Redis

Redis 就是一个使用 C 语言开发的数据库，不过与传统数据库不同的是 **Redis 的数据是存在内存中的**，也就是它是内存数据库，所以读写速度非常快，也支持**持久化**

Redis 被广泛应用于**缓存**方向，也经常用来做**分布式锁**，甚至是**消息队列**

为什么快

- 绝大部分请求是纯粹的内存操作（非常快速）
- 数据处理阶段采用单线程,避免了不必要的上下文切换和竞争条件
- 非阻塞IO - IO多路复用，Redis采用epoll做为I/O多路复用技术的实现

分布式缓存

分布式缓存主要解决的是单机缓存的容量受服务器限制并且无法保存通用的问题。使用的比较多的主要是 **Memcached** 和 **Redis。**不过，现在基本没有看过还有项目使用 **Memcached** 来做缓存，都是直接用 **Redis**

缓存主要是为了提升用户体验以及应对更多的用户

- 高性能；不用从硬盘中读取，直接操作内存，所以速度相当快
- 高并发；一般像 MySQL 这类的数据库的 QPS 大概都在 1w 左右（4 核 8g），但是使用 Redis 缓存之后很容易达到 10w+，甚至最高能达到 30w+（就单机 redis 的情况，redis 集群的话会更高）。

  > QPS（Query Per Second）：服务器每秒可以执行的查询次数；

常见数据结构

- string；string 数据结构是简单的 key-value 类型，场景：一般常用在需要计数的场景，比如用户的访问次数、热点文章的点赞转发数量、分布式锁等等
- list；即链表（双向），场景：发布与订阅或者说消息队列、慢查询
- hash；类似于HashMap(数组+链表)，场景：适合存储对象
- set；类似于HashSet，场景：不能重复的数据交集
- sorted set；和 set 相比，sorted set 增加了一个权重参数 score，场景：各种礼物排行榜

Redis线程模型

redis单线程

redis分客户端和服务端，一次完整的redis请求事件有多个阶段（客户端到服务器的网络连接-->redis读写事件发生-->redis服务端的数据处理（单线程）-->数据返回）。平时所说的**redis单线程模型，本质上指的是服务端的数据处理阶段**，不牵扯网络连接和数据返回，这是理解redis单线程的第一步

I/O多路复用

I/O指网络I/O

多路指的是多个TCP连接（Socket或Channel）

复用指的是复用一个或者多个线程

> BIO -> NIO：
>
> - 网络IO都是通过Socket实现，Server在某一个端口持续监听，客户端通过Socket（IP+Port）与服务器建立连接（ServerSocket.accept），成功建立连接之后，就可以使用Socket中封装的InputStream和OutputStream进行IO交互了。针对每个客户端，Server都会创建一个新线程专门用于处理。
> - 默认情况下，网络IO是阻塞模式，即服务器线程在数据到来之前处于【阻塞】状态，等到数据到达，会自动唤醒服务器线程，着手进行处理。阻塞模式下，一个线程只能处理一个流的IO事件
> - 为了提升服务器线程处理效率，有以下三种思路
>   - **非阻塞[忙轮询]**:采用死循环方式轮询每一个流，如果有IO事件就处理，这样一个线程可以处理多个流，但效率不高，容易导致CPU空转
>   - **Select代理(无差别轮询)**:可以观察多个流的IO事件，如果所有流都没有IO事件，则将线程进入阻塞状态，如果有一个或多个发生了IO事件，则唤醒线程去处理。但是会遍历所有的流，找出流需要处理的流。如果流个数为N，则时间复杂度为O(N)
>   - **Epoll代理**：Select代理有一个缺点，线程在被唤醒后轮询所有的Stream，会存在无效操作。Epoll哪个流发生了I/O事件会通知处理线程，对流的操作都是有意义的，复杂度降低到了O(1)

Redis内部实现采用epoll，采用了epoll+自己实现的简单的事件框架

redis多线程

Redis 4.0 增加的多线程主要是针对一些大键值对的删除操作的命令，使用这些命令就会使用主处理之外的其他线程来“异步处理”

大体上来说，**Redis 6.0 之前主要还是单线程处理**

Redis6.0 引入多线程主要是为了提高网络 IO 读写性能，但是默认是禁用的，只使用主线程

Redis如何判断数据是否过期

Redis通过过期字典来保存数据过期的时间。过期字典存储在redisDb这个结构里

```
typedef struct redisDb {
    ...
    dict *dict;     //数据库键空间,保存着数据库中所有键值对
    dict *expires   // 过期字典,保存着键的过期时间
    ...
} redisDb;
```

过期的数据的删除策略