一、阅前热身

为了更加形象的说明同步异步、阻塞非阻塞，我们以小明去买奶茶为例。

1、同步与异步

①同步与异步的理解

> 同步与异步的重点在消息通知的方式上，也就是调用结果通知的方式。

- 同步
  当一个同步调用发出去后，调用者要一直等待调用结果的通知后，才能进行后续的执行

- 异步：
  当一个异步调用发出去后，调用者不能立即得到调用结果的返回。

异步调用，要想获得结果，一般有两种方式：
1、主动轮询异步调用的结果;
2、被调用方通过callback来通知调用方调用结果。

②：生活实例

同步买奶茶：小明点单交钱，然后等着拿奶茶；
异步买奶茶：小明点单交钱，店员给小明一个小票，等小明奶茶做好了，再来取。

异步买奶茶，小明要想知道奶茶是否做好了，有两种方式：
1、小明主动去问店员，一会就去问一下：“奶茶做好了吗？”...直到奶茶做好。
2、等奶茶做好了，店员喊一声：“小明，奶茶好了！”，然后小明去取奶茶。

2、阻塞与非阻塞

①阻塞与非阻塞的理解

> 阻塞与非阻塞的重点在于进/线程等待消息时候的行为，也就是在等待消息的时候，当前进/线程是挂起状态，还是非挂起状态。

- 阻塞
  阻塞调用在发出去后，在消息返回之前，当前进/线程会被挂起，直到有消息返回，当前进/线程才会被激活.

- 非阻塞
  非阻塞调用在发出去后，不会阻塞当前进/线程，而会立即返回。

②：生活实例

阻塞买奶茶：小明点单交钱，干等着拿奶茶，什么事都不做；
非阻塞买奶茶：小明点单交钱，等着拿奶茶，等的过程中，时不时刷刷微博、朋友圈...

3、总结

通过上面的分析，我们可以得知：

> 同步与异步，重点在于消息通知的方式;阻塞与非阻塞，重点在于等消息时候的行为。

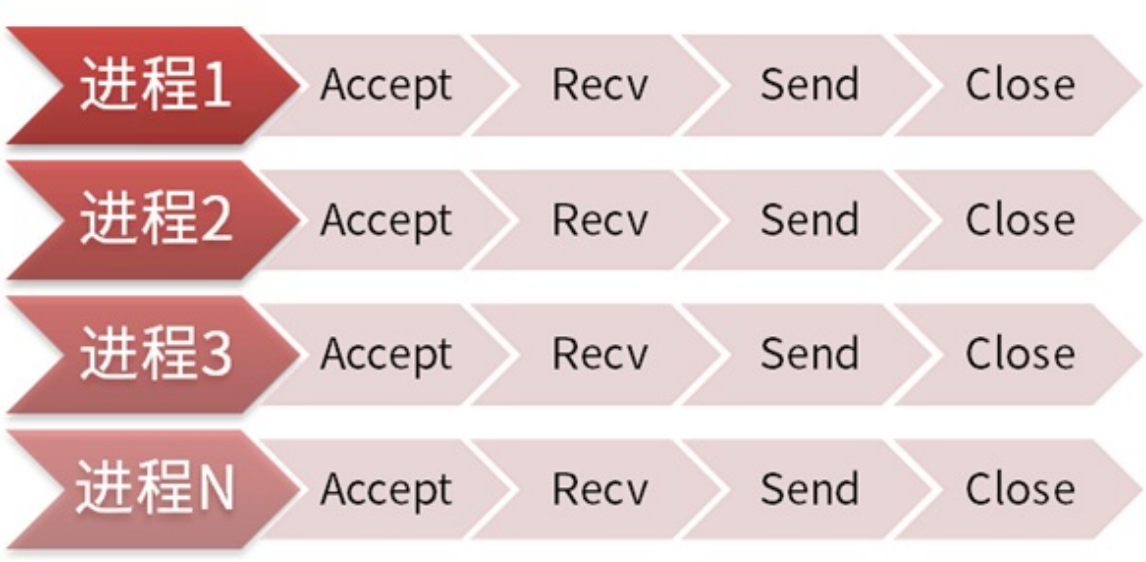
所以，就有了下面4种组合方式

- 同步阻塞：小明在柜台干等着拿奶茶；
- 同步非阻塞：小明在柜台边刷微博边等着拿奶茶；
- 异步阻塞：小明拿着小票啥都不干，一直等着店员通知他拿奶茶；
- 异步非阻塞：小明拿着小票，刷着微博，等着店员通知他拿奶茶。

二、Nginx如何处理高并发

1、Apache面对高并发，为什么很无力？

Apache处理一个请求是**同步阻塞**的模式。



每到达一个请求，Apache都会去fork一个子进程去处理这个请求，直到这个请求处理完毕。

面对低并发，这种模式没什么缺点，但是，面对高并发，就是这种模式的软肋了。

- 1个客户端占用1个进程，那么，进程数量有多少，并发处理能力就有多少，但操作系统可以创建的进程数量是有限的。
- 多进程就会有进程间的切换问题，而进程间的切换调度势必会造成CPU的额外消耗。当进程数量达到成千上万的时候，进程间的切换就占了CPU大部分的时间片，而真正进程的执行反而占了CPU的一小部分，这就得不偿失了。

下面，举例说明这2种场景是多进程模式的软肋：

- 及时消息通知程序
  比如及时聊天程序，一台服务器可能要维持数十万的连接(典型的C10K问题)，那么就要启动数十万的进程来维持。这显然不可能。

- 调用外部Http接口时
  假设Apache启动100个进程来处理请求，每个请求消耗100ms，那么这100个进程能提供1000qps。

但是，在我们调用外部Http接口时，比如QQ登录、微博登录，耗时较长，假设一个请求消耗10s，也就是1个进程1s处理0.1个请求，那么100个进程只能达到10qps，这样的处理能力就未免太差了。

> 注：什么是C10K问题？
> 网络服务在处理数以万计的客户端连接时，往往出现效率低下甚至完全瘫痪，这被称为C10K问题。（concurrent 10000 connection）

综上，我们可以看出，Apache是同步阻塞的多进程模式，面对高并发等一些场景，是很苍白的。

2、Nginx何以问鼎高并发？

传统的服务器模型就是这样，因为其同步阻塞的多进程模型，无法面对高并发。
那么，有没有一种方式，可以让我们在一个进程处理所有的并发I/O呢？
答案是有的，这就是I/O复用技术。

①、I/O复用是神马？

最初级的I/O复用

所谓的I/O复用，就是多个I/O可以复用一个进程。

上面说的同步阻塞的多进程模型不适合处理高并发，那么，我们再来考虑非阻塞的方式。

采用非阻塞的模式，当一个连接过来时，我们不阻塞住，这样一个进程可以同时处理多个连接了。

比如一个进程接受了10000个连接，这个进程每次从头到尾的问一遍这10000个连接：“**有I/O事件没？有的话就交给我处理，没有的话我一会再来问一遍。**”
然后进程就一直从头到尾问这10000个连接，如果这1000个连接都没有I/O事件，就会造成CPU的空转，并且效率也很低，不好不好。

升级版的I/O复用

上面虽然实现了基础版的I/O复用，但是效率太低了。于是伟大的程序猿们日思夜想的去解决这个问题...终于！

我们能不能引入一个代理，这个代理可以同时观察许多I/O流事件呢？

当没有I/O事件的时候，这个进程处于阻塞状态；当有I/O事件的时候，这个代理就去通知进程醒来？

于是，早期的程序猿们发明了两个代理---select、poll。

select、poll代理的原理是这样的：

> 当连接有I/O流事件产生的时候，就会去唤醒进程去处理。

但是进程并不知道是哪个连接产生的I/O流事件，于是进程就挨个去问：“请问是你有事要处理吗？”.....问了99999遍，哦，原来是第100000个进程有事要处理。那么，前面这99999次就白问了，白白浪费宝贵的CPU时间片了！痛哉，惜哉...

注:select与poll原理是一样的，只不过select只能观察1024个连接，poll可以观察无限个连接。

上面看了，select、poll因为不知道哪个连接有I/O流事件要处理，性能也挺不好的。

那么，如果发明一个代理，每次能够知道哪个连接有了I/O流事件，不就可以避免无意义的空转了吗？

于是，超级无敌、闪闪发光的epoll被伟大的程序员发明出来了。

epoll代理的原理是这样的：

> 当连接有I/O流事件产生的时候，epoll就会去告诉进程哪个连接有I/O流事件产生，然后进程就去处理这个进程。

如此，多高效！

②、基于epoll的Nginx

有了epoll，理论上1个进程就可以无限数量的连接，而且无需轮询，真正解决了c10k的问题。

Nginx是基于epoll的，异步非阻塞的服务器程序。自然，Nginx能够轻松处理百万级的并发连接，也就无可厚非了。

三、swoole如何处理高并发以及异步I/O的实现

1、swoole介绍

swoole是PHP的一个扩展。
简单理解：swoole=异步I/O+网络通信
PHPer可以基于swoole去实现过去PHP无法实现的功能。
具体请参考swoole官网：swoole官网

2、swoole如何处理高并发

①Reactor模型介绍

IO复用异步非阻塞程序使用经典的Reactor模型，Reactor顾名思义就是反应堆的意思，它本身不处理任何数据收发。只是可以监视一个socket(也可以是管道、eventfd、信号)句柄的事件变化。

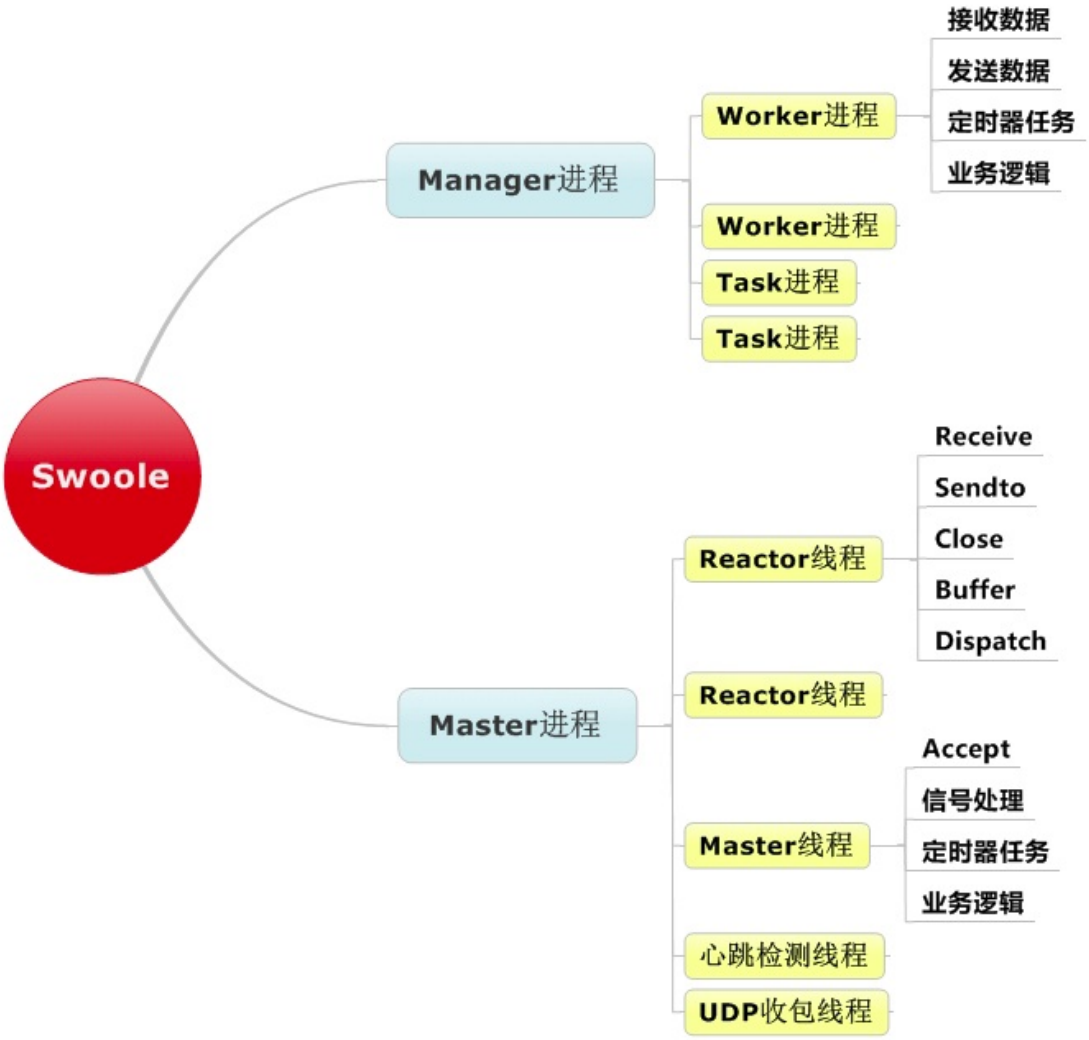
> 注:什么是句柄？句柄英文为handler，可以形象的比喻为锅柄、勺柄。也就是资源的唯一标识符、资源的ID。通过这个ID可以操作资源。

Reactor只是一个事件发生器，实际对socket句柄的操作，如connect/accept、send/recv、close是在callback中完成的。

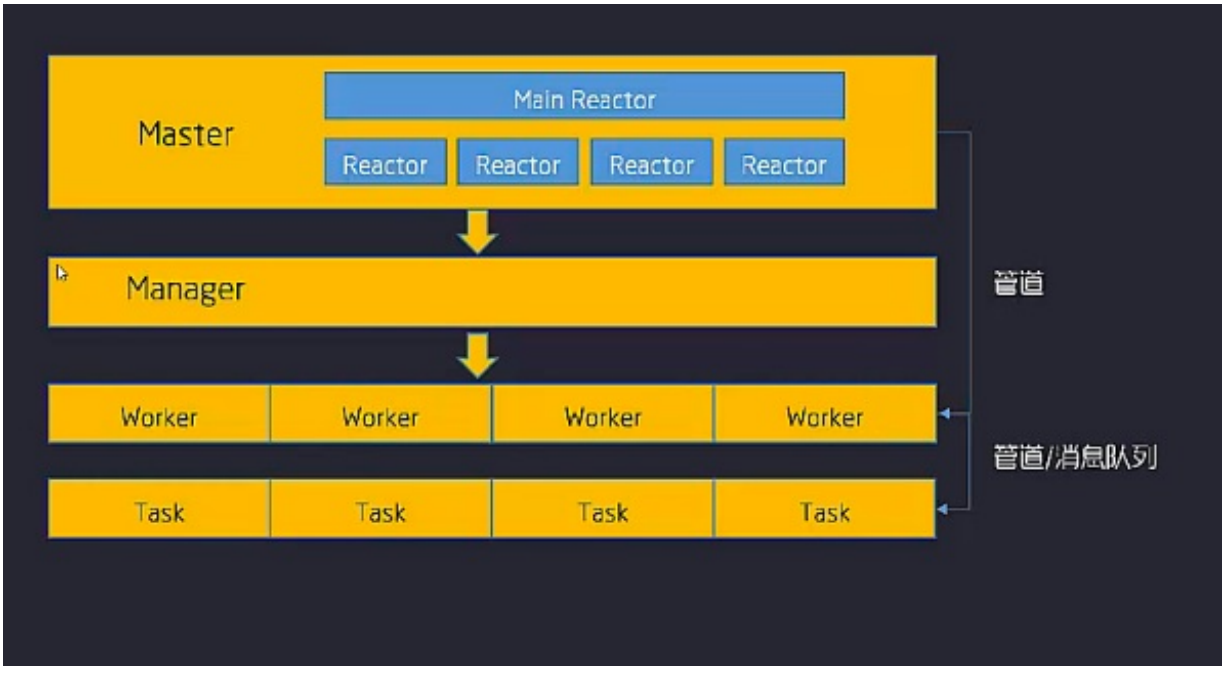
②swoole的架构

swoole采用 **多线程Reactor+多进程Worker**

swoole的架构图如下：



swoole的处理连接流程图如下：



当请求到达时,swoole是这样处理的：

```
请求到达 Main Reactor
        |
        |
Main Reactor根据Reactor的情况，将请求注册给对应的Reactor
(每个Reactor都有epoll。用来监听客户端的变化)
        |
        |
客户端有变化时，交给worker来处理
        |
        |
worker处理完毕，通过进程间通信(比如管道、共享内存、消息队列)发给对应的reactor。
        |
        |
reactor将响应结果发给相应的连接
        |
        |
    请求处理完成
```

因为reactor基于epoll，所以每个reactor可以处理无数个连接请求。
如此，swoole就轻松的处理了高并发。

3、swoole如何实现异步I/O

基于上面的Swoole结构图，我们看到swoole的worker进程有2种类型：
一种是 普通的worker进程，一种是 task worker进程。

worker进程是用来处理普通的耗时不是太长的请求；
task worker进程用来处理耗时较长的请求，比如数据库的I/O操作。

我们以异步Mysql举例：

```
耗时较长的Mysql查询进入worker
             |
             |
worker通过管道将这个请求交给taskworker来处理
             |
             |
worker再去处理其他请求
             |
             |
task worker处理完毕后，处理结果通过管道返回给worker
             |
             |
worker 将结果返回给reactor
             |
             |
reactor将结果返回给请求方
```

如此，通过worker、task worker结合的方式，我们就实现了异步I/O。

四、参考文章

Nginx 多进程模型是如何实现高并发的？
PHP并发IO编程之路
epoll 或者 kqueue 的原理是什么？

IO 多路复用是什么意思？

更多精彩，请关注公众号“聊聊代码”，让我们一起聊聊“左手代码右手诗”的事儿。

你可能感兴趣的文章

目录