Mysql

MySQL是一个[关系型数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)，由瑞典MySQL AB 公司开发，目前属于 [Oracle](https://baike.baidu.com/item/Oracle) 旗下产品。MySQL 是最流行的[关系型数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是[开放源码](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E6%BA%90%E7%A0%81)这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

Redis

针对比较容易出现大量更新和读取操作的实时状态信息来制定其存储方案。这类实时更新的状态数据的特点是每次更新的数据量不大，并不需要很大的存储空间来处理单个更新的数据，但是其更新的频率较快而且更新的总体次数较多，所以需要一种相对较为简单并能够提供较高的数据读取性能的存储机制。同时这些数据通常是以键值对的形式进行存储与更新，一般以状态的类型为键，状态的具体数据值为值，所以我们选择键值对存储的方案。

键值对存储是一种对数据的最简单的组织方式。通常只提供对数据的获取、存储、删除等操作，不像一般的关系型数据库可以执行复杂的 SQL 查询语言，所以采用键值对存储非常适合实时更新的状态这样的不涉及过多业务逻辑的场景。而且键值对存储一般采用基于内存的存储机制，专注于数据的高效存储与索引，并且由于其结构的简单性能够带来比较好的分布式扩展和高容错性的优势，这比传统的处理大量结构数据的关系型数据库具备更好的读写性能。

基于内存的键值存储方案采用将数据和索引都缓存在内存中的机制，以计算机内存资源来换取实时数据处理的性能，利用操作系统内存高速的读写速度提供高性能的读写，并且避开了访问传统关系型数据库时磁盘 I/O 速度问题及并发量不够的问题。

Redis的外围由一个键、值映射的[字典](https://zh.wikipedia.org/wiki/%E5%85%B3%E8%81%94%E6%95%B0%E7%BB%84)构成。与其他[非关系型数据库](https://zh.wikipedia.org/wiki/%E9%9D%9E%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93)主要不同在于：Redis中值的类型不仅限于[字符串](https://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E4%B8%B2)。值的类型决定了值本身支持的操作。Redis支持不同无序、有序的[列表](https://zh.wikipedia.org/wiki/%E4%B8%B2%E5%88%97_(%E6%8A%BD%E8%B1%A1%E8%B3%87%E6%96%99%E5%9E%8B%E5%88%A5))，无序、有序的[集合](https://zh.wikipedia.org/w/index.php?title=%E9%9B%86%E5%90%88_(%E6%8A%BD%E8%B1%A1%E8%B3%87%E6%96%99%E5%9E%8B%E5%88%A5)&action=edit&redlink=1)间的交集、并集等高级服务器端原子操作。

Redis的持久化

所谓数据持久化就是就是将数据放到断电后数据不会丢失的设备中,Redis的数据持久化有两种方式,第一种方式是快照方式(Snapshotting),第二种是追加命令方式(Append Only File)。快照方式的主要原理就是在某个时间点把内存中的所有数据的快照保存一份到磁盘上。在条件达到时通过Fork—个子进程把内存中的数据写到一个临时文件中来实现保存数据快照。在所有数据写完后再把这个临时文件用原子函数rename(2)重命名为目标rdb文件。这种实现方式充分利用Fork的copy on write。追加命令方式的主要原理是进程每次将收到的写操作命令以追加的方式写到同一个文件AOF中。而当加载的时候,通过重新播放即可得到数据集原有的状态。AOF以增长方式进行存储的,所以每次写的信息比快照少很多,只与单位时间内的写操作数目成正比,Redis主进程是在将写操作命令放到通往AOF文件的缓冲区之后,才对客户端的请求做出回应。

缓存

Redis 支持内存存储中的各种类型的 value，有字符串型（String）、链表型（List）、集合型 (Set)和有序集合型 (Zset)，多种数据类型是其重要的优势地位。各种数据类型具有 push 功能、pop 功能、add 功能 remove 功能更可以交集并集和差集乃至更丰富的功能组合。在数据类型的基础上，Redis 支持多种多样的排序方式[4]。与 Memcached 排序方式相同，Redis 为了保证效率，将数据全部存储在内存中缓存（备份、快照会存到硬盘中）。Redis 按周期将新的数据写入磁盘或把变更数据写入追加的记录文件中，在这个基础上实现了主从同步。

Redis 是 key-value 开源的缓存数据库（基于内存）。官方性能测试结果：set操作每秒 110000 次，get 操作每秒 81000 次。

异步消息队列

1. 解耦

在项目启动之初来预测将来项目会碰到什么需求，是极其困难的。消息系统在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。这允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束。

1. 冗余

有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。许多消息队列所采用的"插入-获取-删除"范式中，在把一个消息从队列中删除之前，需要你的处理系统明确的指出该消息已经被处理完毕，从而确保你的数据被安全的保存直到你使用完毕。

1. 扩展性

因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的，只要另外增加处理过程即可。不需要改变代码、不需要调节参数。扩展就像调大电力按钮一样简单。

1. 灵活性 & 峰值处理能力

在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见；如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。

1. 可恢复性

系统的一部分组件失效时，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。

1. 顺序保证

在大多使用场景下，数据处理的顺序都很重要。大部分消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。［Kafka保证一个Partition内的消息的有序性；nsq不保证消息的顺序性］

1. 缓冲

在任何重要的系统中，都会有需要不同的处理时间的元素。例如，加载一张图片比应用过滤器花费更少的时间。消息队列通过一个缓冲层来帮助任务最高效率的执行———写入队列的处理会尽可能的快速。该缓冲有助于控制和优化数据流经过系统的速度。

1. 异步通信

很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

第三章

持久化数据库设计

关系型数据库的最大特点就是事务的一致性：传统的关系型数据库读写操作都是事务的，具有ACID的特点，这个特性使得关系型数据库可以用于几乎所有对一致性有要求的系统中，如典型的银行系统。

但是，在交互式应用中，尤其有些场景下，一致性却不是显得那么重要，用户A看到的内容和用户B看到同一用户C内容更新不一致是可以容忍的，或者 说，两个人看到同一好友的数据更新的时间差那么几秒是可以容忍的，因此，关系型数据库的最大特点在这里则不是那么重要了。

非关系型数据库提出另一种理念，例如，以键值对存储，且结构不固定，每一个元组可以有不一样的字段，每个元组可以根据需要增加一些自己的键值对，这 样就不会局限于固定的结构，可以减少一些时间和空间的开销。使用这种方式，用户可以根据需要去添加自己需要的字段，这样，为了获取用户的不同信息，不需要 像关系型数据库中，要对多表进行关联查询。仅需要根据id取出相应的value就可以完成查询。但非关系型数据库由于很少的约束，他也不能够提供像SQL所提供的where这种对于字段属性值情况的查询。并且难以体现设计的完整性。他只适合存储一些较为简单的数据，对于需要进行较复杂查询的数据，SQL数据库显的更为合适。

关系型数据库也有以下瓶颈：

1. 高并发读写需求

交互式应用的用户并发性非常高，往往达到每秒上万次读写请求，对于传统关系型数据库来说，硬盘I/O是一个很大的瓶颈

2. 海量数据的高效率读写

应用每天产生的数据量是巨大的，对于关系型数据库来说，在一张包含海量数据的表中查询，效率是非常低的

3. 高扩展性和可用性

在基于web的结构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展 是非常痛苦的事情，往往需要停机维护和数据迁移。

Kafka

一种分布式的，基于发布/订阅的消息系统。以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上数据也能保证常数时间的访问性能；高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒100K条消息的传输；支持Kafka Server间的消息分区，及分布式消费，同时保证每个partition内的消息顺序传输；同时支持离线数据处理和实时数据处理

* 1. 高承载能力

服务端的高承载能为即为承载多人同时在线的能力,也就是服务端处理并发事务的效率,为了达到高承载能力的要求服务端架构就需要具有高效的进程间通信、高效的逻辑处理和高效的存储效率。

* 1. 可扩展性

在架构设计中为了达到可拓展性的目的,核心的工作是需要设计通用性较强的各类通用模块，如：不同类型微服务功能模块，数据存储模块和数据缓存模块等一系列能够支撑大部分业务的通用模块,这样的设计能够大大减少重复开发工作。

* 1. 可移植性

一个优秀的服务端架构不仅仅要能满足单一项目的特殊需求,并且还能适用于同一领域甚至不同领域的系统需求,从而实现开发效率的提高和开发成本的降低。

* 1. 低耦合

交互式应用的服务端是一系列功能模块构成的集合体,各个模块之间如果耦合度按高将会导致后期开发难度增大,所以服务端架构各模块之间低耦合将会提高架构的性能和开发效率

本文低耦合的设计目标主要是通过对应用的业务进行归类和模块划分,各个模块之间尽量减少业务交叉和重复,模块之间的通过远程调用通信模块进行通信,

总结

在实现模块后对模块进行了性能测试，模块体现了良好的支持高可用高并发的需求性能，问题表和回答表都可以支持万级的高QPS访问。

在本次实践中，对微服务架构也有了深刻的认识，并总结出微服务的优势：微服务模块每个模块专注于自己的功能，模块与模块之间是高内聚低耦合的。由于微服务的独立部署和开发特性，对于不同的微服务模块可以用不同的编程语言

微服务架构的优点：

每个服务都比较简单，只关注与一个业务功能

微服务架构方式是耦合的，可以提供更高的灵活性。

微服务可通过最佳及最合适的不同的编程语言与工具进行开发，能够做到有的放矢地解决针对性问题。

每个微服务可由不同团队独立开发，互不影响，加快推出市场的速度。

微服务架构是持续交付的巨大推动力，允许在频繁发布不同服务的同时保持系统其他部分的可用性和稳定性。

于此同时，微服务架构也有一些缺点：

微服务的一些想法在实践上是好的，但当整体实现时也会呈现出其复杂性。

运维开销及成本增加：整体应用可能只需部署至一小片应用服务区集群，而微服务架构可能变成需要构建/测试/部署/运行数十个独立的服务，并可能需要支持多种语言和环境。这导致一个整体式系统如果由20个微服务组成，可能需要40~60个进程。

必须有坚实的DevOps开发运维一体化技能：开发人员需要熟知运维与投产环境，开发人员也需要掌握必要的数据存储技术如NoSQL，具有较强DevOps技能的人员比较稀缺，会带来招聘人才方面的挑战。

隐式接口及接口匹配问题：把系统分为多个协作组件后会产生新的接口，这意味着简单的交叉变化可能需要改变许多组件，并需协调一起发布。在实际环境中，一个新品发布可能被迫同时发布大量服务，由于集成点的大量增加，微服务架构会有更高的发布风险。

分布式系统的复杂性：作为一种分布式系统，微服务引入了复杂性和其他若干问题，例如网络延迟、容错性、消息序列化、不可靠的网络、异步机制、版本化、差异化的工作负载等，开发人员需要考虑以上的分布式系统问题。

可测性的挑战：在动态环境下服务间的交互会产生非常微妙的行为，难以可视化及全面测试。经典微服务往往不太重视测试，更多的是通过监控发现生产环境的异常，进而快速回滚或采取其他必要的行动。但对于特别在意风险规避监管或投产环境错误会产生显著影响的场景下需要特别注意。