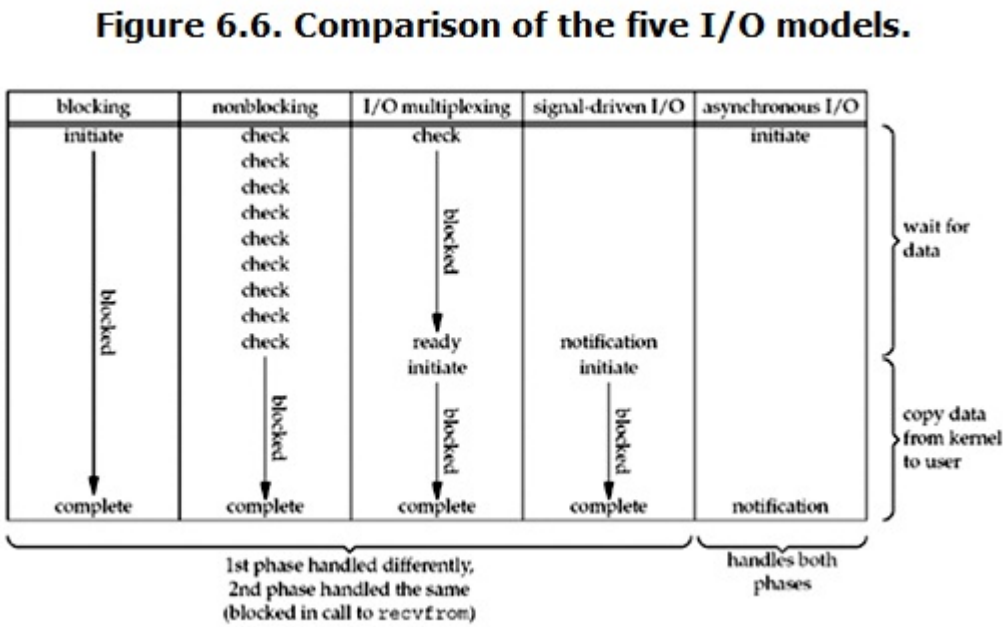
web server io模型演进

解释1

> 阻塞IO和非阻塞IO的区别就在于：阻塞/非阻塞, 它们是程序在等待消息（调用函数）时的状态，应用程序的调用是否立即返回！
>
> 同步IO和异步IO的区别就在于：同步/异步, 它们是消息的通知机制，同步IO是调用完成之后返回来通知的，异步是调用完成之后不等返回，自己做自己的事，然后消息完成之后自然会有它的方式(状态、通知、回调等)来通知它。
>
> 所以，他们关注的点是不一样的，阻塞和非阻塞关注的是调用是否立即返回，同步和异步关注的是消息怎么通知给程序的。

解释2



Stevens给出的定义（其实是POSIX的定义）是这样子的：

> A synchronous I/O operation causes the requesting process to be blocked until that I/O operation completes;
> An asynchronous I/O operation does not cause the requesting process to be blocked;

两者的区别就在于synchronous IO做”IO operation”的时候会将process阻塞。按照这个定义，之前所述的blocking IO，non-blocking IO，IO multiplexing都属于synchronous IO。有人可能会说，non-blocking IO并没有被block啊。这里有个非常“狡猾”的地方，定义中所指的”IO operation”是指真实的IO操作，就是例子中的recvfrom这个system call。non-blocking IO在执行recvfrom这个system call的时候，如果kernel的数据没有准备好，这时候不会block进程。但是，当kernel中数据准备好的时候，recvfrom会将数据从kernel拷贝到用户内存中，这个时候进程是被block了，在这段

时间内，进程是被block的。而asynchronous IO则不一样，当进程发起IO 操作之后，就直接返回再也不理睬了，直到kernel发送一个信号，告诉进程说IO完成。在这整个过程中，进程完全没有被block。

socket阻塞与非阻塞，同步与异步、I/O模型，select与poll、epoll比较
Linux IO模式及 select、poll、epoll详解

下面是各种io模型的演示，参见示例代码

同步阻塞单进程

1. 启动server:
   php socket_server_sync_block.php
2. 分别用telnet启动3个client，并发送数据

```
[shangjie@10-10-213-219 socket]$ telnet 10.10.213.219 1234
Trying 10.10.213.219...
Connected to 10.10.213.219.
Escape character is '^]'.
hi
hi
```

3. 查看server收到消息

```
[shangjie@10-10-213-219 socket]$ php socket_server_sync_block.php
2018-08-14 10:50:21     accept new connection 10.10.213.219:55027
2018-08-14 10:50:23     recv data: "hi\r\n"
2018-08-14 10:50:54     10.10.213.219:55027 idle too long
```

30s后未收到新消息，server断开连接

4. 一个client连接未断开期间新建一个连接，发送一行数据

```
[shangjie@10-10-213-219 socket]$ php socket_server_sync_block.php
2018-08-14 10:50:21     accept new connection 10.10.213.219:55027
2018-08-14 10:50:23     recv data: "hi\r\n"
2018-08-14 10:50:54     10.10.213.219:55027 idle too long


2018-08-14 11:01:04     accept new connection 10.10.213.219:57140
2018-08-14 11:01:05     recv data: "hi\r\n"
2018-08-14 11:01:35     10.10.213.219:57140 idle too long
2018-08-14 11:01:35     accept new connection 10.10.213.219:57163
2018-08-14 11:01:35     recv data: "ggg\r\n"
```

05-35s后client1超时断开，期间client2阻塞（此时发送数据已到达接收缓冲区）

5. 单开一个窗口，用ss命令查看端口连接状态:

```
Every 2.0s: ss state all sport = :1234 -n

Netid  State      Recv-Q Send-Q Local Address:Port               Peer Address:Port
tcp    LISTEN     1      32        *:1234                  *:*
tcp    ESTAB      0      0      10.10.213.219:1234              10.10.213.219:57140
tcp    ESTAB      5      0      10.10.213.219:1234              10.10.213.219:57163
```

可以看到，第一行表示有1个连接**已完成三次握手，等待被accept**，最多可以有32个被等待连接（知识点：tcp三次握手的两个队列，半连接队列和已连接队列 TCP网络编程中connect()、listen()和accept()三者之间的关系）
第二行表示正在处理中的连接，
第三行表示等待被accept的连接，且接收队列中已经累积了5个字节(“ggg\r\n”共5个字节)

Recv-Q Send-Q
对listen状态，Recv-Q表示已建立的连接还有多少未被accept，Send-Q表示backlog即已建立连接队列的最大值
对非listen状态，Recv-Q表示内核的接收队列中未被应用程序处理的字节数，Send-Q表示内核的发送队列中未收到ack的字节数

同步阻塞+多进程

1. 启动server
   php socket_server_multiprocess.php
2. 启动两个client

```
[shangjie@10-10-213-219 socket]$ ps aux|grep php
shangjie  1050  0.0  0.1 263308 12536 pts/26   S+   11:27   0:00 php socket_server_multiprocess.php
shangjie  1112  0.0  0.0 263308  4764 pts/26   S+   11:27   0:00 php socket_server_multiprocess.php
shangjie  1132  0.0  0.2 164428 18012 pts/21   T    Jul06   1:35 vim socket_server_event_echo.php
shangjie  1149  0.0  0.0 263308  4764 pts/26   S+   11:27   0:00 php socket_server_multiprocess.php
```

```
[shangjie@10-10-213-219 socket]$ pstree -ph 1050
php(1050)─┬─php(1112)
          └─php(1149)
```

可以看到一个主进程生成了两个子进程，分别与两个client交互

> 同步阻塞单进程模型最大的问题在于同一个进程即负责监听，又负责通信。当通信阻塞时，无法创建新连接。故用多进程优化为：
>
> 一个监听进程，只负责监听，每接到一个新连接即生成一个子进程与之通信
>
> 多进程的问题在于，占用资源多，能够支持的连接数有限。

leader-follower

> 同步阻塞多进程稍加改造，在server启动时生成多个子进程等待accpt。有新请求到来时只有一个子进程竞争成功，处理请求，完毕后继续等待accept。

> php-fpm, apache, workerman
> 多进程模型的缺点在于
> 这种模型严重依赖进程的数量解决并发问题，一个客户端连接就需要占用一个进程，工作进程的数量有多少，并发处理能力就有多少。操作系统可以创建的进程数量是有限的。
> 启动大量进程会带来额外的进程调度消耗。数百个进程时可能进程上下文切换调度消耗占CPU不到1%可以忽略不计，如果启动数千甚至数万个进程，消耗就会直线上升。调度消耗可能占到 CPU 的百分之几十甚至 100%。

IO多路复用（IO异步化）

> 多路复用意为可在一个进程内同时处理多个连接， 对socket来说有select/poll/epoll/kqueue这些IO模型可以同时处理多个连接的读写事件。
> 在各个不同的操作系统平台上的接口不同，也产生了很多对它们的封装。例如 libevent、libev、libuv
> pecl 中提供一些 PHP 扩展，如 event 是 libevent 的扩展，ev 是 libev 的扩展。

select

1. 启动server
   php socket_server_select.php
2. 启动多个client

```
[shangjie@10-10-213-219 socket]$ php socket_server_select.php
Connection accepted from 10.10.213.219:59451
Now there are total 1 clients.
Connection accepted from 10.10.213.219:59462
Now there are total 2 clients.
recv msg:hi

send msg:hi

recv msg:go

send msg:go
```

server与client交互效果和多进程类似，可同时处理多个连接，只需要一个server进程

> 使用select来处理监听及消息通信
> 对监听端口来说，有可读事件的含义为有新连接请求，此时执行accept并将新端口加入可读集合；
>
> 对非监听端口，有可读事件的含义为有新消息或对方关闭连接，区分处理即可。
> select模型最大的问题大于可支持并发数有限，为1024（由操作系统参数 __FD_SETSIZE指

> 定， 可打开的最大文件描述符），且性能随连接数增加下降（这个也很好理解，如同时监听1000个连接的可读事件，当只有一个连接可读时，select会遍历所有连接）

poll

> 解决了1024连接数过少的问题，可支持几十w连接，但轮询检测事件的问题仍然存在。

epoll

> 可维护任意数量连接，且无需轮询。

even扩展

php使用event扩展
按epoll->poll->select的顺序取得可用的事件模型
代码参见socket_server_event_simple.php和socket_server_event_echo.php

ev扩展

php使用ev扩展
代码参见socket_server_ev_echo.php

使用PHP编写基于事件驱动的HTTP Server

io异步化的代码中如果执行同步逻辑，整体会退化为同步执行
php中常见的curl, file_get_contents, mysql, redis访问等均为同步执行，可以使用第三方实现的异步组件来替代

IO多路复用+多进程

单个进程内异步io, 多个进程充分利用多核
代码参见

```
Every 2.0s: ss state all sport = :1234 -n

Netid  State      Recv-Q Send-Q Local Address:Port               Peer Address:Port
tcp    LISTEN     0      32         *:1234                  *:*
tcp    LISTEN     0      32         *:1234                  *:*
tcp    LISTEN     0      32         *:1234                  *:*
tcp    LISTEN     0      32         *:1234                  *:*
tcp    LISTEN     0      32         *:1234                  *:*
tcp    TIME-WAIT  0      0       10.10.213.219:1234             10.10.213.219:33949
tcp    TIME-WAIT  0      0       10.10.213.219:1234             10.10.213.219:33952
```

1个管理进程，4个工作进程

事件驱动，异步非阻塞IO

高性能IO之Reactor模式

react-php

> The main loop is the only thing that is going to be blocking. It has to check for the events, so it could react for the incoming data. When we execute for example sleep(10), the loop will not be executed during these 10 seconds. And everything that loop is going to do during this time will be delayed by these seconds. Never block the loop, for situations when you need to wait, you should use timers.
>
> Also, everything that could take longer than about one millisecond should be reconsidered. When you cannot avoid using blocking functions the common recommendation is to fork this process, so you can continue running the event loop without any delays.
>
> Loop主循环应该是程序中唯一会发生阻塞的地方。它会检测发生的事件，并触发相应回调处理。如果在某个回调中发生如sleep的阻塞行为，主Loop也会阻塞，本来预期在此期间执行的事件都会相应延迟。
>
> 永远不要阻塞主Loop！如果确实需要阻塞，考虑使用定时器。
> 执行超过1ms的事件都需要注意。如果无法避免阻塞，那推荐fork一个子进程在子进程中执行，从而不会造成主Loop延迟。

基于react-php的聊天室

1. 启动server
   php react_event_chatserver.php
2. 启动多个client(telnet终端)

react-php 6000+star

react+多进程/多线程/协程

Nginx：多进程Reactor
Nginx+Lua：多进程Reactor+协程
Golang：单线程Reactor+多线程协程
Swoole：多线程Reactor+多进程Worker
swoole 10000+star

php同步方式写异步

https://github.com/tencent-php/tsf

基础知识

tcp连接

TCP连接的状态详解以及故障排查
tcp 三次握手和四次断连深入分析：连接状态和socket API的关系
TCP Keepalive基本知识

IO模型演进

PHP并发IO编程之路
Socket 深度探索 4 PHP
Event-Driven PHP with ReactPHP: Event Loop And Timers

其他

监听超时

stream_socket_accept的第二个参数指定监听超时时间，timeout
Override the default socket accept timeout. Time should be given in seconds.
未指定时使用pool默认超时60s

读写超时

stream_set_timeout指定已建立连接的读写超时，可精确到毫秒
Sets the timeout value on stream, expressed in the sum of seconds and microseconds.

When the stream times out, the ‘timed_out’ key of the array returned by stream_get_meta_data() is set to TRUE, although no error/warning is generated.

每次fread, fwrite返回后用stream_get_meta_data获取steam信息，如time_out的值为true表示超时，可关闭连接。
fread本身不支持超时设置，只可以设置阻塞非阻塞。但fread的FILE指针是通过socket转过来，而socket是可以设置接收发送超时的，所以使用fread接收socket数据时也就具有超时的属性。

未指定时使用pool默认超时60s