——PHP高性能Web开发之

第3讲 PHP性能优化

目录

CONTENTS

目录

CONTENTS

PHP程序性能瓶颈

- **网络环境**：网络因素可能是访问速度的最大瓶颈；把应用程序部署在更高的带宽环境中，将会显著提升应用程序的访问速度；另一方面，带宽越高，应用程序的并发访问支持能力就越大。
- **服务器CPU**：如果仅仅加载普通的HTML网页，CPU能力对访问速度的影响可能微乎其微；如果加载动态PHP脚本，多核心的CPU将会显著提升应用程序的访问性能。
- **共享内存**：共享内存用于进程间通信，并用于存储在多个进程（如缓存的数据和代码）之间共享的资源；如果分配的共享内存不足，则尝试访问使用共享内存（如数据库连接或可执行代码）的资源将无法正常工作。
- **文件系统**：内存的访问速度远远大于硬盘的读写速度；而不同类型硬盘的读写速度也是相差悬殊。

PHP程序性能瓶颈

- **进程管理**：在多线程环境中运行PHP性能会得到显著的提升；如果你的Web服务器只是用来提供PHP服务，请选择Linux或Unix操作系统而不是Windows操作系统。
- **PHP运行模式**：PHP在Windows系统中主要有三种运行模式，可以使用**php_sapi_name( )**函数返回当前的运行模式。
  - ✓ Mod（模块）方式：只支持Apache服务器
  - ✓ FastCGI方式：可以使用 cgi-fcgi模式，或者 fpm-fcgi模式
  - ✓ CLI方式：命令行模式
- **与其它服务的交互**：如果你的程序需要与运行在其它服务器上的服务交互，那么交互的这个服务可能是性能的瓶颈所在；例如 较慢的数据库服务支撑大量的复杂的SQL请求。

性能测试工具

要进行PHP代码的性能优化，首先要找出当前脚本的性能状况。可以使用一些开源的性能测试工具测试应用程序的性能状况。

- Chrome浏览器开发者工具
- **Apache ab工具（压力测试工具）**
- XHProf性能测试工具
- **Xdebug性能测试工具**

Chrome浏览器开发者工具

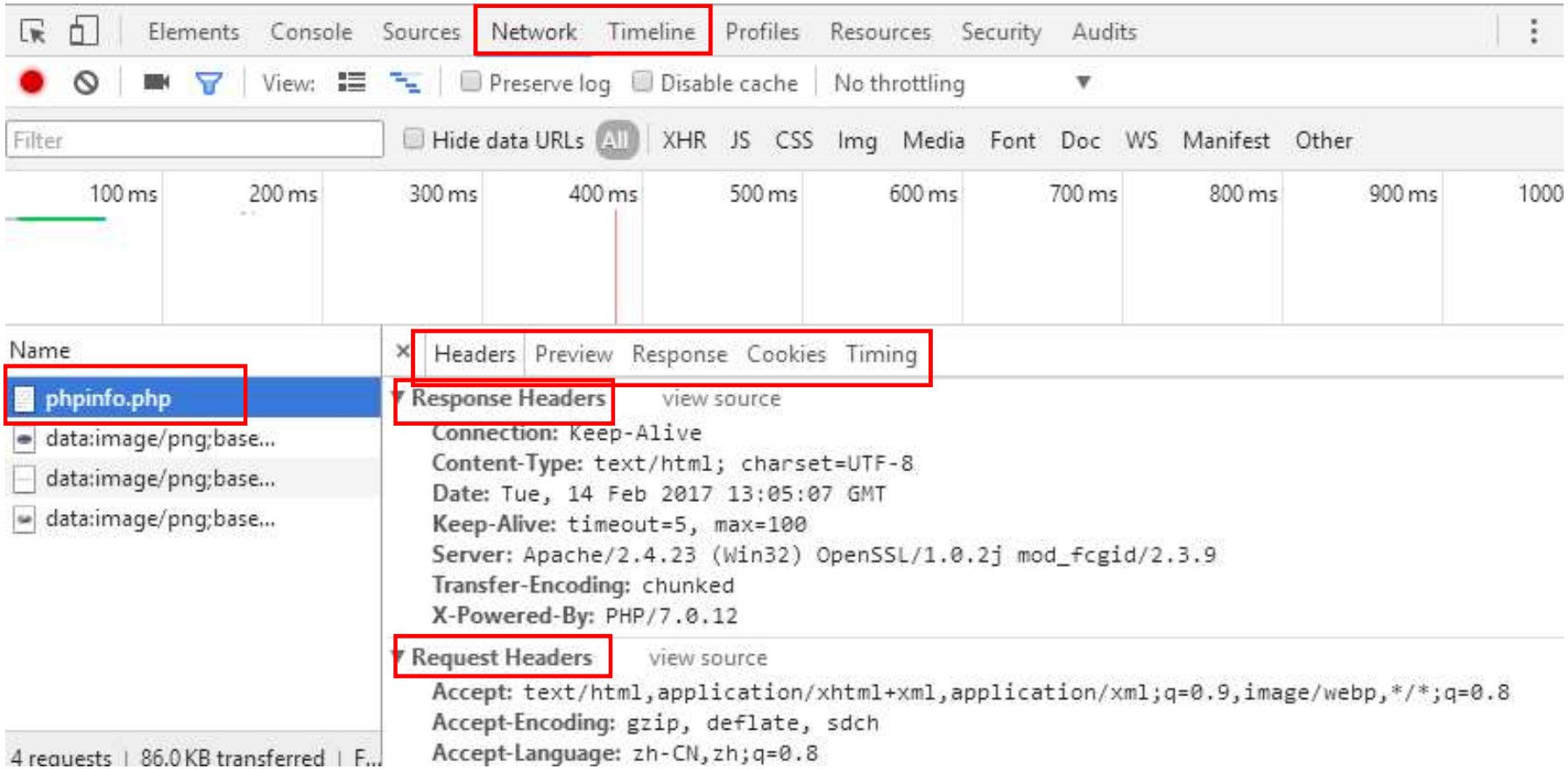
- 使用Chrome浏览器开发者工具，可以显示一次HTTP请求的请求消息、响应消息、响应时间等信息。
- 使用方法：

✓ 在Chrome浏览器中访问某一个网页（http://协议，不能是本地file://协议）

✓ F12快捷键，打开浏览器开发者工具；选中 “Network”面板

✓ 刷新网页，查看网络消息

Chrome浏览器开发者工具



ab压力测试工具

- Apache Benchmark (ab)工具是最著名的压力测试工具，它能够通过模拟对特定URL任意数量请求来对Web服务器进行压力测试。
- ab工具反馈的测试信息有：

✓ 传输的总数据大小（以字节为单位）

✓ Web服务器在模拟流量下每秒可以支持的请求总数

✓ 完成一次请求所花费的最长时间（以毫秒为单位）

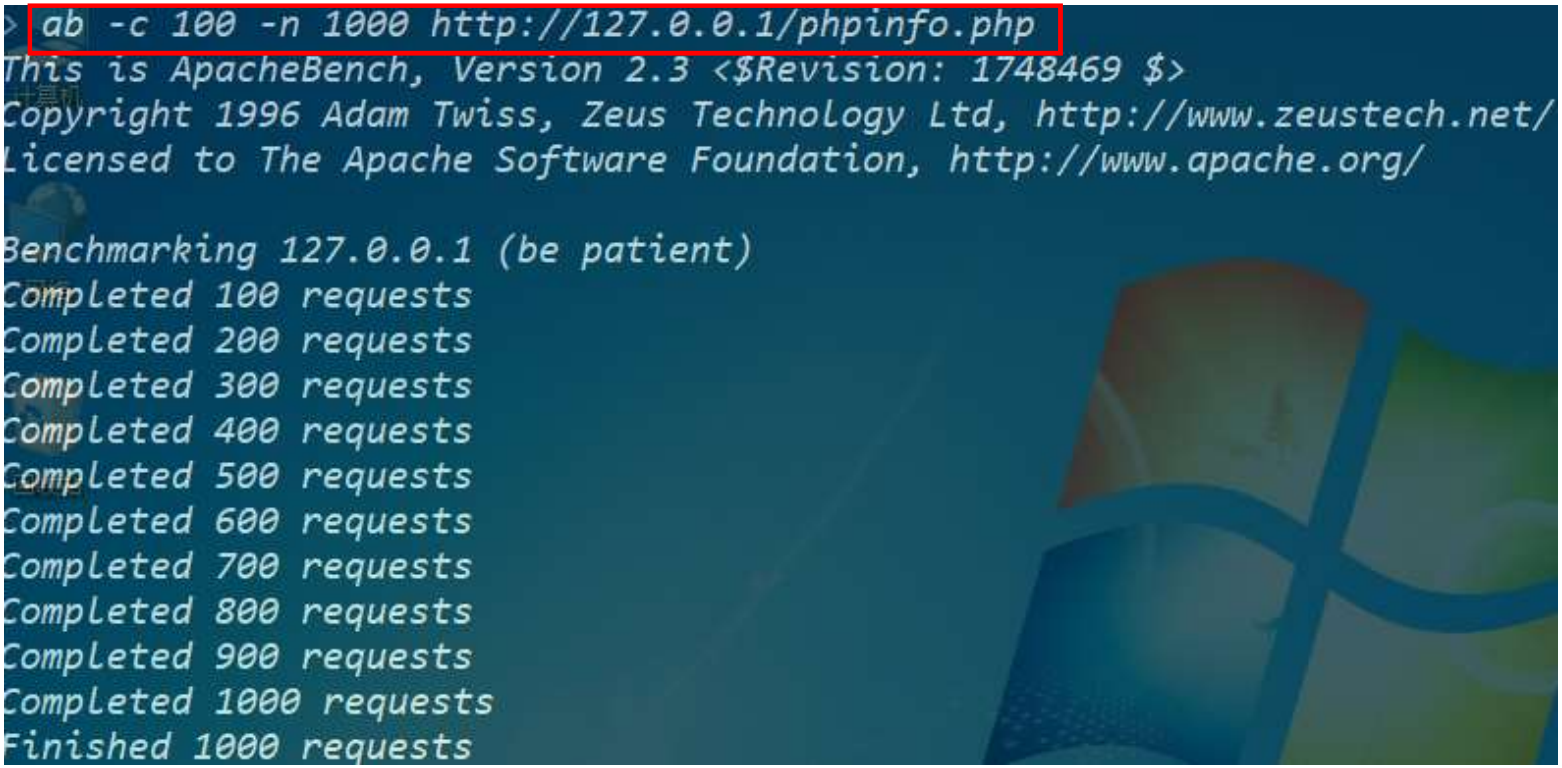
✓ 完成一次请求所花费的最短时间（以毫秒为单位）

- 基本使用方法： ab -n 次数 -c 并发数 请求URL

✓ ab工具位于Apache安装目录的 bin 目录下，直接在命令行中使用即可

✓ ab命令包含很多命令选项，直接输入 ab ，即可给出提示信息

ab压力测试工具


ab -c 100 -n 1000 http://127.0.0.1/phpinfo.php
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

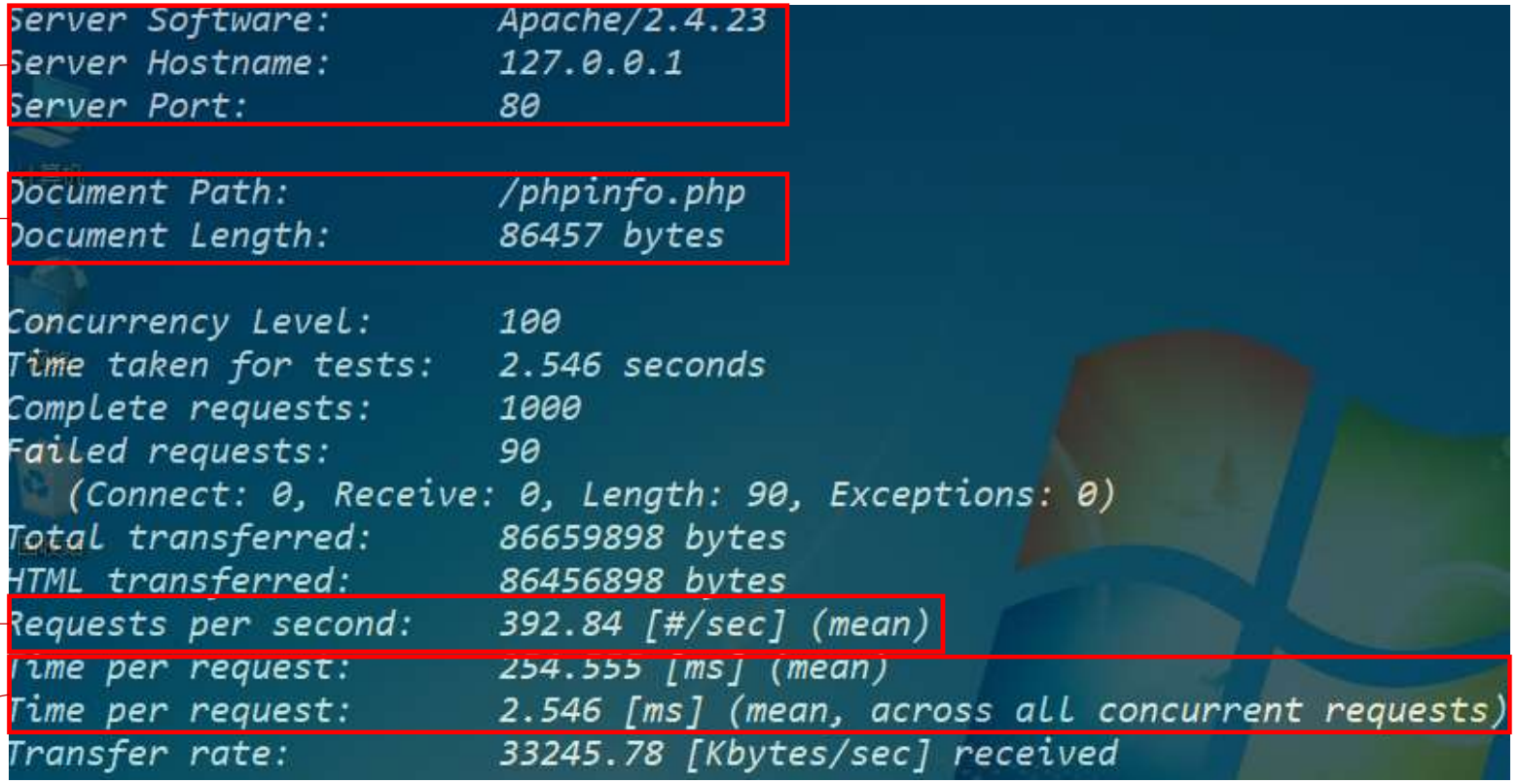
ab压力测试工具

服务器信息

访问网页信息

吞吐率，每秒事务数

每个请求处理时间并发请求处理时间



xDebug性能测试工具

- 安装xDebug扩展：直接在 phpstudy 中开启即可
- 配置xDebug扩展

✓ xdebug.profiler_enable = 0　　设置是否开启性能测试

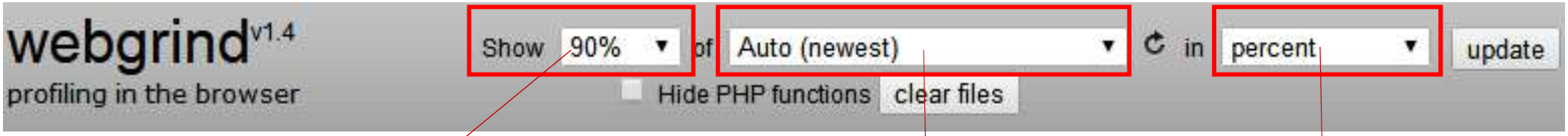
✓ xdebug.profiler_enable_trigger = 1　　若未开启性能测试，使用url查询字符串形式显示调用性能测试工具

✓ xdebug.profiler_output_dir=“D:\Program Files\tmp\xdebug“　　性能分析报告保存路径

✓ 注意，若未开启性能测试，在访问url时，需要给出查询字符串　**?XDEBUG_PROFILE=1**

xDebug性能测试工具

- 安装和使用webgrind分析工具
- ✓ 下载 webgrind：https://github.com/jokkedk/webgrind
- ✓ 安装 webgrind ： 直接解压缩到服务器根目录下即可
- ✓ 使用 webgrind工具：浏览器访问



显示调用开销时间的前百分比信息

选择待分析的文件（xdebug生成的文件）

结果显示形式（百分比形式）

xDebug性能测试工具

- 安装和使用webgrind分析工具
- ✓ 结果分析
  - Invocation Count：被调用次数
  - Total Self Cost：自身调用开销
  - Total Inclusive Cost：从程序开始到结束，该函数的总调用开销

用户定义函数

类方法

Function	Invocation Count	Total Self Cost	Total Inclusive Cost
Composer\Autoload\includeFile	220	20.81	37.49
Composer\Autoload\ClassLoader->findFileWithExtension	220	4.84	6.11
Illuminate\Container\Container->make	136	3.69	33.72
Illuminate\Container\Container->getContextualConcrete	181	3.03	4.37
Composer\Autoload\ClassLoader->findFile	220	2.15	8.37
Illuminate\Support\Arr::get	68	2.02	3.39
Illuminate\Foundation\Application->make	136	2.02	36.57
php::spl_autoload_call	222	1.90	50.58

PHP内置函数

目录

CONTENTS

PHP性能优化策略

- PHP性能优化策略：

✓ 服务器环境的优化：网络环境、CPU、内存、外部服务器等

✓ Opcode 和 Opcache 优化

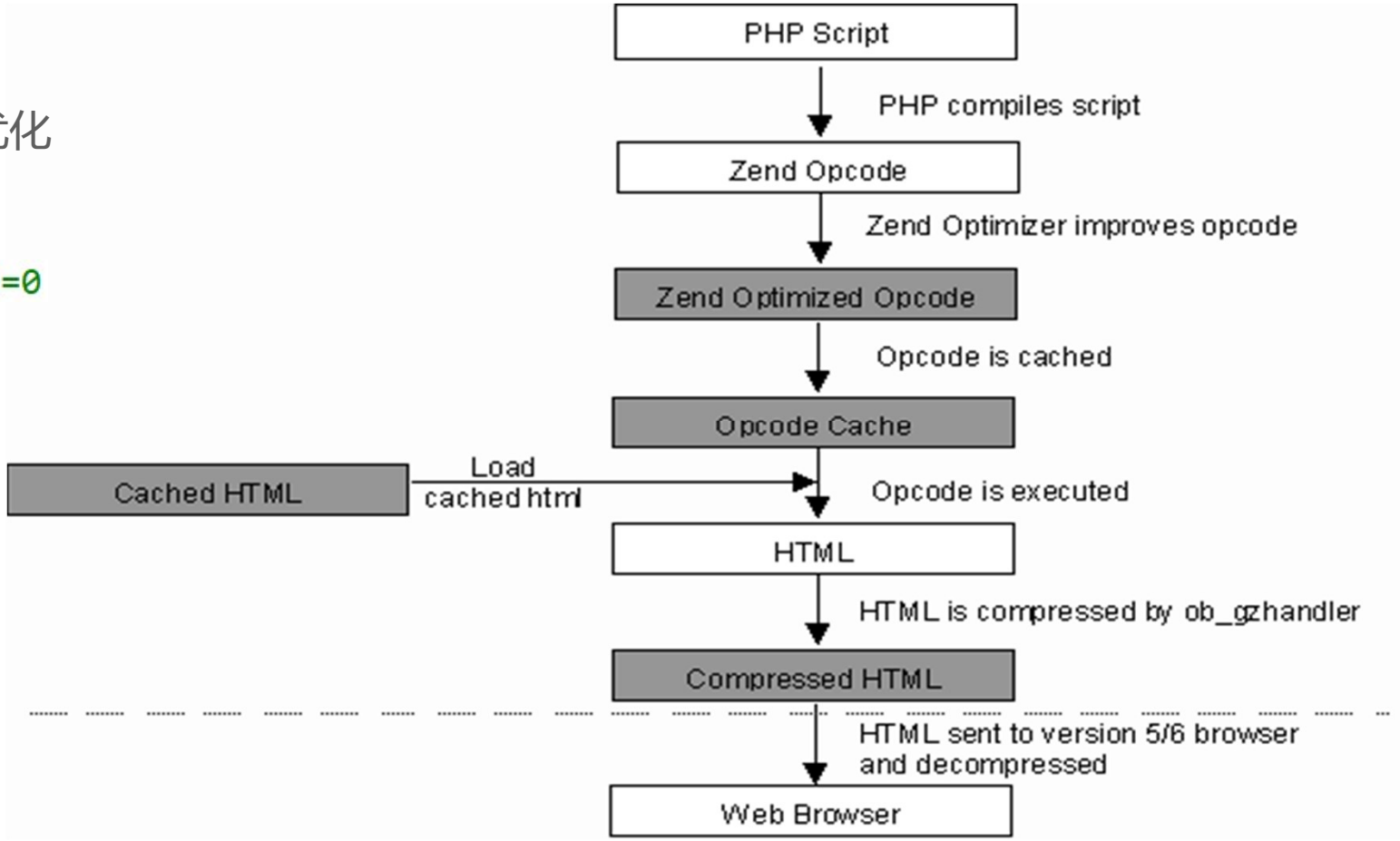
✓ 缓存机制

✓ PHP代码层优化

PHP性能优化策略

- Zend Enginer
- ✓ Zend Optimizer优化
- ✓ 开启 Opcache

```
;opcache.enable=0
```



PHP性能优化策略

- 缓存机制
- ✓ 数据库缓存：Memcached、Redis
- ✓ 静态页面缓存：减少数据库查询

PHP性能优化策略

- PHP代码层优化
- ✓ 引用传递函数参数问题：大数组、对象类型
- ✓ 数据库查询 预处理 问题
- ✓ 其它细节问题：http://net-beta.net/ubench/

目录

CONTENTS

Facebook遇到的问题

假设你有个 PHP 写的网站遇到了性能问题，经分析后发现很大一部分资源就耗在 PHP 上，这时你会怎么优化 PHP 性能?

- 方案1：迁移到性能更好的语言上，如 Java、C++、Go。
- 方案2：通过 RPC 将功能分离出来用其它语言实现，让 PHP 做更少的事情，比如 Twitter 就将大量业务逻辑放到了 Scala 中，前端的 Rails 只负责展现。
- 方案3：写 PHP 扩展，在性能瓶颈地方换 C/C++。
- 方案4：优化 PHP 本身性能。
- Facebook的选择：

✓ 方案1：放弃多年的经验积累，从头开始学习，成本巨大。

✓ 方案2：引入C++，但是C++开发成本太高，不适合使用在经常修改的应用中。

✓ 方案3：开发成本过高，且效果不明显（性能瓶颈是高并发情况下累积的结果）。

PHP性能优化思路

- 方案1：PHP 语言层面的优化；Facebook开发出XHProf性能分析工具，在一定程序上提升了性能，但并没有很好解决问题。
- 方案2：优化 PHP 的官方实现（也就是 Zend）；提升Zend执行性能，需要对程序的底层执行有深刻理解，并且修改代码太大，甚至不如重构。
- 方案3：将 PHP 编译成其它语言的 bytecode（字节码），借助其它语言的虚拟机（如JVM）来运行；VM总是为某个语言优化的，其它语言在上面实现会遇到很多瓶颈。
- 方案4：将 PHP 转成 C/C++，然后编译成本地代码；该方案即为HHVM的前身（HipHop Compile），相对于VM来说，实现简单，但是很难支持PHP中的动态方法，不太适合于动态语言。
- 方案5：开发更快的 PHP 虚拟机；即HHVM。

HHVM和Hack

- HipHop Virtual Machine (HHVM) 是一个基于即时（JIT）编译的开源虚拟机，作为PHP和Hack编程语言的执行引擎。 HHVM是由Facebook在 HPHPC 引擎上基础上扩展而来的，在HPHPC的基础上，引入对PHP动态特性的支持，同时大幅提升了PHP引擎的执行速度。
- **Hack**：Facebook推出的编程语言，同时结合了动态类型语言（如PHP语言）和静态类型语言（如C语言）的特性，需要在HHVM引擎上执行，而不能在Zend引擎上执行。

✓ Hack相对于PHP，最显著的区别在于提供了静态类型的支持。

Hack

```hack
<?hh // strict
class WidgetContainer
{
    protected Vector<Widget> $widgets;

    public function __construct(array<Widget> $widgets = array())
    {
        foreach ($widgets as $widget) {
            $this->addWidget($widget);
        }
    }

    public function addWidget(Widget $widget) : this
    {
        $this->widgets[] = $widget;

        return this;
    }
```

Hack开始标识

定义属性时，指明数据类型

方法参数中，指明数据类型

方法的返回值

HHVM&Hack VS PHP7

Facebook推出的HHVM和Hack，给PHP引擎（Zend）带来了极大的挑战。PHP开发组成员，开发方向转向Zend引擎的性能优化和PHP语言的性能优化上，从而带来了PHP7的发布。

PHP7大量借鉴了Hack的语法特性，引入静态类型的特点，保证了Zend引擎的性能，同时大大提高了PHP程序的开发效率。

在性能上HHVM大大超过PHP7以下的版本，而PHP7已经十分接近（并在某些程序上超过）HHVM的性能。在实际的应用中，建议选择PHP7作为主力生产工具，享受现代PHP带来的快乐！

PHP7新语法

PHP7提供了一些新的语法特征：

- ✓ 类型提示
- ✓ 返回类型
- ✓ 匿名类
- ✓ 闭包函数的call方法
- ✓ 新的运算符
- ✓ 批量引入命名空间支持
- ✓ 常量数组
- ✓ 生成器（yield）
- ✓ ……

类型提示

- 在PHP7之前，只支持 对象类型或数组类型 的类型提示，在PHP7中，支持所有数据类型的类型提示。

✓ 支持的类型：int 、float 、string 、Bool 、array 、类名或接口名

✓ 开启强制类型提示指令： declare(strict_types = 1)

```php
<?php
declare(strict_types=1);    // 开启强制类型声明
function test(int $a, string $b)
{
    var_dump($a, $b);
}
test(3, 'hello');
```

返回类型

- PHP7中，支持指定函数的返回类型：

✓ 基本格式：function 函数名(参数列表)：**返回类型**{ …… }

✓ 若已经设置 declare(strict_types = 1)，则返回类型不匹配时，会出现错误。

```php
<?php
declare(strict_types=1);    // 开启强制类型声明
function add(int $a, int $b) : int
{
    return $a + $b;
}
echo add(3, 3.14);
```

匿名类

- PHP7中，支持创建匿名类的对象：
- ✓ 基本格式：$obj = new class( ) { ..... }
- ✓ 匿名类可以嵌套在普通类中，但是不能使用普通类中的属性，可以通过构造参数传递
- ✓ 匿名类可以 extends 父类、implements 接口、 use traits
- ✓ 匿名类不能被序列化（因为没有类的结构）
- ✓ 匿名类的主要作用：使用指定接口的对象

```php
class Outer
{
    private $prop = 1;
    public function func2()
    {
        return new class($this->prop) extends Outer {
            public $prop2;
            public function __construct($prop)
            {
                $this->prop2 = $prop;
            }
        };
    }
}

echo (new Outer)->func2()->prop2;
```

Closure::call( )

- PHP7中，闭包函数支持使用 $this 关键字

✓ 使用 call 方法调用

✓ 作用：将一个闭包函数动态绑定到一个新的对象实例并调用执行该函数

```php
class Customer {
    private $firstname;
    private $lastname;
    public function __construct($firstname, $lastname)
    {
        $this->firstname = $firstname;
        $this->lastname = $lastname;
    }
}
$customer = new Customer('张', '三');
$greeting = function ($message) {
    return "$message $this->firstname $this->lastname!";
};
echo $greeting->call($customer, 'Hello');
```

新的运算符

- PHP7中，添加了两个新的运算符：

✓ ?? 运算符：用于执行isset()检测的三元运算的快捷方式

```php
// before php7
$page = isset($_GET['page']) ? $_GET['page'] : 1;
// after php7
$page = $_GET['page'] ?? 1;
```

✓ <=> 运算符：比较两个表达式 $a 和 $b，如果 $a 小于、等于或大于 $b时，它分别返回-1、0或1

```php
// before php7
if ($a < $b) {
    echo -1;
} else if ($a == $b) {
    echo 0;
} else {
    echo 1;
}
// after php7
echo $a <=> $b;
```

use命名空间

- PHP7中，支持使用一个 use 从同一个 namespace 中导入类、函数和常量。

```php
// PHP 7 之前版本需要使用多次 use
use some\namespace\ClassA;
use some\namespace\ClassB;
use some\namespace\ClassC as C;

use function some\namespace\fn_a;
use function some\namespace\fn_b;
use function some\namespace\fn_c;

use const some\namespace\ConstA;
use const some\namespace\ConstB;
use const some\namespace\ConstC;

// PHP 7+ 之后版本可以使用一个 use 导入同一个 namespace 的类
use some\namespace\{ClassA, ClassB, ClassC as C};
use function some\namespace\{fn_a, fn_b, fn_c};
use const some\namespace\{ConstA, ConstB, ConstC};
```

常量数组

- PHP7中，支持通过 define( ) 定义常量数组：

```php
define('SITES', [
    'Google',
    'Runoob',
    'Taobao'
]);

var_dump(SITES);
```

生成器yield

- 生成器提供了一种更容易的方法来实现简单的对象迭代，相比较定义类实现 Iterator 接口的方式，性能开销和复杂性大大降低。
- ✓ 借助关键字 yield 实现
- ✓ 自php5.5+即可使用
- ✓ 使用场景：
  - 内存溢出情况：
  - 协程工作情况：

```php
// 传统形式
echo memory_get_usage(), "\n";
$arr = range(1, 1000000);
echo memory_get_usage(), "\n";

// 使用生成器
function xrange($start, $limit)
{
    for ($i = $start; $i <= $limit; $i++) {
        yield($i);
    }
}
echo memory_get_usage(), "\n";
$arr = xrange(1, 1000000);
echo memory_get_usage(), "\n";
```

感谢聆听！

THANK YOU FOR YOUR ATTENTION