

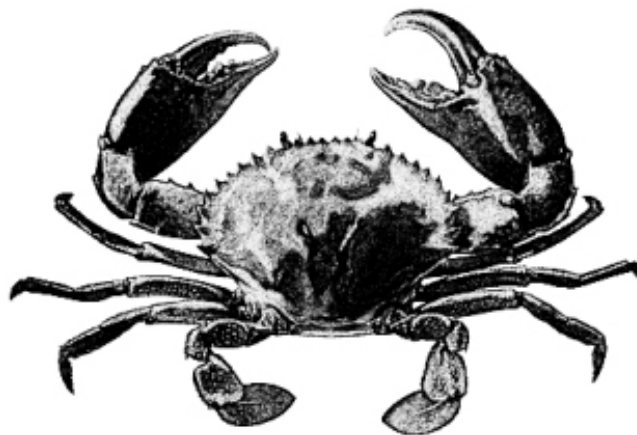
---

# Yaf(Yet Another Framework)用户手册

惠新宸

[<laruence at php.net>](mailto:laruence@php.net)

[Yaf官方网站](#) [在线文档](#) [报告Bug Yaf on PECL](#)



最后更新: \$Id: userguide.xml 1134 2011-03-24 13:10:09Z huixinchen \$

---

## 目录

### 前言

#### 1. 关于Yaf

[1.1. Yaf的特点](#)

[1.2. Yaf的优点](#)

[1.3. 流程图](#)

[1.4. Yaf的性能](#)

[对比](#)

[测试结果](#)

[1个并发](#)

[100个并发](#)

[说明](#)

#### 2. Yaf安装/配置

## 2.1. Yaf的安装

在 Linux 系统下安装

PHP 5.2+

在 Windows 系统下安装

PHP 5.2+

## 2.2. Yaf定义的常量

## 2.3. Yaf的配置项

## 3. 快速开始

### 3.1. 需要些什么?

### 3.2. Hello World

目录结构

入口文件

重写规则

配置文件

控制器

视图文件

运行

### 3.3. 使用代码生成工具

## 4. 配置文件

### 4.1. 必要的配置项

### 4.2. 可选的配置项

## 5. 自动加载器

### 5.1. 全局类和自身类(本地类)

### 5.2. 类的加载规则

## 6. 使用Bootstrap

### 6.1. 简介

### 6.2. 使用Bootstrap

## 7. 使用插件

### 7.1. 简介

### 7.2. Yaf支持的Hook

### 7.3. 定义插件

### 7.4. 注册插件

### 7.5. 插件目录

## 8. 路由和路由协议

### 8.1. 概述

### 8.2. 设计

### 8.3. 默认情况

### 8.4. 使用路由

### 8.5. 路由协议详解

默认路由协议

Yaf\_Route\_Simple

Yaf\_Route\_Supervar

Yaf\_Route\_Map

Yaf\_Route\_Rewrite

Yaf\_Route\_Regex

### 8.6. 自定义路由协议

## 9. 在命令行使用Yaf

### 9.1. 简介

### 9.2. 使用样例

### 9.3. 分发请求

## 10. 异常和错误

### 10.1. 概述

### 10.2. 异常模式

### 10.3. 错误模式

## 11. 内建的类

### 11.1. Yaf\_Application

Yaf\_Application::\_\_construct

Yaf\_Application::bootstrap

Yaf\_Application::app

Yaf\_Application::environ

Yaf\_Application::run

Yaf\_Application::execute

Yaf\_Application::getDispatcher

Yaf\_Application::getConfig

Yaf\_Application::getModules

### 11.2. Yaf\_Bootstrap\_Abstract

### 11.3. Yaf\_Loader

Yaf\_Loader::getInstance  
Yaf\_Loader::import  
Yaf\_Loader::autoload  
Yaf\_Loader::registerLocalNamespace  
Yaf\_Loader::isLocalName  
Yaf\_Loader::getLocalNamespace  
Yaf\_Loader::clearLocalNamespace

#### 11.4. Yaf\_Dispatcher

Yaf\_Dispatcher::getInstance  
Yaf\_Dispatcher::disableView  
Yaf\_Dispatcher::enableView  
Yaf\_Dispatcher::autoRender  
Yaf\_Dispatcher::returnResponse  
Yaf\_Dispatcher::flushInstantly  
Yaf\_Dispatcher::setErrorHandler  
Yaf\_Dispatcher::getApplication  
Yaf\_Dispatcher::getRouter  
Yaf\_Dispatcher::getRequest  
Yaf\_Dispatcher::registerPlugin  
Yaf\_Dispatcher::setAppDirectory  
Yaf\_Dispatcher::setRequest  
Yaf\_Dispatcher::initView  
Yaf\_Dispatcher::setView  
Yaf\_Dispatcher::setDefaultController  
Yaf\_Dispatcher::setDefaultModule  
Yaf\_Dispatcher::setDefaultAction  
Yaf\_Dispatcher::throwException  
Yaf\_Dispatcher::catchException  
Yaf\_Dispatcher::dispatch

#### 11.5. Yaf\_Plugin\_Abstract

#### 11.6. Yaf\_Registry

Yaf\_Registry::set  
Yaf\_Registry::get  
Yaf\_Registry::has  
Yaf\_Registry::del

### 11.7. Yaf\_Session

### 11.8. Yaf\_Config\_Abstract

Yaf\_Config\_Ini

Yaf\_Config\_Simple

### 11.9. Yaf\_Controller\_Abstract

Yaf\_Controller\_Abstract::getModuleName

Yaf\_Controller\_Abstract::getRequest

Yaf\_Controller\_Abstract::getResponse

Yaf\_Controller\_Abstract::getView

Yaf\_Controller\_Abstract::initView

Yaf\_Controller\_Abstract::setViewPath

Yaf\_Controller\_Abstract::getViewPath

Yaf\_Controller\_Abstract::render

Yaf\_Controller\_Abstract::display

Yaf\_Controller\_Abstract::forward

Yaf\_Controller\_Abstract::redirect

### 11.10. Yaf\_Action\_Abstract

### 11.11. Yaf\_View\_Interface

Yaf\_View\_Simple

Yaf\_View\_Simple::assign

Yaf\_View\_Simple::render

Yaf\_View\_Simple::display

Yaf\_View\_Simple::setScriptPath

Yaf\_View\_Simple::getScriptPath

Yaf\_View\_Simple::\_\_set

Yaf\_View\_Simple::\_\_get

Yaf\_View\_Simple::get

### 11.12. Yaf\_Request\_Abstract

Yaf\_Request\_Http

Yaf\_Request\_Simple

Yaf\_Request\_Abstract::getException

Yaf\_Request\_Abstract::getModuleName

Yaf\_Request\_Abstract::getControllerName

Yaf\_Request\_Abstract::getActionName

Yaf\_Request\_Abstract::getParams

Yaf\_Request\_Abstract::getParam  
Yaf\_Request\_Abstract::setParam  
Yaf\_Request\_Abstract::getMethod  
Yaf\_Request\_Abstract::isCli  
Yaf\_Request\_Abstract::isGet

#### 11.13. Yaf\_Response\_Abstract

Yaf\_Response\_Http  
Yaf\_Response\_Cli  
Yaf\_Response\_Abstract::setBody  
Yaf\_Response\_Abstract::appendBody  
Yaf\_Response\_Abstract::prependBody  
Yaf\_Response\_Abstract::getBody  
Yaf\_Response\_Abstract::clearBody  
Yaf\_Response\_Abstract::response  
Yaf\_Response\_Abstract::setRedirect  
Yaf\_Response\_Abstract::\_\_toString

#### 11.14. Yaf\_Router

Yaf\_Router::addRoute  
Yaf\_Config::addConfig  
Yaf\_Router::getRoutes  
Yaf\_Router::getRoute  
Yaf\_Router::getCurrentRoute  
Yaf\_Router::isModuleName  
Yaf\_Router::route

#### 11.15. Yaf\_Route\_Interface

#### 11.16. Yaf\_Exception

Yaf\_Exception\_StartupError  
Yaf\_Exception\_RouterFailed  
Yaf\_Exception\_DispatchFailed  
Yaf\_Exception\_LoadFailed  
Yaf\_Exception\_LoadFailed\_Module  
Yaf\_Exception\_LoadFailed\_Controller  
Yaf\_Exception\_LoadFailed\_Action  
Yaf\_Exception\_LoadFailed\_View  
Yaf\_Exception\_TypeError

---

[下一页](#)  
[前言](#)

---

## 前言

随着PHP的发展, PHP框架层出不穷, 但到底用不用PHP框架, 还存在很大的争论, 反对者认为使用框架会降低性能, 经常举例的就是Zend Framework. 而支持者则认为, 采用框架能提高开发效率, 损失点性能也是值得的.

而这些也正是公司内框架种类繁多的一个原因, 有的项目组为了性能而选择某些框架, 而另外一些项目组, 则为了更好的封装选择了另外的框架

那, 有没有俩全的办法呢? 也就是说, 有没有那么一个框架, 既不会有损性能, 又能提高开发效率呢.

Yaf, 就是为了这个目标而生的.

Yaf有着和Zend Framework相似的API, 相似的理念, 而同时又保持着对Bingo的兼容, 以此来提高开发效率, 规范开发习惯. 本着对性能的追求, Yaf把框架中不易变的部分抽象出来, 采用PHP扩展实现(c语言), 以此来保证性能. 在作者自己做的[简单测试](#)中, Yaf和原生的PHP在同样功能下, 性能损失小于10%, 而和Zend Framework的对比中, Yaf的性能是Zend Framework的50-60倍.

---



• Home

• PHP源码分析


• PHP应用

• JS/CSS


• 随笔


• 留言


• 博客声明



风雪之隅







PHP语言, PHP扩展, Zend引擎相关的研究,技术,新闻分享 – 左手代码 右手诗

14 Jun 12 PHP的Calling Scope

昨天在Yaf交流群, 大草原同学批评我变懒了, Blog很久没更新了, 今天刚好有人在Segmentfalut上问了我一个问题, 我在微博上也做了简单的解答, 不过感觉一句话说不清楚, 就写篇blog凑个数吧. 😊

问题在这里, 因为太长, 我就不copy过来了: [这是php中 \\_call和 \\_callStatic在被继承后会产生bug?](#)

这个问题乍看, 确实很容易让人迷惑, 但实际上, 造成这样的误解的根本原因在于: 在PHP中, 判断静态与否不是靠"::"(PAAMAYIM\_NEKUDOTAYIM)符号, 而是靠calling scope.

那么, 什么是calling scope?

Filed in [PHP应用](#) with [14 Comments](#)

02 May 12 让PHP更快的提供文件下载

一般来说, 我们可以通过直接让URL指向一个位于Document Root下面的文件, 来引导用户下载文件.

但是, 这样做, 就没办法做一些统计, 权限检查, 等等的工作. 于是, 很多时候, 我们采用让PHP来做转发, 为用户提供文件下载.

Filed in [PHP应用](#) with [36 Comments](#)

30 Apr 12 如何为PHP贡献代码

PHP在之前把源代码迁移到了git下管理, 同时也在github(<https://github.com/php/php-src>)上做了镜像, 这样一来, 就方便了更多的开发者为PHP来贡献代码.










**Larurence**  
PHP开发组成员, PECL开发者. Yaf, Taint等Pecl扩展作者.

Advanced Random Posts

-  [PFA - PHP for Android](#)
-  [HTTP 204和205的应用](#)
-  [PHP数组的Hash冲突实例](#)
-  [如何调试PHP的Core之获取基本信息](#)
-  [PHP调试技术手册发布\(1.0.0 pdf\)](#)

Recent Comments

-  vergil on [Yaf-一个PHP扩展实现的PHP框架](#)
-  [PHP+MySQL手工注入技术 | 包头黑客网络技术博客](#) on [PHP5.4的新特性](#)
-  [PHP+MySQL手工注入技术 | 包头黑客网络技术博客](#) on [PHP受locale影响的函数](#)
-  Yaseng on [我对PHP5.4的一个改进](#)
-  webguo在路上 » [PHP哈希表碰撞攻击原理](#) on [PHP5.2.\\*防止Hash冲突拒绝服务攻击的Patch](#)
-  [SQL注入攻击之 mysql\\_set\\_charset | 包头黑客网络技术博客](#) on [深入理解SET NAMES和mysql\(i\)\\_set\\_charset的区别](#)
-  funlake on [PHP的Calling Scope](#)
-  [PHP爱好者](#) on [PHP的Calling Scope](#)
-  scncpb on [留言](#)
-  [IE 下 JQUERY AJAX 请求 官方原理 | 南龙的小站](#) on [浏览器缓存机制](#)

今天写这篇文章, 就是为了给在国内的同学们, 愿意为PHP开源社区做贡献的同学们, 做个示例, 如何为PHP来贡献你的智慧.

Filed in [PHP应用](#), [随笔](#) with [7 Comments](#)

### 01 Apr 12 **PHP对程序员的要求更高**

今天是愚人节, 但我这个文章标题可不是和大家开玩笑. 😊

首先, 大家都知道, PHP也是一种编译型脚本语言, 和其他的预编译型语言不同, 它不是编译成中间代码, 然后发布.. 而是每次运行都需要编译..

为此, 也就有了一些Opcode Cache, 比如APC, 比如eacc. 还有Zend O+.

那么为什么PHP不把编译/执行分开呢?

Filed in [PHP应用](#), [随笔](#) with [38 Comments](#)

### 18 Feb 12 **Taint-0.3.0(A XSS codes sniffer) released**

最近几天忙里偷闲, 一直在完善taint, 今天我觉得终于算做到了80%的满意了, 根据80:20原则, 我觉得可以做为一个里程碑的版本了 😊.

什么是Taint? An extension used for detecting XSS codes(tainted string), And also can be used to spot sql injection vulnerabilities, shell inject, etc.

经过我实际测试, [Taint-0.3.0](#)能检测出实际的一些开源产品的(别问是什么)隐藏的XSS code, SQL注入, Shell注入等漏洞, 并且这些漏洞如果要用静态分析工具去排查, 将会非常困难, 比如对于如下的例子:

```
<?php
$name = $_GET["name"];
$value = strval($_GET["tainted"]);

echo $$name;
```

对于请求:

```
http://****.com/?name=value&tainted=xxx
```

静态分析工具, 往往无能为力, 而Taint却可以准确无误的爆出这类型问题.

Filed in [PHP Extension](#), [PHP应用](#), [PHP源码分析](#) with [38 Comments](#)

#### Tags

Apache browser bug C++ charset COOKIE core c写PHP扩展 debug encoding engine Extension GET IE javascript js json Module mysql namespace nginx **PHP** php5.4 PHP5.4新特性 PHP extension php原理 PHP应用 PHP扩展 php源码 php源码分析 SAPI session utf8 variable vim Yaf Zend/PHP 乱码 作用域 原理 开发php扩展 性能 扩展PHP 扩展开发 正则 GNU C/C++ (3) Js/CSS (24) Linux/Unix (15) MySQL/PostgreSQL (7) PHP Extension (16) PHP应用 (146) PHP源码分析 (75) 转载 (30) 随笔 (70)

WP Cumulus Flash tag cloud by [Roy Tanck](#) and [Luke Morton](#) requires [Flash Player](#) 9 or better.

#### Projects

Yaf(用PHP扩展实现的PHP框架)  
Plua(Lua for PHP)  
Mpass(PHP多进程Socket解决方案)  
bTwitter(Twitter Bash客户端)  
PHP Opcode Dumper  
Info Store Platform  
Eve JsLoader  
Pallas PHP CMS

#### friends

-  [80Sec](#)
-  [cc0cc](#)
-  [CFC4N](#)
-  [Errorrik](#)
-  [glemir's](#)
-  [lterse's BLOG](#)
-  [Jessica](#)
-  [pplxh](#)
-  [rainX](#)
- 

### 14 Feb 12 PHP Taint – 一个用来检测XSS/SQL/Shell注入漏洞的扩展

之前, 小顿和我提过一个想法, 就是从PHP语言层面去分析,找出一些可能的注入漏洞代码. 当时我一来没时间, 而来也确实不知道从何处下手..

直到上周的时候, 我看到了这个RFC: [RFC:Taint](#).

但是这个RFC的问题在于, 它需要为PHP打Patch, 修改了PHP本身的数据结构, 这对于以后维护, 升级PHP来说, 很不方便, 也会有一些隐患.

虽然这样, 但这个RFC却给了我一个启发, 于是我就完成了这样的一个扩展:[Taint Extension](#)

Filed in [PHP Extension](#), [PHP应用](#), [PHP源码分析](#) with [44 Comments](#)

### 08 Feb 12 PHP-5.3.9远程执行任意代码漏洞(CVE-2012-0830)

还记得我之前说的[PHP Hash Collisions Ddos](#)漏洞吧? 最初的时候, 开发组给出的修复方案, 采用的是如果超过max\_input\_vars, 就报错(E\_ERROR), 继而导致PHP出错结束. 而后来, 为了更加轻量级的解决这个问题, 我们又改善了一下, 变成了如果超过max\_input\_vars, 就发出警告(E\_WARNING), 并且不再往目的数组添加, 但是流程继续. 然后我们发布了5.3.9.

这个新的修复方法初衷是好的, 但是却带来一个严重的问题(5.3.10中已经修复), 这个问题最初是由Stefan Esser发现的. 请看之前(5.3.9)最终的修复方案([php\\_register\\_variable\\_ex](#)):

Filed in [PHP应用](#), [PHP源码分析](#) with [7 Comments](#)

### 02 Feb 12 我们什么时候应该使用异常?

先说个题外话: 在公司做了两件事, 是我觉得很有意义的, 第一就是成立了一个PHP邮件组, 第二就是成立了一个Hi群. 目前俩者都有超过500 phpers在里面. 我一直认为, 构建一个交流平台, 让同学们能顺畅, 简单的沟通, 是营造积极的技术学习氛围的基础和前提. 让每个人的问题不会成为别人的问题, 则是最直接的利益.

昨天, 有同事在邮件组提了个问题:

  PHP应该什么时候使用 Exception ? 它的性能如何?

这个问题也算是一个久经争论的经典问题了. 我谈谈我的个人看法.

Filed in [PHP应用](#), [随笔](#) with [28 Comments](#)

 [Sara Golemon](#)

 [siko](#)

 [stauren](#)

 [Think in code](#)


 [冰的河](#)

 [刘青炎](#)

 [周翔's blog](#)


 [天韵之星](#)

 [思考的Pyt](#)

 [抚琴居](#)

 [神仙](#)

 [风吹倒的男人](#)

 [黑夜路人](#)

### Visitor ClustrMaps



01 Feb 12 使用exit(-1)为什么得到255退出码?

今天有人在[微博](#)上问了一个问题, 使用:

```
string exec ( string $command [, array &$output [, int &$return_var ]] )
```

调用一个程序, 程序退出-1, 但是PHP得到的为什么是255?

Filed in [PHP应用](#) with [6 Comments](#)

24 Jan 12 大家新年好~

写博客3年了, 也借此认识了不少朋友.

到今天, 博客的google订阅数3,870, 那就祝愿所有订阅我博客的同学们, 身体永远健康, 所有不经意来到我博客的同学们, 身体比较健康!!

龙年大吉.

thanks

Filed in [随笔](#) with [17 Comments](#)

---

## 第 1 章 关于Yaf

Yaf是一个C语言编写的PHP框架

### 1.1. Yaf的特点



重要

剑的三层境界：一是手中有剑，心中亦有剑；二是手中无剑，心中有剑；三是手中无剑，心中亦无剑

在和其他用PHP写的PHP框架来比的话，Yaf就是剑的第二层境界。框架不在你手中，而在PHP的"心"中。

目前PHP的框架层出不穷，其中不乏很多优秀的框架，比如Zend官方支持的Zend Framework, Yii, ci等等。但在这繁多的框架也就造成了公司内多种框架的业务产品。这些框架之间的不同，也就导致了多种版本的类库，框架，约定，规范，，，，

那么，为什么现在开源社区没有一个成熟的用PHP扩展开发的框架呢？

用PHP扩展写PHP框架的难点

1. 难于开发. 要完成一个PHP扩展的PHP框架，需要作者有C背景，有PHP扩展开发背景，更要有PHP框架的设计经验。
2. 目标用户群小. 现在国内很多中小型站都是使用虚拟主机，并不能随意的给PHP添加扩展，所以这些大部分的中小型企业，个人博客的用户就无法使用。
3. 维护成本高. 要维护PHP扩展，不仅仅需要精通于C的开发和调试，更要精通于Zend API，并且升级维护的周期也会很长。

那既然这样，为什么还要用PHP扩展来开发框架呢，或者说，这可行么？

用PHP扩展写PHP框架的可行性

1. 扩展逻辑相对比较稳定, 一般不易变化. 把它们抽象出来, 用扩展实现, 不会带来额外的维护负担.
2. 框架逻辑复杂, 自检耗时耗内存都比较可观, 而如果用扩展来实现, 就能大幅减少这部分对资源的消耗.

---

[上一页](#)

[前言](#)

[起始页](#)

[下一页](#)

**1.2. Yaf的优点**

---

## 1.2. Yaf的优点



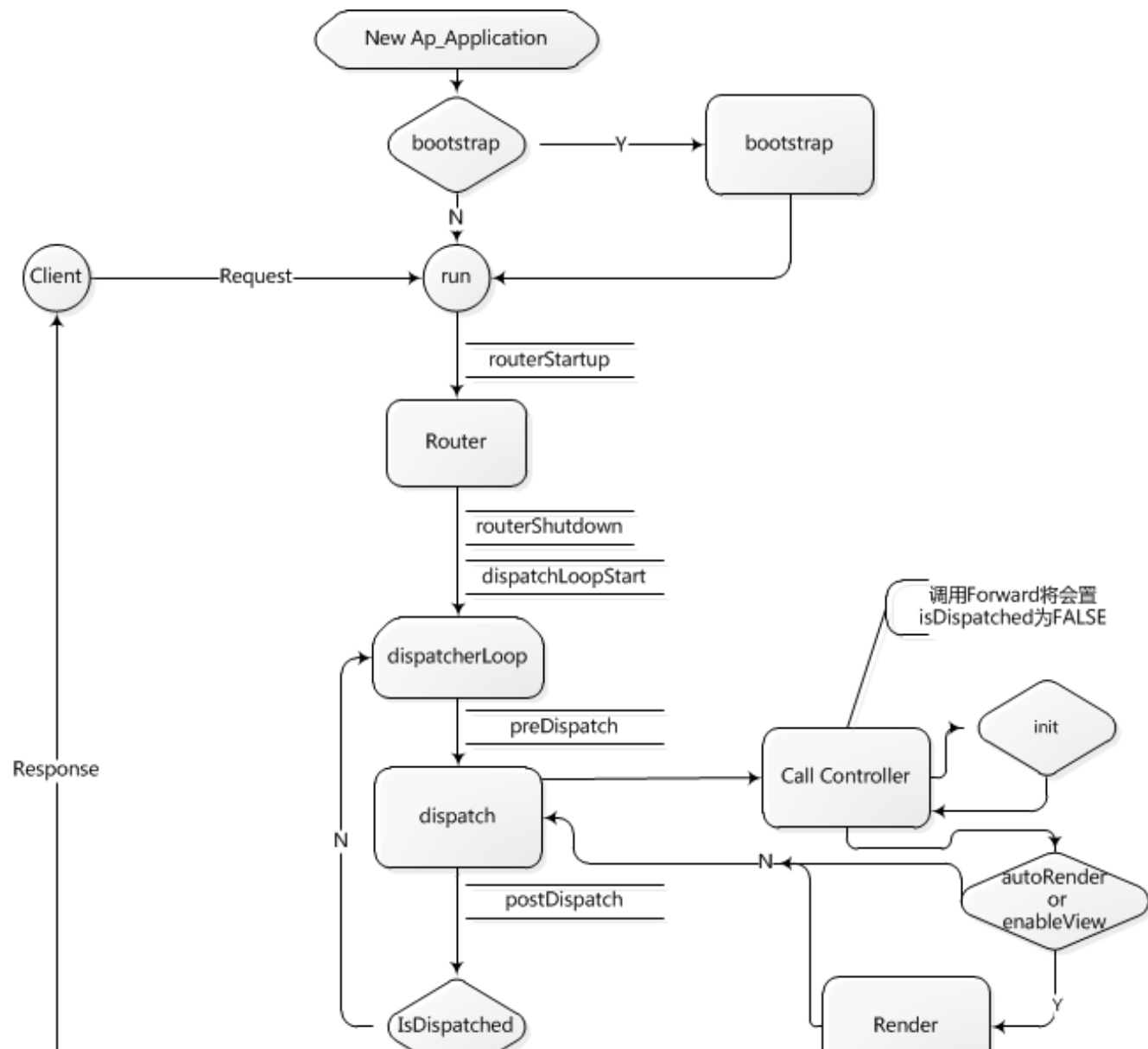
重要

天下武功无坚不破，唯快不破

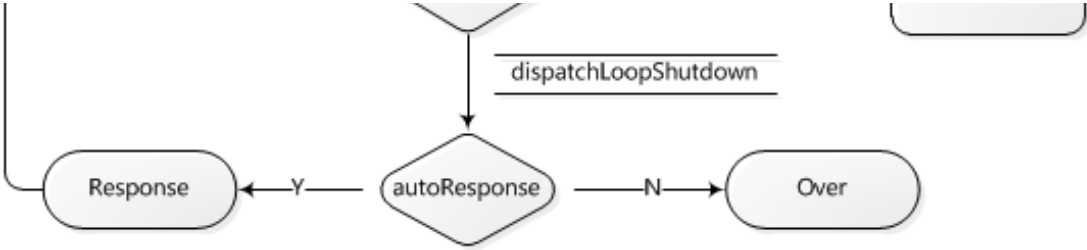
1. 用C语言开发的PHP框架，相比原生的PHP，几乎不会带来额外的性能开销.
2. 所有的框架类，不需要编译，在PHP启动的时候加载，并常驻内存.
3. 更短的内存周转周期，提高内存利用率，降低内存占用率.
4. 灵巧的自动加载. 支持全局和局部两种加载规则，方便类库共享.
5. 高性能的视图引擎.
6. 高度灵活可扩展的框架，支持自定义视图引擎，支持插件，支持自定义路由等等.
7. 内建多种路由，可以兼容目前常见的各种路由协议.
8. 强大而又高度灵活的配置文件支持. 并支持缓存配置文件，避免复杂的配置结构带来的性能损失.
9. 在框架本身,对危险的操作习惯做了禁止.
10. 更快的执行速度, 更少的内存占用.

## 1.3. 流程图

Yaf提供了完善的API, 并支持Bootstrap和插件机制. 整体流程图如下:







[上一页](#)

1.2. Yaf的优点

[上一级](#)

起始页

[下一页](#)

1.4. Yaf的性能

## 1.4. Yaf的性能

### 对比

为了极限的看出Yaf框架的性能如何, 作者并没有和其他框架做对比, 而是和原生的PHP做对比, 测试代码如下:

#### 例 1.1. 1.测试用原生的PHPorig.php

```
<?php
class IndexController {
    public function actionIndex() {
        echo "Larurence";
    }
}

$controller = new IndexController();
$controller->actionIndex();
?>
```

#### 例 1.2. 2.测试用的Yaf的入口文件ap.php

```
<?php
$conf = array(
    "application.directory" => "/home/larurence/local/www/htdocs/ap",
);

$app = new Yaf_Application($conf);
$app->run();
```

#### 例 1.3. 2.测试用的Yaf的默认控制器Index.php

```
<?php
class IndexController extends Yaf_Controller {
    public function actionIndex() {
        $this->disableView(); //关闭视图输出
        echo "Larurence";
    }
}
?>
```

### 测试结果

作者简单的采用了ab作为测试工具, 分别在并发1, 100, 200的情况下对两者做了测试.

## 1个并发

## 例 1.4. 请求1000次, 原生的PHP

```
$ ./ab -n100 -c1 http://127.0.0.1/orig.php
```

Document Path: orig.php  
Document Length: 8 bytes

Concurrency Level: 1  
Time taken for tests: 0.463 seconds  
Complete requests: 1000  
Failed requests: 0  
Write errors: 0  
Total transferred: 130000 bytes  
HTML transferred: 8000 bytes  
Requests per second: 2159.41 [#/sec] (mean)  
Time per request: 0.463 [ms] (mean)  
Time per request: 0.463 [ms] (mean, across all concurrent requests)  
Transfer rate: 274.14 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 0.0	0	0
Processing:	0	0 0.2	0	5
Waiting:	0	0 0.2	0	5
Total:	0	0 0.2	0	5

Percentage of the requests served within a certain time (ms)

50%	0
66%	0
75%	0
80%	0
90%	0
95%	0
98%	0
99%	1
100%	5 (longest request)

## 例 1.5. 请求100次, Yaf

```
$ ./ab -n100 -c1 http://127.0.0.1/ap/index.php
```

Document Path: /ap/index.php  
Document Length: 8 bytes

Concurrency Level: 1  
Time taken for tests: 0.525 seconds  
Complete requests: 1000  
Failed requests: 0  
Write errors: 0  
Total transferred: 130000 bytes  
HTML transferred: 8000 bytes  
Requests per second: 1906.24 [#/sec] (mean)  
Time per request: 0.525 [ms] (mean)  
Time per request: 0.525 [ms] (mean, across all concurrent requests)  
Transfer rate: 242.00 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 0.0	0	0
Processing:	0	0 0.3	0	7

```

Waiting:      0    0    0.3    0    7
Total:        0    0    0.3    1    7
ERROR: The median and mean for the total time are more than twice the standard
       deviation apart. These results are NOT reliable.

```

Percentage of the requests served within a certain time (ms)

```

50%    1
66%    1
75%    1
80%    1
90%    1
95%    1
98%    1
99%    1
100%   7 (longest request)

```

## 100个并发

### 例 1.6. 请求1000次, 原生的PHP

```

$ ./ab -n1000 -c100 http://127.0.0.1/orig.php

Document Path:      orig.php
Document Length:    8 bytes

Concurrency Level:   100
Time taken for tests: 0.287 seconds
Complete requests:   1000
Failed requests:     0
Write errors:        0
Total transferred:   130000 bytes
HTML transferred:    8000 bytes
Requests per second: 3478.82 [#/sec] (mean)
Time per request:    28.745 [ms] (mean)
Time per request:    0.287 [ms] (mean, across all concurrent requests)
Transfer rate:       441.65 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      0    0    1.0     0     6
Processing:    5   27   4.8    27    35
Waiting:       5   27   4.8    27    35
Total:         6   27   4.6    27    36

Percentage of the requests served within a certain time (ms)
50%    27
66%    28
75%    29
80%    31
90%    35
95%    35
98%    35
99%    35
100%   36 (longest request)

```

### 例 1.7. 请求1000次, Yaf

```

$ ./ab -n1000 -c100 http://127.0.0.1/ap/index.php

```

```

Document Path:      /ap/index.php
Document Length:    8 bytes

Concurrency Level:   100
Time taken for tests: 0.316 seconds
Complete requests:   1000
Failed requests:     0
Write errors:        0
Total transferred:   130000 bytes
HTML transferred:    8000 bytes
Requests per second: 3165.24 [#/sec] (mean)
Time per request:    31.593 [ms] (mean)
Time per request:    0.316 [ms] (mean, across all concurrent requests)
Transfer rate:       401.84 [Kbytes/sec] received

```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 1.0	0	6
Processing:	6	30 5.6	27	44
Waiting:	6	30 5.6	27	44
Total:	6	30 5.6	27	44

#### Percentage of the requests served within a certain time (ms)

50%	27
66%	32
75%	34
80%	36
90%	37
95%	40
98%	42
99%	42
100%	44 (longest request)

## 说明

在测试的过程中, 通过vmstat观察, 机器的Idle一直保持在50-60左右.

总体来看, Yaf的性能比起原生的PHP, 损失的程度在10%左右, 另外考虑到因为Yaf有一次IO操作(载入Controller), 而原生的PHP并没有, 那么基本可以认为使用了Yaf框架以后, 性能损失在10%以内.



### 重要

在测试的过程中, 并没有多次挑选最好的数据来把测试结果弄的漂亮点. 因为有一些时候, Yaf的性能几乎和原生的PHP的性能差别在2%以内.

最后, 要说明一点: 测试结果, 只是一个简单的说明, 并不是为了证明什么结论. 因为框架的时间和真正应用逻辑的耗时比起来, 真的是很小的一部分.

[上一页](#)
[1.3. 流程图](#)
[上一级](#)
[起始页](#)
[下一页](#)
[第 2 章 Yaf安装/配置](#)

## 第 2 章 Yaf安装/配置

### 2.1. Yaf的安装

Yaf只支持PHP5.2及以上的版本. 并支持最新的PHP5.3.3

Yaf需要SPL的支持. SPL在PHP5中是默认启用的扩展模块

Yaf需要PCRE的支持. PCRE在PHP5中是默认启用的扩展模块

在 **Linux** 系统下安装

#### PHP 5.2+

下载Yaf的最新版本, 解压缩以后, 进入Yaf的源码目录, 依次执行(其中**PHP\_BIN**是PHP的bin目录):

##### 例 2.1. 编译Yaf

```
$PHP_BIN/phpize
./configure --with-php-config=$PHP_BIN/php-config
make
make install
```

然后在php.ini中载入ap.so, 重启PHP.

Yaf\_Request\_Abstract的getPost, getQuery等方法, 并没有对应的setter方法. 并且这些方法是直接从PHP内部的\$\_POST, \$\_GET等大变量的原身变量只读的查询值, 所以就有一个问题: 通过在PHP脚本中对这些变量的修改, 并不能反映到getPost/getQuery等方法上.

##### 例 2.2. POST变量只读

```
<?php
class Index_Controller extends Yaf_Controller_Abstract {
    public function indexAction() {
        $_POST['name'] = "new_name";
        //此时对POST的修改, 并不能反映到getPost上
        echo $this->getRequest()->getPost("name"); //old_name
    }
}
```

当然, 这样的设计是经过深思熟虑的, 也可以不依赖PHP的variable\_orders的配置, 但是带来一个问题就是, QA和Rd无法通过修改这些变量来做测试数据.

所以, Yaf提供了一个debug模式, 在这个模式下, `getPost/getQuery/getServer/getCookie`将从符号表中的对应变量查询值.从而可以让我们直接对PHP的超级变量做的修改能反映到对应的Yaf\_Request\_Abstract的方法上.

**重要**

请不要在正式的环境中, 以debug模式编译Yaf, 一个原因是这样做会有性能损耗, 第二个原因是这样做, 有悖于`$_POST`等大变量的只读特性.

为了提醒大家这一点, 这种模式下在Yaf\_Application的构造函数中会触发一个E\_STRICT的提示:

Strict Standards: you are running ap in debug mode

**例 2.3. 以debug模式编译Yaf**

```
$PHP_BIN/phpize
./configure --enable-ap-debug --with-php-config=$PHP_BIN/php-config
make
make install
```

**在 Windows 系统下安装****PHP 5.2+**

c:\php\ext。

**注意**

PHP5.3开始已经不支持VC6的编译, 目前只能提供PHP5.3以上版本使用的php\_yaf.dll, 如有需要, 请联系Larurence

[上一页](#)[下一页](#)[1.4. Yaf的性能](#)[起始页](#)[2.2. Yaf定义的常量](#)

## 2.2. Yaf定义的常量

表 2.1.

常量(启用命名空间后的常量名)	说明
YAF_VERSION(Yaf\VERSION)	Yaf框架的三位版本信息
YAF_ENVIRON(Yaf\ENVIRON)	Yaf的环境常量, 指明了要读取的配置的节, 默认的是product
YAF_ERR_STARTUP_FAILED(Yaf\ERR\STARTUP_FAILED)	Yaf的错误代码常量, 表示启动失败, 值为512
YAF_ERR_ROUTE_FAILED(Yaf\ERR\ROUTE_FAILED)	Yaf的错误代码常量, 表示路由失败, 值为513
YAF_ERR_DISPATCH_FAILED(Yaf\ERR\DISPATCH_FAILED)	Yaf的错误代码常量, 表示分发失败, 值为514
YAF_ERR_NOTFOUND_MODULE(Yaf\ERR\NOTFOUD\MODULE)	Yaf的错误代码常量, 表示找不到指定的模块, 值为515
YAF_ERR_NOTFOUND_CONTROLLER(Yaf\ERR\NOTFOUD\CONTROLLER)	Yaf的错误代码常量, 表示找不到指定的Controller, 值为516
YAF_ERR_NOTFOUND_ACTION(Yaf\ERR\NOTFOUD\ACTION)	Yaf的错误代码常量, 表示找不到指定的Action, 值为517




YAF_ERR_NOTFOUND_VIEW(Yaf\ERR\NOTFOUD\VIEW)	Yaf的错误代码常量, 表示找不到指定的视图文件, 值为 <b>518</b>
YAF_ERR_CALL_FAILED(Yaf\ERR\CALL_FAILED)	Yaf的错误代码常量, 表示调用失败, 值为 <b>519</b>
YAF_ERR_AUTOLOAD_FAILED(Yaf\ERR\AUTOLOAD_FAILED)	Yaf的错误代码常量, 表示自动加载类失败, 值为 <b>520</b>
YAF_ERR_TYPE_ERROR(Yaf\ERR\TYPE_ERROR)	Yaf的错误代码常量, 表示关键逻辑的参数错误, 值为 <b>521</b>

## 2.3. Yaf的配置项

表 2.2. Yaf 配置选项

选项名称	默认值	可修改范围	更新记录
yaf.environ	product	PHP_INI_ALL	环境名称, 当用INI作为Yaf的配置 文件时, 这个指明了Yaf将要在 INI配置中读取的节的名字
yaf.library	NULL	PHP_INI_ALL	全局类库的目录路径
yaf.cache_config	0	PHP_INI_SYSTEM	是否缓存配置文件(只针对INI配 置文件生效), 打开此选项可在复 杂配置的情况下提高性能
yaf.name_suffix	1	PHP_INI_ALL	在处理Controller, Action, Plugin, Model的时候, 类名中关 键信息是否是后缀式, 比如 UserModel, 而在前缀模式下则 是ModelUser
yaf.name_separator	""	PHP_INI_ALL	在处理Controller, Action, Plugin, Model的时候, 前缀和名 字之间的分隔符, 默认为空, 也就 是UserPlugin, 加入设置为"_", 则判断的依据就会变 成:"User_Plugin", 这个主要是 为了兼容ST已有的命名规范
yaf.forward_limit	5	PHP_INI_ALL	forward最大嵌套深度
yaf.use_namespace	0	PHP_INI_SYSTEM	开启的情况下, Yaf将会使用命名 空间方式注册自己的类, 比如 Yaf_Application将会变成Yaf \Application
yaf.use_spl_autoload	0	PHP_INI_ALL	开启的情况下, Yaf在加载不成功 的情况下, 会继续让PHP的自动加 载函数加载, 从性能考虑, 除非特 殊情况, 否则保持这个选项关闭



警告

在开启`yaf.cache_config`的情况下, Yaf会使用INI文件路径作为Key, 这就有一个陷阱, 就是如果在一台服务器上同时运行俩个应用, 那么它们必须不能使用同一个路径名下的INI配置文件, 否则就会出现Application Path混乱的问题. 所以, 尽量不要使用相对路径.

---

[上一页](#)[2.2. Yaf定义的常量](#)[上一级](#)[起始页](#)[下一页](#)[第 3 章 快速开始](#)

## 第 3 章 快速开始

本章将给出关于 **Yaf** 的快速使用手册.

### 3.1. 需要些什么?

假设**Yaf**已经正确编译, 安装.

---

## 3.2. Hello World

### 目录结构

对于Yaf的应用, 都应该遵循类似下面的目录结构.

例 3.1. 一个典型的目录结构

```
- index.php //入口文件
- .htaccess //重写规则
+ conf
  |- application.ini //配置文件
application/
  + controllers
    - Index.php //默认控制器
  + views
    |+ index //控制器
    - index.phtml //默认视图
  + modules //其他模块
  - library
  - models //model 目录
  - plugins //插件目录
```

### 入口文件

入口文件是所有请求的入口, 一般都借助于rewrite规则, 把所有的请求都重定向到这个入口文件.

例 3.2. 一个经典的入口文件`application/index.php`

```
<?php
define("APP_PATH",  dirname(__FILE__));
$app = new Yaf_Application(APP_PATH . "/conf/application.ini");
$app->run();
```

### 重写规则

除非我们使用基于query string的路由协议([Yaf\\_Route\\_Simple](#), [Yaf\\_Route\\_Supervar](#)), 否则我们就需要使用WebServer提供的Rewrite规则, 把所有这个应用的请求, 都定向到上面提到的入口文件.

**例 3.3. Apache的Rewrite**

```
#.htaccess, 当然也可以写在httpd.conf
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php
```

**例 3.4. Nginx的Rewrite**

```
server {
    listen ****;
    server_name domain.com;
    root document_root;
    index index.php index.html index.htm;

    if (!-e $request_filename) {
        rewrite ^/(.*) /index.php/$1 last;
    }
}
```

**例 3.5. Lighttpd的Rewrite**

```
$HTTP["host"] =~ "(www.)?domain.com$" {
    url.rewrite = (
        "^/(.+)/?$" => "/index.php/$1",
    )
}
```

**注意**

每种Server要启用Rewrite都需要特别设置, 如果对此有疑问.. RTFM

**配置文件**

在Yaf中, 配置文件支持继承, 支持分节, 并对PHP的常量进行支持. 你不用担心配置文件太大造成解析性能问题, 因为Yaf会在第一个运行的时候载入配置文件, 把格式化后的内容保持在内存中. 直到配置文件有了修改, 才会再次载入.

**例 3.6. 一个简单的配置文件application/conf/application.ini**

```
[product]
;支持直接写PHP中的已定义常量
application.directory=APP_PATH "/application/"
```

## 控制器

在Yaf中, 默认模块/控制器/动作, 都是以Index命名的, 当然, 这是可通过配置文件修改的.

对于默认模块, 控制器的目录是在application目录下的controllers目录下, Action的命名规则是"名字+Action"

### 例 3.7. 默认控制器application/controllers/Index.php

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() { //默认Action
    }
}
?>
```

## 视图文件

Yaf支持简单的视图引擎, 并且支持用户自定义自己的视图引擎, 比如Smarty.

对于默认模块, 视图文件的路径是在application目录下的views目录中以小写的action名的目录中.

### 例 3.8. 一个默认Action的视图application/views/index/index.phtml

```
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    Hello World!
</body>
</html>
```

## 运行

### 例 3.9. 在浏览器输入

```
http://www.yourhostname.com/application/index.php
```

看到了Hellow World输出吧？



注意

我没有看到Hello world

如果没有看到, 那么请查看PHP的错误日志, 找出问题在哪里.

---

[上一页](#)

[第 3 章 快速开始](#)

[上一级](#)

[起始页](#)

[下一页](#)

[3.3. 使用代码生成工具](#)



---

## 3.3. 使用代码生成工具

Yaf提供了代码生成工具, 所以也可以通过使用代码生成工具yaf\_cg来完成这个简单的入门 Demo

### 例 3.10. 代码生成工具的使用

```
php-ap-src/tools/cg/yaf_cg sample
```

将得到的sample目录, 拷贝到Webserver的documentRoot目录下

然后访问:


### 例 3.11. 在浏览器输入

```
http://www.yourhostname.com/sample/
```

# 第 4 章 配置文件

## 4.1. 必要的配置项

Yaf和用户共用一个配置空间，也就是在Yaf\_Application初始化时刻给出的配置文件中的配置。作为区别，Yaf的配置项都以ap开头。Yaf的核心必不可少的配置项只有一个(其实，这个也可以有默认参数，但是作者觉得完全没有配置，显得太寒酸了)。

 注意

Yaf通过在不同的环境中，选取不同的配置节，再结合配置可继承，来实现一套配置适应多种环境(线上,测试,开发)。

表 4.1. Yaf不可缺少的配置项

名称	值类型	说明
application.directory	String	应用的绝对目录路径

## 4.2. 可选的配置项

表 4.2. Yaf可选配置项

名称	值类型	默认值	说明
application.ext	String	php	PHP脚本的扩展名
application.bootstrap	String	Bootstrapapplication.php	Bootstrap路径(绝对路径)
application.library	String	application.directory + "/library"	本地(自身)类库的绝对目录地址
application.baseUri	String	NULL	在路由中, 需要忽略的路径前缀, 一般不需要设置, Yaf会自动判断.
application.dispatcher.defaultModule	String	index	默认模块
application.dispatcher.throwException	Bool	True	在出错的时候, 是否抛出异常
application.dispatcher.catchException	Bool	False	是否使用默认异常捕获Controller, 如果开启, 在有未捕获的异常的时候, 控制权会交给ErrorController的errorAction方法, 可以通过\$request->getException()获得此异常对象
application.dispatcher.defaultController	String	index	默认控制器
application.dispatcher.defaultAction	String	index	默认动作
application.view.ext	String	phtml	视图模板扩展名

---

[上一页](#)  
第 4 章 配置文件

[上一级](#)  
[起始页](#)

[下一页](#)  
第 5 章 自动加载器

# 第 5 章 自动加载器

Yaf在自启动的时候, 会通过SPL注册一个自己的Autoloader, 出于性能考虑, 对于框架相关的MVC类, Yaf Autoloader只以目录映射的方式尝试一次.

具体的目录映射规则如下:

表 5.1. Yaf目录映射规则

类型	后缀(或者前缀, 可以通过php.ini中ap.name_suffix来切换)	映射路径
控制器	Controller	默认模块下为{项目路径}/controllers/, 否则为{项目路径}/modules/{模块名}/controllers/
数据模型	Model	{项目路径}/models/
插件	Plugin	{项目路径}/plugins/

而对于非框架MVC相关的类, Yaf支持全局类和自身类的两种加载方式, 并且Yaf支持大小写敏感和不敏感两种方式来处理文件路径.

## 5.1. 全局类和自身类(本地类)

Yaf为了方便在一台服务器上部署的不同产品之间共享公司级别的共享库, 支持全局类和本地类两种加载方式.

全局类是指, 所有产品之间共享的类, 这些类库的路径是通过ap.library在php.ini(当然, 如果PHP在编译的时候, 支持了with-config-file-scan-dir, 那么也可以写在单独的ap.ini中)

而本地类是指, 产品自身的类库, 这些类库的路径是通过在产品的配置文件中, 通过ap.library配置的.

在Yaf中, 通过调用Yaf\_Loader的registerLocalNamespace方法, 来申明那些类前缀是本地类, 即可.



### 注意

在`use_spl_autoload`关闭的情况下, Yaf Autoloader在一次找不到的情况下, 会立即返回, 而剥夺其后的自动加载器的执行机会.

---

[上一页](#)

[4.2. 可选的配置项](#)

[起始页](#)

[下一页](#)

[5.2. 类的加载规则](#)

## 5.2. 类的加载规则

而类的加载规则，都是一样的：Yaf规定类名中必须包含路径信息，也就是以下划线"\_"分割的目录信息。Yaf将依照类名中的目录信息，完成自动加载。如下的例子，在没有申明本地类的情况下：

### 例 5.1. 一个映射的例子Zend\_Dummy\_Foo

```
//Yaf将在如下路径寻找类Foo_Dummy_Bar  
{类库路径(PHP.INI 中指定的ap. library)}/Foo/Dummy/Bar.php
```

而，如果通过如下方式调用了registerLocalNamespace：

### 例 5.2. 注册本地类

```
//申明，凡是以Foo和Local 开头的类，都是本地类  
$loader = Yaf_Loader::Instance();  
$loader->registerLocalNamespace(array("Foo", "Local"));
```

那么对于刚才的例子，将会在如下路径寻找Foo\_Dummy\_Bar

### 例 5.3. 一个映射的例子Zend\_Dummy\_Foo

```
//Yaf将在如下路径寻找类Foo_Dummy_Bar  
{类库路径(conf/application.ini 中指定的ap. library)}/Foo/Dummy/Bar.php
```

## 第 6 章 使用Bootstrap

### 6.1. 简介

**Bootstrap**, 也叫做引导程序. 它是Yaf提供的一个全局配置的入口, 在Bootstrap中, 你可以做很多全局自定义的工作.

---



## 6.2. 使用Bootstrap

在一个Yaf\_Application被实例化之后, 运行(Yaf\_Application::run)之前, 可选的我们可以运行Yaf\_Application::bootstrap

### 例 6.1. 调用Bootstrap

```
<?php
$app = new Yaf_Application("conf.ini");
$app
    ->bootstrap() //可选的调用
    ->run();
}
```

当bootstrap被调用的时刻, Yaf\_Application就会默认的在APPLICATION\_PATH下, 寻找Bootstrap.php, 而这个文件中, 必须定义一个Bootstrap类, 而这个类也必须继承自Yaf\_Bootstrap\_Abstract.

实例化成功之后, 所有在Bootstrap类中定义的, 以\_init开头的方法, 都会被依次调用, 而这些方法都可以接受一个Yaf\_Dispatcher实例作为参数.



注意

也可以通过在配置文件中修改application.bootstrap来变更Bootstrap类的位置.

一个Bootstrap的例子:

### 例 6.2. Bootstrap

```
<?php
```

```
/**
 * 所有在Bootstrap类中，以_init开头的方法，都会被Yaf调用，
 * 这些方法，都接受一个参数: Yaf_Dispatcher $dispatcher
 * 调用的次序，和申明的次序相同
 */
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initConfig() {
        $config = Yaf_Application::app()->getConfig();
        Yaf_Registry::set("config", $config);
    }

    public function _initDefaultName(Yaf_Dispatcher $dispatcher) {
        $dispatcher->setDefaultModule("Index")->setDefaultController
("Index")->setDefaultAction("index");
    }
}
```



#### 注意

方法在Bootstrap类中的定义出现顺序，决定了它们的被调用顺序。比如对于上面的例子，\_initConfig会第一个被调用。

参见

[Yaf\\_Bootstrap\\_Abstract](#)

[上一页](#)

第 6 章 使用Bootstrap

[上一级](#)

[起始页](#)

[下一页](#)

第 7 章 使用插件

## 第 7 章 使用插件

### 7.1. 简介

Yaf支持用户定义插件来扩展Yaf的功能, 这些插件都是一些类. 它们都必须继承自 `Yaf_Plugin_Abstract`. 插件要发挥功效, 也必须现实的在Yaf中进行注册, 然后在适当的实际, Yaf就会调用它.

---

## 7.2. Yaf支持的Hook

Yaf定义了6个Hook，它们分别是：

表 7.1. Yaf的Hook

触发顺序	名称	触发时机	说明
1	routerStartup	在路由之前触发	这个是7个事件中，最早的一个。但是一些全局自定的工作，还是应该放在Bootstrap中去完成
2	routerShutdown	路由结束之后触发	此时路由一定正确完成，否则这个事件不会触发
3	dispatchLoopStartup	分发循环开始之前被触发	
4	preDispatch	分发之前触发	如果在一个请求处理过程中，发生了forward，则这个事件会被触发多次
5	postDispatch	分发结束之后触发	此时动作已经执行结束，视图也已经渲染完成。和preDispatch类似，此事件也可能触发多次
6	dispatchLoopShutdown	分发循环结束之后触发	此时表示所有的业务逻辑都已经运行完成，但是响应还没有发送

---

[上一页](#)

第 7 章 使用插件

[上一级](#)

[起始页](#)

[下一页](#)

7.3. 定义插件

## 7.3. 定义插件

插件类是用户编写的,但是它需要继承自`Yaf_Plugin_Abstract`. 对于插件来说,上一节提到的7个Hook,它不需要全部关心,它只需要在插件类中定义和上面事件同名的方法,那么这个方法就会在该事件触发的时候被调用.

而插件方法,可以接受两个参数,`Yaf_Request_Abstract`实例和`Yaf_Response_Abstract`实例. 一个插件类例子如下:

### 例 7.1. 插件类

```
<?php
class UserPlugin extends Yaf_Plugin_Abstract {

    public function routerStartup(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
    }

    public function routerShutdown(Yaf_Request_Abstract
$request, Yaf_Response_Abstract $response) {
    }
}
```

这个例子中,插件`UserPlugin`只关心两个事件. 所以就定义了俩个方法.

## 7.4. 注册插件

插件要生效, 还需要向Yaf\_Dispatcher注册, 那么一般的插件的注册都会放在Bootstrap中进行. 一个注册插件的例子如下:

### 例 7.2. 注册插件

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initPlugin() {
        $user = new UserPlugin();
        $dispatcher->registerPlugin($user);
    }
}
```

## 7.5. 插件目录

一般的, 插件应该防止在APPLICATION\_PATH下的plugins目录, 这样在自动加载的时候, 加载器通过类名, 发现这是个插件类, 就会在这个目录下查找.

当然, 插件也可以放在任何你想防止的地方, 只要你能把这个类加载进来就可以

参见

**Yaf\_Plugin\_Abstract**  
使用Bootstrap



---

## 第 8 章 路由和路由协议

### 8.1. 概述

路由器主要负责解析一个请求并且决定什么`module`、`controller`、`action`被请求；它同时也定义了一种方法来实现用户自定义路由，这也使得它成为最重要的一个MVC组组件。

为了方便自定义路由，Yaf摒弃了0.1版本中的自定义路由器方式，而采用了更为灵活的路由器和路由协议分离的模式。

也就是一个固定不变的路由器，配合各种可自定义的路由协议，来实现灵活多变的路由策略。

## 8.2. 设计

作为一个应用中的路由组件是很重要的,理所当然的路由组件是抽象的,这样允许作为开发者的我们很容易的设计出我们自定义的路由协议.然而,默认的路由组件其实已经服务得我们很好了.记住,如果我们需要一个非标准的路由协议时候,我们就可以自定义一个自己的路由协议,而不用采用默认的路由协议.事实上,路由组件有两个部分: 路由器(`Yaf_Router`)和路由协议(`Yaf_Route_Abstract`).

路由协议事实上主要负责匹配我们预先定义好的路由协议,意思就是我们只有一个路由器,但我们可以有许多路由协议. 路由器主要负责管理和运行路由链,它根据路由协议栈倒序依次调用各个路由协议,一直到某一个路由协议返回成功以后, 就匹配成功.



### 小心

路由注册的顺序很重要, 最后注册的路由协议, 最先尝试路由, 这就有个陷阱. 请注意.

路由的过程发生派遣过程的最开始,并且路由解析仅仅发生一次.路由过程在何控制器动作(`Controller, Action`)被派遣之前被执行,一旦路由成功,路由器将会把解析出得到的信息传递给请求对象(`Yaf_Request_Abstract` object), 这些信息包括`moduel`、`controller`、`action`、用户`params`等. 然后派遣器(`Yaf_Dispatcher`)就会按照这些信息派遣正确的控制器动作. 路由器也有插件钩子,就是`routeStartup`和`routeShutdown`,他们在路由解析前后分别被调用.

## 8.3. 默认情况

默认情况下,我们的路由器是[Yaf\\_Router](#), 而默认使用的路由协议是[Yaf\\_Route\\_Static](#),是基于HTTP路由的, 它期望一个请求是HTTP请求并且请求对象是使用[Yaf\\_Request\\_Http](#)

---

## 8.4. 使用路由

使用路由既可以让他很复杂，同时也能让它很简单，这是归于你的应用。然而使用一个路由是很简单的，你可以添加你的路由协议给路由器，这样就OK了！不同的路由协议如下所示：

[Yaf\\_Route\\_Simple](#)

[Yaf\\_Route\\_Supervar](#)

[Yaf\\_Route\\_Static](#)

[Yaf\\_Route\\_Map](#)

[Yaf\\_Route\\_Rewrite](#)

[Yaf\\_Route\\_Regex](#)

首先让我们来看看路由器是如何让路由协议与之一起工作的。在我们添加任何路由协议之前我们必须得到 一个路由器(Yaf\_Router)实例，我们通过派遣器的getRouter方法来得到默认的路由器：

### 例 8.1. 例子

```
<?php
//通过派遣器得到默认的路由器
$router = Yaf_Dispatcher::getInstance()->getRouter();
?>
```

一旦我们有了路由器实例,我们就能通过它来添加我们自定义的一些路由协议：

```
<?php
$router->addRoute('myRoute', $route);
$router->addRoute('myRoute1', $route)
```

除此之外,我们还可以直接添加在配置中定义我们路由协议：

### 例 8.2. 配置添加路由协议的例子

```
[common]
;自定义路由
;顺序很重要
routes.regex.type="regex"
routes.regex.match="#^/list/([^/]*)/([^/]*)#"
routes.regex.route.controller=Index
routes.regex.route.action=action
routes.regex.map.1=name
routes.regex.map.2=value
;添加一个名为simple的路由协议
routes.simple.type="simple"
routes.simple.controller=c
```

```

routes.simple.module=m
routes.simple.action=a
;添加一个名为supervar的路由协议
routes.supervar.type="supervar"
routes.supervar.varname=r

[product : common]
;product节是Yaf默认关心的节，添加一个名为rewrite的路由协议
routes.rewrite.type="rewrite"
routes.rewrite.match="/product/:name/:value"

```

**注意**

路由协议的顺序很重要，所以Yaf保证添加顺序和在配置文件中的顺序相同

例 8.3. 然后在Bootstrap中通过调用Yaf\_Router::addConfig添加定义在配置中的路由协议

```

<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initRoute(Yaf_Dispatcher $dispatcher) {
        $router = Yaf_Dispatcher::getInstance()->getRouter();
        /**
         * 添加配置中的路由
         */
        $router->addConfig(Yaf_Registry::get("config")->routes);
    }
}

```

//其实路由器也提供给我们不同的方法来得到和设置包含在它内部的信息,一些重要的方法如下:  
 getCurrentRoute() //在路由结束以后, 获取起作用的路由协议  
 getRoute(), getRoutes();//看函数基本意思也就知道.

[上一页](#)
[8.3. 默认情况](#)
[上一级](#)
[起始页](#)
[下一页](#)
[8.5. 路由协议详解](#)

## 8.5. 路由协议详解

### 默认路由协议

默认的路由协议Yaf\_Route\_Static, 就是分析请求中的request\_uri, 在去除掉base\_uri以后, 获取到真正的负载路由信息的request\_uri片段, 具体的策略是, 根据"/"对request\_uri分段, 依次得到Module,Controller,Action, 在得到Module以后, 还需要根据Yaf\_Application::\$modules来判断Module是否是合法的Module, 如果不是, 则认为Module并没有体现在request\_uri中, 而把原Module当做Controller, 原Controller当做Action:

#### 例 8.4. 默认路由协议

```
<?php
/**
 * 对于请求request_uri为"/ap/foo/bar/dummy/1"
 * base_uri为"/ap"
 * 则最后参加路由的request_uri为"/foo/bar/dummy/1"
 * 然后, 通过对URL分段, 得到如下分节
 * foo, bar, dummy, 1
 * 然后判断foo是不是一个合法的Module, 如果不是, 则认为结果如下:
 */
array(
    'module'      => '默认模块',
    'controller'  => 'foo',
    'action'      => 'bar',
    'params'      => array(
        'dummy'   => 1,
    )
)

/**
 * 而如果在配置文件中定义了ap.modules="Index, Foo",
 * 则此处就会认为foo是一个合法模块, 则结果如下
 */
array(
    'module'      => 'foo',
    'controller'  => 'bar',
    'action'      => 'dummy',
    'params'      => array(
        1 => NULL,
    )
)
```



重要

当只有一段路由信息的时候, 比如对于上面的例子, 请求的URI为/ap/foo, 则默认路由和下面要提到的Yaf\_Route\_Supervar会首先判断`ap.action_prefer`, 如果为真, 则把foo当做Action, 否则当做Controller

## Yaf\_Route\_Simple

Yaf\_Route\_Simple是基于请求中的query string来做路由的, 在初始化一个Yaf\_Route\_Simple路由协议的时候, 我们需要给出3个参数, 这3个参数分别代表在query string中Module, Controller, Action的变量名:

### 例 8.5. Yaf\_Route\_Simple

```
<?php
/**
 * 指定3个变量名
 */
$route = new Yaf_Route_Simple("m", "c", "a");
$router->addRoute("name", $route);
/**
 * 对于如下请求: "http://domain.com/index.php?c=index&a=test"
 * 能得到如下路由结果
 */
array(
    'module'    => '默认模块',
    'controller' => 'index',
    'action'    => 'test',
)
```



#### 注意

只有在query string中不包含任何3个参数之一的情况下, Yaf\_Route\_Simple才会返回失败, 将路由权交给下一个路由协议。

## Yaf\_Route\_Supervar

Yaf\_Route\_Supervar和Yaf\_Route\_Simple相似, 都是在query string中获取路由信息, 不同的是, 它获取的是一个类似包含整个路由信息的request\_uri

### 例 8.6. Yaf\_Route\_Supervar

```
<?php
/**
 * 指定supervar变量名
 */
$route = new Yaf_Route_Supervar("r");
$router->addRoute("name", $route);
/**
```

```

* 对于如下请求: "http://domain.com/index.php?r=a/b/c"
* 能得到如下路由结果
*/
array(
  'module'    => 'a',
  'controller' => 'b',
  'action'    => 'c',
)

```

**注意**

在query string中不包含supervar变量的时候, Yaf\_Route\_Supervar会返回失败, 将路由权交给下一个路由协议.

## Yaf\_Route\_Map

Yaf\_Route\_Map是一种简单的路由协议, 它将REQUEST\_URI中以'/'分割的节, 组合在一起, 形成一个分层的控制器或者动作的路由结果. Yaf\_Route\_Map的构造函数接受俩个参数, 第一个参数表示路由结果是作为动作的路由结果, 还是控制器的路由结果. 默认的是动作路由结果. 第二个参数是一个字符串, 表示一个分隔符, 如果设置了这个分隔符, 那么在REQUEST\_URI中, 分隔符之前的作为路由信息载体, 而之后的作为请求参数.

### 例 8.7. 映射路由协议

```

<?php
/**
 * 对于请求request_uri为"/ap/foo/bar"
 * base_uri为"/ap"
 * 则最后参加路由的request_uri为"/foo/bar"
 * 然后, 通过对URL分段, 得到如下分节
 * foo, bar
 * 组合在一起以后, 得到路由结果foo_bar
 * 然后根据在构造Yaf_Route_Map的时候, 是否指明了控制器优先,
 * 如果没有, 则把结果当做是动作的路由结果
 * 否则, 则认为是控制器的路由结果
 * 默认的, 控制器优先为FALSE
 */

```

## Yaf\_Route\_Rewrite

Yaf\_Route\_Rewrite是一个强大的路由协议, 它能满足我们绝大部分的路由需求:

### 例 8.8. Yaf\_Route\_Rewrite

```

<?php
//创建一个路由协议实例

```



```

$route = new Yaf_Route_Rewrite(
    'product/:ident',
    array(
        'controller' => 'products',
        'action' => 'view'
    )
);
//使用路由器装载路由协议
$router->addRoute('product', $route);

```

在这个例子中, 我们试图匹配Url指定到一个单一的产品, 就像<http://domain.com/product/choclolat-bar>. 为了实现这点, 我们在路由协议中传递了2个变量到路由协议Yaf\_Route\_Rewrite的构造函数其中. 第一个变量('product/:ident')就是匹配的路径, 第二个变量(array变量)是路由到的动作控制器; 路径使用一个特别的标识来告诉路由协议如何匹配到路径中的每一个段,这个标识有有两种,可以帮助我们创建我们的路由协议,如下所示:

a) :

b) \*

冒号(:)指定了一个段,这个段包含一个变量用于传递到我们动作控制器中的变量,我们要设置好事先的变量名,比如在上面我们的变量名就是'ident',因此,倘若我们访问<http://domian.com/product/chocoloate-bar>将会创建一个变量名为ident并且其值是'chocoloate-bar'的变量,我们然后就可以在我们的动作控制器ProductsController/viewAction下获取到它的值: `$this->getRequest()->getParam('ident');`

星号(\*)被用做一个通配符, 意思就是在Url中它后面的所有段都将作为一个通配数据被存储. 例如,如果我们 有路径'path/product/:ident/\*'(就是路由协议中设置的第一个变量), 并且我们访问的Url为<http://domain.com/product/chocolate-bar/lue1/another/value2>,那么所有的在'chocolate-bar'后面的段都将被做成变量名/值对,因此这样会给我们下面的结果:

```

ident = chocolate-bar
test = value1
another = value2

```

这种行为也就是我们平常默认使用的路由协议的行为,记住变量名/值要成对出现,否则像/test/value1/这样的将不会这种另一个变量,我们有静态的路由协议部分,这些部分简单地被匹配来满足我们的路由协议,在我们的例子中,静态部分就是product; 就像你现在看到的那样,我们的Yaf\_Route\_Rewrite路由协议提供给我们极大的灵活性来控制我们的路由

## Yaf\_Route\_Regex

到目前为止,我们之前的路由协议都很好的完成了基本的路由操作,我们常用的也是他们,然而它们会有一些限制,这就是我们为什么要引进正则路由(Yaf\_Route\_Regex)的原因. 正则路由给予我们preg正则的全部力量,但同时也使得我们的路由协议变得更加复杂了一些.即使是他们有点复杂,我还是希望你能好好掌握它,因为它比其他路由协议要灵活一点点. 一开始,我们先对之前的产品案例改用使用正则路由:

### 例 8.9. Yaf\_Route\_Regex

```

<?php
$route = new Yaf_Route_Regex(

```

```

        'product/([a-zA-Z-_0-9]+)',
        array(
            'controller' => 'products',
            'action' => 'view'
        )
    );
    $router->addRoute('product', $route);

```

你可以看到,我们现在移动我们的正则到我们的`path`(构造函数的第一个参数)中来了,就像之前的那样,这个正则路由协议现在应该是匹配是一个数字、字母、`-`和`_`组成的`ident`变量的字符提供给我们,但是,你一定会问,`ident`变量在哪呢? 好,如果你使用了这个正则路由协议,我们可以通过变量`1(one)`来获取其值,即可以在控制器里用: `$this->getRequest()->getParam(1)`来获取,其实这里如果看过正则的都知道这就是反向引用中的`\1`. 然而,你一定会想为什么要定义的这么的垃圾,我们不能够记住或者弄清每一个数字代表的是什么变量(其实我刚开始看的时候也是一样的感受). 为了改变这点,正则路由协议的构造函数提供了第3个参数来完成数字到变量名的映射:

#### 例 8.10. Yaf\_Route\_Regex

```

<?php
$route = new Yaf_Route_Regex(
    'product/([a-zA-Z-_0-9]+)',
    array(
        'controller' => 'products',
        'action' => 'view'
    ),
    array(
        //完成数字到字符变量的映射
        1 => 'ident'
    )
);
$router->addRoute('product', $route);

```

这样,我们就简单的将变量`1`映射到了`ident`变量名,这样就设置了`ident`变量,同时你也可以在控制器里面获取到它的值.

[上一页](#)
[8.4. 使用路由](#)
[上一级](#)
[起始页](#)
[下一页](#)
[8.6. 自定义路由协议](#)

## 8.6. 自定义路由协议

当然, 这个世界上没有绝对的事情. 所以万一现在所有的路由协议都不能满足你的需求, 那么你可以自己实现你自己的路由协议, 你要做的是, 申明你的路由协议实现了 `Yaf_Route_Interface` 接口即可.

---

---

## 第 9 章 在命令行使用Yaf

### 9.1. 简介

Yaf支持在命令行下运行, 以此来方便开发人员调试.

---

## 9.2. 使用样例

要使得Yaf在命令行模式下运行, 唯一要变更的就是请求体, 默认的请求是由Yaf\_Application实例化, 并且交给Yaf\_Dispatcher的, 而在命令行模式下, Yaf\_Application并不能正确的实例化一个命令行请求, 所以需要变更一下, 请求需要手动实例化.

### 例 9.1. 实例化一个Yaf\_Request\_Simple

```
<?php
$request = new Yaf_Request_Simple();
print_r($request);
```

如上面的例子, Yaf\_Request\_Simple的构造函数可以不接受任何参数, 在这种情况下, Yaf\_Request\_Simple会在命令行参数中, 寻找一个字符串参数, 如果找到, 则会把请求的request\_uri置为这个字符串.



#### 注意

当然, Yaf\_Request\_Simple是可以接受5个参数的, 具体的可见Yaf\_Request\_Simple类说明.

现在让试着运行上面的代码:

### 例 9.2.

```
$ php request.php
```

输出:

```
Yaf_Request_Simple Object
(
    [module] =>
    [controller] =>
    [action] =>
    [method] => CLI
    [params:protected] => Array
    (
    )

    [language:protected] =>
    [_base_uri:protected] =>
    [uri:protected] =>
    [dispatched:protected] =>
    [routed:protected] =>
)
```

现在让我们变更下我们的运行方式:

### 例 9.3.

```
$ php request.php "request_uri=/index/hello"
```

输出:

```
Yaf_Request_Simple Object
(
    [module] =>
    [controller] =>
    [action] =>
    [method] => CLI
    [params:protected] => Array
    (
    )

    [language:protected] =>
    [_base_uri:protected] =>
```

```
[uri:protected] => index/hello //注意这里
[dispatched:protected] =>
[routed:protected] =>
)
```

看到差别了么？

当然，我们也可以完全指定Yaf\_Request\_Simple::\_\_construct的5个参数：

#### 例 9.4. 带参数实例化一个Yaf\_Request\_Simple

```
<?php
$request = new Yaf_Request_Simple("CLI", "Index", "Controller", "Hello",
array("para" => 2));
print_r($request);
```

运行输出：

```
$ php request.php
Yaf_Request_Simple Object
(
    [module] => Index
    [controller] => Controller
    [action] => Hello
    [method] => CLI
    [params:protected] => Array
    (
        [para] => 2
    )

    [language:protected] =>
    [_base_uri:protected] =>
    [uri:protected] =>
    [dispatched:protected] =>
    [routed:protected] => 1 //注意这里
)
```

可以看到一个比较特别的, `routed`属性变为了`TRUE`, 这就代表着如果我们手动指定了构造函数的参数, 那么这个请求不会再经过路由, 而直接是路由完成状态.

---

[上一页](#)

第 9 章 在命令行使用Yaf

[上一级](#)

[起始页](#)

[下一页](#)

9.3. 分发请求



## 9.3. 分发请求

现在请求已经改造完成了, 那么接下来就简单了, 我们只需要把我们传统的入口文件:

例 9.5. 入口文件

```
<?php
$app = new Yaf_Application("conf.ini");
$app->bootstrap()->run();
```

改为:

例 9.6. 入口文件

```
<?php
$app = new Yaf_Application("conf.ini");
$app->getDispatcher()->dispatch(new Yaf_Request_Simple());
```

这样, 我们就可以通过在命令行中运行Yaf了

参见

[Yaf\\_Request\\_Simple](#)

---

## 第 10 章 异常和错误

### 10.1. 概述

Yaf实现了一套错误和异常捕获机制, 主要是对常见的错误处理和异常捕获方法做了一个简单抽象, 方便应用组织自己的错误统一处理逻辑.

Yaf自身出错时候, 根据配置可以分别采用抛异常或者触发错误的方式来通知错误. 在 `apliation.dispatcher.throwException`(配置文件, 或者通过`Yaf_Dispatcher::throwException(true)`)打开的情况下, Yaf会抛异常, 否则则会触发错误.

那么对应的, 就有俩套错误处理方式可供应用选用.

## 10.2. 异常模式

在`application.dispatcher.catchException`(配置文件, 或者可通过`Yaf_Dispatcher::catchException(true)`)开启的情况下, 当Yaf遇到未捕获异常的时候, 就会把运行权限, 交给当前模块的Error Controller的Error Action动作, 而异常或作为请求的一个参数, 传递给Error Action.

在Error Action中可以通过`$request->getRequest()->getParam("exception")`获取当前发生的异常.



### 重要

从Yaf1.0.0.12开始, 也可以通过`$request->getException()`来获取当前发生的异常, 而如果Error Action定义了一个名为`$exception`的参数, 也可以直接通过这个参数获取当前发生的异常.

### 例 10.1. Error Controller

```
<?php
/**
 * 当有未捕获的异常, 则控制流会流到这里
 */
class ErrorController extends Yaf_Controller_Abstract {
    /**
     * 也可通过$request->getException() 获取到发生的异常
     */
    public function errorAction($exception) {
        assert($exception === $exception->getCode());
        $this->getView()->assign("code", $exception->getCode());
        $this->getView()->assign("message", $exception->getMessage());
    }
}
```

有了这样的最终异常处理逻辑, 应用就可以在出错的时候直接抛出异常, 在统一异常处理逻辑中, 根据不同的异常逻辑, 处理错误, 记录日志. 一个常用的Error Action如下:

### 例 10.2.

```
<?php
/**
 * 当有未捕获的异常, 则控制流会流到这里
 */
class ErrorController extends Yaf_Controller_Abstract {
    /**
     * 也可通过$request->getException() 获取到发生的异常
     */
    public function errorAction($exception) {
```

```

switch($exception->getCode()) {
    case YAF_ERR_LOADFAILED:
    case YAF_ERR_LOADFAILED_MODULE:
    case YAF_ERR_LOADFAILED_CONTROLLER:
    case YAF_ERR_LOADFAILED_ACTION:
        //404
        header("Not Found");
        break;

    case CUSTOM_ERROR_CODE:
        //自定义的异常
        ....
        break;
}
}

```

当然, 一个更为易读的方式:

### 例 10.3.

```

<?php
/**
 * 当有未捕获的异常, 则控制流会流到这里
 */
class ErrorController extends Yaf_Controller_Abstract {
    /**
     * 此时可通过$request->getException() 获取到发生的异常
     */
    public function errorAction() {
        $exception = $this->getRequest()->getException();
        try {
            throw $exception;
        } catch (Yaf_Exception_LoadFailed $e) {
            //加载失败
        } catch (Yaf_Exception $e) {
            //其他错误
        }
    }
}

```

[上一页](#)
[第 10 章 异常和错误](#)
[上一级](#)
[起始页](#)
[下一页](#)
[10.3. 错误模式](#)

---

## 10.3. 错误模式

---

## 第 11 章 内建的类

### 11.1. The Yaf\_Application class

#### 简介

`Yaf_Application` 代表一个产品/项目, 是 Yaf 运行的主导者, 真正执行的主体. 它负责接收请求, 协调路由, 分发, 执行, 输出.

在 PHP5.3 之后, 打开 `ap.use_namespace` 的情况下, 也可以使用 `Yaf\Application`.

```
final Yaf_Application {
    protected Yaf_Config _config ;
    protected Yaf_Dispatcher _dispatcher ;
    protected static Yaf_Application _app ;
    protected boolean _run = FALSE ;
    protected string _environ ;
    protected string _modules ;
    public void __construct ( mixed $config ,
                             string $section = ap.
    environ );
    public Yaf_Application bootstrap ( void );
    public Yaf_Response_Abstract run ( void );
    public Yaf_Dispatcher getDispatcher ( void );
    public Yaf_Config_Abstract getConfig ( void );
    public string environ ( void );
    public string getModules ( void );
    public static Yaf_Application app ( void );
    public mixed execute ( callback $function ,
                           mixed $parameter = NULL ,
                           mixed $... = NULL );
}
```

#### 属性说明

`_app`

---

`Yaf_Application`通过特殊的方式实现了单利模式, 此属性保存当前实例

---

#### `_config`

---

全局配置实例

---

#### `_dispatcher`

---

`Yaf_Dispatcher`实例

---

#### `_modules`

---

存在的模块名, 从配置文件中`ap.modules`读取

---

#### `_environ`

---

当前的环境名, 也就是`Yaf_Application`在读取配置的时候, 获取的配置节名字

---

#### `_run`

---

布尔值, 指明当前的`Yaf_Application`是否已经运行

---

[上一页](#)

[下一页](#)

[10.3. 错误模式](#)

[起始页](#)

[Yaf\\_Application::\\_\\_construct](#)

## 名称

**Yaf\_Application::\_\_construct**

(Since Yaf 1.0.0.0)

```
public void Yaf_Application::__construct(mixed $config,  
                                           string $section = ap.  
                                           environ );
```

初始化一个Yaf\_Application, 如果\$config是一个INI文件, 那么\$section指明要读取的配置节.

## 参数

***config***

关联数组的配置, 或者一个指向ini格式的配置文件的路径的字符串, 或者是一个Yaf\_Config\_Abstract实例

## 返回值

void

## 例子

**例 11.1. Yaf\_Application::\_\_construct的例子**

&lt;?php



```
$config = array(  
    "ap" => array(  
        "directory" => "/usr/local/www/ap",  
    ),  
);  
$app = new Yaf_Application($config);  
?>
```

输出

```
object(Yaf_Application)#1 (6) {  
    ...  
}
```

---

[上一页](#)

11.1. The Yaf\_Application class

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Application::bootstrap

## 名称

**Yaf\_Application::bootstrap**

(Since Yaf 1.0.0.0)

```
Yaf_Application Yaf_Application::bootstrap(void);
```

指示Yaf\_Application去寻找Bootstrap(默认在ap.directory/Bootstrap.php), 并执行所有在Bootstrap类中定义的, 以\_init开头的方法. 一般用作在处理请求之前, 做一些个性化定制.

Bootstrap并不会调用run, 所以还需要在bootstrap以后调用Application::run来运行Yaf\_Application实例

## 参数

```
void
```

本方法不需要参数

## 返回值

Yaf\_Application

## 例子

**例 11.2. Yaf\_Application::bootstrap 的例子**

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
```

```
$app = new Yaf_Application($config);  
$app->bootstrap()->run();  
?>
```

参见

[Yaf\\_Application::run](#)

[Yaf\\_Bootstrap\\_Abstract](#)

---

[上一页](#)

[Yaf\\_Application::\\_\\_construct](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Application::app](#)

## 名称

### Yaf\_Application::app

(Since Yaf 1.0.0.0)

```
static Yaf_Application Yaf_Application::app(void);
```

获取当前的Yaf\_Application实例

#### 参数

void

本方法不需要参数

#### 返回值

Yaf\_Application

#### 例子

##### 例 11.3. Yaf\_Application::app 的例子

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
$app = new Yaf_Application($config);
assert($app === Yaf_Application::app());
?>
```



---

[上一页](#)

Yaf\_Application::bootstrap

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Application::environ

## 名称

**Yaf\_Application::environ**

(Since Yaf 1.0.0.5)

```
string Yaf_Application::environ(void);
```

获取当前Yaf\_Application的环境名

## 参数

```
void
```

本方法不需要参数

## 返回值

当前的环境名, 也就是ini\_get("yaf.environ")

## 例子

**例 11.4. Yaf\_Application::environ 的例子**

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
$app = new Yaf_Application($config);
print($app->environ());
?>
```



---

[上一页](#)

Yaf\_Application::app

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Application::run

## 名称

### Yaf\_Application::run

(Since Yaf 1.0.0.0)

```
boolean Yaf_Application::run(void);
```

运行一个Yaf\_Application, 开始接受并处理请求. 这个方法只能调用一次, 多次调用并不会有特殊效果.

#### 参数

```
void
```

本方法不需要参数

#### 返回值

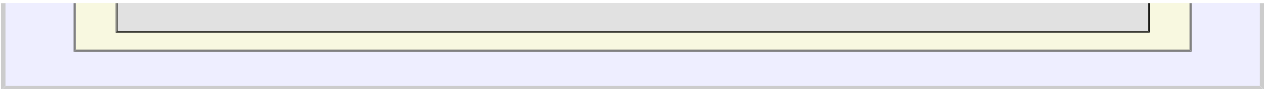
boolean

#### 例子

##### 例 11.5. Yaf\_Application::run 的例子

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
$app = new Yaf_Application($config);
$app->run();
?>
```





---

[上一页](#)

Yaf\_Application::environ

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Application::execute

## 名称

**Yaf\_Application::execute**

(Since Yaf 1.0.0.17)

```
mixed Yaf_Application::execute( callback $function ,  
                                mixed $parameter = NULL ,  
                                $parameter $... = NULL );
```

在Yaf\_Application的环境下, 运行一个用户自定义函数过程. 主要在使用Yaf做简单的命令行脚本的时候, 应用Yaf的外围环境, 比如: 自动加载, 配置, 视图引擎等.

**注意**

如果需要使用Yaf的路由分发, 也就是说, 如果是需要在CLI下全功能运行Yaf, 请参看[在命令行下使用Yaf](#)

## 参数

***\$function***

要运行的函数或者方法, 方法可以通过array(\$obj, "method\_name")来定义.

***\$parameter***

零个或者多个要传递给函数的参数.

## 返回值

被调用函数或者方法的返回值

## 例子

## 例 11.6. Yaf\_Application::execute 的例子

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
$app = new Yaf_Application($config);
$app->execute("main");

function main() {
}
?>
```

参见

[在命令行下使用Yaf](#)

---

[上一页](#)

[Yaf\\_Application::run](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Application::getDispatcher](#)

名称

**Yaf\_Application::getDispatcher**

(Since Yaf 1.0.0.6)

```
Yaf_Config_Abstract Yaf_Application::getDispatcher(void );
```

获取当前的分发器

参数

**void**

本方法不需要参数

返回值

Yaf\_Dispatcher实例

例子

**例 11.7. Yaf\_Application::getDispatcher 的例子**

```
<?php
define ("APPLICATION_PATH", dirname(__FILE__));

$app = new Yaf_Application("conf/application_simple.ini");
```

```
//bootstrap
$app->getDispatcher()->setAppDirectory(APPLICATION_PATH .
"/action/")->getApplication()->bootstrap()->run();

//当然也可以使用
$dispatcher = Yaf_Dispatcher::getInstance()->setAppDirectory(APPLICATION_PATH .
"/action/")->getApplication()->bootstrap()->run();

?>
```

参见

[Yaf\\_Dispatcher::getInstance](#)

---

[上一页](#)

[Yaf\\_Application::execute](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Application::getConfig](#)

## 名称

### Yaf\_Application::getConfig

(Since Yaf 1.0.0.0)

```
Yaf_Config_Abstract Yaf_Application::getConfig(void );
```

获取Yaf\_Application读取的配置项.

#### 参数

void

本方法不需要参数

#### 返回值

Yaf\_Config\_Abstract

#### 例子

##### 例 11.8. Yaf\_Application::getConfig 的例子

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
```

```
$app = new Yaf_Application($config);  
print_r($app->getConfig(' application'));  
?>
```

输出

```
Yaf_Config Object  
(  
    [_config: private] => Array  
    (  
        [ap] => Array  
        (  
            [directory] => /usr/local/www/ap  
        )  
    )  
)
```

---

[上一页](#)

[Yaf\\_Application::getDispatcher](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Application::getModules](#)

## 名称

**Yaf\_Application::getModules**

(Since Yaf 1.0.0.5)

```
string Yaf_Application::getModules(void );
```

获取在配置文件中声明的模块。

## 参数

```
void
```

本方法不需要参数

## 返回值

```
string
```

## 例子

**例 11.9. Yaf\_Application::getModules 的例子**

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
        "modules"   => "Index",
```



```
),  
);  
$app = new Yaf_Application($config);  
print_r($app->getModules());  
?>
```

输出

```
Array  
(  
    [0] => Index  
)
```

---

[上一页](#)[Yaf\\_Application::getConfig](#)[上一级](#)[起始页](#)[下一页](#)[11.2. The Yaf\\_Bootstrap class](#)

## 11.2. The Yaf\_Bootstrap class

### 简介

**Yaf\_Bootstrap\_Abstract**提供了一个可以定制**Yaf\_Application**的最早的时机, 它相当于一段引导, 入口程序. 它本身没有定义任何方法. 但任何继承自**Yaf\_Bootstrap**的类中的以**\_init**开头的方法, 都会在**Yaf\_Application::bootstrap**时刻被调用. 调用的顺序和这些方法在类中的定义顺序相同. **Yaf**保证这种调用顺序.



#### 注意

这些方法, 都可以接受一个**Yaf\_Dispatcher**参数.

### 例子

#### 例 11.10. Yaf\_Bootstrap\_Abstract的例子

```
<?php
/**
 * 所有在Bootstrap类中, 以_init开头的方法, 都会被Yaf调用,
 * 这些方法, 都接受一个参数: Yaf_Dispatcher $dispatcher
 * 调用的次序, 和声明的次序相同
 */
class Bootstrap extends Yaf_Bootstrap_Abstract{
    /**
     * 注册一个插件
     * 插件的目录是在application_directory/plugins
     */
    public function _initPlugin(Yaf_Dispatcher $dispatcher) {
        $user = new UserPlugin();
        $dispatcher->registerPlugin($user);
    }

    /**
     * 添加配置中的路由
     */
    public function _initRoute(Yaf_Dispatcher $dispatcher) {
```

```

        $router = Yaf_Dispatcher::getInstance()->getRouter();
        $router->addConfig(Yaf_Registry::get("config")->routes);
        /**
         * 添加一个路由
         */
        $route = new Yaf_Route_Rewrite(
            "/product/list/:id/",
            array(
                "controller" => "product",
                "action"      => "info",
            )
        );

        $router->addRoute('dummy', $route);
    }

    /**
     * 自定义视图引擎
     */
    public function _initSmarty(Yaf_Dispatcher $dispatcher) {
        $smarty = new Smarty_Adapter(null, Yaf_Registry::get("config")->get("smarty"));
        Yaf_Dispatcher::getInstance()->setView($smarty);
    }
}

```

在入口文件:

```

<?php
/* 默认的, Yaf_Application将会读取配置文件中在php.ini中设置的ap. enviro n的配置节 */
$application = new Yaf_Application("conf/sample.ini");

/* 如果没有关闭自动response(通过Yaf_Dispatcher::getInstance()->autoResponse(FALSE)),
 * 则$response会被自动输出, 此处也不需要再次输出Response
 */
$response = $application
    ->bootstrap() /*实例化Bootstrap, 依次调用Bootstrap中所有_init开头的方法*/
    ->run();
?>

```

参见

Yaf\_Application::bootstrap

[上一页](#)

Yaf\_Application::getModules

[上一级](#)

[起始页](#)

[下一页](#)

11.3. The Yaf\_Loader class

## 11.3. The Yaf\_Loader class

### 简介

Yaf\_Loader类为Yaf提供了自动加载功能, 它根据类名中包含的路径信息实现类的定位和自动加载.

Yaf\_Loader也提供了对传统的require\_once的替代方案, 相比传统的require\_once, 因为舍弃对require的支持, 所以性能能有一丁点小优势.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Loader.

```
final Yaf_Loader {
    protected static Yaf_Loader _instance ;
    protected string _library_directory ;
    protected string _global_library_directory ;
    protected string _local_ns ;
    public static Yaf_Loader getInstance ( string
$local_library_directory = NULL ,
                                string
$global_library_directory = NULL );
    public Yaf_Loader registerLocalNamespace ( mixed $namespace );
    public boolean getLocalNamespace ( void );
    public boolean clearLocalNamespace ( void );
    public boolean isLocalName ( string $class_name );
    public static boolean import ( string $file_name );
    public boolean autoload ( string $class_name );
}
```

### 属性说明

#### \_instance

Yaf\_Loader实现了单利模式, 一般的它由Yaf\_Application负责初始化. 此属性保存当前实例

#### \_library\_directory

本地(自身)类加载路径, 一般的, 属性的值来自配置文件中的`ap.library`

---

#### `_global_library_directory`

---

全局类加载路径, 一般的, 属性的值来自`php.ini`中的`ap.library`

---

#### `_local_ns`

---

本地类的类名前缀, 此属性通过`Yaf_Loader::registerLocalNamespace`来添加新的值

---

[上一页](#)

[11.2. The Yaf\\_Bootstrap class](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::getInstance](#)

## 名称

## Yaf\_Loader::getInstance

(Since Yaf 1.0.0.5)

```
public static Yaf_Loader Yaf_Loader::getInstance( string  
$local_library_directory = NULL ,  
string  
$global_library_directory = NULL );
```

获取当前的Yaf\_Loader实例

## 参数

*\$local\_library\_directory*

本地(自身)类库目录, 如果留空, 则返回已经实例化过的Yaf\_Loader实例



## 小心

Yaf\_Loader是单利模式, 所以即使第二次以不同的参数实例化一个Yaf\_Loader, 得到的仍然是已经实例化的第一个实例.

*\$global\_library\_directory*

全局类库目录, 如果留空则会认为和\$local\_library\_directory相同.

返回值

Yaf\_Loader

例子

例 **11.11. Yaf\_Loader::getInstance** 的例子

```
<?php  
$loader = Yaf_Loader::getInstance();  
?>
```

---

[上一页](#)

11.3. The Yaf\_Loader class

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Loader::import



## 名称

### Yaf\_Loader::import

(Since Yaf 1.0.0.5)

```
public static boolean Yaf_Loader::import( string $file_name );
```

导入一个PHP文件, 因为Yaf\_Loader::import只是专注于一次包含, 所以要比传统的require\_once性能好一些

#### 参数

*\$file\_name*

要载入的文件路径, 可以为绝对路径和相对路径. 如果为相对路径, 则会以应用的本地类目录(ap.library)为基目录.

#### 返回值

成功返回TRUE, 失败返回FALSE.

#### 例子

例 11.12. Yaf\_Loader::import 的例子

```
<?php
//绝对路径
Yaf_Loader::import("/usr/local/foo.php");

//相对路径, 会在APPLICATION_PATH. "/library"下加载
Yaf_Loader::import("plugins/User.php");
```

```
?>
```

参见

**[Yaf\\_Loader::autoload](#)**

---

[上一页](#)

[Yaf\\_Loader::getInstance](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::autoload](#)

## 名称

**Yaf\_Loader::autoload**

(Since Yaf 1.0.0.5)

```
public static boolean Yaf_Loader::autoload( string $class_name );
```

载入一个类, 这个方法被Yaf用作自动加载类的方法, 当然也可以手动调用.

## 参数

*\$class\_name*

要载入的类名, 类名必须包含路径信息, 也就是下划线分隔的路径信息和类名. 载入的过程中, 首先会判断这个类名是否是本地类, 如果是本地类, 则使用本地类类库目录, 否则使用全局类目录. 然后判断`ap.lowercase_path`是否开启, 如果开启, 则会把类名中的路径部分全部小写. 然后加载, 执行.

```
/** ap.lowercase_path=0 */  
Foo_Bar_Dummy表示这个类存在于类库目录下的Foo/Bar/Dummy.php  
  
/** ap.lowercase_path=1 */  
Foo_Bar_Dummy表示这个类存在于类库目录下的foo/bar/Dummy.php
```



## 注意

在`php.ini`中的`ap.lowercase_path`开启的情况下, 路径信息中的目录部分都会被转换成小写.

## 返回值

成功返回TRUE



## 注意

在`php.ini`中的`ap.use_spl_autoload`关闭的情况下, 即使类没有找到, `Yaf_Loader::autoload`也会返回TRUE, 剥夺其后面的自动加载函数的执行权利.

## 例子

**例 11.13. Yaf\_Loader::autoload 的例子**

```
<?php
Yaf_Loader::autoload("Baidu_ST_Dummy_Bar");
?>
```

参见

[Yaf\\_Loader::import](#)

---

[上一页](#)

[Yaf\\_Loader::import](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::registerLocalNamespace](#)

名称

Yaf\_Loader::registerLocalNamespace

(Since Yaf 1.0.0.5)

```
public Yaf_Loader Yaf_Loader::registerLocalNamespace( mixed $local_name_prefix );
```

注册本地类前缀, 是的对于以这些前缀开头的本地类, 都从本地类库路径中加载.

参数

*\$local\_name\_prefix*

字符串或者是数组格式类名前缀, 不包含前缀后面的下划线.

返回值

Yaf\_Loader

例子

例 11.14. Yaf\_Loader::registerLocalNamespace 的例子

```
<?php
Yaf_Loader::getInstance()->registerLocalNamespace("Foo");
Yaf_Loader::getInstance()->registerLocalNamespace(array("Foo", "Bar"));
?>
```

参见

[Yaf\\_Loader::isLocalName](#)

[Yaf\\_Loader::getLocalNamespace](#)

[Yaf\\_Loader::clearLocalNamespace](#)

---

[上一页](#)

[Yaf\\_Loader::autoload](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::isLocalName](#)

## 名称

**Yaf\_Loader::isLocalName**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Loader::isLocalName( string $class_name );
```

判断一个类, 是否是本地类.

## 参数

***\$class\_name***

字符串的类名, 本方法会根据下划线分隔截取出类名的第一部分, 然后在Yaf\_Loader的 `_local_ns` 中判断是否存在, 从而确定结果.

## 返回值

boolean

## 例子

例 11.15. Yaf\_Loader::isLocalName 的例子

```
<?php
Yaf_Loader::getInstance()->registerLocalNamespace("Foo");

Yaf_Loader::getInstance()->isLocalName("Foo_Bar");//TRUE
Yaf_Loader::getInstance()->isLocalName("FooBar");//FALSE
?>
```

参见

[Yaf\\_Loader::registerLocalNamespace](#)

[Yaf\\_Loader::getLocalNamespace](#)

[Yaf\\_Loader::clearLocalNamespace](#)

---

[上一页](#)

[Yaf\\_Loader::registerLocalNamespace](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::getLocalNamespace](#)



## 名称

### Yaf\_Loader::getLocalNamespace

(Since Yaf 1.0.0.5)

```
public array Yaf_Loader::getLocalNamespace( void );
```

获取当前已经注册的本地类前缀

#### 参数

**void**

本方法不需要参数

#### 返回值

成功返回字符串

#### 例子

例 **11.16. Yaf\_Loader::getLocalNamespace** 的例子

```
<?php
Yaf_Loader::getInstance()->registerLocalNamespace(array("Foo", "Bar"));
print(Yaf_Loader::getInstance()->getLocalNamespace());
?>
```

输出:

: Foo: Bar:

参见

[Yaf\\_Loader::registerLocalNamespace](#)

[Yaf\\_Loader::isLocalName](#)

[Yaf\\_Loader::clearLocalNamespace](#)

---

[上一页](#)

[Yaf\\_Loader::isLocalName](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Loader::clearLocalNamespace](#)

## 名称

### Yaf\_Loader::clearLocalNamespace

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Loader::clearLocalNamespace( void );
```

清除已注册的本地类前缀

#### 参数

void

本方法不需要参数

#### 返回值

成功返回TRUE, 失败返回FALSE

#### 例子

例 11.17. Yaf\_Loader::clearLocalNamespace 的例子

```
<?php
Yaf_Loader::getInstance()->clearLocalNamespace();
?>
```

参见

[Yaf\\_Loader::registerLocalNamespace](#)

[Yaf\\_Loader::isLocalName](#)

[Yaf\\_Loader::getLocalNamespace](#)

---

[上一页](#)

[Yaf\\_Loader::getLocalNamespace](#)

[上一级](#)

[起始页](#)

[下一页](#)

[11.4. The Yaf\\_Dispatcher class](#)

## 11.4. The Yaf\_Dispatcher class

### 简介

Yaf\_Dispatcher实现了MVC中的C分发, 它由Yaf\_Application负责初始化, 然后由Yaf\_Application::run启动, 它协调路由来的请求, 并分发和执行发现的动作, 并收集动作产生的响应, 输出响应给请求者, 并在整个过程完成以后返回响应.

在PHP5.3之后, 打开ap.use\_namespace的情况下, 也可以使用 Yaf\Dispatcher.

```
final Yaf_Dispatcher {
    protected static Yaf_Dispatcher _instance ;
    protected Yaf_Router_Interface _router ;
    protected Yaf_View_Abstract _view ;
    protected Yaf_Request_Abstract _request ;
    protected array _plugins ;
    protected boolean _render ;
    protected boolean _return_response = FALSE ;
    protected boolean _instantly_flush = FALSE ;
    protected string _default_module ;
    protected string _default_controller ;
    protected string _default_action ;
    public static Yaf_Dispatcher getInstance ( void );
    public Yaf_Dispatcher disableView ( void );
    public Yaf_Dispatcher enableView ( void );
    public boolean autoRender ( bool $flag );
    public Yaf_Dispatcher returnResponse ( boolean $flag );
    public Yaf_Dispatcher flushInstantly ( boolean $flag );
    public Yaf_Dispatcher setErrorHandler ( mixed $callback ,
                                            int
$error_type = E_ALL / E_STRICT );
    public Yaf_Application getApplication ( void );
    public Yaf_Request_Abstract getRequest ( void );
    public Yaf_Router_Interface getRouter ( void );
    public Yaf_Dispatcher registerPlugin ( Yaf_Plugin_Abstract
$plugin );
    public Boolean setAppDirectory ( string $directory );
    public Yaf_Dispatcher setRequest ( Yaf_Request_Abstract
```

```

$request );
public Yaf_View_Interface initView ( void );
public Yaf_Dispatcher setView ( Yaf_View_Interface $view );
public Yaf_Dispatcher setDefaultModule ( string
$default_module_name );
public Yaf_Dispatcher setDefaultController ( string
$default_controller_name );
public Yaf_Dispatcher setDefaultAction ( string
$default_action_name );
public Yaf_Dispatcher throwException ( boolean $switch
= FALSE );
public Yaf_Dispatcher catchException ( boolean $switch
= FALSE );
public Yaf_Response_Abstract dispatch ( Yaf_Request_Abstract
$request );
}

```

属性说明

#### \_instance

Yaf\_Dispatcher实现了单利模式, 此属性保存当前实例

#### \_request

当前的请求

#### \_router

路由器, 在Yaf0.1之前, 路由器是可更改的, 但是Yaf0.2以后, 随着路由器和路由协议的分离, 各种路由都可以通过配置路由协议来实现, 也就取消了自定义路由器的功能

#### \_view

当前的视图引擎, 可以通过Yaf\_Dispatcher::setView来替换视图引擎为自定义视图引擎(比如Smarty/Firekylin等常见引擎)

#### \_plugins

已经注册的插件, 插件一经注册, 就不能更改和删除

#### \_render

标示着, 是否在动作执行完成后, 调用视图引擎的render方法, 产生响应. 可以通过

Yaf\_Dispatcher::disableView和Yaf\_Dispatcher::enableView来切换开关状态

---

### **`_return_response`**

标示着,是否在产生响应以后, 不自动输出给客户端, 而是返回给调用者. 可以通过Yaf\_Dispatcher::returnResponse来切换开关状态

---

### **`_instantly_flush`**

标示着, 是否在有输出的时候, 直接响应给客户端, 不写入Yaf\_Response\_Abstract对象.



注意

如果此属性为TRUE, 那么将忽略Yaf\_Dispatcher::\$\_return\_response

---

### **`_default_module`**

默认模块名, 在路由的时候, 如果没有指明模块, 则会使用这个值, 也可以通过配置文件中的ap.dispatcher.defaultModule来指定

---

### **`_default_controller`**

默认控制器名, 在路由的时候, 如果没有指明控制器, 则会使用这个值, 也可以通过配置文件中的ap.dispatcher.defaultController来指定

---

### **`_default_action`**

默认动作名, 在路由的时候, 如果没有指明动作, 则会使用这个值, 也可以通过配置文件中的ap.dispatcher.defaultAction来指定

[上一页](#)

[Yaf\\_Loader::clearLocalNamespace](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::getInstance](#)

## 名称

**Yaf\_Dispatcher::getInstance**

(Since Yaf 1.0.0.5)

```
public static Yaf_Dispatcher Yaf_Dispatcher::getInstance( void );
```

获取当前的Yaf\_Dispatcher实例

## 参数

**void**

该方法不需要参数

## 返回值

Yaf\_Dispatcher

## 例子

**例 11.18. Yaf\_Dispatcher::getInstance 的例子**

```
<?php  
$dispatcher = Yaf_Dispatcher::getInstance();  
?>
```



[上一页](#)

11.4. The Yaf\_Dispatcher class

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::disableView

## 名称

**Yaf\_Dispatcher::disableView**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::disableView( void );
```

关闭自动Render. 默认是开启的, 在动作执行完成以后, Yaf会自动render以动作名命名的视图模板文件.

## 参数

```
void
```

本方法不需要参数

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

## 例 11.19. Yaf\_Dispatcher::disableView的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    /**
     * Controller的init方法会被自动首先调用
     */
    public function init() {
        /**
         * 如果是Ajax请求, 则关闭HTML输出
         */
        if ($this->getRequest()->isXmlHttpRequest()) {
            Yaf_Dispatcher::getInstance()->disableView();
        }
    }
}
```

参见

**Yaf\_Dispatcher::  
enableView**

---

[上一页](#)

[Yaf\\_Dispatcher::getInstance](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::enableView](#)

## 名称

**Yaf\_Dispatcher::enableView**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::enableView( void );
```

开启自动Render。默认是开启的，在动作执行完成以后，Yaf会自动render以动作名命名的视图模板文件。

## 参数

**void**

本方法不需要参数

## 返回值

成功返回Yaf\_Dispatcher，失败返回FALSE

## 例子

## 例 11.20. Yaf\_Dispatcher::enableView的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    /**
     * Controller的init方法会被自动首先调用
     */
    public function init() {
        /**
         * 如果不是Ajax请求，则开启HTML输出
         */
        if (!$this->getRequest()->isXmlHttpRequest()) {
            Yaf_Dispatcher::getInstance()->enableView();
        }
    }
}
?>
```

参见

**Yaf\_Dispatcher::enableView**

[上一页](#)

Yaf\_Dispatcher::disableView

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::autoRender

## 名称

## Yaf\_Dispatcher::autoRender

(Since Yaf 1.0.0.11)

```
public boolean Yaf_Dispatcher::autoRender( boolean $switch );
```

开启/关闭自动渲染功能。在开启的情况下(Yaf默认开启), Action执行完成以后, Yaf会自动调用View引擎去渲染该Action对应的视图模板。

## 参数

*\$switch*

开启状态

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

## 例 11.21. Yaf\_Dispatcher::autoRender的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        if ($this->getRequest()->isXmlHttpRequest()) {
            //如果是Ajax请求, 关闭自动渲染, 由我们手工返回Json响应
            Yaf_Dispatcher::getInstance()->autoRender(FALSE);
        }
    }
}
?>
```

## 参见

```
Yaf_Dispatcher::enableView
Yaf_Dispatcher::
disableView
```

---

<a href="#">上一页</a>	<a href="#">上一级</a>	<a href="#">下一页</a>
Yaf_Dispatcher::enableView	<a href="#">起始页</a>	Yaf_Dispatcher::returnResponse

名称

Yaf\_Dispatcher::returnResponse

(Since Yaf 1.0.0.5)

```
public void Yaf_Dispatcher::returnResponse( boolean $switch );
```

是否返回Response对象, 如果启用, 则Response对象在分发完成以后不会自动输出给请求端, 而是交给程序员自己控制输出.

参数

*\$switch*

开启状态

返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

例子

例 11.22. Yaf\_Dispatcher::returnResponse的例子

```
<?php
$application = new Yaf_Application("config.ini");
```



```
/* 关闭自动响应，交给rd自己输出*/  
$response =  
$application->getDispatcher()->returnResponse(TRUE)->getApplication()->run();  
  
/** 输出响应*/  
$response->response();  
?>
```

参见

[Yaf\\_Dispatcher::enableView](#)

[Yaf\\_Dispatcher::  
disableView](#)

---

[上一页](#)

[Yaf\\_Dispatcher::autoRender](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::flushInstantly](#)

## 名称

**Yaf\_Dispatcher::flushInstantly**

(Since Yaf 1.0.0.5)

```
public void Yaf_Dispatcher::flushInstantly( boolean $switch );
```

切换自动响应. 在`Yaf_Dispatcher::enableView()`的情况下, 会使得Yaf\_Dispatcher调用Yaf\_Controller\_Abstract::display方法, 直接输出响应给请求端

## 参数

*\$switch*

开启状态

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

**例 11.23. Yaf\_Dispatcher::flushInstantly的例子**

```
<?php
$application = new Yaf_Application("config.ini");

/* 立即输出响应 */
Yaf_Dispatcher::getInstance()->flushInstantly(TRUE);

/* 此时会调用Yaf_Controller_Abstract::display方法 */
$application->run();

?>
```

参见

[Yaf\\_Dispatcher::enableView](#)

[Yaf\\_Dispatcher::disableView](#)

---

[上一页](#)

[Yaf\\_Dispatcher::returnResponse](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::setErrorHandler](#)

## 名称

### Yaf\_Dispatcher::setErrorHandler

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setErrorHandler( mixed $callback ,
                                                int $error_code = E_ALL | E_STRICT );
```

设置错误处理函数, 一般在[ap.throwException](#)关闭的情况下, Yaf会在出错的时候触发错误, 这个时候, 如果设置了错误处理函数, 则会把控制交给错误处理函数处理.

## 参数

### *\$callback*

错误处理函数, 这个函数需要最少接受俩个参数: 错误代码(*\$error\_code*)和错误信息(*\$error\_message*), 可选的还可以接受三个参数: 错误文件(*\$err\_file*), 错误行(*\$err\_line*)和错误上下文(*\$errcontext*)

### *\$error\_code*

要捕获的错误类型

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

### 例 11.24. Yaf\_Dispatcher::setErrorHandler的例子

```
<?php
/**
 * 一般可放在Bootstrap中定义错误处理函数
 */
function myErrorHandler($errno, $errstr, $errfile, $errline)
{
    switch ($errno) {
        case YAF_ERR_NOTFOUND_CONTROLLER:
        case YAF_ERR_NOTFOUND_MODULE:
        case YAF_ERR_NOTFOUND_ACTION:
            header("Not Found");
```

```
break;

default:
    echo "Unknown error type: [$errno] $errstr<br />\n";
    break;
}

return true;
}

Yaf_Dispatcher::getInstance()->setErrorHandler("myErrorHandler");
?>
```

参见

[异常和错误](#)

---

[上一页](#)

Yaf\_Dispatcher::flushInstantly

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::getApplication

## 名称

**Yaf\_Dispatcher::getApplication**

(Since Yaf 1.0.0.8)

```
public Yaf_Application Yaf_Dispatcher::getApplication( void );
```

获取当前的Yaf\_Application实例

## 参数

**void**

该方法不需要参数

## 返回值

Yaf\_Application实例

## 例子

例 11.25. Yaf\_Dispatcher::getApplication 的例子

```
<?php
$application = Yaf_Dispatcher::getInstance()->getApplication();
//不过，还是推荐大家使用
$application = Application::app();
?>
```

## 参见

[Yaf\\_Application::app](#)



## 名称

**Yaf\_Dispatcher::getRouter**

(Since Yaf 1.0.0.5)

```
public Yaf_Router Yaf_Dispatcher::getRouter( void );
```

获取路由器

## 参数

**void**

该方法不需要参数

## 返回值

Yaf\_Router实例

## 例子

**例 11.26. Yaf\_Dispatcher::getRouter 的例子**

```
<?php
$router = Yaf_Dispatcher::getInstance()->getRouter();
?>
```



## 名称

## Yaf\_Dispatcher::getRequest

(Since Yaf 1.0.0.5)

```
public Yaf_Request_Abstract Yaf_Dispatcher::getRequest( void );
```

获取当前的请求实例

## 参数

void

该方法不需要参数

## 返回值

Yaf\_Request\_Abstract实例

## 例子

例 11.27. Yaf\_Dispatcher::getRequest 的例子

```
<?php
$request = Yaf_Dispatcher::getInstance()->getRequest();
?>
```

## 名称

### Yaf\_Dispatcher::registerPlugin

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::registerPlugin( Yaf_Plugin_Abstract $plugin );
```

注册一个插件.

#### 参数

*\$plugin*

一个Yaf\_Plugin\_Abstract派生类的实例.

#### 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

#### 例子

例 11.28. Yaf\_Dispatcher::registerPlugin的例子

```
<?php
/**
 * 所有在Bootstrap类中, 以_init开头的方法, 都会被Yaf调用,
 * 这些方法, 都接受一个参数: Yaf_Dispatcher $dispatcher
 * 调用的次序, 和声明的次序相同
 */
```

```
class Bootstrap extends Yaf_Bootstrap_Abstract{
    /**
     * 注册一个插件
     * 插件的目录是在application_directory/plugins
     */
    public function _initPlugin(Yaf_Dispatcher $dispatcher) {
        $user = new UserPlugin();
        $dispatcher->registerPlugin($user);
    }
}

/**
 * 插件类定义
 * UserPlugin.php
 */
class UserPlugin extends Yaf_Plugin_Abstract {

    public function routerStartup(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
        echo "Plugin routerStartup called <br/>\n";
    }

    public function routerShutdown(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
        echo "Plugin routerShutdown called <br/>\n";
    }

    public function dispatchLoopStartup(Yaf_Request_Abstract
$request, Yaf_Response_Abstract $response) {
        echo "Plugin DispatchLoopStartup called <br/>\n";
    }

    public function preDispatch(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
        echo "Plugin PreDispatch called <br/>\n";
    }

    public function postDispatch(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
        echo "Plugin postDispatch called <br/>\n";
    }

    public function dispatchLoopShutdown(Yaf_Request_Abstract
$request, Yaf_Response_Abstract $response) {
        echo "Plugin DispatchLoopShutdown called <br/>\n";
    }

    public function preResponse(Yaf_Request_Abstract $request,
Yaf_Response_Abstract $response) {
        echo "Plugin PreResponse called <br/>\n";
    }
}
```



参见

---

[上一页](#)

Yaf\_Dispatcher::getRequest

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::setAppDirectory

## 名称

**Yaf\_Dispatcher::setAppDirectory**

(Since Yaf 1.0.0.0)

```
boolean Yaf_Dispatcher::setAppDirectory(string $directory);
```

改变APPLICATION\_PATH, 在这之后, 将从新的APPLICATION\_PATH下加载控制器/视图, 但注意, 不会改变自动加载的路径.

## 参数

*\$directory*

绝度路径的APPLICATION\_PATH

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

例 11.29. Yaf\_Dispatcher::setAppDirectory 的例子

```
<?php
$config = array(
    "ap" => array(
        "directory" => "/usr/local/www/ap",
    ),
);
```

```
);  
$app = new Yaf_Application($config);  
$app->getDispatcher()->setAppDirectory("/usr/local/new/application")->getApplication  
()->run();  
?>
```

---

[上一页](#)[Yaf\\_Dispatcher::registerPlugin](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Dispatcher::setRequest](#)

## 名称

**Yaf\_Dispatcher::setRequest**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setRequest( Yaf_Request_Abstract $request );
```

设置请求对象

## 参数

*\$request*

一个Yaf\_Request\_Abstract实例

*\$error\_code*

要捕获的错误类型

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

## 例 11.30. Yaf\_Dispatcher::setRequest的例子

```
<?php
$request = new Yaf_Request_Simple("Index", "Index", "index");
Yaf_Dispatcher::getInstance()->setRequest($request);
```

## 参见



[上一页](#)

Yaf\_Dispatcher::setAppDirectory

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::initView



## 名称

**Yaf\_Dispatcher::initView**

(Since Yaf 1.0.0.9)

```
public Yaf_View_Interface Yaf_Dispatcher::initView( string $tpl_dir );
```

初始化视图引擎, 因为Yaf采用延迟实例化视图引擎的策略, 所以只有在使用前调用此方法, 视图引擎才会被实例化



## 注意

如果你需要自定义视图引擎, 那么需要在调用Yaf\_Dispatcher::setView自定义视图引擎之后, 才可以调用此方法, 否则将得不到正确的视图引擎, 因为默认的此方法会实例化一个Yaf\_View\_Simple视图引擎

## 参数

***\$tpl\_dir***

视图的模板目录的绝对路径.

## 返回值

Yaf\_View\_Interface实例

例子

例 11.31. Yaf\_Dispatcher::initView 的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract {
    public function _initViewParameters(Yaf_Dispatcher $dispatcher) {
        $dispatcher->initView(APPLICATION_PATH . "/views/")->assign
("webroot", WEBROOT);
    }
}
?>
```

[上一页](#)[Yaf\\_Dispatcher::setRequest](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Dispatcher::setView](#)

名称

Yaf\_Dispatcher::setView

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setView( Yaf_View_Interface $request );
```

设置视图引擎

参数

*\$view*

一个实现了Yaf\_View\_Interface的视图引擎实例

返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

例子

例 11.32. Yaf\_Dispatcher::setView的例子

```
<?php
/**
 * 所有在Bootstrap类中, 以_init开头的方法, 都会被Yaf调用,
```

```

* 这些方法, 都接受一个参数: Yaf_Dispatcher $dispatcher
* 调用的次序, 和声明的次序相同
*/
class Bootstrap extends Yaf_Bootstrap_Abstract{
/**
 * 自定义视图引擎
 */
    public function _initSmarty(Yaf_Dispatcher $dispatcher) {
        $smarty = new Smarty_Adapter(null, Yaf_Registry::get("config")->get("smarty"));
        Yaf_Dispatcher::getInstance()->setView($smarty);
    }
}

/**
 * 视图引擎定义
 * Smarty/Adapter.php
 */
class Smarty_Adapter implements Yaf_View_Interface
{
    /**
     * Smarty object
     * @var Smarty
     */
    public $_smarty;

    /**
     * Constructor
     *
     * @param string $tmplPath
     * @param array $extraParams
     * @return void
     */
    public function __construct($tmplPath = null, $extraParams = array()) {

        require "Smarty.class.php";
        $this->_smarty = new Smarty;

        if (null !== $tmplPath) {
            $this->_smarty->setScriptPath($tmplPath);
        }

        foreach ($extraParams as $key => $value) {
            $this->_smarty->$key = $value;
        }
    }

    /**
     * Assign variables to the template
     *
     * Allows setting a specific key to the specified value, OR passing
     * an array of key => value pairs to set en masse.

```

```
*
* @see __set()
* @param string|array $spec The assignment strategy to use (key or
* array of key => value pairs)
* @param mixed $value (Optional) If assigning a named variable,
* use this as the value.
* @return void
*/
public function assign($spec, $value = null) {
    if (is_array($spec)) {
        $this->_smarty->assign($spec);
        return;
    }

    $this->_smarty->assign($spec, $value);
}

/**
 * Processes a template and returns the output.
 *
 * @param string $name The template to process.
 * @return string The output.
 */
public function render($name) {
    return $this->_smarty->fetch($name);
}
}
?>
```

参见

[上一页](#)

[Yaf\\_Dispatcher::initView](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::setDefaultController](#)

## 名称

**Yaf\_Dispatcher::setDefaultController**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setDefaultController( string $default_controller_name );
```

设置路由的默认控制器, 如果在路由结果中不包含控制器信息, 则会使用此默认控制器作为路由控制器结果

## 参数

***\$default\_controller\_name***

默认控制器名, 请注意需要首字母大写

## 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

## 例子

例 11.33. Yaf\_Dispatcher::setDefaultController的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initDefaultName(Yaf_Dispatcher $dispatcher) {
        /**
```

```
        * 这个只是举例，本身Yaf默认的就是"Index"
        */
        $dispatcher->setDefaultModule("Index")->setDefaultController
("Index")->setDefaultAction("index");
    }
}
```

参见

[Yaf\\_Dispatcher::](#)

[setDefaultModule](#)

[Yaf\\_Dispatcher::setDefaultAction](#)

---

[上一页](#)

[Yaf\\_Dispatcher::setView](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::setDefaultModule](#)

名称

Yaf\_Dispatcher::setDefaultModule

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setDefaultModule( string $default_module_name );
```

设置路由的默认模块, 如果在路由结果中不包含模块信息, 则会使用此默认模块作为路由模块结果

参数

*\$default\_module\_name*

默认模块名, 请注意需要首字母大写

返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

例子

例 11.34. Yaf\_Dispatcher::setDefaultModule的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initDefaultName(Yaf_Dispatcher $dispatcher) {
        /**
```



```
        * 这个只是举例，本身Yaf默认的就是"Index"
        */
        $dispatcher->setDefaultModule("Index")->setDefaultController
("Index")->setDefaultAction("index");
    }
}
```

参见

[Yaf\\_Dispatcher::](#)

[setDefaultController](#)

[Yaf\\_Dispatcher::setDefaultAction](#)

---

[上一页](#)

[Yaf\\_Dispatcher::setDefaultController](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::setDefaultAction](#)

名称

Yaf\_Dispatcher::setDefaultAction

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::setDefaultAction( string $default_module_name );
```

设置路由的默认动作, 如果在路由结果中不包含动作信息, 则会使用此默认动作作为路由动作结果

参数

*\$default\_module\_name*

默认动作名, 请注意需要全部小写

返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

例子

例 11.35. Yaf\_Dispatcher::setDefaultAction的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{

    public function _initDefaultName(Yaf_Dispatcher $dispatcher) {
        /**
```

```
    * 这个只是举例，本身Yaf默认的就是"Index"  
    */  
    $dispatcher->setDefaultController("Index")->setDefaultAction("index");  
}  
}
```

参见

[Yaf\\_Dispatcher::setDefaultModule](#)

[Yaf\\_Dispatcher::  
setDefaultController](#)

---

[上一页](#)

[Yaf\\_Dispatcher::setDefaultModule](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Dispatcher::throwException](#)

## 名称

### Yaf\_Dispatcher::throwException

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::throwException( boolean $switch );
```

切换在Yaf出错的时候抛出异常, 还是触发错误.

当然,也可以在配置文件中[使用](#)`ap.dispatcher.thorwException=$switch`达到同样的效果, 默认的是开启状态.

#### 参数

##### *\$switch*

如果为TRUE,则Yaf在出错的时候采用抛出异常的方式. 如果为FALSE, 则Yaf在出错的时候采用触发错误的方式.

#### 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

#### 例子

##### 例 11.36. Yaf\_Dispatcher::throwException的例子

```
<?php
$app = new Yaf_Application("conf.ini");
/**
 * 关闭抛出异常
 */
Yaf_Dispatcher::getInstance()->throwException(FALSE);
?>
```

参见

**Yaf\_Dispatcher::  
catchException**

---

[上一页](#)

Yaf\_Dispatcher::setDefaultAction

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::catchException

## 名称

### Yaf\_Dispatcher::catchException

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::catchException( boolean $switch );
```

在ap.dispatcher.throwException开启的状态下, 是否启用默认捕获异常机制

当然, 也可以在配置文件中使使用ap.dispatcher.catchException=\$switch达到同样的效果, 默认的是开启状态.

#### 参数

##### *\$switch*

如果为TRUE, 则在有未捕获异常的时候, Yaf会交给Error Controller的Error Action处理.

#### 返回值

成功返回Yaf\_Dispatcher, 失败返回FALSE

#### 例子

##### 例 11.37. Yaf\_Dispatcher::catchException的例子

```
<?php
$app = new Yaf_Application("conf.ini");
/**
 * 开启捕获异常
 */
Yaf_Dispatcher::getInstance()->catchException(TRUE);
?>
```

参见

**Yaf\_Dispatcher::  
throwException**

异常和错误

---

[上一页](#)

Yaf\_Dispatcher::throwException

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Dispatcher::dispatch

## 名称

### Yaf\_Dispatcher::dispatch

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Dispatcher::dispatch  
( Yaf_Request_Abstract $request );
```

开始处理流程, 一般的不需要用户调用此方法, `Yaf_Application::run` 会自动调用此方法.

#### 参数

*\$request*

一个Yaf\_Request\_Abstract实例

#### 返回值

成功返回一个Yaf\_Response\_Abstract实例, 错误会抛出异常.

#### 参见

`Yaf_Application::run`



---

[上一页](#)

Yaf\_Dispatcher::catchException

[上一级](#)

[起始页](#)

[下一页](#)

11.5. The Yaf\_Plugin\_Abstract  
class

## 11.5. The Yaf\_Plugin\_Abstract class

### 简介

Yaf\_Plugin\_Abstract是Yaf的插件基类, 所有应用在Yaf的插件都需要继承实现这个类, 这个类定义了7个方法, 依次在7个时机的时候被调用.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Plugin\Abstract`

```
abstract Yaf_Plugin_Abstract {
    public void routeStartup( Yaf_Request_Abstract $request ,
                             Yaf_Response_Abstract $response );
    public void routeShutdown( Yaf_Request_Abstract $request ,
                               Yaf_Response_Abstract $response );
    public void dispatchLoopStartup( Yaf_Request_Abstract
    $request ,
                                     Yaf_Response_Abstract
    $response );
    public void preDispatch( Yaf_Request_Abstract $request ,
                             Yaf_Response_Abstract $response );
    public void postDispatch( Yaf_Request_Abstract $request ,
                              Yaf_Response_Abstract $response );
    public void dispatchLoopShutdown( Yaf_Request_Abstract
    $request ,
                                     Yaf_Response_Abstract
    $response );
}
```



### 注意

插件有两种部署方式, 一种是部署在`plugins`目录下, 通过名称中的后缀(可通过`ap.name_suffix`和`ap.name_separator`来改变具体命名形式)来使得自动加载器可以正确加载. 另外一种是在类库, 由普通加载规则加载, 但无论哪种方式, 用户定义的插件都需要继承自 `Yaf_Plugin_Abstract`.

参见

**`Yaf_Dispatcher::registerPlugin`**

---

[上一页](#)

[Yaf\\_Dispatcher::dispatch](#)

[上一级](#)

[起始页](#)

[下一页](#)

[11.6. Yaf\\_Registry](#)

---

## 11.6. Yaf\_Registry

### 简介

Yaf\_Registry, 对象注册表(或称对象仓库)是一个用于在整个应用空间(application space)内存储对象和值的容器. 通过把对象存储在其中,我们可以在整个项目的任何地方使用同一个对象.这种机制相当于一种全局存储. 我们可以通过Yaf\_Registry类的静态方法来使用对象注册表. 另外,由于该类是一个数组对象,你可以使用数组形式来访问其中的类方法.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Registry`

```
Yaf_Registry {
    public static Yaf_Registry has ( string $name );
    public static Yaf_Registry get ( string $name );
    public static Yaf_Registry set ( string $name ,
                                     mixed $value );
    public static Yaf_Registry del ( string $name );
}
```

名称

Yaf\_Registry::set

(Since Yaf 1.0.0.5)

```
public static Yaf_Registry Yaf_Registry::set( string $name ,  
                                              mixed $value );
```

往全局注册表添加一个新的项

参数

*\$name*

要注册的项的名字

*\$value*

要注册的项的值

返回值

Yaf\_Registry

例子

## 例 11.38. Yaf\_Registry::set 的例子

```
<?php
/** 存入 */
Yaf_Registry::set('config', Yaf_Application::app()->getConfig());

/* 之后可以在任何地方获取到 */
$config->Yaf_Registry::get("config");
?>
```

[上一页](#)[11.6. Yaf\\_Registry](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Registry::get](#)

名称

Yaf\_Registry::get

(Since Yaf 1.0.0.5)

```
public static Yaf_Registry Yaf_Registry::get( string $name );
```

获取注册表中寄存的项

参数

*\$name*

要获取的项的名字

返回值

成功返回要获取的注册项的值, 失败返回FALSE

例子

例 11.39. Yaf\_Registry::get 的例子

```
<?php
/** 存入 */
Yaf_Registry::set('config', Yaf_Application::app()->getConfig());

/* 之后可以在任何地方获取到 */
$config->Yaf_Registry::get("config");
?>
```

---

[上一页](#)[Yaf\\_Registry::set](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Registry::has](#)



名称

Yaf\_Registry::has

(Since Yaf 1.0.0.5)

```
public static Yaf_Registry Yaf_Registry::has( string $name );
```

查询某一项目是否存在于注册表中

参数

*\$name*

要查询的项的名字

返回值

存在返回TRUE, 不存在返回FALSE

例子

例 11.40. Yaf\_Registry::has 的例子

```
<?php
/** 存入 */
Yaf_Registry::set('config', Yaf_Application::app()->hasConfig());

assert(Yaf_Registry::has("config"));
?>
```

---

[上一页](#)[Yaf\\_Registry::get](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Registry::del](#)

名称

Yaf\_Registry::del

(Since Yaf 1.0.0.5)

```
public static Yaf_Registry Yaf_Registry::del ( string $name );
```

删除存在于注册表中的名为\$name的项目

参数

*\$name*

要删除的项的名字

返回值

成功返回TRUE, 失败返回FALSE

例子

例 11.41. Yaf\_Registry::del 的例子

```
<?php
/** 存入 */
Yaf_Registry::set('config', Yaf_Application::app()->delConfig());

Yaf_Registry::del("config");
?>
```

---

[上一页](#)[Yaf\\_Registry::has](#)[上一级](#)[起始页](#)[下一页](#)[11.7. Yaf\\_Session](#)

---

## 11.7. Yaf\_Session

### 简介

Yaf\_Session是Yaf对Session的包装, 实现了Iterator, ArrayAccess, Countable接口, 方便使用.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Session`

```
final Yaf_Session implements Iterator , ArrayAccess , Countable {
    public static Yaf_Session getInstance ( void );
    public Yaf_Session start ( void );
    public mixed get ( string $name = NULL );
    public boolean set ( string $name ,
                        mixed $value );
    public mixed __get ( string $name );
    public boolean __set ( string $name ,
                        mixed $value );
    public boolean has ( string $name );
    public boolean del ( string $name );
    public boolean __isset ( string $name );
    public boolean __unset ( string $name );
}
```

## 11.8. The Yaf\_Config\_Abstract class

### 简介

Yaf\_Config\_Abstract被设计在应用程序中简化访问和使用配置数据。它为在应用程序代码中访问这样的配置数据提供了一个基于用户接口的嵌入式对象属性。配置数据可能来自于各种支持等级结构数据存储的媒体。Yaf\_Config\_Abstract实现了Countable, ArrayAccess 和 Iterator 接口。这样, 可以基于Yaf\_Config\_Abstract对象使用count()函数和PHP语句如foreach, 也可以通过数组方式访问Yaf\_Config\_Abstract的元素。

Yaf\_Config\_Ini为存储在Ini文件的配置数据提供了适配器。Yaf\_Config\_Simple为存储在PHP的数组中的配置数据提供了适配器。

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Config\Abstract`

```
Abstract Yaf_Config_Abstract implements Iterator , ArrayAccess , Countable {
    protected array _config ;
    protected array _readonly ;
    public mixed get ( string $name = NULL );
    public mixed __get ( string $name );
    public mixed __isset ( string $name );
    public mixed __set ( string|int $name ,
                        mixed $value );
    public mixed set ( string|int $name ,
                     mixed $value );
    public mixed count ( void );
    public mixed offsetGet ( string|int $name );
    public mixed offsetSet ( string|int $name ,
                           mixed $value );
    public mixed offsetExists ( string|int $name );
    public mixed offsetUnset ( string|int $name );
    public void rewind ( void );
    public mixed key ( void );
    public mixed next ( void );
    public mixed current ( void );
    public boolean valid ( void );
    public array toArray ( void );
    public boolean readonly ( void );
}
```

属性说明

#### \_config

配置实际的保存容器

#### \_readonly

表示配置是否容许修改, 对于Yaf\_Config\_Ini来说, 永远都是TRUE

## The Yaf\_Config\_Ini class

### 简介

Yaf\_Config\_Ini为存储在Ini文件的配置数据提供了适配器。

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Config\_Ini



#### 重要

当使用INI文件作为Yaf\_Application的配置的时候, 可以打开[ap.cache\\_config](#)来提升性能

### 说明

Yaf\_Config\_Ini允许开发者通过嵌套的对象属性语法在应用程序中用熟悉的 INI 格式存储和读取配置数据。INI格式在提供拥有配置数据键的等级结构和配置数据节之间的继承能力方面具有专长。配置数据等级结构通过用点或者句号(.)分离键值。一个节可以扩展或者通过在节的名称之后带一个冒号(:)和被继承的配置数据的节的名称来从另一个节继承。

#### 例 11.42. INI 文件

```
[base]
database.master.host = localhost
[production : base]
; Yaf的配置
application.directory = /usr/local/www/production
; 应用的配置
webhost = www.example.com
database.adapter = pdo_mysql
database.params.host = db.example.com
database.params.username = dbuser
database.params.password = secret
database.params.dbname = dbname
; 开发站点配置数据从生产站点配置数据集成并如果需要可以重写
[dev : production]
application.directory = /usr/dev/htdocs
database.params.host = dev.example.com
database.params.username = devuser
database.params.password = devsecret
```

dev节, 将得到production节的所有配置, 并间接获得base节的配置 并且覆盖application.directory的配置为"/usr/dev/htdocs"

Yaf\_Config\_Abstract实现了\_\_get方法, 所以获取配置将会变得很容易

#### 例 11.43. 获取配置

```
$config = new Yaf_Config_Ini('/path/to/config.ini', 'staging');
echo $config->database->get("params")->host; // 输出 "dev.example.com"
echo $config->get("database")->params->dbname; // 输出 "dbname"
```

## The Yaf\_Config\_Simple class

## 简介

Yaf\_Config\_Simple为存储在数组中的配置数据提供了适配器。

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Config\_Simple

---

[上一页](#)[11.7. Yaf\\_Session](#)[上一级](#)[起始页](#)[下一页](#)[11.9. The Yaf\\_Controller\\_Abstract class](#)



## 11.9. The Yaf\_Controller\_Abstract class

### 简介

**Yaf\_Controller\_Abstract**是Yaf的MVC体系的核心部分。MVC是指Model-View-Controller, 是一个用于分离应用逻辑和表现逻辑的设计模式。

**Yaf\_Controller\_Abstract**体系具有可扩展性, 可以通过继承已有的类, 来实现这个抽象类, 从而添加应用自己的应用逻辑。

对于Controller来说, 真正的执行体是在Controller中定义的一个一个的动作, 当然这些动作也可以定义在Controller外: 参看**[Yaf\\_Controller\\_Abstract::\\$action](#)**

与一般的框架不同, 在Yaf中, 可以定义动作的参数, 这些参数的值来自对Request的路由结果中的同名参数值。比如对于如下的控制器:

#### 例 11.44. Yaf\_Controller\_Abstract参数动作 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction($name, $value) {
    }
}
?>
```

在使用默认路由的情况下, 对于请求<http://domain.com/index/index/name/a/value/2>我们知道会在Request对象中生成两个参数name和value, 而注意到动作indexAction的参数, 与此同名, 于是在indexAction中, 可以有如下两种方式来获取这两个参数:

#### 例 11.45. Yaf\_Controller\_Abstract参数动作 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction($name, $value) {
        //直接获取参数;
        echo $name, $value; //a2
        //通过Request对象获取
        echo $this->getRequest()->getParam("name"); //a
    }
}
?>
```

**注意**

需要注意的是, 这些参数是来自用户请求URL, 所以使用前一定要做安全化过滤. 另外, 为了防止PHP抛出参数缺失的警告, 请尽量定义有默认值的参数.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Controller\Abstract`.

```
abstract Yaf_Controller_Abstract {
    protected array actions ;
    protected Yaf_Request_Abstract _request ;
    protected Yaf_Response_Abstract _response ;
    protected Yaf_View_Interface _view ;
    protected string _script_path ;
    private void __construct ( void );
    public void init ( void );
    public string getModuleName ( void );
    public Yaf_Request_Abstract getRequest ( void );
    public Yaf_Response_Abstract getResponse ( void );
    public Yaf_View_Interface getView ( void );
    public Yaf_View_Interface initView ( void );
    public boolean setViewPath ( string $view_directory );
    public string getViewPath ( void );
    public Yaf_Response_Abstract render ( string $action_name ,
                                         array $tpl_vars = NULL );
    public boolean display ( string $action_name ,
                             array $tpl_vars = NULL );
    public boolean forward ( string $action ,
                             array $invoke_args = NULL );
    public boolean forward ( string $controller ,
                             string $action ,
                             array $invoke_args = NULL );
    public boolean forward ( string $module ,
                             string $controller ,
                             string $action ,
                             array $invoke_args = NULL );
    public boolean redirect ( string $url );
}
```

属性说明

## actions

有些时候为了拆分比较大的Controller, 使得代码更加清晰和易于管理, Yaf支持将具体的动作分开定义. 每个动作都需要实现 `Yaf_Action_Abstract` 就可以通过定义`Yaf_Controller_Abstract:: $actions`来指明那些动作对应于具体的那些分离的类. 比如:

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public $actions = array (
        "index" => "actions/Index.php",
    );
}
```

这样, 当路由到动作Index的时候, 就会加载APPLICATION\_PATH . "/actions/Index.php", 并且在这个脚本文件中寻找IndexAction(可通过[ap.name\\_suffix](#)和[ap.name\\_separator](#)来改变具体命名形式), 继而调用这个类的execute方法.



注意

在[ap.st\\_compatible](#)打开的情况下, 会产生额外的查找逻辑.

### \_request

当前的请求实例, 属性的值由Yaf\_Dispatcher保证, 一般通过Yaf\_Controller\_Abstract::getRequest来获取此属性.

### \_response

当前的响应对象, 属性的值由Yaf\_Dispatcher保证, 一般通过Yaf\_Controller\_Abstract::getResponse来获取此属性.

### \_view

视图引擎, Yaf才会延时实例化视图引擎来提高性能, 所以这个属性直到显示的调用了Yaf\_Controller\_Abstract::getView或者Yaf\_Controller\_Abstract::initView以后才可用

### \_script\_path

视图文件的目录, 默认值由Yaf\_Dispatcher保证, 可以通过Yaf\_Controller\_Abstract::setViewPath来改变这个值.

参见

[Yaf\\_Action\\_Abstract](#)

[上一页](#)

11.8. The Yaf\_Config\_Abstract class

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Controller\_Abstract::getModuleName

## 名称

### Yaf\_Controller\_Abstract::getModuleName

(Since Yaf 1.0.0.5)

```
public string Yaf_Controller_Abstract::getModuleName( void );
```

获取当前控制器所属的模块名

#### 参数

void

本方法不需要参数

#### 返回值

成功返回模块名,失败返回NULL

#### 例子

例 11.46. Yaf\_Controller\_Abstract::getModuleName 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        echo $this->getModuleName();
    }
}
?>
```



## 名称

### Yaf\_Controller\_Abstract::getRequest

(Since Yaf 1.0.0.5)

```
public Yaf_Request_Abstract Yaf_Controller_Abstract::getRequest( void );
```

获取当前的请求实例

## 参数

void

该方法不需要参数

## 返回值

Yaf\_Request\_Abstract实例

## 例子

例 11.47. Yaf\_Controller\_Abstract::getRequest 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $request = $this->getRequest();
    }
}
?>
```

Yaf\_Controller\_Abstract::getModuleName

[起始页](#)

Yaf\_Controller\_Abstract::getResponse

## 名称

### Yaf\_Controller\_Abstract::getResponse

(Since Yaf 1.0.0.5)

```
public Yaf_Response_Abstract Yaf_Controller_Abstract::getResponse( void );
```

获取当前的响应实例

## 参数

void

该方法不需要参数

## 返回值

Yaf\_Response\_Abstract实例

## 例子

例 11.48. Yaf\_Controller\_Abstract::getResponse 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $response = $this->getResponse();
    }
}
?>
```



Yaf\_Controller\_Abstract::getRequest

[起始页](#)

Yaf\_Controller\_Abstract::getView

## 名称

### Yaf\_Controller\_Abstract::getView

(Since Yaf 1.0.0.5)

```
public Yaf_View_Interface Yaf_Controller_Abstract::getView( void );
```

获取当前的视图引擎

## 参数

void

该方法不需要参数

## 返回值

Yaf\_View\_Interface实例

## 例子

### 例 11.49. Yaf\_Controller\_Abstract::getView 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $view = $this->getView();
    }
}
?>
```

Yaf\_Controller\_Abstract::getResponse

[起始页](#)

Yaf\_Controller\_Abstract::initView

## 名称

**Yaf\_Controller\_Abstract::initView**

(Since Yaf 1.0.0.5)

```
public Yaf_View_Interface Yaf_Controller_Abstract::initView( void );
```

初始化视图引擎，因为Yaf采用延迟实例化视图引擎的策略，所以只有在使用前调用此方法，视图引擎才会被实例化

## 参数

**void**

该方法不需要参数

## 返回值

Yaf\_View\_Interface实例

## 例子

**例 11.50. Yaf\_Controller\_Abstract::initView 的例子**

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $view = $this->initView();
        /* 此后就可以直接通过获取Yaf_Controller_Abstract::$_view
        来访问当前视图引擎 */

        $this->_view->assign("webroot", "http://domain.com/");
    }
}
?>
```



## 名称

### Yaf\_Controller\_Abstract::setViewPath

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Controller_Abstract::setViewPath( string $view_directory );
```

更改视图模板目录, 之后Yaf\_Controller\_Abstract::render就会在整个目录下寻找模板文件

#### 参数

*\$view\_directory*

视图模板目录, 绝对目录.

#### 返回值

成功返回Yaf\_Controller\_Abstract, 失败返回FALSE

#### 例子

例 11.51. Yaf\_Controller\_Abstract::setViewPath 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->setViewPath("/usr/local/www/tpl/");
    }
}
?>
```

#### 参见

Yaf\_Controller\_Abstract::render  
Yaf\_Controller\_Abstract:::  
getViewPath

[上一页](#)

Yaf\_Controller\_Abstract::initView

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Controller\_Abstract::getViewPath

## 名称

### Yaf\_Controller\_Abstract::getViewPath

(Since Yaf 1.0.0.5)

```
public string Yaf_Controller_Abstract::getViewPath( void );
```

获取当前的模板目录

## 参数

void

本方法不需要参数

## 返回值

成功返回模板目录,失败返回NULL

## 例子

例 11.52. Yaf\_Controller\_Abstract::getViewPath 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        echo $this->getViewPath();
    }
}
?>
```

## 参见



Yaf\_Controller\_Abstract::setViewPath

[上一页](#)

Yaf\_Controller\_Abstract::setViewPath

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Controller\_Abstract::render

## 名称

**Yaf\_Controller\_Abstract::render**

(Since Yaf 1.0.0.5)

```
public Yaf_Response_Abstract Yaf_Controller_Abstract::render( string $action ,  
                                                             array $tpl_vars  
= NULL );
```

渲染视图模板, 得到渲染结果



注意

此方法是对Yaf\_View\_Interface::render的包装

## 参数

*\$action*

要渲染的动作名

*\$tpl\_vars*

传递给视图引擎的渲染参数, 当然也可以使用Yaf\_View\_Interface::assign来替代

## 返回值

Yaf\_Response\_Abstract实例

## 例子

## 例 11.53. Yaf\_Controller\_Abstract::render 的例子

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {
```

```
        /* 首先关闭自动渲染 */
        Yaf_Dispatcher::getInstance()->disableView();
    }

    public function indexAction() {
        $this->initView();

        /* 自己输出响应 */
        echo $this->render("test.phtml");
    }
?>
```

---

[上一页](#)[Yaf\\_Controller\\_Abstract::getViewPath](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Controller\\_Abstract::display](#)

## 名称

**Yaf\_Controller\_Abstract::display**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Controller_Abstract::display( string $action ,  
                                                array $tpl_vars = NULL );
```

渲染视图模板, 并直接输出渲染结果



注意

此方法是对Yaf\_View\_Interface::display的包装

## 参数

***\$action***

要渲染的动作名

***\$tpl\_vars***

传递给视图引擎的渲染参数, 当然也可以使用Yaf\_View\_Interface::assign来替代

## 返回值

成功返回TRUE, 失败返回FALSE

## 例子

## 例 11.54. Yaf\_Controller\_Abstract::display 的例子

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {  
        /* 首先关闭自动渲染 */  
        Yaf_Dispatcher::getInstance()->disableView();  
    }  
  
    public function indexAction() {  
        $this->initView();  
    }  
}
```

```
        /* 自己输出响应 */  
        $this->display("test.phtml", array("name" => "value"));  
    }  
}  
?>
```

---

[上一页](#)[Yaf\\_Controller\\_Abstract::render](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Controller\\_Abstract::forward](#)

## 名称

## Yaf\_Controller\_Abstract::forward

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Controller_Abstract::forward( string $action ,  
                                                array $params = NULL );
```

```
public boolean Yaf_Controller_Abstract::forward( string $controller ,  
                                                string $action ,  
                                                array $params = NULL );
```

```
public boolean Yaf_Controller_Abstract::forward( string $module ,  
                                                string $controller ,  
                                                string $action ,  
                                                array $params = NULL );
```

将当前请求转给另外一个动作处理



## 注意

Yaf\_Controller\_Abstract::forward只是登记下要forward的目的地, 并不会立即跳转. 而是会等到当前的Action执行完成以后, 才会进行新一轮dispatch.

## 参数

***\$module***

要转给动作的模块, 注意要首字母大写, 如果为空, 则转给当前模块

***\$controller***

要转给动作的控制器, 注意要首字母大写, 如果为空, 则转给当前控制器

***\$action***

要转给的动作, 注意要全部小写

***\$params***关联数组, 附加的参数, 可通过[Yaf\\_Request\\_Abstract::getParam](#)获取

返回值

成功返回Yaf\_Controller\_Abstract, 失败返回FALSE

例子

例 11.55. Yaf\_Controller\_Abstract::forward 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        /**
         * 如果用户没登陆, 则转给登陆动作
         */
        if($user_not_login) {
            $this->forward("login");
        }
    }
}
?>
```

[上一页](#)

[Yaf\\_Controller\\_Abstract::display](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Controller\\_Abstract::redirect](#)

## 名称

**Yaf\_Controller\_Abstract::redirect**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Controller_Abstract::redirect( string $url );
```

重定向请求到新的路径

## 参数

*\$url*

要定向的路径

## 返回值

成功返回Yaf\_Controller\_Abstract, 失败返回FALSE

## 例子

例 11.56. Yaf\_Controller\_Abstract::redirect 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        if($user_not_login)
            $this->redirect("/login/");
    }
}
?>
```





---

## 11.10. The Yaf\_Action\_Abstract class

### 简介

Yaf\_Action\_Abstract是MVC中C的动作, 一般而言动作都是定义在Yaf\_Controller\_Abstract的派生类中的, 但是有的时候, 为了使得代码清晰, 分离一些大的控制器, 则可以采用单独定义Yaf\_Action\_Abstract来实现.

Yaf\_Action\_Abstract体系具有可扩展性, 可以通过继承已有的类, 来实现这个抽象类, 从而添加应用自己的应用逻辑.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Action_Abstract`.

```
abstract Yaf_Action_Abstract extends Yaf_Action_Controller {  
    public abstract void execute ( void );  
}
```

参见

[Yaf\\_Controller\\_Abstract](#)

## 11.11. The Yaf\_View\_Interface class

### 简介

Yaf\_View\_Interface是为了提供可扩展的, 可自定的视图引擎而设立的视图引擎接口, 它定义了用在Yaf上的视图引擎需要实现的方法和功能.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 `Yaf\View_Interface`

```
interface Yaf_View_Interface {
    public string render( string $view_path ,
                        array $tpl_vars = NULL );
    public boolean display( string $view_path ,
                        array $tpl_vars = NULL );
    public boolean assign( mixed $name ,
                        mixed $value = NULL );
    public boolean setScriptPath( string $view_directory );
    public string getScriptPath( void );
}
```

## The Yaf\_View\_Simple class

### 简介

Yaf\_View\_Simple是Yaf自带的视图引擎, 它追求性能, 所以并没有提供类似Smarty那样的多样功能, 和复杂的语法.

对于Yaf\_View\_Simple的视图模板, 就是普通的PHP脚本, 对于通过Yaf\_View\_Interface::assign的模板变量, 可在视图模板中直接通过变量名使用.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 `Yaf\View_Simple`

```
Yaf_View_Simple extends Yaf_View_Interface {
    protected array $_tpl_vars ;
    protected string $_script_path ;
    public string render ( string $view_path ,
                        array $tpl_vars
    = NULL );
```

```

    public boolean display ( string $view_path ,
                                array $tpl_vars
    = NULL );
    public boolean setScriptPath ( string $view_directory );
    public string getScriptPath ( void );
    public boolean assign ( string $name ,
                                mixed $value );
    public boolean __set ( string $name ,
                                mixed $value
    = NULL );
    public mixed __get ( string $name );
}

```

属性说明

### \_tpl\_vars

所有通过Yaf\_View\_Simple::assign分配的变量, 都保存在此属性中

### \_script\_path

当前视图引擎的模板文件基目录

[上一页](#)

[上一级](#)

[下一页](#)

11.10. The Yaf\_Action\_Abstract  
class

[起始页](#)

Yaf\_View\_Simple::assign

## 名称

## Yaf\_View\_Simple::assign

(Since Yaf 1.0.0.0)

```
public boolean Yaf_View_Simple::assign( mixed $name ,  
                                         mixed $value = NULL );
```

为视图引擎分配一个模板变量, 在视图模板中可以直接通过`${$name}`获取模板变量值

## 参数

*\$name*

字符串或者关联数组, 如果为字符串, 则`$value`不能为空, 此字符串代表要分配的变量名. 如果为数组, 则`$value`须为空, 此参数为变量名和值的关联数组.

*\$value*

分配的模板变量值



## 注意

如果`$name`不是合法的PHP变量名, 比如整数,或者是包含"`|`"的字符串, 那么在视图模板文件中, 将不能直接通过`${$name}`来访问这个变量. 当然, 你还是可以在视图模板文件中通过`$this->_tpl_vars[$name]`来访问这个变量.

## 返回值

成功返回Yaf\_View\_Simple, 失败返回FALSE

## 例子

## 例 11.57. Yaf\_View\_Simple::assign 的例子

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {  
        $params = array(  
            'name' => 'value',  
        );  
    }  
}
```

```
        $this->getView()->assign($params)->assign("foo", "bar");  
    }  
}  
?>
```

参见

[Yaf\\_View\\_Simple::\\_\\_set](#)

---

[上一页](#)

11.11. The Yaf\_View\_Interface class

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_View\\_Simple::render](#)

## 名称

### Yaf\_View\_Simple::render

(Since Yaf 1.0.0.0)

```
public string Yaf_View_Simple::render( string $view_path ,  
                                       array $tpl_vars = NULL );
```

渲染一个视图模板, 得到结果

#### 参数

##### *\$view\_path*

视图模板的文件, 绝对路径, 一般这个路径由Yaf\_Controller\_Abstract提供

##### *\$tpl\_vars*

关联数组, 模板变量

#### 返回值

成功返回视图模板执行结果, 失败返回NULL

#### 例子

例 **11.58. Yaf\_View\_Simple::render** 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        echo $this->getView()->render($this->_script_path . "/test.phtml");
    }
}
?>
```

[上一页](#)[Yaf\\_View\\_Simple::assign](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_View\\_Simple::display](#)



## 名称

### Yaf\_View\_Simple::display

(Since Yaf 1.0.0.0)

```
public string Yaf_View_Simple::display( string $view_path ,  
                                         array $tpl_vars = NULL );
```

渲染一个视图模板, 并直接输出给请求端

#### 参数

##### *\$view\_path*

视图模板的文件, 绝对路径, 一般这个路径由Yaf\_Controller\_Abstract提供

##### *\$tpl\_vars*

关联数组, 模板变量

#### 返回值

成功返回TRUE, 失败返回FALSE

#### 例子

**例 11.59. Yaf\_View\_Simple::display 的例子**

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        $this->getView()->display($this->_script_path . "/test.phtml");
    }
}
?>
```

[上一页](#)[Yaf\\_View\\_Simple::render](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_View\\_Simple::setScriptPath](#)

## 名称

### Yaf\_View\_Simple::setScriptPath

(Since Yaf 1.0.0.13)

```
public boolean Yaf_View_Simple::setScriptPath( string $view_directory );
```

设置模板的基目录, 默认的Yaf\_Dispatcher会设置此目录为APPLICATION\_PATH . "/views".

#### 参数

*\$view\_directory*

视图模板的基目录, 绝对地址

#### 返回值

成功返回TRUE, 失败返回FALSE

#### 例子

例 11.60. Yaf\_View\_Simple::setScriptPath 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        $this->getView()->setScriptPath("/tmp/views/");
    }
}
?>
```

Yaf\_View\_Simple::display

[起始页](#)

Yaf\_View\_Simple::getScriptPath

## 名称

### Yaf\_View\_Simple::getScriptPath

(Since Yaf 1.0.0.13)

```
public string Yaf_View_Simple::getScriptPath( void );
```

获取当前的模板目录

#### 参数

void

此方法不需要参数

#### 返回值

成功返回目前的视图目录, 失败返回NULL

#### 例子

例 11.61. Yaf\_View\_Simple::getScriptPath 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        echo $this->getView()->getScriptPath();
    }
}
?>
```



## 名称

### Yaf\_View\_Simple::\_\_set

(Since Yaf 1.0.0.0)

```
public boolean Yaf_View_Simple::__set( mixed $name ,
                                       mixed $value = NULL );
```

为视图引擎分配一个模板变量, 在视图模板中可以直接通过`${$name}`获取模板变量值

#### 参数

##### *\$name*

字符串或者关联数组, 如果为字符串, 则\$value不能为空, 此字符串代表要分配的变量名. 如果为数组, 则\$value须为空, 此参数为变量名和值的关联数组.

##### *\$value*

分配的模板变量值

#### 返回值

成功返回Yaf\_View\_Simple, 失败返回FALSE

#### 例子

##### 例 11.62. Yaf\_View\_Simple::\_\_set 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->getView()->name = "value";
    }
}
?>
```

参见

Yaf\_View\_Simple::  
assign

[上一页](#)

Yaf\_View\_Simple::getScriptPath

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_View\_Simple::\_\_get



## 名称

### Yaf\_View\_Simple::\_\_get

(Since Yaf 1.0.0.0)

```
public string Yaf_View_Simple::__get( string $name );
```

获取视图引擎的一个模板变量值

#### 参数

*\$name*

模板变量名

#### 返回值

成功返回变量值,失败返回NULL

#### 例子

例 11.63. Yaf\_View\_Simple::\_\_get 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->initView();
    }
    public function indexAction() {
        //通过__get直接获取变量值
        echo $this->_view->name;
    }
}
?>
```

参见

Yaf\_View\_Simple:::  
get

[上一页](#)

Yaf\_View\_Simple::\_\_set

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_View\_Simple::get

## 名称

### Yaf\_View\_Simple::get

(Since Yaf 1.0.0.0)

```
public string Yaf_View_Simple::get( string $name );
```

获取视图引擎的一个模板变量值

#### 参数

*\$name*

模板变量名

#### 返回值

成功返回变量值,失败返回NULL

#### 例子

例 11.64. Yaf\_View\_Simple::get 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->initView();
    }
    public function indexAction() {
        echo $this->_view->get("name");
    }
}
?>
```

参见

Yaf\_View\_Simple::  
\_\_get

[上一页](#)

Yaf\_View\_Simple::\_\_get

[上一级](#)

[起始页](#)

[下一页](#)

11.12. The Yaf\_Request\_Abstract class

## 11.12. The Yaf\_Request\_Abstract class

### 简介

代表了一个实际请求, 一般的不用自己实例化它, Yaf\_Application在run以后会自动根据当前请求实例它

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Request\_Abstract

```
abstract class Yaf_Request_Abstract {
    protected string $_method ;
    protected string $_module ;
    protected string $_controller ;
    protected string $_action ;
    protected array $_params ;
    protected string $_language ;
    protected string $_base_uri ;
    protected string $_request_uri ;
    protected boolean $_dispatched ;
    protected boolean $_routed ;
    public string getModuleName ( void );
    public string getControllerName ( void );
    public string getActionName ( void );
    public boolean setModuleName ( string $name );
    public boolean setControllerName ( string $name );
    public boolean setActionName ( string $name );
    public Exception getException ( void );
    public mixed getParams ( void );
    public mixed getParam ( string $name ,
                                mixed $default
    = NULL );
    public mixed setParam ( string $name ,
                                mixed $value );
    public mixed getMethod ( void );
    abstract public mixed getLang ( void );
    abstract public mixed getQuery ( string $name = NULL );
    abstract public mixed getPost ( string $name = NULL );
    abstract public mixed getEnv ( string $name = NULL );
}
```

```

abstract public mixed  getServer ( string  $name = NULL );
abstract public mixed  getCookie ( string  $name = NULL );
abstract public mixed  getFiles ( string  $name = NULL );
abstract public bool   isGet ( void );
abstract public bool   isPost ( void );
abstract public bool   isHead ( void );
abstract public bool   isXmlHttpRequest ( void );
abstract public bool   isPut ( void );
abstract public bool   isDelete ( void );
abstract public bool   isOption ( void );
abstract public bool   isCli ( void );
public bool   isDispatched ( void );
public bool   setDispatched ( void );
public bool   isRouted ( void );
public bool   setRouted ( void );
}

```

属性说明

#### \_method

当前请求的Method, 对于命令行来说, Method为"CLI"

#### \_language

当前请求的希望接受的语言, 对于Http请求来说, 这个值来自分析请求头Accept-Language. 对于不能鉴别的情况, 这个值为NULL.

#### \_module

在路由完成后, 请求被分配到的模块名

#### \_controller

在路由完成后, 请求被分配到的控制器名

#### \_action

在路由完成后, 请求被分配到的动作名

#### \_params

当前请求的附加参数

#### \_routed

表示当前请求是否已经完成路由

### \_dispatched

表示当前请求是否已经完成分发

### \_request\_uri

当前请求的Request URI

### \_base\_uri

当前请求Request URI要忽略的前缀, 一般不需要手工设置, Yaf会自己分析. 只是当Yaf分析出错的时候, 可以通过[application.baseUri](#)来手工设置.

## The Yaf\_Request\_Http class

简介

代表了一个实际的Http请求, 一般的不用自己实例化它, Yaf\_Application在run以后会自动根据当前请求实例它

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Request\_Http

```
final Yaf_Request_Http extends Yaf_Request_Abstract {
    public void __construct ( string $request_uri = NULL ,
                                string
                                $base_uri = NULL );
    public mixed getLang ( void );
    public mixed getQuery ( string $name = NULL );
    public mixed getPost ( string $name = NULL );
    public mixed getEnv ( string $name = NULL );
    public mixed getServer ( string $name = NULL );
    public mixed getCookie ( string $name = NULL );
    public mixed getFiles ( string $name = NULL );
    public bool isGet ( void );
    public bool isPost ( void );
    public bool isHead ( void );
    public bool isXmlHttpRequest ( void );
    public bool isPut ( void );
    public bool isDelete ( void );
    public bool isOption ( void );
    public bool isCli ( void );
    public bool isDispatched ( void );
```

```

public bool  setDispatched ( void );
public bool  isRouted ( void );
public bool  setRouted ( void );
public string getBaseUri ( void );
public boolean setBaseUri ( string $base_uri );
public string getRequestUri ( void );
}

```

属性说明

## The Yaf\_Request\_Simple class

简介

代表了一个实际的请求, 一般的不用自己实例化它, Yaf\_Application在run以后会自动根据当前请求实例它

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Request_Simple`

```

final Yaf_Request_Simple extends Yaf_Request_Abstract {
    public void __construct ( string $module ,
                                string
                                $controller ,
                                string $action ,
                                string $method ,
                                array $params
                                = NULL );
    public mixed  getLang ( void );
    public mixed  getQuery ( string $name = NULL );
    public mixed  getPost ( string $name = NULL );
    public mixed  getEnv ( string $name = NULL );
    public mixed  getServer ( string $name = NULL );
    public mixed  getCookie ( string $name = NULL );
    public mixed  getFiles ( string $name = NULL );
    public bool  isGet ( void );
    public bool  isPost ( void );
    public bool  isHead ( void );
    public bool  isXmlHttpRequest ( void );
    public bool  isPut ( void );
    public bool  isDelete ( void );
    public bool  isOption ( void );
    public bool  isSimple ( void );
    public bool  isDispatched ( void );
    public bool  setDispatched ( void );
}

```



```
public bool isRouted ( void );  
public bool setRouted ( void );  
}
```

属性说明

---

[上一页](#)

[Yaf\\_View\\_Simple::get](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Request\\_Abstract::  
getException](#)

## 名称

**Yaf\_Request\_Abstract::getException**

(Since Yaf 1.0.0.12)

```
public Exception Yaf_Request_Abstract::getException( void );
```

本方法主要用于在异常捕获模式下，在异常发生的情况时流程进入Error控制器的error动作时，获取当前发生的异常对象

## 参数

**void**

该方法不需要参数

## 返回值

在有异常的情况下，返回当前异常对象。没有异常的情况下，返回NULL

## 例子

**例 11.65. Yaf\_Request\_Abstract::getException 的例子**

```
<?php
class ErrorController extends Yaf_Controller_Abstract {
    public function errorAction() {
        $exception = $this->getRequest()->getException();
    }
}
?>
```



名称

Yaf\_Request\_Abstract::getModuleName

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::getModuleName( void );
```

获取当前请求被路由到的模块名.

参数

void

该方法不需要参数

返回值

路由成功以后, 返回当前被分发处理此次请求的模块名. 路由之前, 返回NULL

例子

例 11.66. Yaf\_Request\_Abstract::getModuleName 的例子

```
<?php
class ErrorController extends Yaf_Controller_Abstract {
    public function errorAction() {
        echo "current Module: " . $this->getRequest()->getModuleName();
    }
}
?>
```

---

[上一页](#)[Yaf\\_Request\\_Abstract::getException](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Request\\_Abstract::getControllerName](#)

## 名称

### Yaf\_Request\_Abstract::getControllerName

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::getControllerName( void );
```

获取当前请求被路由到的控制器名。

#### 参数

void

该方法不需要参数

#### 返回值

路由成功以后, 返回当前被分发处理此次请求的控制器名. 路由之前, 返回NULL

#### 例子

例 11.67. Yaf\_Request\_Abstract::getControllerName 的例子

```
<?php
class ErrorController extends Yaf_Controller_Abstract {
    public function errorAction() {
```

```
        echo "current Controller: " . $this->getRequest()->getControllerName();  
    }  
}  
?>
```

---

[上一页](#)

Yaf\_Request\_Abstract::getModuleName

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Request\_Abstract::getActionName

名称

Yaf\_Request\_Abstract::getActionName

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::getActionName( void );
```

获取当前请求被路由到的动作(Action)名.

参数

void

该方法不需要参数

返回值

路由成功以后, 返回当前被分发处理此次请求的动作名. 路由之前, 返回NULL

例子

例 11.68. Yaf\_Request\_Abstract::getActionName 的例子



```
<?php
class ErrorController extends Yaf_Controller_Abstract {
    public function errorAction() {
        echo "current Action: " . $this->getRequest()->getActionName();
    }
}
?>
```

---

[上一页](#)[Yaf\\_Request\\_Abstract::getControllerName](#)[上一级](#)[起始页](#)[下一页](#)[Yaf\\_Request\\_Abstract::getParams](#)

## 名称

### Yaf\_Request\_Abstract::getParams

(Since Yaf 1.0.0.5)

```
public array Yaf_Request_Abstract::getParams( void );
```

获取当前请求中的所有路由参数, 路由参数不是指\$\_GET或者\$\_POST, 而是在路由过程中, 路由协议根据Request Uri分析出的请求参数.

比如, 对于默认的路由协议Yaf\_Route\_Static, 路由如下请求URL: <http://www.domain.com/module/controller/action/name1/value1/name2/value2/> 路由结束后将会得到俩个路由参数, name1和name2, 值分别是value1, value2.



#### 注意

路由参数和\$\_GET,\$\_POST一样, 是来自用户的输入, 不是可信的. 使用前需要做安全过滤.

## 参数

void

本方法不需要参数

## 返回值

当前所有的路由参数

## 例子

### 例 11.69. Yaf\_Request\_Abstract::getParams的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        $this->getRequest()->getParams();
    }
}
```



[上一页](#)

Yaf\_Request\_Abstract::getActionName

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Request\_Abstract::getParam

## 名称

## Yaf\_Request\_Abstract::getParam

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::getParam( string $name ,  
                                              mixed $default_value = NULL );
```

获取当前请求中的路由参数, 路由参数不是指\$\_GET或者\$\_POST, 而是在路由过程中, 路由协议根据Request Uri分析出的请求参数.

比如, 对于默认的路由协议Yaf\_Route\_Static, 路由如下请求URL: <http://www.domain.com/module/controller/action/name1/value1/name2/value2/> 路由结束后将会得到俩个路由参数, name1和name2, 值分别是value1, value2.



## 注意

路由参数和\$\_GET,\$\_POST一样, 是来自用户的输入, 不是可信的. 使用前需要做安全过滤.

## 参数

***\$name***

要获取的路由参数名

***\$default\_value***

如果设定此参数, 如果没有找到\$name路由参数, 则返回此参数值.

返回值

找到返回对应的路由参数值, 如果没有找到, 而又设置了\$default\_value, 则返回default\_value, 否则返回NULL.

例子

例 11.70. Yaf\_Request\_Abstract::getParam 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        echo "user id:" . $this->getRequest()->getParam("userid", 0);
    }
}
?>
```

[上一页](#)

[Yaf\\_Request\\_Abstract::getParams](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Request\\_Abstract::setParam](#)

## 名称

**Yaf\_Request\_Abstract::setParam**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Request_Abstract::setParam( string $name ,  
                                              mixed $value );
```

为当前的请求,设置路由参数.

## 参数

***\$name***

路由参数名

***\$value***

值

## 返回值

成功返回Yaf\_Request\_Abstract实例自身, 失败返回FALSE

## 例子

**例 11.71. Yaf\_Request\_Abstract::setParam的例子**

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function indexAction() {  
        $this->getRequest()->setParam("userid", 0);  
    }  
}  
?>
```

---

[上一页](#)

Yaf\_Request\_Abstract::getParam

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Request\_Abstract::getMethod

## 名称

**Yaf\_Request\_Abstract::getMethod**

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::getMethod( void );
```

获取当前请求的类型, 可能的返回值为GET,POST,HEAD,PUT,CLI等.

## 参数

**void**

本方法不需要参数

## 返回值

当前请求的类型.

## 例子

**例 11.72. Yaf\_Request\_Abstract::getMethod的例子**

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        if ($this->getRequest()->getMethod() == "CLI") {
            echo "running in cli mode";
        }
    }
}
?>
```



[上一页](#)

Yaf\_Request\_Abstract::setParam

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Request\_Abstract::isCli

## 名称

### Yaf\_Request\_Abstract::isCli

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::isCli( void );
```

获取当前请求是否为CLI请求

## 参数

void

本方法不需要参数

## 返回值

是CLI请求返回TRUE, 不是返回FALSE

## 例子

### 例 11.73. Yaf\_Request\_Abstract::isCli的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        if ($this->getRequest()->isCli()) {
            echo "running in Cli mode";
        }
    }
}
?>
```

[上一页](#)

Yaf\_Request\_Abstract::getMethod

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Request\_Abstract::isGet

## 名称

### Yaf\_Request\_Abstract::isGet

(Since Yaf 1.0.0.5)

```
public string Yaf_Request_Abstract::isGet( void );
```

获取当前请求是否为GET请求

## 参数

void

本方法不需要参数

## 返回值

是GET请求返回TRUE, 不是返回FALSE

## 例子

### 例 11.74. Yaf\_Request\_Abstract::isGet的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function indexAction() {
        if ($this->getRequest()->isGet()) {
            echo "running in Get mode";
        }
    }
}
?>
```

[上一页](#)

Yaf\_Request\_Abstract::isCli

[上一级](#)

[起始页](#)

[下一页](#)

11.13. The Yaf\_Response\_Abstract class

## 11.13. The Yaf\_Response\_Abstract class

### 简介

响应对象和请求对象相对应, 是发送给请求端的响应的载体

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 `Yaf \Response_Abstract`

```
abstract Yaf_Response_Abstract {
    protected array _body ;
    protected array _header ;
    public boolean setBody ( string $body ,
                                string $name
    = NULL );
    public boolean prependBody ( string $body ,
                                string
    $name = NULL );
    public boolean appendBody ( string $body ,
                                string $name
    = NULL );
    public boolean clearBody ( void );
    public string getBody ( void );
    public boolean response ( void );
    public boolean setRedirect ( string $url );
    public string __toString ( void );
}
```

属性说明

#### \_body

响应给请求的Body内容

#### \_header

响应给请求的Header, 目前是保留属性

## The Yaf\_Response\_Http class

### 简介

Yaf\_Response\_Http是在Yaf作为Web应用的时候默认响应载体

```
final Yaf_Response_Http extends Yaf_Response_Abstract {  
    protected array _code = 200 ;  
}
```

### 属性说明

#### \_code

响应给请求端的HTTP状态码

## The Yaf\_Response\_Cli class

### 简介

Yaf\_Response\_Cli是在Yaf作为命令行应用的时候默认响应载体

```
final Yaf_Response_Cli extends Yaf_Response_Abstract {  
}
```

---

[上一页](#)[上一级](#)[下一页](#)[Yaf\\_Request\\_Abstract::isGet](#)[起始页](#)[Yaf\\_Response\\_Abstract::setBody](#)

## 名称

**Yaf\_Response\_Abstract::setBody**

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::setBody( string $body ,  
                                              string $name = NULL );
```

设置响应的Body, \$name参数是保留参数, 目前没有特殊效果, 留空即可

## 参数

***\$body***

要响应的字符串, 一般是一段HTML, 或者是一段JSON(返回给Ajax请求)

***\$name***

保留参数, 目前没有特殊效果

## 返回值

成功返回Yaf\_Response\_Abstract, 失败返回FALSE

## 例子

**例 11.75. Yaf\_Response\_Abstract::setBody 的例子**

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {  
        $this->getResponse()->setBody("Hello World");  
    }  
}  
?>
```



参见

[Yaf\\_Response\\_Abstract::appendBody](#)

[Yaf\\_Response\\_Abstract::prependBody](#)

---

[上一页](#)

11.13. The Yaf\_Response\_Abstract class

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Response\\_Abstract::appendBody](#)

## 名称

### Yaf\_Response\_Abstract::appendBody

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::appendBody( string $body ,  
                                                    string $name = NULL );
```

往已有的响应body后附加新的内容, \$name参数是保留参数, 目前没有特殊效果, 留空即可

#### 参数

##### *\$body*

要附加的字符串, 一般是一段HTML, 或者是一段JSON(返回给Ajax请求)

##### *\$name*

保留参数, 目前没有特殊效果

#### 返回值

成功返回Yaf\_Response\_Abstract, 失败返回FALSE

#### 例子

##### 例 11.76. Yaf\_Response\_Abstract::appendBody 的例子

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {  
        $this->getResponse()->appendBody("Hello World");  
    }  
}  
?>
```

参见

[Yaf\\_Response\\_Abstract::setBody](#)

[Yaf\\_Response\\_Abstract::prependBody](#)

---

[上一页](#)

[Yaf\\_Response\\_Abstract::setBody](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Response\\_Abstract::prependBody](#)

## 名称

**Yaf\_Response\_Abstract::prependBody**

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::prependBody( string $body ,  
                                                    string $name = NULL );
```

往已有的响应body前插入新的内容, \$name参数是保留参数, 目前没有特殊效果, 留空即可

## 参数

***\$body***

要插入的字符串, 一般是一段HTML, 或者是一段JSON(返回给Ajax请求)

***\$name***

保留参数, 目前没有特殊效果

## 返回值

成功返回Yaf\_Response\_Abstract, 失败返回FALSE

## 例子

**例 11.77. Yaf\_Response\_Abstract::prependBody 的例子**

```
<?php  
class IndexController extends Yaf_Controller_Abstract {  
    public function init() {  
        $this->getResponse()->prependBody("Hello World");  
    }  
}  
?>
```

参见

[Yaf\\_Response\\_Abstract::setBody](#)

[Yaf\\_Response\\_Abstract::prependBody](#)

---

[上一页](#)

[Yaf\\_Response\\_Abstract::appendBody](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Response\\_Abstract::getBody](#)

## 名称

**Yaf\_Response\_Abstract::getBody**

(Since Yaf 1.0.0.0)

```
public string Yaf_Response_Abstract::getBody( void );
```

获取已经设置的响应body

## 参数

**void**

本方法不需要参数(起码暂时不需要)

## 返回值

成功返回已设置的body值, 失败返回NULL

## 例子

## 例 11.78. Yaf\_Response\_Abstract::getBody 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        echo $this->getResponse()->getBody();
    }
}
?>
```

Yaf\_Response\_Abstract::prependBody

[起始页](#)

Yaf\_Response\_Abstract::clearBody

## 名称

### Yaf\_Response\_Abstract::clearBody

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::clearBody( void );
```

清除已经设置的响应body

## 参数

void

本方法不需要参数(起码暂时不需要)

## 返回值

成功返回Yaf\_Response\_Abstract, 失败返回FALSE

## 例子

例 11.79. Yaf\_Response\_Abstract::clearBody 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->getResponse()->clearBody();
    }
}
?>
```



Yaf\_Response\_Abstract::getBody

[起始页](#)

Yaf\_Response\_Abstract::response

## 名称

**Yaf\_Response\_Abstract::response**

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::response( void );
```

发送响应给请求端

## 参数

**void**

本方法不需要参数(起码暂时不需要)

## 返回值

成功返回TRUE, 失败返回FALSE

## 例子

例 11.80. Yaf\_Response\_Abstract::response 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->getResponse()->response();
    }
}
?>
```

Yaf\_Response\_Abstract::clearBody

[起始页](#)

Yaf\_Response\_Abstract::setRedirect

## 名称

**Yaf\_Response\_Abstract::setRedirect**

(Since Yaf 1.0.0.0)

```
public boolean Yaf_Response_Abstract::setRedirect( string $url );
```

重定向请求到新的路径



注意

和Yaf\_Controller\_Abstract::forward不同, 这个重定向是HTTP 301重定向

## 参数

*\$url*

要重定向到的URL

## 返回值

成功返回Yaf\_Response\_Abstract, 失败返回FALSE

## 例子

例 11.81. Yaf\_Response\_Abstract::setRedirect 的例子

```
<?php
class IndexController extends Yaf_Controller_Abstract {
    public function init() {
        $this->getResponse()->setRedirect("http://domain.com/");
    }
}
?>
```

Yaf\_Response\_Abstract::response

[起始页](#)

Yaf\_Response\_Abstract::\_\_toString

## 名称

### Yaf\_Response\_Abstract::\_\_toString

(Since Yaf 1.0.0.0)

```
public string Yaf_Response_Abstract::__toString( void );
```

魔术方法

参数

void

本方法不需要参数

返回值

Yaf\_Response\_Abstract中的body值

---

## 11.14. The Yaf\_Router class

### 简介

Yaf的路由器, 负责分析请求中的request uri, 得出目标模板, 控制器, 动作.

在PHP5.3之后, 打开`ap.use_namespace`的情况下, 也可以使用 `Yaf\Router`

```
final Yaf_Router {
    protected array _routes ;
    protected string _current_route ;
    public Yaf_Router addRoute ( string $name ,
    Yaf_Route_Interface $route );
    public boolean addConfig ( Yaf_Config_Abstract
    $routes_config );
    public array getRoutes ( void );
    public array getRoute ( string $name );
    public string getCurrentRoute ( void );
    public boolean route ( Yaf_Request_Abstract $request );
    public boolean isModuleName ( string $name );
}
```

### 属性说明

#### \_routes

---

路由器已有的路由协议栈, 默认的栈底总是名为"default"的`Yaf_Route_Static`路由协议的实例.

#### \_current\_route

---

在路由成功后, 路由生效的路由协议名

---

Yaf\_Response\_Abstract::  
\_\_toString

[起始页](#)

Yaf\_Router::addRoute



## 名称

### Yaf\_Router::addRoute

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Router::addRoute( string $name ,
                                     Yaf_Route_Interface $route );
```

给路由器增加一个名为\$name的路由协议

## 参数

### *\$name*

要增加的路由协议的名字

### *\$route*

要增加的路由协议, Yaf\_Route\_Interface的一个实例

## 返回值

成功返回Yaf\_Router, 失败返回FALSE, 并抛出异常(或者触发错误)

## 例子

### 例 11.82. Yaf\_Router::addRoute 的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{
    public function _initRoute(Yaf_Dispatcher $dispatcher) {
        /**
         * 添加一个路由
         */
        $route = new Yaf_Route_Rewrite(
            "/product/list/:id/",
            array(
                "controller" => "product",
                "action"      => "info",
            )
        );
    }
}
```

```
    }  
    $router->addRoute('dummy', $route);  
?>
```

参见

**Yaf\_Router::  
addConfig**  
路由和路由协议

---

[上一页](#)

11.14. The Yaf\_Router class

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Config::addConfig](#)

## 名称

### Yaf\_Config::addConfig

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Config::addConfig( Yaf_Config_Abstract $routes_config );
```

给路由器通过配置增加一簇路由协议

## 参数

### *\$routes\_config*

一个Yaf\_Config\_Abstract的实例, 它包含了一簇路由协议的定义, 一个例子是:

#### 例 11.83. INI 路由协议簇的例子

```
; ini 配置文件
[product]
routes.regex.type="regex"
routes.regex.route="#^list/([^\/]*)/([^\/]*)#"
routes.regex.default.controller=Index
routes.regex.default.action=action
routes.regex.map.1=name
routes.regex.map.2=value

routes.simple.type="simple"
routes.simple.controller=c
routes.simple.module=m
```

```
routes.simple.action=a

routes.supervar.type="supervar"
routes.supervar.varname=r

routes.rewrite.type="rewrite"
routes.rewrite.route="/product/:name/:value"
routes.rewrite.default.controller=product
routes.rewrite.default.action=info
```

返回值

成功返回Yaf\_Config, 失败返回FALSE, 并抛出异常(或者触发错误)

例子

例 11.84. Yaf\_Config::addConfig 的例子

```
<?php
class Bootstrap extends Yaf_Bootstrap_Abstract{
    public function _initRoute(Yaf_Dispatcher $dispatcher) {
        $router = Yaf_Dispatcher::getInstance()->getRouter();
        $router->addConfig(Yaf_Registry::get("config")->routes);
    }
?>
```

参见

**Yaf\_Router::**

**addRoute**

路由和路由协议

[上一页](#)

Yaf\_Router::addRoute

[上一级](#)

[起始页](#)

[下一页](#)

Yaf\_Router::getRoutes

名称

Yaf\_Router::getRoutes

(Since Yaf 1.0.0.5)

```
public array Yaf_Router::getRoutes( void );
```

获取当前路由器中的所有路由协议

参数

void

本方法不需要参数

返回值

成功返回当前路由器的路由协议栈内容, 失败返回FALSE

例子

例 11.85. Yaf\_Router::getRoutes 的例子

```
<?php
    $routes = Yaf_Dispatcher::getInstance()->getRouter()->getRoutes();
?>
```

参见

[Yaf\\_Router::getRoute](#)

[Yaf\\_Router::  
getCurrentRoute](#)

[路由和路由协议](#)

---

[上一页](#)

[Yaf\\_Config::addConfig](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Router::getRoute](#)

名称

Yaf\_Router::getRoute

(Since Yaf 1.0.0.5)

```
public Yaf_Route_Interface Yaf_Router::getRoute( string $name );
```

获取当前路由器的路由协议栈中名为\$name的协议

参数

*\$name*

要获取的协议名

返回值

成功返回目的路由协议, 失败返回NULL

例子

例 11.86. Yaf\_Router::getRoute 的例子



```
<?php
/** 路由器中永远都存在一个名为default的Yaf_Route_Static路由协议实例 */
$routes = Yaf_Dispatcher::getInstance()->getRouter()->getRoute('default');
?>
```

参见

[Yaf\\_Router::getRoutes](#)

[Yaf\\_Router::  
getCurrentRoute](#)

[路由和路由协议](#)

---

[上一页](#)

[Yaf\\_Router::getRoutes](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Router::getCurrentRoute](#)

## 名称

**Yaf\_Router::getCurrentRoute**

(Since Yaf 1.0.0.5)

```
public string Yaf_Router::getCurrentRoute( void );
```

在路由结束以后, 获取路由匹配成功, 路由生效的路由协议名

## 参数

**void**

本方法不需要参数

## 返回值

成功返回生效的路由协议名, 失败返回NULL

## 例子

例 **11.87. Yaf\_Router::getCurrentRoute** 的例子

```
<?php
class UserPlugin extends Yaf_Plugin_Abstract {

    public function routerShutdown(Yaf_Request_Abstract
    $request, Yaf_Response_Abstract $response) {
```

```
        echo "生效的路由协议是:" . Yaf_Dispatcher::getInstance()->getRouter  
() ->getCurrentRoute();  
    }  
}  
?>
```

参见

[Yaf\\_Router::getRoute](#)

[Yaf\\_Router::](#)

[getRoutes](#)

[路由和路由协议](#)

---

[上一页](#)

[Yaf\\_Router::getRoute](#)

[上一级](#)

[起始页](#)

[下一页](#)

[Yaf\\_Router::isModuleName](#)

## 名称

**Yaf\_Router::isModuleName**

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Router::isModuleName( string $name );
```

判断一个Module名, 是否是申明存在的Module

**注意**

通过`ap.modules`在配置文件中申明加载的模块名列表

## 参数

***\$name***

Module名

## 返回值

如果是返回TRUE, 不是返回FALSE

## 例子

例 11.88. Yaf\_Router::isModuleName 的例子

```
<?php
$routes = Yaf_Dispatcher::getInstance()->isModuleName("Index")
?>
```

Yaf\_Router::getCurrentRoute

[起始页](#)

Yaf\_Router::route

## 名称

### Yaf\_Router::route

(Since Yaf 1.0.0.5)

```
public boolean Yaf_Router::route( Yaf_Request_Abstract  
$request );
```

路由一个请求, 本方法不需要主动调用, Yaf\_Dispatcher::dispatch会自动调用本方法

#### 参数

*\$request*

一个Yaf\_Request\_Abstract实例

#### 返回值

成功返回TRUE, 失败返回FALSE

#### 例子

例 11.89. Yaf\_Router::route 的例子

```
<?php  
?>
```

参见

[路由和路由协  
议](#)

---

[上一页](#)

Yaf\_Router::isModuleName

[上一级](#)

[起始页](#)

[下一页](#)

11.15. The Yaf\_Route class

---

## 11.15. The Yaf\_Route class

### 简介

Yaf\_Route\_Interface是Yaf路由协议的标准接口, 它的存在使得用户可以自定义路由协议

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 `Yaf\Route_Interface`

```
Interface Yaf_Route {  
    abstract public boolean route ( Yaf_Request_Abstract  
$request );  
}
```

参见

`Yaf_Route_Static`

`Yaf_Route_Simple`

`Yaf_Route_Supervar`

`Yaf_Route_Rewrite`

`Yaf_Route_Regex`



## 11.16. The Yaf\_Exception class

### 简介

Yaf\_Exception是Yaf使用的异常类型, 它继承自Exception, 并实现了异常链.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 `Yaf\Exception`



#### 注意

只有在[ap.throw\\_exception](#)([php.ini](#))或者[ap.throwException](#)(配置文件)开启的情况下, Yaf才会抛出异常, 否则Yaf在出错的时候将[trigger\\_error](#), 这种情况下, 可以使用[Yaf\\_Dispatcher::setErrorHandler](#)来捕获错误.

```
Yaf_Exception {
    protected string message ;
    protected string code ;
    private Exception _previous ;
    public void __construct ( string $message ,
                                int $code = 0 ,
                                Exception
                                $previous = NULL );
    final public string Exception::getMessage ( void );
    final public int Exception::getCode ( void );
    public final Exception getPrevious ( void );
    final public string Exception::getFile ( void );
    final public int Exception::getLine ( void );
}
```

### 属性说明

**message**

### 异常信息

## code

异常代码

## \_previous

此异常之前的异常

## Yaf\_Exception\_StartupError

### 简介

继承自Yaf\_Exception, 在Yaf启动失败的时候抛出.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\nStartupError

```

Yaf_Exception_StartupError extends Yaf_Exception {
    protected string code = YAF_ERR_STARTUP_FAILED ;
}

```

## Yaf\_Exception\_RouterFailed

### 简介

继承自Yaf\_Exception, 在路由失败的时候抛出.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\nRouterFailed

```

Yaf_Exception_RouterFailed extends Yaf_Exception {
    protected string code = YAF_ERR_ROUTER_FAILED ;
}

```

## Yaf\_Exception\_DispatchFailed

### 简介

继承自Yaf\_Exception, 在分发失败的时候抛出.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception

## \DispatchFailed

```
Yaf_Exception_DispatchFailed extends Yaf_Exception {
    protected string code = YAF_ERR_DISPATCH_FAILED ;
}
```

## Yaf\_Exception\_LoadFailed

### 简介

继承自Yaf\_Exception, 在加载需要类失败的时候抛出.

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\nLoadFailed

```
Yaf_Exception_LoadFailed extends Yaf_Exception {
    protected string code = YAF_ERR_AUTOLOAD_FAILED ;
}
```

## Yaf\_Exception\_LoadFailed\_Module

### 简介

继承自Yaf\_Exception\_LoadFailed, 在找不到路由指定的模块时抛出

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\nLoadFailed\Module

```
Yaf_Exception_LoadFailed_Module extends Yaf_Exception_LoadFailed
{
    protected string code = YAF_ERR_NOTFOUND_MODULE ;
}
```

## Yaf\_Exception\_LoadFailed\_Controller

### 简介

继承自Yaf\_Exception\_LoadFailed, 在找不到路由指定的控制器时抛出

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\nLoadFailed\Controller

```

Yaf_Exception_LoadFailed_Controller extends
Yaf_Exception_LoadFailed {
    protected string code = YAF_ERR_NOTFOUND_CONTROLLER ;
}

```

## Yaf\_Exception\_LoadFailed\_Action

### 简介

继承自Yaf\_Exception\_LoadFailed, 在找不到路由指定的动作时抛出

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\_LoadFailed\Action

```

Yaf_Exception_LoadFailed_Action extends Yaf_Exception_LoadFailed
{
    protected string code = YAF_ERR_NOTFOUND_ACTION ;
}

```

## Yaf\_Exception\_LoadFailed\_View

### 简介

继承自Yaf\_Exception\_LoadFailed, 在找不到指定的视图模板文件时抛出

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\_LoadFailed\View

```

Yaf_Exception_LoadFailed_View extends Yaf_Exception_LoadFailed {
    protected string code = YAF_ERR_NOTFOUND_VIEW ;
}

```

## Yaf\_Exception\_TypeError

### 简介

继承自Yaf\_Exception, 在关键逻辑参数出错的时候抛出

在PHP5.3之后, 打开[ap.use\\_namespace](#)的情况下, 也可以使用 Yaf\Exception\_TypeError

```
Yaf_Exception_TypeError extends Yaf_Exception {  
    protected string code = YAF_ERR_TYPE_ERROR ;  
}
```

---

[上一页](#)

[上一级](#)

11.15. The Yaf\_Route class

[起始页](#)