进程、线程、协程

参考链接：<https://www.cnblogs.com/Survivalist/p/11527949.html>

进程：进程是一个具有独立功能的程序在一个数据集上的一次动态执行的过程，是操作系统进行资源分配和调度的一个独立单位，是应用程序的载体。由程序、数据集合和进程控制块组成

线程：线程时程序中执行的一个单一的顺序控制流程，时程序执行流的最小单元，时处理器调度和分派的基本单位；一个进程可以有一个或多个线程

协程：一种基于线程之上，但又比线程更加轻量级的存在，由程序员自己写程序来管理的轻量级线程叫用户空间线程，具有对内核来说不可见的特性

进程与现成的区别：

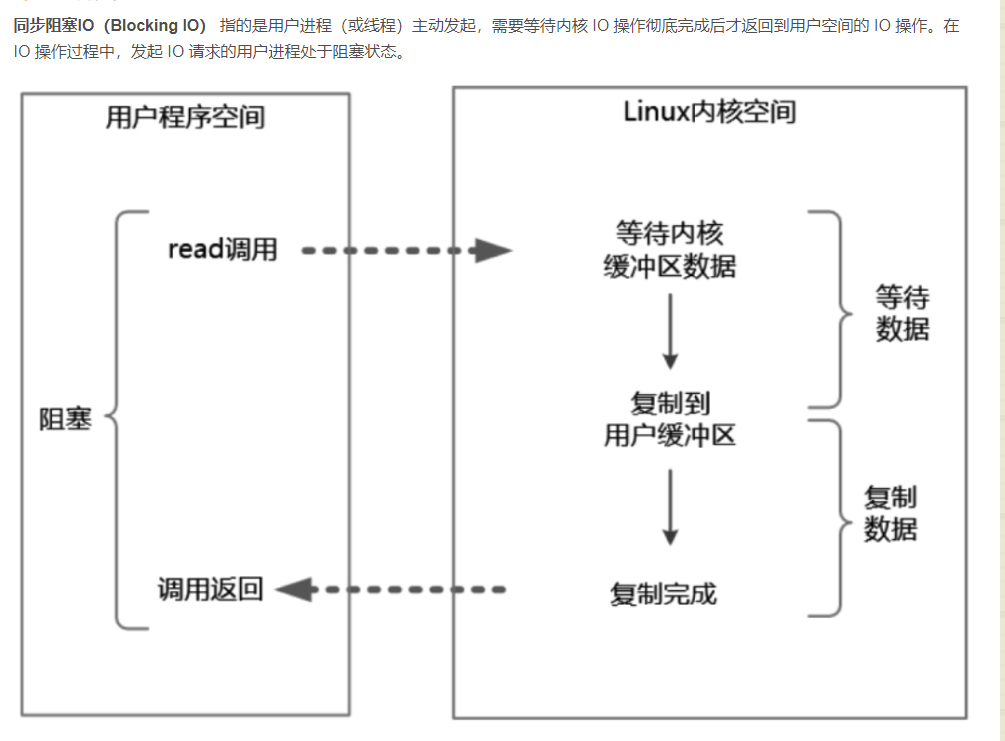
1. 线程时程序执行的最小单位，而进程时操作系统分配资源的最小单位
2. 一个进程由多个线程组成，线程时一个进程中代码的不同执行线路
3. 进程之间相互独立，而同一个进程下的各个线程之间共享程序的内存空间及资源，某进程内的线程在其他进程是不可见的
4. 调度和切换，线程的上下文切换比进程要快很多

线程与协程的区别：

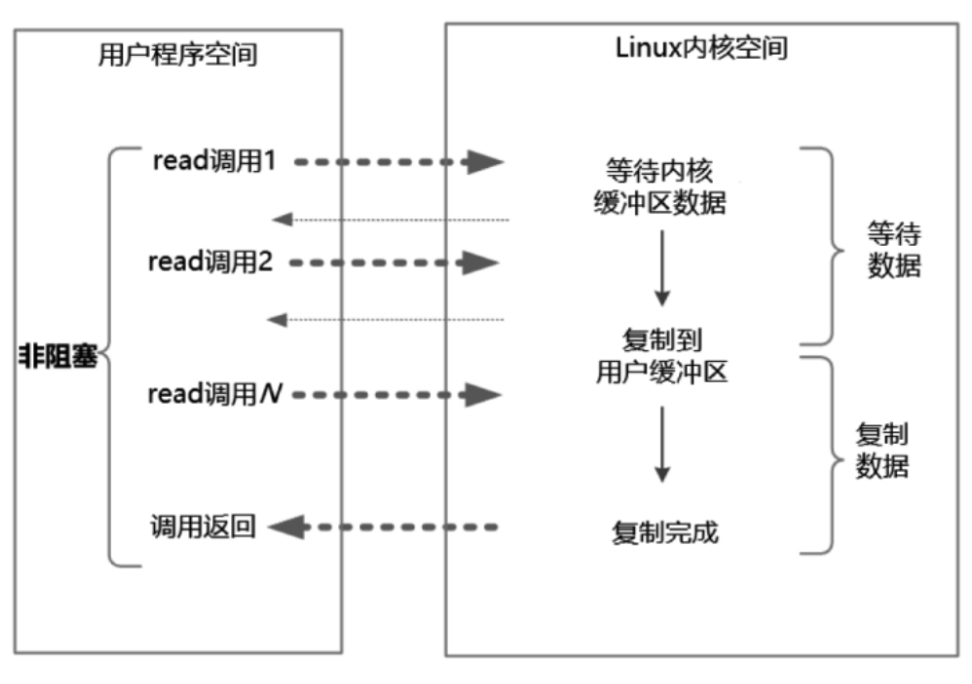
1. 线程的切换由操作系统负责调度，协程由用户自己进行调度，因此可以减少上下文切换，提高效率
2. 协程的默认Stack大小时1M，而协程更轻量，接近1K，因此可以在相同的内存中开启更多的协程
3. 由于在同一个线程上，因此可以避免竞争关系而使用锁
4. 适用于被阻塞的、需要大量并发的场景，但不适用与大量计算的多线程

IO模型

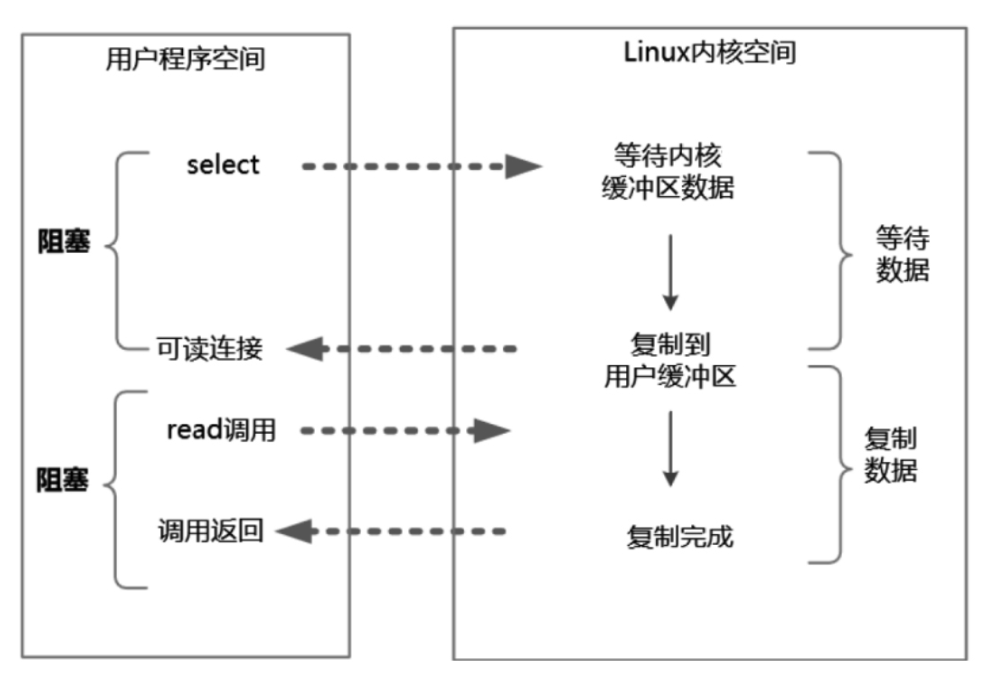
同步阻塞



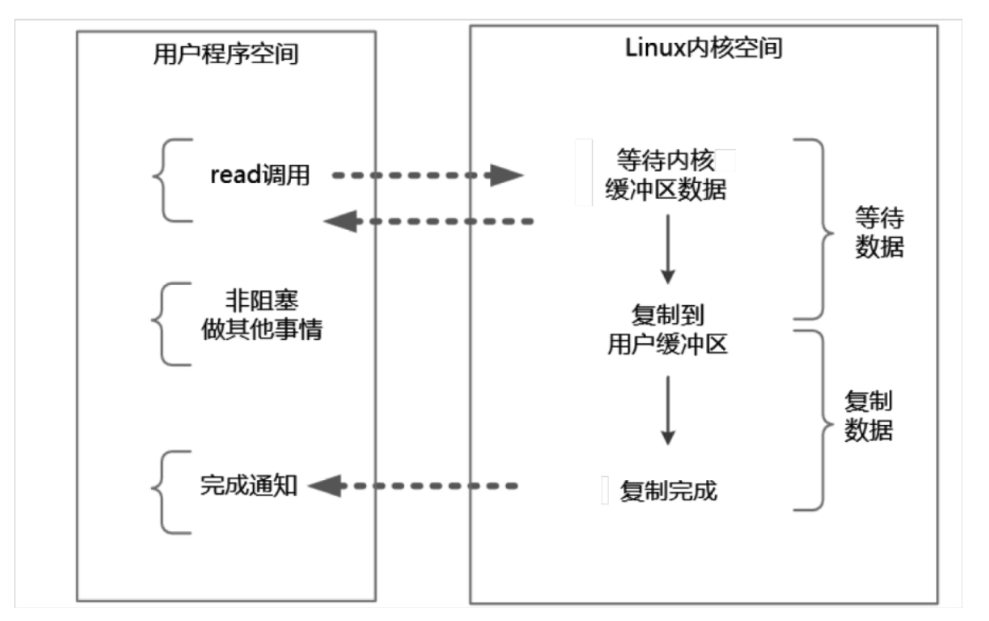
同步非阻塞



IO多路复用



异步IO



信号驱动IO

当进程发起一个 IO 操作，会向内核注册一个信号处理函数，然后进程返回不阻塞；当内核数据就绪时会发送一个信号给进程，进程便在信号处理函数中调用 IO 读取数据。

信号驱动 IO 不同于 AIO 的是依旧存在阻塞状态，即用户进程获取到数据就绪信号后阻塞进行 IO 操作。

IO多路复用

参考地址：<https://blog.csdn.net/adminpd/article/details/124553590>

Select调用过程:

1）用户进程需要监控某些资源 fds，在调用 select 函数后会阻塞，操作系统会将用户线程加入这些资源的等待队列中。

2）直到有描述符就绪（有数据可读、可写或有 except）或超时（timeout 指定等待时间，如果立即返回设为 null 即可），函数返回。

3）select 函数返回后，中断程序唤起用户线程。用户可以遍历 fds，通过 FD\_ISSET 判断具体哪个 fd 收到数据，并做出相应处理。

select 的缺点

1）每次调用 select 都需要将进程加入到所有监视 fd 的等待队列，每次唤醒都需要从每个队列中移除。 这里涉及了两次遍历，而且每次都要将整个 fd\_set 列表传递给内核，有一定的开销。

2）当函数返回时，系统会将就绪描述符写入 fd\_set 中，并将其拷贝到用户空间。进程被唤醒后，用户线程并不知道哪些 fd 收到数据，还需要遍历一次。

受 fd\_set 的大小限制，32 位系统最多能监听 1024 个 fd，64 位最多监听 2048 个。

Poll原理与select相似，都需要来回拷贝全部监听的文件描述符，不同的是：

1）poll 函数采用链表的方式替代原来 select 中 fd\_set 结构，因此可监听文件描述符数量不受限。

2）poll 函数返回后，可以通过 pollfd 结构中的内容进行处理就绪文件描述符，相比 select 效率要高。

3）新增水平触发：也就是通知程序 fd 就绪后，这次没有被处理，那么下次 poll 的时候会再次通知同个 fd 已经就绪。

epoll

epoll 使用一个文件描述符管理多个描述符，将用户进程监控的文件描述符的事件存放到内核的一个事件表中，这样在用户空间和内核空间只需拷贝一次。

epoll为何高效：

1） epoll 精巧的使用了 3 个方法来实现 select 方法要做的事，分清了频繁调用和不频繁调用的操作。

epoll\_ctrl 是不太频繁调用的，而 epoll\_wait 是非常频繁调用的。而 epoll\_wait 却几乎没有入参，这比 select 的效率高出一大截，而且，它也不会随着并发连接的增加使得入参越发多起来，导致内核执行效率下降。

2） mmap 的引入，将用户空间的一块地址和内核空间的一块地址同时映射到相同的一块物理内存地址（不管是用户空间还是内核空间都是虚拟地址，最终要通过地址映射映射到物理地址），使得这块物理内存对内核和对用户均可见，减少用户态和内核态之间的数据交换。

3）红黑树将存储 epoll 所监听的 FD。高效的数据结构，本身插入和删除性能比较好，时间复杂度O(logN)。

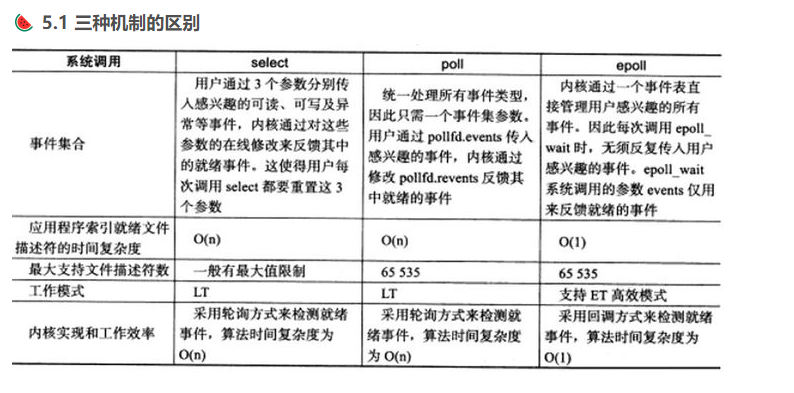
epoll优点：

1）没有最大并发连接的限制，能打开的 FD 的上限远大于 1024。

2）效率提升，不是轮询的方式，不会随着 FD 数目的增加效率下降。

3）内存拷贝，利用 mmap() 文件映射内存加速与内核空间的消息传递，即 epoll 使用 mmap 减少复制开销。

4）新增 ET 模式。



Php.swoole,go的进程模型的区别

为什么swoole、go更适合做高并发业务

IO模型

go语言使用单线程eventloop处理IO事件，多线程实现协程调度，执行用户层代码

swoole使用多线程eventloop处理IO事件，多进程执行用户层php代码

Go对与IO事件的处理是单线程的，无法利用多核，吞吐量稍弱于swoole

在实际的TCP/UDP密集IO压测中，swoole表现要稍优于go

Go协程(goroutine)是运行在多线程上的，线程可以共享堆栈和文件描述符，功能更强大，在实现连接池、并发库方面更有优势。额外的带来的一个问题是，存在数据同步问题，需要用户自行考虑加锁。

Swoole的用户代码运行在多进程环境，无需考虑加锁问题。但无法直接访问内存和资源。需要借助Task进程实现中转。

语言性能

go语言是静态编译的，语言本身的性能大大超过php，密集计算更有优势

php是动态解释执行的，语言性能较差，不适合密集计算程序

Linux命令，看日志，看cpu使用情况(top, vmstat)，看内存使用情况(free, vmsta)，看进程(ps , top, pstree)，看连接数(netstat -an)

PHP8新特性：

1. 新增联合参数，联合参数使一个变量有多个类型值，而不是一个（类似于C语言中的struct）
2. 新增weakmap新特性，允许你创建对象到任意值得映射，同时也不会阻止作为建的对象被垃圾回收
3. 新增ValueError异常，当传递值到函数时，如果一个时一个无效类型，者会抛出该异常，php8之前仅告警
4. 重写方法时允许可变参数，子类重写父类方法时，任何数量的参数现在都可以被替换成可变参数，只要对应的参数类型兼容即可
5. 静态返回类型，可以使用static关键字标识某个方法返回该方法当前所属的类，即使它是继承的
6. 对象的类名字变量， objet：：class 等于get\_class(object)
7. 变量语法调整， new 和instanceof关键字现在可以用于任意表达式
8. Stringable接口， 只要某个类实现了\_\_toString方法，即被视作自动实现了Stringable接口
9. Trait现在可以定义抽象是否方法
10. Throw现在可已被用作表达式
11. 参数列表中允许出现可选的尾部逗号 function a($para1, $para2,) {}
12. 捕获异常而不存储到变量 catch (InvalidArgumentException) {}
13. 新增对mixed类型的支持
14. 新增对注解的支持
15. 新增构造函数属性提示支持，是一个语法简写而已，可以将属性声明和构造函数属性初始化合并到一起
16. 新增match表达式， echo match(1) {0 =>’c’, 1 => ‘a’,}; => c
17. 新增安全运算符“?->”的支持。$user?->getArrdrss()?->country?->code;
18. 引入了2款JIT编译引擎，Tracing JIT最具前景，可以将php代码转为传统的机器吗，而非通过zend虚拟机来执行

Redis

布隆过滤器：相比set集合的去重功能而言，布隆过滤器在空间上的能节省90%，但是它的不足之处时去重率大约在99%，也就是有1%左右的误判率。

实现原理：

1. 添加key：当使用布隆过滤器添加key时，会使用不同的hash函数对key存储的元素进行hash计算，从而会得到多个哈希值，根据哈希值计算出一个整数索引值，将该索引值与位数组长度取余运算，最终得到一个位数组位置，并将该位置变为1,每个hash函数会计算出一个不同的值
2. 判定是否存在：首先对给定的元素再次执行hash计算，得到与添加元素时相同的位数组位置，判断所得位置是否位1，如果其中一个位0， 则说明元素不存在，若都为1，则说明元素有可能存在
3. 为什么时可能存在：那些被置为1的位置，也可能是其他元素的操作而改变的，这就是布隆过滤器存在误判的根本原因

如何保证数据库和缓存数据一直性：

1. 更新数据时同是删除缓存(业务上是否许可)
2. 更新数据可，监听成功后，异步删除缓存
3. 更新缓存数据时禁用数据库缓存，且走数据库主节点
4. 针对更新的数据，特定时间内读redis主节点，避免主从不一致
5. 进行数据一致性校验

事务ACID：原子性，一致性，隔离性，持久性

读未提交（Read uncommitted）、读提交（read committed）-处理脏读、可重复读（repeatable read）-处理不可重复读和串行化（Serializable）-处理幻读

http1.0 http1.1 http2.0区别

1. 连接方式http1.0为短连接，http1.1支持长连接
2. http1.1加入了大量的状态吧
3. 缓存处理：http1.0主要通过header里的If-Modified-Since,Expires来作为缓存判断的标准，http1.1则引入了更多的缓存控制策略如Entity tag，If-Unmodified-Since, If-Match, If-None-Match 等更多可供选择的缓存头来控制缓存策略。
4. 宽带优化及网络连接的使用：HTTP1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。
5. Host头处理：http1.1在请求头中加入了Host字段

http2.0新特性：

1. 新的二进制格式：HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。
2. 多路复用：即连接共享，即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。
3. Header压缩：HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。
4. 服务端发送：同SPDY一样，HTTP2.0也具有server push功能。

Innodb和Myisam的区别

1、innodb支持事务，而myisam不支持事务。

2、innodb支持外键，而myisam不支持外键。

3、innodb默认表锁，使用索引检索条件时是行锁，而myisam是表锁（每次更新增加删除都会锁住表）。

4、innodb和myisam的索引都是基于b+树，但他们具体实现不一样，innodb的b+树的叶子节点是存放数据的，myisam的b+树的叶子节点是存放指针的。

5、innodb是聚簇索引，必须要有主键，一定会基于主键查询，但是辅助索引就会查询两次，myisam是非聚簇索引，索引和数据是分离的，索引里保存的是数据地址的指针，主键索引和辅助索引是分开的。

6、innodb不存储表的行数，所以select count( \* )的时候会全表查询，而myisam会存放表的行数，select count(\*）的时候会查的很快。