# S1

## 进程?

操作系统当中一个**进程**，就是一个正在运行的程序

Eg php aa.php 相当于创建了一个进程，这个进程会在系统中贮存，申请他自己的内存空间，系统资源，并且运行相应的程序

### 进程核心

进程核心1 内存：

进程核心2 上下文环境：

### 多进程

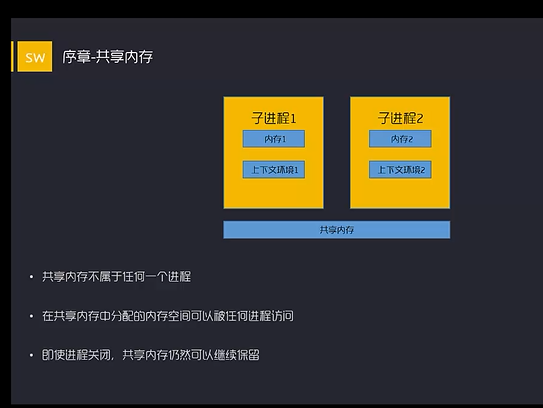
\*\* 复制 **主进程的内存和上下文环境**，子进程中变量修改不会影响父进程中的变量



### 进程间通讯?

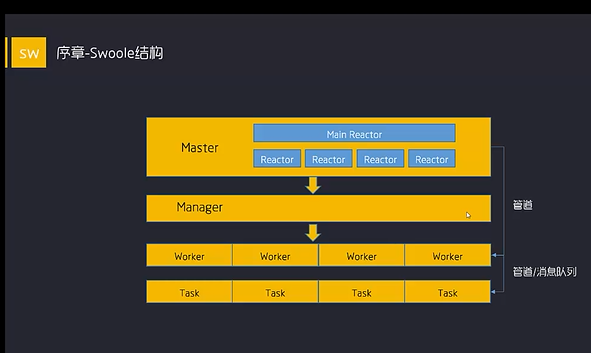
–共享内存（其中一种通讯方式）

不属于任何一个进程，任何进程都可以读取、修改



通过几个命令看一下 系统当中共享内存的分片

Ipcs –m



Web 依赖nginx php-fpm

Fpm 多进程php解析器

当新请求过来以后，fpm会创建一个进程去处理请求，系统开销是创建和销毁进程，导致响应效率不是特别高

## Swoole架构（可以分为三层）

Swlooe采用了和fpm完全不同的架构

### （第一层）Master进程

用于处理swoole核心的事件驱动，拥有若干reactor线程

每个reactor线程用于，每个线程拥有运行着一个epol函数的实例，对事件监听，在reactor中实现

A来自客户端的链接

B本地通讯用的管道

C异步操作用的文件描述符

链接、管道、文件

### （第二层）Manager 管理进程—仅仅只做进程管理和分配

创建管理下级进程（work、taskwork）

### （第三层）工作进程a&b

a Work 进程：主逻辑进程，用于处理来自客户端的请求

b Taskwork进程：异步工作进程，主要处理 耗时较长的同步任务

swoole流程：

a 在master进程当中，当reactror接收到客户端的数据的时候，这些数据会通过管道发送给work进程，由work进程进行处理

b 当work进程需要投递任务到task进程时，也是通过管道来实现数据的投递

**新客户端连接来到时，首先会main reactor线程接收到，然后将这个连接的读写操作的监听注册到reactor线程当中，并通知对应的work进程处理对应的onconnect回调（接收到连接的回调）**

**当客户端发送数据之后，reactor线程会收到这些数据，并通过管道发送给work进程进行处理，work进程如果需要投递任务，work进程会把数据通过管道发送给taskwork进程**

**Taskwork进程处理完之后，返回给work进程，work再通知reactor线程发送数据交回给客户端**

## 进程间通信 – 基于管道

可以配置swoole配置参数来设置，使通信走系统的消息队列

OnTask回调

Finsh函数

Task-常见问题

Task传递数据的大小

数据小于8k 直接通过管道传输

数据大于8k 超过了swoole的buff空间 写入临时文件传递（传文件名）

Task传递对象

可以通过序列化传递一个对象的拷贝；不是对象的引用

Task中对对象的改变，不会反映到Worker进程中

数据库的链接、网络链接对象不可传递

Task的onFinish回调

Task的onFinish回调会发回调用task方法的**worker进程**（耗时较长的同步任务执行完毕后）投递者 – 数据从worker到taskworker再回来，路径是可循的

## Work、Task进程

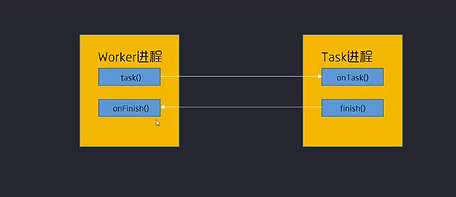
Task进程是swoole当中独立于work进程的工作进程，用于处理一些耗时较长的逻辑。这些逻辑在task进程中处理时，并不影响work进程去处理来自客户端的请求，由此，大大提高了swoole扩展处理并发的能力

如下图，work进程当中，通过调用task()方法发送数据通知到taskwork进程

Taskwork进程会在ontask回调中接收到这些数据进行处理，处理完成之后，调用finsh()函数或者直接return返回消息给work进程

Work进程在onfinsh函数中收到这些消息 并进行处理

Unixsock管道 通讯



====================================================================

新增Test类，有index属性

onReceive方法中new var\_dump并task给taskwork进程中

onTask中修改index属性值为2 打印 data 打印 $this->index

onFinish中修打印$this->test

==============================================================

**Task进程有哪些实际的用处、通过一个模拟的mysql连接池进行示范**

创建一个空server，设置指定的回调函数

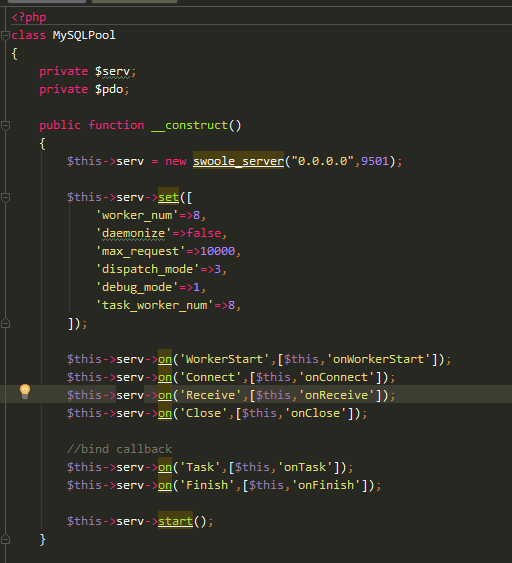
在onWorkerStart函数中创建了一个PDO连接

在onReceive的时候收到来自客户端的请求

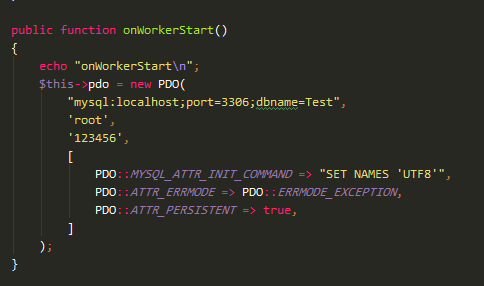
并将请求中对应的sql数据传递给Task，在TaskWorker中进行sql处理

处理完成后，返回数据给worker进程，由worker进程通知给客户端

这样就实现了一个异步的mysql操作



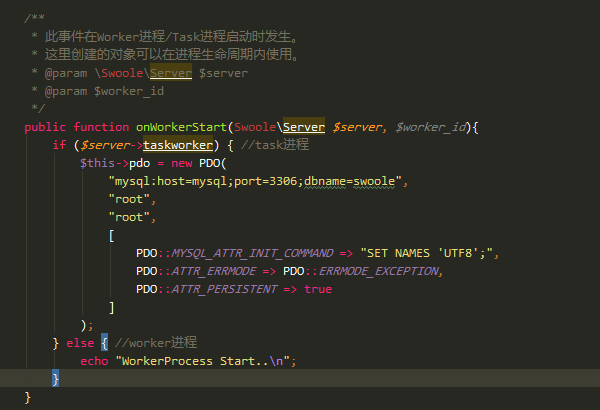
在onWorkStart函数中创建pdo连接



在onReceive的时候，收到来自客户端的请求

并将请求中对应的sql的数据传递给Task进程，在task进程中进行mysql的处理，返回数据给work进程，由work进程通知给客户端，实现异步mysql操作

\*onWorkerStart 并不区分是Worker进程还是TaskerWorker进程,可以用swoole对象的属性$server->taskworker去区分,eg



## Timer定时器

用于取代crontab等类似定时器，实现一个精度更高的毫秒级定时器

在版本1.7时代 swoole的timer进程还是一个线程，线程当中进行了一个循环，循环在一定的间隔进行唤醒，来检查每个定时器是否可以调用

后来发现 a这种方式创建的定时器，数量不能太多，太多了每次遍历的时间会变长，可能导致前面定时器还没执行完，下次定时器已经又响应了；b 线程存在隐患，可能由于某些原因挂掉，导致整个定时器失效

因此，在新版本的swoole中，timer已经使用了epoll的timeout时间进行派递

一个定时器注册一个epoll实例的timeout回调，当这个ep在指定的毫秒内没有处理到指定的事件的时候，就会中断，并且回调一个指定的函数，这个函数会去检查存储在内存当中的所有定时器，他们是否可以被运行

在新的swoole扩展当中，timer进程是基于Reacter线程的epoll实例来运行的。在Taskworker进程当中是没有Reachor线程的，不能调用swoole的异步IO的这些函数，因此在Taskworker进程中使用的是系统定时器也就是timefb相关的函数（自行google）

Timer的实现是基于epoll的timeout机制，

为了提高timer的检索效率，使用堆存放timer（最小堆）存放的索引是每个定时器距离下一次响应剩余的时间，时间越小，离堆顶越近。每次遍历的时候，就从堆顶往下检索。

### 使用：

1 swoole\_timer\_tick() 多次

2 swoole\_timer\_after() 一次性

3

Ep 的回调timeout机制 堆 剩余时间

堆算法

使用

永久定时器 tick

1. After

### 传递参数

a 可以通过tick方法的第三个参数传参

b 可以使用use闭包

### 传递对象

在同一个进程中，

# 2 EventIO & Process

## 2.1【序章】异步非阻塞IO，进程相关知识

### 2.1.1进程(补充内容)

a 复制一个进程的时候，会复制内存和上下文环境

b \*\***补充**\*\* 子进程会复制在父进程中创建的IO句柄（fd描述符）

IO句柄 -- 比如说 打开的一个文件、创建的一个socket连接

在父进程中打开一个文件，得到一个fp描述符，在子进程中同样会拥有一个句柄，并且可以对同一个文件进行读写操作，多进程环境中同时对一个文件写操作，会使该文件内容乱掉，这个时候我们需要 文件锁、对文件写操作进行互斥 process中具体细讲

### 2.1.2进程间通讯方式 管道&消息队列

1 **管道**（特殊的描述符），由父进程创建，一组两个管，单向，一个用来读，一个用来写。

子进程也拥有2个描述符，父进程写，子进程读，或者子进程写，父进程读

在子进程被创建之前就创建管道 保证两个进程都拥有管道 fork函数调用前创建管道，保证父进程子进程都有该管道的实例

子进程 父进程都可以主动关闭管道，当一端关闭后，另一端的读取操作为直接返回0

2 **消息队列** 独立于进程之外的空间

通过指定一个key值创建一个消息队列

只要拥有相同的key，就可以访问到同一个消息队列。保证了多个进程之间链接到同一个消息队列

消息队列中传递的数据有大小限制，数据块，不是流式的数据 更像UDP方式

默认65535??

一个消息队列被创建后，会一直保留，直到被主动关闭

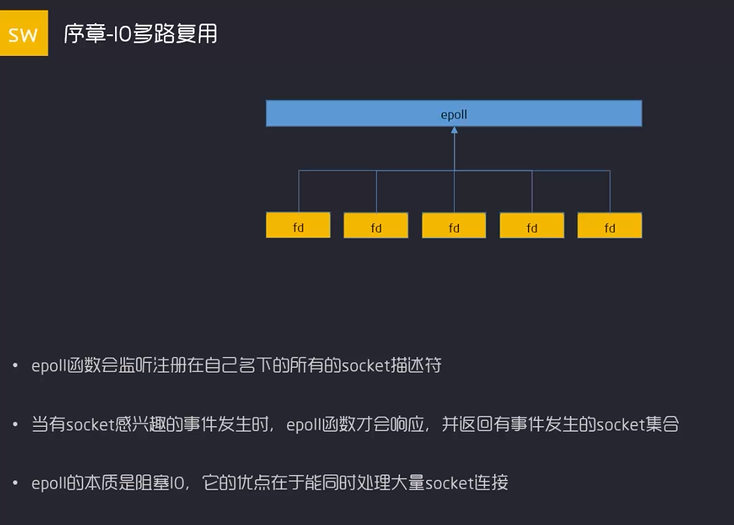
好处：一个服务进程，正在处理来自客户端的请求的时候，突然进程挂了，可能当前正在处理的请求或者还没来得及处理的请求就丢了，后边服务再起来就找不到这些请求了

消息队列可以做到 进程没有了，消息仍然在队列当中，把进程再拉起来的时候，仍然可以读到消息队列中的信息，可以继续处理

### IO多路复用 c10k c1000k 优点 在同一个进程内同时处理很多个描述符

开进程开线程 一个一个处理 做不到~

网络编程服务的本质



## 2.2 EventLoop（事件循环） 异步IO的使用 常见问题的解决方案和实例

### 2.2.1 事件循环EventLoop

Swoole提供了一个epoll上层的封装，并且提供了一个线程，当通过swoole event一系列的函数 去发起创建事件循环的时候，swoole会在底层启动一个reacter线程，这个线程中会实际运行一个epoll的实例，需要注册描述符到这个epoll实例当中，并为他设置 read 和 write的监听

官网文档：<https://wiki.swoole.com/wiki/page/242.html>

6个函数

swoole\_event\_add 将一个socket描述符加入到swoolereactor事件监听当中

swoole\_event\_Set 从新设定 需要add过

swoole\_event\_del 移除某个描述符

swoole\_event\_exit 退出事件循环 client中有效

swoole\_event\_write

~~wait （php5.4以上已经没用了wait）~~



实例 **命令行聊天室**

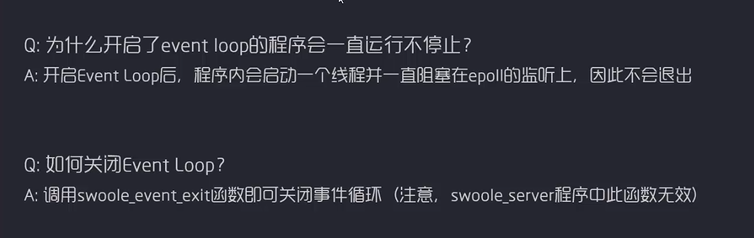
主要应用点

异步读取来自服务器的数据

异步读取来之中断的输入

手动退出聊天室

常见问题



## 2.3 Process 如何使用Process对象、Process通信的使用及实例

## 2.4 Q&A