### 一、背景

随着的互联网技术的不断发展，浏览器逐渐成了集大成的CS客户端，页面功能在向系统级、软件级靠拢的趋势下，开发团队也需要一些软件工程的方法来开发WEB项目，如**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/lipc_/article/details/_blank)**驱动、面向对象，而模块化编程更是成为一种急需应用的技术。

### 二、原因

为方便文件管理、增加复用，我们需要细化**[js](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/lipc_/article/details/_blank)**文件，每个文件负责单一职能，称之为模块，明确每个文件的职能，当交互功能较复杂时，引用的文件也越加庞大，此时我们就需要模块化管理。

如我们编写一个模块组件Dialog.js，其依赖event.js / bootstrap.js / component.js，而event.js又依赖 base.js / core.js / el.js，bootstrap.js依赖 boot.js / under.js / almond.js，同理可能依赖更多的js，传统方法我们用到Dialog.js时实际上需要引用9个js，如下

<html><body>

...

<script src="/public/base.js"></script>

<script src="/public/core.js"></script>

<script src="/public/el.js"></script>

<script src="/public/boot.js"></script>

<script src="/public/under.js"></script>

<script src="/public/almond.js"></script>

<script src="/module/event.js"></script>

<script src="/module/bootstrap.js"></script>

<script src="/module/component.js"></script>

<script src="/widget/dialog.js"></script></body></html>

我们将其细分，Dialog所依赖的每个js都负责了某一独立功能，分开便于其它组件复用，也利于学习理解。

但将这9个js都到html里显示引用，工作量却非常可观，且不能弄错路径及加载顺序，若后续Dialog.js又需要多引用一个js，所有的项目都要手动增加，且这样依次同步加载下来会导致会占用浏览器线程，可能导致假死。

### 三、作用

模块化的作用就是解决上面提到的两点问题。

1、实现js文件的异步加载，避免浏览器假死。  
2、管理模块之间的依赖，便于模块的维护

### 四、实现

模块化，就是把一个相对独立的功能，单独成一个文件，可输入指定依赖、输出指定的函数，供外界调用，其它都是内部的。这样即可方便不同的项目重复使用，也不会对其造成额外的影响。而既然要给不同的项目应用，特别像**[Git](http://lib.csdn.net/base/git" \o "Git知识库" \t "http://blog.csdn.net/lipc_/article/details/_blank)**上的模块是面向全球范围的，必然要形成可遵循的规范，开发者和使用者都有据可寻。

目前通用**[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/lipc_/article/details/_blank)**规范分为服务端（CommonJS），客户端（AMD / CMD）。

CommonJS是针对**[javascript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/lipc_/article/details/_blank)**的服务端，一切模块都是通过require来同步加载调用， Nodejs采用的就是这个规范。

var module1 = require( 'module1'),

module2 = require( 'module2' );

function module3(){

...

}

exports.module3 = module3;

注：通过require加载module1、2，操作后通过exports导致module3。

这里我们主要讲客户端模块

客户端与服务端的不同在于异步加载，之所以异步是因为CommonJS规则应用在服务端时，文件都是本地硬盘读取，不受限于网络，等待硬盘读取时间很短。而浏览器要考虑网速，若长时间等待会导致浏览器假死。

引用形式如下图

IMG_256

### 五、AMD

是"Asynchronous Module Definition"的缩写，意思就是"异步模块定义"，

以require.js为代表，官网[http://requirejs.org/](http://requirejs.org/" \t "http://blog.csdn.net/lipc_/article/details/_blank) ，其形式主要是

=============main.js===================

<script src="js/require.js" data-main="js/Dialog"></script>

==============Dialog.js=======================

define( ['event', 'bootstrap', 'component' ], function( event, bootstrap, component ){

event.doSomthing();

bootstrap.doSomthing();

component.doSomthing();

var Dialog = function(){

...

}

return {

Dialog: Dialog

}

});

注：require接收两个参数，第一个参数是数组，含需加载的模块，第二个参数是回调，在模块加载成功后执行，且回调中的参数对应模块的加载顺序，这样主程序只需要加载一个JS文件，Dialog.js即可，其自会去加载event.js等文件，event.js再用同样的方式去加载base.js等，使用者无需考虑Dialog加载了多少引用，也无法担心其使用后会增减依赖。

### 六、CMD

以sea.js为代表，官网 [http://seajs.org/docs/](http://seajs.org/docs/" \t "http://blog.csdn.net/lipc_/article/details/_blank)，理念为需要时再执行，即依赖就近

=============main.js===================

<script src="js/sea.js"></script>

<script>seajs.use("../static/hello/src/main")</script>

==============Dialog.js=======================

define( function( require, exports, module ){

var event = require( 'event' );

event.doSomthing();

var bootstrap = require( 'bootstrap' );

bootstrap.doSomthing();

var component = require( 'component' );

component.doSomthing();

var Dialog = function(){

...

}

exports.Dialog = Dialog;

})

注：模块引用本身就在回调里，并且在使用前引用即可，所以sea.js强调的是懒加载，这也就是AMD和CMD之间唯一的区别。实际上，require.js也支持CMD的写法，不过依赖前置是其官方推荐。我最初因为对国产和玉伯的支持敬佩，使用了seajs，近来为了弄清楚AMD，项目全部用了requirejs，虽然网上及seajs官方对两种逻列了一些分别，但使用下来，感觉并无优劣。

### 七、UMD

因服务端与客户端对模块化加载方式的不同，就会人想做一个兼容的模块UMD（Universal Module Definition），来解决跨平台的问题。但对于我们开发者来讲实则没有什么意义，也不深入了。

### 八、require.js 使用

require.config({

baseUrl: 'module/js', //配置path下的统一目的，以便path中不会分别配置

path: { //配置module路径，以便引用时不用写全路径

module1: 'module1',

module2: 'module2',

module3: 'http://www.site.com/module3' //若引用的是第三方，可以全部全路径

},

shim: { //对不符合AMD规则的模块引用

'underscore': {

exports: '\_' //配置其原对外接口

},

'backbone': {

deps: ['underscore', 'jquery'], //配置其依赖项

exports: 'Backbone'

}

}

});

define( ['module1', 'module2' ], function( module1, module2 ){

module1.doSomthing();

module2.doSomthing();

function module3(){

...

}

return {

module3: module3

}

})

### 九、require.js 时间戳

require.js没有找到可配置时间戳，可修改require.js源文件中的load方法

load: function (id, url) {

url += '?' + timestamp;

req.load(context, id, url);

},

新增 url += '?' + timestamp; 这一处语句，即为url添加timestamp，timestamp可以另外通过配置文件配置。我使用nodejs开发，在服务端的config文件配置 timestamp: '2015091112'，将其插入到客户端 ejs文件内

<script>

var timestamp = '<%= timestamp %>';

</script>

这样即可在 require.js 文件内调用。

### 十、require.js 合并

官网提供了r.js用于该模块化的打包 [https://github.com/jrburke/r.js](https://github.com/jrburke/r.js" \t "http://blog.csdn.net/lipc_/article/details/_blank)  
r.js是将通过require.js引用的模块化文件都合并到一个文件里，以减少文件加载数量。  
项目参考 [https://github.com/xiaolulu/mynodejs.git|](https://github.com/xiaolulu/mynodejs.git|" \t "http://blog.csdn.net/lipc_/article/details/_blank) d7a0f54705a10ac329dd2f62a7d569da2 |  
首先下载r.js，可以到静态文件根目录，新建build.js用于配置r.js，常见的配置如下

build.js

({

appDir: './', //项目根目录，给后面的baseUrl做参考;非必要项;默认为与r.js平级目录

dir: './min', //压缩后的文件的输出目录

baseUrl: './', //模块相对目录，与后面的paths配置使用

modules: [

{ name:'login'},

{ name: 'register' },

{ name: 'center' }

],

fileExclusionRegExp: /^(r|build)\.js|.\*\.scss$/, //过滤文件不压缩

optimizeCss: 'standard', //css的标准方式

optimize: 'uglify', //设置是否压缩;uglify(压缩格式)|none(只合并不压缩)

removeCombined: true, //设置移除压缩后的文件

paths: { //设置模块别名，方便引用，压缩后require.js引用的模块路径就在这里定义的，不是require.config里的

md5: 'core/js/md5',

validate: 'widget/validate/validate',

all: 'public/js/all',

login: 'module/issue/login/login',

register: 'module/issue/register/register',

center: 'module/user/center/center'

}

})

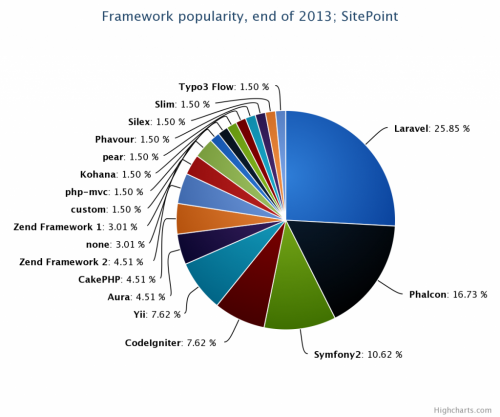
shell执行 node r.js -o build.js

Php的代码规范

单引号代替双引号来包含字符串，这样做会更快一些。因为 PHP 会在双引号包围的字符串中搜寻变量，单引号则不会，注意：只有 echo 能这么做，它是一种可以把多个字符串当作参数的“函数”(译注：PHP 手册中说 echo 是语言结构，不是真正的函数，故把函数加上了双引号)。  
  
  
  
1、如果能将类的方法定义成 static，就尽量定义成 static，它的速度会提升将近4倍。  
  
2、$row['id'] 的速度是 $row[id] 的7倍。  
  
3、echo 比 print 快，并且使用 echo 的多重参数(译注：指用逗号而不是句点)代替字符串连接，比如：echo $str1,$str2。  
  
4、在执行 for 循环之前确定最大循环数，不要每循环一次都计算最大值，最好运用 foreach 代替。  
  
5、注销那些不用的变量尤其是大数组，以便释放内存。  
  
6、尽量避免使用 \_\_get、\_\_set、\_\_autoload。  
  
7、require\_once() 代价昂贵。  
  
8、include 文件时尽量使用绝对路径，因为它避免了 PHP 去 include\_path 里查找文件的速度，解析操作系统路径所需的时间会更少。  
  
9、如果你想知道脚本开始执行(译注：即服务器端收到客户端请求)的时刻，使用 $\_SERVER['REQUEST\_TIME']要好于 time()。  
  
10、函数代替正则表达式完成相同功能。  
  
11、str\_replace 函数比 preg\_replace 函数快，但 strtr 函数的效率是 str\_replace 函数的四倍。  
  
12、如果一个字符串替换函数，可接受数组或字符作为参数，并且参数长度不太长，那么可以考虑额外写一段替换代码，使得每次传递参数是一个字符，而不是只写一行代码接受数组作为查询和替换的参数。  
  
13、使用选择分支语句(译注：即 switch case)好于使用多个 if、else if 语句。  
  
14、用 @ 屏蔽错误消息的做法非常低效，极其低效。  
  
15、打开 apache 的 mod\_deflate 模块，可以提高网页的浏览速度。  
  
16、数据库连接当使用完毕时应关掉，不要用长连接。  
  
17、错误消息代价昂贵。  
  
18、在方法中递增局部变量，速度是最快的。几乎与在函数中调用局部变量的速度相当。  
  
19、递增一个全局变量要比递增一个局部变量慢 2 倍。  
  
20、递增一个对象属性(如：$this->prop++)要比递增一个局部变量慢3倍。  
  
21、递增一个未预定义的局部变量要比递增一个预定义的局部变量慢9至10倍。  
  
22、仅定义一个局部变量而没在函数中调用它，同样会减慢速度(其程度相当于递增一个局部变量)。PHP 大概会检查看是否存在全局变量。  
  
23、方法调用看来与类中定义的方法的数量无关，因为我(在测试方法之前和之后都)添加了 10 个方法，但性能上没有变化。  
  
24、派生类中的方法运行起来要快于在基类中定义的同样的方法。  
  
25、调用带有一个参数的空函数，其花费的时间相当于执行 7 至 8 次的局部变量递增操作。类似的方法调用所花费的时间接近于 15 次的局部变量递增操作。  
  
26、Apache 解析一个 PHP 脚本的时间要比解析一个静态 HTML 页面慢 2 至 10 倍。尽量多用静态 HTML 页面，少用脚本。  
  
27、除非脚本可以缓存，否则每次调用时都会重新编译一次。引入一套 PHP 缓存机制通常可以提升 25% 至 100% 的性能，以免除编译开销。  
  
28、尽量做缓存，可使用 memcached。memcached 是一款高性能的内存对象缓存系统，可用来加速动态 Web 应用程序，减轻数据库负载。对运算码 (OP code)的缓存很有用，使得脚本不必为每个请求做重新编译。  
  
29、当操作字符串并需要检验其长度是否满足某种要求时，你想当然地会使用 strlen() 函数。此函数执行起来相当快，因为它不做任何计算，只返回在zval 结构(C的内置数据结构，用于存储PHP变量)中存储的已知字符串长度。但是，由于 strlen() 是函数，多多少少会有些慢，因为函数调用会经过诸多步骤，如字母小写化(译注：指函数名小写化，PHP 不区分函数名大小写)、哈希查找，会跟随被调用的函数一起执行。在某些情况下，你可以使用isset() 技巧加速执行你的代码。  
  
(举例如下)  
  
if (strlen($foo) < 5) { echo "Foo is too short"; }  
  
(与下面的技巧做比较)  
  
if ( ! isset($foo{5})) { echo "Foo is too short"; }  
  
调用 isset() 恰巧比 strlen() 快，因为与后者不同的是，isset() 作为一种语言结构，意味着它的执行不需要函数查找和字母小写化。也就是说，实际上在检验字符串长度的顶层代码中你没有花太多开销。  
  
34、当执行变量$i的递增或递减时，$i++ 会比 ++$i 慢一些。这种差异是 PHP 特有的，并不适用于其他语言，所以请不要修改你的 C 或 Java 代码并指望它们能立即变快，没用的。++$i 更快是因为它只需要 3 条指令(opcodes)，$i++ 则需要 4 条指令。后置递增实际上会产生一个临时变量，这个临时变量随后被递增。而前置递增直接在原值上递增。这是最优化处理的一种，正如 Zend 的 PHP 优化器所作的那样。牢记这个优化处理不失为一个好主意，因为并不是所有的指令优化器都会做同样的优化处理，并且存在大量没有装配指令优化器的互联网服务提供商(ISPs)和服务器。  
  
35、并不是事必面向对象(OOP)，面向对象往往开销很大，每个方法和对象调用都会消耗很多内存。  
  
36、并非要用类实现所有的数据结构，数组也很有用。  
  
37、不要把方法细分得过多，仔细想想你真正打算重用的是哪些代码？  
  
38、当你需要时，你总能把代码分解成方法。  
  
39、尽量采用大量的 PHP 内置函数。  
  
40、如果在代码中存在大量耗时的函数，你可以考虑用C扩展的方式实现它们。  
  
41、评估检验(profile)你的代码。检验器会告诉你，代码的哪些部分消耗了多少时间。Xdebug 调试器包含了检验程序，评估检验总体上可以显示出代码的瓶颈。  
  
42、mod\_zip 可作为 Apache 模块，用来即时压缩你的数据，并可让数据传输量降低 80%。  
  
43、在可以用 file\_get\_contents 替代 file、fopen、feof、fgets 等系列方法的情况下，尽量用 file\_get\_contents，因为他的效率高得多！但是要注意 file\_get\_contents 在打开一个 URL 文件时候的 PHP 版本问题；  
  
44、尽量的少进行文件操作，虽然 PHP 的文件操作效率也不低的；  
  
45、优化 SELECT SQL 语句，在可能的情况下尽量少的进行 INSERT、UPDATE 操作(在 update 上，我被恶批过)；  
  
46、尽可能的使用 PHP 内部函数(但是我却为了找个 PHP 里面不存在的函数，浪费了本可以写出一个自定义函数的时间，经验问题啊！)；  
  
47、循环内部不要声明变量，尤其是大变量：对象(这好像不只是PHP里面要注意的问题吧？)；  
  
48、多维数组尽量不要循环嵌套赋值；  
  
49、在可以用 PHP 内部字符串操作函数的情况下，不要用正则表达式；  
  
50、foreach 效率更高，尽量用 foreach 代替 while 和 for 循环；  
  
51、用单引号替代双引号引用字符串；  
  
52、“用 i+=1 代替 i=i+1 。符合 c/c++ 的习惯，效率还高”；  
  
53、对 global 变量，应该用完就 unset() 掉。

## phalcon介绍

phalcon是一款**[C语言](http://lib.csdn.net/base/c" \o "C语言知识库" \t "http://blog.csdn.net/u011142688/article/details/_blank)**写的底层框架,应为是**[c语言](http://lib.csdn.net/base/c" \o "C语言知识库" \t "http://blog.csdn.net/u011142688/article/details/_blank)**底层性能自然不用说,并且功能比较全面目前最新版本为v.2.0.9,当我们在开发非常注重性能的项目时它会是一个不错的选择,下图为14年的框架排行,虽然在15年有所降低但是也在框架使用前列



接下来介绍一下我对它的优缺点的认识:

### 1.1 phalcon的优点

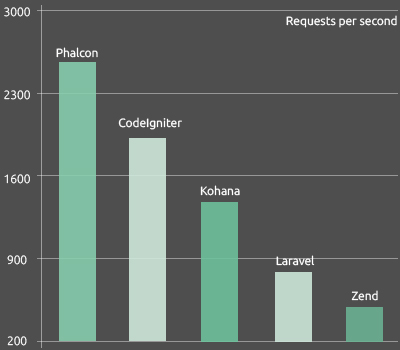
1. 说到优点不得不提的就是性能卓越,号称是最快的php框架
2. 先进的思想多样的功能,phalcon给我的第一感觉就是哇功能这么强大
3. 不俗的热情phalcon在国外的热度基本不用说,并且在国内也有很多爱好者和追捧者并且建立了中文社区等等
4. 灵活搭配,phalcon并不像某些框架,你需要这样这样或者是这样这样你才能使用起来,phalcon基本松耦合的状态可以自己搭配出需要的功能

### 1.2 phalcon的缺点

1. 说道缺点不得不提的就是黑盒开发,在框架内部出了问题很难基本就无解了
2. 中文资料较少,虽然官网给出了中文文档但是翻译着实不全
3. 对新手不友好学习成本高

## 2 phalcon对比其他框架

我们先看一下官方提供的框架性能对比图:



在这里我针对于我们几个常用的框架做了一下对比Yaf(鸟哥写的C语言框架),ThinkPhp(国内运用广泛的mvc框架),PhalApi(纯API框架),phalcon   
分别从性能,功能,易用性,国内社区支持**(此次排名均属于个人观点提供参考)**

### 性能:

yaf从性能上来说是最好的他的速度最快原因是应为底层功能逼phalcon简单 差距非常小

phalapi虽然是轻量级API框架但是和C底层框架也还是有一点的差距的

thinkphp就不用说了

**Yaf -> phalcon -> phalapi -> thinkphp**

### 功能:

phalcon当之无愧排在第一

phalapi应为没有View层所以在功能上并不如thinkphp

yaf连DB层都需要自己采用第三方的所以功能性上面最差

**phalcon -> thinkphp -> phalapi -> Yaf**

### 易用性:

phalapi是国人写的轻量级API框架针对API来说易用性最好

对于国人来说thinkphp针对性非常强所以易用性排在第二

yaf和phalcon基本不相上下

**phalapi -> thinkphp -> phalcon = Yaf**

### 国内社区支持:

在国内社区支持上面thinkphp当值无愧

phalapi虽然是新框架已经进入开源中国TOP 50 也很强给个第二名

yaf虽然是鸟哥写的框架可是占有率感人所以和phalcon打平

**thinkphp -> phalapi -> phalcon = Yaf**

## gulp

### 简介：

gulp是前端开发过程中一种基于流的代码构建工具，是自动化项目的构建利器；她不仅能对网站资源进行优化，而且在开发过程中很多重复的任务能够使用正确的工具自动完成；使用她，不仅可以很愉快的编写代码，而且大大提高我们的工作效率。

gulp是基于Nodejs的自动任务运行器， 她能自动化地完成 javascript、coffee、sass、less、html/image、css 等文件的测试、检查、合并、压缩、格式化、浏览器自动刷新、部署文件生成，并监听文件在改动后重复指定的这些步骤。在实现上，她借鉴了Unix操作系统的管道（pipe）思想，前一级的输出，直接变成后一级的输入，使得在操作上非常简单。

