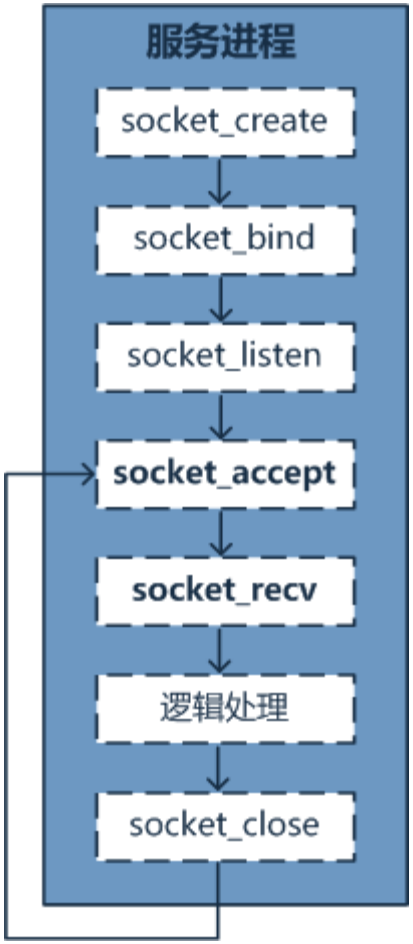
Server模型及其PHP实现

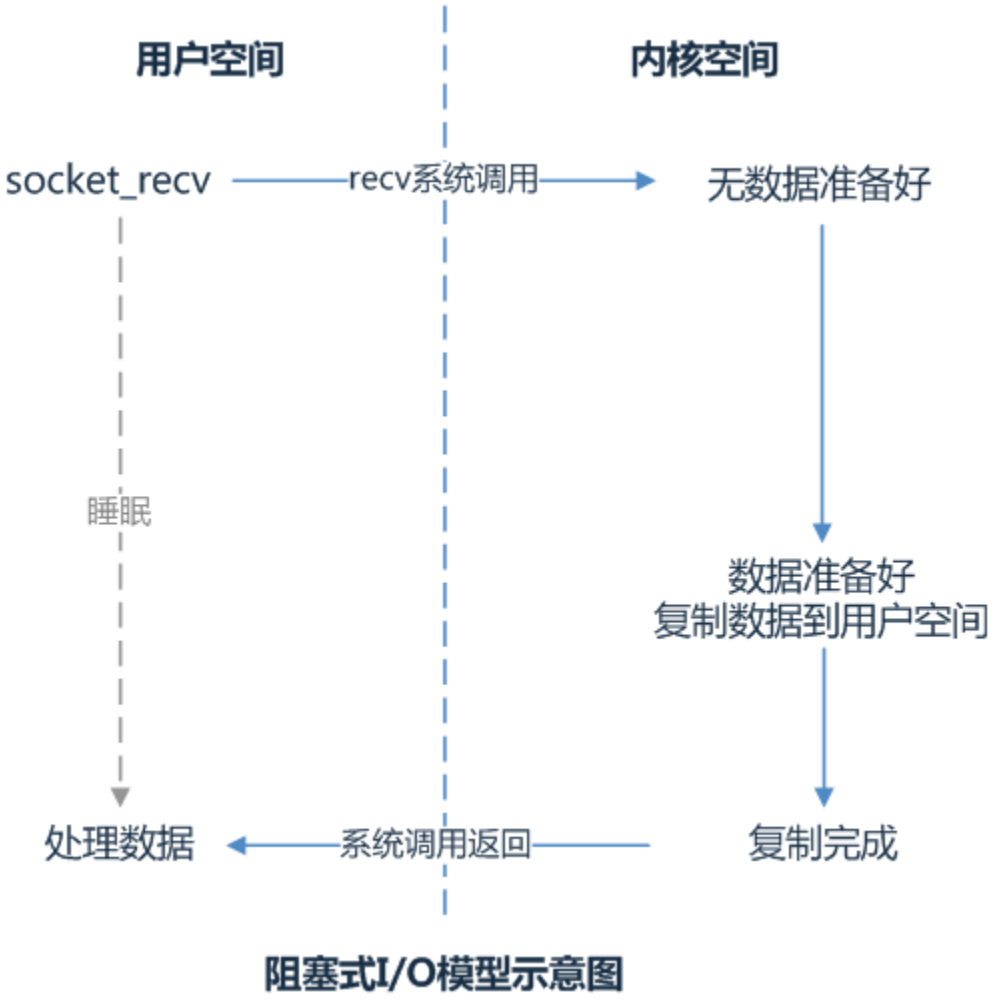
Workerman.net

提纲

1. 介绍一些常用的server架构模型
2. 常用server模型的特点及适用场景
3. server架构需要考虑的技术点
4. 经典server模型PHP实现
5. PHP实现server需要注意的点
6. 一个面向服务的站点架构

单进程阻塞accept模型



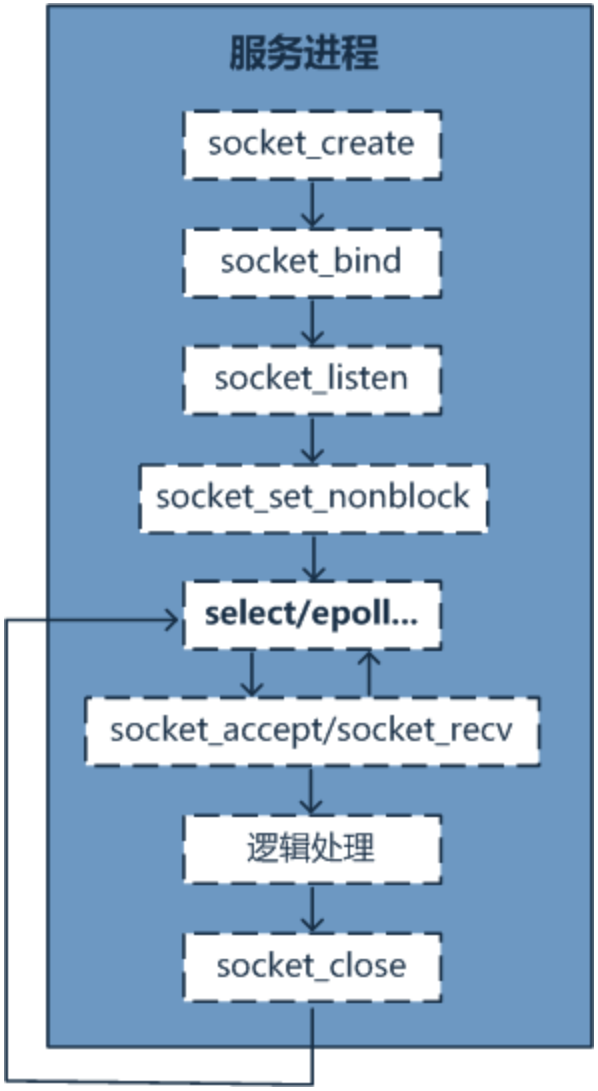


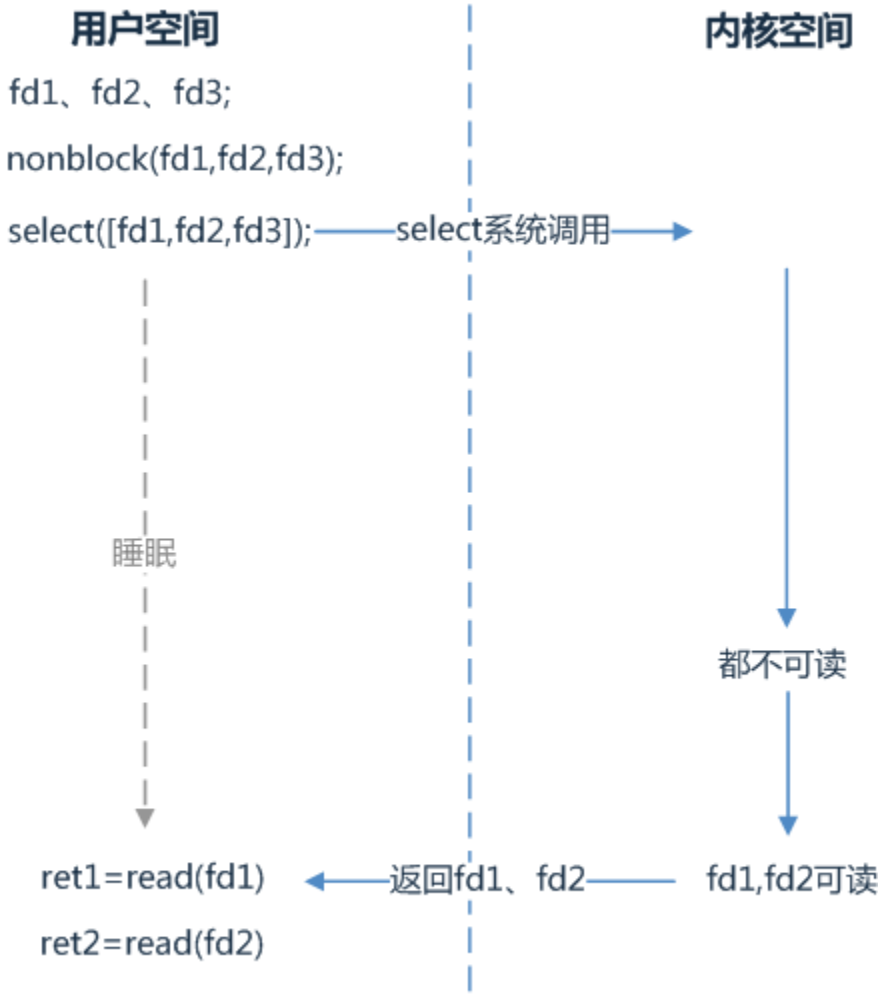
阻塞式I/O模型示意图

单进程阻塞accept模型

1. 单进程
2. 阻塞在accept recv系统调用
3. 每次只能处理一个连接
4. 小心拒绝服务攻击

单进程非阻塞I/O复用模型



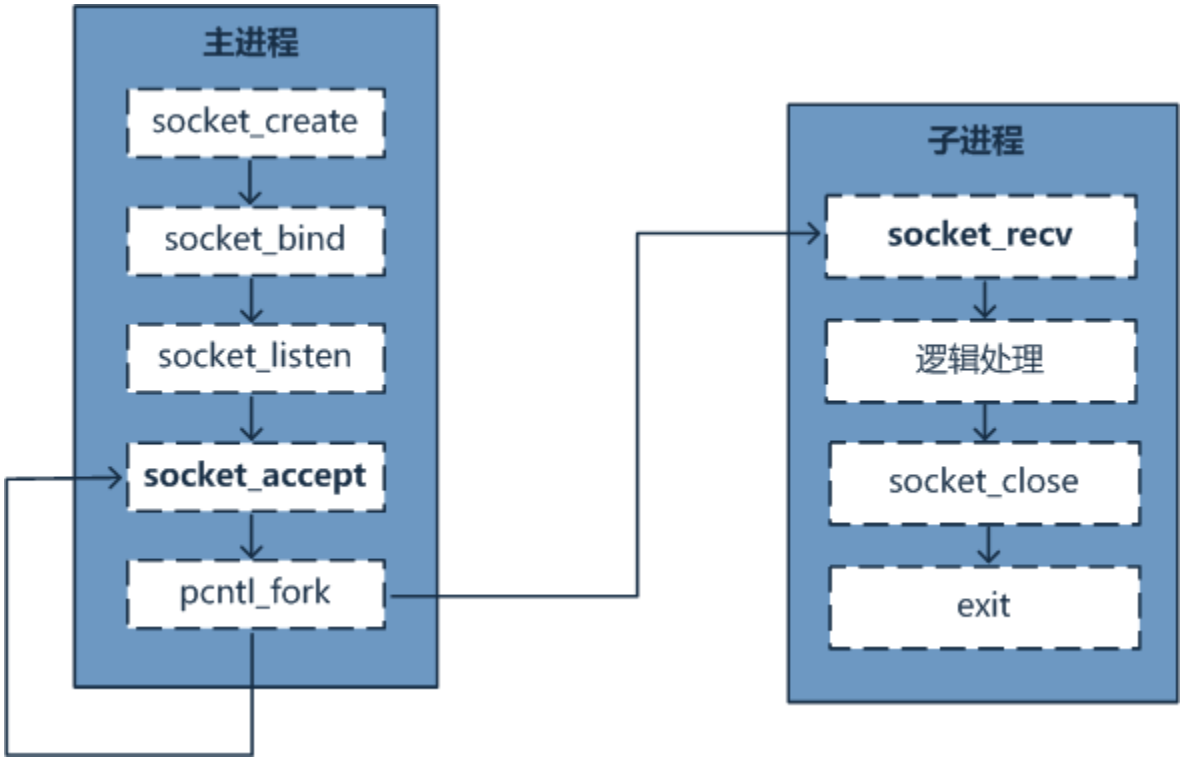


非阻塞I/O复用模型示意图

单进程非阻塞I/O复用模型

1. 单进程
2. 阻塞在select/poll/epoll系统调用
3. 可以处理多个连接
4. 不支持高并发

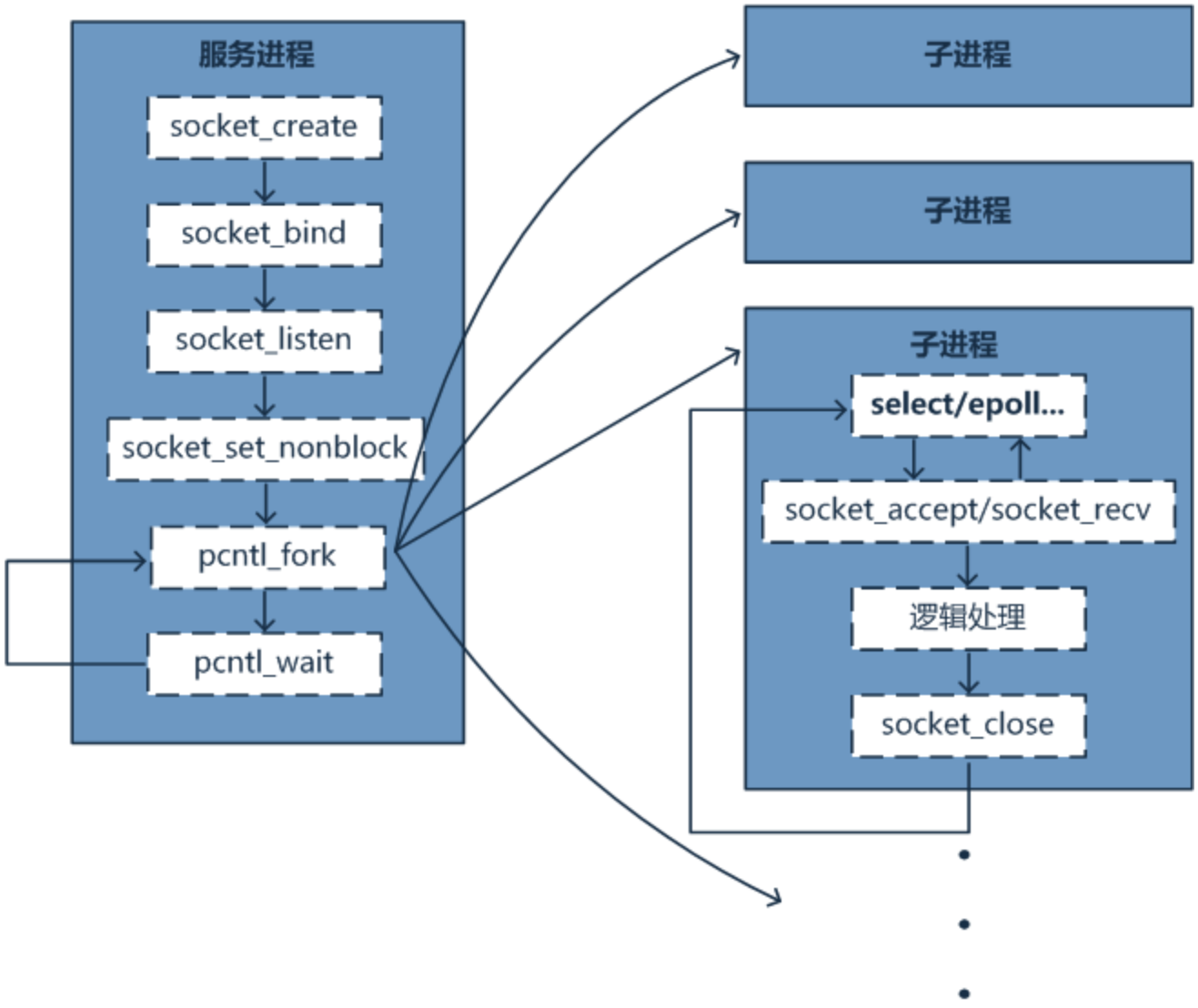
每个连接一个子进程


主进程
socket_create
socket_bind
socket_listen
socket_accept
pcntl_fork
子进程
socket_recv
逻辑处理
socket_close
exit

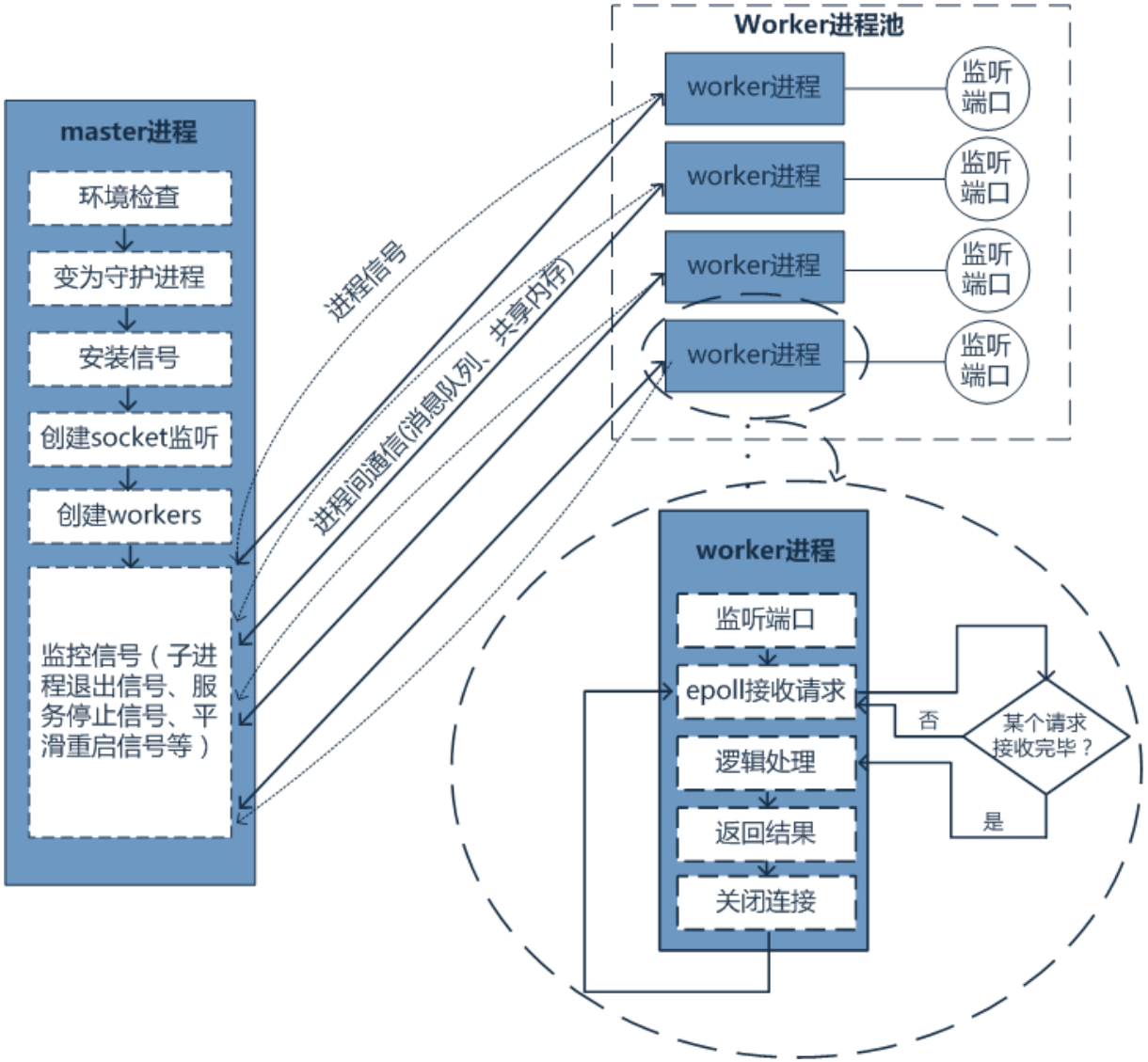
每个连接一个子进程

1. 多进程
2. 主进程阻塞在accept系统调用
3. 子进程阻塞在recv系统调用
4. 进程数量需要监控
5. 有创建进程的开销

预先派生子进程模型


服务进程
socket_create
socket_bind
socket_listen
socket_set_nonblock
pcntl_fork
pcntl_wait
子进程
子进程
子进程
select/epoll...
socket_accept/socket_recv
逻辑处理
socket_close

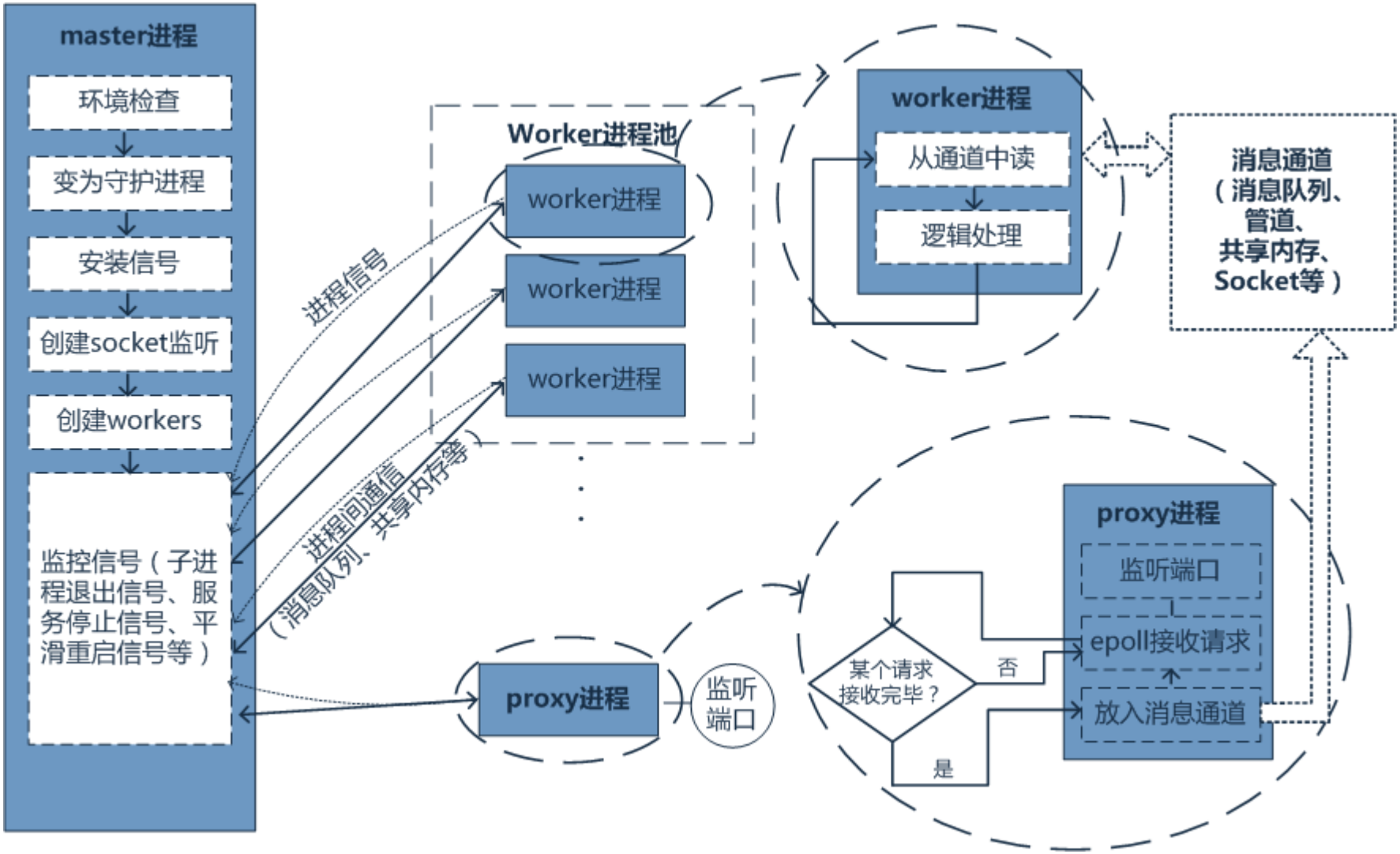
多进程master-worker模型



多进程master-worker模型

1. 多进程
2. master进程负责监控子进程
3. worker进程负责网络I/O收发及逻辑处理
4. 无创建进程开销
5. 支持高并发
6. 有惊群效应

多进程proxy-worker模型



多进程proxy-worker模型

1. proxy进程只负责网络I/O
2. worker进程只负责逻辑处理
3. master负责监控proxy、worker进程
4. proxy worker间有进程通信开销
5. proxy worker间通信方式决定应用场景
6. worker处理能力不影响proxy网络I/O
7. 单proxy无惊群效应

Server架构需要考虑的技术点

1. 超时处理
2. 服务平滑重启
3. 文件监控及自动更新
4. 进程异常监控
5. 多协议支持
6. 以指定用户运行worker
7. worker PHP语法检查
8. 锁机制解决惊群问题
9. 批量accept，提高性能

PHP超时控制

如何控制脚本超时

① socket超时设置

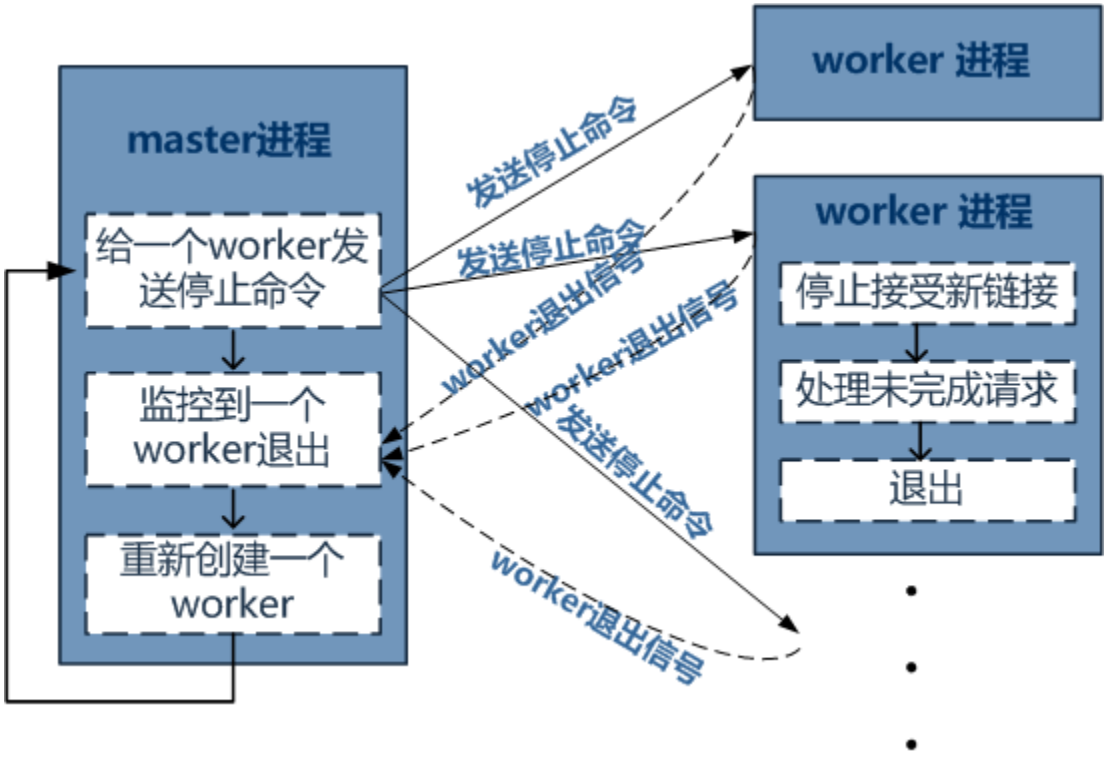
② select/poll/epoll 超时

③ pcntl_alarm pcntl_signal

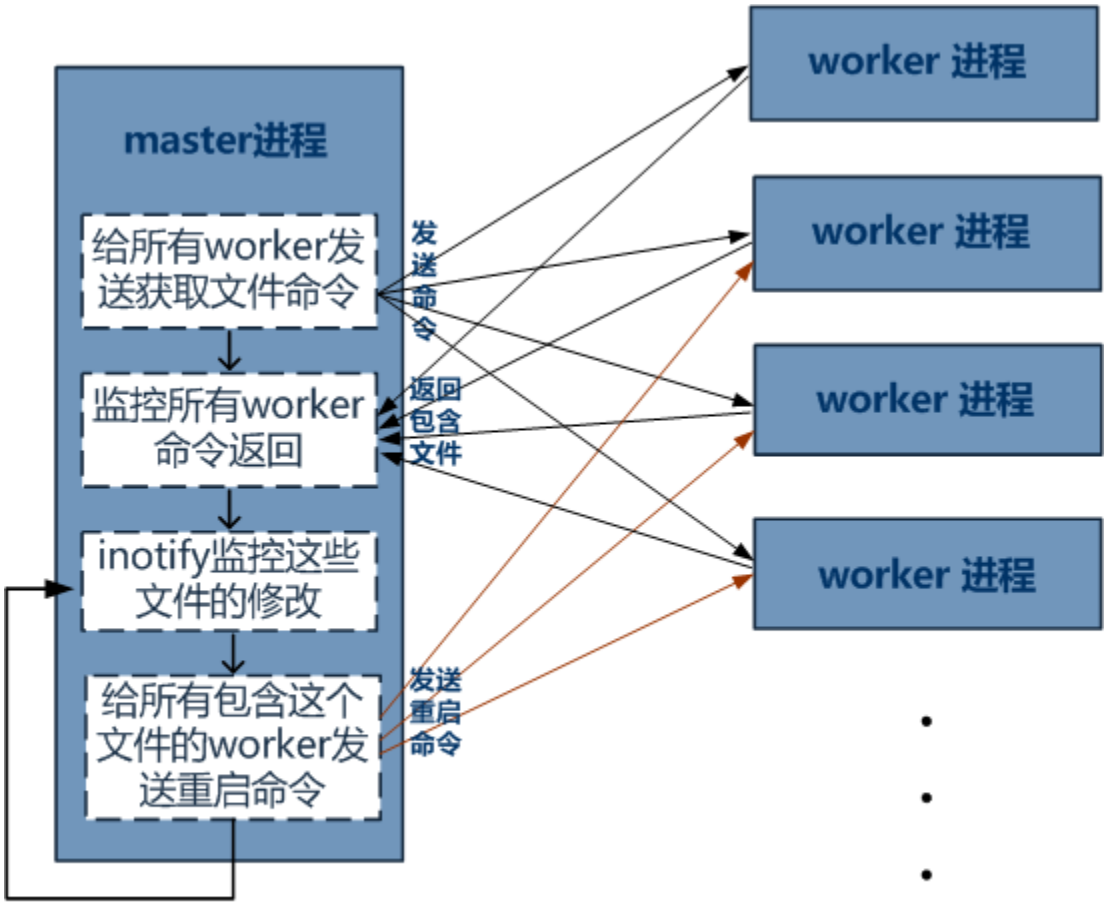
④ register_tick_function();

⑤ set_time_limit();

服务平滑重启


master进程
给一个worker发送停止命令
监控到一个worker退出
重新创建一个worker
worker 进程
worker 进程
停止接受新链接
处理未完成请求
退出
发送停止命令
发送停止命令
worker退出信号
worker退出信号
发送停止命令
worker退出信号

文件监控及自动更新



Inotify 监控文件的更新

进程监控之监控子进程退出

监控子进程退出的方法

① 监听SIGCHLD信号,调用pcntl_wait*获取退出进程pid及状态

② I/O复用监听进程间的管道可读,调用pcntl_wait*

③ 直接调用pcntl_wait*获取退出进程pid及退出状态

注意监控进程退出状态

进程监控之监控子进程内存泄露

如何控制内存泄露

① 把业务逻辑封装在一个函数或者方法中

② 注意全局变量和类的静态成员的膨胀

③ 每个worker接受固定次数的请求后回收该进程

④ 监控worker内存使用情况,内存增长一定大小后回收该进程

⑤ ini_set(memory，xxM)

PHP会将不用的内存交还系统

以指定用户运行worker

① posix_setgid();

② posix_setuid();

PHP语法检查

① php_check_syntax() (PHP 5 <= 5.0.4)

② exec(‘php -l file.php’ , $out, $ret)

③ 子进程include file.php;exit;父进程判断退出状态

异步server需要考虑的技术点

1. 队列的持久化
2. worker处理失败重试
3. 高可用

异步server队列持久化

① 直接使用磁盘-永久存储 速度慢

② 使用系统消息队列-服务器重启后丢失

③ 内存磁盘/dev/shm-定时同步到磁盘

④ mmap内存映射-永久存储速度快

异步server失败重试

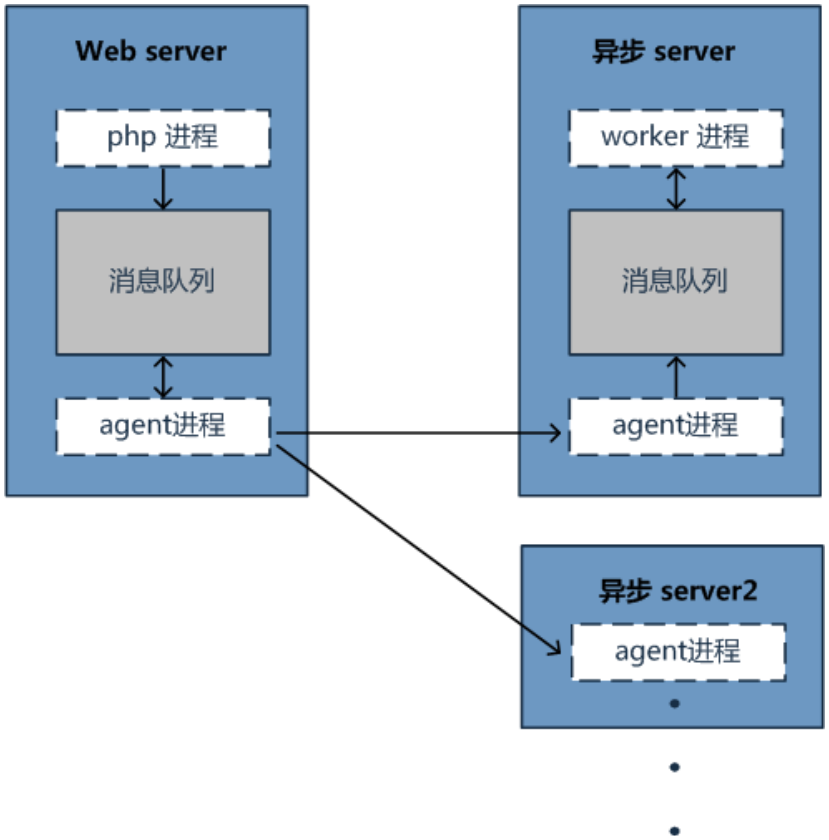
① 某个请求完全失败

② 某个请求部分失败

③ 复杂逻辑分步处理，成功做标记

④ 重试时机 重试时间递增

异步server高可用

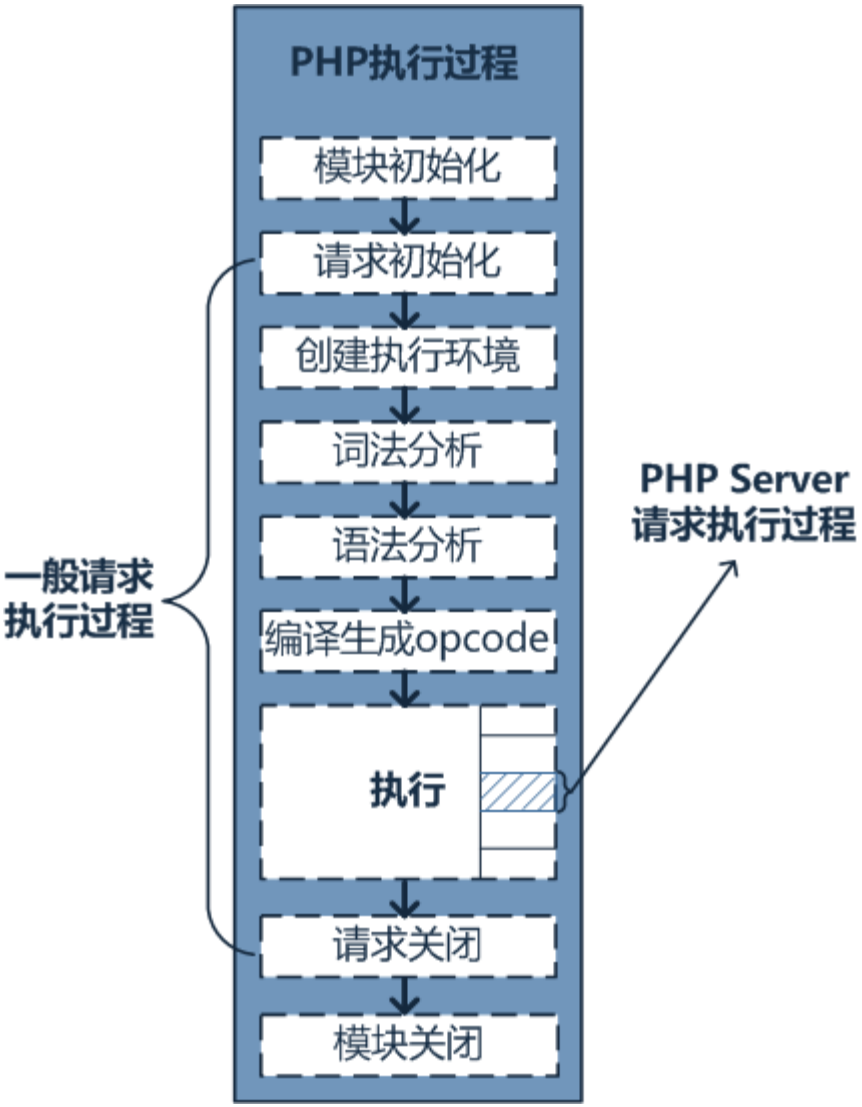


① client也有缓冲队列

② server不可用时保证前端不出错

③ server集群，单台挂了可以继续服务

从PHP运行机制分析PHP Server



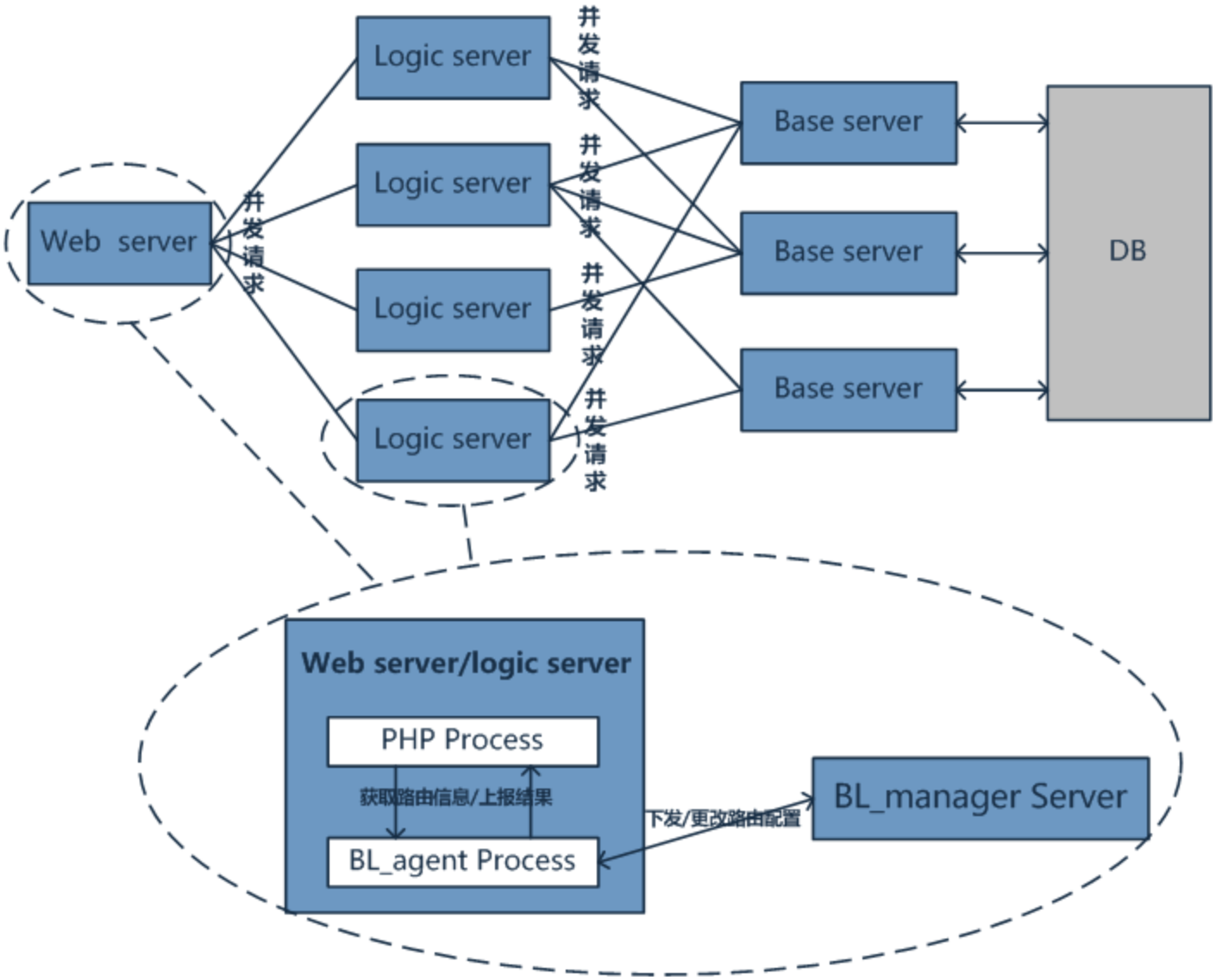
Linux5种I/O模型

1. 阻塞I/O （PHP支持）
2. 非阻塞I/O （PHP支持）
3. I/O复用 （PHP libevent,libev,libuv）
4. 信号驱动I/O
5. 异步I/O (PHP eio_*)

PHP支持的进程间通信机制

1. 共享内存 shm_* shmop_*
2. 消息队列 msg_*
3. 信号 pcntl_signal pcntl_wait*
4. 信号量 sem_*
5. 管道 PIPE管道/全双工管道/FIFO命名管道
6. socket 命名/无名UNIX Socket

一个面向服务的架构模型



压测数据

空跑：3w+/s
50字节数据包直接回包：2.7w/s
1.5k数据包直接回包:2w/s
1.5k数据写磁盘后回包:1.7w/s

谢谢