# linux大并发服务器开发

### 1.大并发服务器思路

#### 1.应用程序角度

|  |
| --- |
| 1.消息队列+数据连接池+线程池 |
| 2.应用服务器的负载均衡，增加一个任务服务器来实现，任务服务器可以监视应用服务器的负载，CPU高、IO高、并发高、内存换页高查询到这些信息之后，选择负载最低的服务器分配任务 应用服务器被动的接收任务。应用服务器主动到任务服务器接收任务进行处理，这种做法更科学 |

##### 服务器性能的四大杀手

|  |  |
| --- | --- |
| 1.数据copy | 缓存解决 |
| 2.环境切换 | （理性创建线程）该不该用多线程 单核服务器（采用状态机编程，效率最高）大量的任务提交到服务器，增加了线程间的切换开销 多进程切换一样的思想。多核服务器，多线程能够充分发挥多核性能 |
| 3.内存分配 | 内存池 |
| 4.锁竞争 | 尽可能减少锁的竞争 |

#### 2.数据库角度

|  |
| --- |
| 1. 主要的业务逻辑移到应用服务器处理，数据库只做辅助的逻辑处理 |
| 1. 数据库读写分离 数据库的读操作 >> 写操作，对数据库进行负载均衡 replication机制 |
| 1. 数据分区 （分库，分表） 分库，垂直分区（用户库，业务库，基础库） 水平分区（每个库都有用户表，业务表，基础表） |

#### 3.缓存机制

|  |
| --- |
| 1.缓存，减少服务器的访问。最大问题是缓存的更新 --> 缓存的time out。如果缓存失效，重新去数据库查询，这种方法实时性比较差。一旦数据库中数据更新，则立即通知cache更新 |
| 2. 如果数据量较大，内存不够，将不活跃的数据换出内存。换出算法 FIFO LRU（least recently used最近最少使用）LFU(least frequently used 最不频繁使用) |
| 3. nosql （反sql）非关系数据 key/value 分布式redis/memcache |

### 2.细节点

|  |
| --- |
| 网络I/O |
| signal(SIGPIPE, SIG\_IGN)  如果客户端关闭套接字  而服务器调用了一次write，服务器会接受一个RST sgement（TCP传输层）  如果服务器再次调用了write，这个时候就会产生SIGPIPE信号。 |
| TIME\_WAIT状态，对大并发服务器的影响  尽量避免服务器避免出现 TIME\_WAIT状态  如果服务端主动断开连接（先于client调用close），服务端就会进入TIME\_WAIT，而内核保存了一些资源。在协议设计上，应该让客户端主动断开连接，这样就把TIME\_WAIT状态分散到大量的客户端。如果客户端不活跃了，一些恶意的客户端不断开连接，这样就会占用的服务端的连接资源，所以服务端也要有个踢掉不活跃的连接机制。 |