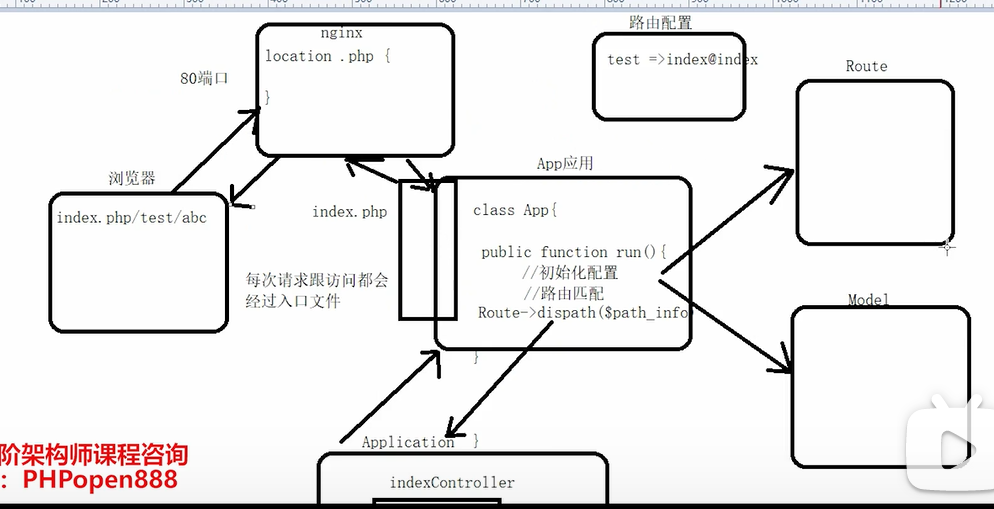
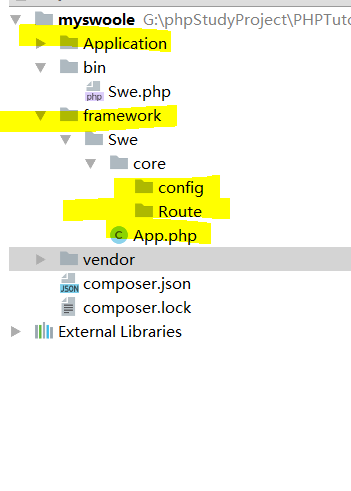


## 传统的php框架架构



## 2.composer快速构建框架

### 2.1创建框架基本文件



Application 应用目录

Framework框架文件

App.php 执行应用

Vendor 扩展

### 2.2 编写composer.json

{  
 **"name"**: **"my/swoole"**,  
 **"description"**: **"myswoole Framework."**,  
 **"keywords"**: [  
 **"framework"**,  
 **"myswoole"** ],  
 **"autoload"**: {  
 **"psr-4"**: {  
 **"App\\"**: **"application/"**,  
 **"Swe\\"**: **"framework/Swe"** }  
 },  
 **"require"**: {  
 **"php"**: **"^7.1.3"** }  
}

### 2.3生成composer文件

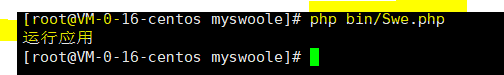
Composer undate

### 2.4引入自动加载并运行框架

在bin下新建Swe.php

**require** *dirname*(***\_\_DIR\_\_***).**'/vendor/autoload.php'**;  
  
(**new** \Swe\App())->run();

运行框架



## 3.传统框架与基于swoole框架的区别

## 4.通过注解方式实现路路由组件

### 4.1递归遍历文件

//文件遍历  
$dir=$this->tree(APP\_PATH,'Controller');

//遍历文件  
public function tree($dir,$filter){  
 $dirs=glob($dir.'/\*');  
 $dirFiles=[];  
 foreach($dirs as $dir){  
 if(is\_dir($dir)){  
 $res=$this->tree($dir,$filter);  
 if(is\_array($res)){  
 foreach ($res as $v){  
 $dirFiles[]=$v;  
 }  
 }  
 }else{  
 //判断是否是控制器  
 if(stristr($dir,$filter)){  
 $dirFiles[]=$dir;  
 }  
  
 }  
 }  
 return $dirFiles;  
}

### 4.2加载文件

*//将文件目录实例化*$filename=*explode*('/',$file);  
*//获取类名*$className=*explode*('.',*end*($filename))[0];  
*//加载文件*$file=*file\_get\_contents*($file,false,null,0,500);

### 4.3匹配命名空间

*//匹配文件命名空间 \s表示空格 i同时匹配大小写字母 .匹配除换行符\n之外的任意字符  
preg\_match*('/namespace\s(.\*)/i',$file,$namespace);  
*//$namespace的值 "App\Controller;*if(isset($namespace[1])){  
 *//消除 命名空间中的空格 引号 分号 App\ Controller;* $namespace=*str\_replace*(['"',';',' '],'',$namespace[1]);

### 4.4实例化类并注册路由

*//实例化类*$obj=$className;  
$reflect=new \ReflectionClass($obj);  
*//类的注解*$classComment=$reflect->getDocComment();  
foreach ($reflect->getMethods() as $method){  
 $methodComment=$method->getDocComment();*//方法注解  
 //路由信息收集 权限认证等其他信息* $annotation= new RequestMapping($classComment,$methodComment,$reflect,$method);  
  
 *//注册路由* (new RequestMappingParser())->parse($annotation);  
}

### 4.5 匹配分发路由

$res=Route::*diapatchRoute*($method,$pathInfo);

public static function diapatchRoute($method,$pathInfo){  
 switch ($method){  
 case 'GET':  
 foreach(self::*$route*[$method] as $v){  
 *//判断路径是否在注册路由上* if($pathInfo==$v['routeInfo']){  
 $handle=*explode*('@',$v['handle']);  
 $class=$handle[0];  
 $method=$handle[1];  
 return (new $class)->$method();*//执行控制器方法*  
 }  
  
 }  
 break;  
 case 'POST':  
 break;  
 }  
}

## 5.IOC容器

容器就是一个巨大工厂，用于存放和管理对象的生命周期，并且能够解决程序的依赖关系，实现解耦。

### 5.1IOC容器构建

## 6.config封装

## Swoole生命周期

## 7.热加载

### 7.1什么是热加载

不重启服务器修改配置文件

### 7.2swoole如何实现热加载

在swoole我们可以向主进程中发送不同的信号，主进程根据受到的不同信号类型做出相应的处理。

如：

当USR1信号被发送给Master进程后Master进程会将同样的信号通过Mannager进程转发worker进程，收到此信号的worker进程会在处理完正在执行的逻辑后，释放进程内存，关闭自己，然后由manager进程重启一个新的worker进程。新的worker进程会占用新的内存空间，重新加载文件。

因此只有在worker进程中的事件进行加载，并通过include方式才会生效

在onworker()中加载文件

#### 定时器判断文件（Md5\_file）是否变化 reload重启

##### 单独写一个reload类用于判断文件的变换

<?php  
  
  
namespace Swe\core;  
  
  
class Reload  
{  
 public $watch;*//监控文件夹* public $md5Flage;*//上次文件MD5的值* private static *$intance*;  
  
 public static function getInstance(){  
 if(isset(self::*$instance*)){  
 return self::*$instance*;  
 }  
 return new self();  
 }  
 public function reload(){  
 *//当前MD5值和上一次是否有区别* $md5=$this->getMd5();  
 if($md5!=$this->md5Flage){  
 $this->md5Flage=$md5;  
 return true;  
 }  
 }  
  
 public function getMd5(){  
 $md5="";  
 foreach ($this->watch as $dir){  
 $md5.= self::*md5File*($dir);  
 }  
 return *md5*($md5);  
 }  
 public static function md5File($dir){  
 *//遍历文件中的所有文件，得到所有文件的MD5散列值* if(!*is\_dir*($dir)){  
 return "";  
 }  
 $md5File=array();  
 $d=*dir*($dir);  
 while(false!==($entry=$d->read())){  
 if($entry!=='.'&&$entry!=='..'){  
 if(*is\_dir*($dir.'/'.$entry)){  
 $md5File[]=self::*md5File*($dir.'/'.$entry);  
 }elseif (*substr*($entry,-4)==='.php'){  
 $md5File[]=*md5\_file*($dir.'/'.$entry);  
 }  
 $md5File[]=$entry;  
 }  
 }  
 $d->close();  
 return *md5*(*implode*('',$md5File));  
 }  
}

##### Onworker()reload Worker/Task 进程



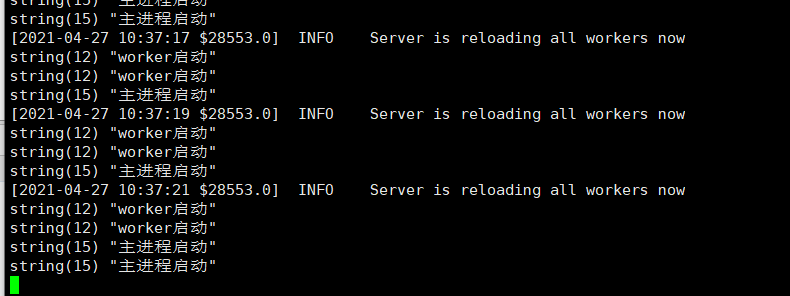
Swoole\Server->reload(bool $only\_reload\_taskworker = false): bool

安全地重启所有 Worker/Task 进程。

在onworkerstrat中reload所有定义的woker都会重新启动一次

因此我们在onstart（）回调中进行reload重启所有 Worker/Task 进程。

**启动后在主进程（master）的主线程回调onstart函数**



## 8.分布式框架RPC通讯机制

### RPC远程调用

#### 1.什么是RPC

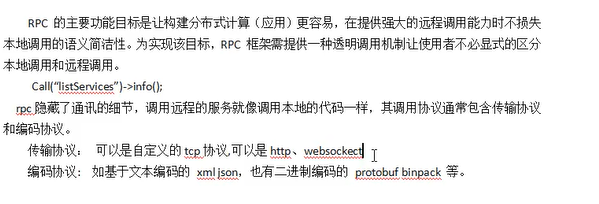
RPC是一种通过网络从远程计算机上请求服务而不需要了解底层网络

技术的协议。

比如：有两台服务器。一个应用放在A服务器上，想要调用B服务器上应用的方法或函数，由于不在一个内存空间，不能直接调用，就需要通过网络来表带调用的寓意和传达调用的数据，这就是RPC

#### 2.为什么需要RPC

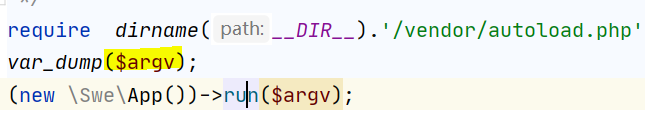
RPC的主要功能目标是让构建分布式计算(应用)更容易，在提供强大的远程调用能力时不损失本地调用的语义简洁性，为实现该目标，RPC框架需提供一种透明机制让使用者不必显示的区分本地调用和远程调用

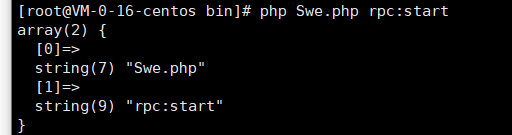


#### 3.RPC服务端封装

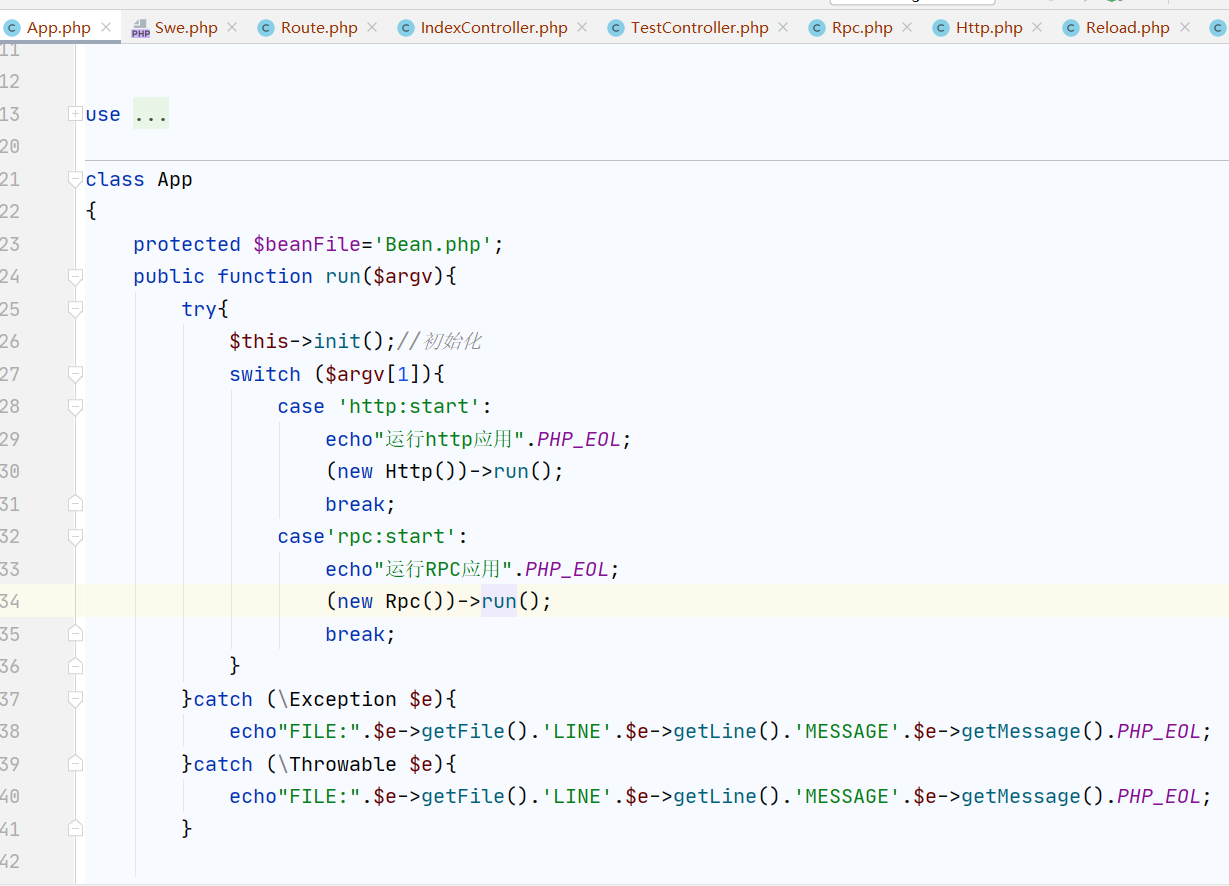
##### 3.1获取启动的命令

不同命令运行tcp http



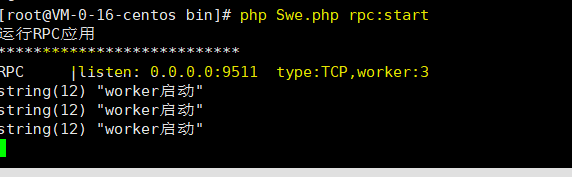


##### 3.2根据不同的命令启动不用服务



##### 3.3创建Rpc服务端

<?php  
  
  
namespace Swe\core\Rpc;  
  
  
use Swe\core\Bean\BeanFactory;  
use Swe\core\Http;  
use Swe\core\Reload;  
use Swe\core\Route\Route;  
  
class Rpc extends Http  
{  
 public function run(){  
 BeanFactory::*get*('config')->load();  
 $config=BeanFactory::*get*('config')->get('rpc');  
*// var\_dump($config);  
 //* $this->server=new \Swoole\Server($config['host'],$config['port']);  
 $this->server->set($config['setting']);  
 *//在swoole worker进程中触发* $this->server->on('start',[$this,'start']);  
 $this->server->on('workerStart',[$this,'workerStart']);  
 $this->server->on('receive',[$this,'receive']);  
 $this->server->start();  
 }  
 public function receive($server, $fd, $reactorId,$data){  
 *var\_dump*($server,$data);  
 }  
  
  
 public function start(){  
 $reload=Reload::*getInstance*();  
 $reload->watch=[*CONFIG\_PATH*,*APP\_PATH*,*ROOT\_PATH*];  
 $reload->md5Flage=$reload->getMd5();  
*// var\_dump($reload->getMd5());  
 //定时监控 Swoole\Timer::tick* \Swoole\Timer::*tick*(2000,function($timer\_id)use($reload){  
*// var\_dump('主进程启动');  
 //判断是否需要重启* if($reload->reload()){  
 $this->server->reload();*//重新启动worker进程* }  
 });  
 $config=BeanFactory::*get*('config')->get('rpc');  
 echo"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*".*PHP\_EOL*;  
 echo *sprintf*("RPC |listen: %s:%d type:TCP,worker:%d",  
 $config['host'],$config['port'],$config['setting']['worker\_num']).*PHP\_EOL*;  
 }  
}



##### 3.4同时启动http、Rpc

在swoole中有多端口监听

Server 可以监听多个端口，每个端口都可以设置不同的协议处理方式，例如 80 端口处理 HTTP 协议，9507 端口处理 TCP 协议。SSL/TLS 传输加密也可以只对特定的端口启用。

//返回port对象

$port1 = $server->listen("127.0.0.1", 9501, SWOOLE\_SOCK\_TCP);

$port2 = $server->listen("127.0.0.1", 9502, SWOOLE\_SOCK\_UDP);

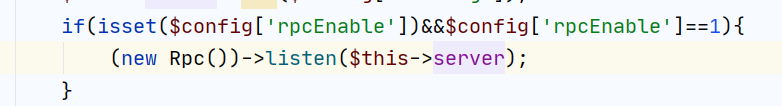
$port3 = $server->listen("127.0.0.1", 9503, SWOOLE\_SOCK\_TCP | SWOOLE\_SSL);

[addListener()](https://wiki.swoole.com/#/server/methods?id=addlistener)

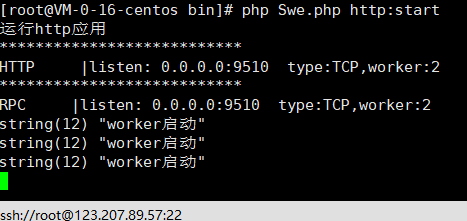
增加监听的端口。业务代码中可以通过调用 [**Server->getClientInfo**](https://wiki.swoole.com/#/server/methods?id=getclientinfo) 来获取某个连接来自于哪个端口。

Swoole\Server->addListener(string $host, int $port, int $sockType = SWOOLE\_SOCK\_TCP): bool|Swoole\Server\Port

运行 http时追加对rpc的监听

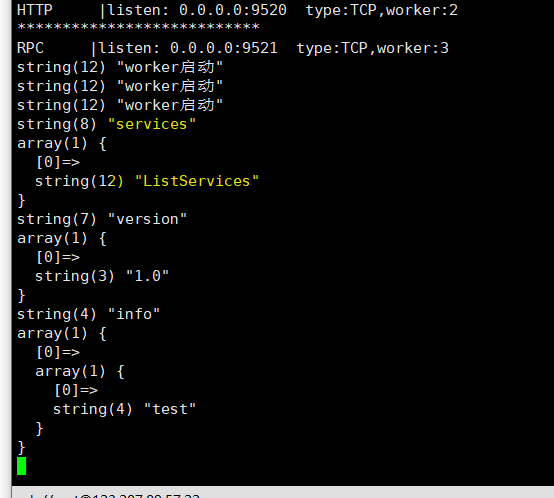


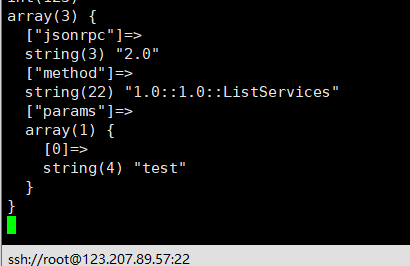




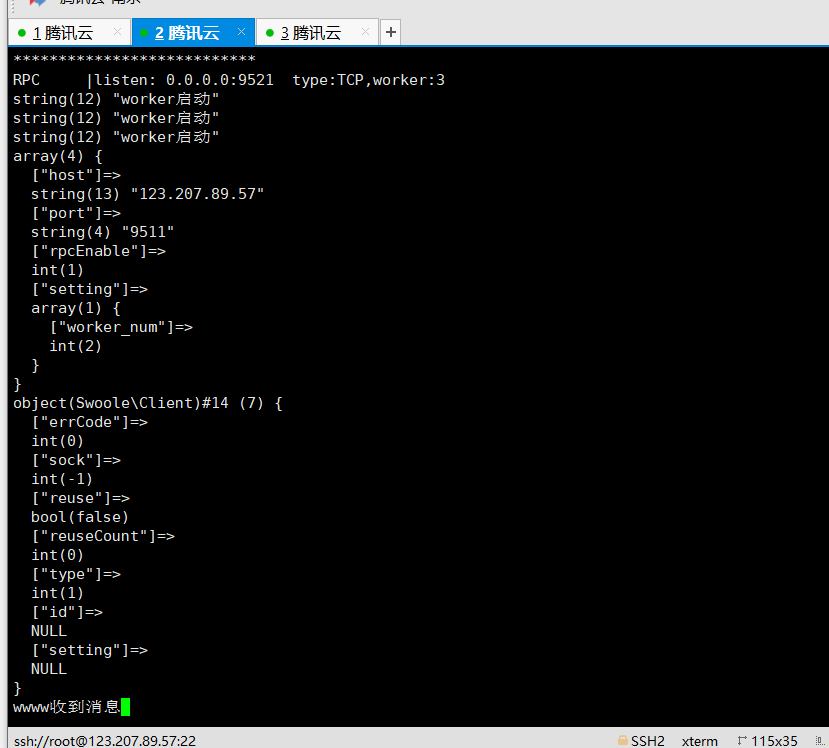
#### 4.RPC客户端封装



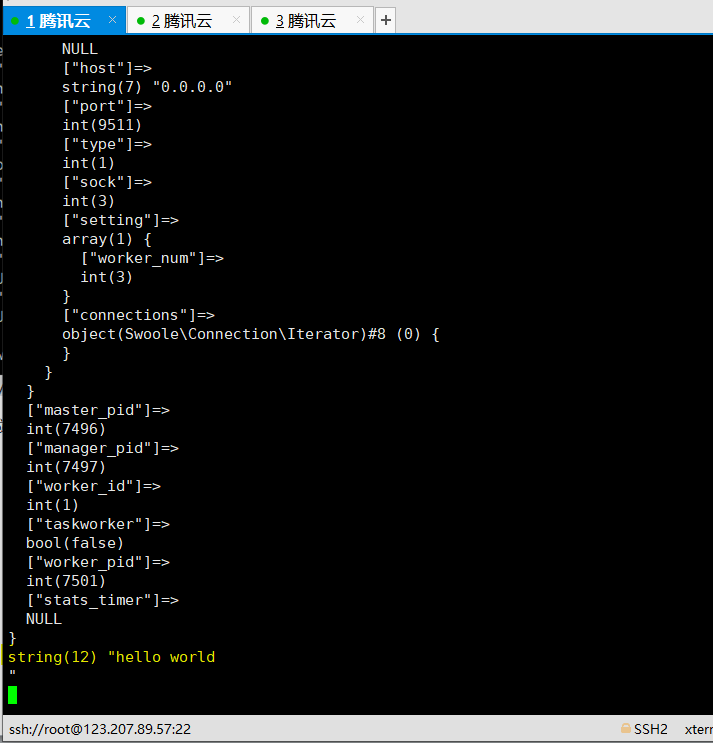




客户端



服务端



## 9.服务注册中心

### 9.1什么是服务注册中心

### 熔断

### 降级