何决策的问题。



常见架构-中介式

状态决策的步骤：

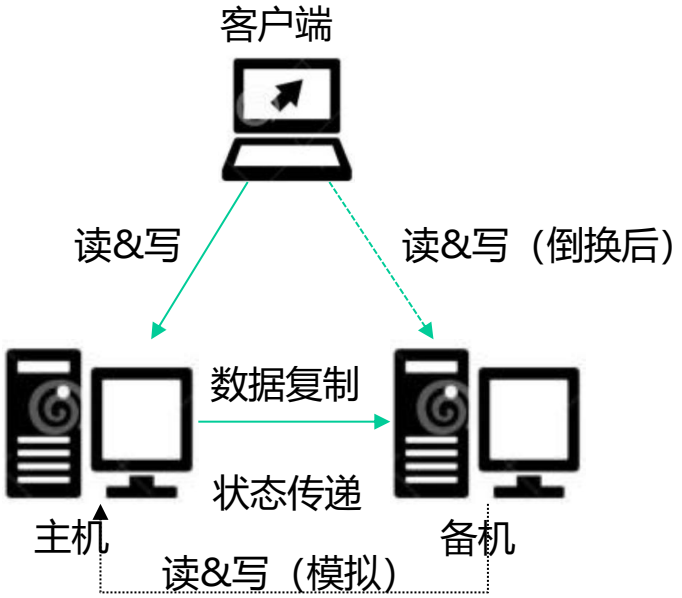
- 1、无论主机还是备机，初始状态都是备机，并且只要与中介断开连接，就将自己降级为备机，因此可能出现双备机的情况。
- 2、主机与中介断连后，中介能够立刻告知备机，备机将被升级为主机。
- 3、如果是网络中断导致主机与中介断连，主机自己会降级为备机，网络恢复后，旧的主机以新的备机身份向中介上报自己的状态。
- 4、如果是掉电重启或者进程重启，旧的主机初始状态为备机，与中介恢复连接后，发现已经有主机了，保持自己备机状态不变。
- 5、主备机与中介连接都正常的情况下，按照实际的状态决定是否进行倒换，如，响应超时。



常见架构-模拟式

- 模拟式：主备机之间并不传递任何状态数据，而是备机模拟成为一个客户端，向主机发起模拟的读写操作，根据读写操作的响应情况来判断主机的状态。

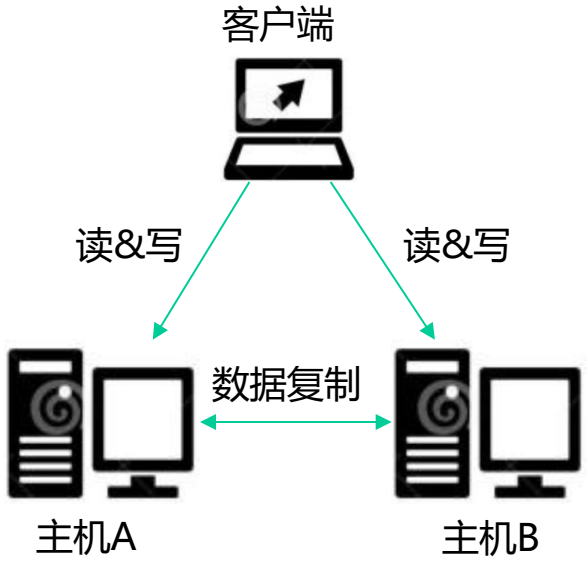
- 模拟式相比互连式的优缺点：
  - 1、实现更加简单，省去了状态传递通道的建立和管理工作。
  - 2、模拟式读写操作获取状态信息只有响应信息（如，HTTP404，超时，响应时间超过3s等），没有互连式那么多样，基于有限的状态来做状态决策，可能出现偏差。



4.2.4 主主复制

基本设计思路

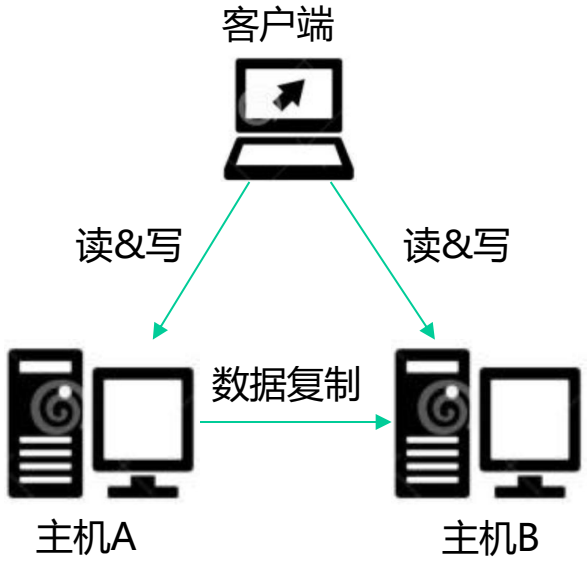
- 主主复制：两台机器都是主机，互相将数据复制给对方，客户端可以任意挑选其中一台进行读写操作。
  - 1、两台主机都存储数据，通过复制通道将数据复制到另一台主机。
  - 2、正常情况下，客户端可以将读写操作发送给任意一台主机。
  - 3、一台主机故障情况下，客户端只需要将读写操作发送给主机B即可，反之亦然。
  - 4、如果故障主机能够恢复，则客户端继续访问两台主机，两台主机继续相互复制对方数据。
  - 5、如果故障主机不能恢复，则需要人工操作，增加一台新的机器为主机。



基本设计思路

- 6、原有故障主机不能恢复的情况下，成功写入原有故障主机但没有复制到正常主机的数据会丢失。
- 7、如果两台主机间复制延迟，则可能出现客户端刚写入的数据，在另一台主机上读取不到。

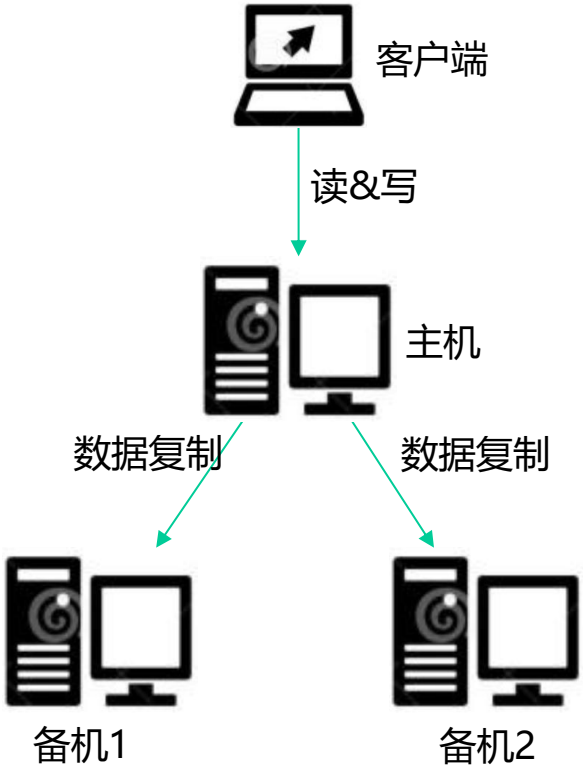
- 相较之下的优缺点：
  - 两台主机，无倒换概念
  - 客户端无须区分主备机身份
  - 必须保证数据能够双向复制，然而很多数据无法双向复制，如两个库新插入数据的ID均为100，或者两人对某产品（100件）进行购买，甲从A买了1件，乙从B买了2件，如果复制B to A，则两个数据库都是98。实际应该是97。



4.2.5 数据集群

数据集中集群

- 数据集中集群称为一主多备/从。数据都只能往主机写，而读操作可以参考主备，主从的架构进行灵活变化。
- 复杂度高：
  - 多备即多通道，增加了主机的复制压力，同时增加了对正常读写的压力（实践中，需要考虑降低该压力）
  - 多通道，情况不一，容易导致数据不一致，需要在备机之间进行数据一致性检查和修正。
  - 多备对单主状态的检测结果不一致，容易出现不同的判断和决策。
  - 单主多备，当主机宕机，如何重新选主，需要算法。



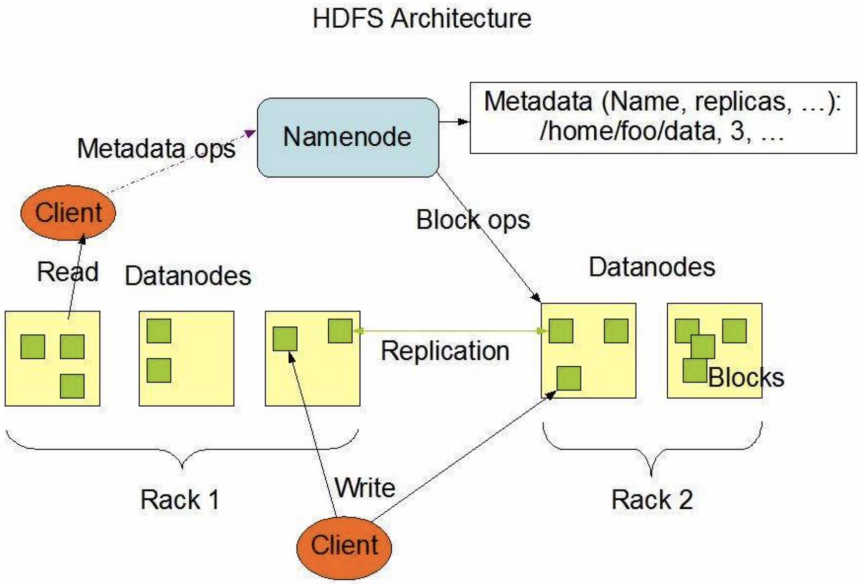
数据分散集群

- 数据分散集群指多个服务器组成一个集群，每台服务器都会负责存储一部分数据，同时，为了提升硬件利用率，每台服务器又会备份一部分数据。
- 数据分散集群的复杂度在于如何将数据分配到不同的服务器上，需要思考的方面有：

  - 均衡性：保证数据分区基本均衡
  - 容错性：部分服务器故障后，这些服务器上的数据分区需要分配给其他服务器
  - 可伸缩性：当集群容量不够，扩充新的服务器后，算法能够自动将数据分区迁移到新服务器，并保证扩容后所有服务器的均衡性。

数据分散集群

- 数据分散集群中，必须要有一个角色来负责执行数据分配算法
  - 可以是独立服务器，如HDFS架构
  - 也可以是集群选举出的服务器，也称之为“主机”，但职责完全不同，如Elasticsearch



Elasticsearch架构

集中VS分散

- 写数据角色
  - 数据集中集群架构中，客户端只能将数据写到主机
  - 数据分散集群架构中，客户端可以向任意服务器中读写数据。
- 应用场景
  - 数据集中集群适合数据量不大，集群机器数量不多的场景，如ZooKeeper 集群，一般个位数机器
  - 数据分散集群，由于其良好的可伸缩性，适合业务数据量巨大、集群机器数量庞大的业务场景，如Hadoop 集群，可达上千台服务器

第4章

数据层的软件架构技术

Thanks for listening

涂志莹
哈尔滨工业大学计算机学院
企业与服务计算研究中心