# 0 yii2-WebApplication

Created by 杨超, last modified on 2018 Jun 08


- 废话三两句 – 关于学习
- 0 历史起源
- 1 Web Application概述
- 2 yii2 Web进程入口
- 3 Application阅读
  - 3.1 Application继承关系
  - 3.2 不一样的阅读
    - 3.2.1 web/application
    - 3.2.2 小结
    - 3.2.3 执行结束之后
- END


> 对于yii2来说，运行的入口就是WebApplication，所以我从这里入手进行阅读解析，后续再讲ConsoleApplicatioin的

## ~~废话三两句 – 关于学习~~

关于学习和阅读，我一直觉得本质上是理解和思考的过程，所以开始整理之前的阅读和文档，并且重新阅读yii2源码的时候，我有了下面的思考

类比 哲学三大问 是什么？从哪里来？到哪里去?

我也希望坚持写完之后可以回答 关于 技术Yii2的三大问题

yii2 总结起来是什么？为什么有这个 & 解决了什么问题？未来可以怎么应用 或者说 我们应该如何改进来满足我们日常和高阶的需求 或者说 如何更加优雅?

## 0 历史起源

Yii的很多想法来自其他著名Web编程框架和应用程序, 下面是一个简短的清单

```
Prado：这是 Yii 的主要思想来源。Yii 采用了基于组件和事件驱动编程模式，数据库抽象层，模块化的应用架构，国际化和本地化，和许多它的其他特点和功能。
   ---- 这个框架是04年 xueqiang创建的一个框架，扫了一眼有些yii2的雏形但是还是相对原始 目前start数只有142 link: http://www.pradoframewor
Ruby on Rails：Yii 继承它的配置的思想。还引用它的 Active Record 的 ORM 设计模式。
   ---- 本人之前写过Ruby On Rails，这门语言相对激进，倾向于用近似人的写作语言来写程序，对于看懂来说很优雅但是对于理解阅读原理实现很难，但里面的
jQuery：这是集成在 Yii 为基础的 JavaScript 框架。
Symfony：Yii 引用它的过滤设计和插件架构。
```

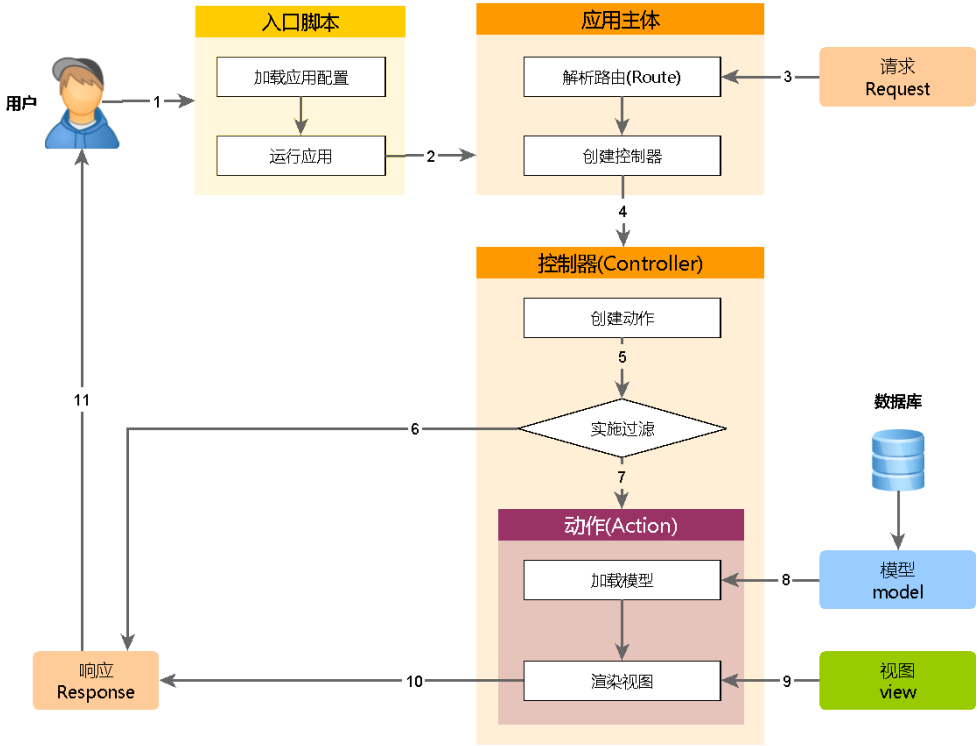
Joomla: Yii 引用其模块化设计和信息翻译方案。

总结: 提取关键字，所以对于yii2来说 起码它认为有价值的东西是 ActvieRecord / 组件 / Event / filter / config(配置化) / 模块化

# 1 Web Application概述

yii2请求过程示意图

这张图从yii2官网抠下来的, 大家感性认识一下就行 我个人觉得基本没用




1 用户提交指向 入口脚本 web/index.php 的请求
2 入口脚本会加载 配置数组 并创建一个 应用 实例用于处理该请求
3 应用会通过 request（请求） 应用组件 解析被请求的 路由
4 应用创建一个 controller（控制器） 实例具体处理请求
5 控制器会创建一个 action（动作） 实例并为该动作执行相关的 Filters（访问过滤器）
6 如果任何一个过滤器验证失败，该动作会被取消
7 如果全部的过滤器都通过，该动作就会被执行
8 动作会加载一个数据模型，一般是从数据库中加载

9 动作会渲染一个 View（视图），并为其提供所需的数据模型
10 渲染得到的结果会返回给 response（响应） 应用组件
11 响应组件会把渲染结果发回给用户的浏览器

# 2 yii2 Web进程入口

入口文件： web/index.php

不要问我是怎么知道的，不信你可以看nginx配置 或者 改成别的文件试试?

看这句就行了, 反正加载了一大堆的common和应用自定义的配置文件之后，就new Application 然后运行了

```
$application = new yii\web\Application($config);
```

# 3 Application阅读

## 3.1 Application继承关系

web/Appliaction --> base/Application --> Module --> ServiceLocator --> component --> BaseObject

## 3.2 不一样的阅读

建议 阅读的时候对照源码一步步进去看，我这次换个思路 按照继承关系 倒叙的讲代码实现

前提是需要你至少看过一遍代码 或者 跟着list一步步的去看过这几个类

| level | 类名 | 作用 |
| --- | --- | --- |
| 0 | BaseObject | get/set/property – 没错这里实现了最基础的属性，支持get/set方法来实现属性 --> 不信你去看呀 |

| level | 类名 | 作用 |
|---|---|---|
| 1 | Component | 1 event/behaviors – 这一层开始支持event/behaviors，也就是说只有是一个组件 才能拥有这些特性<br>2 get&set 对上一层的get/set的增强, 就是增加了behavior/events handler支持<br>上一次的BaseObject只支持比较朴素的函数，这里支持了yii2特有的event/behavior里面的方法<br>总结:<br>微观上: 某种意义上 Component就是对BaseObject的扩展，增加事件和行为的支持<br>宏观上: 你可以认为这一层的组件就是 yii2里的特有的object，我们日常的开发基于OOP的思想，新开发的class应该至少基于Component才对 |
| 2 | ServiceLocator | 0 seviceLocator 本身是一种“反模式”，在web开发里面行之有效 类似操作系统的内存快表 或者 windows里的注册表<br>（都不懂？那就理解成 serviceMap可以根据id快速的找到你要的service/component)<br>通俗解释，后续我会专门写一章讲这个<br>这一层实际上就是将componentId 为index放到locator里面，所以在controller里面就可以非常简单的写出 Yii::$app->db/log/audit 这种代码来轻松的调起对应的模块<br>实现的方法也很简单，就是有两个数组<br>`$_definitions` definitions记下id对应的定义，是string的一个class名 还是 一个数组<br>`* $object = Yii::createObject('yii\db\Connection');`<br>`* $object = Yii::createObject([`<br>`* 'class' => 'yii\db\Connection',`<br>`* 'dsn' => 'mysql:host=127.0.0.1;dbname=demo',`<br>`* 'username' => 'root',`<br>`* 'password' => '',`<br>`* 'charset' => 'utf8',`<br>`* ]);`<br>`*`<br>`$_components` components记下id对应的定义 生成后的instance实例， 也就是Yii::createObject之后生成的实例 |
| 3 | Module | 0 这个class是Module/Application的基类，定义了很多controller/module的通用方法 – 比如最终的runAction的执行就在这里<br>重要函数list, 看名字就可以猜出意思: beforeAction / afterAction / runAction / createController<br>1 关于createController 有一篇解析文章写的不错 http://www.jb51.net/article/89138.htm<br>简单来说, 就是解析route之后找到对应的controller 然后new出来对应的instance |

| level | 类名 | 作用 |
| --- | --- | --- |
| 4 | base/Application | *0 看到这里的已经接近山顶了 尼采曾经说过： 智者之所以喜欢 志同道合的人，是因为不志同道合走不到同一高度也碰不到啊?*<br>1 bootstrap 这个方法就是预先加载一些必须预加载的module 比如 coreBootstrap 不管你是否显式的使用，比如log/debug等<br>2 run 方法<br>这里基本上就是个壳 主要函数handleRequest在具体的 Web/Console Application里实现<br>然后在before/handling/after的时候更改state并且trigger对应的事件 |
| 5 | web/application | 这个太长，放下面单开 |

### 3.2.1 web/application

提一句 和console/application一样都是对base的具体实现，地位等价

`base/application: $response = $this->handleRequest($this->getRequest());`

```
public function handleRequest($request)
{
    // 第一部分 对路由 和 参数的解析
    if (empty($this->catchAll)) {
        try {
            list($route, $params) = $request->resolve();
        } catch (UrlNormalizerRedirectException $e) {
            ...
        }
    }

    try {
        Yii::trace("Route requested: '$route'", __METHOD__);
        $this->requestedRoute = $route;
        // 第二部分 具体的函数调用执行
        $result = $this->runAction($route, $params);
        if ($result instanceof Response) {
            return $result;
        }
```

```
            // 第三部分 response类对result结果再加工，最终return回去
            $response = $this->getResponse();
            if ($result !== null) {
                $response->data = $result;
            }

            return $response;
        } catch (InvalidRouteException $e) {
            throw new NotFoundHttpException(Yii::t('yii', 'Page not found.'), $e->getCode(), $e);
        }
    }
```

**第一部分 对路由 和 参数的解析**
1.1 catchAll 如果没有设置, 则执行下面的正常解析 / 如果有 则直接使用
　@关于catchAll 是routeMap的一种延伸，可以用来临时将网站的部分或者全部请求重定向到一个url上，比如出现紧急维护的时候可能很有用
1.2 request 最终调用 UrlManager 的 parseRequest route+params
　@UrlManger里的解析依照urlRule的规则 会使用normalizer来进行一些特殊符号（比如'/')的匹配

**第二部分 具体的函数调用执行**

这一步其实就是直接用 Module里的runAction
**2.1 createController**

说白了，就是定位到controller的文件位置，找出对应的位置 以及 actionID

```
$oldController = Yii::$app->controller;
Yii::$app->controller = $controller;
$result = $controller->runAction($actionID, $params);
if ($oldController !== null) {
Yii::$app->controller = $oldController;
}
```

**2.2 createAction**
找到controller里的method，ReflectionMethod
runWithParams
**2.2.1 要注意一般来说，绝大部分情况都是 执行的是InlineAction里的runWithParams，直接执行对应的函数**

```
public function runWithParams($params)
{
$args = $this->controller->bindActionParams($this, $params);
Yii::trace('Running action: ' . get_class($this->controller) . '::' . $this->actionMethod . '()', __METHOD__);
if (Yii::$app->requestedParams === null) {
Yii::$app->requestedParams = $args;
}
return call_user_func_array([$this->controller, $this->actionMethod], $args);
}
}
```

2.2.2 还有一种情况，就是直接Action 也就是actions里的函数，比如 SearchAction 这时候就回去找到 [[ run ]] 函数去执行 就好了

这里就很简单了，构造完prams里的参数值知会，就直接执行了

```
if ($this->beforeRun()) {
$result = call_user_func_array([$this, 'run'], $args);
$this->afterRun();
return $result;
}
```

### 3.2.2 小结

Anyway 走到这里，controller::action函数被找到然后执行，返回result

其实对后端来说某种意义上这个时候就已经“执行完毕”了，也就是说结果数组 或者 html就已经被render出来了，如果log在这里可以看到近似最终的结果了

### 3.2.3 执行结束之后

简单收个尾，实际上最终还要返回reponse
主要就是这么一句话: Response->data = $result
Response默认是： 'response' => ['class' => 'yii\web\Response'],
这里面细节很多，比较碎小，但主要的核心是 formatter 格式化返回的内容 json/xml ? 本地化 或者 可以自定义一些formatter
有兴趣自看

# END

基本上讲完了Application的运行概述，并且借机把用到的模块按照继承关系大致梳理了一下，其实每个模块 甚至 每个过程（比如route的解析 / event handler的绑定）都可以写个专题来讲的，后续希望能发动更多的同志们一起补齐，个人也会坚持

Like Be the first to like this

No labels