Devlink
第二届PHP全球开发者大会
2016

# Swoole在车轮互联的应用与实践

@hantianfeng Rango-韩天峰 / 车轮互联

# 关于我

- 车轮互联总架构师
- **PHP官方扩展开发组成员**
- 微博：**@hantianfeng**
- **Github: https://github.com/matyhtf**

# 分享内容

一. **Swoole在四层架构服务化治理（SOA）方面的应用**

二. **基于Swoole开发公共组件与平台服务**

三. **新技术的尝试与实践（TSF、Http2.0）**

01

# Swoole在四层架构服务化治理（SOA）方面的应用

# 为什么要进行服务化治理（SOA）

# 基于数据库表、Redis实现服务

1. 存储层未隔离，数据不可迁移调整，耦合性极高
2. 扩展性、可维护性极差

# PHP代码(函数/类)实现服务

1. 客户端必须include/require一个文件
2. 代码修改必须通知所有业务方升级
3. 可能存在不同版本的兼容问题
4. 无法跨语言，不支持C++、Java等其他语言程序

# HTTP+JSON的Web API

1. 解决了耦合问题，可以提供良好的服务隔离
2. 优点：目前最通用的服务治理方案
3. 缺点：Http不支持并发，长连接支持差，不支持订阅与消息主动推送

# 基于Swoole实现Service方案

1. 基于swoole提供的包头+包体自动协议处理，Server/Client两端无需写任何底层代码
2. 支持单连接并发，客户端只需要与服务器建立一条连接
3. 支持php-fpm中使用TCP长连接，需要依赖swoole扩展
4. 自带Task进程池功能，可直接将慢速请求异步执行
5. 支持跨语言调用，C++、Java等其他语言程序也可以方便使用
6. Server/Client两端可以实现异步

```php
<?php
$serv = new swoole_server("127.0.0.1", 9501, SWOOLE_BASE);

$serv->set(array(
                'open_length_check'     => true,
                'dispatch_mode'         => 1,
//              'worker_num'            => 4,
                'package_length_type'   => 'N',
                'package_length_offset' => 0,       //第N个字节是包长度的值
                'package_body_offset'   => 4,       //第几个字节开始计算长度
                'package_max_length'    => 2000000,  //协议最大长度
));
```

```php
<?php
$client = new swoole_client(SWOOLE_SOCK_TCP);

$client->set(array(
    'open_length_check'     => true,
    'package_length_type'   => 'N',
    'package_length_offset' => 0,       //第N个字节是包长度的值
    'package_body_offset'   => 4,       //第几个字节开始计算长度
    'package_max_length'    => 2000000,  //协议最大长度
));
```

```
struct
{
    uint32_t length;
    uint32_t type;
    uint32_t uid;
    uint32_t serid;
    char body[0];
}
```

- length: 包体的长度
- type: 包体的打包格式，=1使用PHP串化格式，=2使用JSON格式，其他格式暂未支持
- uid: 用户自定义的ID，保留字段
- serid: Request/Response 串号

# 为什么不用Thrift、ProtoBuf

1. 优点：解包/打包性能好，IDL，自动生成多语言调用代码，对静态语言友好
2. 缺点：服务提供者需要编写维护IDL文件，门槛较高，不方便抓包调试。

## Service客户端 [编辑]

```
$service = new Service('chelun'); //创建一个车轮的Service客户端        (注：CI框架请用
```

底层库会根据当前PHP的环境，自动获取对应的Service服务器集群机器IP，并发起网络请求。

### 设置调用端环境信息 [编辑]

如表示当前的APP是哪个，APPKey等相关信息

```
$service->putEnv('app', 'chelun');
```

### 串行调用远程方法 [编辑]

```
$res = $service->call('User\Info::unlock', '18958653669', 1);
$result = $res->getResult();  //如果返回NULL，表示网络调用失败了，请检查$res->code

$res = $service->call('User\Info::unlock', '18958653669', 1);
$result = $res->getResult();

$res = $service->call('User\Info::unlock', '18958653669', 1);
$result = $res->getResult();
```

## 并行调用远程方法 [编辑]

```php
$res1 = $service->call('User\Info::unlock', '18958653669', 1);
$res2 = $service->call('User\Info::unlock', '18958653669', 1);
$res3 = $service->call('User\Info::unlock', '18958653669', 1);

//0.5表示500毫秒超时，$n表示成功返回的请求个数。如果少于发起的请求数，证明有个别请求
$n = $service->wait(0.5);

$result1 = $res1->getResult();
$result2 = $res2->getResult();
$result3 = $res3->getResult();
```

```php
<?php
namespace CheLun\User;

include __DIR__ . '/_init.php';

use Swoole;
use User\UserIdentityCategory;
use User\UserIdentity;

class Info {

	/**
	 * 锁定帐号
	 * @param $username
	 * @param int $seconds
	 * @return array
	 */
	static function lock($username, $seconds = 86400) {
		if (\FrequencyLimit::addCount('login:' . $username, $seconds, \user_model::ERROR_PASSWORD_LIMIT)) {
			return ['code' => 1];
		} else {
			return ['code' => 500, 'message' => '操作Redis失败，请稍后重试'];
		}
	}
```

```
[root@s0084-gz service]# php server.php start -d
[root@s0084-gz service]# ps aux|grep Pay
root     46806  0.0  0.0 610180  9648 ?        Ssl  13:58   0:00 PayServer: master -host=127.0.0.1 -port=8808
root     46807  0.0  0.0 381540  9152 ?        S    13:58   0:00 PayServer: manager
root     46812  0.0  0.0 385460  9296 ?        S    13:58   0:00 PayServer: worker
root     46813  0.0  0.0 385460  9296 ?        S    13:58   0:00 PayServer: worker
root     46814  0.0  0.0 385460  9296 ?        S    13:58   0:00 PayServer: worker
root     46815  0.0  0.0 385460  9296 ?        S    13:58   0:00 PayServer: worker
root     46820  0.0  0.0 103248   856 pts/30   S+   13:58   0:00 grep Pay
[root@s0084-gz service]# php server.php
================================================================================
Usage: php server.php start|stop|reload
================================================================================
           -d, --daemon      启用守护进程模式
    -h, --host [<value>]     指定监听地址
    -p, --port [<value>]     指定监听端口
                --help       显示帮助界面
            -b, --base       使用BASE模式启动
  -w, --worker [<value>]     设置Worker进程的数量
  -r, --thread [<value>]     设置Reactor线程的数量
  -t, --tasker [<value>]     设置Task进程的数量
```

# 环境推断

1. Server支持 Windows（仅用于开发）、Linux
2. Client支持stream、sockets、swoole 3种网络客户端


服务发现
故障转移
集群管理
服务治理（SOA）
监控报警
负载均衡
调用统计

# 配置中心

1. 配置文件为JSON格式，可以拉取也可以推送
2. 每台服务器节点安装一个NodeAgent程序，实现推送
3. 客户端读取JSON配置文件即可，不存在时从远端拉取
4. 可以根据集群key得到机器列表
5. 为什么不用PHP Serialize格式存储配置？

# 为什么不用ZooKeeper

1. 需要维护ZooKeeper集群，存在额运维外成本
2. 读取配置存在网络IO，消耗较大。读取本地JSON文件单进程可达100万次，时间为微妙级别
3. 程序需要连接到ZooKeeper并维持心跳，存在额外开发成本

   Nginx、TwemProxy等软件不方便修改增加ZooKeeper
4. 紧急情况下，业务开发人员无法临时修改本地配置解决问题，必须要了解ZooKeeper的相关知识

# 服务发现 & 负载均衡

1. 服务器程序 *onStart* 时调用 *curl http://config_center/api/online* 注册到集群，并设置为在线
2. 终止运行脚本 *php server.php stop* 前调用 curl http://config_center/api/offline 从集群中摘除
3. 配置中心收到节点变更时会主动推送新的机器列表到调用端
4. 基于客户端实现的权重+随机，选择不同的集群节点
5. 连接被拒绝时自动切换到节点，解决单个节点重启时出错

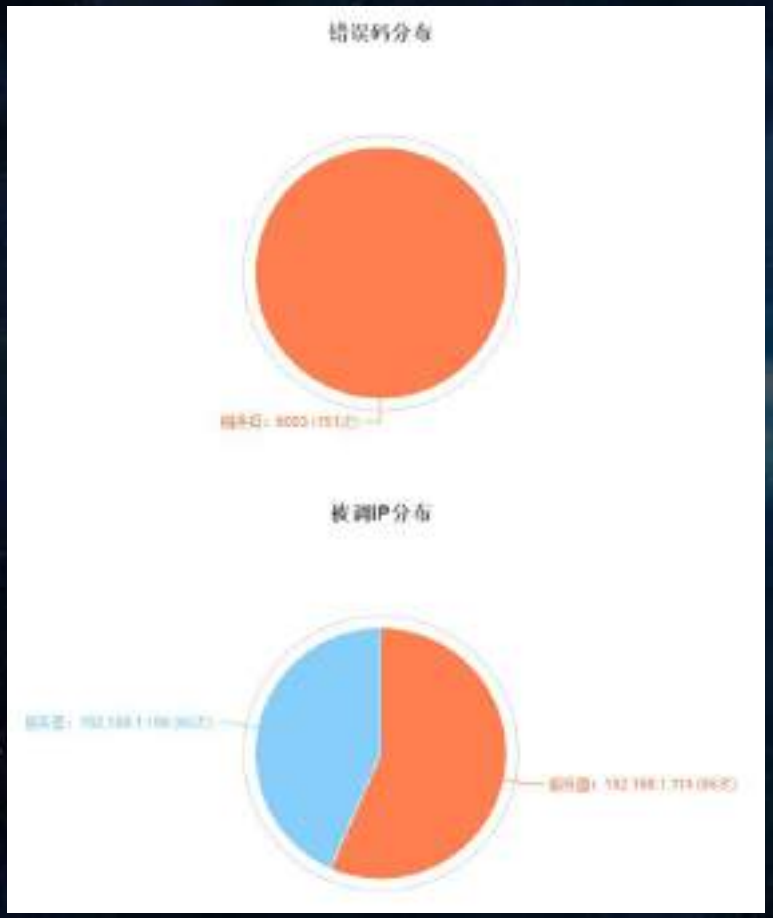
# KeepAlived守护进程

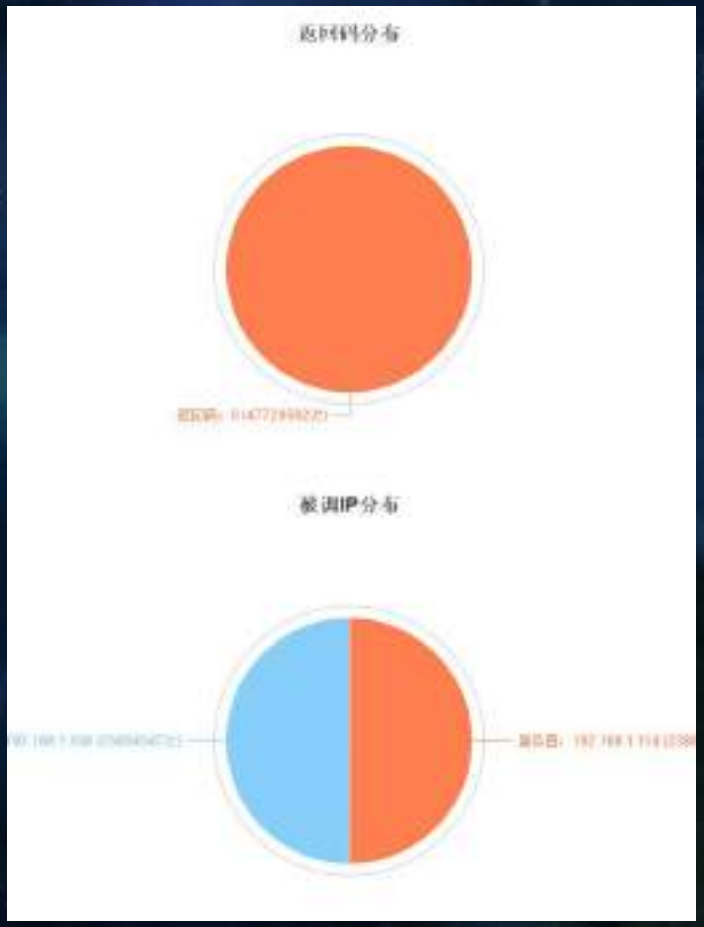
1. 每秒调用一次Service接口侦测集群每个节点是否可用
2. 发现节点无法访问，自动将此节点从Service集群中摘除
3. 发现节点重新可用时，自动将此节点加入Service集群
4. 配置有变更时，自动推送新配置到调用端

# 统计与监控报警

| 接口名称 | 时间 | 调用次数 | 成功次数 | 失败次数 | 成功率 | 响应最大值 | 响应最小值 | 平均响应时间 | 失败平均时间 |
|---|---|---|---|---|---|---|---|---|---|
| Service->ChaLun->User->Token: check | 00:00 ~ 23:59 | 47,720,632 | 47,720,632 | 0 | 100% | 321ms | 0ms | 1.37ms | 0ms |
| Service->ChaLun->User->Info: isblocked | 00:00 ~ 23:59 | 20,217,901 | 20,217,758 | 151 | 100% | 500ms | 0ms | 1.27ms | 500ms |
| Service->ChaLun->Forum->User: getMost | 00:00 ~ 23:59 | 19,125,811 | 19,125,771 | 40 | 100% | 501ms | 0ms | 0.56ms | 500.35ms |
| Service->ChaLun->User->Info: gets | 00:00 ~ 23:59 | 1,257,104 | 1,257,080 | 24 | 100% | 2178ms | 0ms | 70.5ms | 500.04ms |
| Service->KJZ->Crs->City: getCityById | 00:00 ~ 23:59 | 1,003,280 | 1,003,280 | 0 | 100% | 1014ms | 0ms | 1.14ms | 0ms |

# 统计与监控报警



# 统计与监控报警

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Service->CheLun->User->Token::check | 18:45 ~ 18:50 | 195,447 | 195,447 | 0 | 100% | 202ms | 0ms | 1ms | 0ms |
| Service->CheLun->User->Token::check | 18:50 ~ 18:55 | 195,846 | 195,846 | 0 | 100% | 129ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 18:55 ~ 19:00 | 192,205 | 192,205 | 0 | 100% | 243ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:00 ~ 19:05 | 193,641 | 193,641 | 0 | 100% | 205ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:05 ~ 19:10 | 191,228 | 191,228 | 0 | 100% | 205ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:10 ~ 19:15 | 195,286 | 195,286 | 0 | 100% | 152ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:15 ~ 19:20 | 193,705 | 193,705 | 0 | 100% | 214ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:20 ~ 19:25 | 193,876 | 193,876 | 0 | 100% | 420ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:25 ~ 19:30 | 195,251 | 195,251 | 0 | 100% | 206ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:30 ~ 19:35 | 196,855 | 196,855 | 0 | 100% | 204ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:35 ~ 19:40 | 198,460 | 198,460 | 0 | 100% | 208ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:40 ~ 19:45 | 207,626 | 207,626 | 0 | 100% | 209ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:45 ~ 19:50 | 202,181 | 202,181 | 0 | 100% | 237ms | 1ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:50 ~ 19:55 | 200,920 | 200,920 | 0 | 100% | 212ms | 0ms | 2ms | 0ms |
| Service->CheLun->User->Token::check | 19:55 ~ 20:00 | 200,106 | 200,106 | 0 | 100% | 242ms | 0ms | 2ms | 0ms |

# 统计与监控报警


模块名称
Service前端
报警策略
开启
关闭
报警间隔时间(分钟)
30
成功率阀值(0-100)
99.00
调用量波动阀值(0-100)
300.00
负责人
韩天峰 [hantianfeng]
备份负责人
王欢 [wanghuan]
徐志强 [xuzhiqiang]
项东东 [xiangdongdong]
石光启 [shiguangqi]


中国电信
15:17
模调报警
紧急告警:2016-04-13 14:40-14:45 Service前端->Service->CheLun->Forum->Subject::getMostViewed 5分钟内调用21次，失败1次，成功率95.24%低于99.00%，服务器详情，192.168.1.108失败1次，错误码详情，8003:1次，请尽快处理。
紧急告警:2016-04-13 15:00-15:05 车轮-CGI->chelun->CGI->manage->son_manager 5分钟内调用5次，失败4次，成功率20.00%低于95.00%，服务器详情，0.0.0.0失败4次，错误码详情，4444:4次，请尽快处理。

# 基于发布系统自动部署

1. 发布代码后执行*php server.php reload*
2. Service程序会重启工作进程加载最新的代码

你这么牛X，为啥非要用PHP？

你咋不用C++呢？

# 几个有趣的段(shi)子(shi)

1. 菜鸟问：如何去掉字符串两边的空格(trim)？老鸟：你先安装个包
2. 菜鸟问：怎么实现URL路由到类方法上？老鸟：这个很难实现
3. 菜鸟问：为什么从网上复制粘贴下来的代码不能用啊？老鸟：…

02

# 基于Swoole开发公共组件与平台服务

# NodeAgent

1. 部署到线上每台机器
2. 加密传输大文件（1G）
3. mcrypt扩展AES 128位加密
4. 收集机器节点信息
5. 发送reload信号到Server程序
6. 配置中心基于NodeAgent程序实现配置文件主动推送

# MySQL-Proxy

1. 基于mysqli+mysqlnd+swoole_mysql_query实现
2. 支持php-fpm长连接
3. 后端使用连接池可以有效减少MySQL服务器的连接数
4. 支持MySQL后端服务路由，php-fpm到MySQL-Proxy只需要建立一个连接，即可向到多台MySQL服务器发送SQL
5. 不支持事务处理

# MySQL-Proxy

1. 早期版本基于 MYSQLI_ASYNC、mysqli::reap_async_query、swoole_mysqli_get_sock
2. 存在2个问题，1) 结果较大可能会阻塞 2) 缓存区设置较小，有大量read, poll 系统调用导致sys很高
3. 新的API swoole_mysql_query 自行解析MySQL二进制协议，阻塞时自动让出，实现了真正的异步非阻塞。并且改为 64K 缓存区，大大减少了系统调用次数。

# MySQL协议(query请求)

- 3字节长度 + 1字节packet_id + 1字节cmd + n字节SQL语句
- http://blog.csdn.net/wind520/article/details/43964821

| 类型值 | 命令 | 功能 |
| --- | --- | --- |
| 0x00 | COM_SLEEP | （内部线程状态） |
| 0x01 | COM_QUIT | 关闭连接 |
| 0x02 | COM_INIT_DB | 切换数据库 |
| 0x03 | COM_QUERY | SQL查询请求 |
| 0x04 | COM_FIELD_LIST | 获取数据表字段信息 |

# MySQL协议(ResuleSet)

| 响应报文类型 | 第1个字节取值范围 |
| --- | --- |
| OK 响应报文 | 0x00 |
| Error 响应报文 | 0xFF |
| Result Set 报文 | 0x01 - 0xFA |
| Field 报文 | 0x01 - 0xFA |
| Row Data 报文 | 0x01 - 0xFA |
| EOF 报文 | 0xFE |

# 短网址服务

- **http://chelun.com/url/D2M2qX**
- **3位检验码 + 自增ID（62进制）**
- **swoole_http_server + redis 单机性能高达 3W+ QPS**
- **作为车轮互联商业广告的流量入口和出口**
- **统计设备号、UID、IP、UV、PV、地理位置等信息，为运营部门提供数据**
- **统计逻辑基于Task功能实现，不影响核心逻辑**

# 03

# 新技术的尝试与实践

# TSF协程框架应用

- 腾讯TSF框架的应用，解决爬虫程序并发问题
- 传统curl同步阻塞模式，外网通信较慢，需要启动大量进程
- 10s每次请求 x 100进程 -> 10 QPS
- TSF单进程并发1000请求 -> 1000 QPS

# HTTP2.0

- 支持并发，通过标记 stream_id 实现多条并发数据流
- 头部压缩，静态表 + 动态表 + Huffman Table
- Push Promise 支持主动推送
- 相比HTTP 1.0，仅底层通信方式变更，应用代码无需变更
- 目前支持HTTP 2.0的软件：Nginx, Swoole

# HTTP2.0+Swoole

```php
$ssl_dir = realpath('../../tests/ssl');
$serv = new swoole_http_server("0.0.0.0", 9501, SWOOLE_BASE, SWOOLE_SOCK_TCP | SWOOLE_SSL);
$serv->set([
    'ssl_cert_file' => $ssl_dir . '/ssl.crt',
    'ssl_key_file' => $ssl_dir . '/ssl.key',
    'open_http2_protocol' => true,
]);

$serv->on('Request', function(swoole_http_request $request, swoole_http_response $response) {
    $response->end("<h1>Hello Swoole!</h1>\n");
});

$serv->start();
```

# THANK YOU

- 期待2017年再见-

# Q & A