1、为什么这么快

1. 完全基于内存操作
2. 单线程运行，省去了上下文切换成本
3. 使用非阻塞的IO多路复用机制
4. 惰性删除机制

2、redis其实是单线程和多线程并存的

单线程部分：使用文件事件分派器来处理事件队列，文件分配器是单线程的

多线程部分：持久化，异步删除，集群数据同步

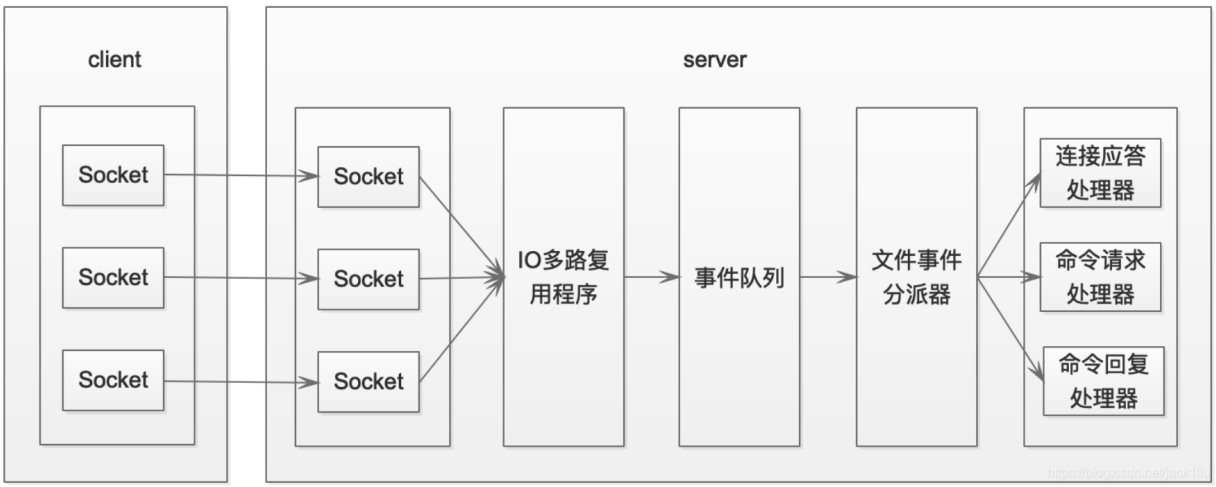
3、什么是IO多路复用技术

一般来说NIO模型是线程发起一个IO请求之后，可以立即返回（非阻塞），返回之后一直等待IO缓冲数据就绪（等待就是同步，不等待这里就变成了AIO），等待的机制就是定时轮询检查。在Java中的NIO（new io）则是发起多个请求之后，有另外的线程不断轮询一堆IO缓冲区，准备好了就进行交给等待的线程去执行

redis利用了linux的epoll命令，将用户socket对应的fd（文件句柄）注册仅epoll，epoll帮忙监听哪些socket有消息到达，这样就避免了大量的无用操作，这其实相当于是io多路复用程序对socket的监听

4、redis的各模块配合机制

如图为 Redis 的客户端与服务端整体事件处理流程图：



流程图主要由以下几部分组成：

- 多路 Socket。
- IO 多路复用程序。
- 事件队列。
- 文件事件分派器。
- 多类型事件处理器。

redis的请求分几种类型：建立连接事件，写、读请求事件，返回结果事件

a. 建立连接事件：

i. io多路复用监听到建立连接的请求事件后，将请求事件写入队列，单线程的文件事件分派器从队列中获取请求事件，交给连接应答处理器进行处理，处理时会与客户端建立一个Socket，实现连接

b. 写，读请求事件：

1. 客户端发送一个读写命令，IO多路复用监听到事件后，写入队列，文件事件分派器从队列获取到事件后，交给命令请求处理器处理，处理后产生一个返回结果事件，写入事件队列，由文件事件分派器交给命令回复处理器处理

redis的瓶颈主要是在网络IO操作上，内部的分派其实并不是短板

Redis 的多线程部分只是用来处理网络数据的读写和协议解析，执行命令仍然是单线程。 之所以这么设计是不想 Redis 因为多线程而变得复杂，需要去控制 key、lua、事务、LPUSH/LPOP 等等的并发问题。

参考连接：https://blog.csdn.net/jack1liu/article/details/115833656