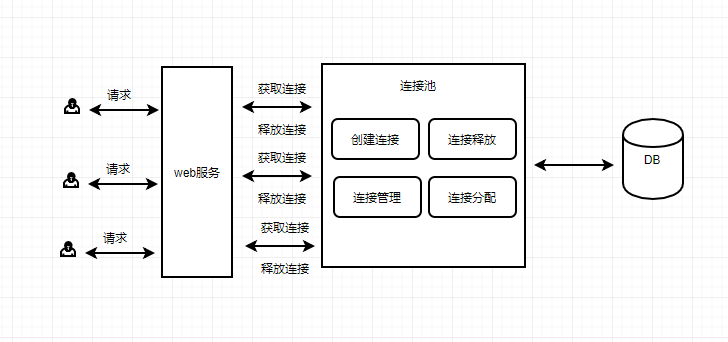
在写这篇文章之前，看了好几篇实现连接池的文章，都是写的很不好的。摆明忽略了连接池的很多特性，很多都不具有抗高并发和连接复用。所以自己觉得有必须把最近几天，实现一个比较完整的php数据库连接池的点滴记录下来，望能帮助各位，感激者望多点赞和打赏。

**一、数据库连接池基本概念**

所谓的数据库连接池，一般指的就是程序和数据库保持一定数量的数据库连接不断开，并且各请求的连接可以相互复用，减少重复新建数据库连接的消耗和避免在高并发的情况下出现数据库max connections等错误。自己总结了一下，如果要实现一个数据库连接池，一般有几个特点：

* 连接复用，不同的请求连接，可以放回池中，等待下个请求发分配和调用
* 连接数量一般维持min-max的最大最少值之间
* 对于空闲连接的回收
* 可以抗一定程度的高并发，也就是说当一次并发请求完池中所有的连接时，获取不到连接的请求可等待其他连接的释放

总结几个特性后，一个基本连接池，大致要实现下图功能：



1. 创建连接：连接池启动后，初始化一定的空闲连接，指定为最少的连接min。当连接池为空，不够用时，创建新的连接放到池里，但不能超过指定的最大连接max数量。
2. 连接释放：每次使用完连接，一定要调用释放方法，把连接放回池中，给其他程序或请求使用。
3. 连接分配：连接池中用pop和push的方式对等入队和出队分配与回收。能实现阻塞分配，也就是在池空并且已创建数量大于max，阻塞一定时间等待其他请求的连接释放，超时则返回null。
4. 连接管理：对连接池中的连接，定时检活和释放空闲连接等

**二、Fpm+数据库长连接的实现**

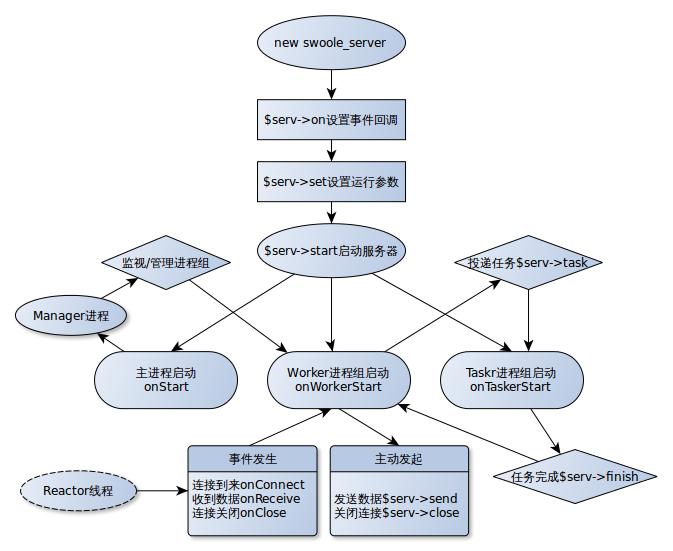
1. 利用fpm实现：例如你要实例一个100连接数的池，开启100个空闲fpm,然后每个fpm的连接都是数据库长连接。一般pm.max\_spare\_servers = 8这个配置项就是维持连接池的空闲数量，然后pm.max\_children = 50就是最大的连接数量。和fpm的进程数量一致。

**三、基于swoole的实现**

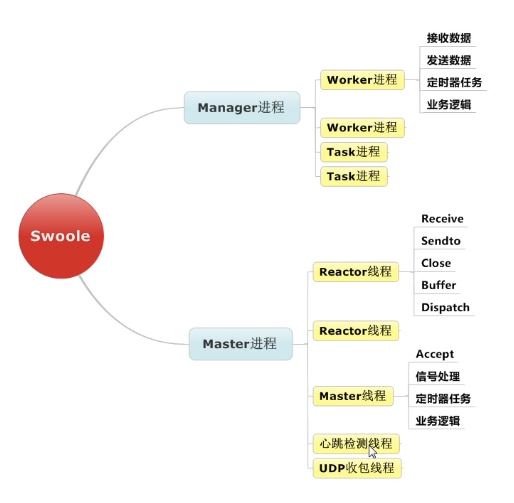
* swoole简单介绍（更多参阅swoole官网）

      swoole是一个PHP实现异步网络通信的引擎或者扩展，其中实现了很多传统PHP-fpm没有的东西，例如异步的客户端，异步Io，常驻内存，协程等等，一个个优秀的扩展，其中异步和协程等概念能应用于高并发场景。缺点是文档和入门的门槛都比较高，需要排坑。附上swoole的运行流程和进程结构图：

**运行流程图**

****

**进程/线程架构图**

****

* 基于swoole现实时的注意事项

首先，为了减少大家对之后运行示例代码产生不必要的天坑，先把注意事项和场景问题放前面：

1、程序中使用了协程的通信管道channel（与go的chan差不多的），其中swoole2是不支持chan->pop($timeout)中timeout超时等待的，所以必须用swoole4版本

2、使用swoole协程扩展的时候，一定不能装xdebug之类的扩展，否则报错。官方说明为：<https://wiki.swoole.com/wiki/page/674.html>，同时参考如下了解更多关于swoole协程的使用和注意：<https://wiki.swoole.com/wiki/page/749.html>

3、笔者使用的环境为：PHP 7.1.18和swoole4作为此次开发的环境

* 基于swoole现实连接池的方法

首先，此次利用swoole实现连接池，运用到swoole以下技术或者概念

1、连接变量池，这里可以看做一个数组或者队列，利用swoole全局变量的常驻内存特性，只要变量没主动unset掉，数组或队列中的连接对象可以一直保持，不释放。主要参考：<https://wiki.swoole.com/wiki/page/p-zend_mm.html>

2、协程。协程是纯用户状态的线程，通过协作的方式而不是抢占的方式来切换。首先此次的连接池两处用到协程：

* 一个是mysql的协程客户端，为什么要用协程客户端，因为如果是用同步客户端PDO，在一个进程处理内，就算有几百个连接池，swoole worker进程中用普通的PDO方式,随便并发多少个请求，每一个请求都只能等上一个请求执行完毕，woker才处理下一个请求，这里就算阻塞了。为了让一个worker支持阻塞切换出cpu去处理其他请求，所以要用到协程的协助切换，或者异步客户端也可以，但是异步客户端使用起来嵌套太多，很不方便。swoole协程可以无感知的用同步的代码编写方式达到异步IO的效果和性能。
* 第二个是底层实现了协程切换和调度的channel，以下详述什么是channel

3、Coroutine/channel通道，类似于go语言的chan，支持多生产者协程和多消费者协程。底层自动实现了协程的切换和调度。高并发时，容易出连接池为空时，如果用一般的array或者splqueue()作为介质存储连接对象变量，不能产生阻塞等待其他请求释放的效果，也就是说只能直接返回null.。所以这里用了一个swoole4协程中很牛逼的channel通过管道作为存储介质，它的出队方法pop($timeout)可以指定阻塞等待指定时间后返回。注意，是swoole2是没有超时timeout的参数，不适用此场景。在go语言中，如果chan等待或者push了没有消费或者生产一对一的情况，是会发生死锁。所以swoole4的timeout应该是为了避免无限等待为空channel情况而产生。主要参考：

<https://wiki.swoole.com/wiki/page/p-coroutine_channel.html>

channel切换的例子：

<?php

use \Swoole\Coroutine\Channel;

$chan = new Channel();

go(function () use ($chan) {

echo "我是第一个协程，等待3秒内有push就执行返回" . PHP\_EOL;

$p = $chan->pop(2);#1

echo "pop返回结果" . PHP\_EOL;

var\_dump($p);

});

go(function () use ($chan) {

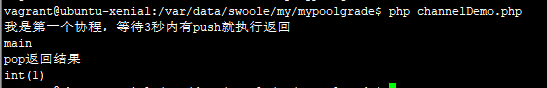
co::sleep(1);#2

$chan->push(1);

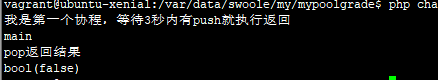
});

echo "main" . PHP\_EOL;

#1处代码会首先执行，然后遇到pop(),因为channel还是空，会等待2s。此时协程会让出cpu,跳到第二个协程执行，然后#2出睡眠1秒，push变量1进去channel后返回#1处继续执行，成功取车通过中刚push的值1.运行结果为：



如果把#2处的睡眠时间换成大于pop()的等待时间，结果是：



* 根据这些特性最终实现连接池的抽象封装类为：

<?php

/\*\*

\* 连接池封装.

\* User: user

\* Date: 2018/9/1

\* Time: 13:36

\*/

use Swoole\Coroutine\Channel;

abstract class AbstractPool

{

private $min;//最少连接数

private $max;//最大连接数

private $count;//当前连接数

private $connections;//连接池组

protected $spareTime;//用于空闲连接回收判断

//数据库配置

protected $dbConfig = array(

'host' => '10.0.2.2',

'port' => 3306,

'user' => 'root',

'password' => 'root',

'database' => 'test',

'charset' => 'utf8',

'timeout' => 2,

);

private $inited = false;

protected abstract function createDb();

public function \_\_construct()

{

$this->min = 10;

$this->max = 100;

$this->spareTime = 10 \* 3600;

$this->connections = new Channel($this->max + 1);

}

protected function createObject()

{

$obj = null;

$db = $this->createDb();

if ($db) {

$obj = [

'last\_used\_time' => time(),

'db' => $db,

];

}

return $obj;

}

/\*\*

\* 初始换最小数量连接池

\* @return $this|null

\*/

public function init()

{

if ($this->inited) {

return null;

}

for ($i = 0; $i < $this->min; $i++) {

$obj = $this->createObject();

$this->count++;

$this->connections->push($obj);

}

return $this;

}

public function getConnection($timeOut = 3)

{

$obj = null;

if ($this->connections->isEmpty()) {

if ($this->count < $this->max) {//连接数没达到最大，新建连接入池

$this->count++;

$obj = $this->createObject();

} else {

$obj = $this->connections->pop($timeOut);//timeout为出队的最大的等待时间

}

} else {

$obj = $this->connections->pop($timeOut);

}

return $obj;

}

public function free($obj)

{

if ($obj) {

$this->connections->push($obj);

}

}

/\*\*

\* 处理空闲连接

\*/

public function gcSpareObject()

{

//大约2分钟检测一次连接

swoole\_timer\_tick(120000, function () {

$list = [];

/\*echo "开始检测回收空闲链接" . $this->connections->length() . PHP\_EOL;\*/

if ($this->connections->length() < intval($this->max \* 0.5)) {

echo "请求连接数还比较多，暂不回收空闲连接\n";

}#1

while (true) {

if (!$this->connections->isEmpty()) {

$obj = $this->connections->pop(0.001);

$last\_used\_time = $obj['last\_used\_time'];

if ($this->count > $this->min && (time() - $last\_used\_time > $this->spareTime)) {//回收

$this->count--;

} else {

array\_push($list, $obj);

}

} else {

break;

}

}

foreach ($list as $item) {

$this->connections->push($item);

}

unset($list);

});

}

}

* 同步PDO客户端下实现

<?php

/\*\*

\* 数据库连接池PDO方式

\* User: user

\* Date: 2018/9/8

\* Time: 11:30

\*/

require "AbstractPool.php";

class MysqlPoolPdo extends AbstractPool

{

protected $dbConfig = array(

'host' => 'mysql:host=10.0.2.2:3306;dbname=test',

'port' => 3306,

'user' => 'root',

'password' => 'root',

'database' => 'test',

'charset' => 'utf8',

'timeout' => 2,

);

public static $instance;

public static function getInstance()

{

if (is\_null(self::$instance)) {

self::$instance = new MysqlPoolPdo();

}

return self::$instance;

}

protected function createDb()

{

return new PDO($this->dbConfig['host'], $this->dbConfig['user'], $this->dbConfig['password']);

}

}

$httpServer = new swoole\_http\_server('0.0.0.0', 9501);

$httpServer->set(

['worker\_num' => 1]

);

$httpServer->on("WorkerStart", function () {

MysqlPoolPdo::getInstance()->init();

});

$httpServer->on("request", function ($request, $response) {

$db = null;

$obj = MysqlPoolPdo::getInstance()->getConnection();

if (!empty($obj)) {

$db = $obj ? $obj['db'] : null;

}

if ($db) {

$db->query("select sleep(2)");

$ret = $db->query("select \* from guestbook limit 1");

MysqlPoolPdo::getInstance()->free($obj);

$response->end(json\_encode($ret));

}

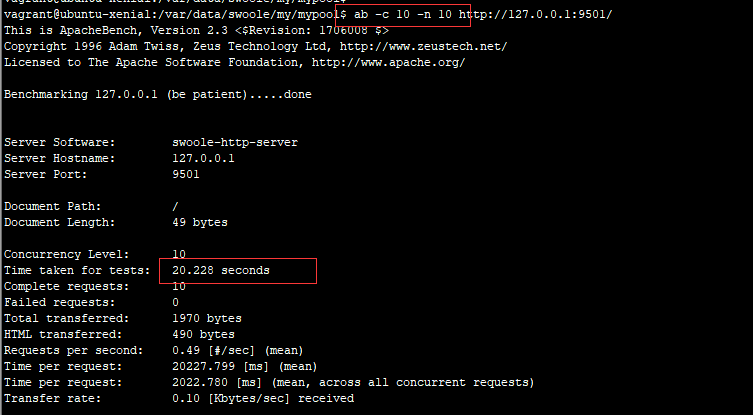
});

$httpServer->start();

代码调用过程详解：  
1、server启动时，调用init()方法初始化最少数量(min指定)的连接对象，放进类型为channelle的connections对象中。在init中循环调用中，依赖了createObject()返回连接对象，而createObject()  
中是调用了本来实现的抽象方法，初始化返回一个PDO db连接。所以此时，连接池connections中有min个对象。

2、server监听用户请求，当接收发请求时，调用连接数的getConnection()方法从connections通道中pop()一个对象。此时如果并发了10个请求，server因为配置了1个worker,所以再pop到一个对象返回时，遇到sleep()的查询，因为用的连接对象是pdo的查询，此时的woker进程只能等待，完成后才能进入下一个请求。因此，池中的其余连接其实是多余的，同步客户端的请求速度只能和woker的数量有关。  
3、查询结束后，调用free()方法把连接对象放回connections池中。

ab -c 10 -n 10运行的结果，单个worker处理，select sleep(2) 查询睡眠2s,同步客户端方式总共运行时间为20s以上，而且mysql的连接始终维持在一条。结果如下：



* 协程客户端Coroutine\MySQL方式的调用

<?php

/\*\*

\* 数据库连接池协程方式

\* User: user

\* Date: 2018/9/8

\* Time: 11:30

\*/

require "AbstractPool.php";

class MysqlPoolCoroutine extends AbstractPool

{

protected $dbConfig = array(

'host' => '10.0.2.2',

'port' => 3306,

'user' => 'root',

'password' => 'root',

'database' => 'test',

'charset' => 'utf8',

'timeout' => 10,

);

public static $instance;

public static function getInstance()

{

if (is\_null(self::$instance)) {

self::$instance = new MysqlPoolCoroutine();

}

return self::$instance;

}

protected function createDb()

{

$db = new Swoole\Coroutine\Mysql();

$db->connect(

$this->dbConfig

);

return $db;

}

}

$httpServer = new swoole\_http\_server('0.0.0.0', 9501);

$httpServer->set(

['worker\_num' => 1]

);

$httpServer->on("WorkerStart", function () {

//MysqlPoolCoroutine::getInstance()->init()->gcSpareObject();

MysqlPoolCoroutine::getInstance()->init();

});

$httpServer->on("request", function ($request, $response) {

$db = null;

$obj = MysqlPoolCoroutine::getInstance()->getConnection();

if (!empty($obj)) {

$db = $obj ? $obj['db'] : null;

}

if ($db) {

$db->query("select sleep(2)");

$ret = $db->query("select \* from guestbook limit 1");

MysqlPoolCoroutine::getInstance()->free($obj);

$response->end(json\_encode($ret));

}

});

$httpServer->start();

代码调用过程详解  
1、同样的，协程客户端方式下的调用，也是实现了之前封装好的连接池类AbstractPool.php。只是createDb()的抽象方法用了swoole内置的协程客户端去实现。  
2、server启动后，初始化都和同步一样。不一样的在获取连接对象的时候，此时如果并发了10个请求，同样是配置了1个worker进程在处理，但是在第一请求到达，pop出池中的一个连接对象，执行到query()方法，遇上sleep阻塞时，此时，woker进程不是在等待select的完成，而是切换到另外的协程去处理下一个请求。完成后同样释放对象到池中。当中有重点解释的代码段中getConnection()中。

public function getConnection($timeOut = 3)

{

$obj = null;

if ($this->connections->isEmpty()) {

if ($this->count < $this->max) {//连接数没达到最大，新建连接入池

$this->count++;

$obj = $this->createObject();#1

} else {

$obj = $this->connections->pop($timeOut);#2

}

} else {

$obj = $this->connections->pop($timeOut);#3

}

return $obj;

}

当调用到getConnection()时，如果此时由于大量并发请求过多，连接池connections为空，而没达到最大连接max数量时时，代码运行到#1处，调用了createObject(),新建连接返回；但如果连接池connections为空，而到达了最大连接数max时，代码运行到了#2处，也就是$this->connections->pop($timeOut)，此时会阻塞$timeOut的时间，如果期间有链接释放了，会成功获取到，然后协程返回。超时没获取到，则返回false。

3、最后说一下协程Mysql客户端一项重要配置，那就是代码里$dbConfig中timeout值的配置。这个配置是意思是最长的查询等待时间。可以看一个例子说明下：

go(function () {

$start = microtime(true);

$db = new Swoole\Coroutine\MySQL();

$db->connect([

'host' => '10.0.2.2',

'port' => 3306,

'user' => 'root',

'password' => 'root',

'database' => 'test',

'timeout' => 4#1

]);

$db->query("select sleep(5)");

echo "我是第一个sleep五秒之后\n";

$ret = $db->query("select user from guestbook limit 1");#2

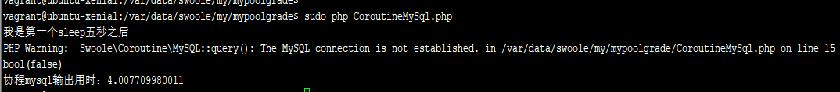
var\_dump($ret);

$use = microtime(true) - $start;

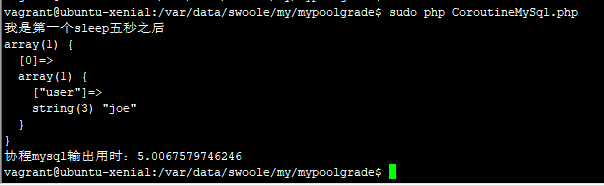
echo "协程mysql输出用时：" . $use . PHP\_EOL;

});

#1处代码，如果timeout配了4s查询超时，而第一条查询select sleep(5)阻塞后，协程切换到下一条sql的执行，其实$db并不能执行成功，因为用一个连接，同一个协程中，其实执行是同步的，所以此时第二条查询在等待4s超时后，没获取到db的连接执行，就会执行失败。而如果第一条查询执行的时间少于这个timeout,那么会执行查询成功。猜猜上面执行用时多少？结果如下：

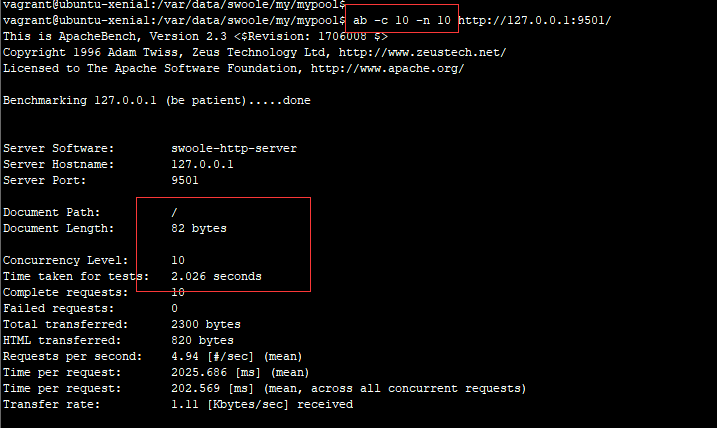


如果把timeout换成6s呢，结果如下：

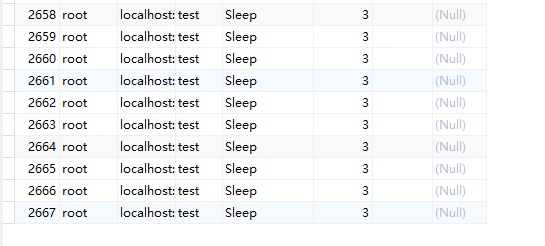


**所以要注意的是，协程的客户端内执行其实是同步的，不要理解为异步，它只是遇到IO阻塞时能让出执行权，切换到其他协程而已，不能和异步混淆。**

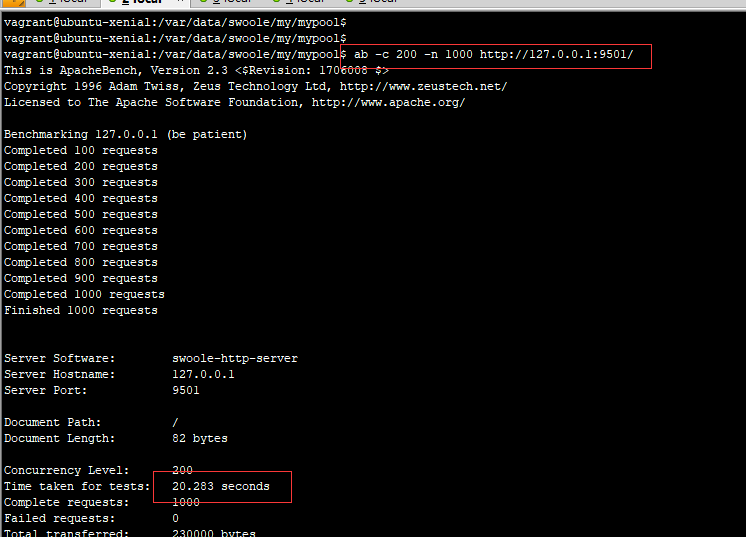
ab -c 10 -n 10运行的结果，单个worker处理，select sleep(2) 查询睡眠2s,协程客户端方式总共运行时间为2s多。结果如下：



数据库此时的连接数为10条（show full PROCESSLIST）：



再尝试 ab -c 200 -n 1000 http://127.0.0.1:9501/，200多个并发的处理，时间是20多秒，mysql连接数达到指定的最大值100个。结果如下：



**四、后言**

现在连接池基本实现了高并发时的连接分配和控制，但是还有一些细节要处理，例如：

* 并发时，建立了max个池对象，不能一直在池中维护这么多，要在请求空闲时，把连接池的数量维持在一个空闲值内。这里是简单做了gcSpareObject()的方法实现空闲处理。直接在初始化woker的时候调用：MysqlPoolCoroutine::getInstance()->init()->gcSpareObject();就会定时检测回收。问题是如何判断程序比较空闲，值得再去优化。
* 定时检测连接时候是活的，剔除死链
* 假如程序忘记调用free()释放对象到池，是否有更好方法避免这种情况？

对于以上，希望各大神看到后，能提供不错的意见！