3.TARS PHP HTTP服务端与客户端开发

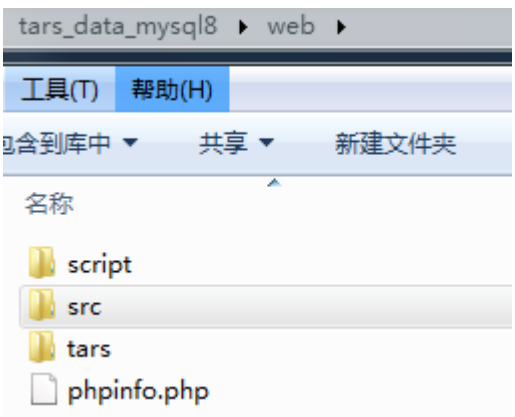
开发PHP HTTP协议服务端

我们使用标签为 `php7mysql8` 的 `tangramor/docker-tars` 镜像来进行PHP HTTP协议服务端的开发（假设你用的是Windows）：

```
docker run -d --name mysql8 \
  -e MYSQL_ROOT_PASSWORD=password -p 3306:3306 \
  -v /c/Users/tangramor/mysql8_data:/var/lib/mysql \
  mysql:8 --sql_mode="" --innodb_use_native_aio=0

docker run -d -it --name tars_mysql8 \
  --link mysql8 \
  --env DBIP=mysql8 \
  --env DBPort=3306 \
  --env DBUser=root \
  --env DBPassword=password \
  -p 3000:3000 -p 80:80 \
  -v /c/Users/tangramor/tars_mysql8_data:/data \
  tangramor/docker-tars:php7mysql8
```

这两个命令分别启动了mysql容器（8.0版）和 `tangramor/docker-tars:php7mysql8` 容器 **tars_mysql8**，并将本地的一个目录 `/c/Users/tangramor/tars_mysql8_data` 挂载为容器的 /data 目录，同时它还把 3000 和 80 端口暴露出来了。

我们进入 `/c/Users/tangramor/tars_mysql8_data/web` 目录，在其下创建对应的目录结构：`scripts`、`src` 和 `tars`，



在 `tars` 目录下创建 `tars.proto.php` 文件，对服务进行描述，我们在部署时会使用这些信息：

```
<?php
return array(
    'appName' => 'PHPTest',
    'serverName' => 'PHPHttpServer',
    'objName' => 'obj',
);
```

在 `src` 目录下创建 `component` 和 `controller` 两个目录，在 `src/component` 目录下新建文件 `Controller.php`：

```
<?php
/**
 * Created by PhpStorm.
 * User: liangchen
 * Date: 2018/5/8
 * Time: 下午2:43.
 */
namespace HttpServer\component;
use Tars\core\Request;
use Tars\core\Response;
class Controller
{
    protected $request;
    protected $response;
    public function __construct(Request $request, Response $response)
    {
        // 验证cookie、get参数、post参数、文件上传
        $this->request = $request;
        $this->response = $response;
    }
    public function getResponse()
    {
        return $this->response;
    }
    public function getRequest()
    {
        return $this->request;
    }
    public function cookie($key, $value = '', $expire = 0, $path = '/',
$domain = '', $secure = false, $httponly = false)
    {
        $this->response->cookie($key, $value, $expire, $path, $domain,
$secure, $httponly);
```

```
    }
    // 给客户端发送数据
    public function sendRaw($result)
    {
        $this->response->send($result);
    }
    public function header($key, $value)
    {
        $this->response->header($key, $value);
    }
    public function status($http_status_code)
    {
        $this->response->status($http_status_code);
    }
}
```

在 `src/controller` 目录下新建文件 `IndexController.php` （注意：此文件在官方例子上做了删减）：

```
<?php
/**
 * Created by PhpStorm.
 * User: liangchen
 * Date: 2018/5/8
 * Time: 下午2:42.
 */
namespace HttpServer\controller;
use HttpServer\component\Controller;

class IndexController extends Controller
{
    // curl "172.16.0.161:28887/Index/index" -i
    public function actionIndex()
    {
        $this->cookie('key', 1, 10000000, '/', 'www.github.com');
        $this->sendRaw('success');
    }
    // curl "172.16.0.161:28887/Index/testHeader" -i
    public function actionTestHeader()
    {
        $this->header('test', 1111);
    }
    // curl "172.16.0.161:28887/Index/testStatus" -i
    public function actionTestStatus()
    {
        $this->status(401);
    }
```

```
    // Get请求
    // curl "172.16.0.161:28887/Index/get?param=1111&c=d&d=e&e=f" -i
    public function actionGet()
    {
        $param = $this->request->data['get']['param'];
        $this->sendRaw('success:'.$param);
    }
    // Post请求
    // curl -d "user=admin&passwd=12345678"  "172.16.0.161:28887/Index/post1"
-i
    public function actionPost1()
    {
        // 对于content-type为application/x-www-form-urlencoded 的form data
        $admin = $this->request->data['post']['user'];
        $this->sendRaw('success:'.$admin);
    }
    // Post请求
    //curl -H "Content-Type:application/json" -X POST -d '{"user": "admin",
"passwd":"12345678"}' "172.16.0.161:28887/Index/post2" -i
    public function actionPost2()
    {
        // 对于对于content-type为application/json
        $json = $this->request->data['post'];
        $admin = json_decode($json, true)['user'];
        $this->sendRaw('success:'.$admin);
    }
    // Get请求
    // curl -F "image=@profile.jpeg" -F "phone=123456789"
"172.16.0.161:28887/Index/file" -i
    public function actionFile()
    {
        $fileName = $this->request->data['files']['image']['name'];
        $type = $this->request->data['files']['image']['type'];
        $tmp_name = $this->request->data['files']['image']['tmp_name'];
        $size = $this->request->data['files']['image']['size'];
        $this->sendRaw('success:'.var_export($this->request->data['files']
['image'], true));
    }
}
```

在 `src` 目录下新建 `index.php` 文件作为入口：

```
<?php
/**
 * Created by PhpStorm.
 * User: dingpanpan
 * Date: 2017/12/2
```

```
 * Time: 17:00.
 */
require_once __DIR__.'/vendor/autoload.php';
use \Tars\cmd\Command;
//php tarsCmd.php  conf restart
$config_path = $argv[1];
$pos = strpos($config_path, '--config=');
$config_path = substr($config_path, $pos + 9);
$cmd = strtolower($argv[2]);
$class = new Command($cmd, $config_path);
$class->run();
```

在 `src` 目录下新建 `composer.json` ：

```
{
    "name" : "tars-http-server-demo",
    "description": "tars http server",
    "require": {
        "phptars/tars-server": "~0.1",
        "phptars/tars-deploy": "~0.1",
        "phptars/tars2php": "~0.1",
        "phptars/tars-log": "~0.1",
        "ext-zip" : ">=0.0.1"
    },
    "autoload": {
        "psr-4": {
            "HttpServer\\" : "./"
        }
    },
    "minimum-stability": "stable",
    "scripts" : {
        "deploy" : "\\Tars\\deploy\\Deploy::run"
    },
    "repositories": {
        "tars": {
            "type": "composer",
            "url": "https://raw.githubusercontent.com/Tencent/Tars/master/php/
dist/tarsphp.json"
        }
    }
}
```

在 `src` 下新建 `services.php` ，注意 `namespaceName` 必须与上面的 `composer.json` 中配置的一致：

```php
<?php
// 以namespace的方式,在psr4的框架下对代码进行加载
return array(
 'namespaceName' => 'HttpServer\\',
 'monitorStoreConf' => [
   //'className' => Tars\monitor\cache\RedisStoreCache::class,
   //'config' => [
    // 'host' => '127.0.0.1',
    // 'port' => 6379,
    // 'password' => ':'
   //],
   'className' => Tars\monitor\cache\SwooleTableStoreCache::class,
   'config' => [
     'size' => 40960
   ]
 ]
);
```

`monitorStoreConf` 为主调上报信息的存储配置

- `className` 为主调上报信息的存储实现类的类名，默认为 `\Tars\monitor\cache\SwooleTableStoreCache::class` 使用swoole_table存储，tars-monitor中还提供了redis的存储方式，用户也可以自定义新的实现，但是必须实现 `\Tars\monitor\contract\StoreCacheInterface` 接口
- `config` 为主调上报信息的存储实现类的配置信息，在实现类初始化时作为参数传入，默认对应swoole_table的size

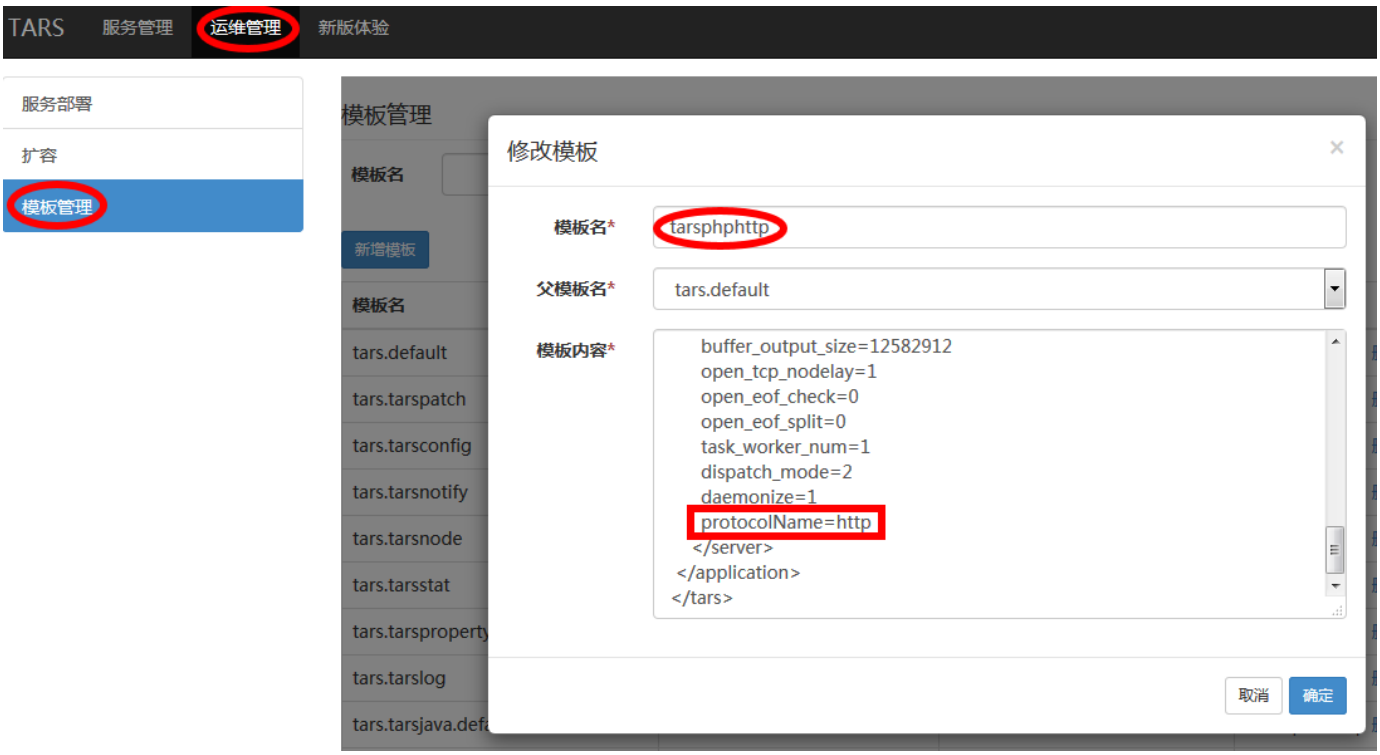
运行 `docker exec -it tars_mysql8 bash` 进入容器 **tars_mysql8**， `cd /data/web/src` 进入工作目录。

执行 `composer install` 加载依赖包；然后执行 `composer run-script deploy` 对项目进行打包，会在 `src` 目录下生成一个 .tgz 文件。

打开TARS平台网页，点击“运维管理” => “模板管理”，拷贝 `tars.tarsphp.default` 的内容，在它的 `<server>` 标签下添加 `protocolName=http` ，使用这个内容新建一个模板 `tarsphphttp` ：

```
<tars>
  <application>
    ...
    <client>
      ...
    </client>
```

```
    <server>
      ...
      protocolName=http
    </server>
  </application>
</tars>
```



将打好的包发布到TARS平台，记得选择php方式，协议选择**非TARS**协议，模版使用刚刚新建的 `tarsphphttp` 。

发布成功后，在系统里执行 `ps -ef` 会发现相关的进程。

```
[root@a07acb36a0f1 ~]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:33 pts/0    00:00:04 /usr/java/jdk-10.0.1/bin/java -jar lib/resin.jar console
root        34     1  0 08:33 pts/0    00:00:02 /usr/local/app/tars/tarsregistry/bin/tarsregistry --config=/u
root        43     1  0 08:33 pts/0    00:00:01 /usr/local/app/tars/tarsAdminRegistry/bin/tarsAdminRegistry -
root        63     1  0 08:33 pts/0    00:00:01 /usr/local/app/tars/tarsconfig/bin/tarsconfig --config=/usr/l
root        72     1  0 08:33 pts/0    00:00:00 /usr/local/app/tars/tarspatch/bin/tarspatch --config=/usr/loc
root        75     1  0 08:33 ?        00:00:04 /usr/local/app/tars/tarsnode/bin/tarsnode --locator=tars.tars
root        82     1  0 08:33 ?        00:00:00 redis-server 127.0.0.1:6379
root       220     0  0 08:33 pts/1    00:00:00 bash
root       256     1  7 08:33 pts/0    00:00:37 /usr/java/jdk-10.0.1/bin/java -Dresin.server=app-0 -Djava.uti
root       330     0  0 08:33 ?        00:00:00 httpd
apache     331   330  0 08:33 ?        00:00:00 httpd
apache     332   330  0 08:33 ?        00:00:00 httpd
apache     333   330  0 08:33 ?        00:00:00 httpd
apache     334   330  0 08:33 ?        00:00:00 httpd
apache     335   330  0 08:33 ?        00:00:00 httpd
root       364    75  0 08:33 ?        00:00:02 /usr/local/app/tars/tarsnode/data/tars.tarslog/bin/tarslog --
root       365    75  0 08:33 ?        00:00:01 /usr/local/app/tars/tarsnode/data/tars.tarsnotify/bin/tarsnot
root       366    75  0 08:33 ?        00:00:01 /usr/local/app/tars/tarsnode/data/tars.tarsproperty/bin/tarsp
root       367    75  0 08:33 ?        00:00:01 /usr/local/app/tars/tarsnode/data/tars.tarsqueryproperty/bin/
root       368    75  0 08:33 ?        00:00:01 /usr/local/app/tars/tarsnode/data/tars.tarsquerystat/bin/tars
root       369    75  0 08:33 ?        00:00:01 /usr/local/app/tars/tarsnode/data/tars.tarsstat/bin/tarsstat
root       537     1  0 08:41 ?        00:00:00 [tars_start.sh] <defunct>
root       557     1  0 08:41 ?        00:00:00 PHPTest.PHPServer: master process
root       558   557  0 08:41 ?        00:00:00 PHPTest.PHPServer: manager process
root       562   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: task worker process
root       563   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: event worker process
root       564   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: event worker process
root       565   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: event worker process
root       566   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: event worker process
root       567   558  0 08:41 ?        00:00:00 PHPTest.PHPServer: task worker process
root       573   220  0 08:41 pts/1    00:00:00 ps -ef
```

我们可以使用 `IndexController.php` 注释里的 curl 请求方式来测试相应的服务（当然要修改IP和端口）。

```
[root@900dbef166c8 ~]# curl -i -d "user=admin&passwd=12345678" \
> "172.21.0.3:20002/Index/post1"
HTTP/1.1 200 OK
Server: swoole-http-server
Content-Type: text/html
Connection: keep-alive
Date: Tue, 28 Aug 2018 01:34:57 GMT
Content-Length: 13

success:admin[root@900dbef166c8 ~]#
```

开发TARS服务客户端

因为TARS PHP的HTTP协议服务端，可以直接通过HTTP协议访问，所以不需要为它专门开发客户端。这里的TARS服务客户端开发，是指在TARS PHP的HTTP服务端，如果需要访问其它的TARS服务，怎么样创建对应的PHP客户端。

其实参考 TARS CPP 服务端与客户端开发 和 TARS PHP TCP服务端与客户端开发 就可以知道如何做了，这里把官方的例子补全以方便理解。

首先，完成 TARS PHP TCP服务端与客户端开发 服务端的开发与部署，然后把该服务使用的 `test.tars` 文件复制到 `tars` 目录，然后再在 `tars` 目录下新建 `tarsclient.proto.php` ：

```
<?php
return array(
    'appName' => 'PHPTest',
    'serverName' => 'PHPServer',  //这个是PHP TCP服务端的服务名称
    'objName' => 'obj',
    'withServant' => false, //决定是服务端,还是客户端的自动生成
    'tarsFiles' => array(
        './test.tars'
    ),
    'dstPath' => '../src/servant',
    'namespacePrefix' => 'HttpServer\servant',
);
```

`withServant` 必须为false，因为我们这里是生成客户端代码； `dstPath` 指定到 `src` 目录下的 `servant` ，tars2php 工具会自动在该路径创建目录及客户端文件； `namespacePrefix` 需要呼应我们在前面的 `composer.json` 里定义的 psr-4 的名称。

在 `scripts` 目录下创建 `tars2php.sh` ，并赋予执行权限 `chmod u+x tars2php.sh` ：

```
#!/bin/bash

cd ../tars/

php /root/phptars/tars2php.php ./tarsclient.proto.php
```

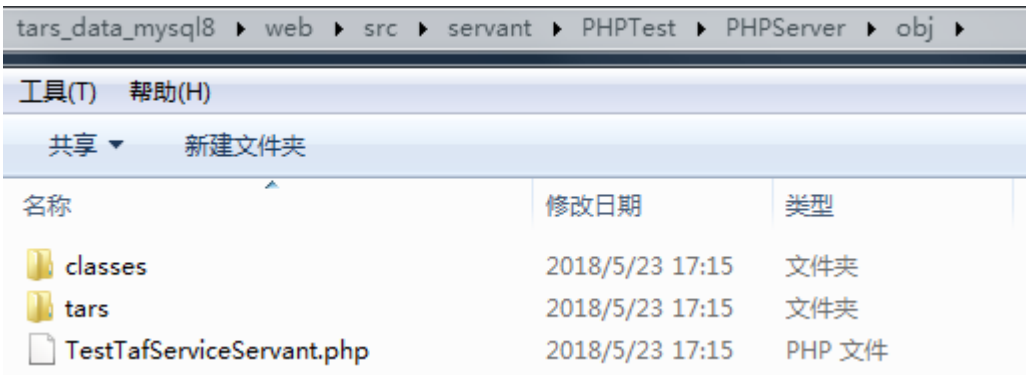
在 `src` 下新建 `conf` 目录，在 `src/conf` 目录下建立 `ENVConf.php` ，注意修改tarsregistry的服务所在IP：

```
<?php
/**
 * Created by PhpStorm.
 * User: liangchen
 * Date: 2018/5/17
 * Time: 下午4:15.
 */
namespace HttpServer\conf;
class ENVConf
{
    public static $locator
        = 'tars.tarsregistry.QueryObj@tcp -h 172.16.0.161 -p 17890';
```

```
    public static $socketMode = 2;
    public static function getTarsConf()
    {
        $table = $_SERVER->table;
        $result = $table->get('tars:php:tarsConf');
        $tarsConf = unserialize($result['tarsConfig']);
        return $tarsConf;
    }
}
```

在 `src` 目录下新建目录 `servant` ，然后执行 `cd scripts && ./tars2php.sh` ，可以看到 `src/servant` 目录下面生成一个三级文件夹 `PHPTest/PHPServer/obj` ，包含：

- classes文件夹 - 存放tars中的struct生成的文件
- tars文件夹 - 存放tars文件
- TestTafServiceServant.php - RPC客户端类文件



修改 `src/controller/IndexController.php` ：

```
<?php
/**
 * Created by PhpStorm.
 * User: liangchen
 * Date: 2018/5/8
 * Time: 下午2:42.
 */
namespace HttpServer\controller;
use HttpServer\component\Controller;
use HttpServer\conf\ENVConf;
use HttpServer\servant\PHPTest\PHPServer\obj\classes\ComplicatedStruct;
use HttpServer\servant\PHPTest\PHPServer\obj\classes\LotofTags;
use HttpServer\servant\PHPTest\PHPServer\obj\classes\OutStruct;
use HttpServer\servant\PHPTest\PHPServer\obj\classes\SimpleStruct;
use HttpServer\servant\PHPTest\PHPServer\obj\TestTafServiceServant;
```

```
use Tars\client\CommunicatorConfig;

class IndexController extends Controller
{
    // curl "172.16.0.161:28887/Index/index" -i
    public function actionIndex()
    {
        $this->cookie('key', 1, 10000000, '/', 'www.github.com');
        $this->sendRaw('success');
    }
    // curl "172.16.0.161:28887/Index/testHeader" -i
    public function actionTestHeader()
    {
        $this->header('test', 1111);
    }
    // curl "172.16.0.161:28887/Index/testStatus" -i
    public function actionTestStatus()
    {
        $this->status(401);
    }
    // Get请求
    // curl "172.16.0.161:28887/Index/get?param=1111&c=d&d=e&e=f" -i
    public function actionGet()
    {
        $param = $this->request->data['get']['param'];
        $this->sendRaw('success:'.$param);
    }
    // Post请求
    // curl -d "user=admin&passwd=12345678"  "172.16.0.161:28887/Index/post1"
-i
    public function actionPost1()
    {
        // 对于content-type为application/x-www-form-urlencoded 的form data
        $admin = $this->request->data['post']['user'];
        $this->sendRaw('success:'.$admin);
    }
    // Post请求
    //curl -H "Content-Type:application/json" -X POST -d '{"user": "admin",
"passwd":"12345678"}' "172.16.0.161:28887/Index/post2" -i
    public function actionPost2()
    {
        // 对于对于content-type为application/json
        $json = $this->request->data['post'];
        $admin = json_decode($json, true)['user'];
        $this->sendRaw('success:'.$admin);
    }
    // Get请求
    // curl -F "image=@profile.jpeg" -F "phone=123456789"
```

```
"172.16.0.161:28887/Index/file" -i
    public function actionFile()
    {
        $fileName = $this->request->data['files']['image']['name'];
        $type = $this->request->data['files']['image']['type'];
        $tmp_name = $this->request->data['files']['image']['tmp_name'];
        $size = $this->request->data['files']['image']['size'];
        $this->sendRaw('success:'.var_export($this->request->data['files']
['image'], true));
    }

    //--------以下调用了TARS PHP TCP服务--------

    // curl "172.16.0.161:28887/Index/testSelf?a=b" -i
    public function actionTestSelf()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(3);

        $servant = new TestTafServiceServant($config);
        $result = $servant->testSelf();

        $this->sendRaw('result:'.$result);
    }

    // curl "172.16.0.161:28887/Index/TestProperty?a=b" -i
    public function actionTestProperty()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);
        $result = $servant->testProperty();

        $this->sendRaw('result:'.$result);
    }

    // curl "172.16.0.161:28887/Index/TestLotofTags?a=b" -i
    public function actionTestLotofTags()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);
```

```
        $servant = new TestTafServiceServant($config);
        $tags = new LotofTags();
        $tags->id = 999;
        $outTags = new LotofTags();

        $result = $servant->testLofofTags($tags, $outTags);

        $this->sendRaw(json_encode($outTags, JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestBasic?a=b" -i
    public function actionTestBasic()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);
        $ret = $servant->testBasic(true, 1, '333', $d, $e, $f);

        $this->sendRaw(json_encode([$d, $e, $f, $ret],
JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestStruct?a=b" -i
    public function actionTestStruct()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);

        $b = new SimpleStruct();
        $b->count = 1098;

        $d = new OutStruct();

        $str = $servant->testStruct(100, $b, $d);

        $this->sendRaw(json_encode([$d, $str], JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestMap?a=b" -i
    public function actionTestMap()
```

```
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);

        $b = new SimpleStruct();
        $b->count = 1098;

        $m1 = ['a' => 'b'];

        $d = new OutStruct();
        $result = $servant->testMap(88, $b, $m1, $d, $m2);

        $this->sendRaw('result:'.$result);
    }

    // curl "172.16.0.161:28887/Index/TestVector?a=b" -i
    public function actionTestVector()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);
        $v1 = ['hahaha'];

        $simpleStruct = new SimpleStruct();
        $simpleStruct->count = 1098;
        $v2 = [$simpleStruct];

        $result = $servant->testVector(999, $v1, $v2, $v3, $v4);

        $this->sendRaw(json_encode([$v3, $v4], JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestReturn?a=b" -i
    public function actionTestReturn()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);
```

```
        $simpleStruct = $servant->testReturn();

        $this->sendRaw(json_encode($simpleStruct, JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestReturn2?a=b" -i
    public function actionTestReturn2()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);
        $map = $servant->testReturn2();

        $this->sendRaw(json_encode($map, JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestComplicatedStruct?a=b" -i
    public function actionTestComplicatedStruct()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);

        $cs = new ComplicatedStruct();
        $cs->str = '111';
        $b = new SimpleStruct();
        $b->count = 1098;
        $cs->mss->pushBack(['a' => $b]);
        $cs->rs = $b;

        $vcs = [$cs];
        $ocs = new ComplicatedStruct();

        $result = $servant->testComplicatedStruct($cs, $vcs, $ocs, $ovcs);

        $this->sendRaw('result:'.$result);
    }

    // curl "172.16.0.161:28887/Index/TestComplicatedMap?a=b" -i
    public function actionTestComplicatedMap()
    {
        $config = new CommunicatorConfig();
```

```
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);

        $cs = new ComplicatedStruct();
        $cs->str = '111';
        $b = new SimpleStruct();
        $b->count = 1098;
        $cs->mss->pushBack(['a' => $b]);
        $cs->rs = $b;

        $mcs = ['test' => $cs];
        $result = $servant->testComplicatedMap($mcs, $omcs);

        $this->sendRaw(json_encode($omcs, JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestEmpty?a=b" -i
    public function actionTestEmpty()
    {
        $config = new CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('PHPTest.PHPHttpServer');
        $config->setSocketMode(ENVConf::$socketMode);

        $servant = new TestTafServiceServant($config);

        $d = new OutStruct();
        $ret = $servant->testEmpty(111, $b1, $in2, $d, $v3, $v4);

        $this->sendRaw(json_encode([$ret], JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestConf?a=b" -i
    public function actionTestConf()
    {
        $tarsConf = ENVConf::getTarsConf();
        $this->sendRaw(json_encode($tarsConf, JSON_UNESCAPED_UNICODE));
    }

    // curl "172.16.0.161:28887/Index/TestRemoteLog?a=b" -i
    public function actionTestRemoteLog()
    {
        $config = new \Tars\client\CommunicatorConfig();
        $config->setLocator(ENVConf::$locator);
        $config->setModuleName('tedtest');
```

```
        $config->setCharsetName('UTF-8');

        $logServant = new \Tars\log\LogServant($config);
        $result = $logServant->logger('PHPTest', 'PHPHttpServer', 'ted.log',
'%Y%m%d', ['hahahahaha']);

        $this->sendRaw(json_encode($result, JSON_UNESCAPED_UNICODE));
    }
}
```

进入 `src` 目录执行 `composer run-script deploy` 对项目进行打包，会在 `src` 目录下生成一个新的 .tgz 文件。

将打好的包发布到TARS平台，因为之前已经发布过一版，所以会产生一个新的版本。

然后我们可以用 `IndexController.php` 注释里的 curl 请求方式来测试相应的服务（当然要修改IP和端口）。

使用Kong来实现TARS PHP HTTP服务网关

按照 Kong Docker 的说明，启动Kong的容器：

```
# 创建容器网络
docker network create kong-net

# 使用PostgreSQL数据库
docker run -d --name kong-database \
  --network=kong-net \
  -p 5432:5432 \
  -e "POSTGRES_USER=kong" \
  -e "POSTGRES_DB=kong" \
  postgres:9.6

# 下面的命令运行一次即可，用于创建数据库与初始化数据
docker run --rm \
  --network=kong-net \
  --link kong-database:kong-database \
  -e "KONG_DATABASE=postgres" \
  -e "KONG_PG_HOST=kong-database" \
  -e "KONG_CASSANDRA_CONTACT_POINTS=kong-database" \
  kong kong migrations up

# 启动Kong容器
docker run -d --name kong \
```

```
  --network=kong-net \
  --link kong-database:kong-database \
  -e "KONG_DATABASE=postgres" \
  -e "KONG_CASSANDRA_CONTACT_POINTS=kong-database" \
  -e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
  -e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \
  -e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
  -e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
  -e "KONG_ADMIN_LISTEN=0.0.0.0:8001" \
  -e "KONG_ADMIN_LISTEN_SSL=0.0.0.0:8444" \
  -p 8000:8000 \
  -p 8443:8443 \
  -p 8001:8001 \
  -p 8444:8444 \
  kong

# 把之前的容器 tars_mysql8 连接到 kong-net 网络
# 否则kong无法通过hostname来与 tars_mysql8 建立连接
docker network connect kong-net tars_mysql8
```

我们可以使用 `curl -i http://192.168.99.100:8001/` (Windows下Docker虚机的IP)或 `curl -i http://localhost:8001/` (Linux or Mac) 来检查Kong服务是否已可使用。注意 `8000` 和 `8443` 是Kong用来接收API访问请求的端口（分别是HTTP和HTTPS协议）； `8001` 和 `8444` 是Kong的管理API所使用的端口（分别是HTTP和HTTPS协议），这两个端口不应该对外部网络开放。

下面我们创建一个Kong的服务（Service）：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/services/ \
  --data 'name=tars-service' \
  --data 'url=http://tars_mysql8:20002/Index/'
```

注意这里 `url` 里使用的 `tars_mysql8` 是我们一开始创建的TARS容器名称；我们把 `/Index/` 路径（Path）也写在 `url` 里面了，这样我们在后面请求服务时就不用再带上这个路径了；另外端口也要配置为我们前面部署服务时指定的端口 20002。如果返回的信息一切正确，我们就可以为这个Service增加访问的路由（Route）：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/services/tars-service/routes \
  --data 'hosts[]=tars.net'
```

这条路由命令的意思是，只要是访问域名为 `tars.net` ，就将请求转发给我们增加的服务 `tars-service` 。检查返回的信息，如果没有问题，我们就可以通过Kong来访问TARS PHP HTTP服务了：

```
curl -i -d "user=admin&passwd=12345678" \
  --url "http://192.168.99.100:8000/post1" \
  --header 'Host: tars.net'
```

我们使用Kong当然希望能使用它提供的服务鉴权功能，这里我们为 `tars-service` 服务安装key-auth插件（Plugin）：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/services/tars-service/plugins/ \
  --data 'name=key-auth'
```

使用下面的命令来检查一下key-auth是否生效，生效的话返回的结果会告诉我们 `401 Unauthorized` ，消息体是 `"message": "No API key found in request"` 。

```
curl -i -X GET \
  --url http://192.168.99.100:8000/ \
  --header 'Host: tars.net'
```

既然key-auth插件已生效，我们需要创建一个服务消费者（Consumer）来获取访问权限：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/consumers/ \
  --data "username=tester"
```

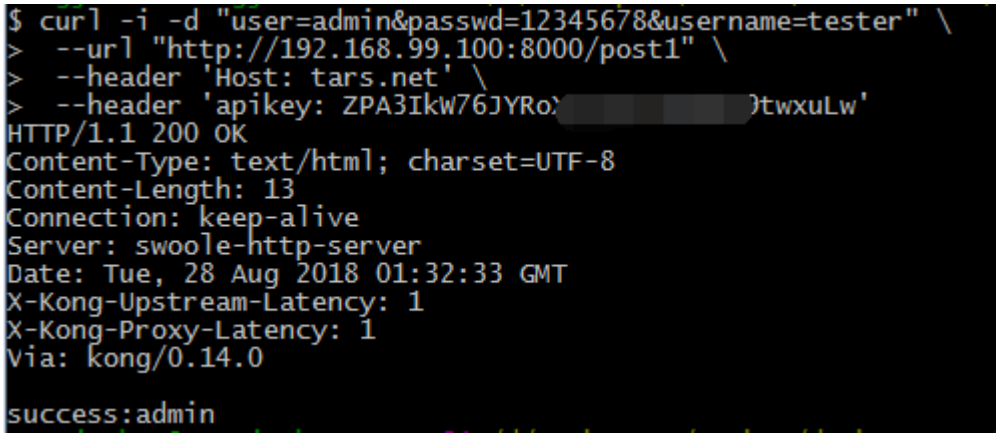
返回结果如果没有问题，我们为消费者 `tester` 生成API Key （如果没有最后一行 –data 参数，key-auth插件会自动生成一个key）：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/consumers/tester/key-auth/ \
  --data 'key=ENTER_KEY_HERE'
```

然后我们就可以使用消费者 `tester` 和它的API Key来访问我们的API了：

```
curl -i -d "user=admin&passwd=12345678&username=tester" \
  --url "http://192.168.99.100:8000/post1" \
```

```
  --header 'Host: tars.net' \
  --header 'apikey: ENTER_KEY_HERE'
```



使用Kong来实现负载均衡（Load Balance）

因为TARS PHP HTTP服务不能利用到TARS的负载均衡能力（非TARS协议，不走TARS的流量处理），所以我们可以利用Kong来实现负载均衡。

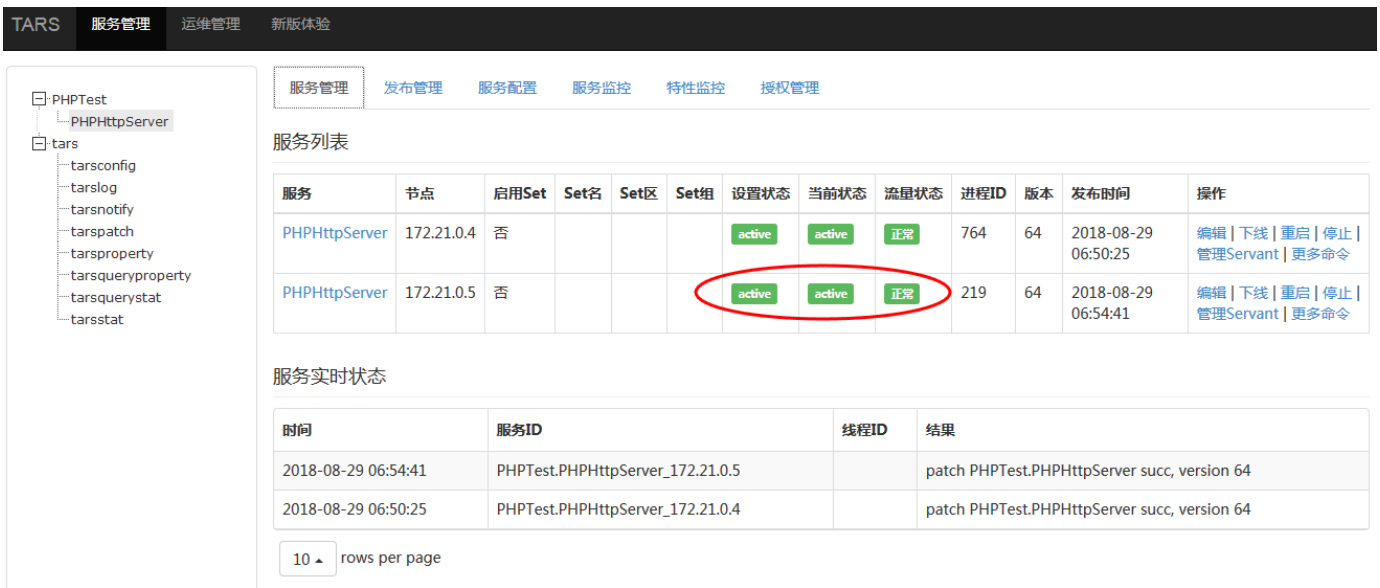
首先，我们添加一个节点容器：

```
docker run -d -it --name tars_node \
  --network=kong-net \
  -e MASTER=tars_mysql8 --link tars_mysql8 \
  -v /c/Users/tangramor/tars_node_data:/data \
  tangramor/tars-node:php7mysql8
```

上面的命令启动了 `tangramor/tars-node:php7mysql8` 容器 **tars_node**，连接到自定义的网络 `kong-net` ，将本地的一个目录 `/c/Users/tangramor/tars_node_data` 挂载为容器的 /data 目录，指定了主节点（MASTER）为我们之前创建的 `tars_mysql8` 容器并连接了该容器。我们可以 `docker exec -it tars_node bash` 进入容器然后运行 `ip address` 获得该容器IP。

然后我们进入TARS平台网页，点击“运维管理” => “扩容”，选择我们已经部署的应用，把目标IP设为节点容器 `tars_node` 的IP，预扩容、扩容，这样一个新的节点就接入了。

然后回到“服务管理”页面，在新节点上再部署一遍我们的应用代码（不需要上传，直接用已有的版本代码即可），没有问题的话应用就会很快运行在新节点上。



我们可以使用 `IndexController.php` 注释里的 curl 请求方式来测试相应的服务（修改IP和端口为新节点的）。

确认了服务可用，我们就可以给Kong的服务（Service）添加上游（Upstream）配置了：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/upstreams/ \
  --data "name=tars_mysql8"
```

这条命令给我们前面定义的服务（Service） `tars-service` 添加了一个upstream，它不是以Service的名称来确定的，而是以Service使用的Host（这里是 `tars_mysql8` ）来定位服务，更多的选项参数可以参考 Kong的Upstream文档 。然后我们把我们已经部署了服务的两个节点都加入到这个upstream里，这需要创建两个目标（Target）对象：

```
curl -i -X POST \
  --url http://192.168.99.100:8001/upstreams/tars_mysql8/targets \
  --data "target=tars_mysql8:20002&weight=100"

curl -i -X POST \
  --url http://192.168.99.100:8001/upstreams/tars_mysql8/targets \
  --data "target=tars_node:20002&weight=100"
```

注意我们用到了容器名，它在Docker网络中就是该容器的Hostname， `Hostname:Port` 就是 `target` 的参数值， `weight` 参数值是负载均衡用来计算负载权重的设置，缺省为 100，取值为 0-1000，如果取 0 就是禁用了该target。

如果返回没有问题，我们的负载均衡就设置完成了。我们可以使用和之前一样的请求来访问服务，Kong会自动把服务分发到两个节点中的某一个：

```
curl -i -d "user=admin&passwd=12345678&username=tester" \
  --url "http://192.168.99.100:8000/post1" \
  --header 'Host: tars.net' \
  --header 'apikey: ENTER_KEY_HERE'
```

使用SuperBenchmark对demo进行性能测试

为了了解TARS PHP HTTP服务的性能，我们需要找一个工具来对上面的demo进行性能测试。Apache的压测工具 `ab` 或者 Web Bench 是我们常用的命令行工具，不过它们的输出也是字符类型，不够直观，这里我们选用了一个使用 d3.js 实现HTML准实时展示的压测工具 SuperBenchmarker。

在Windows上安装SuperBenchmarker需要通过 chocolatey 工具，这个工具类似Linux下的APT或者Mac下的Homebrew等软件管理软件，提供了通过命令行自动安装Windows工具的功能。先安装 `chocolatey` ，“以管理员身份运行”命令提示符，然后输入下面的命令：

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))"
&& SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

根据提示安装好 `chocolatey` 后，我们就可以用它安装 `SuperBenchmarker` ：

```
cinst SuperBenchmarker
```

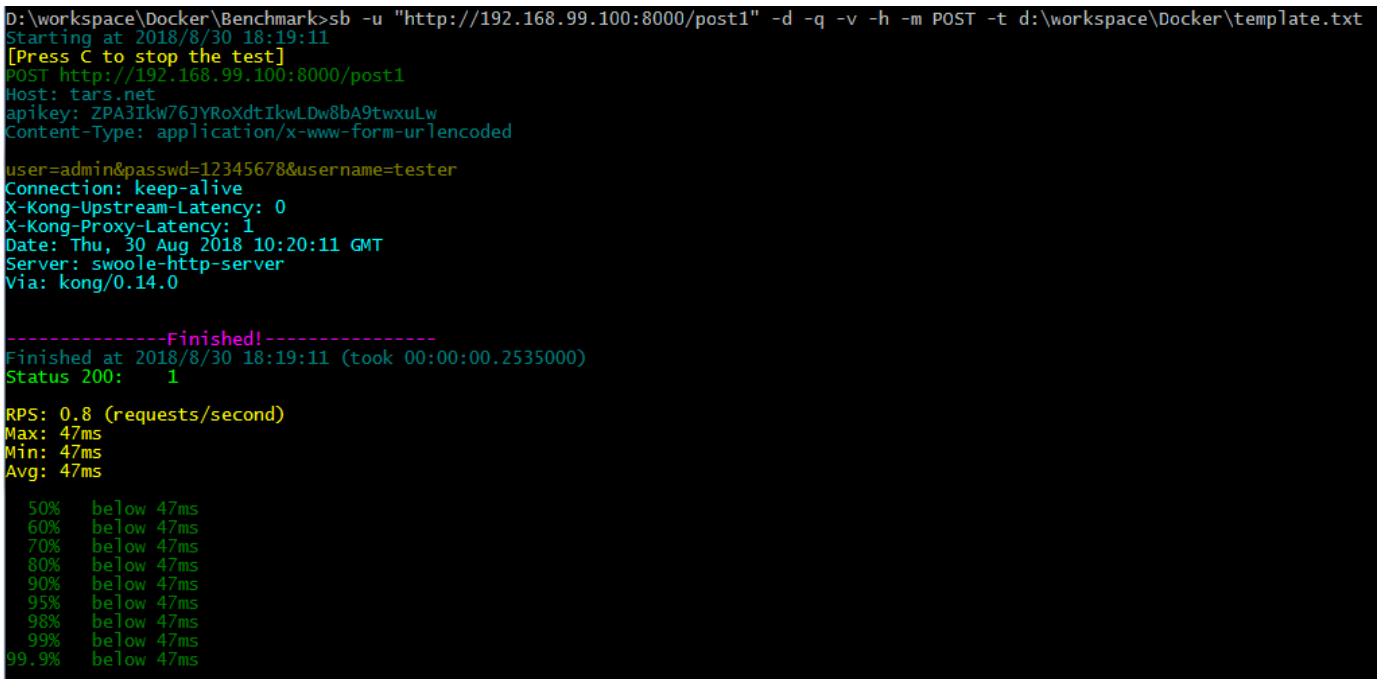
一旦安装完成，我们就可以使用 `sb` 这个带着浓浓中二气息的命令来进行压力测试了。因为前面一直用一个POST方法做测试，这里我们为了继续用这个方法，需要创建一个外部文件 `template.txt` 以便SuperBenchmarker引用：

```
Host: tars.net
apikey: ENTER_KEY_HERE
Content-Type: application/x-www-form-urlencoded

user=admin&passwd=12345678&username=tester
```

注意前面三行是Header参数，它们与最后一行之间一定要有空行，最后一行是POST的Body数据，我们使用的是 `application/x-www-form-urlencoded` 格式，因为要测试的接口（post1）要求这个格式。

我们先干跑（Dry-run）一下来测试命令请求是否正确：

```
sb -u "http://192.168.99.100:8000/post1" -d -q -v -h -m POST -t d:\workspace
\Docker\template.txt
```



- -d：–dryRun，用于执行一次测试
- -q：–onlyRequest，在dry-run模式时只显示请求
- -v：–verbose，显示详细信息
- -h：–headers，显示请求与响应的头部信息
- -m：–method，HTTP请求方式，缺省为GET
- -t：–template，请求模版文件的路径

其它选项可以参考SuperBenchmarker的帮助信息 `sb -h` 。

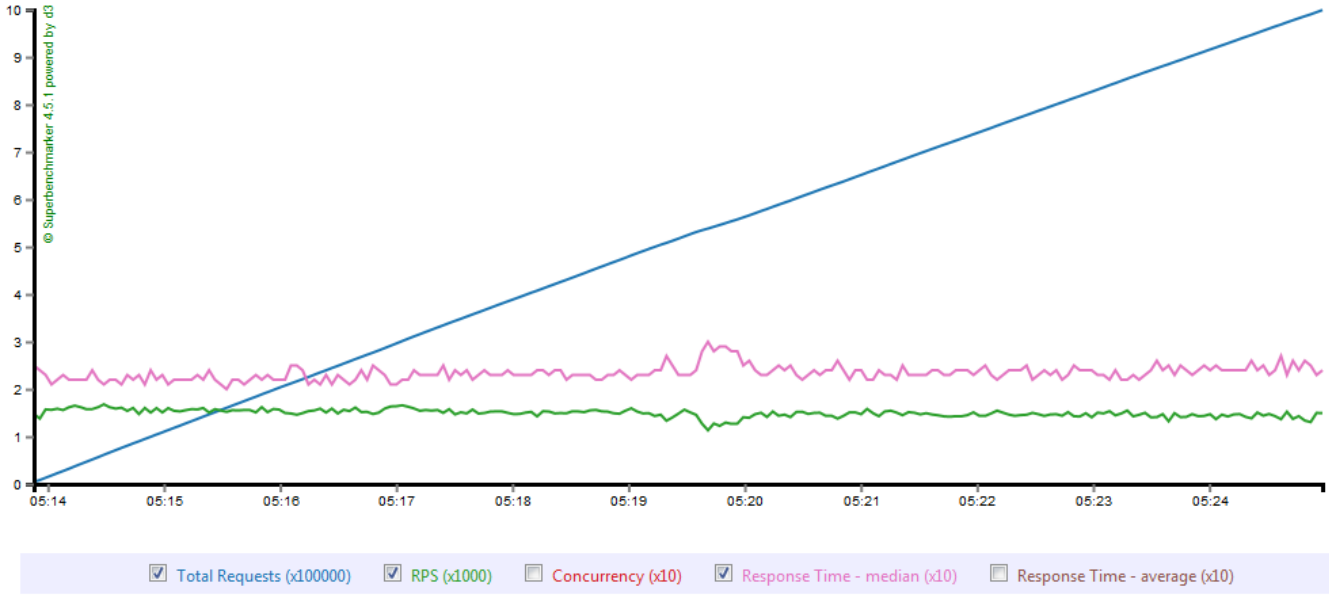
然后我们再测试一下返回值是否正确（去掉 `-q` ）：

```
sb -u "http://192.168.99.100:8000/post1" -d -v -h -m POST -t d:\workspace
\Docker\template.txt
```

```
D:\workspace\Docker\Benchmark>sb -u "http://192.168.99.100:8000/post1" -d -v -h -m POST -t d:\workspace\Docker\template.txt
Starting at 2018/8/30 18:19:57
[Press C to stop the test]
POST http://192.168.99.100:8000/post1
Host: tars.net
apikey: ZPA3IkW76JYRoXdtIkwLDw8bA9twxuLw
Content-Type: application/x-www-form-urlencoded

Connection: keep-alive
X-Kong-Upstream-Latency: 2
X-Kong-Proxy-Latency: 1
Date: Thu, 30 Aug 2018 10:20:57 GMT
Server: swoole-http-server
Via: kong/0.14.0

success:admin

---------------Finished!---------------
Finished at 2018/8/30 18:19:57 (took 00:00:00.2575000)
Status 200:    1

RPS: 0.8 (requests/second)
Max: 46ms
Min: 46ms
Avg: 46ms

  50%   below 46ms
  60%   below 46ms
  70%   below 46ms
  80%   below 46ms
  90%   below 46ms
  95%   below 46ms
  98%   below 46ms
  99%   below 46ms
99.9%   below 46ms
```

一切正常的话，我们跑100万次请求来看看，并发请求这里我设为50个：

```
sb -u "http://192.168.99.100:8000/post1" -m POST -t d:\workspace\Docker
\template.txt -n 1000000 -c 50
```



Requests: 1000000

RPS: 1491.2

90th Percentile: 39ms

95th Percentile: 45ms

99th Percentile: 61ms

Average: 25.1ms

Min: 0ms

Max: 218ms

Status: finished

Start time: 2018-08-30T17:13:49.768+08:00

Finish time: 2018-08-30T17:25:00.350+08:00

Command: "C:\ProgramData\chocolatey\lib\SuperBenchmarker\tools\sb.exe" -u "http://192.168.99.100:8000/post1" -m POST -t d:\workspace\Docker\template.txt -n 1000000 -c 50

Status breakdown:

- 200: 1000000

很直观的结果，我就不解读了~