单线程的redis为什么这么快

主要归功于以下三点：

1. 纯内存操作
2. 单线程操作，避免了频繁的上下文切换
3. 采用了非阻塞I/O多路复用机制

补充：I/O多路复用

打一个比方：小曲在S城开了一家快递店，负责同城快送服务。小曲因为资金限制，雇佣了一批快递员，然后小曲发现资金不够了，只狗买一辆车送快递。

经营方式一

客户每送来一份快递，小曲就让一个快递员盯着，然后快递员开车去送快递。慢慢的小曲就发现了这种经营方式存在下述问题：

* 几十个快递员基本上时间都花在了抢车上了，大部分快递员都处在闲置状态，谁抢到了车，谁就能取送快递；
* 随着快递的增多，快递员也越来越多，小曲发现快递店里越来越挤，没办法雇佣新的快递员了；
* 快递员之间的协调很花时间。

综合上述缺点，小曲痛定思痛，提出了下面的经营方式 —— 经营方式二

小曲只雇佣一个快递员。然后呢，客户送来的快递，小曲按送达地点标注好，然后依次放在一个地方。最后，那个快递员依次的去取快递，一次拿一个，然后开着车去送快递，送好了就回来拿下一个快递。

对比

上述两种经营方式对比，是不是明显觉得第二种效率更高、更好呢。在上述比喻中：

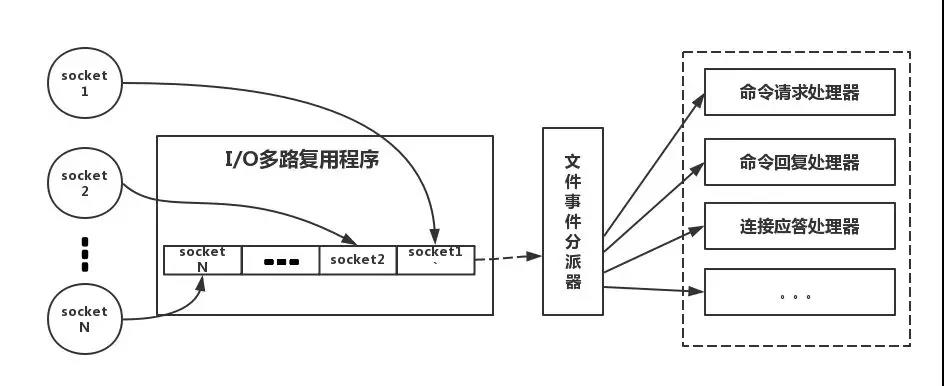
* 每个快递员 ——-> 每个线程
* 每个快递 ——-> 每个socket（I/O流）
* 快递的送达地点 ——-> socket的不同状态
* 客户送快递的请求 ——-> 来自客户端的请求
* 小曲的经营方式 ——-> 服务端运行的代码
* 一辆车 ——-> CPU的一个核

于是我们有如下结论

经营方式一就是传统的并发模型，每个I/O流（快递）都有一个新的线程（快递员）管理。

经营方式二就是I/O多路复用。只有单个线程（一个快递员），通过跟踪每个I/O流的状态（每个快递的送达地点）来管理多个I/O流。

下面类比到真实的redis线程模型，如图所示：



参照上图，简单来说就是，我们的redis-client在操作的时候，会产生具有不同事件类型的socket。在服务端，有一段I/O多路复用程序，将其置入队列之中。然后，文件事件分派器，依次去队列中取，转发到不同的事件处理器中。

需要说明的是，这个I/O多路复用机制，redis还提供了select、epoll、evport、kqueue等多路复用函数库，大家可以自行去了解。