# 1 什么是C10K问题？

英文原址 : <http://www.kegel.com/c10k.html>

中文翻译： https://www.oschina.net/translate/c10k

对于高性能即时通讯（或者说互联网编程）比较关注的开发者，应该对C10K问题（即单机1万个并发连接问题）都有所了解。“C10K”概念最早由Dan Kegel发布于个人站点，即出自其经典的《The C10K problem》一文。

截止2016年，高性能网络编程技术领域里经过众多开发者的努力，已很好地解决C10K问题

即便如此，对于即时通讯（或者说网络编程）开发者而言，研究C10K问题仍然价值巨大，因为技术的发展都是有规律和线索可循的，了解C10K问题及其解决思路，通过举一反三，或许可以为你以后面对类似问题提供更多可借鉴的思想和解决问题的实践思路。

参考：<https://www.cnblogs.com/zafu/p/8251936.html>

# 2 C10K问题系列文章

本文是C10K问题系列文章中的第2篇，总目录如下：

1. 《[高性能网络编程(一)：单台服务器并发TCP连接数到底可以有多少](http://www.52im.net/thread-561-1-1.html" \t "_blank)》
2. 《[高性能网络编程(二)：上一个10年，著名的C10K并发连接问题](http://www.52im.net/thread-566-1-1.html" \t "_blank)》（本文）
3. 《[高性能网络编程(三)：下一个10年，是时候考虑C10M并发问题了](http://www.52im.net/thread-568-1-1.html)》
4. 《[高性能网络编程(四)：从C10K到C10M高性能网络应用的理论探索](http://www.52im.net/thread-578-1-1.html" \t "_blank)》
5. 《[高性能网络编程经典：《The C10K problem(英文)》[附件下载]](http://www.52im.net/thread-560-1-1.html" \t "_blank)》

# 3、C10K问题的提出者

<ignore\_js\_op>   
Dan Kegel：软件工程师  
  
目前工作在美国的洛杉矶，当前受雇于Google公司。从1978年起开始接触计算机编程，是Winetricks的作者、也是Wine 1.0的管理员，同时也是Crosstool（ 一个让 gcc/glibc 编译器更易用的工具套件）的作者。发表了著名的《The C10K problem》技术文章，是Java JSR-51规范的提交者并参与编写了Java平台的NIO和文件锁，同时参与了RFC 5128标准中有关NAT 穿越（P2P打洞）技术的描述和定义。

# 4、C10K问题的由来

大家都知道互联网的基础就是网络通信，早期的互联网可以说是一个小群体的集合。互联网还不够普及，用户也不多，一台服务器同时在线100个用户估计在当时已经算是大型应用了，所以并不存在什么 C10K 的难题。互联网的爆发期应该是在www网站，浏览器，雅虎出现后。最早的互联网称之为Web1.0，互联网大部分的使用场景是下载一个HTML页面，用户在浏览器中查看网页上的信息，这个时期也不存在C10K问题。  
  
Web2.0时代到来后就不同了，一方面是普及率大大提高了，用户群体几何倍增长。另一方面是互联网不再是单纯的浏览万维网网页，逐渐开始进行交互，而且应用程序的逻辑也变的更复杂，从简单的表单提交，到即时通信和在线实时互动，C10K的问题才体现出来了。因为每一个用户都必须与服务器保持TCP连接才能进行实时的数据交互，诸如Facebook这样的网站同一时间的并发TCP连接很可能已经过亿。

早期的腾讯QQ也同样面临C10K问题，只不过他们是用了UDP这种原始的包交换协议来实现的，绕开了这个难题，当然过程肯定是痛苦的。如果当时有epoll技术，他们肯定会用TCP。众所周之，后来的手机QQ、微信都采用TCP协议。  
  
实际上当时也有异步模式，如：select/poll模型，这些技术都有一定的缺点：如selelct最大不能超过1024、poll没有限制，但每次收到数据需要遍历每一个连接查看哪个连接有数据请求。

这时候问题就来了，最初的服务器都是基于进程/线程模型的，新到来一个TCP连接，就需要分配1个进程（或者线程）。而进程又是操作系统最昂贵的资源，一台机器无法创建很多进程。如果是C10K就要创建1万个进程，那么单机而言操作系统是无法承受的（往往出现效率低下甚至完全瘫痪）。如果是采用分布式系统，维持1亿用户在线需要10万台服务器，成本巨大，也只有Facebook、Google、雅虎等巨头才有财力购买如此多的服务器。  
  
基于上述考虑，如何突破单机性能局限，是高性能网络编程所必须要直面的问题。这些局限和问题最早被Dan Kegel 进行了归纳和总结，并首次成系统地分析和提出解决方案，后来这种普遍的网络现象和技术局限都被大家称为 C10K 问题。

# 5、技术解读C10K问题

C10K 问题的最大特点是：设计不够良好的程序，其性能和连接数及机器性能的关系往往是非线性的。  
  
举个例子：如果没有考虑过 C10K 问题，一个经典的基于 select 的程序能在旧服务器上很好处理 1000 并发的吞吐量，它在 2 倍性能新服务器上往往处理不了并发 2000 的吞吐量。这是因为在策略不当时，大量操作的消耗和当前连接数 n 成线性相关。会导致单个任务的资源消耗和当前连接数的关系会是 O(n)。而服务程序需要同时对数以万计的socket 进行 I/O 处理，积累下来的资源消耗会相当可观，这显然会导致系统吞吐量不能和机器性能匹配。  
  
以上这就是典型的C10K问题在技术层面的表现。这也是为何同样的功能，大多数开发人员都能很容易地从功能上实现，但一旦放到大并发场景下，初级与高级开发者对同一个功能的技术实现所体现出的实际应用效果，则是截然不同的。  
  
所以说，一些没有太多大并发实践经验的技术同行，所实现的诸如即时通讯应用在内的网络应用，所谓的理论负载动不动就宣称能支持单机上万、上十万甚至上百万的情况，是经不起检验和考验的。

# 6、C10K问题的本质

C10K问题本质上是操作系统的问题。对于Web1.0/2.0时代的操作系统而言， 传统的同步阻塞I/O模型都是一样的，处理的方式都是requests per second，并发10K和100的区别关键在于CPU。  
  
创建的进程线程多了，数据拷贝频繁（缓存I/O、内核将数据拷贝到用户进程空间、阻塞）， 进程/线程上下文切换消耗大， 导致操作系统崩溃，这就是C10K问题的本质！  
  
可见，解决C10K问题的关键就是尽可能减少这些CPU等核心计算资源消耗，从而榨干单台服务器的性能，突破C10K问题所描述的瓶颈。

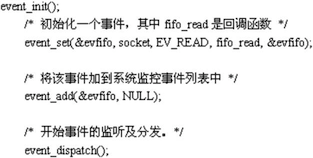
# 7、C10K问题的解决方案探讨 要解决这一问题，从纯网络编程技术角度看，主要思路有两个：

* 一个是对于每个连接处理分配一个独立的进程/线程；
* 另一个思路是用同一进程/线程来同时处理若干连接。

## 1思路一：每个进程/线程处理一个连接 这一思路最为直接。但是由于申请进程/线程会占用相当可观的系统资源，同时对于多进程/线程的管理会对系统造成压力，因此这种方案不具备良好的可扩展性。 因此，这一思路在服务器资源还没有富裕到足够程度的时候，是不可行的。即便资源足够富裕，效率也不够高。总之，此思路技术实现会使得资源占用过多，可扩展性差。

## 2思路二：每个进程/线程同时处理多个连接（IO多路复用） IO多路复用从技术实现上又分很多种，我们逐一来看看下述各种实现方式的优劣。 ● 实现方式1：传统思路最简单的方法是循环挨个处理各个连接，每个连接对应一个 socket，当所有 socket 都有数据的时候，这种方法是可行的。但是当应用读取某个 socket 的文件数据不 ready 的时候，整个应用会阻塞在这里等待该文件句柄，即使别的文件句柄 ready，也无法往下处理。 实现小结：直接循环处理多个连接。 问题归纳：任一文件句柄的不成功会阻塞住整个应用。 ● 实现方式2：select要解决上面阻塞的问题，思路很简单，如果我在读取文件句柄之前，先查下它的状态，ready 了就进行处理，不 ready 就不进行处理，这不就解决了这个问题了嘛？于是有了 select 方案。用一个 fd\_set 结构体来告诉内核同时监控多个文件句柄，当其中有文件句柄的状态发生指定变化（例如某句柄由不可用变为可用）或超时，则调用返回。之后应用可以使用 FD\_ISSET 来逐个查看是哪个文件句柄的状态发生了变化。这样做，小规模的连接问题不大，但当连接数很多（文件句柄个数很多）的时候，逐个检查状态就很慢了。因此，select 往往存在管理的句柄上限（FD\_SETSIZE）。同时，在使用上，因为只有一个字段记录关注和发生事件，每次调用之前要重新初始化 fd\_set 结构体。

|  |  |
| --- | --- |
| 1 | intselect(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout); |

实现小结：有连接请求抵达了再检查处理。  
问题归纳：句柄上限+重复初始化+逐个排查所有文件句柄状态效率不高。  
  
● 实现方式3：poll 主要解决 select 的前两个问题：通过一个 pollfd 数组向内核传递需要关注的事件消除文件句柄上限，同时使用不同字段分别标注关注事件和发生事件，来避免重复初始化。  
  
实现小结：设计新的数据结构提供使用效率。  
问题归纳：逐个排查所有文件句柄状态效率不高。  
  
● 实现方式4：epoll既然逐个排查所有文件句柄状态效率不高，很自然的，如果调用返回的时候只给应用提供发生了状态变化（很可能是数据 ready）的文件句柄，进行排查的效率不就高多了么。epoll 采用了这种设计，适用于大规模的应用场景。实验表明，当文件句柄数目超过 10 之后，epoll 性能将优于 select 和 poll；当文件句柄数目达到 10K 的时候，epoll 已经超过 select 和 poll 两个数量级。  
  
实现小结：只返回状态变化的文件句柄。  
问题归纳：依赖特定平台（Linux）。  
  
因为Linux是互联网企业中使用率最高的操作系统，Epoll就成为C10K killer、高并发、高性能、异步非阻塞这些技术的代名词了。FreeBSD推出了kqueue，Linux推出了epoll，Windows推出了IOCP，Solaris推出了/dev/poll。这些操作系统提供的功能就是为了解决C10K问题。epoll技术的编程模型就是异步非阻塞回调，也可以叫做Reactor，事件驱动，事件轮循（EventLoop）。Nginx，libevent，node.js这些就是Epoll时代的产物。  
  
● 实现方式5：由于epoll, kqueue, IOCP每个接口都有自己的特点，程序移植非常困难，于是需要对这些接口进行封装，以让它们易于使用和移植，其中libevent库就是其中之一。跨平台，封装底层平台的调用，提供统一的 API，但底层在不同平台上自动选择合适的调用。按照libevent的官方网站，libevent库提供了以下功能：当一个文件描述符的特定事件（如可读，可写或出错）发生了，或一个定时事件发生了，libevent就会自动执行用户指定的回调函数，来处理事件。目前，libevent已支持以下接口/dev/poll, kqueue, event ports, select, poll 和 epoll。Libevent的内部事件机制完全是基于所使用的接口的。因此libevent非常容易移植，也使它的扩展性非常容易。目前，libevent已在以下操作系统中编译通过：Linux，BSD，Mac OS X，Solaris和Windows。使用libevent库进行开发非常简单，也很容易在各种unix平台上移植。一个简单的使用libevent库的程序如下：  
  
<ignore\_js\_op> 

# 8、参考资料

[1] [为什么QQ用的是UDP协议而不是TCP协议？](http://www.52im.net/thread-279-1-1.html)  
[2] [移动端IM/推送系统的协议选型：UDP还是TCP？](http://www.52im.net/thread-33-1-1.html)  
[3] [高性能网络编程经典：《The C10K problem(英文)》[附件下载]](http://www.52im.net/thread-560-1-1.html)  
[4] [高性能网络编程(一)：单台服务器并发TCP连接数到底可以有多少](http://www.52im.net/thread-561-1-1.html)  
[5] 《The C10K problem （[英文在线阅读](http://www.kegel.com/c10k.html)、[英文PDF版下载](http://www.52im.net/thread-560-1-1.html)、[中文译文](http://blog.csdn.net/goldou/article/details/2579781)）》  
[6] <ignore\_js\_op>http://www.52im.net/static/image/filetype/pdf.gif [搜狗实验室技术交流文档《C10K问题探讨》(52im.net).pdf](http://www.52im.net/forum.php?mod=attachment&aid=MTUyNnxiODNmMTJlZXwxNTE1NDg1MzAxfDB8NTY2) (350.83 KB , 下载次数: 23 , 售价: 2 金币)   
[7] [[通俗易懂]深入理解TCP协议（上）：理论基础](http://www.52im.net/thread-513-1-1.html)  
[8] [[通俗易懂]深入理解TCP协议（下）：RTT、滑动窗口、拥塞处理](http://www.52im.net/thread-515-1-1.html)  
[9] 《[TCP/IP详解 卷1：协议 (在线阅读版)](http://www.52im.net/topic-tcpipvol1.html)》

# 9、更多资料

《[TCP/IP详解](http://www.52im.net/topic-tcpipvol1.html" \t "_blank) - [第11章·UDP：用户数据报协议](http://docs.52im.net/extend/docs/book/tcpip/vol1/11/)》  
《[TCP/IP详解](http://www.52im.net/topic-tcpipvol1.html" \t "_blank) - [第17章·TCP：传输控制协议](http://docs.52im.net/extend/docs/book/tcpip/vol1/17/)》  
《[TCP/IP详解](http://www.52im.net/topic-tcpipvol1.html) - [第18章·TCP连接的建立与终止](http://docs.52im.net/extend/docs/book/tcpip/vol1/18/)》  
《[TCP/IP详解](http://www.52im.net/topic-tcpipvol1.html) - [第21章·TCP的超时与重传](http://docs.52im.net/extend/docs/book/tcpip/vol1/21/)》  
《[技术往事：改变世界的TCP/IP协议（珍贵多图、手机慎点）](http://www.52im.net/thread-520-1-1.html" \t "_blank)》  
《[通俗易懂-深入理解TCP协议（上）：理论基础](http://www.52im.net/thread-513-1-1.html" \t "_blank)》  
《[通俗易懂-深入理解TCP协议（下）：RTT、滑动窗口、拥塞处理](http://www.52im.net/thread-515-1-1.html" \t "_blank)》  
《[理论经典：TCP协议的3次握手与4次挥手过程详解](http://www.52im.net/thread-258-1-1.html" \t "_blank)》  
《[理论联系实际：Wireshark抓包分析TCP 3次握手、4次挥手过程](http://www.52im.net/thread-275-1-1.html" \t "_blank)》  
《[计算机网络通讯协议关系图（中文珍藏版）](http://www.52im.net/thread-180-1-1.html)》  
《[UDP中一个包的大小最大能多大？](http://www.52im.net/thread-29-1-1.html" \t "_blank)》  
《[Java新一代网络编程模型AIO原理及Linux系统AIO介绍](http://www.52im.net/thread-306-1-1.html)》  
《[NIO框架入门(一)：服务端基于Netty4的UDP双向通信Demo演示](http://www.52im.net/thread-367-1-1.html" \t "_blank)》  
《[NIO框架入门(二)：服务端基于MINA2的UDP双向通信Demo演示](http://www.52im.net/thread-373-1-1.html)》  
《[NIO框架入门(三)：iOS与MINA2、Netty4的跨平台UDP双向通信实战](http://www.52im.net/thread-378-1-1.html)》  
《[NIO框架入门(四)：Android与MINA2、Netty4的跨平台UDP双向通信实战](http://www.52im.net/thread-388-1-1.html)》  
《[P2P技术详解(一)：NAT详解——详细原理、P2P简介](http://www.52im.net/thread-50-1-1.html)》  
《[P2P技术详解(二)：P2P中的NAT穿越(打洞)方案详解](http://www.52im.net/thread-542-1-1.html)》  
《[P2P技术详解(三)：P2P技术之STUN、TURN、ICE详解](http://www.52im.net/thread-557-1-1.html)》  
《[高性能网络编程(一)：单台服务器并发TCP连接数到底可以有多少](http://www.52im.net/thread-561-1-1.html)》  
《[高性能网络编程(二)：上一个10年，著名的C10K并发连接问题](http://www.52im.net/thread-566-1-1.html)》  
《[高性能网络编程(三)：下一个10年，是时候考虑C10M并发问题了](http://www.52im.net/thread-568-1-1.html)》  
《[高性能网络编程(四)：从C10K到C10M高性能网络应用的理论探索](http://www.52im.net/thread-578-1-1.html)》