目錄

SwooleDistributed 1.7更新

1.修复redis和mysql的一些bug，统一同步和异步的调用方法和回应结果的结构。redis基本所有的命令均已测试和统一。

2.增加了单元测试模块。

3.增加了mysql，redis，controller的单元测试用例。

4.已知bug修复。

SwooleDistributed 1.6更新

通过协程写法统一异步和同步客户端。

4大模块Model，Controller，View，Task，Model处理异步简单事务，Task处理同步耗时事务，原本开发者需要在Model中使用异步客户端（异步redis，异步mysql），在Task中使用同步客户端（redis扩展，mysql-pdo扩展），代码风格完全不一致，无法重用。现在最新版本的SD框架将托管异步和同步客户端的调用，使用协程模式书写的代码在Model和Task中完全通用，并且Task和Model之间可以完成互相调用，代码100%可重用，采用协程代码风格开发者将可以忽略异步与同步的区别。通过task调用model的方法可以很容易将一个耗时任务优化到任务队列中而不需要更改任何代码。

SwooleDistributed 1.5更新

bug修复，稳定性优化

SwooleDistributed 1.4更新

bug修复

SwooleDistributed 1.3更新

1. 增强http功能。
2. 提供了静态页面的访问，加了www目录，优化了404页面，优化了默认的路由。
3. 优化http代码性能提升
4. 自动reload，可以使用inotify扩展增强性能。
5. 支持WebSocket，服务器可以同时接收tcp，http，websocket的访问，并且绑定uid后可以互相推送消息。websocket的发送消息的api和tcp一样，会自动判断是websocket还是tcp连接。
6. 结构略微调整，更为清晰。

SwooleDistributed 1.2更新

完善协程，协程中允许使用异常。 controller中添加了异常的回调，方便统一处理异常回复客户端。

SwooleDistributed 1.1更新

修复了单机模式下send的bug。
在1.0基础上增加了mysql，redis，task，model异步协程模式。
增加了protobuf pack，实现基于protobuf的RPC。

SwooleDistributed 1.0使用手册

本手册适用于最新版本SwooleDistributed1.0(发布时间2016-08-03)。
依赖Swoole扩展及composer，monolog/monolog，hassankhan/config，voku/anti-xss，league/plates，Miner
需要注意Swoole版本>1.8.9，php版本>7.0.0，考虑到效率问题和未来方向目前没有对php5进行适配。
只能正常运行在linux系统下。

qq交流群：569037921

使用前须知

SwooleDistributed 是一个完整的TCP/HTTP框架，提供了服务器程序开发的绝大多数支持，liunx集群已经有很多成熟的工具，SwooleDistributed 将利用这些工具建设服务器集群。

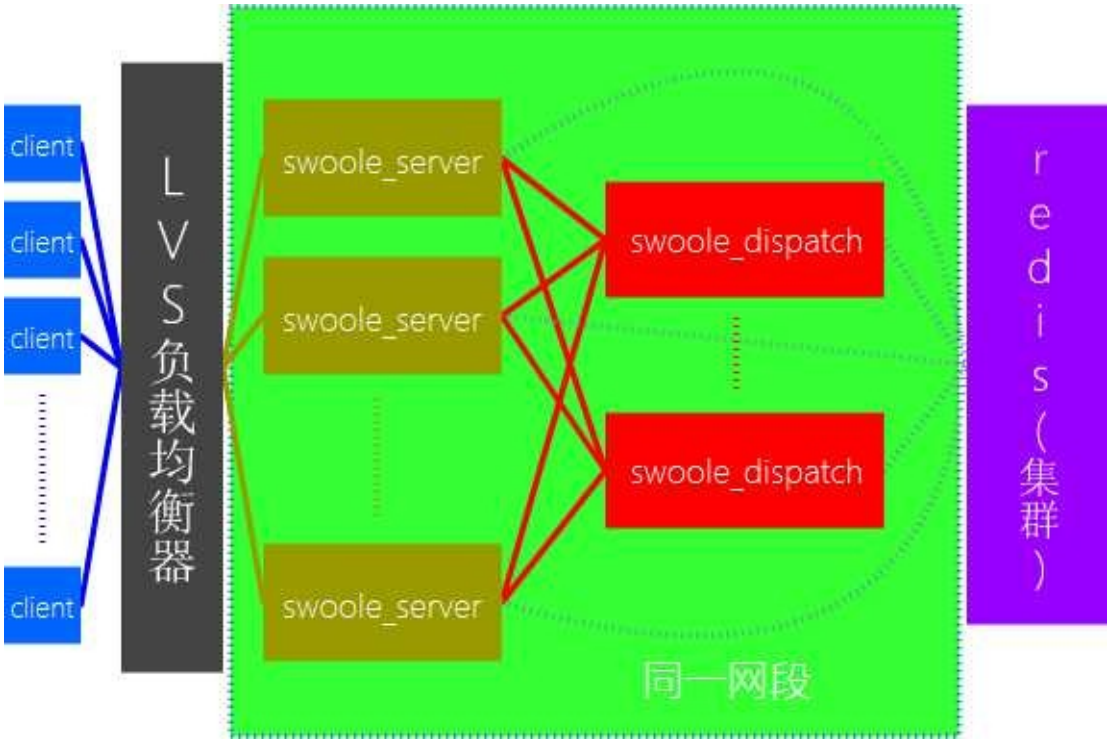
单机模式只需要启动 SwooleDistributed Server 即可无需多余的配置，下面注重介绍集群部署。

工具准备：LVS，keepalive，redis

SwooleDistributed 集群概念介绍：SwooleDistributed 每个服务都是无状态的服务器，和传统的gateway方式不同，SwooleDistributed 为了更好的利用LVS的性能将数据的同步全部交于redis存储，而服务器之间的通讯交给后方的dispatch服务器处理。传统的gateway服务器将维护大量的用户数据，作为一个消息中心集中处理用户的数据和及时转发业务服务器的回复，这种设计会使gateway作为服务器的数据传递中心，在遇到大流量数据的情况gateway会成为整个服务器的性能瓶颈。而LVS作为负载均衡器可以利用linux下的ip隧道等技术将来往的流量平均引流到后排服务器上，业务服务器的回复是直接传递给客户端无需中间服务器转达，这样能够承担更大的流量。并且由于SwooleDistributed 是无状态的服务器更可以实现弹性部署。

部署准备：自行搭建好LVS和keeplived，后排SwooleDistributed服务器和dispatch服务器确保在同一个内网段，根据业务需要安排SwooleDistributed服务器和dispatch服务器的数量，一般4台SwooleDistributed搭配1台dispatch即可拓扑图如下，更高的级别还需设置redis的主从读写分离，将dispatch和只读redis配置到同一台物理机上通过unixsock进行通讯。SwooleDistributed和dispatch采用了自发现服

务，无需进行集群配置，建议先启动dispatch后启动SwooleDistributed，否则最多需要30秒dispatch和SwooleDistributed才能建立连接。



SwooleDistributed源码地址

https://github.com/tmtbe/SwooleDistributed

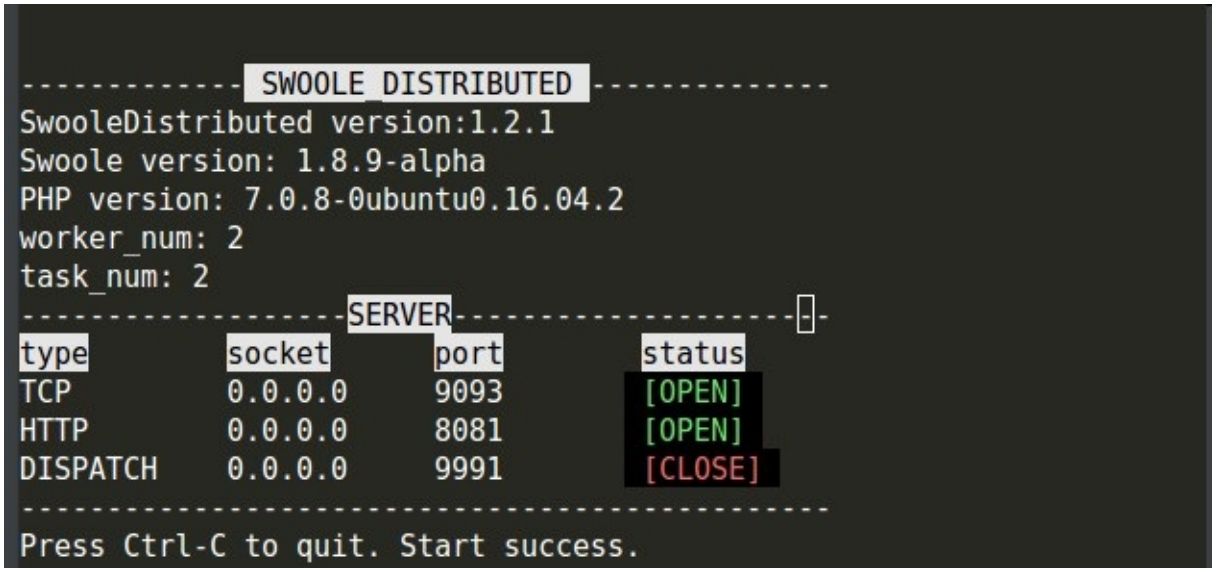
SwooleDistributed启动方式

```
php start_swoole_dispatch.php start //启动dispatch服务器（集群才需要启动）
php start_swoole_server.php start //启动SwooleDistributed服务器
```

start -d 代表以守护进程方式启动
命令：
restart 默认为守护进程方式重启

stop 停止服务

reload 平滑重启



SwooleDistributed的测试

源码中包含一个start_swoole_client.php文件可以通过这个进行tcp的测试，至于http测试可以用ab或者其他工具。

start_swoole_client的原理：

这个文件通过配置多个服务器地址可以测试集群环境，他的压测是是包含准确性测试要求服务器返回和请求一样的消息才算成功。用于测试集群环境消息的准确性和性能指标。由于采用的是swoole异步客户端for循环会占满socket的缓冲区，强烈建议压测服务器独立部署测试。

开源协议

```
  Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, rep
roduction,
      and distribution as defined by Sections 1 through 9 of thi
s document.

      "Licensor" shall mean the copyright owner or entity author
ized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity a
nd all
      other entities that control, are controlled by, or are und
er common
      control with that entity. For the purposes of this definit
ion,
      "control" means (i) the power, direct or indirect, to caus
e the
      direction or management of such entity, whether by contrac
t or
      otherwise, or (ii) ownership of fifty percent (50%) or mor
e of the
      outstanding shares, or (iii) beneficial ownership of such
entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.
```

```
      "Source" form shall mean the preferred form for making mod
ifications,
      including but not limited to software source code, documen
tation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanica
l
      transformation or translation of a Source form, including
but
      not limited to compiled object code, generated documentati
on,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Sourc
e or
      Object form, made available under the License, as indicate
d by a
      copyright notice that is included in or attached to the wo
rk
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source
or Object
      form, that is based on (or derived from) the Work and for
which the
      editorial revisions, annotations, elaborations, or other m
odifications
      represent, as a whole, an original work of authorship. For
 the purposes
      of this License, Derivative Works shall not include works
that remain
      separable from, or merely link (or bind by name) to the in
terfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, includin
g
      the original version of the Work and any modifications or
additions
```

```
      to that Work or Derivative Works thereof, that is intentio
nally
      submitted to Licensor for inclusion in the Work by the cop
yright owner
      or by an individual or Legal Entity authorized to submit o
n behalf of
      the copyright owner. For the purposes of this definition,
"submitted"
      means any form of electronic, verbal, or written communica
tion sent
      to the Licensor or its representatives, including but not
limited to
      communication on electronic mailing lists, source code con
trol systems,
      and issue tracking systems that are managed by, or on beha
lf of, the
      Licensor for the purpose of discussing and improving the W
ork, but
      excluding communication that is conspicuously marked or ot
herwise
      designated in writing by the copyright owner as "Not a Con
tribution."

      "Contributor" shall mean Licensor and any individual or Le
gal Entity
      on behalf of whom a Contribution has been received by Lice
nsor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and condi
tions of
      this License, each Contributor hereby grants to You a perp
etual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevoc
able
      copyright license to reproduce, prepare Derivative Works o
f,
      publicly display, publicly perform, sublicense, and distri
bute the
      Work and such Derivative Works in Source or Object form.
```

```
   3. Grant of Patent License. Subject to the terms and conditio
ns of
      this License, each Contributor hereby grants to You a perp
etual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevoc
able
      (except as stated in this section) patent license to make,
 have made,
      use, offer to sell, sell, import, and otherwise transfer t
he Work,
      where such license applies only to those patent claims lic
ensable
      by such Contributor that are necessarily infringed by thei
r
      Contribution(s) alone or by combination of their Contribut
ion(s)
      with the Work to which such Contribution(s) was submitted.
 If You
      institute patent litigation against any entity (including
a
      cross-claim or counterclaim in a lawsuit) alleging that th
e Work
      or a Contribution incorporated within the Work constitutes
 direct
      or contributory patent infringement, then any patent licen
ses
      granted to You under this License for that Work shall term
inate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of
 the
      Work or Derivative Works thereof in any medium, with or wi
thout
      modifications, and in Source or Object form, provided that
 You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
```

```
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent n
otices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative
Works
          that You distribute, all copyright, patent, trademark,
 and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any par
t of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of i
ts
          distribution, then any Derivative Works that You distr
ibute must
          include a readable copy of the attribution notices con
tained
          within such NOTICE file, excluding those notices that
do not
          pertain to any part of the Derivative Works, in at lea
st one
          of the following places: within a NOTICE text file dis
tributed
          as part of the Derivative Works; within the Source for
m or
          documentation, if provided along with the Derivative W
orks; or,
          within a display generated by the Derivative Works, if
 and
          wherever such third-party notices normally appear. The
 contents
          of the NOTICE file are for informational purposes only
 and
          do not modify the License. You may add Your own attrib
ution
          notices within Derivative Works that You distribute, a
```

```
longside
          or as an addendum to the NOTICE text from the Work, pr
ovided
          that such additional attribution notices cannot be con
strued
          as modifying the License.

      You may add Your own copyright statement to Your modificat
ions and
      may provide additional or different license terms and cond
itions
      for use, reproduction, or distribution of Your modificatio
ns, or
      for any such Derivative Works as a whole, provided Your us
e,
      reproduction, and distribution of the Work otherwise compl
ies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state o
therwise,
      any Contribution intentionally submitted for inclusion in
the Work
      by You to the Licensor shall be under the terms and condit
ions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede
or modify
      the terms of any separate license agreement you may have e
xecuted
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use
the trade
      names, trademarks, service marks, or product names of the
Licensor,
      except as required for reasonable and customary use in des
cribing the
      origin of the Work and reproducing the content of the NOTI
CE file.
```

```
   7. Disclaimer of Warranty. Unless required by applicable law
or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASI
S,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either expre
ss or
      implied, including, without limitation, any warranties or
conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FO
R A
      PARTICULAR PURPOSE. You are solely responsible for determi
ning the
      appropriateness of using or redistributing the Work and as
sume any
      risks associated with Your exercise of permissions under t
his License.

   8. Limitation of Liability. In no event and under no legal th
eory,
      whether in tort (including negligence), contract, or other
wise,
      unless required by applicable law (such as deliberate and
grossly
      negligent acts) or agreed to in writing, shall any Contrib
utor be
      liable to You for damages, including any direct, indirect,
 special,
      incidental, or consequential damages of any character aris
ing as a
      result of this License or out of the use or inability to u
se the
      Work (including but not limited to damages for loss of goo
dwill,
      work stoppage, computer failure or malfunction, or any and
 all
      other commercial damages or losses), even if such Contribu
tor
      has been advised of the possibility of such damages.
```

```
   9. Accepting Warranty or Additional Liability. While redistri
buting
      the Work or Derivative Works thereof, You may choose to of
fer,
      and charge a fee for, acceptance of support, warranty, ind
emnity,
      or other liability obligations and/or rights consistent wi
th this
      License. However, in accepting such obligations, You may a
ct only
      on Your own behalf and on Your sole responsibility, not on
 behalf
      of any other Contributor, and only if You agree to indemni
fy,
      defend, and hold each Contributor harmless for any liabili
ty
      incurred by, or claims asserted against, such Contributor
by reason
      of your accepting any such warranty or additional liabilit
y.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the follo
wing
      boilerplate notice, with the fields enclosed by brackets "
[]"
      replaced with your own identifying information. (Don't inc
lude
      the brackets!)  The text should be enclosed in the appropr
iate
      comment syntax for the file format. We also recommend that
 a
      file or class name and description of purpose be included
on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.
```

```
   Copyright Jincheng.Zhang [tmtbe@163.com]

   Licensed under the Apache License, Version 2.0 (the "License"
);
   you may not use this file except in compliance with the Licen
se.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, so
ftware
   distributed under the License is distributed on an "AS IS" BA
SIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
   See the License for the specific language governing permissio
ns and
   limitations under the License.
```

SwooleDistributed特性

基于Swoole扩展开发

SwooleDistributed是基于swoole1.x扩展开发，有关swoole请参考

swoole

支持分布式系统

SwooleDistributed搭载dispatch组件实现分布式部署，2者均是可以集群部署无单点故障。

单服务多协议

SwooleDistributed服务器通过配置不同的端口，可以同时处理tcp和http请求，比如开放http api实现给所有tcp的客户端广播消息这种需求在SwooleDistributed中实现起来非常简单。tcp和http请求通过自定义路由器可以指向同一个controller，再通过设置tcp，http方法的前缀可以分离tcp和http的处理方法。

支持全局广播或者向任意客户端推送数据

SwooleDistributed提供非常方便的API，可以全局广播数据、可以向某个群体广播数据、也可以向某个特定客户端推送数据。配合定时任务，也可以定时推送数据。

支持投递任务

Task是swoole扩展中task的封装，在SwooleDistributed可以非常方便的进行异步（同步）任务投递。

支持定时任务

Timer是SwooleDistributed的定时器，通过Config配置文件结合Task可以实现丰富的定时任务。

支持异步Redis

SwooleDistributed实现了异步Redis连接池的封装，只需调用redis方法而不需要参与维护连接池。

使用方法与调用结果和php-redis扩展保持一致。

支持异步Mysql（支持事务）

SwooleDistributed借助Miner实现了Mysql的语法构建器。
Miner的用法：Miner
SwooleDistributed实现了异步Mysql连接池的封装，只需调用Mysql方法而不需要参与维护连接池。
另外SwooleDistributed率先支持异步mysql事务。

支持模板引擎

SwooleDistributed的模板引擎采用了plates。
plates的文档：plates

高并发，高性能，低配置

SwooleDistributed使用了swoole的扩展，内部的controller和model采用了对象池的模式，对象重用性能强悍。
SwooleDistributed和dispatch实现了内网自动发现，集群部署零配置。

异步与同步

SwooleDistributed中Controller和Model均是支持异步代码，可以直接写回调无需担心高并发下可能存在的数据错乱。
Task是同步堵塞的，服务于Controller和Model用于处理耗时的任务。

支持协程

非swoole2.0提供的协程，使用yield关键字，支持php7
所有的异步方法都提供了协程风格，书写代码更加流畅。
协程支持协程嵌套，异常处理。

统一异步与同步的写法

通过协程统一了异步和同步的代码风格，task，model之间现在可以互相调用。

化繁为简，从零开始的框架设计

前言

经历了一个又一个项目，也接触了很多的PHP框架，我欣赏CI的简约，又贪婪swoole的效率，我将CI和swoole很草率的结合到了一起。起初呢风平浪静，慢慢的就遇到了不少的瓶颈，毕竟CI的设计理念还是贴合FPM模式，如何更加得心应手的使用swoole，同时追求开发上和运行时的效率呢，最主要的还是要方便扩展，就萌生了SwooleDistributed这个开源框架的想法。

在SwooleDistributed发布之前，开源社区还没有过针对swoole的分布式框架，起初的目的并不是一个完整的应用框架，而是一个简单的分布式通讯框架，后来需求变多了，框架也就慢慢的丰满了。

分布式

分布式这东西并不是有多神秘，但一个框架在基础构思中就包含分布式的思想，那无疑方便对以后的扩展。分布式系统涉及到多太物理机之间的调控，配置起来也是较为麻烦，SwooleDistributed使用了内网发现的技术手段，自动发现集群环境的物理机进行连接，简化了配置，甚至达到了无配置。

SwooleDistributed在前期可以控制成本的使用单机模式进行部署，也可以在后期进行水平扩展，对逻辑代码无需任何的改动。

你所需要的就是多增加物理机，跑上服务器就行啦。

MVCT

解决了底层的分布式通讯问题，接下来就是MVC结构的搭建了，这部分借鉴了CI的设计，使用Loader模块加载对应的Model，通过路由访问对应的Controller。相信使用过CI框架的工程师很容易就上手。
此外引入了Swoole独特的Task，将swoole的Task进行了封装优化，更加易于使用。

```
//TestTask.php
class TestTask extends Task
{
    public function test()
    {
        print_r("test timer task\n");
    }
}
//TestController.php
class TestController extends Controller
{
  public function http_test_task()
    {
        $task = $this->loader->task('TestTask');
        $task->test();
        $result = yield $task->coroutineSend();
        $this->http_output->end($result);
    }
}
```

通过定时器和Task的结合，开发者可以很方便的制作定时任务，而这一切只需要简单的配置即可。

```
/**
 * timerTask定时任务
 * （必填）task名称 task_name
 * （必填）执行task的方法 method_name
 * （选填）执行区间 [start_time,end_time) 格式： Y-m-d H:i:s 没有代表
一直执行
 * （必填）执行间隔 interval_time 单位： 秒
 * （选填）最大执行次数 max_exec，默认不限次数
 */
$config['timerTask'][] = [
    'task_name'=>'TestTask',
    'method_name'=>'test',
    'start_time'=>'Y-m-d 00:00:00',
    'end_time'=>'Y-m-d 23:59:59',
    'interval_time'=>'2',
];
```

异步连接池

swoole提供了redis和mysql的异步客户端，大大提高的服务端的效率，但问题又来了，如果使用异步客户端，就必须维护一个异步连接池。

SwooleDistributed设计了一个通用的连接池模块，通过这个模块可以快捷简单的创建客户端的连接池，不仅如此redis和mysql的连接池早已封装在核心代码中，开发者只需调用其中的Api无需关注连接池的维护。

swoole的异步redis客户端是基于hredis和我们通常使用的redis扩展不一样，在使用批量方法时回调的值是有所区别，SwooleDistributed屏蔽了这一点不同之处，使用起来和平常的redis扩展一致。

异步的mysql客户端事务通常是比较难写的，SwooleDistributed同样针对mysql事务进行了封装，使用起来也非常的方便。

此外mysql也提供了一个语法构建器，可以简单方便的构建mysql语法。

```
 public function test_coroutine()
    {
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();
        $result = yield $mySqlCoroutine;
        $redisCoroutine = $this->redis_pool->coroutineSend('get'
, 'test');
        $result = yield $redisCoroutine;
        return $result;
    }
```

协程

之前我去过很多家公司，交流的过程中发现有些公司使用php做短连接，用golang做长连接，我问他们为何不用swoole呢？回答都是异步回调太难写了。 确实异步回调写起来很不好看，可能会有多层回调的嵌套，复杂点的代码非常的难看，swoole2.0已经使用了协程，但首先是新功能稳定性尚且不知，其次不支持php7，于是我呢就对现有框架进行了一次大的调整，通过yield关键字实现了全异步的协程风格。使用起来非常的简单，和一般的写法没有什么太大的区别，只要遵循一个原则，涉及到异步的地方加上yield关键字就可以。

```
    /**
     * 协程测试
     */
    public function http_testCoroutine()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $result = yield $this->testModel->test_coroutine();
        $this->http_output->end($result);
    }
```

协程之间支持嵌套，也支持throw Exception异常捕获，总之用起来和同步的写法基本一致。 协程之间不存在堵塞，也可以通过控制yield的位置调整send和rev的调度策略。 比如：

```
$mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->select('*')
->from('account')->where('uid', 10303)->coroutineSend();
$result = yield $mySqlCoroutine;
$redisCoroutine = $this->redis_pool->coroutineSend('get', 'test'
);
$result = yield $redisCoroutine;
```

这段代码调度策略是mysql_send->mysql_rev->redis_send->redis_rev;
我们调整下yield的位置

```
$mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->select('*')
->from('account')->where('uid', 10303)->coroutineSend();
$redisCoroutine = $this->redis_pool->coroutineSend('get', 'test'
);
$result = yield $mySqlCoroutine;
$result = yield $redisCoroutine;
```

这段代码调度策略变成了mysql_send->redis_send->mysql_rev->redis_rev;

同时支持TCP和HTTP

这样你就可以开启一个服务同时处理TCP和HTTP请求了，代码复用率就高很多了，同时SwooleDistributed还提供了一些策略，方便隔离http和tcp的路由请求。

Protobuf

提供了一个Protobuf的RPC实例，其实你可以通过框架的Pack和Route模块自由扩展，提供这个例子无非是我自己项目的需要顺便就发出来了，当然你可以自己实现。

其他

很多细小的功能拉，去这看看吧
https://www.gitbook.com/book/tmtbe/swooledistributed。

入门指引

大致介绍下SwooleDistributed框架。

不要修改或者增加Server目录下代码，开发者应该操作的应该是app目录。

目录结构

```
├── src // 代码文件夹
│   ├── test // 这里是单元测试用例目录
│   ├── app // 这里是开发者应用项目
│   │   ├── Controllers  // Controllers目录
│   │   ├── Models  // Models目录
│   │   ├── Tasks  // Tasks目录
│   │   ├── Views  // Views目录
│   │   ├── Pack  // 自定义TCP解包类目录
│   │   ├── Route // 自定义路由类目录
│   │   └── AppServer.php // app服务器
│   │
│   ├── www // 这里是放置静态文件的目录
│   │
│   ├── config // config目录
│   │   ├── config.php  // 服务器配置
│   │   ├── businessConfig.php   // 业务的一些配置
│   │   ├── datebase.php        // 数据库配置
│   │   ├── redis.php        // redis配置
│   │   ├── fileHeader.php    // 文件扩展名与http头的对照表
│   │   └── timerTask.php        // 定时任务配置
│   │
│   └── Server // 框架目录
│       ├── Controllers  // Controllers目录
│       ├── Models  // Models目录
│       ├── Tasks  // Tasks目录
│       │   ├── UdpDispatchTask.php  // 支持集群自发现服务的定时任务
│       │   └── UnitTestTask.php  // 支持Task的单元测试任务
│       ├── Views  // Views目录
│       │   └── error_404.php  // http访问时404页面模板
```

```
|       ├── Pack  // 自定义TCP解包类目录
|       |   ├── IPack.php  // 自定义pack的接口
|       |   ├── SerializePack.php  // SerializePack
|       |   ├── MsgPack.php  // MsgPack，需要安装msgpack扩展
|       |   └── JsonPack.php  // JsonPack
|       ├── Route // 自定义路由类目录
|       |   ├── IRoute.php  // 自定义route的接口
|       |   └── NormalRoute.php  // 提供的默认的route方案
|       ├── CoreBase // 框架核心代码
|       |   ├── Child.php  // 基类
|       |   ├── InotifyProcess //自动reload类
|       |   ├── Controller.php  // Controller基类
|       |   ├── ControllerFactory.php  // Controller工厂
|       |   ├── CoreBase.php  // core
|       |   ├── HttpInput.php  // Http输入
|       |   ├── HttpOutput.php  // Http输出
|       |   ├── Loader.php  // Loader
|       |   ├── Model.php  // Model基类
|       |   ├── ModelFactory.php  // Model工厂
|       |   ├── SwooleException.php  // swoole异常
|       |   ├── Coroutine.php  // 调度器
|       |   ├── CoroutineTask.php  // 调度器任务
|       |   ├── CoroutineNull //空
|       |   ├── ICoroutineBase.php  // 调度器任务接口
|       |   ├── TaskCoroutine.php //任务的协程
|       |   ├── Task.php  // Task基类
|       |   ├── TaskProxy.php  // Task代理
|       |   └── XssClean.php  // xss clean
|       ├── Client  // Client目录，目前存有HttpClient
|       |   ├── Client.php  // Client
|       |   ├── GetHttpClientCoroutine.php  // 获取httpclient的协
程
|       |   ├── HttpClient.php  // HttpClient
|       |   └── HttpClientRequestCoroutine.php  //http请求的协程
|       ├── DataBase // 数据库核心代码
|       |   ├── AsynPool.php  // 通用异步连接池
|       |   ├── AsynPoolManager.php  // 异步连接池管理器
|       |   ├── IAsynPool.php  // 异步连接池的接口
|       |   ├── Miner.php  // mysql语法生成器
|       |   ├── MySqlCoroutine //mysql的协程
```

```
|       |   ├── RedisCoroutine //redis的协程
|       |   ├── MysqlAsynPool.php  // mysql异步连接池
|       |   └── RedisAsynPool.php  // Redis异步连接池
|       ├── Test // 单元测试框架
|       |   ├── DocParser.php  // 文档解析器
|       |   ├── SwooleTestException.php  // 测试用异常
|       |   ├── TestCase.php  // 测试用例基类
|       |   ├── TestHttpCoroutine.php  // 用于获取httpController请
求结果
|       |   ├── TestModule.php //单元测试组件
|       |   ├── TestRequest.php //httpRequest
|       |   ├── TestResponse.php  // httpResponse
|       |   └── TestTcpCoroutine.php  // 用于获取tcpController请求
结果
|       ├── helpers   // 帮助函数库
|       |   └── Common.php  // Common工具函数
|       ├── SwooleDispatchClient.php   // dispatch服务
|       ├── SwooleDistributedServer.php   // SwooleDistributed服
务器
|       ├── SwooleHttpServer.php   // SwooleHttp服务器（基类）
|       ├── SwooleWebSocketServer.php   // SwooleWebSocket服务器
（基类）
|       ├── SwooleServer.php   // Swoole服务器（基类）
|       └── SwooleMarco.php    // 全局定义
|
├── composer.json //composer依赖管理json
├── LICENSE //开源协议
├── start_swoole_server.php // swoole_server启动脚本
├── start_swoole_dispatch.php // swoole_dispatch启动脚本
└── start_swoole_client.php // 测试工具
```

说明

开发者只需要关注app目录的文件即可，server目录为框架目录，开发者不要改动，这样方便后续框架升级。

start_swoole_server.php start_swoole_dispatch.php 分别是进程启动脚本，开发者一般不需要改动这俩个脚本。

MVC

mvc架构，Controller，Moder，View分别为对应模块的基类，具体用法和注意事项参考对应的模块介绍。
框架优先匹配app目录下的类，找不到会取访问server文件夹。

Views 视图目录，如果没有启动会报错，自行添加这个目录就可以。

Route

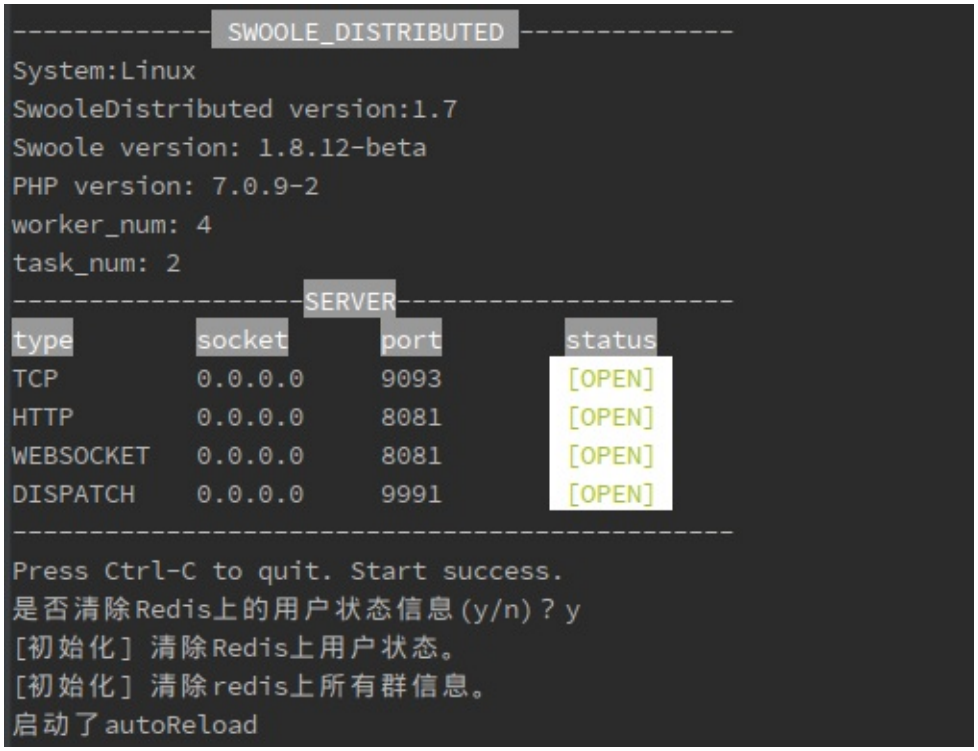
开发者可以自定义路由，需要实现IRoute接口，并在config中配置。
框架优先匹配app目录下的类，找不到会取访问server文件夹。
http访问的时候如果找不到最后会去匹配www目录下的静态文件，如果还没有则返回404。

Pack

开发者可以自定义压包解包，需要实现IPack接口，并在config中配置。
框架优先匹配app目录下的类，找不到会取访问server文件夹。

启动服务器

```
php start_swoole_server.php start
```



以下方法将以守护进程模式启动服务器

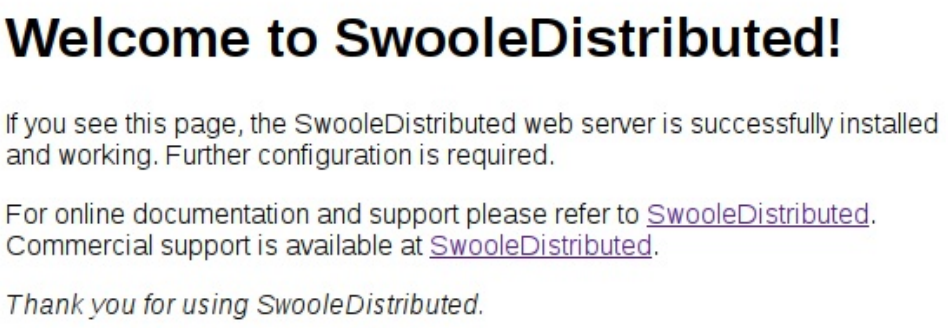
```
php start_swoole_server.php start -d
```

访问服务器

这里介绍http的访问。

主页：http://localhost:8081/

如果出现以下界面代表你的服务器搭建成功



路由规则

http://localhost:8081/TestController/test

以上代码会先在/app/Controllers目录下寻找TestController控制器，如果没有再去/Server/Controllers目录下寻找，如果依旧没有找到将返回404界面。

test是方法名，它将和businessConfig中的$config['http']['method*prefix']*合并成访问的方法名，默认前缀名为‘*http*’，可以通过修改配置自己设置。

所以以上url会访问到/Server/Controllers下的TestController控制器的http_test方法并输出helloworld。

静态文件的访问

你可以使用nignx配置静态目录，或者直接使用SD框架提供的静态目录，目录名为WWW。

http://localhost:8081/swd.jpg

在www文件夹中存放一张swd.jpg的文件就可以通过上述url访问了。

服务器启动命令

启动

调试模式启动服务器

```
php start_swoole_server.php start
```

守护进程启动服务器

```
php start_swoole_server.php start -d
```

重启

会自动结束进程然后重新启动一个守护进程模式的服务器

```
php start_swoole_server.php restart
```

重载

不会断开客户端链接，进行代码的重载。升级服务器逻辑，客户端无感知。

```
php start_swoole_server.php reload
```

停止

停止服务器

```
php start_swoole_server.php stop
```

单元测试

测试test目录下所有的测试类

```
php start_swoole_server.php test
```

测试test目录下指定的测试类

```
php start_swoole_server.php test XXXX
```

使用协程的优势

Swoole是高性能异步框架，正因为异步所以高性能，但是异步也有异步的不好，写逻辑代码有时候就非常的不方便，需要多层嵌套回调，弄得代码的可读性很差维护起来非常的不方便，那么如何解决这个弊端呢？那就是使用协程。

Swoole1.x没有提供协程模式，2.0版本开始有了协程版客户端但是不支持PHP7，并且还在开发测试阶段不算稳定。SwooleDistributed框架提供了一套基于yield关键字的协程写法。

回调风格和协程风格的区别

举例说明：

回调风格：

```
    /**
     * mysql 测试
     * @throws \Server\CoreBase\SwooleException
     */
    public function mysql_test()
    {
        $this->mysql_pool->dbQueryBuilder->select('*')->from('ac
count')->where('sex', 1);
        $this->mysql_pool->query(function ($result) {
            print_r($result);
        });
        $this->destroy();
    }
```

协程风格：

```
    /**
     * mysql 测试
     * @throws \Server\CoreBase\SwooleException
     */
    public function mysql_test()
    {
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('sex', 1)->coroutineSend();
        $result = yield $mySqlCoroutine;
        print_r($result);
        $this->destroy();
    }
```

上述代码还只是基础，想想看多次请求并且依赖的情况 回调风格：（仅仅2层还能忍）

```
    public function test($callback)
    {
        $this->redis_pool->get('test',function ($uid)use($callba
ck){
            $this->mysql_pool->dbQueryBuilder->select('*')->from
('account')->where('uid', $uid);
            $this->mysql_pool->query(function ($result)use($call
back) {
                call_user_func($callback,$result);
            });
        });
    }
```

协程风格：

```
    public function test()
    {
        $redisCoroutine = $this->redis_pool->getCoroutine()->get
('test');
        $uid = yield $redisCoroutine;
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', $uid)->coroutineSend();
        $result = yield $mySqlCoroutine;
        return $result;
    }
```

上面的代码是在一个model中，按照以往的回调风格，controller调用这个model还需要传进去一个回调函数才能获取到值（我觉得这么做的人一定会疯），而协程风格你只需要按部就班的return结果就行了。和同步的代码唯一的区别就是多了一个yield关键字。

好了我们再看调用这个model的controller怎么写。
回调风格：（去死吧，眞眞不想写）

```
    /**
     * 非协程测试（丧心病狂）
     */
    public function http_testNOCoroutine()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $this->testModel->test(function($result){
           $this->http_output->end($result);
        });
    }
```

协程风格：（小清新）

```
    /**
     * 协程测试
     */
    public function http_testCoroutine()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $result = yield $this->testModel->test();
        $this->http_output->end($result);
    }
```

同样只是要多加一个yield关键字。是不是很方便～，注意只有这个model的方法内使用了yield，控制器调用的时候才需要加yield关键字。普通的方法是不需要的，当然你加了也不会报错。

协程中的异常？

swooleDistributed框架已经将协程的使用变得很方便了，至于异常和平常的写法一毛一样。

```
    public function test_exceptionII()
    {
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();
        $result = yield $mySqlCoroutine;
        throw new \Exception('test');
    }
```

无论你是在model中还是在controller中，还是在model的model中。。。。
总之throw出来就行，上一层使用try可以捕获到错误。
这里还有个小小的体验优化,当报错一直到controller层的时候，会被controller的onExceptionHandle捕获到，你可以重写这个函数实现异常情况下的客户端回复。

协程的嵌套

随便嵌套。enjoy yourself。

协程的调度顺序

调节yield的位置即可调节接收的顺序。

```
    $redisCoroutine = $this->redis_pool->getCoroutine()->get('te
st');
    $uid = yield $redisCoroutine;
    $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->select(
'*')->from('account')->where('uid', $uid)->coroutineSend();
    $result = yield $mySqlCoroutine;
```

上面的代码调度顺序是redis_send->redis_rev->mysql_send->mysql_rev。

```
    $redisCoroutine = $this->redis_pool->getCoroutine()->get('te
st');
    $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->select(
'*')->from('account')->where(123, $uid)->coroutineSend();
    $uid = yield $redisCoroutine;
    $result = yield $mySqlCoroutine;
```

上面的代码调度顺序是redis_send->mysql_send->redis_rev->mysql_rev。

支持？

mysql，redis，httpclient,task，都支持协程，你还想要什么大声说出来。
其实swooleDistributed给协程都进行了标准化的封装，你很容易就能实现自己的协程。至于怎么做，看代码吧骚年。

通过协程的方法屏蔽异步同步的区别

同步

sd框架中Task就是一个典型的同步案例。task中不允许调用异步的api。

异步

sd框架中除了task基本都是异步的，异步最好使用异步api达到更高的效率。

如何选择

在1.6版本之前，task中如果使用mysql和redis都必须调用同步的客户端，而且同步客户端和异步客户端的调用方法以及返回结构都不一样，这样model和task的代码完全无法重用，也更不可能通过task去调用model。

福利

1.6版本后实现了框架进行异步和同步的选择，1.7版本对于这种新的方式进行了优化和完善。 通过协程的方式写的代码可以同时在modle和task中运行。也不用关心同步和异步的写法不同，他们的调用方式和返回值都保持绝对的一致。

```
$value = yield $this->redis_pool->getCoroutine()->setex('test',
10, 'testRedis');
```

比如这段代码在model和task中均能被正确的执行。

同时在task中也可以调用model的方法。

```
$testModel = $this->loader->model('TestModel', $this);
$result = yield $testModel->test_task();
```

至此难为程序员的异步和同步的区别现在被完美的解决了。

Mysql语法构建器

参考Miner。
github：https://github.com/jstayton/Miner

Getting Started

Composing SQL with Miner is very similar to writing it by hand, as much of the syntax maps directly to methods:

```
$Miner = new Miner();
$Miner->select('*')
      ->from('shows')
      ->innerJoin('episodes', 'show_id')
      ->where('shows.network_id', 12)
      ->orderBy('episodes.aired_on', Miner::ORDER_BY_DESC)
      ->limit(20);
```

Now that the statement is built,

```
$Miner->getStatement();
```

returns the full SQL string with placeholders (?), and

```
$Miner->getPlaceholderValues();
```

returns the array of placeholder values that can then be passed to your database connection or abstraction layer of choice. Or, if you'd prefer it all at once, you can get the SQL string with values already safely quoted:

```
$Miner->getStatement(false);
```

If you're using PDO, however, Miner makes executing the statement even easier:

```
$PDOStatement = $Miner->execute();
```

Miner works directly with your PDO connection, which can be passed during creation of the Miner object

```
$Miner = new Miner($PDO);
```

or after

```
$Miner->setPdoConnection($PDO);
```

Usage

SELECT

```
SELECT *
FROM shows
INNER JOIN episodes
  ON shows.show_id = episodes.show_id
WHERE shows.network_id = 12
ORDER BY episodes.aired_on DESC
LIMIT 20
```

With Miner:

```
$Miner->select('*')
      ->from('shows')
      ->innerJoin('episodes', 'show_id')
      ->where('shows.network_id', 12)
      ->orderBy('episodes.aired_on', Miner::ORDER_BY_DESC)
      ->limit(20);
```

INSERT

```
INSERT HIGH_PRIORITY shows
SET network_id = 13,
    name = 'Freaks & Geeks',
    air_day = 'Tuesday'
```

With Miner:

```
$Miner->insert('shows')
      ->option('HIGH_PRIORITY')
      ->set('network_id', 13)
      ->set('name', 'Freaks & Geeks')
      ->set('air_day', 'Tuesday');
```

REPLACE

```
REPLACE shows
SET network_id = 13,
    name = 'Freaks & Geeks',
    air_day = 'Monday'
```

With Miner:

```
$Miner->replace('shows')
      ->set('network_id', 13)
      ->set('name', 'Freaks & Geeks')
      ->set('air_day', 'Monday');
```

UPDATE

```
UPDATE episodes
SET aired_on = '2012-06-25'
WHERE show_id = 12
  OR (name = 'Girlfriends and Boyfriends'
        AND air_day != 'Monday')
```

With Miner:

```
$Miner->update('episodes')
      ->set('aired_on', '2012-06-25')
      ->where('show_id', 12)
      ->openWhere(Miner::LOGICAL_OR)
      ->where('name', 'Girlfriends and Boyfriends')
      ->where('air_day', 'Monday', Miner::NOT_EQUALS)
      ->closeWhere();
```

DELETE

```
DELETE
FROM shows
WHERE show_id IN (12, 15, 20)
LIMIT 3
```

With Miner:

```
$Miner->delete()
      ->from('shows')
      ->whereIn('show_id', array(12, 15, 20))
      ->limit(3);
```

Methods

- __construct

SELECT

- select
- getSelectString
- mergeSelectInto

INSERT

- insert
- getInsert
- getInsertString
- mergeInsertInto

REPLACE

- replace
- getReplace
- getReplaceString
- mergeReplaceInto

UPDATE

- update
- getUpdate
- getUpdateString
- mergeUpdateInto

DELETE

- delete
- getDeleteString
- mergeDeleteInto

OPTIONS

- option
- calcFoundRows
- distinct
- getOptionsString
- mergeOptionsInto

SET / VALUES

- set

- values
- getSetPlaceholderValues
- getSetString
- mergeSetInto

FROM

- from
- innerJoin
- leftJoin
- rightJoin
- join
- getFrom
- getFromAlias
- getFromString
- getJoinString
- mergeFromInto
- mergeJoinInto

WHERE

- where
- andWhere
- orWhere
- whereIn
- whereNotIn
- whereBetween
- whereNotBetween
- openWhere
- closeWhere
- getWherePlaceholderValues
- getWhereString
- mergeWhereInto

GROUP BY

- groupBy
- getGroupByString
- mergeGroupByInto

HAVING

- having
- andHaving
- orHaving
- havingIn
- havingNotIn
- havingBetween
- havingNotBetween
- openHaving
- closeHaving
- getHavingPlaceholderValues
- getHavingString
- mergeHavingInto

ORDER BY

- orderBy
- getOrderByString
- mergeOrderByInto

LIMIT

- limit
- getLimit
- getLimitOffset
- getLimitString

Statement

- execute
- getStatement

- getPlaceholderValues
- isSelect
- isInsert
- isReplace
- isUpdate
- isDelete
- __toString
- mergeInto

Connection

- setPdoConnection
- getPdoConnection
- setAutoQuote
- getAutoQuote
- autoQuote
- quote

单元测试

仿照PHPUnit提供一个简易的单元测试框架

TestCase

这是基类，请继承。

方法

test开头的public方法才能作为测试用例，其余将被忽略。

setUpBeforeClass与tearDownAfterClass

setUpBeforeClass() 与 tearDownAfterClass() 模板方法将分别在测试用例类的第一个测试运行之前和测试用例类的最后一个测试运行之后调用

setUp与tearDown

测试类的每个测试方法都会运行一次 setUp() 和 tearDown() 模板方法

coroutineRequestHttpController

启动一个http的模拟访问。

```
        $testRequest = new TestRequest('/TestController/test');
        $testResponse = yield $this->coroutineRequestHttpControl
ler($testRequest);
        $this->assertEquals($testResponse->data, 'helloworld');
```

TestRequest中可以设置请求的一些方法。 testResponse为返回的数据。其中data为返回的值，其余见类成员。

coroutineRequestTcpController

启动一个tcp的模拟访问

```
        if ($this->config['server']['pack_tool'] != 'JsonPack')
{
            $this->markTestSkipped('协议解包不是JsonPack');
        }
        $data = ['controller_name' => 'TestController', 'method_
name' => 'test', 'data' => 'helloWorld'];
        $reusult = yield $this->coroutineRequestTcpController($d
ata);
        $this->assertCount(2, $reusult);
```

$data传进去的是一个协议体

$result是返回的服务器具体操作步骤，详情可以自己打印。

特别注意这是模拟的方式，所以服务器不会产生任何的send操作，只是记录操作。

使用controller内提供的方法才能被记录，get_instance()的方法不会被记录，可能还会产生错误。

markTestSkipped

表示该测试被跳过。

@needTestTask

标注needTestTask

被标注的将会在测试的时候额外进行task同步测试。

@codeCoverageIgnore

标注codeCoverageIgnore

被标注的会在测试的时候被忽略

@depends

标注depends

对测试方法之间的显式依赖关系进行声明。

被标注的将产生依赖，和phpunit一样

```
    public function testEmpty()
    {
        $stack = [];
        $this->assertEmpty($stack);

        return $stack;
    }

    /**
     * @depends testEmpty
     */
    public function testPush(array $stack)
    {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);

        return $stack;
    }

    /**
     * @depends testPush
     */
    public function testPop(array $stack)
    {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
```

在上例中，第一个测试， testEmpty()，创建了一个新数组，并断言其为空。随后，此测试将此基境作为结果返回。第二个测试，testPush()，依赖于 testEmpty() ，并将所依赖的测试之结果作为参数传入。最后，testPop() 依赖于 testPush()。

@dataProvider

标注dataProvider

测试方法可以接受任意参数。这些参数由数据供给器方法提供。用 @dataProvider 标注来指定使用哪个数据供给器方法。

数据供给器方法必须声明为 public，其返回值要么是一个数组，其每个元素也是数组

例子：使用带有命名数据集的数据供给器

```
/**
 * @dataProvider additionProvider
 */
public function testAdd($a, $b, $expected)
{
    $this->assertEquals($expected, $a + $b);
}

public function additionProvider()
{
    return [
        'adding zeros'  => [0, 0, 0],
        'zero plus one' => [0, 1, 1],
        'one plus zero' => [1, 0, 1],
        'one plus one'  => [1, 1, 3]
    ];
}
```

如果测试同时从 @dataProvider 方法和一个或多个 @depends 测试接收数据，那么来自于数据供给器的参数将先于来自所依赖的测试的。来自于所依赖的测试的参数对于每个数据集都是一样的

例子: 在同一个测试中组合使用 @depends 和 @dataProvider

```
    public function provider()
    {
        return [['provider1'], ['provider2']];
    }

    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

    public function testProducerSecond()
    {
        $this->assertTrue(true);
        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     * @dataProvider provider
     */
    public function testConsumer()
    {
        $this->assertEquals(
            ['provider1', 'first', 'second'],
            func_get_args()
        );
    }
```

各种简易断言

assertEquals

assertEmpty

assertNotEmpty

.....

Config配置

config文件夹中的配置文件

config.php

服务器的配置

1. http_server http端口的设置，不填写代表不开启http监听

```
$config['http_server']['socket'] = '0.0.0.0';
$config['http_server']['port'] = 8080;
```

2.websocket的配置

```
/**
 * 是否启用websocket
 */
$config['websocket']['enable'] = true;
/*WEBSOCKET_OPCODE_TEXT = 0x1，UTF-8文本字符数据
WEBSOCKET_OPCODE_BINARY = 0x2，二进制数据*/
$config['websocket']['opcode'] = WEBSOCKET_OPCODE_BINARY;
```

1. server

```
$config['server']['socket'] = '0.0.0.0';
$config['server']['port'] = 9093;
$config['server']['dispatch_port'] = 9991;
$config['server']['name'] = 'SwooleServer';
$config['server']['send_use_task_num'] = 20;
$config['server']['log_path'] = '/../../';
$config['server']['log_max_files'] = 15;
$config['server']['log_level'] = \Monolog\Logger::DEBUG;
$config['server']['pack_tool'] = 'JsonPack';
$config['server']['route_tool'] = 'NormalRoute';
$config['server']['set'] = [
 'reactor_num' => 2, //reactor thread num
 'worker_num' => 4,    //worker process num
 'backlog' => 128,   //listen backlog
 'open_tcp_nodelay' => 1,
 'dispatch_mode' => 5,
 'task_worker_num' => 5,
 'enable_reuse_port' => true,
];
```

其中：
dispatch_port 是server连接dispatch的端口号。
send_use_task_num 是群发消息超过多少自动使用task方式发送
pack_tool 自定义pack的类名
route_tool 自定义route的类名
set 参考swoole的set

2. dispatch_server

```
$config['dispatch_server']['socket'] = '0.0.0.0';
$config['dispatch_server']['port'] = 60000;
$config['dispatch_server']['name'] = 'SwooleDispatch';
$config['dispatch_server']['password'] = 'Hello Dsipatch';
$config['dispatch_server']['set'] = [
 'reactor_num' => 2, //reactor thread num
 'worker_num' => 4,    //worker process num
 'backlog' => 128,   //listen backlog
 'open_tcp_nodelay' => 1,
 'dispatch_mode' => 3,
 'enable_reuse_port' => true,
];
//主从redis提高读的速度
//启动这个服务一定确保dispatch服务器上一定有一个redis只读服务器
$config['dispatch_server']['redis_slave'] = ['unix:/var/run/
redis/redis.sock',0];
//是否启动集群的支持
$config['use_dispatch'] = false;
```

其中：
port 是指发送udp广播的端口
password 是指dispatch和server的验证密码
set 参考swoole的set
redis_slave 当dispatch和redis只读实例在同一台物理机时可以使用unixsock提高效率

3. asyn_process_enable

```
//异步服务是否启动一个新进程（启动后异步效率会降低2倍，但维护连接池只有一个）
$config['asyn_process_enable'] = false;
```

默认为false，每个进程都会维护个连接池，这样会导致连接的浪费但是效率最高。
true将启动一个新的进程专门维护连接池，会有2次进程间通讯，效率只比同步方式高一点点。

4. use_dispatch
   默认为false工作在单机模式，如果需要集群的支持需要手动修改为true。

5. auto_reload_enable

```
//是否启用自动reload
$config['auto_reload_enable'] = true;
```

开启后src目录有更新会立即reload，需要安装inotify扩展

```
sudo pecl install inotify
```

database.php

设置数据库连接 其中： active 激活的mysql配置 asyn_max_count 是指异步连接池最多维护的连接数量

redis.php

设置redis连接 其中： active 激活的redis配置 asyn_max_count 是指异步连接池最多维护的连接数量

businessConfig.php

```
/**
 * tcp访问时方法的前缀
 */
$config['tcp']['method_prefix'] = '';
/**
 * http访问时方法的前缀
 */
$config['http']['method_prefix'] = 'http_';
/**
 * websocket访问时方法的前缀
 */
$config['websocket']['method_prefix'] = '';

//http服务器绑定的真实的域名或者ip:port，一定要填对,否则获取不到文件的绝对路径
$config['http']['domain'] = 'http://localhost:8081';

//默认访问的页面
$config['http']['index'] = 'index.html';

//是否服务器启动时自动清除群组信息
$config['autoClearGroup'] = true;
```

设置前缀后访问controller的方法会根据设置加上前缀名。 比如http：//localhost/Test/test，对应的方法是Test controller中的http_test。 如果没有http_test方法将会返回404页面。

timerTask.php

定时器任务的配置文件
值得一提的是，timerTask.php这个配置文件是支持热重载的，reload后会重新读取配置。

```
/**
 * timerTask定时任务
 * （选填）task名称 task_name
 * （选填）model名称 model_name  task或者model必须有一个优先匹配task
 * （必填）执行task的方法 method_name
 * （选填）执行区间 [start_time,end_time) 格式： Y-m-d H:i:s 没有代表
一直执行
 * （必填）执行间隔 interval_time 单位： 秒
 * （选填）最大执行次数 max_exec，默认不限次数
 * （选填）是否立即执行 delay，默认为false立即执行
 */
//dispatch发现广播，实现集群的实现
$config['timerTask'][] = [
    'task_name' => 'UdpDispatchTask',
    'method_name' => 'send',
    'interval_time' => '30'
];
```

task_name指的是Tasks中的类名。

model_name指的是Models中的类名。

其中代码中包含了2个定时器。

UdpDispatchTask：实现集群服务自发现，删除后自发现服务就不存在了。

TestTask：测试，可以删除。

LVS+Keepalive 比较详细的安装配置文档

LVS说明:

目前有三种IP负载均衡技术（VS/NAT、VS/TUN和VS/DR,八种调度算法（rr,wrr,lc,wlc,lblc,lblcr,dh,sh）。

在 调度器的实现技术中，IP负载均衡技术是效率最高的。在已有的IP负载均衡技术中有通过网络地址转换（Network Address Translation）将一组服务器构成一个高性能的、高可用的虚拟服务器，我们称之为VS/NAT技术（Virtual Server via Network Address Translation），大多数商品化的IP负载均衡调度器产品都是使用此方法，如Cisco的LocalDirector、F5的Big/IP和 Alteon的ACEDirector。在分析VS/NAT的缺点和网络服务的非对称性的基础上，我们提出通过IP隧道实现虚拟服务器的方法VS/TUN （Virtual Server via IP Tunneling），和通过直接路由实现虚拟服务器的方法VS/DR（Virtual Server via Direct Routing），它们可以极大地提高系统的伸缩性。所以，IPVS软件实现了这三种IP负载均衡技术，它们的大致原理如下（我们将在其他章节对其工作原 理进行详细描述）

Virtual Server via Network Address Translation（VS/NAT）

通过网络地址转换，调度器重写请求报文的目标地址，根据预设的调度算法，将请求分派给后端的眞实服务器；眞实服务器的响应报文通过调度器时，报文的源地址被重写，再返回给客户，完成整个负载调度过程。

Virtual Server via IP Tunneling（VS/TUN）

采用NAT技术时，由于请求和响应报文都必须经过调 度器地址重写，当客户请求越来越多时，调度器的处理能力将成为瓶颈。为了解决这个问题，调度器把请求报文通过IP隧道转发至眞实服务器，而眞实服务器将响 应直接返回给客户，所以调度器只处理请求报文。由于一般网络服务应答比请求报文大许多，采用 VS/TUN技术后，集群系统的最大吞吐量可以提高10倍。

Virtual Server via Direct Routing（VS/DR）

VS/DR通过改写请求报文的MAC地址，将请求 发送到眞实服务器，而眞实服务器将响应直接返回给客户。同VS/TUN技术一样，VS/DR技术可极大地提高集群系统的伸缩性。这种方法没有IP隧道的开 销，对集群中的眞实服务器也没有必须支持IP隧道协议的要求，但是要求调度器与眞实服务器都有一块网卡连在同一物理网段上。

针对不同的网络服务需求和服务器配置，IPVS调度器实现了如下八种负载调度算法：使用比较多的是以下四种：

轮叫（Round Robin）

调度器通过"轮叫"调度算法将外部请求按顺序轮流分配到集群中的眞实服务器上，它均等地对待每一台服务器，而不管服务器上实际的连接数和系统负载。

加权轮叫（Weighted Round Robin）

调度器通过"加权轮叫"调度算法根据眞实服务器的不同处理能力来调度访问请求。这样可以保证处理能力强的服务器处理更多的访问流量。调度器可以自动问询眞实服务器的负载情况，并动态地调整其权值。

最少链接（Least Connections）

调度器通过"最少连接"调度算法动态地将网络请求调度到已建立的链接数最少的服务器上。如果集群系统的眞实服务器具有相近的系统性能，采用"最小连接"调度算法可以较好地均衡负载。

加权最少链接（Weighted Least Connections）

在集群系统中的服务器性能差异较大的情况下，调度器采用"加权最少链接"调度算法优化负载均衡性能，具有较高权值的服务器将承受较大比例的活动连接负载。调度器可以自动问询眞实服务器的负载情况，并动态地调整其权值。

1、拓扑描述：(一定要理解这个拓扑关系） 负载服务器master眞实IP 192.168.1.252 负载服务器backup眞实IP 192.168.1.230 负载服务器虚拟IP 192.168.1.229 后端WEB服务器IP 192.168.1.220 后端WEB服务器IP 192.168.1.231 2、升级内核

```
#yum install kernel
```

3、重启服务器，使用新的内核 4、删除旧版、升级新版内核

```
#rpm -e kernel-2.6.18-53.el5
#rpm -e kernel-devel-2.6.18-53.el5
#rpm -e kernel-headers-2.6.18-53.el5 --nodeps
#yum install kernel-headers
#yum install kernel-devel
```

5、下载软件

```
#wget http://www.linuxvirtualser... ... ipvsadm-1.24.tar.gz
#wget http://www.keepalived.org/...
```

6、安装ipvsadm-1.24 //master和backup

```
#        rpm -ivh ipvsadm-1.24-6.src.rpm
#        cd /usr/src/redhat/SOURCES
#        tar -zxvf ipvsadm-1.24.tar.gz
#        cd ipvsadm
#       uname -r  //查询版本
       2.6.18-53.el5xen
#        ln -s  /usr/src/kernels/2.6.18-53.el5xen-i686/  /usr/sr
c/linux  //假如这里的内核版本不一样的话，make的时候会出现错误。
#        make;make install
```

7,安装keepalived. 在负载均衡服务器上执行 master和backup 1、解压

```
        #tar -zxvf keepalived-1.1.15.tar.gz
        #cd keepalived-1.1.15
        #./configure --prefix=/usr/local/keepalived
        #make;make install
        #cp /usr/local/keepalived/sbin/rc.d/init.d/keepalived /sb
in/rc.d/init.d/
        #cp /usr/local/keepalived/sbin/sysconfig/keepalived /sbin
/sysconfig/
        #mkdir /sbin/keepalived
        #cp /usr/local/keepalived/sbin/keepalived/keepalived.conf
 /sbin/keepalived/
        #cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
        #service keepalived start|stop
```

8、开启负载服务器路由机制 //master和backup

```
#         vi /sbin/sysctl.conf 保证有如下内容
        net.ipv4.ip_forward = 1
        执行
#         sysctl -p
```

9、建立负载服务器启动脚本 //master和backup

```
#vi /sbin/lvsdr.sh
#!/bin/bash
VIP=192.168.1.229
RIP1=192.168.1.220
RIP2=192.168.1.231
/sbin/rc.d/init.d/functions
case "$1" in
start)
       echo "start LVS of DirectorServer"
       #Set the Virtual IP Address
       /sbin/ifconfig eth0:1 $VIP broadcast $VIP netmask 255.255
.255.255 up
       /sbin/route add -host $VIP dev eth0:1
       #Clear IPVS Table
       /sbin/ipvsadm -C
       #Set Lvs
       /sbin/ipvsadm -A -t $VIP:80 -s wrr
       /sbin/ipvsadm -a -t $VIP:80 -r $RIP1:80 -g
       /sbin/ipvsadm -a -t $VIP:80 -r $RIP2:80 -g
       #Run Lvs
       /sbin/ipvsadm
;;
stop)
echo "Close LVS Directorserver"
/sbin/ifconfig eth0:1 down
/sbin/ipvsadm -C
;;
*)
echo "Usage0{start|stop}"
exit 1
esac
```

10、分配权限

```
#chmod 755 /sbin/lvsdr.sh
```

11、执行测试

```
#        /sbin/lvsdr.sh start
       查看ifconfig是否有ifcfg-eth0:1   (有就对了)
       查看route -n 路由表是否多了eth0:1路由(有就对了)
#        /sbin/lvsdr.sh stop
       查看ifconfig是否有ifcfg-eth0:1   (无就对了)
       查看route -n 路由表是否多了eth0:1路由(无就对了)
#        /sbin/lvsdr.sh  adsa
       是否提示参数错误,只能使用{start|stop}。
```

12、配置后端WEB服务器

```
       在192.168.1.231和192.168.1.220上分别建立如下脚本。
#        vi /sbin/realdr.sh
       #!/bin/bash
       VIP=192.168.1.229
       /sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.2
55.255 up
       /sbin/route add -host $VIP dev lo:0
       echo "1">/proc/sys/net/ipv4/conf/default/arp_ignore
       echo "2">/proc/sys/net/ipv4/conf/default/arp_announce
       echo "1">/proc/sys/net/ipv4/conf/all/arp_ignore
       echo "2">/proc/sys/net/ipv4/conf/all/arp_announce
       sysctl -p
```

13、配置权限

```
#        chmod 755 /sbin/realdr.sh
```

14、在两台web服务器上分别执行其指命。

```
/sbin/realdr.sh start
```

15.配置keepalived.conf配置文件 //master和backup

```
#vi /etc/keepalived/keepalived.conf
! Configuration File for keepalived
```

```
global_defs {
  notification_email {
    dalianlxw@139.com
  }
  notification_email_from xwluan@tsong.cn
  smtp_server 222.73.214.147
  smtp_connect_timeout 30
  router_id LVS_DEVEL
}
vrrp_instance VI_1 {
   state MASTER  //备份服务器设置为backup
   interface eth0
   virtual_router_id 51
   priority 100  ////备份服务器设置小于100
   advert_int 1
   authentication {
       auth_type PASS
       auth_pass 1111
   }
   virtual_ipaddress {
       192.168.1.229
   }
}
virtual_server 192.168.1.229 80 {
   delay_loop 6  //隔6秒查询
   lb_algo wrr   //lvs算法
   lb_kind DR    //(Direct Route)
   persistence_timeout 60  //同一IP的连接60秒内被分配到同一台realserv
er
   inhibit_on_failure  //当web挂掉的时候，前面请求的用户，可以继续打开
网页，但是后面的请求不会调度到挂掉的web上面。
   protocol TCP   //用TCP协议检查realserver状态
   real_server 192.168.1.220 80 {
       weight 3    //权重
       TCP_CHECK {
           connect_timeout 10   //10秒无响应超时
           nb_get_retry 3
           delay_before_retry 3
       }
   }
```

```
    real_server 192.168.1.231 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 10
            nb_get_retry 3
            delay_before_retry 3
        }
    }

}
```

16,启动keepalived

```
# /etc/rc.d/init.d/keepalived start
```

17:设置成自启动

```
#vi /etc/rc.local //里面添加
/etc/init.d/keepalived restart
/etc/lvsdr.sh start
```

测试算法：

我的测试环境中，算法使用的是wrr,和wlc这两种。

权重问题：

当lvs配置文件lvs-dr.sh改变权重以及keepalived配置文件keepalived.conf修改权重后，哪个文件重新启动，哪个文件的权重生效。同时权重在master和backup上面可以设置不同。

当 算法是rr的时候，权重没有作用，但是当算法是wlc和wrr的时候，必须设置权重，可以根据服务器的性能和配置，来确定权重的大小，当权重大的时 候，lvs调度的服务也就多，同时权重高的服务器先收到链接。当小的时候，lvs调度的比较少。当权重为0的时候，表示服务器不可用，

测试LVS

1 当我把master的lvs服务关掉的时候，会将用户请求自动切换到backup上面进行工作。

2 当我把web服务关掉的时候，lvs上面会显示web消失，当启用后，web会自动显示 web关闭后：

```
[root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddressort Scheduler Flags
 -> RemoteAddressort           Forward Weight ActiveConn InActCo
nn
TCP  192.168.1.229:80 wlc
 -> 192.168.1.231:80             Route   10     0          0

[root@localhost ~]# ipvsadm -ln
```

web启用后：

```
root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddressort Scheduler Flags
 -> RemoteAddress:Port           Forward Weight ActiveConn InAct
Conn
TCP  192.168.1.229:80 wlc
 -> 192.168.1.231:80             Route   1      0          0

 -> 192.168.1.220:80             Route   1      0          0
```

3 当master服务器down的时候，backup自动会接替服务，当master起来的时候，backup会自动断掉。

AppServer

这个类是继承SwooleDistributedServer，提供给开发者维护的。

clearState

开服前的清理操作，只支持同步的写法。

onOpenServiceInitialization

支持协程

开服初始化的回调，开发者可在这里写开服的处理，这里保证无论多少进程只执行一次。

onUidCloseClear($uid)

支持协程

当一个绑定uid的连接close后的清理

SwooleDistributedServer

方法均是支持集群的。

- bindUid

```
    /**
     * 将fd绑定到uid,uid不能为0
     * @param $fd
     * @param $uid
     */
    function bindUid($fd, $uid)
```

- unBindUid

```
    /**
     * 解绑uid，链接断开自动解绑
     * @param $uid
     */
    function unBindUid($uid)
```

- uidIsOnline

```
    /**
     * uid是否在线(异步时候需要提供callback,task可以直接返回结果)
     * @param $uid
     * @param $callback
     * @return bool
     */
    function uidIsOnline($uid, $callback)
```

- countOnline

```
    /**
     * 获取在线人数(异步时候需要提供callback,task可以直接返回结果)
     * @param $callback
     * @return int
     */
    function countOnline($callback)
```

- addToGroup

```
    /**
     * 添加到群
     * @param $uid int
     * @param $group_id int
     */
    function addToGroup($uid, $group_id)
```

- removeFromGroup

```
    /**
     * 从群里移除
     * @param $uid
     * @param $group_id
     */
    function removeFromGroup($uid, $group_id)
```

- delGroup

```
    /**
     * 删除群
     * @param $group_id
     */
    function delGroup($group_id)
```

- sendToUid

```
    /**
     * 向uid发送消息
     * @param $uid
     * @param $data
     * @param $fromDispatch
     */
    function sendToUid($uid, $data, $fromDispatch = false)
```

- sendToAll

```
    /**
     * 广播
     * @param $data
     */
    function sendToAll($data)
```

- sendToGroup

```
    /**
     * 发送给群
     * @param $groupId
     * @param $data
     */
    function sendToGroup($groupId, $data)
```

- sendToUids

```
    /**
     * 批量发送消息
     * @param $uids
     * @param $data
     * @param $fromDispatch
     */
    function sendToUids($uids, $data, $fromDispatch = false)
```

- getRedis Task中使用的同步redis

```
    /**
     * 获取同步redis
     * @return \Redis
     * @throws SwooleException
     */
    function getRedis()
```

- onUidCloseClear 当绑定了一个uid的客户端下线时候的回调，用于清理数据，这个回调里支持协程。
- setTemplateEngine 用于设置模板引擎的一些方法
- get_instance

```
    /**
     * 获取实例
     * @return SwooleDistributedServer
     */
    static function &get_instance()
```

Loader

加载器

- model

```
/**
 * 获取一个model
 * @param $model string
 * @param $parent CoreBase
 */
function model($model, $parent)
```

- task

```
/**
 * 获取一个task
 * @param $task
 * @return TaskProxy
 */
function task($task)
```

- view

```
/**
 * view 返回一个模板
 * @param $template
 * @return \League\Plates\Template\Template
 */
public function view($template)
```

示例：

```
  /**
   * html测试
   */
  public function http_html_test()
  {
      $template = $this->loader->view('server::error_404');
      $this->http_output->end($template->render(['controller
'=>'TestController\html_test','message'=>'页面不存在！']));
  }
```

> server::error_404 是指Server/Views下的error_404.php文件
> app::error_404 是指app/Views下的error_404.php文件

Controller

MVC架构中的'C'。

- $request_type 请求的类型是TCP还是HTTP。

```
SwooleMarco::TCP_REQUEST;
SwooleMarco::HTTP_REQUEST;
```

- $fd 当前客户端的fd，TCP模式下才有值
- $uid 当前客户端绑定的uid，TCP模式下才有值
- $client_data TCP连接客户端传来的消息，通过Pack处理后的结构体
- $request HTTP对应swoole的swoole_http_request
- $response HTTP对应swoole的swoole_http_response
- $http_input HttpInPut类
- $http_output HttpOutPut类
- $redis_pool RedisAsynPool类
- $mysql_pool MysqlAsynPool类
- $client 用于获取异步http_client
- $loader Loader类用于加载model，task，view
- $logger 日志Logger类
- $pack TCP IPack自定义解包压包类
- $config Config 配置类
- $server swoole_server
- destroy 销毁方法。对象池模式，当客户端请求处理完毕后调用destory，回收controller

- send

```
/**
   * 向当前客户端发送消息
   * @param $data
   * @param $distory
   * @throws SwooleException
   */
function send($data, $distory = true)
```

- sendToUid

```
/**
   * sendToUid
   * @param $uid
   * @param $data
   * @throws SwooleException
*/
function sendToUid($uid, $data, $distory = true)
```

- sendToUids

```
/**
   * sendToUids
   * @param $uids
   * @param $data
   * @param $distory
   * @throws SwooleException
*/
function sendToUids($uids, $data, $distory = true)
```

- sendToAll

```
/**
   * sendToAll服务器广播
   * @param $data
   * @param $distory
   * @throws SwooleException
*/
function sendToAll($data, $distory = true)
```

- sendToGroup

```
/**
   * 发送给群
   * @param $groupId
   * @param $data
   * @param bool $distory
   * @throws SwooleException
*/
function sendToGroup($groupId, $data, $distory = true)
```

- kickUid

```
/**
     * 踢用户
     * @param $uid
*/
function kickUid($uid)
```

- bindUid

```
/**
   * bindUid
   * @param $fd
   * @param $uid
   * @param bool $isKick
*/
function bindUid($fd, $uid, $isKick = true)
```

*unBindUid

```
/**
    * unBindUid
    * @param $uid
*/
function unBindUid($uid)
```

> 注意事项

1. controller是对象池模式，所以避免内存泄露确保处理完客户端请求后执行destroy，并且重写destroy方法进行上下文的数据清理。
2. model和task通过loader对应的方法进行导入。

   ```
   /**
    * @var AppModel
    */
   public $AppModel;

   /**
    * http测试
    */
   public function http_test()
   {
       $this->AppModel = $this->loader->model('AppModel', $thi
   s);
       $this->http_output->end($this->AppModel->test());
   }
   ```

   通过loader方法导入的model不需要主动调用model的destroy方法，在controller触发destroy时会自动销毁model。

3. Task在controller中其实是TaskProxy，这是一个Task的代理，并且是单例的，不要试图去缓存，正确的用法是随用随loader

```
public function http_test_task()
{
    $AppTask = $this->loader->task('AppTask');
    $AppTask->testTask();
    $AppTask->startTask(function ($serv, $task_id, $data) {
        $this->http_output->end($data);
    });
}
```

4. 使用get_instance()可以調用SwooleDistributedServer中的方法

```
/**
 * 绑定uid
 */
public function bind_uid()
{
    get_instance()->bindUid($this->fd, $this->client_data->
data);
    $this->destroy();
}
```

协程模式

model中可以调用redis，mysql，task的协程模式，但是请注意，如果model是使用协程的，那么controller或者model调用这个包含协程model接口时，也要加上yeild关键字。 例子：

```
    public function http_testCoroutine()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $result = yield $this->testModel->test_coroutine();
        $this->http_output->end($result);
    }
```

异常捕获的回调

```
//Controller.php
/**
     * 异常的回调
     * @param \Exception $e
     */
    public function onExceptionHandle(\Exception $e){
       switch ($this->request_type)
       {
           case SwooleMarco::HTTP_REQUEST:
               $this->http_output->end($e->getMessage());
               break;
           case SwooleMarco::TCP_REQUEST:
               $this->send($e->getMessage());
               break;
       }
    }
```

在controller代码中的throw都会触发这个回调。方便大家继承Controller重写onExceptionHandle，实现统一的异常客户端回复。

由于协程的存在，异常嵌套处理比较麻烦，下面将其情况一一说明。
1.普通的controller异常测试，onExceptionHandle将接收到回调

```
    /**
     * 普通的controller异常测试
     * @throws \Exception
     */
    public function http_testExceptionHandleII()
    {
        throw new \Exception('ExceptionHandle');
    }
```

2.普通model内的异常测试,onExceptionHandle将接收到回调

```
    /**
     * 普通model的异常测试
     * @throws \Exception
     */
    public function http_testExceptionHandleIII()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $result = $this->testModel->test_exception();
    }
```

3.协程的model内的异常测试,onExceptionHandle将接收到回调

```
    /**
     * 协程的model报错需要增加try catch捕获，否则无法抛出
     */
    public function http_testExceptionHandleIV()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is)；
        $result = yield $this->testModel->test_exceptionII();
    }
```

4.总结：无论协程或者是多层协程嵌套，还是普通的代码，或者普通代码混合协程，都可以一路向上throw。并且可以通过try catch捕获。和正常代码没有任何区别。

initialization

这个方法会在调用其他方法前每次都调用。这样方便做统一的处理。

Model

Model是专门和数据打交道的模块。 同样是支持异步的。

- $redis_pool redis异步连接池
- $mysql_pool mysql异步连接池
- $client 用于获取异步http_client

例子：

```
class AppModel extends Model
{
    public function test(){
        return 123456;
    }
}
class AppController extends Controller
{
    /**
     * @var AppModel
     */
    public $AppModel;
    /**
     * http测试
     */
    public function http_test()
    {
        $this->AppModel = $this->loader->model('AppModel', $this
);
        $this->http_output->end($this->AppModel->test());
    }
}
```

协程模式

model中可以调用redis，mysql，task的协程模式，但是请注意，如果model是使用协程的，那么controller或者model调用这个包含协程model接口时，也要加上yeild关键字。 例子：

```
    public function http_testCoroutine()
    {
        $this->testModel = $this->loader->model('TestModel', $th
is);
        $result = yield $this->testModel->test_coroutine();
        $this->http_output->end($result);
    }
```

View

视图，适用于http

模板引擎参考plates

Task

Task是异步任务模块服务于controller和model

Task

> Task是同步阻塞操作，所以不能使用异步redis和异步mysql

- sendToUid

```
 /**
 * sendToUid
 * @param $uid
 * @param $data
 */
function sendToUid($uid,$data)
```

- sendToUids

```
 /**
 * sendToUids
 * @param $uids
 * @param $data
 */
function sendToUids($uids,$data)
```

- sendToAll

```
 /**
 * sendToAll
 * @param $data
 */
function sendToAll($data)
```

- getRedis

```
 /**
 * 获取同步redis
 * @return \Redis
 * @throws SwooleException
 */
function getRedis()
```

- getMysql

```
/**
 * 获取同步mysql
 * @return \Server\DataBase\Miner
 */
function getMysql()
```

TaskProxy

> controller和model中获取的Task其实是TaskProxy

- startTask

```
 /**
 * 开始异步任务
 */
function startTask($callback,$id = -1)
```

- startTaskWait

```
 /**
 * 开始同步任务
 */
function startTaskWait($timeOurt = 0.5, $id = -1)
```

Controller和Model中的用法

AppTask.php

```
    class AppTask extends Task
    {
        public function testTask()
        {
            return "test task\n";
        }

    }
```

AppController.php

```
    class AppController extends Controller
    {
        public function http_test_task()
        {
            $AppTask = $this->loader->task('AppTask');
            $AppTask->testTask();
            $AppTask->startTask(function ($serv, $task_id, $data
) {
                $this->http_output->end($data);
            });
        }
    }
```

TimerTask的用法

添加到配置文件timerTask.php中

```
  $config['timerTask'][] = [
      'task_name'=>'AppTask',
      'method_name'=>'testTask',
      'start_time'=>'Y-m-d 00:00:00',
      'end_time'=>'Y-m-d 23:59:59',
      'interval_time'=>'2',
  ];
```

协程模式(只对异步)

示例：

```
        $task = $this->loader->task('TestTask');
        $task->test();
        $result = yield $task->coroutineSend();
        $this->http_output->end($result);
```

使用协程会大大简化代码的书写，提高代码的可读性。 上面的代码通过yield关键字返回了异步回调的值。

HttpInput

http输入

- $request
  对应swoole_http_request
- post_get

```
 /**
 * post_get
 * @param $index
 * @param $xss_clean
 * @return string
 */
function post_get($index, $xss_clean = true)
```

- get_post

```
/**
 * get_post
 * @param $index
 * @param $xss_clean
 * @return string
 */
function get_post($index, $xss_clean = true)
```

- getAllPostGet

```
/**
 * 获取所有的post和get
 */
function getAllPostGet()
```

- getAllHeader

```
/**
 * getAllHeader
 * @return array
 */
function getAllHeader()
```

- post

```
/**
 * post
 * @param $index
 * @param $xss_clean
 * @return string
 */
function post($index, $xss_clean = true)
```

- get

```
/**
 * get
 * @param $index
 * @param $xss_clean
 * @return string
 */
function get($index, $xss_clean = true)
```

- get_rawContent

```
 /**
 * 获取原始的POST包体
 * @return mixed
 */
function get_rawContent()
```

- cookie

```
    /**
     * cookie
     * @param $index
     * @param $xss_clean
     * @return string
     */
    function cookie($index, $xss_clean = true)
```

- get_request_header

```
    /**
     * get_request_header
     * @param $index
     * @param $xss_clean
     * @return string
     */
    function get_request_header($index, $xss_clean = true)
```

HttpOutput

http输出

- $response
  对应swoole_http_response
- set_status_header

```
/**
 * Set HTTP Status Header
 *
 * @param    int    the status code
 * @param    string
 * @return HttpOutPut
 */
function set_status_header($code = 200)
```

- set_header

```
/**
 * set_header
 * @param $key
 * @param $value
 * @return $this
 */
function set_header($key, $value)
```

- set_content_type

```
    /**
     * Set Content-Type Header
     *
     * @param    string $mime_type Extension of the file we're
  outputting
     * @return    HttpOutPut
     */
    function set_content_type($mime_type)
```

- end

```
    /**
     * 发送
     * @param string $output
     * @param bool $gzip
     * @param bool $destory
     */
    function end($output = '',$gzip=true,$destory = true)
```

- endFile

```
    /**
     * 输出文件（会自动销毁）
     * @param $root_file
     * @param $file_name
     * @return bool
     */
    public function endFile($root_file,$file_name){
```

例子：

SERVER_DIR：对应Server目录

APP_DIR：对应app目录

```
  /**
   * html测试
   */
  public function http_html_file_test()
  {
     $this->http_output->endFile(SERVER_DIR,'Views/test.html
');
  }
```

RedisAsynPool

异步redis连接池

示例：

```
$this->redis_pool->hMGet('test',[1,2,3,4], function ($uids) {
});
```

协程模式

```
        $this->redis_pool->getCoroutine();
```

这个函数在注释中表示返回的是\Redis扩展的一个抽象类其实返回的是一个迭代器，我们只需要他的代码提示，1.7版本已经将同步和异步的使用方式做了完全的统一，和\Redis扩展的用法完全一致。

示例：

```
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();
        $mysql_result = yield $mySqlCoroutine;
        $redisCoroutine = $this->redis_pool->getCoroutine()->get
('test');
        $redis_result = yield $redisCoroutine;
```

使用协程会大大简化代码的书写，提高代码的可读性。 上面的代码通过yield关键字返回了异步回调的值。 执行顺序 mysql_send->mysql_rev->redis_send->redis_rev;

```
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();

        $redisCoroutine = $this->redis_pool->getCoroutine()->get
('test');
        $mysql_result = yield $mySqlCoroutine;
        $redis_result = yield $redisCoroutine;
```

执行顺序 mysql_send->redis_send->mysql_rev->redis_rev;

MysqlAsynPool

异步mysql连接池

普通用法

示例：

```
    /**
     * mysql 测试
     * @throws \Server\CoreBase\SwooleException
     */
    public function mysql_test()
    {
        $this->mysql_pool->dbQueryBuilder->select('*')->from('ac
count')->where('sex', 1);
        $this->mysql_pool->query(function ($result) {
            print_r($result);
        });
        $this->destroy();
    }
```

其中dbQueryBuilder对应的是Miner，关于用法参考Miner

协程事务

非协程版事务相对复杂不建议使用。

```
    /**
     * mysql 事务协程测试
     */
    public function http_mysql_begin_coroutine_test()
    {
        $id = yield $this->mysql_pool->coroutineBegin($this);
        $update_result = yield $this->mysql_pool->dbQueryBuilder
->update('user_info')->set('sex', '0')->where('uid', 36)->corout
ineSend($id);
        $result = yield $this->mysql_pool->dbQueryBuilder->selec
t('*')->from('user_info')->where('uid', 36)->coroutineSend($id);
        if ($result['result'][0]['channel'] == 888) {
            $this->http_output->end('commit');
            yield $this->mysql_pool->coroutineCommit($id);
        } else {
            $this->http_output->end('rollback');
            yield $this->mysql_pool->coroutineRollback($id);
        }
    }
```

协程模式

示例：

```
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();
        $mysql_result = yield $mySqlCoroutine;
        $redisCoroutine = $this->redis_pool->coroutineSend('get'
, 'test');
        $redis_result = yield $redisCoroutine;
```

使用协程会大大简化代码的书写，提高代码的可读性。上面的代码通过yield关键字返回了异步回调的值。执行顺序 mysql_send->mysql_rev->redis_send->redis_rev;

```
        $mySqlCoroutine = $this->mysql_pool->dbQueryBuilder->sel
ect('*')->from('account')->where('uid', 10303)->coroutineSend();

        $redisCoroutine = $this->redis_pool->coroutineSend('get'
, 'test');
        $mysql_result = yield $mySqlCoroutine;
        $redis_result = yield $redisCoroutine;
```

执行顺序 mysql_send->redis_send->mysql_rev->redis_rev;

Client

目前仅仅用作获取一个HttpClient

- getHttpClient

```
/**
 * 获取一个http客户端
 * @param $base_url
 * @param $callBack
 */
public function getHttpClient($base_url,$callBack)
```

使用协程风格

```
   $httpClient = yield $this->client->coroutineGetHttpClient('ht
tp://localhost:8081');
```

HttpClient

异步http客户端

- 设置头

```
/**
 * @param $headers
 */
public function setHeaders($headers)
```

- 设置cookies

```
/**
 * @param $cookies
 */
public function setCookies($$cookies)
```

- get请求

```
/**
 * @param $path
 * @param $query
 * @param $callback
 */
public function get($path, $query, $callback)
```

- post请求

```
/**
 * @param $path
 * @param $data
 * @param $callback
 */
public function post($path, $data, $callback)
```

使用协程风格

```
  //协程方式获取一个http客户端
  $httpClient = yield $this->client->coroutineGetHttpClient('htt
p://localhost:8081');
  //协程方式发送一个Get请求
  $result_get = yield $httpClient->coroutineGet("/TestController
/test_request",['id'=>123]);
  //协程方式发送一个Post请求
  $result_post = yield $httpClient->coroutinePost("/TestControll
er/test_request",['id'=>123]);
```