欢迎访问非官方 Kohana 3.0 维基百科

表示外部链接

英文原版地址：http://www.kerkness.ca/wiki/doku.php

Welcome to the Unofficial Kohana 3.0 Wiki

中文翻译：Yuzek

5/28/2010 9:31:25 PM

QQ:78508136

Email:yuzek@qq.com

前言

非常感谢你从百忙中下载和阅读本手册。Kohana3 是一款非常精美的框架，说她精美是因为她的结构非常的简单，就像是一个细胞一样，但正是由于她像细胞，所以她可以自由组合，功能强大。给你无限的可能，她完全面向对象，松耦合，扩展性强，是我看到过的最干净的框架。就像我前文所说她就像是一个细胞，然后分裂出一个细胞，一个一个，想怎么组合就怎么组合，只要符合她的规则，而且 kohana 本身也是一个细胞。这可能就是 HMVC 的魅力吧。

由于 Kohana3 的中文资料不是很全面，官方手册上也缺少例子，使很多人学习困难，所以我翻译了这个维基，使更多英语不好的朋友能很快上手，希望对你有帮助。

翻译这个维基百科大概花费了我 2 周的时间，由于本人英语水平有限，所以很多地方可能有错误或语病，希望你能指出或者修复。另外由于英文维基不停的在更新，所以我不能保证我所提供的版本是最新版本，你可以直接访问英文官方网站 http://www.kerkness.ca/wiki/doku.php 来获得最新信息。

Yuzek

2010 年 5 月 28 日

Kohana 是一个 PHP5 的框架。它采用了模型、视图、控制器组成的结构模式。它的目标是为了安全，轻量级和容易使用。

- Kohana 用户手册
- API 手册
- Kohana 论坛
- GitHub 仓库
- Redmine 项目页面
- 官方邮件列表：捐助可发邮件至 ***kohana@librelist.com***

我该选择什么版本的 Kohana？

- Kohana 2.x 和 3.x 的区别以及你该如何选择正确的版本

安装和配置 Kohana 3.0

- 从 GitHub 安装
- 设置生产环境
- 去除 index.php

如何使用控制器和视图

- 如何创建和使用控制器
- 如何创建和使用视图
- 如何绑定和设置数据到视图
- 在视图中设置和使用全局数据

制作一个模板驱动的网站

- 创建一个模板
- 继承模板控制器
- 基本页面控制器

路由和请求操作

- Kohana 中的 HMVC：请求工厂
- 路由基础
- 忽略溢出的路由
- 建立一个路由和控制器来处理国际化的静态页面
- 多语言路由
- 建立子目录路由
- 创建一个自定义的 404 页面
- 内部和外部的不同请求
- 如何重定向用户请求
- 如何测试路由
- 反向路由和分页

如何使用数据库模块

- 如何打开和配置数据库模块
- 用查询生成器产生 CRUD
- 用查询生成器来高级查询
- 构建复杂的 SELECT 语句
- 用查询生成器分页
- 如何关闭一个数据库连接

Cookies 和 Sessions

- Cookies 和 Sessions 的使用

如何使用 HTML 类

Kohana_HTML 类是一个 HTML 助手。它用来帮助创建 HTML 元素

- 如何转换一个特殊字符为 HTML 实体
- 如何创建一个文本或图像链接
- 如何在一个新窗口打开链接
- 如何生成一个很难被其他人检测到的 email 地址
- 如何创建一个 email 链接
- 如何创建一个 css 链接
- 如何创建一个脚本链接
- 如何显示一张图片

- 如何设置属性

如何使用文本类

Kohana_Text 类是一个文本助手。它用来帮助处理字符串。

- 如何限制一个字符串的单词数
- 如何限制一个字符串的字符数
- 如何轮换两个或更多字符串
- 如何产生一个随机字符串
- 如何把字符串中多个反斜杠变成单个的反斜杠
- 如何从一个字符串中过滤掉特定的词语
- 如何查找相似词语
- 如何自动把 URLs 转成链接
- 如何自动为文本块增加段落标签
- 如何把字节格式转换成人们常用的格式

如何使用 URL 类

Kohana_Url 类是一个 Url 助手。它用来帮助处理 Urls

- 如何获得你网站的基本 Url
- 如何生成网站 Urls
- 如何生成查询字符串
- 如何生成友好的 Url 标题字符串

如何使用表单类

Kohana_Form 类是一个表单助手。它用来帮助处理 HTML 表单元素

- 如何打开和关闭一个表单
- 如何创建一个 input 域
- 如何创建一个隐藏域
- 如何创建一个密码域
- 如何创建一个文件上传域
- 如何创建一个多选框
- 如何创建一个单选框
- 如何创建一个文本区域
- 如何创建一个选择域和下拉菜单
- 如何创建一个表单按钮
- 如何创建表单标签
- 表单验证

如何使用日期类

Kohana_Date 类是一个表单助手。它用来帮助处理 HTML 表单元素

- 确定时区间的偏移（秒）
- 获得一天，小时，分钟所包含的秒，分，小时
- 获得所给时间处于上午还是下午
- 转换一个非 24 小时数字为 24 小时数字
- 获得一个月有几天
- 获得一年有几个月
- 获得起始年到终止年中的年份，并转换为数组
- 获得两个时间戳之间的时差
- 获得所给出时间和现在的差异
- 转换 UNIX 和 DOS 的时间戳

其他 Kohana 类

- 在 Kohana 中使用数组（数组类）
- 在 Kohana 中进行远程调用
- 使用 Atom 和 RSS Feeds
- 使用文件
- 使用数字
- 使用偏转器

国际化

- 设置和文件结构
- 如何设置默认语言
- 设置和检索语言字符串
- 翻译消息
- 多语言网站实例

使用 ORM 对象建模

- ORM 实例
- ORM 概述 （经由 jheathco 维基百科仓库）
- Github.com/Kohana 上的 ORM 教程

使用 Sprig 对象建模

- Sprig 概述 （经由 shadowhand 仓库）
- 获得对象列表（find_all）
- 用 Sprig 用户模型来认证
- 用 Sprig 用户模型使用 sprig-auth 来认证
- 验证一个 Sprig 模型
- 通过 AJAX 来验证一个 Sprig 模型

使用 Jelly 对象建模

- 主要的 Jelly 文章 http://jelly.jonathan-geiger.com/
- Jelly 认证 http://github.com/raeldc/jelly-auth

如何使用 Auth 模块

- 继承 Model_Auth_User 类
- 在控制器中使用身份认证模块

提示和技巧

- 如何更好的在 Kohana 控制器中使用图片
- 如何使用分页模块
- 用 Capistrano 部署 Kohana 应用
- 如何使用 Hudson 安装持续集成
- Kohana 的命令行 CLI 用法[完成]
- 整合 Xajax

Kohana 3.0 速查表

- HTML
- PNG
- PDF

更多文档和教程

- KO3 教程第一部分，安装和基本用法
- KO3 教程第二部分，使用视图
- KO3 教程第三部分，使用控制器类
- KO3 教程第四部分，使用模型
- KO3 教程第五部分，HMVC 中的 H
- KO3 教程第六部分，路由和路由选择
- KO3 教程第七部分，助手
- KO3 教程第八部分，模块

Kohana 3.x vs 2.x

[返回目录]

Kohana PHP 框架当前有 2 个都在开发的版本。

这 2 个版本被称为 Kohana 2.x 和 Kohana 3.x 。从 2.x 升级到 3.x 没什么意义，API 变化很大。

因此，它们不是继承关系的版本，它们可被认为是两个不同的框架。

Kohana 2.x 和 3.x 有什么共同点？

- 它们都产生于 Kohana PHP 社区
- 它们获得相同的论坛支持
- 它们都高安全性，松耦合性并且容易扩展
- 它们都是绝对的 PHP 5 面向对象框架
- 它们 100% 的兼容 UTF-8
- 它们有同样的数据库抽象功能
- 两个框架都能达到它们最初的设计要求
- 两个框架都处于积极的开发状态并且它们会在未来的一段时间内完全被支持

Kohana 2.x 和 3.x 的主要不同点是？

- 两个框架有他们自己的开发团队和目标
- Kohana 2.x 使用 MVC 设计模式，而 Kohana 3.x 使用 HMVC 设计模式
- Kohana 2.x 是建立在以前的老 2.x 版本的基础上的 ,而 Kohana 3.x 是一个没有任何历史遗留问题的全新框架结构
- 老的 2.x 版本的文档将仍然适用于最新的 Kohana 2.x 版本，而 Kohana 3.x 是建立在新的文档基础上，以协助人们在 3.x 框架上工作

我该使用哪个版本的 Kohana？

通常来说，很大程度上你应该以自己如何开发作为选择什么版本 Kohana 的根本。诸如你如何使用框架以及你曾经使用 Kohana 和 PHP 的经验。

从哪个框架运行更快，更安全或者更稳定的立场来看，它们并没有真正的区别。两个框架使用最佳的面向对象编程来开发，都是同样有能力支持大型可扩展解决方案的最佳做法。

这里有一些问题你不妨问一下自己，以帮助你来确定哪个是对你来说最好的框架。

你是否有一个现存的 Kohana 应用程序?

如果你准备开始一个全新的项目，那么你应该考虑使用 Kohana 3.x 框架。

如果你有一个使用 Kohana 2.x 代码库的应用程序。你可能要考虑使用 Kohana 2.x，除非你想重写绝大多数代码。 Kohana 3.x 很少用到 2.x 的代码库，并使用了许多新的约定。从 2.x 的应用程序“升级”到 3.x 根本不可能。如果你要移动一个 2.x 应用程序到 Kohana 3.x 上，那么这将是你重新设计你的应用程序的机会。

你是否想在学习一个框架的时候使用完整的应用程序实例?

因为当前的 Kohana 2.x 版本是一个兼容的在 2.x 代码库基础上的版本。一些网上已经存在的 Kohana 文档对它仍然适用。如果你是一个 PHP 和框架方面的初学者，你可以找到更多的 Kohana 2.x 版本的教程来帮你一步步制作一个应用程序。

每天都会有更多的关于 Kohana 3.x 的文档、代码和可用的教程产生。它们包括本维基百科，以及官方的用户手册，API 浏览器和论坛支持。有些开发者将从中发现比学习如何使用框架和开始制作解决方案更多的信息

如果你已经认真看完了这个维基百科并查看了官方用户手册但仍然不知道该如何使用 Kohana 3.x ，那么你最好考虑使用 2.x 版本的框架。

你更喜欢工作在 HMVC 还是 MVC 设计模式上？

如果你想在你的应用程序中使用 HMVC 设计模式 ,那么你可以选择 Kohana 3.x 来帮助你开发。Kohana 3.x 出色的支持了 HMVC 的灵活性。
HMVC 的初始读本可见于：: Kohana 中的 HMVC

如果你更想专用 MVC 设计模式。那么你将发现 Kohana 2.x 和 3.x 都能满足你碰到的需求。它们都有相似的方法来支持这个设计模式。

从 GitHub 安装 Kohana 3.0

[返回目录]

Kohana 3.0 的源代码位于GitHub。要使用 github 安装 Kohana 首先你必须要安装 git。
访问 http://help.github.com 来获悉如何在你的平台上安装 git 的详细信息并按照这些步骤做。

```
git clone git://github.com/kohana/kohana.git
cd kohana/
git submodule init
git submodule update
```

添加子模块

完成下列步骤来添加一个新的子模块：

1 运行下列代码 – git 为每个新子模块添加仓库路径 例如：

```
git submodule add git://github.com/shadowhand/sprig.git modules/sprig
```

2 然后初始化并更新子模块

```
git submodule init
git submodule update
```

去除子模块

去掉一个不再需要的子模块只需要完成下面几步

1 打开 .gitmodules 并且去掉子模块的引用，它看起来像这样

```
[submodule "modules/auth"]
    path = modules/auth
    url = git://github.com/kohana/auth.git
```

2 打开 .git/config 并且去掉子模块的引用

```
[submodule "modules/auth"]
    url = git://github.com/kohana/auth.git
```

3 运行 git rm –cached 路径/到/子模块 例如：

```
git rm --cached modules/auth
```

注意：不要再路径最后放反斜杠，如果你在命令的最后放了反斜杠，它会失败。

设置生产环境

[返回目录]

在将你的应用程序移入生产之前，这里有一些你要对你的应用程序需要做的事。

1．查看文档中的配置页。这个包括了不同环境中需要改变的大部分全局设置。通常来所，你最好在生产环境的网站上启用缓存并且关闭详细信息（Kohana::init 设置）。如果你有一些路由规则，缓存路由同样是有帮助的。

2. 在 application/bootstrap.php 设置捕捉所有异常使敏感数据不会被堆栈追踪泄露出去。从 Shadowhand 的wingsc.com source 查看示例。

3. 打开 APC 或者某些操作码缓存工具。这个能让 PHP 自身轻松的增强性能。更复杂的系统能从操作码缓存中获得更多的性能。

```
/**
 * 设置环境字符串的域名 (默认 'development').
 */
if ($_SERVER['SERVER_NAME'] !== 'localhost') Kohana::$environment = 'production';
```

```
/**
 * 初始化 Kohana 基本环境
 */
$settings = array(
  'base_url' => '/',
  'index_file' => FALSE
);
switch (Kohana::$environment) {
  case 'production':
    $settings += array(
      'profiling' => TRUE,
      'caching' => FALSE
    );
    break;
  default:
    $settings += array(
      'profiling' => FALSE,
      'caching' => TRUE
    );
    break;
}
Kohana::init($settings);
```

```
/**
    * 使用 PATH_INFO 执行主要的请求.如果没有 URI 被指定，URI 将被自动检测
    */
   $request = Request::instance($_SERVER['PATH_INFO']);

   try
   {
        // 尝试执行响应
        $request->execute();
   }
   catch (Exception $e)
   {
        if ( Kohana::$environment == 'development' )
        {
             // 只是重新抛出异常
             throw $e;
        }

        // 记录错误
        Kohana::$log->add(Kohana::ERROR, Kohana::exception_text($e));

        // 创建一个 404 的响应
        $request->status = 404;
        $request->response = View::factory('template')
             ->set('title', '404')
             ->set('content', View::factory('errors/404'));
   }

   if ($request->send_headers()->response)
   {
        // 获得总共用的内存和执行时间
        $total = array(
             '{memory_usage}' => number_format((memory_get_peak_usage() -
KOHANA_START_MEMORY) / 1024, 2).'KB',
             '{execution_time}' => number_format(microtime(TRUE) - KOHANA_START_TIME, 5).'
seconds');

        // 将统计插入到响应中
        $request->response = str_replace(array_keys($total), $total, $request->response);
   }
```

```
/**
 * 显示请求的响应
 */
echo $request->response;
```

去除 index.php

[返回目录]

这里有两个步骤能让 index.php 从 URL 中被移除

首先编辑 bootstrap 文件。查找 bootstrap 文件中的 Kohana::init 这一行 ,并且在下面增加 index_file 参数：

```
Kohana::init(array(
        'base_url'      => '/blog', // You need to edit this also to your needs!
        'index_file' => '',
));
```

你可能认为这样足够了，但是你还需要在 .htaccess 文件中改变一行。

重命名 example.htaccess 文件为 .htaccess 并且更改下面这行代码：

```
RewriteBase /kohana/
```

修改为和 bootstrap 文件中的 base_url 一样。如果你的 Kohana 安装在如 example.com/blog/ 的位置，那么就修改为：

```
RewriteBase /blog/
```

检修#1

如果无法工作（出现 “interval Server Error” 或者 “No input File Specified” ）请尝试修改：

```
RewriteRule ^(?:application|modules|system)\b - [F,L]
```

到

```
RewriteRule ^(application|modules|system)/ - [F,L]
```

还有修改

```
RewriteRule .* index.php/$0 [PT]
```

到

```
RewriteRule .* index.php [L]
```

（从http://cssmysite.com/post/18 可找到详情）

检修#2

如果仍然无法正常使用，请确定 .htaccess 文件是否在 httpd.conf 中被允许启用（然后重启 Apache）

```
        <Directory "/var/www/html/example.com/blog">
        Order allow,deny
        Allow from all
        AllowOverride All
        </Directory>
```

（从http://forum.kohanaphp.com/comments.php?DiscussionID=3445&page=1#Item_0 可找到详情）

如何创建和使用控制器

[返回目录]

控制器是一个应用程序中位于模型和视图之间的一个类文件。当数据需要被改变时，控制器将信息传递给模型。当数据需要加载时，控制器从模型请求得到信息。控制器将信息从模型传递到视图，最终显示给客户。

控制器的约定

- 必须位于控制器（或子控制器）目录
- 控制器文件名必须小写，如：articles.php
- 控制器中的类必须和文件名关联，首字母大写并用 Controller_ 开头。如：Controller_Articles
- 必须继承 Controller 父类
- 控制器中需要被外部访问的方法必须定义为 public 并且用 action_ 开头（如：public action_index()），如果不是这样定义，那么就不能通过路由请求到该方法。
- 要想从控制器输出信息，给 $this->request->response 赋值就可

请求一个控制器

默认情况下从 url 请求一个控制器可以在 url 中附加上控制器名和动作名，如下：

```
http://example.com/index.php/<控制器>/<动作>
```

例如。下面请求了 basic 控制器的 index 动作

```
http://example.com/index.php/basic/index
```

创建一个控制器

应用程序中的控制器一般放在 application/classes/controller 文件夹中。如果是作为模块那么则放在 module/模块名/classes/controller 文件夹下。

Basic 控制器

```
application/classes/controller/basic.php
```

```
class Controller_Basic extends Controller
{
    public function action_index()
    {
        $this->request->response = "Hello World";
    }
}
```

现在你如果访问 你的域名.com/basic/index （或者用 URL 重写也就 rewrite 功能访问 你的域名.com/index.php/basic/index）你就会看到

```
Hello World
```

路由一个 URI 到控制器

Kohana 3.0 的 Route 类 支持你无限制的设置特殊格式的请求到指定的控制器和动作。

可以在你的 application/bootstrap.php 文件中定义路由，或者在 module/你的模块/init.php 文件中使用 Route::set() 方法

查看本维基百科的路由基础章节开始学习

如何创建和使用视图

[返回目录]

视图是那些包含显示你应用程序信息的文件。他们通常是 HTML， CSS 和 JavaScript，但是同样能做任何你需要的事，诸如 XML 或者 Json 的 AJAX 输出。视图的目的是让你的应用程序中的逻辑和显示部分分离，并使你的逻辑部分提高可重用性以及使代码看起来更简洁。

没错，视图通常是那些你希望显示信息的代码。例如，循环一个产品信息的数组并将它们每一个都显示为一个新的表格。但视图仍然是 PHP 文件，所以你能写任何代码。

在哪里创建你的视图文件

视图文件保存在你的应用程序或者模块里的 views 文件夹中

```
application/views
```

```
modules/mymodule/views
```

你能在 views 文件夹中创建子文件夹来更好的组织你的文件

```
application/views/pages/about.php
application/views/products/details.php
application/views/errors/404.php
```

如何使用视图

典型的做法是在控制器里使用 View::factory() 方法来实例一个视图。
你的视图可以赋值给 Request->response 属性。

```
public function action_index()
{
    $this->request->response = View::factory('pages/about');
}
```

当你将视图像上面示例那样赋值给 request->response 属性的时候，视图会在应用程序需要它的时候自动显示。同时你可以使用视图内的 render() 方法来手动显示结果

```
public function action_index()
{
    $rendered_view = View::factory('pages/about')->render();

    $this->request->response = $rendered_view;
}
```

视图中的视图

你也可以把一个视图放到另一个视图里

```
<p>This is my main view file</p>

<?php echo View::factory('pages/menu')->render(); ?>
```

如何绑定和设置数据到视图

[返回目录]

视图本身只是一个 PHP 脚本。大多数情况下，如果你想提供动态内容让你的视图显示。你可以用几个不同的方法来设置和绑定动态数据到你的视图。

可以参考下列 /application/views/pages/about.php 视图

```
<h1><?php echo $title ?></h1>
<p><?php echo $date ?></p>
```

分配数据到视图变量

```
public function action_index()
{
    $view = View::factory('pages/about');

    $view->title = "The date is";
    $view->date = date('m/d/Y');

    $this->request->response = $view;
}
```

另一种方式你可以在创建视图时这样设置

```
public function action_index()
{
    $data['title'] = "The date is";
    $data['date'] = date('m/d/y');
    $view = View::factory('pages/about', $data);

    $this->request->response = $view;
}
```

绑定数据到视图

```
public function action_index()
{
    $view = View::factory('pages/about')
                ->bind('date', $date)
                ->bind('title', $title);

    $title = "The date is";
    $date = date('m/d/Y');
```

```
        $this->request->response = $view;
    }
```

使用视图内的 set() 方法

```
    public function action_index()
    {
        $view = View::factory('pages/about')
                        ->set('date', date('m/d/Y'))
                        ->set('title', "The date is");

        $this->request->response = $view;
    }
```

在视图中设置和使用全局数据

[返回目录]

你的应用程序可以同时显示几个视图文件并且它们使用传来的相同数据。例如：你可以在你的 header 模板和页面的 body 中显示相同的页面标题。你可以使用 View::set_global() 和 View::bind_global() 方法来赋值到全局变量。

设置全局变量

```
View::set_global('page_title', 'This is my page title');
```

绑定全局变量

绑定全局变量提供了一些很有趣方法，用来使用传递给视图的数据

```
View::bind_global('page_title', $page_title );
```

用法

在下面的示例中，一个 page/index 的请求显示了总共 3 个视图。主要的模板视图，page/about 视图和 sidebars/about 视图。

所有 3 个视图都会接收传入的 $page_title 属性并会在任何时候按请求自动修改值，直到你手动显示视图或者你允许请求自动处理显示

```
class Controller_Website extends Controller_Template
{
    public $page_title;

    public function before()
    {
        parent::before();

        View::bind_global('page_title', $this->page_title);
    }
}
```

```
class Controller_Page extends Controller_Website
{
	$this->request->response = View::factory('page/about');

	$this->page_title = 'Edit Item';

	$this->template->sidebar = View::factory('sidebars/about');
}
```

创建一个模板

[返回目录]

第一步，当用 Kohana 创建一个基于模板的网站时，当然就是创建模板。

默认情况下，Kohana 假定模板文件叫做 *template.php*，并且位于 views 文件夹

```
/home/kerkness/kohana/application/views/demo/template.php
```

你的模板文件要包含完整的你网站的 headers 和 footers 并且 echo 那些在你的控制器中要被插入的包含动态内容的变量。下面是一个包含动态内容的非常基本的模板

- title(string) – 这是页面的 title
- scripte(array) – 这是页面中需要导入的 Javascript 脚本组成的数组
- styles(array) – 这是模板中需要导入的 CSS 样式表组成的数组
- content(string) – 这是页面的内容

 点击这里查看源文件

```
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="en-US">
<head profile="http://gmpg.org/xfn/11">
    <title><?php echo $title ?></title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />

    <?php foreach ($styles as $file => $type) echo HTML::style($file, array('media' => $type)), "\n" ?>
    <?php foreach ($scripts as $file) echo HTML::script($file), "\n" ?>

</head>
<body>
    <?php echo $content ?>
</body>
</html>
```

你的模板可以是非常复杂或者非常简单，只要你喜欢，可以根据你的需要添加许多动态元素

继承模板控制器

[返回目录]

第二步，当用 Kohana 创建一个模板的网站需要继承 Controller_Template 类。虽然这不是必须的，因为你可以直接使用 Controller_Template 。继承控制器是一种很好的做法，那样你能轻松的设置默认值并且基于请求自定义输出。

我们的控制器该如何组织

- Controller_Template extends Controller
  - Controller_Demo extends Controller_Template
    - Controller_Page extends Controller_Demo

我们自定义的模板控制器叫作 demo.php ，创建在下列目录中

```
/home/kerkness/kohana/application/classes/controller/demo.php
```

这是我们定制的模板控制器

 点击这里查看源文件

```
<?php defined('SYSPATH') or die('No direct script access.');

  class Controller_Demo extends Controller_Template
  {

        public $template = 'demo/template';

        /**
         * before()方法在你的控制器动作执行前被调用
          * 在我们的模板控制器中，我们覆盖了这个方法，那么我们就能设置默认值。
         * 那么这些变量需要改变的时候我们的控制器也能使用它们
         */
        public function before()
        {
              parent::before();

                    if ($this->auto_render)
                    {
                          // Initialize empty values
                          $this->template->title   = '';
                          $this->template->content = '';
```

```
                $this->template->styles = array();
                $this->template->scripts = array();

            }
        }

        /**
         * after()方法在控制器动作执行后调用
         * 在我们的模板控制器中，我们覆写了这个方法，那么我们就能
         * 在模板显示之前做最后的一些改变
         */
        public function after()
        {
            if ($this->auto_render)
            {
                $styles = array(
                    'media/css/screen.css' => 'screen, projection',
                    'media/css/print.css' => 'print',
                    'media/css/style.css' => 'screen',
                );

                $scripts = array(
                    'http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js',
                );

                $this->template->styles = array_merge( $this->template->styles, $styles );
                $this->template->scripts = array_merge( $this->template->scripts,
$scripts );
            }
            parent::after();
        }
    }
```

我们的模板控制器执行了两个主要函数：

1. 准备默认值并通过使用 before() 方法在动作执行前配置
2. 通过使用 after() 方法在动作执行后修改并验证数值并在所有响应之前显示

在上面的实例中，before() 方法将我们的控制器中的变量设置为空变量。after()方法增加了我们的网站所有页面都需要的默认 javascript 和 css 文件

基本页面控制器

[返回目录]

现在我们已经设置好了一个继承自 Kohana 模板控制器的模板。那么我们可以定制让我们的网站如何响应。现在是时候创建一个实际的控制器页面来让我们能够开始提供服务。

我们的页面控制器被叫做 page.php ，他创建在下列目录

```
/home/kerkness/kohana/application/classes/controller/page.php
```

点击这里查看源文件

这是我们的页面控制器，它包含了两个动作，一个用来加载 home 页，另一个用来加载 contact 页。

```
<?php defined('SYSPATH') or die('No direct script access.');

  class Controller_Page extends Controller_Demo {

        public function action_home()
        {
              $this->template->title = __('Welcome To Acme Widgets');

              $this->template->content = View::factory('page/home' );
        }

        public function action_contact()
        {
              $this->template->title = __('Contact Information for Acme Widgets');

              $this->template->content = View::factory('page/contact' );
        }

  }
```

现在我们已经创建了 2 个视图页面。它们是 home.php 和 contact.php，它们创建在了下列目录。

```
  /home/kerkness/kohana/application/views/page/home.php
  /home/kerkness/kohana/application/views/page/contact.php
```

点击这里查看源文件

你可以使用这里两个视图文件。目前它们是很基本的，但是可以一定程度的定制。

home.php

```
<h1>Welcome to our homepage.</h1>

<p>We love our new Kohana web site</p>
```

contact.php

```
<h1>How to contact us.</h1>

<p>We don't like to be contacted so don't bother.</p>
```

Kohana 中的 HMVC

[返回目录]

Kohana 3.0 中众多强大特征中的一个就是能在请求流中的任何时候调用另一个请求。这种分层的 MVC 方法允许你组装一个复杂的客户层，真正利用了面向对象的强大优势。

一个最佳的分层结构

- 减少程序中不同部分之间的依赖
- 支持代码重用，组件，和模块
- 增加了扩展性同时减轻了维护

一些在客户端结构上使用到的 HMVC 设计

- 模块化界面元素或部件
- 应用和菜单控件
- 服务器的交互
- 可重用的应用流

HMVC 基本原理

一种理解 HMVC 的简单方法就是把它认为是没有额外服务器调用的 AJAX。比如，如果你有一个要显示用户列表的 AJAX 动作，你能在其他控制器中重复使用，而不是复制方法。

请求工厂

Kohana 中实现 HMVC 的办法是使用Request::factory() 方法。使用请求工厂，你能随时使用和完全执行在请求流中的 Kohana 请求。

Request::factory() 方法接受一个路由的 URI，并把它作为一个参数。当和 Kohana 强大的路由特征组成时，将会为你建立的应用程序带来完全的扩展性。

在控制器中使用请求工厂

下列示例将为你展示如何在另一个控制器内使用请求工厂。尽管它不能完全突出 HMVC 强大功能，它能展示如何让两个分开的请求位于同层。

```
class Controller_Static extends Controller
{
    /**
     * The following action loads page.
     * A sub request is called to load a dynamic menu
     */
```

```
    public function action_page()
    {
        $page_name = Request::instance()->param('page');

        $this->request->response = View::factory('page/'.$page_name)
                    ->bind('menu', $menu);

        $menu = Request::factory('static/menu')->execute()->response;
    }

    public function action_menu()
    {
        $page_name = Request::instance()->param('page');

        $this->request->response = View::factory('page/menu')
                    ->bind('links', $links);

        $links = Kohana::config('menu')->$page_name;
    }
}
```

在视图中使用请求工厂

另一个有效的方法是使用请求工厂来调用一个视图请求。比如下面的例子，我们用视图调用了动态菜单和动态页脚，而不是在控制器中。

```
<h1><?php echo $page_title ?></h1>

<?php echo Request::factory('page/menu')->execute()->response ?>

<div id="container">
  <?php echo $content ?>
</div>

<?php echo Request::factory('page/menu')->execute()->response ?>
```

把请求工厂作为 Kohana 和其他开源项目集成的基础

如果你分析了 Kohana 的 bootstrap.php 文件，你会注意到当 Request::instance 将 Request 类的实例创建之前并没有发生什么神奇的事。

在调用 Request::instance() 和 Request::factory() 之间的唯一区别是 instance()创建了一个唯一的 Request 类，并且它对请求做了一些主要的处理以及输出了一些必要的响应头(header)。

一个重要的注意点是创建一个 Request 的唯一实例或者输出头并不是完全必须的。只要 Kohana 被初始化，仅仅使用 Request::factory() 同样能完全执行性 Kohana 请求。这就是 Kohana 如何使用 Kohana-for-Wordpress 插件来整合 Wordpress。

路由基础

[返回目录]
Kohana 3.0 中的 路由类支持你无限制的配置特殊格式的请求来映射到指定的控制器和动作上。
你可以在 application/bootstrap.php 文件或者在 modules/我的模块/init.php 文件中调用 Route::set() 方法来定义路由

你可以在 application/bootstrap.php 文件中看到如下的默认路由。每个路由至少有一个名称一个 uri 以及给 uri 设置一个默认值。

```
Route::set('default', '(<controller>(/<action>(/<id>)))')
    ->defaults(array(
        'controller' => 'welcome',
        'action' => 'index',
    ));
```

这个路由将会处理诸如下列格式的请求。

```
http://example.php/index.php/<controller>/<action>/<id>
```

这个请求的 URI 中所有部分都是可选的。如果路由中的某一部分没有指明，那么路由会使用默认值。
这里有一个相同的路由，但是控制器和动作部分都不能可选。

```
Route::set('required', '<controller>/<action>(/<id>)')
    ->defaults(array(
        'controller' => 'welcome',
        'action' => 'index',
    ));
```

一个基本的定制路由

重点：当创建定制路由时，它们总是要定义在默认路由之前。你的默认路由总是定义在最后。

下列路由将把所有 example.com/index.php/products/<action>/<id> 的请求转到 inventory 控制器的动作上。

```
Route::set('products', 'products(/<action>(/<id>))')
    ->defaults(array(
        'controller' => 'inventory',
        'action' => 'index',
    ));
```

一个没有可选部分的定制路由

下面的路由将 example.com/index.php/custom 请求转到了 welcome 控制器的 index 动作上。

```
Route::set('custom', 'custom')
        ->defaults(array(
            'controller' => 'welcome',
            'action' => 'index',
        ));
```

只有一个动态部分的定制路由

下面的路由将 example.com/index.php/about, example.com/index.php/faq 和 example.com/index.php/locations 请求转到了 page 控制器的 static 动作上

```
Route::set('static', '<page>', array('page' => 'about|faq|locations'))
            ->defaults(array(
                        'controller' => 'page',
                        'action' => 'static',
            ));
```

含有 id 和 slug 的定制路由

必须设置在 defaut 路由之前

```
Route::set('idslug', '<controller>/<action>/<id>-<slug>', array('id'=>'[0-9]+', 'slug'=>'.+'))
            ->defaults(array(
                        'controller' => 'home',
                        'action'       => 'index',
            ));
```

忽略溢出的路由

[返回目录]

默认的 Kohana 路由看起来像这样

```
controller/action/id
```

但是如果你还像将你的路由支持下面这样

```
controller/action/id/and/anything/else/you/want/to/add/at/random
```

你需要在 application/bootstrap.php 文件中改变你的默认路由并且增加一个小的正则表达式

```
Route::set('default', '(<controller>(/<action>(/<id>(/<stuff>))))', array('stuff' => '.*'))
    ->defaults(array(
        'controller' => 'welcome',
        'action' => 'index',
));
```

建立一个路由和控制器来处理国际化的静态页面

[返回目录]

Kohana 能非常好的用来创建动态网站和应用。然后并不是所有的东西都需要动态化。创建每个静态页面制作一个控制器/动作组合是有点小困难的。这里就告诉你如何能创建一个控制器以及路由来处理静态页面并且支持国际化。那样你的网站就能支持多语言了。

路由

我打算做两个支持静态页面的路由示例。第一个假设我们需要支持一组定量的页面并且使用非常干净的 URL。第二个将支持任何网页数并在请求的 URI 中带有 page 前缀

```
/**
 * 这个路由支持页面 about，faq 和 locations
 * 每个页面都能使用它们的名字来访问,例如
 * http://example.com/about 或者 http://example.com/faq
 */
Route::set('static', '<page>', array('page' => 'about|faq|locations'))
        ->defaults(array(
                'controller' => 'page',
                'action' => 'static',
        ));
```

或者使用一个支持动态的页面名的路由

```
/**
 * 这个路由支持任意数量的页面.
 * 每个页面能都按下面的 URL 中的页面名来访问.
 * http://example.com/page/page_name
 */
Route::set('static2', 'page/<page>', array('page' => '.*'))
        ->defaults(array(
                'controller' => 'page',
                'action' => 'static',
        ));
```

控制器

相同的控制器/动作组合能够支持上面两种不同的路由。动作会负责寻找请求的页面名，然后加载相应的视图。

我下面的示例使用了默认的 Kohna 模板控制器

```
<?php defined('SYSPATH') or die('No direct script access.');

class Controller_Pages extends Controller_Template {

	public function action_static()
	{
		// 获得我们请求的页面名 e
		$page = Request::instance()->param('page');

		// 分配适当的视图文件到模板内容
		$this->template->content = View::factory('page/'. i18n::$lang .'/'. $page );
	}
}
```

视图模板

你可能注意到了在上面的控制器中我们使用一个来自 Kohana i18n 类的静态属性来加载视图。
i18::$lang 将返回你 Kohana 网站的默认语言。你可以在 application/bootstrap.php 文件中定义它。
如果你的页面名字叫 about 而你的默认语言是 en-US 。那么控制器将寻找下面的视图文件。

```
page/en-us/about
```

完整的视图路径可能类似下面这样

```
/home/kerkness/kohana/application/views/page/en-us/about.php
```

你必须创建你想支持的所有语言的视图页面。

i18n 就是 internationalization 的缩写，因为首字母 i 和尾字母 n 之间有 18 个字母，所以简写为 i18n，其实就是国际化的意思

多语言路由

[返回目录]

有许多的方法来用 Kohana 完成国际化的功能。一种方法就是在路由里指定语言。

例如

```
http://example.com/en-us/products // For English
http://example.com/fr-fr/products // For French
```

你可以在你的路由里捕捉到请求中的语言部分。下面的示例仅仅允许请求中的英语，法语和荷兰语，默认是英语。

```
Route::set('default', '(<lang>/)(<controller>)(/<action>(/<id>))', array('lang' =>
'(en-us|fr-fr|nl-nl)', 'id'=>'.+'))
        ->defaults(array(
            'lang' => 'en-us',
            'controller' => 'welcome',
            'action' => 'index',
));
```

那些现在可以在控制器中设置语言。下面的示例就在定制的模板控制器中设置了语言。

```
class Controller_Website extends Controller_Template
{
    public function before()
    {
        // Set the language
        I18n::$lang = Request::instance()->param('lang');
    }
}
```

在 application/messages 目录里创建语言文件

```
/home/kerkness/kohana/application/i18n/fr.php
```

```
return array
(
        'Hello World' => 'Bonjour Monde',
);
```

在 application/view 目录下合适的目录里创建特殊语言视图

```
/home/kerkness/kohana/application/views/fr-fr/index.php
```

```
<p>Merci de visiter notre site Internet.</p>

<p>Voici mon contenu de homepage...
```

你所有的控制器都将使用适合的语言。

```
class Controller_Welcome extends Controller_Website
{
	public function action_index()
	{
		// Hello World 字符串将被翻译
		$this->template->title = __('Hello World');

		//加载特定语言的视图
		$this->template->content = View::factory('page/'.I18n::$lang.'/index');
	}
}
```

建立子目录路由

[返回目录]

如果你想在你的控制器里使用子目录，并且动态的在你的路由里得到支持。你可以使用下面的示例来作为一个起点。

支持一个单独子目录

让我们来考虑下列控制器（Controller_Foo_Bar）

```
/home/kerkness/application/classes/controller/foo/bar.php
```

若你想从下列 url 来访问这个控制器

```
http://example.com/foo/bar/action/id
```

那你可以按照下面的路由来做

```
Route::set('default', '(<directory>(/<controller>(/<action>(/<id>))))')
    ->defaults(array(
        'directory'  => 'foo',
        'controller' => 'bar',
        'action'     => 'index',
    ));
```

支持多种子目录

如果你希望如下列 url 那样支持多种子目录

```
http://example.com/my/sub/directory/controller/action/id
```

你可以在路由中加入点正则表达式

```
Route::set('default', '(<directory>(/<controller>(/<action>(/<id>))))', array('directory' =>
    '.+?'))
      ->defaults(array(
        'directory'  => 'foo',
        'controller' => 'bar',
        'action'     => 'index',
    ));
```

一点警告！！

一些事情必须知道。上面的路由示例假定你所有的控制器都是用了子目录。如果你仅仅有一些控制器位于子目录，最好考虑下为每一个都定义像下面示例这样的路由

```
Route::set('default', 'foo/<controller>(/<action>(/<id>))')
    ->defaults(array(
        'directory'  => 'foo',
        'controller' => 'bar',
        'action'     => 'index',
```

```
));
```

创建一个自定义的 404 页面

[返回目录]

为了在你的 Kohana 应用程序中有一个自定义的 404 页面。你需要试着捕捉所有的无效路由并将它们转到一个显示 404 信息的特殊控制器/动作。该路由就是下面例子这样。

大多数定制路由应被定义在你的默认路由之前，而捕捉无效部分的路由应该放在你的默认路由之后。
（*注意：你的默认路由必须比未编辑的 bootstrap 文件里的默认路由来的更特别）
（这个注意点非常含糊，也没给出什么例子或者引用来表述如何编辑默认路由配置到函数属性，因此让用户比较混乱，不知道该如何去做）

```
Route::set('catch_all', '<path>', array('path' => '.+'))
    ->defaults(array(
        'controller' => 'errors',
        'action' => '404'
));
```

不要忘记创建控制器和相应的视图文件

```
/home/kerkness/kohana/application/classes/controller/errors.php
```

```
class Controller_Errors extends Controller
{
    public function action_404()
    {
        $this->request->status = 404;
        $this->request->response = View::factory('errors/404');
    }
}
```

还有视图文件

```
/home/kerkness/kohana/application/views/errors/404.php
```

```
<h1>404 Not Found</h1>
```

示例：

```
// 允许 http://test/security/login, http://test/security/logout, http://test/security/register,
http://test/security/verifyemail
Route::set('security', 'security/<action>', array('action' => 'login|logout|register|verifyemail'))
        ->defaults(array(
                'controller' => 'security',
        ));
```

```
// 允许 http://test/, http://test/index, http://test/restricted
Route::set('home', '(<action>(/<ignore>))', array('action' => 'index|restricted', 'ignore' => '.+'))
        ->defaults(array(
                'controller' => 'home',
                'action'     => 'index',
        ));

// 发送一切到 404
Route::set('catch_all', '<path>', array('path' => '.+'))
        ->defaults(array(
                'controller' => 'errors',
                'action'     => '404',
        ));
```

手动触发 404 页面

- 待办：增加一个抛出异常的示例
- 待办：请提供如何为适当功能编辑默认路由的详细资料或参考

内部和外部的不同请求

[返回目录]

当使用 Kohana HMVC 的时候，它能很完美的区分来自外部（那些来自客户端的）和内部（你的应用程序内部制造的子请求）的请求。我们来验证下的模板控制器中的 before() 方法内的请求是来自内部还是外部。

```
<?php
class Controller_Sample extends Controller_Template {

	public $internal = FALSE;

	public function before()
	{
		parent::before();

		if ($this->request != Request::instance()) {
			$this->internal = TRUE;
		}
	}
}
```

另外这里有个很好的技巧来重新定义函数，以便你来区分内部或者是外部的请求。

```
<?php
class Controller_Sample extends Controller {
    public function before()
    {
		if ($this->internal == TRUE) {
			$this->request->action = 'internal_'.$this->request->action;
		}
    }
    public function action_index()
    {
		// 如果请求时外部的，这个动作将被调用
    }
    public function action_internal_index()
    {
			//如果请求时内部的，这个动作将被调用
    }
}
```

相当有用，是吧？你可以构建子请求的页面，同时你也可以发送 ajax 或者 HTTP 请求道指定控制器。

讨论

1. “$this->request === Request::instance()”有能力测试出请求来自内部吗？
2. 是的，这些不需要继承 Request 类，这是多余的。

如何重定向用户请求

[返回目录]

如果你想重定向用户请求道一个新的页面或者新的路由。你可以使用 Request 类的 redirect() 方法.

用法

```
Request::instance()->redirect('http://kohanaphp.com');
```

```
Request::instance()->redirect('controller/action');
```

```
Request::instance()->redirect( Route::get('demo-sample')->uri(array('category'=>'books', 'id'=>5)) );
```

如何测试路由

[返回目录]

你可能经常想测试下你刚创建的路由。你可以直接使用 URI，但是你也可以用这里的一部分来做这个。这个部分看起来就像这样：

```
// 这里是你定义的路由
// ...
Route::set('post', 'post/<id>', array('id' => '[\d]+'))
    ->defaults(array(
        'controller' => 'post',
        'action'     => 'index',
));
// ...

// 测试 URI
$uri = 'post/1';
// This will loop trough all the defined routes and
// tries to match them with the URI defined above
foreach (Route::all() as $r)
{
    echo Kohana::debug($r->matches($uri));
}
exit;
```

正如你所看到的，你放置的这一小部分只是放在 bootstrap.php 文件里的路由区域的下方，并开始调试你的路由。当你运行这个的时候你会获得类似这样的信息：

```
array(3) (
    "id" => string(1) "1"
    "controller" => string(4) "post"
    "action" => string(5) "index"
)
array(2) (
    "controller" => string(4) "post"
    "action" => string(1) "1"
)
```

注意：这和你所打开什么页面并无关系，只要你的 bootstrap.php 被调用（它在每个页面都有用到）

反向路由和分页

[返回目录]

Kohana 3 中路由中最牛逼的事之一就是你能按定义好的路由属性自动生成一个 URL 链接

这就允许你改变路由的模式，并且把它应用在控制器的两端，无论何处路由都能生成控制器

反向路由示例

按下面例子在你的 bootstrap.php 内设置路由

```
Route::set('article_posts', 'article/read(/<id>(/<title>(/<page>)))', array('id' => '[0-9]+', 'page'
=> '[0-9]+))
        ->defaults(array(
                'controller'                    => 'article',
                'action'        => 'view',
                'id'                            => NULL,
                'title'                         => NULL,
                'page'                          => NULL,
));
```

将它反向路由！

```
$route = Route::get('article_posts')->uri(array('id' => 35, 'title' => 'my-test-article', 'page' => 2));
echo $route;
```

结果是：

```
article/read/35/my-test-article/2
```

现在让我们说，我们决定要改变 URL 中显示页面的方式，所以我们改变 article_posts 路由

```
Route::set('article_posts', 'article/read(/<id>(/<title>(/?page=<page>)))', array('id' => '[0-9]+',
'page' => '[0-9]+'))
        ->defaults(array(
                'controller'                    => 'article',
                'action'        => 'view',
                'id'                            => NULL,
                'title'                         => NULL,
                'page'                          => NULL,
));
```

输出它：

```
		$route = Route::get('article_posts')->uri(array('id' => 35, 'title' => 'my-test-article', 'page' => 2));
		echo $route;
```

结果是：

```
article/read/35/my-test-article/?page=2
```

哈哈，做得好！

路由和分页

另一个关于反向路由牛逼的事情就是你可以使用它们为你分页！如果你想尝试这个例子，先确认下你有分页模块(pagination)，并且在你的 bootstrap.php 中启用了它。（也就是在 bootstrap.php 中去掉了 pagination 前面的注释）

例如，在我们的 Controller_Article 中，我们有 action_view() 动作

```
public function action_view()
{
		$article['pages'] = 5;

		$pagination = Pagination::factory(array(
				'current_page'     => array('source' => 'route', 'key' => 'page'),
				'total_items'      => $article['pages'],
				'items_per_page' => 1,
				'view'                => 'pagination/basic',
				'auto_hide'         => FALSE,
		  ));

		echo $pagination;
}
```

现在，当我们浏览页面，你就能看到分页已经为我们生成了设置的非常好的链接，哈哈！,

如何打开和配置数据库模块

[返回目录]

Kohana 3.0 有一个很强壮的数据库模块。默认情况下数据库模块支持 MySQL 和 PHP-PDO 驱动

数据库模块已经包含在了 Kohana 3.0 安装程序之中，但是还需要在使用之前启动它。在你的 application/bootstrap.php 文件里修改 Kohana::modules() 方法中 database 模块，就像下面这样。

```
Kohana::modules(array(
            'userguide'          => MODPATH.'userguide',
            'database'           => MODPATH.'database',    // Database access
            'pagination'         => MODPATH.'pagination',
  ));
```

当模块启动以后，你还需要提供一个配置文件来使模块知道如何连接到你的数据库。你能在 modules/database/config/database.php 中找到一个配置文件的示例。复制这个配置文件到你的应用层

```
  cp -a modules/database/config/database.php application/config/database.php
```

展开配置文件并且为你的数据库连接做一些必要的修改。下列的示例文件展示了 2 个 mysql 连接。你能定义许多你所需要的数据库连接，但是你必须确定有一个连接叫 default

```
return array
  (
            'default' => array
            (
                        'type'          => 'mysql',
                        'connection' => array(
                                    'hostname'     => 'localhost',
                                    'username'     => 'dbuser',
                                    'password'     => 'mypassword',
                                    'persistent' => FALSE,
                                    'database'     => 'my_db_name',
                        ),
                        'table_prefix' => '',
                        'charset'         => 'utf8',
                        'caching'         => FALSE,
                        'profiling'       => TRUE,
            ),
            'remote' => array(
                        'type'          => 'mysql',
```

```
                        'connection' => array(
                                'hostname'     => '55.55.55.55',
                                'username'     => 'remote_user',
                                'password'     => 'mypassword',
                                'persistent' => FALSE,
                                'database'     => 'my_remote_db_name',
                        ),
                        'table_prefix' => '',
                        'charset'          => 'utf8',
                        'caching'          => FALSE,
                        'profiling'       => TRUE,
                ),
);
```

MySQL 数据库能接受下面的连接配置选项

- 字符串的主机名 hostname　　*端口和套接字可以添加到主机名
  例如：localhost:3306
- 字符串的套接字 socket
- 字符串的用户名 username
- 字符串的密码　　password
- 布尔值的持久链接　　persistent
- 字符串的数据库名　　database

PDO 数据库能接受下列这些连接配置选项

- 字符串的数据源 dsn
- 字符串的用户名 username
- 字符串的密码　　password
- 布尔值的持久链接　　persistent
- 字符串的标识符 identifier

用查询生成器产生 CRUD

[返回目录]

Kohana 3.0 数据查询生成器能为你的数据库简单的创建 CRUD 语句（CRUD = Create Read Update Delete 就是数据库的基本操作创建，读取，更新，删除）。

这里的示例使用了下面的数据库结构

```
CREATE TABLE `users` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `username` varchar(30) NOT NULL,
  `password` varchar(40) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

创建数据库记录

创建数据库记录需要使用 DB::insert() 方法。这个请求的基本格式如下。

```
DB::insert('table_name', array('column'))->values(array('column_value'))->execute();
```

如果命令成功，你就能得到一个包含 insert_id 和 total_rows (总共影响到的行数)组成的数组

用法

在 try/catch 块中执行调用，那么你就能捕捉到异常。

```
try
{
    list($insert_id, $total_rows) = DB::insert('users', array('username','password'))
        ->values(array('myusername',sha1('mypassword')))
        ->execute();

    echo $insert_id .' '.$total_rows ;
}
catch ( Database_Exception $e )
{
    echo $e->getMessage();
}
```

读取数据库记录

读取数据库记录需要使用到 DB::select() 方法。这个请求的基本格式如下

```
// 返回一个结果对象
$result = DB::select('column')->from('table_name')->execute();

// 结果作为数组返回
$result = DB::select('column')->from('table_name')->execute()->as_array();

// 结果作为标准类对象返回 s
$result = DB::select('column')->from('table_name')->as_object()->execute();

// 仅返回第一行
$result = DB::select('column')->from('table_name')->execute()->current();
```

你可以按你所需要的在上面的示例中选择一个方法。

```
// 返回一个列
$result = DB::select('column')->from('table_name')->execute()->current();

//返回 3 列
$result = DB::select('column', 'column2',
'column3')->from('table_name')->execute()->current();
```

```
// 列名的别名
$result = DB::select(array('longcolumnname1', 'col1'), array('longcolumnname2',
'aliascol2'))->from('table_name')->execute()->current();
```

你可以用 where() 方法来选择特定的记录

```
$result = DB::select()->from('table_name')->where('column','=','value')->execute();
```

有多种多样的方法可用来 构建复杂的 SELECT 语句。查看 API 文档来了解更多细节

用法

在 try/catch 块中执行调用，那么你就能捕捉到异常。

```
public function action_select()
{
    try
    {
        $user = DB::select()->from('users')
                    ->where('id','=', $this->request->param('id'))
                    ->execute()->current();

        echo Kohana::debug($user);
    }
    catch( Database_Exception $e )
    {
        echo $e->getMessage();
    }
}
```

更新数据库记录

更新数据库记录需要用到 DB::update() 方法. 这个请求的基本格式如下

```
$total_rows =
DB::update('table_name')->set(array('column'=>'value'))->where('column','=','value')->execute()
;
```

如果命令成功，你就能得到结果中影响到的行数

用法

在 try/catch 块中执行调用，那么你就能捕捉到异常。

```
try
{
    $total_rows = DB::update('users')->set(array('username'=>'newuser'))
            ->where('id','=',2)->execute();

    echo Kohana::debug($total_rows);

}
catch( Database_Exception $e )
{
    echo $e->getMessage();
}
```

删除数据库记录

删除数据库记录需要用到 DB::delete() 方法。这个请求的基本格式如下

```
$total_rows = DB::delete('table_name')->where('column','=','value')->execute();
```

如果命令成功，你就能得到结果中影响到的行数

用法

在 try/catch 块中执行调用，那么你就能捕捉到异常。

```
try
{
    $total_rows = DB::delete('users')->where('id','=',2)->execute();

    echo Kohana::debug($total_rows);

}
catch( Database_Exception $e )
{
    echo $e->getMessage();
}
```

用查询生成器来高级查询

[返回目录]

查询生成器能快速并简单的生成和运行简单的 SQL 查询。
然后某些文档缺少了更多复杂的查询。

语句和表达式

有些时候你想运行一个快速更新来使某列的数值增加，诸如一个视图或者列数。SQL 看起来就像这样

```
UPDATE `pages` SET `views` = views + 1 WHERE `id` = 1
```

假如是这样，我们能使用 DB::expr(‘’) 函数来创建表达式

```
DB::update('pages')
        ->set(array('views' => DB::expr('views + 1')))
        ->where('id', '=', 1)
        ->execute();
```

执行

当调用执行，例如在一个 InnoDB 数据库，我们的查询不是 SELECT,INSERT,UPDATE,或者 DELETE

```
BEGIN WORK;
```

我们能使用定制的查询函数来调用这个

```
DB::query(NULL, "BEGIN WORK")->execute();
```

同样，如果你想的话，你可以回调 INSERT 和 UPDATE 查询所创建的 TRUE/FALSE 情况来决定是否委托(COMMIT)或者回滚(ROLLBACK)

```
DB::query(NULL, "BEGIN WORK")->execute();

// 页面数据数组
$page_data = array(
        'id'        => NULL,
        'title'     => 'My New Page',
);

$page = (bool) DB::insert('pages', array_keys($page_data))
                ->values($page_data)
                ->execute();

$page_cat_data = array(
        'page_count'        => DB::expr('page_count + 1'),
```

```
);

$page_cat = (bool) DB::update('page_categories')
                        ->set($page_cat_data)
                        ->where('id', '=', 1)
                        ->execute();

if($page AND $page_cat)
{
        DB::query(NULL, "COMMIT")->execute();
}
else
{
        DB::query(NULL, "ROLLBACK")->execute();
}
```

联合（Joins）

用查询生成器来执行一个联合是非常简单的。在这个例子中我们得到了一个 articles 和 users 的表，我们希望得到 user 的 username 和 article 的 title

SQL:

```
SELECT `articles`.`title`, `users`.`username`
FROM `articles`
    JOIN `users`
    ON (`users`.`id` = `articles`.`user_id`)
WHERE `articles`.`id` = 1
LIMIT 1
```

查询生成器：

```
$query = DB::select('articles.title', 'users.username')
                        ->from('articles')
                        ->where('articles.id', '=', 1)
                                ->join('users')
                                ->on('users.id', '=', 'articles.user_id')
                        ->limit(1)
                        ->execute();
```

构建复杂的 SELECT 语句

[返回目录]

这里有多种方法来构建复杂的 SELECT 语句

选择特定的列

```
DB::select('column1','column2')->from('table_name');
```

```
SELECT `column1`, `column2` FROM `table_name`
```

选择列 AS

```
DB::select(array('column','my_column'))->from('table_name')->compile($db);
```

```
SELECT `column` AS `my_column` FROM `table_name`
```

join()

```
DB::select()->from('table_name')->join('table_2')->on('table_2.table_id', '=', 'table_name.id');
```

```
SELECT * FROM `table_name` JOIN `table_2` ON `table_2`.`table_id` = `table_name`.`id`
```

group_by()

```
DB::select()->from('table_name')->group_by('column');
```

```
SELECT * FROM `table_name` GROUP BY `column`
```

```
DB::select()->from('table_name')->group_by(array('column1', 'mycol'));
```

```
SELECT * FROM `table_name` GROUP BY `column1` AS `mycol`
```

having()

```
DB::select()->from('table_name')->having('column','=','value');
```

```
SELECT * FROM `table_name` HAVING `column` = 'value'
```

and_having()

```
DB::select()->from('table_name')->having('column','=','value')->and_having('column2','=','value')
;
```

```
  SELECT * FROM `table_name` HAVING `column` = 'value' AND `column2` = 'value'
```

or_having()

```
DB::select()->from('table_name')->having('column','=','value')->or_having('column2','=','value');
```

```
DB::select()->from('table_name')->having('column','=','value')->or_having('column2','=','value');
```

having_open()

```
  DB::select()->from('table_name')->having_open()->having('column','=','value')
        ->or_having('column2','=','value')->having_close();
```

```
  SELECT * FROM `table_name` HAVING (`column` = 'value' OR `column2` = 'value')
```

and_having_open()

```
DB::select()->from('table_name')->where('column','=','value')->and_having_open()->having('col
umn2','=','value')
        ->and_having('column3','=','value')->and_having_close();
```

```
  SELECT * FROM `table_name` WHERE `column` = 'value' HAVING (`column2` = 'value' AND
`column3` = 'value')
```

or_having_open()

```
DB::select()->from('table_name')->where('column','=','value')->or_having_open()->having('colu
mn2','=','value')
        ->or_having('column3','=','value')->or_having_close();
```

```
  SELECT * FROM `table_name` WHERE `column` = 'value' HAVING (`column2` = 'value' OR
`column3` = 'value')
```

order_by()

```
DB::select()->from('table_name')->order_by('column', 'ASC');
```

```
SELECT * FROM `table_name` ORDER BY `column` ASC
```

limit()

```
DB::select()->from('table_name')->limit(10);
```

```
SELECT * FROM `table_name` LIMIT 10
```

offset()

```
DB::select()->from('table_name')->limit(10)->offset(50);
```

```
SELECT * FROM `table_name` LIMIT 10 OFFSET 50
```

where()

```
DB::select()->from('table_name')->where('column','=','value');
```

```
SELECT * FROM `table_name` WHERE `column` = 'value'
```

and_where()

```
DB::select()->from('table_name')->where('column','=','value')->and_where('column2','=','value');
```

```
SELECT * FROM `table_name` WHERE `column` = 'value' AND `column2` = 'value'
```

or_where()

```
DB::select()->from('table_name')->where('column','=','value')->or_where('column2','=','value');
```

```
SELECT * FROM `table_name` WHERE `column` = 'value' OR `column2` = 'value'
```

where_open()

```
DB::select()->from('table_name')->where_open()->where('column','=','value')
        ->or_where('column2','=','value')->where_close();
```

```
SELECT * FROM `table_name` WHERE (`column` = 'value' OR `column2` = 'value')
```

and_where_open()

```
DB::select()->from('table_name')->where('column','=','value')->and_where_open()->where('column2','=','value')
        ->or_where('column3','=','value')->and_where_close();
```

```
SELECT * FROM `table_name` WHERE `column` = 'value' AND (`column2` = 'value' OR `column3` = 'value')
```

or_where_open()

```
DB::select()->from('table_name')->where('column','=','value')->or_where_open()->where('column2','=','value')
        ->and_where('column3','=','value')->or_where_close();
```

```
SELECT * FROM `table_name` WHERE `column` = 'value' OR (`column2` = 'value' AND `column3` = 'value')
```

如何使用分页模块

[返回目录]

Kohana 3.0 带来了一个模块来协助给数据库记录分页。分页的作用是处理把数据库记录分成几个页面并且支持给用户上一页和下一页的链接。

下面的例子使用了 DB::select() 查询生成器，然而分页模块并不依赖任何类型的数据集。你可以使用这个模块而让用户不注意到数据页是通过了哪个数据源的。

启用模块

分页(Pagination)模块包含在了 Kohana3.0 的安装中，但是你在使用前必须要启用它。像下列示例那样在你的 application/bootstrap.php 文件中修改 Kohana::modules() 方法的调用，使它包含 Pagination 模块

```
Kohana::modules(array(
        'userguide'         => MODPATH.'userguide',
        'database'      => MODPATH.'database',    // Database access
        'pagination'        => MODPATH.'pagination',
));
```

基本用法

下面的控制器用查询生成器从数据库加载了一个记录集。生成了一个允许用户一页页浏览记录的分页控件。

```
public function action_page()
{
    // 定义我们的模板视图，绑定变量
        $this->template->content = View::factory('page/list')
                ->bind('results', $results)
                ->bind('page_links', $page_links);

    // 从数据库获得记录总数
        $count = DB::select(DB::expr('COUNT(*) AS
mycount'))->from('users')->execute('alternate')->get('mycount');

    // Create an instance of Pagination class and set values
        $pagination = Pagination::factory(array(
                'total_items'       => $count,
                'items_per_page' => 20,
        ));
```

```
        // 加载当前页面的指定结果集
            $results = DB::select()->from('users')
                                    ->order_by('id','ASC')
                                    ->limit($pagination->items_per_page)
                                    ->offset($pagination->offset)->execute();

        // 生成分页链接
            $page_links = $pagination->render();
    }
```

注意：$pagination->offset 在 3.0.2 版本才被支持。

高级用法

你可以在创建类实例的时候按所定义的配置来修改分页(pagination)类如何运行以及提供的链接

```
$pagination = Pagination::factory(array(
    'current_page'     => array('source' => 'query_string', 'key' => 'p'),
    'total_items'      => $count,
    'items_per_page' => 40,
    'view'                 => 'pagination/page_links',
    'auto_hide'         => FALSE,
));
```

使用配置(config)文件

分页(Pagination)类使用一个配置文件来决定要使用的默认设置。这个配置会和创建一个新类实例的时候定义的配置合并。

创建你自己的配置文件，把 modules/pagination/config/pagination.php 文件复制到 application/config/pagination.php

```
cp modules/pagination/config/pagination.php application/config/pagination.php
```

这里是一个定制配置的示例

```
return array(

        'default' => array(
                    'current_page'     => array('source' => 'query_string', 'key' => 'p'),
                    'total_items'      => 0,
                    'items_per_page' => 40,
                    'view'                 => 'pagination/pretty',
                    'auto_hide'         => FALSE,
        ),
);
```

配置设置的说明

- 数组 current_page：是一个定义了当前页面 source 和 key 的数组。Source 可以是 route 或者 query_string 而 key 可以是任何你想用到的字符串。
  - 如果 source = route 那么分页(Pagination)类会从当前路由需找 key
  - 如果 source = query_string 那么分页(Pagination)类会从当前查询的字符串(query string)中寻找 key
- 整型 total_items： 将包含所有通过分页的项数。这是仅有的一个需要在分页(Pagination)类实例被创建时同时要设置的一个配置。
- 整型 items_per_page：每页显示的项数
- 字符串 view：记录集分页的链接所用到的视图文件。
- 布尔值 auto_hide：如果设置为 true，render()方法将在总页面数小于或者等于 1 的时候不返回任何值。

如何关闭一个数据库连接

[返回目录]

如果你发现正需要强行关闭数据库连接（可能正在 php-cli 下工作）你有一对选择

```
foreach(Database::$instances as $db)
{
    $db->disconnect();
    unset($db);
}
Database::$instances = array();
```

或只是调用

```
Database::$instances = array();
```

Cookie 和 Session 的使用

[返回目录]

Kohana 3.0 提供了一对能使 Cookie 和 Session 做起来简单的类。

在高层次时，Session 和 Cookie 提供相同的功能。他们都允许开发者保存那些以后动作需要用到的临时或者持久的信息。

什么时候使用 Cookie

- Cookies 通常用来存放那些需要保存很长一段时间的非私有的数据。例如存放用户的 id 或者用户偏爱的语言

什么时候使用 Session

- Session 通常被用来存放临时的并且私有的数据
- 非常敏感的数据应该用 Session 类通过 Kohana 数据库或本地适配器来储存。如果 Cookies 适配器也要这样使用的话，通常都已经被预先加密。

更多的信息关于 Session 变量的最佳练习，可以在这里查看session 的七宗罪

如何设置 Session 和 Cookie 的值

```
// 设置 Cookie
Cookie::set('key_name', 'value');
Cookie::set('user_id', $user_id);
```

```
// 设置 Session
Session::instance()->set('key_name', 'value');
Session::instance()->set('user_id', $user_id);
```

如何获取 Session 和 Cookie 的值

```
// 获得 Cookie 数据
$value = Cookie::get('key_name', 'default value');
$value = Cookie::get('user_id', NULL);
```

```
// Getting Session Data
$value = Session::instance()->get('key_name', 'default value');
$value = Session::instance()->get('user_id', NULL);
```

如何删除 Session 和 Cookie 的值

```
// 删除 Cookie
```

```
Cookie::delete('key_name');
Cookie::delete('user_id');
```

```
// 删除 session 变量
Session::instance()->delete('key_name');
Session::instance()->delete('user_id');
```

如果你正在使用 session，你可能想删除或销毁完整的 session

```
if( Session::instance()->destroy() )
{
      echo "Session has been completely removed";
}
```

设置你的 Cookie 的属性

Cookies 使用几个属性用来帮助你决定数据如何加密以及什么网站或域名可以访问它。这些属性设置为 Cookies 类的静态属性

```
// 设置魔法 salt 到 cookie
Cookie::$salt = 'kookykookie';
```

```
// 设置 cookie 多久过期
Cookie::$expiration = 43200;
```

```
// 限制有效的 cookie 路径
Cookie::$path = '/';
```

```
// 限制可以访问 cookie 的域名
Cookie::$domain = 'www.kerkness.ca';
```

```
// 只可以用安全连接传输 cookie
Cookie::$secure = TRUE;
```

```
// 只可以用 HTTP 传输 cookie,不能用 Javascript 传输。
Cookie::$httponly = TRUE;
```

使用不同的 Session 适配器

当你要创建或访问一个 Session 类的实例的时候，你可以决定你想用哪个 session 适配器。

你可以选择的 session 适配器有这些

- Native:将 session 数据默认存放在你的 web 服务器上。例如 ,如果你正在 Apache2 上运行 PHP，那么 session 数据就默认存放在你的 php.ini 文件里设置的路径指定的文件中。
- Database:将 session 数据放在一个数据库中。（需要数据库模块）
- Cookie：将 session 数据存放在一个局部 cookie.

默认的适配器是 native，所以你如果使用 native，那么你不用做任何事。如果你想用 Database 或者 Cookie 适配器的话，你需要使用 Session::instance()来定义适配器的类型。

```
// Cookie 适配器
Session::instance('cookie')->set('key_name', 'value');
$value = Session::instance('cookie')->get('key_name');
```

```
// Database 适配器
Session::instance('database')->set('key_name', 'value');
$value = Session::instance('database')->get('key_name');
```

数据库 Session 模式

如果你使用 session Database 的适配器，那么你需要在你的数据库中建立如同下列这样的表结构。

```
CREATE TABLE `sessions` (
    `session_id` VARCHAR( 24 ) NOT NULL,
    `last_active` INT UNSIGNED NOT NULL,
    `contents` TEXT NOT NULL,
    PRIMARY KEY ( `session_id` ),
    INDEX ( `last_active` )
) ENGINE = MYISAM ;
```

配置你的 Session 适配器

你可以在下面的文件里创建一个 session 配置来为每个 Kohana 适配器应用配置信息。

```
APPPATH/config/session.php
```

下面的配置就是为每个适配器的。

- Cookie Session
  - **name**: 设置 cookie 名
  - **encrypted**:如果 cookie 需要加密那么就用布尔值来标记
  - **lifetime**:cookie 能保存多少秒时间
- Native Session
  - **name**:Session 名
  - **encrypted**: 如果 Session 数据需要加密那么就用布尔值来标记
  - **lifetime**:session 在多少秒时间内有效
- Database Session
  - **group**:数据库使用的连接（也就是在 APPPATH/config/database.php 中定义的）
  - **table**:session 保存 session 数据的表的表明。默认是 session

这里是一个配置文件的示例用来为每个适配器设置可用的配置。

```
return array(
    'cookie' => array(
        'name' => 'cookie_name',
        'encrypted' => TRUE,
        'lifetime' => 43200,
```

```
        ),
        'native' => array(
            'name' => 'session_name',
            'encrypted' => TRUE,
            'lifetime' => 43200,
        ),
        'database' => array(
            'group' => 'default',
            'table' => 'table_name',
        ),
);
```

如何转换一个特殊字符为 HTML 实体

[返回目录]

Html::chars() 方法和 PHP 的 htmlspecialchars() 函数相似。但有一些小小的区别。

- 它会自动使用 UTF-8 字符集转换(替代 ISO-8859-1)
- 它会自动 翻译单引号和双引号到 HTML 实体（替代仅仅双引号）

用法

```
echo Html::chars('<p>"I\'m hungry"&mdash;Cookie Monster said.</p>');
```

将返回

```
&lt;p&gt;&quot;I&#039;m hungry&quot;&amp;mdash;Cookie Monster said.&lt;/p&gt;
```

当设置第二个参数是 FALSE ，HTML 实体会被保留，请注意 &mdash

```
echo Html::chars('<p>"I\'m hungry"&mdash;Cookie Monster said.</p>', FALSE);
```

```
&lt;p&gt;&quot;I&#039;m hungry&quot;&mdash;Cookie Monster said.&lt;/p&gt;
```

如何创建一个文本或图像链接

[返回目录]

Html::anchor() 方法能用来创建一个 HTML 锚标签来连接内部或外部的网页。

基本用法

```
echo Html::anchor('controller/action', 'My Link');
echo Html::anchor('http://kohanaphp.com', 'Kohana PHP');
```

```
<a href="http://example.com/controller/action">My Link</a>
<a href="http://kohanaphp.com">Kohana PHP</a>
```

设置属性

你能用额外的属性来设置你的链接的属性

```
echo Html::anchor('controller/action', 'My Link', array('id'=>'link_id'), 'ftp');
```

```
<a id="link_id" href="ftp://example.com/controller/action">My Link</a>
```

穿件一个图片链接

Html::anchor()类中的 title 属性不能放在外面，所以你能像图片一样插入 html

```
echo Html::anchor('controller/action', Html::image('media/img/icon.png'));
```

```
<a href="http://example.com/controller/action"><img
src="http://example.com/media/img/icon.png" /></a>
```

创建一个链接到一个定义的路由

假设你有一个叫 demo-sample 的路由定义在 bootstrap 中

```
echo Html::anchor(Route::get('demo-sample')->uri(array('category'=>'books', 'id'=>5)), 'My
link');
```

```
<a href="http://example.com/products/details/books/5">My Link</a>
```

路由能被定义在 application/bootstrap.php 文件或模块的 init.php 文件中。

```
Route::set('demo-sample', 'products(/<category>(/<id>))')
        ->defaults(array(
                'controller' => 'products',
```

```
                    'action'        => 'index',
            ));
```

如何在一个新窗口打开链接

[返回目录]

HTML 类包含了一个叫做 Html::$windowed_urls 的静态属性。设置这个属性为 true,所有的 HTTP:HTTPS:请求会自动增加一个 target=”_blank”的属性

用法

如果你想用新窗口打开你的单个链接，你只需要在你的 anchor 里增加 target 属性

```
echo Html::anchor('http://google.com','New Window', array('target'=>'_blank'));
```

如果你想用新窗口打开所有的链接，那么就在你的 application/bootstrap.php 文件中设置 windowed_urls 属性为 TRUE。注意：如果你的 $base_url 包含协议，那么 windowed_urls 将无法识别，它将无法为你工作。

```
Html::$windowed_urls = TRUE;
```

如何生成一个很难被其他人检测到的 email 地址

[返回目录]

Html::email() 方法在所有时间都会创建一个混淆代码版本的 email 地址。

用法

```
echo Html::email('jimmy@fallon.org');
```

```
&#x6a;im&#x6d;&#x79;@&#x66;&#x61;&#108;&#108;o&#x6e;.&#111;r&#x67;
```

如何创建一个 email 链接

[返回目录]

Html::mailto()方法创建了一个混淆版本的 email 地址链接

用法

```
echo Html::mailto('jimmy@fallon.org');
```

```
<a
href="&#109;&#097;&#105;&#108;&#116;&#111;&#058;&#106;im&#109;y&#64;f&#97;l&#x6c
;o&#110;.&#x6f;rg">&#106;im&#109;y&#64;f&#97;l&#x6c;o&#110;.&#x6f;rg</a>
```

如何创建一个 css 链接

[返回目录]

Html::style() 方法创建了一个 css 的链接

用法

```
echo Html::style('media/css/demo/style.css');
```

```
<link type="text/css" href="http://example.com/media/css/demo/style.css" rel="stylesheet" />
```

如果你想让样式表的 URI 中包含 Kohana 的 index 文件，Html::style()可以接受两个参数，属性和一个布尔标签来决定这些。这个将产生一个区别，如果你选择用一个 media 控制器来处理你的样式表

```
echo Html::style('media/css/demo/style.css', array('media'=>'screen, projection'), TRUE);
```

```
<link type="text/css" href="http://example.com/index.php/media/css/demo/style.css" rel="stylesheet" media="screen, projection" />
```

如何创建一个脚本链接

[返回目录]

Html::script()方法能创建一个 javascript 的链接

用法

```
echo Html::script('media/js/demo/script.js');
echo Html::script('http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js');
```

```
  <script type="text/javascript" src="http://example.com/media/js/demo/script.js"></script>
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

Html::script()类也接受一个属性数组和一个布尔标记来觉得你是否想在脚本的 URL 中包含 Kohana 的 index 文件。这将产生一个不同，如果你选择用一个 media 控制器来处理你的脚本。

如何显示一张图片

[返回目录]

Html::image()方法创建了一个 html 的 img 标签用来显示图片。

用法

```
echo Html::image('media/img/icon.png');
```

```
<img src="http://example.com/media/img/icon.png" />
```

Html::image()方法也能接受属性

```
echo Html::image('media/img/icon.png', array('class'=>'no_border'));
```

```
<img src="http://example.com/media/img/icon.png" class="no_border" />
```

如何设置属性

[返回目录]

你注意到了，Html 类和Form 类中的大多数方法将接受一个属性数组。Html::attributes()方法链接了这些属性使它们为一个字符串。

这些属性的优先顺序（不是必须）在Html 类中的attribute_order 中被定义

用法

```
echo Html::attributes(array(
    'class'=>'myclass',
    'disabled'=>TRUE,
    'style'=>'color:#fff;padding:5px',
));
```

```
class="myclass" style="color:#fff;padding:5px" disabled="1"
```

注意：属性的顺序根据优先顺序而重新组织。

如何限制一个字符串的单词数

[返回目录]

Text::limit_words()方法能限制一个断续的单词数。这个方法有 3 个参数。

1. 要被限制的字符串
2. 限制字符串中的单词数。默认 100
3. 添加到返回字符串最后的字符串。默认是 NULL

用法

```
echo Text::limit_words('This is my really cool and super awesome string.', 4, '...');
```

```
This is my really...
```

如何限制一个字符串的字符数

[返回目录]

Text::limit_chars()方法限制了一个短语的字符数。它接受 4 个参数。

1. 要被限制的字符串
2. 限制字符串中的字符数。默认 100
3. 添加到返回字符串最后的字符串。默认是 NULL
4. 一个布尔标记用来确定是否保留那个最后在中间被分隔开的单词。默认是 FALSE

用法

```
echo Text::limit_chars('This is my really cool and super awesome string.', 15, '...')
```

```
This is my real...
```

如何轮换两个或更多字符串

[返回目录]

Text::alternate()方法接受两个或更多字符串参数，并且会在它每次被调用时返回不同的字符窜

```
echo Text::alternate('You are cool','You are smart','You are funny');
echo Text::alternate('You are nice','You are happy','You are stinky');
echo Text::alternate('You are ugly','You are sleepy','You are bored');
```

```
You are cool
You are happy
You are bored
```

如何产生一个随机字符串

[返回目录]

Text::random()方法返回一个你请求的字符和长度组成的随机字符串。你可以请求的随机字符串的类型有：

- Alnum:所有字母和数字
- Alpha:只有字母，大写和小写
- hexdec:基本的 16 进制的符号(0123456789abcdef)
- numeric:仅仅数字
- nozero:除了 0 以外的其他数字
- distinct:除了 0 以外的数字以及大写字母

用法

```
echo Text::random('alnum', 8);
echo Text::random('alpha', 10);
```

```
8Sf45xHX
NAWpNXeljF
```

如何把字符串中多个反斜杠变成单个的反斜杠

[返回目录]

Text::reduce_slashes()方法把字符串中的多个反斜杠变成单个反斜杠

用法

```
echo Text::reduce_slashes('This //Is my string// With //Slashes');
```

```
This /Is my string/ With /Slashes
```

如何从一个字符串中过滤掉特定的词语

[返回目录]

 Text::censor()方法会字符串中过滤掉特定单词。要过滤的词作为一个数组参数传递。词语可以被整个或者局部替换成你在配置中设置的字符串。

用法

创建一个你定义的敏感词配置文件

```
/home/kerkness/kohana/application/config/censor.php
```

```
return array(
        'words' => array(
                'Codeigniter',
                'Zend',
                'CakePHP',
                'Yii',
        ),
);
```

使用 censor()方法从一个字符串中过滤掉这些词语

```
echo Text::censor('I used CakePHP and Zend but I likeYiis', Kohana::config('censor.words'));
```

```
I used ####### and #### but I like###s
```

如果你想定义用什么单词去替换，你可以设置第三个参数

```
echo Text::censor('I used CakePHP and Zend but I likeYiis', Kohana::config('censor.words'), 'Kohana');
```

```
I used Kohana and Kohana but I likeKohanas
```

将第四个参数设置为 FALSE，则可以取代替换局部单词的方法。

```
echo Text::censor('I used CakePHP and Zend but I likeYiis', Kohana::config('censor.words'), '*', FALSE)
```

```
I used ******* and **** but I likeYiis
```

如何查找相似词语

[返回目录]

Text::similar()方法查找出一个字符串数组中相同的部分。它会用字符串数组中第一字符串来作为基础去搜索。

用法

```
echo Text::similar( array('Help', 'Helium', 'Herman') );
```

```
He
```

如何自动把 URL 转成链接

[返回目录]

Kohana 的 Text 类有 3 个方法来将 URL 转换成链接

- Text::auto_link()将 email 和 URL 转换成链接
- Text::auto_link_urls()仅将 URL 转换成链接
- Text::auto_link_emails()仅将 email 转换成链接

用法

注意，email 地址会被混淆，使他们不易被读。

```
echo Text::auto_link('My email is someone@somewhere.ca and my web site is www.kohanaphp.com');
```

```
My email is <a href="&#109;&#097;...&#x61;">&#115;&#x6f;..e&#46;&#99;&#x61;</a> and my web site is <a href="http://www.kohanaphp.com">www.kohanaphp.com</a>
```

```
echo Text::auto_link_urls('My email is someone@somewhere.ca and my web site is www.kohanaphp.com');
```

```
My email is <a href="&#109;&#097;...&#x61;">&#115;&#x6f;..e&#46;&#99;&#x61;</a> and my web site is www.kohanaphp.com
```

如何自动为文本块增加段落标签

[返回目录]

Html::auto_p()方法类似于 nl2br()函数。它会给一块文本增加
和<p>标签

用法

```
$string = "This is a block of
text that has

Some line breaks in it.";

echo Text::auto_p($string);
```

```
<p>This is a block of<br>
text that has</p>

<p>Some line breaks in it.</p>
```

警告

Text::auto_p()离完美还有距离。它用正则表达式解析 HTML。在这个函数中 HTML 注释，属性都包含 < 或者新行。而这些都会被暴露出来。

如何把字节格式转换成人们常用的格式

[返回目录]

Html::bytes() 方法会将字节表示转换成人类方便阅读的格式。这个方法基于Aidan Lister 和Quentin Zervaas 最初的函数

简单用法

```
echo Text::bytes(5500);
echo Text::bytes(17139812000);
```

```
5.50 kB
17.14 GB
```

设置最大单位

```
echo Text::bytes(81620000000, 'MB');
```

```
81620.00 MB
```

设置精确度为 4 位小数

```
echo Text::bytes(91711816100, null, '%01.4f %s', TRUE);
```

```
91.7118 GB
```

设置精确度为 1 位小数，单位在括号中，最大单位是 MB

```
$size = disk_total_space('/home');
echo Text::bytes($size, 'MB', '%01.1f (%s)', TRUE);
```

```
64585.9 (MB)
```

如何获得你网站的基本 Url

[返回目录]

Url::base() 方法提供你网站的基本 url。基本 url 就是你的 Kohana 的 index.php 文件的完整域名和路径。

基本 url 可以在你的 application/bootstrap.php 文件中调用Kohana::init()方法设置 base_url 属性来定义

注意：如果你的 base_url 被定义成一个诸如**http://kerkness.ca** 的完整域名。
那么Url::base()方法将始终使用完全合格的域名来反应。

使用

```
echo Url::base();
```

```
/kohana/
```

如果你想在你的基本 url 中包含 index.php。那么设置第一个属性为 true

```
echo Url::base(TRUE);
```

```
/kohana/index.php/
```

如果你没有在 Kohana::init 中你的 base_url 中定义协议，但是你又想生成协议，设置第二个属性为 TRUE

```
echo Url::base(FALSE, TRUE);
```

```
http://kerkness.ca/kohana/
```

如果你想为你的 url 设置一个特殊的协议，那么你就设置第二个属性为这个协议的字符串。

```
echo Url::base(FALSE, 'ftp');
```

```
ftp://kerkness.ca/kohana/
```

上面的示例假定你在 application/bootstrap.php 中使用了下列这样设置

```
Kohana::init(array(
        'base_url'   => '/kohana/',
        'index_file' => 'index.php',
));
```

如何生成网站 Url

[返回目录]

Url::site()方法允许你为你的网站生成一个完全合格的 url.

用法

```
echo Url::site('controller/action');
```

```
/kohana/index.php/controller/action
```

如果你需要使用一个完全合格的域名，那么设置第二个参数是 TRUE

```
echo url::site('controller/action', TRUE)
```

```
http://kerkness.ca/kohana/index.php/controller/action
```

如果你要定义你的 url 的协议，你可以设置第二个属性为这个协议的字符串

```
echo Url::site('controller/action', 'ftp');
```

```
ftp://kerkness.ca/kohana/index.php/controller/action
```

上面的示例假定你在 application/bootstrap.php 中使用了下列这样设置

```
Kohana::init(array(
        'base_url'    => '/kohana/',
        'index_file' => 'index.php',
));
```

如何生成查询字符串

[返回目录]

Url::query()方法将当前的_GET 数组所提供的参数合并成一个字符串

用法

```
// 如果你的当前的 url 是 http://kerkness.ca/kohana?foo=bar
echo Url::query();
```

```
?foo=bar
```

```
echo Url::query(array('bar'=>'foo'));
```

```
?foo=bar&bar=foo
```

如何生成友好的 Url 标题字符串

[返回目录]

Url::title()方法生成一个 url 友好的字符串。注意：非 ASCII 字符使用之前应该先使用这个函数进行转义

用法

```
echo url::title('This is my title')
```

```
this-is-my-title
```

如何打开和关闭一个表单

[返回目录]

Form::open()和Form::close()方法可以用来创建表单的打开和关闭标签

用法

```
echo Form::open();
echo Form::close();
```

```
<form action="/controller/action" method="post" accept-charset="utf-8">
</form>
```

当调用Form::open()时，你需要声明一个特定的动作，并且提供一个属性数组

```
echo Form::open('auth/login', array('id'=>'myform', 'class'=>'myclass'));
echo Form::close();
```

```
<form action="/auth/login" method="post" id="myform" accept-charset="utf-8"
class="myclass">
</form>
```

如何创建一个 input 域

[返回目录]

Form::input()方法能用来创建几乎所有类型的<input>标签。默认是创建一个 Text 的 input ,除非 type 属性被设置。

```
echo Form::input('name');
```

```
<input type="text" name="name" />
```

增加第二个参数来设置一个 value

```
echo Form::input('name', 'value');
```

```
<input type="text" name="name" value="value" />
```

第三个参数接受一个属性数组

```
echo Form::input('name', 'value', array('disabled'=>TRUE));
```

```
<input type="text" name="name" value="value" disabled="1" />
```

如何创建一个隐藏域

[返回目录]

Form::hidden()方法和 Form::input()能一样好的工作，除了它默认的 type 属性自动设置为了 hidden

```
echo Form::hidden('name');
```

```
<input type="hidden" name="name" />
```

增加第二个参数来声明 value

```
echo Form::hidden('name', 'value');
```

```
<input type="hidden" name="name" value="value" />
```

第三个参数接受一个属性数组

```
echo Form::hidden('name', 'value', array('id'=>'name'));
```

```
<input type="hidden" name="name" value="value" id="name" />
```

如何创建一个密码域

[返回目录]

Form::password()方法和 Form::input()能一样好的工作。除了它默认的 type 属性自动设置为了 password

```
echo Form::password('name');
```

```
<input type="password" name="name" />
```

增加第二个参数来声明 value

```
echo Form::password('name', 'value');
```

```
<input type="password" name="name" value="value" />
```

第三个参数接受一个属性数组

```
echo Form::password('name', 'value', array('disabled'=>TRUE));
```

```
<input type="password" name="name" value="value" disabled="1" />
```

如何创建一个文件上传域

[返回目录]

Form::file()方法和 Form::input()能一样好的工作。除了它默认的 type 属性自动设置为了 file 并且它不能接受 value 参数

```
echo Form::file('name');
```

```
<input type="file" name="name" />
```

第 2 个参数接受一个属性数组

```
echo Form::file('name', array('disabled'=>TRUE));
```

```
<input type="file" name="name" disabled="1" />
```

如何创建一个多选框

[返回目录]

Form::checkbox()方法和 Form::input()能一样好的工作。除了它默认的 type 属性自动设置为了 checkbox 并且有一个附加的属性用来定义框是否被选中。

```
echo Form::checkbox('name', 'yes', TRUE);
```

```
<input type="checkbox" name="name" value="yes" checked="1" />
```

第 4 个参数接受一个属性数组

```
echo Form::checkbox('name', 'yes', FALSE, array('disabled'=>TRUE));
```

```
<input type="checkbox" name="name" value="yes" checked="1" disabled="1" />
```

如何创建一个单选框

[返回目录]

Form::radio()方法和 Form::input()能一样好的工作。除了它默认的 type 属性自动设置为了 radio。该方法仅仅创建了一个单一的单选按钮所以如果要创建一组按钮，只需要调用多次同样的 name 即可。

```
echo Form::radio('name', 'yes', TRUE);
echo Form::radio('name', 'no', FALSE);
```

```
<input type="radio" name="name" value="yes" checked="checked" />
<input type="radio" name="name" value="no" />
```

第 4 个参数接受一个属性数组

```
echo Form::radio('name', 'yes', TRUE, array('disabled'=>TRUE));
```

```
<input type="radio" name="name" value="yes" checked="checked" disabled="1" />
```

如何创建一个文本区域

[返回目录]

Form::textarea() 方法和 Form::input()能一样好的工作。除了它第 4 个参数允许决定是否存在编码的 HTML 实体。默认转换任何东西。

```
echo Form::textarea('name');
```

```
<textarea name="name"></textarea>
```

增加第二个参数来声明 value

```
echo Form::textarea('name', 'This is my body');
```

```
<textarea name="name">This is my body</textarea>
```

第 3 个参数接受一个属性数组

```
echo Form::textarea('name', 'This is my body', array('disabled'=>FALSE));
```

```
<textarea name="name" disabled="1">This is my body</textarea>
```

如果你的文本域包含 HTML 实体，那么设置第 4 个参数为 FALSE 来避免两次编码

```
// &amp;将被两次编码
echo Form::textarea('name', 'You &amp; Me', NULL, TRUE);
```

```
<textarea name="name">You &amp;amp; Me</textarea>
```

```
// &amp; 不会被两次编码
echo Form::textarea('name', 'You &amp; Me', NULL, FALSE);
```

```
<textarea name="name">You &amp; Me</textarea>
```

如何创建一个选择域和下拉菜单

[返回目录]

Form::select() 方法能用来创建选择域或下拉菜单。它包含了一些定义选项和设置哪个选项被选中的参数。

```
echo Form::select('name', array(
        'standard'=>'Standard Shipping',
        'express'=>'Express Shipping',
        'international'=>'International',
));
```

```
<select name="name">
<option value="standard">Standard Shipping</option>
<option value="express">Express Shipping</option>
<option value="international">International</option>
</select>
```

增加第三个参数来定义哪个选项被选中

```
echo Form::select('name', array(
        'standard'=>'Standard Shipping',
        'express'=>'Express Shipping',
        'international'=>'International',
), 'express');
```

```
<select name="name">
<option value="standard">Standard Shipping</option>
<option value="express" selected="selected">Express Shipping</option>
<option value="international">International</option>
</select>
```

第 4 个参数接受一个属性数组

```
echo Form::select('name', array(
        'standard'=>'Standard Shipping',
        'express'=>'Express Shipping',
        'international'=>'International',
), 'express', array('disabled'=>TRUE));
```

```
<select name="name" disabled="1">
```

```
<option value="standard">Standard Shipping</option>
<option value="express" selected="selected">Express Shipping</option>
<option value="international">International</option>
</select>
```

配置组

数组配置（第 2 个参数），键定义了一个配置组的标签，值就是配置组成的数组

```
echo Form::select('product', array(
        'Fruits' => array(
                'apple' => 'Apple',
                'orange' => 'Orange',
                'plum' => 'Plum'
            ),
        'Vegetables' => array(
                'lettuce' => 'Lettuce',
                'parsley' => 'Parsley',
                'cucumber' => 'Cucumber'
            )
    ));
```

...过程...

```
<select name="product">
    <optgroup label="Fruits">
        <option value="apple">Apple</option>
        <option value="orange">Orange</option>
        <option value="plum">Plum</option>
    </optgroup>
    <optgroup label="Vegetables">
        <option value="lettuce">Lettuce</option>
        <option value="parsley">Parsley</option>
        <option value="cucumber">Cucumber</option>
    </optgroup>
</select>
```

如何创建一个表单按钮

[返回目录]

使用 Form::submit()和Form::button()方法可以创建表单按钮。

Form::submit()用来创建一个表单的 submit 按钮

```
echo Form::submit('name', 'Submit');
```

```
<input type="submit" name="name" value="Submit" />
```

第 3 个参数接受一个属性数组

```
echo Form::submit('name', 'Submit', array('disabled'=>TRUE));
```

```
<input type="submit" name="name" value="Submit" disabled="1" />
```

Form::button()用<button>标签代替<input>标签来生成按钮

```
echo Form::button('name', 'Button');
```

```
<button name="name">Button</button>
```

第 3 个参数接受一个属性数组

```
echo Form::button('name', 'Button', array('disabled'=>TRUE));
```

```
<button name="name" disabled="1">Button</button>
```

注意允许使用图片或者其他 HTML 并不会脱离按钮内。

```
echo Form::button('name', Html::image('media/icon.png'));
```

```
<button name="name"><img src="/media/icon.png"/></button>
```

如何创建表单标签

[返回目录]

Form::label()方法能用来创建表单标签

```
echo Form::label('field', 'Field Name');
```

```
<label for="name">Field Name</label>
```

第 3 个参数接受一个属性数组

```
echo Form::label('name', 'Field Name', array('class'=>'error'));
```

```
<label for="name" class="error">Field Name</label>
```

表单验证

[返回目录]

验证类被按下面给出的规则用来验证任何数据的数组。它主要用来验证表单的$_POST 数据。类中有一些规则已经为我们验证了并且允许我们使用回调函数。该类允许你的每个数组域设置自己的错误信息，那么你就能构建你自己的表单了。

首先是过滤器处理的字段，然后是规则，最后是回调函数。

开始

加载库

```
// 创建一个新的验证（Vaildate）类的对象来验证$_POST 变量
$post = new Validate($_POST);

// 合并不同的数组
$post = new Validate(array_merge($_POST, $_FILES));

// 使用工厂方法能联合操作
$post = Validate::factory($_POST)->rule('field_name', 'not_empty')
                                 ->rule('field_name2', 'email');

// 增加一个过滤规则的数组
$rules = array(

                'field_name' => array
                (
                    'not_empty'     => NULL,
                    'max_length'    => array(32),
                ),

                'field_name2' => array
                (
                    'min_length'    => array(4),
                ),

            );
$post = Validate::factory($_POST)->rules('field_name', $rules['field_name'])
                                 ->rules('field_name2', $rules['field_name2']);
```

```
// 你也可以直接使用$_POST 数组(不推荐)
$_POST = new Validate($_POST);
```

增加规则（必须）

验证(Validate)对象创建好之后，你需要给你增加一些规则。他们中的一些用类本身定义。

```
$post = new Validate($_POST);
$post->rule('username', 'not_empty')
      ->rule('password', 'not_empty')
      ->rule('password', 'min_length', array(5))
      ->rule('password', 'max_length', array(42));


$post = Validate::factory($_POST)->rule('username', 'not_empty')
                                 ->rules(
                                        'password' => array(
                                                        'min_length' => array(5),
                                                        'max_length' => array(42),
                                                    )
                                    );
```

规则被回调到函数并和该函数使用一样，但是它们返回一个布尔值，所以他们能验证数据是否通过。我们可以使用我们预定义的类，php，以及我们自己的规则。

预定义函数：

```
$post = Validate::factory($_POST)->rule('age', 'numeric');
```

PHP 函数：

```
$post = Validate::factory($_POST)->rule('age', 'is_numeric');
```

用户定义的函数：

```
$post = Validate::factory($_POST)->rule('age', 'Model_User::check_numeric');

        public static function check_numeric($str)
        {
                // 获得当前本地设置的允许小数点
                list($decimal) = array_values(localeconv());

                return (bool) preg_match('/^-?[0-9'.$decimal.']++$/D', (string) $str);
        }
```

所有字段增加一个规则

同样的规则可以一次性被定义在所有的字段，只需要用 TRUE 来作为字段名

```
// 应用 not_empty 规则到$_POST 所有字段
$post = Validate::factory($_POST)->rule(TRUE, 'not_empty');
```

增加过滤器（可选）

过滤器在验证之前处理。你可以在第一个参数使用 TRUE 来让过滤器应用在所有字段。验证过滤器是任何可以返回字符串的 PHP 函数。

你也可以使用可选的第三个参数来传递一个参数组成的数组给函数。你使用的函数被调用时必须返回值到第一个参数。

注意 Kohana3.x 版本不再有 pre_filter 和 post_filter()。这就意味着你将使用 filter()来预过滤，如果需要，为 post 过滤回调。

你可以为所有字段增加一个过滤器，只要在字段名处使用 TRUE

```
// 去掉 username 的空白
$post = Validate::factory($_POST)->filter('username', 'trim');

// 传递参数到过滤器函数.
// will call: htmlspecialchars($_POST['username'], ENT_QUOTES)
$post = Validate::factory($_POST)->filter('username', 'htmlspecialchars', array(ENT_QUOTES));

// 为所有字段使用一个静态定义的函数作为过滤器
$post = Validate::factory($_POST)->filter(TRUE, 'Model_User::myfilter', array($param1, $param2));
    class Model_User extends Model_Auth_User {
        public static function myfilter($value, $param1, $param2)
        {
            $value = ... //some code// ...

            return $value;
        }
    }


// 使用 filters()来定义一个过滤器数组
$post = Validate::factory($_POST)->filters(
                                            'username' => array(
                                                    'Model_User::myfilter' =>
array($param1, $param2),
                                                    'trim' => NULL,
                                                )
                                        );
```

增加回调（可选）

不同的规则，回调函数可以在字段上使用一些更复杂的验证。如果必须需要在验证对象中增加错误，那么就作为第一个参数来传递。
你可以为所有字段增加一个过滤器，只要在字段名处使用 TRUE

调用 callback()方法：

用户定义的函数

```
// 调用用户定义的函数 reserved_username
$post = Validate::factory($_POST)->callback( 'username', 'reserved_username');
```

用静态方法作为回调函数

```
$post = Validate::factory($_POST)->callback('username', 'Model_User::mystatic_callback');
```

用对象的方法作为回调函数

```
$post = Validate::factory($_POST)->callback( 'username', array($this, 'username_available'));
```

用 callbacks()回调数组

```
$post = Validate::factory($_POST)->callbacks( 'username', array(
                                                    'reserved_username'

'Model_User::mystatic_callback',
                                                     array($this,
'username_available'),
                                                )
                                            );
```

示例：

```
// 使用 email_change()回调函数来验证我们的$_POST 数据。
// 检查用户是否能可以改变它的 email 地址当该邮件地址现在还没有其他用户使用.
$post = Validate::factory($_POST)->callback('email', array($this, 'email_change'));

        /**
         * 检验用户是否能改变 email 变成这样
         *
         * @param      Validate   $array    validate object
         * @param      string     $field    field name
         */
```

```
        public function email_change(Validate $array, $field)
        {
                $exists = (bool) DB::select(array('COUNT("*")', 'total_count'))
                                                        ->from($this->_table_name)
                                                        ->where('email',     '=',
$array[$field])
                                                        ->where('id',        '!=',
$this->id)
                                                        ->execute($this->_db)
                                                        ->get('total_count');

                if ($exists)
                        $array->error($field, 'email_change', array($array[$field]));
        }
```

增加错误

你可以用 error()方法增加错误

```
error($field, $error, array $params = NULL)
```

示例：

```
$post->error('username', 'username_available', array('johndoe'));
```

定义错误信息

可以在 messages 目录里找到国际化的文件。这些目录可以在 system,application,modules 目录中被发现。Kohana 自己的信息文件在 system 目录中。
当错误键在数组中没有被找到的时候，default 键将会被使用。
例如：application/messages/register.php

```
<?php defined('SYSPATH') OR die('No direct access allowed.');

 return array
(
        'email' => array(
                'email_available'       => 'The email address already exists',
            'default'    => 'Invalid Input.',
        ),

        'username' => array(
                'username_available'       => 'The username already exists',
            'default'    => 'Invalid Input.',
        ),

); // application/messages/register.php 的结尾
```

如果错误信息的值在你的 i18n 目录中存在，那么错误信息会自动被翻译。查看翻译信息。

检索错误信息

可以用 errors()方法来检索错误信息。默认返回的数组，包含字段名组成的键(key)，规则组成的值(value)
检索自定义的错误信息，必须传递一个错误信息文件到 errors()方法。

```
$errors = $validation->errors();
```

假定定义了一个规则，规则(‘username’,’ usernam_available’) $errors 数组
包含：

```
array(
        ('username' => 'username_available')
      )
```

使用错误信息文件取得错误：

```
$errors = $validation->errors('register')
```

假定你的信息目录中存在一个 register.php 文件并且内容如上面写的。那么 $errors 将包含：

```
array(
        ('username' => 'The username already exists')
      )
```

检索输入的数据

经过 as_array()来验证插入的数据是否可进入。这对于重新填写表单字段非常有用，例如：

```
$_POST = array_intersect_key( $post->as_array(), $_POST);
```

规则

验证(Validate)类中的特定规则

规则	参数	描述	示例
not_empty	无	如果表单字段为空，返回 FALSE	
min_length	有	如果字段太短，返回 FALSE	**length[5] -** 最少 5 个字符长
max_length	有	如果字段太长，返回 FALSE	**length[30]**- 最多 30 个字符长
exact_length	有	如果字段太长或太短，返回 FALSE	**length[25]**- 仅可以 25 个字符
matches	有	如果字段不匹配参数，返回 FALSE	**matches[再一次密码]**
date	无	如果字段非有效日期，返回 FALSE	
regex	有	如果字段不符合正则表达式，返回 FALSE	**regex[表达式]** – 正则表达式来匹配（包括分隔符）
email	可选	如果 email 无效，返回 FALSE	**mail[TRUE]** – 严格的 rfc822
email_domain	无	如果 email 域名没有有效地 MX 记	

		录，返回 FALSE	
url	无	如果 url 无效，返回 FALSE	
ip	可选	如果 ip 无效，返回 FALSE	**ip[TRUE]** – 允许私有 ip 网络
credit_card	有	如果信用卡无效，返回 FALSE	**credit_card[万事达卡]** – 卡类型或者卡类型的数组
phone	可选	如果电话号码不是有效长度，返回 FALSE	**phone[7,10,11,14]** – 7，10,11 或 14 中的一个长度(默认 7,10 和 11)
alpha	可选	如果字段不是只有字母组成，返回 FALSE	**alpha[TRUE]** – 触发 UTF-8 兼容性
alpha_numeric	可选	如果字段不是只有字母或者数字组成，返回 FALSE	**alpha_numeric[TRUE]** – 触发 UTF-8 兼容性
alpha_dash	可选	如果字段不是只有字母，数字，下划线和破折号组成，返回 FALSE	**alpha_dash[TRUE]** – 触发 UTF-8 兼容性
digit	可选	如果字段不是只有数字字符（无点号或横线），返回 FALSE	**digit[TRUE]** – 触发 UTF-8 兼容
numeric	无	如果字段是一个无效的数字（正负或小数），返回 FALSE	
decimal	可选	如果字段不是完全的小数格式，可选参数是个特定的小数格式，则返回 FALSE	**decimal** – 是任何有效的小数格式 **decimal[4,2]** 是 4 个整数和 2 个小数
range	有	如果字段不是随机在最大和最小范围之内，则返回 FALSE	range[1,10] – 在 1 和 10 之间
color	无	如果字段不是一个适当的 16 进制 HTML 颜色值，则返回 FALSE	

示例

创建和验证表单

```
//验证一个用户注册
        function action_register()
        {
                #例举一个新用户
                $user = ORM::factory('user');

                #加载验证规则，过滤器和回调函数
                $post = $user->validate_create($_POST);

            #验证所有的字段的有效性
                if ($post->check())
                {
```

```
                                        #影响并清理 user 对象的变量
                                        $user->values($post);

                                        #创建账户
                                        $user->save();

                                        #为用户添加登录角色
                                        $login_role = new Model_Role(array('name'
=>'login'));
                                        $user->add('roles',$login_role);

                                        #标记用户进入
                                        Auth::instance()->login($post['username'],
$post['password']);

                                        #重定向用户账号
                                        Request::instance()->redirect('myaccount');
                                }
                                else
                                {
                                  #重新注入$_POST 数据
                                  $_POST = array_intersect_key( $post->as_array(), $_POST);

                                  #影响所要进一步显示的错误
                                        $this->errors = $post->errors('register');
                                }
                }
```

我们的 Model_User

```
class Model_User extends Model_Auth_User {
                protected $_rules = array
                (
                        'username'                              => array
                        (
                                'not_empty'                     => NULL,
                                'min_length'                    => array(4),
                                'max_length'                    => array(32),
                                'regex'                         =>
array('/^[-\pL\pN_.]++$/uD'),
                        ),
                        'password'                      => array
                        (
                                'not_empty'                     => NULL,
```

```
                        'min_length'                    => array(5),
                        'max_length'                    => array(42),
                ),
                'password_confirm'    => array
                (
                        'matches'               => array('password'),
                ),
                'email'                                 => array
                (
                        'not_empty'                     => NULL,
                        'min_length'                    => array(4),
                        'max_length'                    => array(127),
                        'validate::email'       => NULL,
                ),
        );

        protected $_callbacks = array
        (
                'username'                              => array('username_available'),
                'email'                                         =>
array('email_available'),
        );

        public function validate_create(& $array)
        {
                // 初始化验证库并设置一些规则
                $array = Validate::factory($array)
                                                        ->rules('password',
$this->_rules['password'])
                                                        ->rules('username',
$this->_rules['username'])
                                                        ->rules('email',
$this->_rules['email'])
                                                        ->rules('password_confirm',
$this->_rules['password_confirm'])
                                                        ->filter('username', 'trim')
                                                        ->filter('email', 'trim')
                                                        ->filter('password', 'trim')
                                                        ->filter('password_confirm',
'trim');


                #增加 Model_Auth_User 回调函数
                foreach ($this->_callbacks as $field => $callbacks)
```

```
                {
                        foreach ($callbacks as $callback){
                                $array->callback($field, array($this, $callback));
                        }
                }

                return $array;
        }


        public function validate_change(& $array, $save = FALSE){
                // 初始化验证哭并设置一些规则
                $array = Validate::factory($array)->rules('email', $this->_rules['email'])
                                                  ->filter('email', 'trim')
                                                          ->filter('password', 'trim')
                                                  ->callback('email', array($this,
'email_change'));

                if(trim($array['password']) != '')
                        $array->rules('password', array('min_length'=> array(5),
'max_length'=>array(42)));


                return $array;
        }


        /**
         * 验证用户是否改变了 email 成为这个
         *
         * @param      Validate    $array     validate object
         * @param      string      $field     field name
         */
        public function email_change(Validate $array, $field)
        {
                $exists = (bool) DB::select(array('COUNT("*")', 'total_count'))
                                                        ->from($this->_table_name)
                                                        ->where('email',     '=',
$array[$field])
                                                        ->where('id',        '!=',
$this->id)
                                                        ->execute($this->_db)
                                                        ->get('total_count');
```

```
                if ($exists)
                        $array->error($field, 'email_change', array($array[$field]));
}

/**
 * 如果用户名存在于验证规则的 unique_key_exists()失败是否触发错误
 *
 * @param      Validate   $array    validate object
 * @param      string     $field    field name
 * @param      array      $errors   current validation errors
 * @return     array
 */
public function username_available(Validate $array, $field)
{
                if ($this->unique_key_exists($array[$field])) {
                        $array->error($field, 'username_available', array($array[$field]));
                }
}

/**
 * 如果 email 存在于验证规则的 unique_key_exists()失败，是否触发错误
 *
 * @param      Validate   $array    validate object
 * @param      string     $field    field name
 * @param      array      $errors   current validation errors
 * @return     array
 */
public function email_available(Validate $array, $field)
{
                if ($this->unique_key_exists($array[$field])) {
                        $array->error($field, 'email_available', array($array[$field]));
                }
}


/**
 * 测试是否是存在于数据库中唯一的键
 *
 * @param     mixed            value   the value to test
 * @return    boolean
 */
public function unique_key_exists($value)
{
                return (bool) DB::select(array('COUNT("*")', 'total_count'))
```

```
                                                        ->from($this->_table_name)

            ->where($this->unique_key($value), '=', $value)
                                                        ->execute($this->_db)
                                                        ->get('total_count');
            }

            /**
             * 允许一个模型使用 email 和 username 均为唯一标识符的情况下登录
             *
             * @param    string      $value     unique value
             * @return string                   field name
             */
            public function unique_key($value)
            {
                        return Validate::email($value) ? 'email' : 'username';
            }
}
```

验证文件上传

这里是一个验证文件上传的示例

```
                        $validate = Validate::factory($_FILES);
                        $validate->rules('file',
                                    array('Upload::valid' => array(),
                                        'Upload::not_empty' => array(),
                                          'Upload::type' =>array('Upload::type' =>
array('jpg','png','gif')),
                                          'Upload::size' => array('1M'))
                        );


                        if ($validate->check())
                        {
                                    //成功
                                    Upload::save($_FILES['file'],'my_image.png','./',777);
                        }
                        else
                        {
                                    //错误
                                    $this->errors = $validate->errors('upload');
                                    print_r($this->errors);
                        }
```

application/messages/upload.php

```
return array
(
        'file' => array(
        'Upload::valid'         => 'valid msg',
        'Upload::not_empty'             => 'not_empty msg',
        'Upload::type'          => 'type msg',
        'Upload::size'          => 'size msg',
        'default'               => 'default msg'),
);
```

确定时区间的偏移（秒）

[返回目录]

Date::offset()方法会返回 2 个不同时区间所相差的全部秒数。如果你仅仅提供一个时区，那么它将与你本地时区相比较。

用法

```
echo Date::offset('America/New_York');
```

```
3600
```

```
echo Date::offset('Antarctica/Casey', 'America/New_York');
```

```
46800
```

PHP5 中支持的所有时区的列表 点击这里

获得一天，小时，分钟所包含的秒，分，小时

[返回目录]

Date::seconds()方法将会返回一个数组，该数组包括了 1 分钟内的秒数递增集合。

用法

```
Date::seconds();
```

```
Array ( [0] => 00 [1] => 01 [2] => 02 [3] => 03 [4] => 04 [5] => 05 [6] => 06 [7] => 07 [8] => 08 [9] => 09 [10] => 10 [11] => 11 [12] => 12 [13] => 13 [14] => 14 [15] => 15 [16] => 16 [17] => 17 [18] => 18 [19] => 19 [20] => 20 [21] => 21 [22] => 22 [23] => 23 [24] => 24 [25] => 25 [26] => 26 [27] => 27 [28] => 28 [29] => 29 [30] => 30 [31] => 31 [32] => 32 [33] => 33 [34] => 34 [35] => 35 [36] => 36 [37] => 37 [38] => 38 [39] => 39 [40] => 40 [41] => 41 [42] => 42 [43] => 43 [44] => 44 [45] => 45 [46] => 46 [47] => 47 [48] => 48 [49] => 49 [50] => 50 [51] => 51 [52] => 52 [53] => 53 [54] => 54 [55] => 55 [56] => 56 [57] => 57 [58] => 58 [59] => 59 )
```

你也可以随意的设置起点和终点值以及一个递增的步进。

```
Date::seconds(2, 10, 30);
```

```
Array ( [10] => 10 [12] => 12 [14] => 14 [16] => 16 [18] => 18 [20] => 20 [22] => 22 [24] => 24 [26] => 26 [28] => 28 )
```

获得一小时内的分钟数量

Date::minutes()方法会返回一个数组，该数组包括了 1 小时内的分钟的递增集合。Date::minutes()使用的默认步进式 5

```
Date::minutes();
```

```
Array ( [0] => 00 [5] => 05 [10] => 10 [15] => 15 [20] => 20 [25] => 25 [30] => 30 [35] => 35 [40] => 40 [45] => 45 [50] => 50 [55] => 55 )
```

```
Date::minutes(10);
```

```
Array ( [0] => 00 [10] => 10 [20] => 20 [30] => 30 [40] => 40 [50] => 50 )
```

获得一天内的小时数量

Date::hours()方法会返回一个数组，该数组包括了 1 天内的小时的递增集合。

```
Date::hours()
```

```
Array ( [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8 [9] => 9 [10] => 10 [11] => 11 [12] => 12 )
```

设置第 2 个元素为 TRUE，则是 24 小时

```
Date::hours(1, TRUE);
```

```
Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8 [9] => 9 [10] => 10 [11] => 11 [12] => 12 [13] => 13 [14] => 14 [15] => 15 [16] => 16 [17] => 17 [18] => 18 [19] => 19 [20] => 20 [21] => 21 [22] => 22 [23] => 23 )
```

变换步进以及设置一个开始的小时

```
Date::hours(2, TRUE, 10);
```

```
Array ( [10] => 10 [12] => 12 [14] => 14 [16] => 16 [18] => 18 [20] => 20 [22] => 22 )
```

获得所给时间处于上午还是下午

[返回目录]

Date::ampm()方法会按你给的小时返回一个 AM 或者 PM。只能用在 24 小时。

用法

```
echo Date::ampm(8);
```

```
AM
```

```
echo Date::ampm(11);
```

```
AM
```

```
echo Date::ampm(12);
```

```
PM
```

```
echo Date::ampm(18);
```

```
PM
```

转换一个非 24 小时数字为 24 小时数字

[返回目录]

Date::adjust()方法调整了一个非 24 小时数字为 24 小时数字。

用法

```
echo Date::adjust(8, 'PM');
```

```
20
```

```
echo Date::adjust(12, 'PM');
```

```
12
```

```
echo Date::adjust(9, 'PM');
```

```
21
```

获得一个月有几天

[返回目录]

Date::days()方法会返回一个数组，该数组包括了你给的月内的天的递增集合。

用法

```
echo Date::days(5);
```

```
Array ( [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8 [9] => 9 [10] => 10 [11] => 11 [12] => 12 [13] => 13 [14] => 14 [15] => 15 [16] => 16 [17] => 17 [18] => 18 [19] => 19 [20] => 20 [21] => 21 [22] => 22 [23] => 23 [24] => 24 [25] => 25 [26] => 26 [27] => 27 [28] => 28 [29] => 29 [30] => 30 [31] => 31 )
```

你可以设置年

```
echo Date::days(2, 2010);
```

```
Array ( [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8 [9] => 9 [10] => 10 [11] => 11 [12] => 12 [13] => 13 [14] => 14 [15] => 15 [16] => 16 [17] => 17 [18] => 18 [19] => 19 [20] => 20 [21] => 21 [22] => 22 [23] => 23 [24] => 24 [25] => 25 [26] => 26 [27] => 27 [28] => 28 )
```

获得一年有几个月

[返回目录]

Date::months()方法会返回一个数组，该数组包括了 1 年内的月的递增集合。

用法

```
echo Date::months();
```

```
Array ( [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8 [9] => 9 [10] => 10 [11] => 11 [12] => 12 )
```

获得起始年到终止年中的年份，并转换为数组

[返回目录]

 Date::years()方法返回起始年和终止年中的年份组成的数组。使用当前年 +/-5 就是当前年的 最大/最小

用法

```
echo Date::years();
```

```
Array ( [2004] => 2004 [2005] => 2005 [2006] => 2006 [2007] => 2007 [2008] => 2008 [2009] => 2009 [2010] => 2010 [2011] => 2011 [2012] => 2012 [2013] => 2013 [2014] => 2014 )
```

```
echo Date::years(1975, 1981);
```

```
Array ( [1975] => 1975 [1976] => 1976 [1977] => 1977 [1978] => 1978 [1979] => 1979 [1980] => 1980 [1981] => 1981 )
```

获得两个时间戳之间的时差

[返回目录]

Date::span()方法用人类可读格式返回两个时间戳之间的时差

用法

```
echo Date::span(time(), time()+3600, 'hours');
```

```
1
```

```
Date::span(159084000, 227512800);
```

```
Array ( [years] => 2 [months] => 2 [weeks] => 0 [days] => 0 [hours] => 15 [minutes] => 24 [seconds] => 20 )
```

```
Date::span(159084000, 227512800, 'months,days');
```

```
Array ( [months] => 26 [days] => 0 )
```

获得所给出时间和现在的差异

[返回目录]

Date::fuzzy_span()方法用一个失真的方式返回所给出时间和现在时间之间的差异

用法

```
Date::fuzzy_span(159084000);
```

```
several decades ago
```

```
Date::fuzzy_span(1258264800);
```

```
less than a month ago
```

```
Date::fuzzy_span(1261720800);
```

```
in less than a month
```

转换 UNIX 和 DOS 的时间戳

[返回目录]

Date::unix2dos()方法将 UNIX 格式的时间戳转换为 DOS 格式的时间戳

用法

```
Date::unix2dos(159084000);
```

```
2162688
```

转换 DOS 时间戳到 UNIX 格式

Date::dos2unix()方法将 DOS 格式的时间戳转换为 UNIX 格式的时间戳

用法

```
Date::dos2unix(2162688);
```

```
159084000
```

在 Kohana 中处理数组（数组类）

[返回目录]

Arr 类提供了一些数组的辅助方法。

从数组中获得一个值

Arr 类中众多有用的方法之一就是Arr::get()。这个方法能用来检查是否这个值存在于数组。并且当这个值不在数组中时，返回一个默认值。

```
Arr::get( array $array, string $key, mixed $default = NULL );
```

用法示例

```
// 考虑下面的 URL
http://www.example.com/blog/posts?page=3&limit=40

$page = Arr::get($_GET, 'page', 1);
$limit = Arr::get($_GET, 'limit', 20);
$sort_by = Arr::get($_GET, 'sort_by', 'date');
$sort_dir = Arr::get($_GET, 'sort_dir', 'DESC');
```

在上面的示例中当调用 Arr::get 时 $page 和$limit 将包含查询字符串里的值。而$sort_by 和$sort_dir 将使用声明的默认值。

从数组中获得多个值

Arr::extract()方法能被用来从一个数组中取的多个键。如果键不存在于数组中，将用默认值替代。

用法示例

```
//考虑下面的 URL
http://www.example.com/blog/posts?page=3&limit=40

$values = Arr::extract($_GET, array('page','sort_by'), NULL);
```

$values 数组现在看起来像这样..

```
array('page'=>3,'sort_by'=>NULL)
```

使用点号分隔符来获得一个联合数组中的值

Arr::path()方法能用点号分隔符来从数组中取的一个值

用法示例

```
//考虑下面的数组
$data = array(
    'setting' => 'value',
    'options' => array(
        'foo' => 'bar',
        ),
    );

$foo = Arr::path($data, 'options.foo', NULL);
```

确定一个数组是否是关联数组

Arr::is_assoc()方法用来测试一个数组是否是关联数组。

用法示例

```
//考虑下面的数组
$data = array(
    'setting' => 'value',
    'options' => array(
        'foo' => 'bar',
        ),
    );

if(Arr::is_assoc($data))
{
    echo "This IS an associative array";
}
```

用一串数字填充一个数组

Arr::range()方法能用一串数字来填充一个数组。

用法示例

```
$arr = Arr::range(1,5);
```

```
Array ( [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 )
```

```
$arr = Arr::range(2,6);
```

```
Array ( [2] => 2 [4] => 4 [6] => 6 )
```

在一个关联数组开头增加一个值

Arr::unshift()方法能用来在关联数组的开头增加一个值

用法示例

```
$data = array(
    'setting' => 'value',
    'options' => array(
        'foo' => 'bar',
        ),
    );

$data = Arr::unshift($data, 'class', 'standard');
```

```
Array ( [class] => standard [setting] => value [options] => Array ( [foo] => bar ) )
```

递归合并一个或多个数组

Arr::merge()方法能用来递归的合并一个或多个数组并且保留所有的键。注意：这个方法和 PHP 的 array_merge_recursive() 工作的不一样。这个函数附加数字键并且取代了联合键，而不像 array_merge_recursive()附加所有的键。

用法示例

```
$data1 = array(
    'setting' => 'value',
    'options' => array(
        'foo' => 'bar',
        ),
    );

$data2 = array(
    'class' => 'standard',
);

$merged = Arr::merge($data1,$data2);
```

```
Array ( [setting] => value [options] => Array ( [foo] => bar ) [class] => standard )
```

覆盖一个数组里的值

Arr::overwrite()方法能用来将一个输入的数组覆盖目标数组。注意。不存在的键不会被附加。

用法示例

```
$data1 = array(
      'setting' => 'value',
      'options' => array(
            'foo' => 'bar',
            ),
      );

$data2 = array(
      'setting' => 'value2',
      'options' => array(
            'foo' => 'bar2',
            ),
);

$data = Arr::overwrite($data1,$data2);
```

```
Array ( [setting] => value2 [options] => Array ( [foo] => bar2 ) )
```

从一个字符串创建一个可调用的函数和参数

Arr::callback()方法能用来从一个字符串表示法中创建一个可被调用的函数和参数列表。注意，这个函数不能验证回调字符串

```
// 获得回调函数和参数
list($func, $params) = Arr::callback('Foo::bar(apple,orange)');
```

```
// 获得回调函数的结果
$result = call_user_func_array($func, $params);
```

对数组进行二进制搜索

1． 待办：寻找关于此方法的好例子。

在 Kohana 中进行远程调用

[返回目录]

Remote 类用curl 来支持远程服务器通讯选项

检验一个远程 URL 的状态

Remote::status()方法用来返回一个 URL 的状态码

```
echo Remote::status('http://www.kohanaphp.com');
```

```
200
```

调用一个远程 URL

Remote::get()方法用来返回一个远程 URL 的输出。该方法接受 2 个参数

- **url**:你要调用的 url
- **options**:CURL 选项组成的数组。这些是可用的 curl_setopt() 的常量或者等值

在最基本的水平上，你可以载入一个远程网站

```
echo Remote::get('http://www.kohanaphp.com');
```

定义一些基本的 Curl 配置来调用一个远程网站

```
$options = array(
    CURLOPT_REFERER => 'http://www.google.com',
    CURLOPT_USERAGENT => "MozillaXYZ/1.0",
    CURLOPT_HEADER => 0,
    CURLOPT_TIMEOUT => 10,
);

echo Remote::get('http://www.kohanaphp.com', $options);
```

待办：增加更多使用 Curl 配置的复杂示例。

使用 Atom 和 RSS Feeds

[返回目录]

Feed 类支持一对用来使用 RSS 和 Atom feeds 的方法。

解析一个远程 Feed

Feed::parse()方法将解析一个远程的 feed 为一个数组。这个方法需要安装SimpleXML

下面的示例将加载 Kohana 论坛的 Atom feed 到一个数组。

```
$feed = Feed::parse('http://forum.kohanaphp.com/search.php?PostBackAction=Search&Type=Comments&Feed=ATOM');
```

如果你想限制层的数量，那么只要在解析的时候设置第二个属性

```
$feed = Feed::parse('http://forum.kohanaphp.com/search.php?PostBackAction=Search&Type=Comments&Feed=ATOM', 1);
```

创建 Feed

Feed::create()方法用给定的参数来创建 RSS 或者 Atom feed。下面是可接受的参数。

- **info:**你 feed 中的 header 详细信息的数组，如 pubDate 和 description
- **items:**一个填满了你 feed 项目的数组
- **format:**你的 feed 的格式，默认是 rss2
- **encoding:**你的 feed 使用的编码，默认是 UTF-8

下面的例子从一个博客数据例子的数组来创建一个 feed

```
$info = array(
                'title' => 'My Feed Title',
                'pubDate' => date("D, d M Y H:i:s T"),
                'description' => 'My recent blog posts',
            );
$items = array(
                array(
                    'title' => 'My Post Title',
                    'link' => 'blog/post/45',
                    'description' => 'This is the content summary of my post',
                ),
```

```
                array(
                    'title' => 'Another Post Title',
                    'link' => 'blog/post/46',
                    'description' => 'This is the content summary of my post',
                ),
                array(
                    'title' => 'Yet Another Post Title',
                    'link' => 'blog/post/47',
                    'description' => 'This is the content summary of my post',
                ),
            );

  $xml = Feed::create($info, $items);
```

这个将生成下列的 XML

```
  <?xml version="1.0" encoding="UTF-8"?>
  <rss version="2.0">
        <channel>
            <pubDate>Fri, 11 Dec 2009 15:57:51 CST</pubDate>
            <description>My recent blog posts</description>
            <title>My Feed Title</title>
            <link>http://www.example.com/</link>
            <generator>KohanaPHP</generator>
            <item>
                <title>My Post Title</title>
                <link>http://gallery.artmoi.com/blog/post/45</link>
                <description>This is the content summary of my
post</description>
            </item>
            <item>
                <title>Another Post Title</title>
                <link>http://gallery.artmoi.com/blog/post/46</link>
                <description>This is the content summary of my
post</description>
            </item>
            <item>
                <title>Yet Another Post Title</title>
                <link>http://gallery.artmoi.com/blog/post/47</link>
                <description>This is the content summary of my
post</description>
            </item>
        </channel>
  </rss>
```

使用文件

[返回目录]
File 类支持一对能帮助处理文件的方法。

确定文件的 Mime 类型

File 类支持两个来确定文件的 mime 类型的文件。
File::mime() 和 File::mime_by_ext()一样好.

File::mime() 用法

File::mime() 使文件名作为一个参数然后试着去做一些神奇的事来确定 mime 类型。这个方法并不是经常可靠。

```
echo File::mime('/home/example/kohana/media/img/close.gif');
```

```
image/gif
```

File::mime_by_ext()用法

File::mime_by_ext()给了一个简单的方法。它将文件扩展名作为一个参数然后从 Kohana 配置文件 system/config/mimes.php 中查找 mime 类型。

```
echo File::mime_by_ext('gif');
```

```
image/gif
```

这里有更完整的例子

```
$ext = pathinfo('media/img/close.gif', PATHINFO_EXTENSION);
echo File::mime_by_ext($ext);
```

```
image/gif
```

拆分与合并文件到多个部分

File 类支持 File::split() 和File::join() 方法
这些方法允许你或合并文件到多个部分

用法

```
$num_of_parts_split = File::split('/home/example/filename.txt');
```

```
$num_of_parts_joined = File::join('/home/example/filename.txt');
```

待办：增加一些关于内容的示例

使用数字

[返回目录]

Num 类支持一对能帮助处理数字的方法。

确定一个序数的英语格式

（th,st,nd,ect...）

Num::ordinal()方法返回一个序数的英语后缀（th,st,nd,etc）

```
echo Num::ordinal(1);
```

```
st
```

```
echo Num::ordinal(2);
```

```
nd
```

```
echo Num::ordinal(3);
```

```
rd
```

```
echo Num::ordinal(4);
```

```
th
```

如何格式化数字和货币

Num::format()方法能用来返回当前语言环境的数字格式。这个方法有 3 个属性:

- **number:**要格式化的数字
- **places:**十进制数的小数点位
- **monetary：**布尔标记用来确认数字是否是货币值

这个方法使用 php 的 localeconv()方法来获得你本地适当的格式详情。为了 localeconv()能工作，确认一下你在 application/bootstrap.php 文件里设置了你的本地(local)

```
/**
```

```
     * Set the default locale.
     *
     * @see    http://docs.kohanaphp.com/about.configuration
     * @see    http://php.net/setlocale
     */
    setlocale(LC_ALL, 'en_US.utf-8');
```

使用偏转器

[返回目录]

Inflector 类支持数个独一无二的方法用来帮助操控内容。

将单词转换成复数或者单数格式

偏转器(Inflector)类有两个有用的方法用来将单词转换成单数或者复数格式
Inflector::singular() 和Inflector::plural().

使用示例

```
echo Inflector::plural('artwork');
```

```
artworks
```

```
echo Inflector::singular('peoples');
```

```
people
```

讲一个词组转换成骆驼命名法

Inflector::camelize()方法能用来转换一个词组为骆驼命名法。

使用示例

```
echo Inflector::camelize('Ringo Star');
```

```
ringoStar
```

将字符串中的空格用下划线来代替

Inflector::underscore()方法能用来将一个词组中的空格用下划线来代替

使用示例

```
echo Inflector::underscore('I went for a walk');
```

```
I_went_for_a_walk
```

使字符串中的下划线或横线转换成适合人类阅读的样式

Inflector::humanize()方法能用来将下划线或横线解析成适合人类阅读的样式。

使用示例

```
echo Inflector::humanize('I-went_for_a-walk');
```

```
I went for a walk
```

验证一个单词是否不可数

Uncountable 方法经常用来帮助你解决单词是否是复数。但是如果你想用来制作类似这样的英语工具，这里有个粗略的例子可以让你明白如何使用它。

```
$nouns = array('food', 'bottle');
 foreach ($nouns as $noun) {
      echo 'How '.(Inflector::uncountable($noun) ? 'much' : 'many').' '.Inflector::plural($noun).'
'.(Inflector::uncountable($noun) ? 'is' : 'are').' there?'."\n";
      echo 'There '.(Inflector::uncountable($noun) ? 'is' : 'are').' '.(Inflector::uncountable($noun) ? 'a
lot of' : rand( 2 ,  99 )).' '.Inflector::plural($noun).".\n";
 }
```

设置和文件结构

[返回目录]

国际化

国际化文件能在 i18n 目录下找到。这些文件夹能在 system,application 或 modules 目录下找到。Kohana 自己的国际化文件能在 system 目录下找到。

格式

3.0.2 版本的语言区域格式不再仅仅是配置（仍有效）

文化就是国家组织和用户语言

```
<language>
en
de
es
nl
fr
...
<language>-<region>
en-us
en-gb
nl-be
fr-be
...
<language>-<region>-<local>
zh-Hant-HK
...
<language>-<region>-<local>-<you get the idea>
```

本地设置

在 bootstrap.php 中设置默认的时间区域

```
date_default_timezone_set('America/Chicago');
```

默认的文化是 en-us。如果要改变，你可以在 bootstrap 中增加一行。

```
i18n::lang('en-gb'); // 设置语言是英语，国家是英国（Great-Britain）
```

下面的文件结构图解看起来比一大段文本要容易理解的多。

文件结构

```
root
 +- application
 |     +- i18n
 |          +- '<language>'
 |          |    +- '<country>'.php
 |          |
 |          +- en
 |          |    +- gb.php
 |          |    +- us.php
 |          |
 |          +- de
 |          |    +- de.php
 |          |
 |          +- nl
 |          |    +- nl.php
 |          |
 |          +- fr
 |          |    +- ca.php
 |          |
 |          |    +- fr.php
 |          |
 |          +- '<language>'.php
 |          +- de.php
 |          +- en.php
 |          +- fr.php
 |          +- nl.php
 |
 +- system
 |     +- i18n
 |          +- en.php
 |          +- es.php
 |          +- fr.php
```

如何设置默认语言

[返回目录]

尽管这是一件很简单的事情，但是你可能并不清楚怎么做。如果你想设置（或改变）你的网站的默认语言，你需要向下面示例那样改变 i18n 类中的静态属性 $lang。

```
i18n::lang('ru-RU'); // 设置为俄罗斯语言
```

最好在你的 application/bootstrap.php 文件中做这些。

动态地改变默认语言

这里有许多的方法能完成你特定的需求。请将下面的例子放在你的 application/bootstrap.php 文件里的调用 Kohana::init(..)方法之后。它将默认语言保存在一个 $_SESSION 变量中，当一个新的语言在 $_GET 中被发现时，这个变量会被更新。

```
// 定义认可的语言
$languages = array('en-us', 'fr-be', 'es-mx');

// 如果语言并没有在 Session 中设置，使用默认的
if( ! isset($_SESSION['lang']) )
{
    $_SESSION['lang'] = 'en-us';
}

// 寻找语言中的变化
if( isset($_GET['lang']) AND in_array($_GET['lang'], $languages) )
{
    $_SESSION['lang'] = $_GET['lang'];
}
```

设置和检索语言字符串

[返回目录]

设置语言字符串

你需要将你所翻译的字符串增加到它们所属的语言或文化文件中（查看 设置和文件结构）

文件结构：

给全世界说法语的人

Application/i18n/fr.php

```
<?php defined('SYSPATH') or die('No direct script access.');

return array
(
        'French' => 'Français',
        'Hello, world!' => 'Bonjour, monde!',
);   // application/i18n/fr.php 结束
```

给所有在加拿大说法语的人

application/i18n/fr/ca.php

```
<?php defined('SYSPATH') or die('No direct script access.');

return array
(
        'Hello, world!' => 'Bonjour, Canada!',
); //End of application/i18n/fr/ca.php
```

取出语言字符串

翻译的字符串能用 I18n::get() 方法取出。如果在 Kohana 中设置的当前语言的 i18n 文件中没有发现所给的字符串，这个函数将返回相同的字符串。

示例

```
I18n::lang('fr');
echo I18n::get('Hello, world!');
//will outputs: Bonjour, monde!
```

```
I18n::lang('fr-ca');
echo I18n::get('Hello, world!');
//will outputs: Bonjour, Canada!

I18n::lang('fr');
echo I18n::get('Hello, myself!');
//will outputs: Hello, myself!
```

翻译消息

[返回目录]
消息用来确认错误信息。可以在 messages 目录下找到这些文件。这些目录能在 system,application, 或 modules 目录中被找到。Kohana 自己的消息文件在 system 目录中。

设置消息

一个消息文件的示例：

application/messages/validate.php

```
<?php defined('SYSPATH') OR die('No direct access allowed.');

return array(
        'not_empty'       => ':field must not be empty',
        'matches'         => ':field must be the same as :param1',
        'regex'           => ':field does not match the required format',
        'exact_length'    => ':field must be exactly :param1 characters long',
        'min_length'      => ':field must be at least :param1 characters long',
        'max_length'      => ':field must be less than :param1 characters long',
        'in_array'        => ':field must be one of the available options',
        'digit'           => ':field must be a digit',
        'decimal'         => ':field must be a decimal with :param1 places',
        'range'           => ':field must be within the range of :param1 to :param2',
);

// application/messages/validate.php 结束
```

翻译消息

当表单错误时，消息会像示例中的那样被翻译

```
$post = $user->validate_create($_POST); //返回一个 Validate 对象
$content->errors = $post->errors('validate');
```

这将加载一个叫 validate.php 的消息文件并且影响表单错误匹配的字符串

为了翻译那些字符串，我们需要将那些我们需要用到的字符串添加到 i18n/en.php i18n/fr.php 等地方。

application/i18n/en.php

```
<?php defined('SYSPATH') OR die('No direct access allowed.');

return array(
            ':field must not be empty'      => ':field must not be empty',
        etc...
);

// application/i18n/en.php 结束
```

application/i18n/fr.php

```
<?php defined('SYSPATH') OR die('No direct access allowed.');

return array(
            ':field must not be empty'      => ':field ne doit pas être vide',
        etc...
);

// application/i18n/fr.php 结束
```

多语言网站实例

[返回目录]

首先，我们增加几行配置用来告诉系统哪些语言是可用的。

```
<?php defined('SYSPATH') or die('No direct script access.');

return array(
    'language' => 'english',
    'language_abbr' => 'en',
    'lang_uri_abbr' => array("fr" => "french", "en" => "english"),
    'lang_ignore' => 'xx', //标注它将从 i18n 中寻找一个叫 xx 的语言. 因此 i18n:get('txt_sometext') 将返回 'txt_sometext'.
    'lang_desc' => array("en" => "English version", "fr"=>"French version"),
);
//./application/config/appconf.php 结束
```

那么 bootstrap.php 中的路由部分看起来将是这样：

```
/**
 * Load language conf
 */
$langs                  = Kohana::config('appconf.lang_uri_abbr');
$default_lang           = Kohana::config('appconf.language_abbr');
$lang_ignore            = Kohana::config('appconf.lang_ignore');
$langs_abr              = implode('|',array_keys($langs)) ;
if(!empty($langs_abr))
            $langs_abr .= '|' . $lang_ignore;

/**
 * 设置路由。每个路由必须最少有一个名字，一个 URI 和一个 default 的 URI
 */
Route::set('default', '((<lang>)(/)(<controller>)(/<action>(/<id>)))', array('lang' => "({$langs_abr})",'id'=>'.+'))
            ->defaults(array(
                'lang' => $default_lang,
                    'controller' => 'welcome',
                    'action'       => 'index',
```

```
        ));
```

路由模式将允许我们使用像这样的 url

```
http://mywebsite
http://mywebsite/en/
http://mywebsite/fr/
http://mywebsite/fr/mycontroller/myaction
http://mywebsite/mycontroller/myaction
```

但是我们不想重复写那些总是出现在 url 中的默认语言的内容。我们需要扩展 request 类来重新像这样路由 url

```
http://mywebsite/mycontroller/myaction  will redirect to default language =>
http://mywebsite/en/mycontroller/myaction
http://mywebsite will redirect to default language => http://mywebsite/en/
```

```
<?php defined('SYSPATH') or die('No direct script access.');

class Request extends Kohana_Request {
        /**
         * Main request singleton instance. If no URI is provided, the URI will
         * be automatically detected using PATH_INFO, REQUEST_URI, or PHP_SELF.
         *
         * @param    string    URI of the request
         * @return   Request
         */
        public static function instance( & $uri = TRUE)
        {
                $instance = parent::instance($uri);

                $index_page       = Kohana::$index_file;
                $lang_uri_abbr        = Kohana::config('appconf.lang_uri_abbr');
                $default_lang         = Kohana::config('appconf.language_abbr');
                $lang_ignore          = Kohana::config('appconf.lang_ignore');

                /* 从 uri 部分获得 lang_abbr */
                $segments = explode('/',$instance->uri);
                $lang_abbr = isset($segments[0]) ? $segments[0]:'';

                /* 获得当前语言 */
                $cur_lang = $instance->param('lang',$default_lang);
```

```
                        /* 验证无效的缩写 */
                        if( ! isset($lang_uri_abbr[$lang_abbr]))
                        {
                                /* 验证忽略的缩写 */
                                if ($cur_lang != $lang_ignore) {
                                        /* 验证并设置默认的 uri 标识符*/
                                        $index_page .= empty($index_page) ? $default_lang :
"/$default_lang";

                                        /* 插入语言 id 后重定位 */
                                        header('Location: '.Url::base().$index_page . '/' .
$instance->uri);
                                        die();
                                }


                        }


                        return $instance;
                }
}
// ./application/classes/request.php 结束
```

在 layout.php 类的 before()方法中，我们重载了语言参数，所以 i18n 能在 i18n/ (/i18n/en.php /i18n/fr.php)目录中找到文件。通常我们如果在 url 中使用 **语言-文化** （网站/en-us/控制器/动作）是显得很别扭的。

一个继承了 Controller_Template 的布局控制器的示例。

```
<?php defined('SYSPATH') or die('No direct script access.');

  class Controller_Layout extends Controller_Template
  {

        public $template = 'layout';


        /**
         * 你的控制器动作执行之前会先调用 before()方法.
         * 在我们的模板控制器中，我们覆盖了这个方法
         * 那么我们就能设置默认值.如果这些变量需要改变时，那么我们就能使用到了
         */
```

```
        public function before()
        {
            parent::before();

            #用 URI 设置语言
            $lng = Request::instance()->param('lang');
            i18n::$lang = $lng . '-' . $lng;   //我们在网站里不会用到文化，只需要设置语言.


                if ($this->auto_render)
                {
                        // 初始化变量值为空
                        $this->template->title   = '';
                        $this->template->content = '';

                                $this->template->styles = array();
                                $this->template->scripts = array();

                }
        }

        /**
         * 在你的控制器动作执行后会调用 after()方法
         * 在我们的模板控制器中，我们覆盖了这个方法
         * 那么我们能在模板生成之前进行最后的修改
         */
        public function after()
        {
                        if ($this->auto_render)
                        {
                                $styles = array(
                                        'css/main.css',
                                );

                                $scripts = array(
                                        'js/jquery-1.3.2.min.js',
                                );

                                $this->template->styles          =
array_merge( $this->template->styles, $styles );
                                $this->template->scripts         =
array_merge( $this->template->scripts, $scripts );

                        }
```

```
                parent::after();
        }
    }

// ./application/controller/layout.php 结束
```

我们在 URL 类中创建了一些方法，那么我们就不用在视图(view)中关心 url 中用户的语言了

```
<?php defined('SYSPATH') or die('No direct script access.');

class URL extends Kohana_URL {

/**
* 链接生成器
*
* 在增加用户语言后，会返回一个链接的标签
*
*/
public static function link_to($title, $url, $options=array()){

        $option_str = '';
        $site_url    = Url::base();
        $lng = Request::instance()->param('lang');

    foreach($options as $key => $option){
        $option_str .= "{$key}='{$option}' ";
    }

    return "<a href='{$site_url}{$lng}/{$url}' {$option_str}>{$title}</a>";
}

/**
* 转向
*
* 增加用户语言后转到所给的 控制器/方法
*
*/
public static function redirect($to = ''){
        $site_url    = Url::base();
        $lng = Request::instance()->param('lang');

        header("Location: {$site_url}{$lng}/{$to}");
        die();
}
```

```
}
//./application/classes/url.php 结束
```

注意，如果你使用以上的 layout.php 示例，你的控制器应该看起来像这样

```
<?php defined('SYSPATH') or die('No direct script access.');

class Controller_Home extends Controller_Layout {

		function action_index(){
			$this->template->content = View::factory('home');

		}


} // ./application/controller/home.php 结束
```

ORM 实例

[返回目录]

对象关系映射（Object Relational Mapping 简称 ORM）允许你把数据库中的数据
当成一个 PHP 对象来操纵和控制。一旦你定义了 ORM 和你的数据库中数据的关系，那么无论你用任何你喜欢的方式操纵数据，以及保存结果到数据库，都不需要使用 SQL 语言。通过创建按照配置约定的模型之间的关系，大部分从数据库中重复的编写创建，读取，更新和删除信息的查询可以被减少或者完全消除。所有的关系都能被自动用 ORM 库来处理并且你可以像使用标准对象属性一样访问相关数据。

注意：请确定你使用了最新的 3.0.* 来处理，
最新的版本可以访问 http://dev.kohanaphp.com/projects/kohana3/files

启用

第一步是 启用并配置数据库(database)模块

Orm 模块被包括在 Kohana3.0 安装程序中。但是在你使用前需要你去启用它。

在你的 application/bootstrap.php 文件中修改调用的 Kohana::modules() 方法，按照下面的示例来包含 orm 模块。

```
Kohana::modules(array(
        'userguide'         => MODPATH.'userguide',
        'database'          => MODPATH.'database',
        'orm'           => MODPATH.'orm',           // orm access
));
```

没有必须的配置文件

定义模型

简单

如果你的数据库和名称约定（在 v2 版本的文档中有提到）匹配并且你不使用 pdo 链接,模型可以像这样简单的定义：

```
class Model_Account extends ORM
{}
```

定制

一些基本模型属性的定义：

```
class Model_Account extends ORM
{
    protected $_db = 'default'; //或者其他配置中定义的数据组 n

    protected $_table_name   = 'strange_tablename'; // 默认: accounts
    protected $_primary_key = 'strange_pkey';       // 默认: id
    protected $_primary_val = 'strange_name';       // 默认: name (作为主键(primary)的值)

    // $_table_columns 的默认值: 使用数据库自动检查来寻找列和信息
    // 查看所有可能的列属性 http://v3.kohanaphp.com/guide/api/Database_MySQL#list_columns
        protected $_table_columns = array(
        'column_name'     => array('data_type' => 'int',      'is_nullable' => FALSE),
        'column_name2'    => array('data_type' => 'string', 'is_nullable' => TRUE),
    );

    // 这里提到的字段(field)能像属性一样访问，但是并不会被写操作引用
    protected $_ignored_columns = array(
        'helper_field',
    );
}
```

注意：在这里的 Kohana 3.0.3 中，当 table_prefix 在数据库配置中设置就是一个 bug。你应该避免使用这个设置直到 3.0.4

定义关联

查看 jheathco's repo wiki 来了解定义关联的详细情况

加载

你可以使用 ORM::factory 方法或者 ORM::__construct 来创建一个模型实例：

```
$user = ORM::factory('user');
// 或者
$user = new Model_User();
```

构造器和工厂方法也接受一个主键值用来加载给定的模型数据：

```
// 加载 ID 5 的用户
$user = ORM::factory('user', 5);
```

ORM::loaded 检查给定的模型是否已经成功加载。

（了解请看http://github.com/kohana/userguide/blob/master/guide/tutorials.orm.md)

延迟加载

3.0 ORM 实际上并不加载模型的数据，直到它是绝对必要时。这并不像以前的 ORM 版本。例

如，如果你执行下列操作：

$user = ORM::factory('user', 1); $user->name = 'Joe'; $user->save();

上面的阿迪表仅仅有一个数据库查询，就是更新了主键记录为 1。在加载模型数据时并没有执行。模型数据仅仅在它需要提高性能和限制数据库开销的时候才被加载。
echo $user->name 会，另一方面，迫使该模型数据加载。
（了解请看http://wiki.github.com/jheathco/kohana-orm/)

写

简单插入和更新

```
$user = ORM::factory('user', 1);
$user->name = 'Joe';
$user->save();
```

如果要附加必要的插入/更新的逻辑到保存，你可以覆盖你的模型中的 save()函数并使用相同的方法测试 ORM 来确定你的插入或更新。

```
    public function save()
    {
        // 如果它是一个插入...
        if ($this->empty_pk()
            || isset($this->_changed[$this->_primary_key])) {

            ... do insert logic ...
        }

        else {
            .. do update logic ...
        }

        return parent::save();
    }
```

你可以使用 ORM::save_all 方法来更新多条记录。

```
$user = ORM::factory('user');
$user->name = 'Bob';

// 将所有动态记录的 name 改为 Bob
```

```
$user->where('active', '=', TRUE)->save_all();
```

ORM::saved 检查所给的模型是否已经保存
（了解请看http://github.com/kohana/userguide/blob/master/guide/tutorials.orm.md）

删除

删除记录
使用 ORM::delete 和 ORM::delete_all 来删除记录。这些方法和上面描述的 save 使用同样的异常方式，ORM::delete 只接受一个可选的参数，也就是要删除的记录的 id
（了解请看http://github.com/kohana/userguide/blob/master/guide/tutorials.orm.md）

使用数据库表达式来写

分配到列的所有值都有写查询保护来防止 sql 注入溢出。如果必须要指定一个数据库列表达式的结果，使用 DB::expr()

```
$user = ORM::factory('user', 1);
$user->name = 'Joe';
$user->last_modified = DB::expr('now()');
$user->modified_count = DB::expr('modified_count + 1');
$user->save();
```

警告：如果表达式涉及到用户输入，你要自己负责溢出。

模型验证(Validation)

你可以使用 $_rules,$_filters 和$_callbacks 属性(查看validation 文档))来列出规则，过滤器和回调函数。什么是模型中定义规则的最佳做法以及如何验证它们，这是一个很热门的话题。
你决定。

```
class Model_Account extends ORM
{
    ...
    protected $_rules = array(
        'name'         => array('not_empty' => null),
        'website'      => array('url'       => null),
        'zip'          => array('regex'     => array('/^\d{5}(\-\d{4})?$/')),
        'phone_num' => array('phone'     => array(array(10,11,14))),
    );
    protected $_filters = array(TRUE => array('trim' => NULL));

    protected $_callbacks = ... refer to validation docs

    protected $_labels = array(
```

```
            'column_name' => 'pretty name',
        );
}
```

用法：

```
$user = ORM::factory('user', 1);
$user->name = 'Joe';
$user->values($_POST);
if ($user->check()) {
    $user->save();
} else {
    $errors = $user->errors();
}
```

针对非模型字段的验证

有时你需要一个模型字段和独立表单字段（就像迫使两个密码或者 email 字段匹配）的规则参考。
对于那些谁都希望能在模型中定义规则（不是将这些验证放在控制器中），你需要在$_ignored_columns 中注册一个外部字段使模型验证能访问它。

```
class Model_User extends ORM
{
    $_ignored_columns = array(
        'password_confirm',
    );

    protected $_rules = array(
        'password_confirm' => array('not_empty'=> null),
        'password'         => array('not_empty' => null, 'matches', array('password_confirm')),
    );
}
```

读

注意，如果你还不明白Database 库的查询生成器，你可以这样开始，这个库里有如此多关于 orm 的可用的函数。（查看$_db_methods 属性来获得被支持的函数调用的完整列表）。

使用 ORM::find 和 ORM::find_all 方法调用来取得记录。

使用 $user=ORM::factory（‘user’）
->where（‘active’,’ =’ ,TRUE)
->where（‘name’,’ =’ ,’ Bob’）
->find();
来抓取用户名为 Bob 的首条活跃用户。使用 $users=ORM::factory（‘user’）将抓取所有叫 Bob 的用户。

```
...
```

```
    ->find_all();
```

当你使用 ORM::find_all 检索了模型列表，你可以遍历它们作为你的数据库结果来使用：

```
foreach ($users as $user) {

    ...
}
```

Find_all()列表是(一次延迟加载)一个模型对象的数组。如果你使用一个很大的列表，最好使用数据库查询库生成一个数组。

获得对象列表（find_all）

[返回目录]
主要获取自
http://forum.kohanaphp.com/comments.php?DiscussionID=3639&page=1#Item_9

获得对象列表

```
// 加载所有文章
$articles = Sprig::factory('article')->load(NULL, FALSE);
```

```
// 加载开头 5 篇文章
$articles = Sprig::factory('article')->load(NULL, 5);
```

```
// 加载开头 5 篇文章并按 published_date 降序排列
$query = DB::select()->order_by('published_date', 'DESC');
$articles = Sprig::factory('article')->load($query, 5);
```

遍历

count()检查数量,然后使用 foreach

```
...
$articles = Sprig::factory('article')->load($query, 5);
if (count($articles) > 0)
{
    foreach($articles as $article)
    {
        ...
    }
}
...
```

用 Sprig 用户模型来认证

[返回目录]

Sprig 模块 是 Kohana 的一个对象建模系统 ，灵感来自 Django

Sprig 带有一个预定义模型来处理用户，使用主要的 Sprig 类能很容易的提供用户身份验证。

注意，下面的代码设置/使用了一个简单的 cookie 来持有 ID，这并非安全验证用户的方法。因为它很容易被伪造。下面的代码是为了演示控制流和 Sprig 认证看起来会什么样子。请选择一个更安全的身份验证方法。

基本处理

1. 下载并安装 Sprig 模块
2. 在数据库中创建一个用户表
3. 设置你的模板控制器来重定位到没有登录的用户
4. 为用户登录或登出创建用户验证控制器

创建数据库

这里有一个 Sprig 用户模型可以使用的数据库结构

```
CREATE TABLE `users` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `username` varchar(30) NOT NULL,
  `password` varchar(40) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

设置你的模板控制器来重定位到未登录的用户

下面的模板控制器查找每个请求的用户的 cookie。如果发现 cookie 则尝试加载用户记录。如果没有发现 cookie 或者用户记录加载失败，那么控制器重定位到登录表单。

```
abstract class Controller_Website extends Controller_Template {

    // 用户模型的公共变量
        public $user;

    // 使用 TRUE 意味着请求默认需要身份验证 t
```

```
        public $auth_required = TRUE;

        public function __construct(Request $request)
        {
            parent::__construct($request);

        // 如果用户 id 的 cookie 被发现，则尝试加载用户
            if ($id = Cookie::get('user'))
            {
                $user = Sprig::factory('user')
                            ->values(array('id' => $id))
                            ->load();

                if ($user->loaded())
                {
                            // 用户登录
                            $this->user = $user;
                }
            }

        // 如果用户没有日志则需要登录
            if ($this->auth_required AND ! $this->user)
            {
                // 重定位到登录页面
                $request->redirect('auth/login');
            }
        }
}
```

验证控制器

下面的验证控制器用来生成和验证登录的表单。

```
class Controller_Auth extends Controller_Website {

    //这个控制器不需要认证 n
        public $auth_required = FALSE;

    // 登录动作
        public function action_login()
        {
            // 设置你的视图，绑定$user 和$errors
                $this->template->title   = 'Login';
                $this->template->content = View::factory('auth/login')
                            ->bind('user', $user)
```

```
                        ->bind('errors', $errors);

                // 加载一个空用户
                $user = Sprig::factory('user');

        // 加载 sprig 模型中定义的规则到验证工厂
                $post = Validate::factory($_POST)
                        ->rules('username', $user->field('username')->rules)
                        ->rules('password', $user->field('password')->rules);

        // 验证 post
                if ($post->check())
                {
                        // 使用用户名和密码加载用户
                        $user->values($post->as_array())->load();

                        if ($user->loaded())
                        {
                                // 保存用户 id
                                Cookie::set('user', $user->id);

                                // 重定向到首页
                                $this->request->redirect('');
                        }
                        else
                        {
                                $post->error('password', 'invalid');
                        }
                }

                $errors = $post->errors('auth/login');
        }

        public function action_logout()
        {
                $this->template->title = 'logout';
                $this->template->content = View::factory('auth/logout');

                if (isset($_POST['logout']))
                {
                        // 删除用户 cookie
                        Cookie::delete('user');

                        // 重定向到首页
```

```
                $this->request->redirect('');
            }
        }

    } // 用户结束
```

视图

这里是 auth/login 视图

```
<?php echo form::open(NULL, array('id' => 'login')) ?>

    <h1><?php echo __('Login') ?></h1>

    <?php include Kohana::find_file('views', 'errors') ?>

    <ol>
      <li><label><span><?php echo __('Username') ?></span> <?php echo
form::input('username', $user->username) ?></label></li>
      <li><label><span><?php echo __('Password') ?></span> <?php echo
form::password('password') ?></label></li>
    </ol>

    <?php echo form::button(NULL, 'Login', array('type' => 'submit')) ?>

    <?php echo form::close() ?>
```

这里是 auth/logout 视图

```
    <?php echo form::open(NULL, array('id' => 'logout')) ?>

    <h1 class="top">Log Out</h1>

    <p>Are you sure you want to log out?</p>

    <?php echo form::button('logout', 'Yes, please!', array('type' => 'submit')) ?>

    <?php echo form::close() ?>
```

还有当有错误时候，我们的错误视图，它包含进了 auth/login 的 errors 的视图

```
    <?php if ( ! empty($errors)): ?>
    <ul class="errors">
    <?php foreach ($errors as $field => $error): ?>
      <li rel="<?php echo $field ?>"><?php echo ucfirst($error) ?></li>
    <?php endforeach ?>
    </ul>
```

```
<?php endif ?>
```

用 Sprig 用户模型使用 sprig-auth 来认证

[返回目录]

关于如何在sprig (by shadowhand)顶端用 Kohana 内置验证模块使用sprig-auth (by banks) 的简明教程。sprig-auth 实现了 一个验证模块的 sprig 驱动且工作良好。

本教程是建立在本维基手册和论坛源码基础上的。

- 用 Sprig 用户模型来认证
- 在你的控制器中使用身份认证模块

这里有个主要的陷阱，这是 sprig 中的一个导致每个线程无法验证的 bug

- http://github.com/shadowhand/sprig/issues/61

获得 Sprig 和 Sprig-Auth

我从 Github(用下载链接)中下载了最新的版本，将 sprig 和 sprig-auth 放到你的 modules 目录下

```
/modules
  /sprig
    /classes
      ...
  /sprig-auth
    /classes
      ...
```

不幸的是，你必须在每个 bug 之前在这里修改一些东西。（截至 2010/2/17）

/modules/sprig/classes/sprig/core.php

```
    public function load(Database_Query_Builder_Select $query = NULL, $limit = 1)
    {
        $this->state('loading');   // 约在 949 行添加此行
        ...
        ...
                if (count($result))
                {
                  //$this->values($result[0])->state('loaded');   // 注释掉此行
                  $this->state('loading')->values($result[0])->state('loaded'); // 约在 1007 行添加此行
```

```
            }
        ...
    }
```

我们在这里有一些问题，如对象的序列化。但是 banks 在他的代码里做了非常出色的补丁。要知道，一旦真正在 sprig 中进行修复，之后总会有一些代码被移除。

/modules/sprig-auth/sprig-auth-schema.sql – 运行这个脚本在你的数据库中创建表

代码

注意：这里代码中的一些复制于各个源码，诸如上面的链接和 sprig 以及 sprig-auth 中 README.MD 文件里提供的示例。

/application/bootstrap.php

```
Kohana::modules(array(
    'sprig-auth' => MODPATH.'sprig-auth', // Sprig Auth
    'sprig'      => MODPATH.'sprig',      // Sprig
    'auth'       => MODPATH.'auth',       // 基本认证
    'codebench'  => MODPATH.'codebench',  // Benchmarking 工具
    'database'   => MODPATH.'database',   // 数据库访问
    'image'      => MODPATH.'image',      // 图像操纵
    'orm'        => MODPATH.'orm',        // 对相关系映射
    'pagination' => MODPATH.'pagination', // 分页
    'userguide'  => MODPATH.'userguide',  // 用户手册和 API 文档
    ));
```

/application/classes/controllers/auth.php 使用一个 action_register() 函数

待办：检验和过滤输入信息

```
class Controller_Auth extends Controller_Template
{

    // 主要基本的模板
    public $template = 'template/layout/public';

    public $auth_required = FALSE;

    /**
     * Index
     */
    public function action_index()
```

```
{
    // 只是重定位到登录页面
    $this->request->redirect('auth/login');
}

/**
 * 登录
 */
public function action_login()
{
    // 如果一个用户已经登录在这个页面，假如他想做一个新的登录
    Auth::instance()->logout();

    // 加载登录页面
    $this->template->title = 'Login';
    $this->template->content = View::factory('auth/login')
        ->bind('user', $user)
        ->bind('errors', $errors);

    //如果用户正在尝试登录
    if ($_POST)
    {
        // 用户是否选中了记住登录状态的复选框?
        $remember = isset($_POST['remember']) ? TRUE : FALSE;

        // 使用 Auth(身份验证)来登录用户
        Auth::instance()->login($_POST['username'], $_POST['password'], $remember);
        if ( ! Auth::instance()->logged_in() )
        {
            $errors = array('Login failed.  Please use a valid username and password.');
            return;
        }

        //如果成功定位到首页
        $this->request->redirect('user');
    }
}


/**
 * 注销
 */
public function action_logout()
{
```

```
        //如果一个用户已经登录在这个页面，假如他想做一个新的登录
        Auth::instance()->logout();
        Cookie::delete('user');

        // 加载登录页面
        $this->template->title = 'Logout';
        $this->template->content = View::factory('auth/logout')
            ->bind('user', $user)
            ->bind('errors', $errors);
    }

    /**
     * 注册
     *
     * 这个方法假设该人正在注册进入网站
     * 为了简便，他们需要提供 3 个东西：用户名，密码，email 地址
     * 以及其他一切的可选项（目前）
     *
     */
    public function action_register()
    {
        // 如果我们正在处理一个新的注册（不知用户是如何获得此页面的情况下）
        // 我们需要首先注销他们
        Auth::instance()->logout();

        // 实例化一些对象
        $user = Sprig::factory('user');

        // 加载视图
        $this->template->content = View::factory('auth/register')
            ->bind('user', $user)
            ->bind('errors', $errors);

        // 如果有一个 post 并且$_POST 不为空
        if ($_POST)
        {
            try
            {
                // 待办: 清理进入的值
                $user->values($_POST);
                // 为用户添加登录角色
                $user->values(array(
                    'roles' => Sprig::factory('role', array('name' => 'login'))->load()->id,
                ));
```

```
                    // 创建用户
                    $user->create();

                    // 标记用户在里面
                    Auth::instance()->login($_POST['username'], $_POST['password']);

                    // 重定位到用户账户
                    Request::instance()->redirect('user');

                }
                catch (Validate_Exception $e)
                {
                    $errors = $e->array->errors('validate');
                }

            }
        }
}
```

/application/classes/model/user.php

/application/views/template/

验证一个 Sprig 模型

[返回目录]

Sprig 模型的一个优点是能够容纳验证信息。鉴于下列模型：

```
class Model_Project extends Sprig {

    protected function _init()
    {
        $this->_fields += array(
            'id' => new Sprig_Field_Auto,
            'name' => new Sprig_Field_Char(array(
                'max_length' => '32',
                'unique' => TRUE,
            )),
            'description' => new Sprig_Field_Text,
        );
    }

} // 项目结束
```

...你可以传递一个表单 post 到模型，并且检索任何错误：

```
public function action_create()
{
    $this->template->content = View::factory('projects/edit')
        ->bind('project', $project)
        ->bind('errors', $errors);

    $project = Sprig::factory('project');

    if ($name = $this->request->param('name'))
    {
        $project->name = $name;
    }

    if ($_POST)
    {
        try
        {
            $project->values($_POST)->create();
```

```
                $this->request->redirect(Route::get('project')->uri(array('name' =>
$project->name)));
            }
            catch (Validate_Exception $e)
            {
                $errors = $e->array->errors('project/edit');
            }
        }
    }
```

在保存前检查验证

使用Sprig->check()在尝试保存前执行验证（validation）

@待办：示例

给每个字段传递规则

你还可以使用自己的验证对象(validate object)来对个别字段进行验证。

@待办：示例

字段验证规则

查看API 文档和验证属性用的 sprig/field.php 文件，你可以设置每个字段
密切注意他们的默认值。如果一个字段可以是空的，例如，你必须设置空为 TRUE，否则你的模型将无法验证。

通过 AJAX 来验证一个 Sprig 模型

[返回目录]

我把那些使用 KO3 和 Sprig 来轻松用 Ajax 验证一个字段和多个字段的控制器一起列在下面。我想这对大家可能是有用的，它们就是：

Controller_Service 是一个控制器，它用来确认所给出的请求是否来自 ajax，否则就重定位到默认动作。

Controller_Service_Validate 继承了 Controller_Service 并且针对定义过的 Sprig 模型来验证 _POST 数据

用法：

比如说你有下面的 Sprig 模型

```
  class Model_User extends Sprig
  {
protected function _init()
{
        $this->_fields += array(
                'id' => new Sprig_Field_Auto(array(
                        'public' => TRUE,
                )),
                'username' => new Sprig_Field_Char(array(
                        'empty'   => FALSE,
                        'rules'   => array(
                                'regex' => array('/^[\pL_.-]+$/ui')
                        ),
                )),
                'email' => new Sprig_Field_Email(array(
                        'empty'   => FALSE,
                )),
        );
}
```

```
}
```

你可以使用继承 Controller_Service_Validate 并且设置模型属性来为这个模型创建一个验证控制器

```
class Controller_Validate_User extends Controller_Service_Validate
{
        public $model = 'user';
}
```

使用你喜欢的 javascript 方式，只要它可以让你可以制作一个 ajax 调用到这个控制器并通过 Json 来获取验证结果。

下面的示例当 username 改变的时候使用了 jQuery 来验证

```
$(document).ready(function() {
    $(":input[name='username']").change(function(){

        if ( ! $(this).val())
            return;

        $.post('http://www.example.com/validate_user/validate',
            {'username':$(this).val()},
                function (response){
                    if (response.error)
                    {
                        alert( response.errors.username );
                    }
                },
        'json');
    });
});
```

继承 Model_Auth_User 类

[返回目录]

Kohana 包含了继承自 ORM 的 Model_Auth_User 类。你的 Model_User 只须包含额外的代码，诸如验证创建或者你扩展的 users 表中的用户列。

这个文件位于 /application/classes/model/user.php.

```php
<?php
class Model_User extends Model_Auth_User {

	public function validate_create(& $array)
	{
		// 初始化验证库并配置一些规则
		$array = Validate::factory($array)
			->rules('password', $this->_rules['password'])
			->rules('username', $this->_rules['username'])
			->rules('email', $this->_rules['email'])
			->rules('password_confirm', $this->_rules['password_confirm'])
			->filter('username', 'trim')
			->filter('email', 'trim')
			->filter('password', 'trim')
			->filter('password_confirm', 'trim');

		#执行父类(parent)定义的 username 回调
		foreach($this->_callbacks['username'] as $callback){
			$array->callback('username', array($this, $callback));
		}

	#执行父类中定义的 email 回调
		foreach($this->_callbacks['email'] as $callback){
			$array->callback('email', array($this, $callback));
		}

		return $array;
	}
```

在控制器中使用身份认证模块

[返回目录]

注意：在你使用本手册之前，必须遵循以前的

http://kerkness.ca/wiki/doku.php?id=extending_the_model_auth_user_class

创建我们的数据表

如果你还没有数据表，这里是 MySQL 代码

```
CREATE TABLE IF NOT EXISTS `roles` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL,
  `description` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_name` (`name`)
) ENGINE=INNODB   DEFAULT CHARSET=utf8 ;

INSERT INTO `roles` (`id`, `name`, `description`) VALUES
(1, 'login', 'Login privileges, granted after account confirmation'),
(2, 'admin', 'Administrative user, has access to everything.');


CREATE TABLE IF NOT EXISTS `roles_users` (
  `user_id` INT(10) UNSIGNED NOT NULL,
  `role_id` INT(10) UNSIGNED NOT NULL,
  PRIMARY KEY (`user_id`,`role_id`),
  KEY `fk_role_id` (`role_id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8;


CREATE TABLE IF NOT EXISTS `users` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(127) NOT NULL,
  `username` VARCHAR(32) NOT NULL DEFAULT '',
  `password` CHAR(50) NOT NULL,
  `logins` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `last_login` INT(10) UNSIGNED DEFAULT NULL,
  PRIMARY KEY (`id`),
```

```
  UNIQUE KEY `uniq_username` (`username`),
  UNIQUE KEY `uniq_email` (`email`)
) ENGINE=INNODB   DEFAULT CHARSET=utf8 ;

CREATE TABLE IF NOT EXISTS `user_tokens` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `user_id` INT(11) UNSIGNED NOT NULL,
  `user_agent` VARCHAR(40) NOT NULL,
  `token` VARCHAR(32) NOT NULL,
  `created` INT(10) UNSIGNED NOT NULL,
  `expires` INT(10) UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uniq_token` (`token`),
  KEY `fk_user_id` (`user_id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8 ;


ALTER TABLE `roles_users`
  ADD CONSTRAINT `roles_users_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON
DELETE CASCADE,
  ADD CONSTRAINT `roles_users_ibfk_2` FOREIGN KEY (`role_id`) REFERENCES `roles` (`id`) ON
DELETE CASCADE;

ALTER TABLE `user_tokens`
  ADD CONSTRAINT `user_tokens_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON
DELETE CASCADE;
```

注册，登录和登出用户

```
        function action_register()
        {
                #如果用户已经登录
                if(Auth::instance()->logged_in()!= 0){
                        #重定向到用户账户
                    Request::instance()->redirect('account/myaccount');
                }

                #加载视图
                $content = $this->template->content = View::factory('register');

                #如果这里获得一个 post 并且$_POST 不为空
                if ($_POST)
                {
                        #初始化一个新用户
                        $user = ORM::factory('user');
```

```
                                #加载验证规则，过滤器等等...
                                $post = $user->validate_create($_POST);

                                #如果使用用户模型里设置的规则来验证 post 数据
                                if ($post->check())
                                {
                                        #影响并清理用户对象里的变量
                                        $user->values($post);

                                        #创建一个账户
                                        $user->save();

                                        #增加一个登录角色给用户
                                        $login_role = new Model_Role(array('name'
=>'login'));
                                        $user->add('roles',$login_role);

                                        #标记用户在里面
                                        Auth::instance()->login($post['username'],
$post['password']);

                                        #重定位用户账户
                                        Request::instance()->redirect('account/myaccount');
                                }
                                else
                                {
                                  #为显示视图获得错误
                                        $content->errors = $post->errors('register');
                                }
                        }
                }



                public function action_signin()
                {
                        #如果用户已经登录
                        if(Auth::instance()->logged_in()!= 0){
                                #重定位到用户账户
                                Request::instance()->redirect('account/myaccount');
                        }
```

```
                $content = $this->template->content = View::factory('signin');

                #如果这里获得一个 post 并且$_POST 不为空
                if ($_POST)
                {
                        #初始化一个新用户
                        $user = ORM::factory('user');

                        #检查身份验证
                        $status = $user->login($_POST);

                        #如果使用用户模型里设置的规则来验证 post 数据
                        if ($status)
                        {
                                #重定位到用户账户
                                Request::instance()->redirect('account/myaccount');
                        }else
                        {
                          #为显示视图获得错误
                                $content->errors = $_POST->errors('signin');
                        }

                }
        }



        public function action_signout()
        {
                #注销用户
                Auth::instance()->logout();

                #重定向到用户账户，如果注销如预期完成，那么转到登录页面
                Request::instance()->redirect('account/myaccount');
        }
```

用继承控制器的模板来使用身份验证

一个继承 Controller_Template 的布局控制器的示例。

```
<?php defined('SYSPATH') or die('No direct script access.');

class Controller_Layout extends Controller_Template
{
```

```
        public $template = 'layout';

        //控制访问整个控制器，如果没有设置为 FALSE，那么我们将紧紧允许特定的用户角色 d
        // 能设置为一个字符串或者数组，例如 array('login', 'admin') 或者 'login'
        public $auth_required = FALSE;

        // 控制访问单独动作
        // 'adminpanel' => 'admin'就爱你过仅仅允许角色为 admin 的用户访问 action_adminpanel
        // 'moderatorpanel' => array('login', 'moderator') 将仅仅允许角色为登录和主人的用户访问
action_moderatorpanel
        public $secure_actions = FALSE;



        /**
         * before()方法在你的控制器动作执行前调用.
         * 在我们的模板控制器中，我们可以覆盖了这个方法来设置默认值。
         * 这些变量就可以在需要他们被修改的时候被我们使用
         */
        public function before()
        {
            parent::before();

            #打开 session
            $this->session= Session::instance();

            #检查用户身份验证和角色
            $action_name = Request::instance()->action;
            if (($this->auth_required !== FALSE &&
Auth::instance()->logged_in($this->auth_required) === FALSE)
                    || (is_array($this->secure_actions) && array_key_exists($action_name,
$this->secure_actions) &&
                    Auth::instance()->logged_in($this->secure_actions[$action_name]) ===
FALSE))
            {
                    if (Auth::instance()->logged_in()){
                            Request::instance()->redirect('account/noaccess');
                    }else{
                            Request::instance()->redirect('account/signin');
                    }
            }

                if ($this->auto_render)
                {
                        // 初始化为空值
```

```
                        $this->template->title   = '';
                        $this->template->content = '';

                                $this->template->styles = array();
                                $this->template->scripts = array();

                }
        }

        /**
         * after()方法在你的控制器动作执行后被调用
         * 在我们的模板控制器中，我们可以覆盖了这个方法
         * 那么我们就能在所有东西在模板中呈现之前做最后的修改.
         */
        public function after()
        {
                        if ($this->auto_render)
                        {
                                $styles = array(
                                        'css/main.css' => 'screen',
                                );

                                $scripts = array(
                                        'js/jquery-1.3.2.min.js',
                                );

                                $this->template->styles          =
array_merge( $this->template->styles, $styles );
                                $this->template->scripts         =
array_merge( $this->template->scripts, $scripts );

                        }
                        parent::after();
        }
  }

//./application/controller/layout.php 结束
```

控制器为登录并且为管理员的角色授权

```
<?php defined('SYSPATH') or die('No direct script access.');

class Controller_Admin extends Controller_Layout {

        public $auth_required = array('login','admin'); //在访问这个控制器时候需要身份验证
```

```
        function action_index(){
                $this->template->content = View::factory('adminindex');

        }


} // ./application/controller/admin.php 结束
```

只有登录且为管理员的角色可以 post,编辑，删除。其他任何人（甚至没有登录）可以看见的是：

```
<?php defined('SYSPATH') or die('No direct script access.');

class Controller_Admin extends Controller_Layout {

        public $secure_actions      = array('post' => array('login','admin'),
                                            'edit' => array('login','admin'),
                                            'delete' => array('login','admin'));


        function action_index(){ }

      function action_view(){ }

      function action_post(){ }

      function action_edit(){ }

      function action_delete(){ }


} // ./application/controller/admin.php 结束
```

如何更好的在 Kohana 控制器中使用图片

[返回目录]

如果你想为图片使用 Kohana 中的控制器，你可以按照下列步骤：

1. 定义一个可以捕捉所有图片请求的路由
2. 创建一个用来处理图片请求的 Controller_Images
3. 创建一个动作用来把图片返回传递给用户

路由

/home/kerkness/kohana/application/bootstrap.php

```
Route::set('images', 'image/<file>', array('file' => '.+.(?:jpe?g|png|gif)'))
    ->defaults(array(
        'controller' => 'images',
        'action' => 'index',
    ));
```

控制器

/home/kerkness/kohana/application/classes/controller/images.php

```
class Controller_Images extends Controller {

    protected $directory = 'static/images/';

    public function action_index()
    {
        // 创建一个文件名
        $file = $this->directory.$this->request->param('file');

        if ( ! is_file($file))
        {
            throw new Kohana_Exception('Image does not exist')
        }

        //
        // 在这里检查你的权限
        //
```

```
            // 设置 mime 类型
            $this->request->headers['Content-Type'] = File::mime($file);
            $this->request->headers['Content-length'] = filesize($file);

            // 发送设置好的头到浏览器
            $this->request->send_headers();

            // 发送文件
            $img = @ fopen($file, 'rb');
            if ($img) {
                fpassthru($img);
                exit;
            }
        }

    } // 图片结束
```

用法

```
echo Html::image('image/my_image.jpg');
```

如何使用 Hudson 为 KO3 基础对象安装持续集成

[返回目录]
本页面是关于如何为 CI(CodeIgniter 也就是 Kohana 的前身)安装Hudson 的详细说明。查看论坛主题获得更多信息。这篇博客文章也是一个为 PHP 项目安装 Hudson 的很好的资源。

什么是持续集成？

- http://www.martinfowler.com/articles/continuousIntegration.html
- http://en.wikipedia.org/wiki/Continuous_Integration

七个 CI 最佳实践

来自CI 书籍

1. Commit code frequently (make small, localized changes, commit after each task)
2. Don't commit broken code
3. Fix broken builds immediately
4. Write automated developer tests
5. All tests & inspections must pass
6. Run private builds
7. Avoid getting broken code

安装 Hudson

从https://hudson.dev.java.net/下载并按照指示安装

这里的快速起步手册是一个非常有用的安装方面的资源，它展示了如何为你的 Hudson 用户设置访问控制。

安装插件

安装适用于你环境的下列插件。Hudson 做了对你来说所有“重量级”的事。你可以从用户界面，点击“管理 Hudson”，然后”管理插件”。点击 “可用”标签，选择你想要的插件，然后点击安装。

- Mercurial 或者一些其他版本的控制插件

- xUnit ， phpunit 的测试结果
- Clover ，phpunit 的测试覆盖率报告
- PMD ，phpunit 的 PMD 报告
- Checkstyle ， PHP Codesniffer 的报告.

安装一个新的工作

点击“新的工作”，命名它。并且选择“制作一个免费样式的软件项目”。根据你的环境安装你的工作。一旦你创建了一个新的工作，像下面章节里的详细介绍那样设置一个制作脚本。一旦制作运行正常，到新的工作的“配置”链接，并且指出各个相关插件到 ant 脚本中定义的输出的位置。

每个 KO3 应用程序就一个工作

Kohana 需要每个应用程序就配置一个 Hudson 工作。将他们分隔开的主要原因是因为 phpunit 由于常量和类重定义问题而不能跨应用程序来运行。Kohana 没有被设计用来在每个 PHP 脚本执行时运行多种应用程序。这意味着我们不能跨应用程序运行 phpunit。这意味着我们需要将不同制作的工作中的报告，文档等分开。

为你的项目创建一个 build.xml ant 脚本

Hudson 工作本质上是一个用来为你的 Ant 脚本和处理结果揭开序幕的批处理作业。这是一个基于 PHP 项目的 Kohana3 的 ant 脚本示例。它包含了单元测试，代码嗅探，PMD，以及本件

```
<project name="Demonstration" default="build" basedir=".">
  <target name="build" depends="clean, phpunit, phpdoc, phpcs" />

  <!—安装环境 -->
  <property name="ko3_basedir" value="/usr/local/kohana" />
  <property name="project_name" value="demo" />

  <!—为以后的构建初始化环境 -->
  <target name="init">
    <!-- Create the build directories -->
    <mkdir dir="${basedir}/target/logs" />
    <mkdir dir="${basedir}/target/apidocs" />
    <mkdir dir="${basedir}/target/coverage" />
  </target>

  <!—清理以前的版本 -->
  <target name="clean">
    <delete><fileset dir="${basedir}/target/logs" includes="*/**" /></delete>
```

```
    <delete><fileset dir="${basedir}/target/coverage" includes="*/**"/></delete>
    <delete><fileset dir="${basedir}/target/apidocs" includes="*/**" /></delete>
  </target>

  <!—单元测试应用 -->
  <target name="phpunit">
    <exec dir="${basedir}/" executable="phpunit" failonerror="true">
      <arg line=" --log-xml ${basedir}/target/logs/phpunit.xml
                  --log-pmd ${basedir}/target/logs/phpunit.pmd.xml
                  --coverage-clover ${basedir}/target/coverage/clover.xml
                  --coverage-html ${basedir}/target/coverage
                  --bootstrap=${project_name}/docroot/index.php
${ko3_basedir}/modules/unittest/tests.php" />
    </exec>
  </target>

  <!—生成应用和所有模块的 php 文档 -->
  <target name="phpdoc">
    <exec executable="phpdoc" dir="${basedir}/">
      <arg line="--filename ${project_name}/*.php,modules/*.php
                 --customtags type
                 --target ${basedir}/target/apidocs" />
    </exec>
  </target>

  <!—  生成应用和所有模块的 checkstyle
       - 始终为 codesniffer 任务申报可选的@error 属性.
         否则所有的错误将被记录在 checkstyle.xml 文件,
         这个结果在一个无效的 xml 文档中.
  -->
  <target name="phpcs">
    <exec executable="phpcs"
          dir="${basedir}"
          error="/dev/null">
          <arg line=" --report=checkstyle
                      --report-file=${basedir}/target/logs/checkstyle.xml
                      --tab-width=4
                      ${basedir}/ ${basedir}/modules"/>
    </exec>
  </target>
</project>
```

Kohana 的命令行 CLI 用法

[返回目录]
从命令行调用一个 Kohana 版本 3 的控制器 URI，你需要使用 –uri 参数。
例如

```
php index.php --uri=welcome/index
```

这也是一个有效的调用

```
http://example.com/index.php/welcome/index

或者

http://example.com/welcome/index
```

更多资源

更多高级的 CLI 用法，检查

- http://github.com/Wouterrr/mangoQueue/tree/master/classes/controller/ - mangoQueue，一个写在 Kohana 中的队列系统，使用 MongoDB

整合 Xajax

[返回目录]

Xajax 是一个很方便的 PHP ajax 库，它可以简化一些 ajax 调用（相较于原来的 jQuery 示例），

你可以在http://www.xajaxproject.org 找到它

下载

注意：如果你从http://xajaxproject.org/en/download/网站上下载它，它将包含一些**不推荐使用**的警告。根据你的 PHP 配置，Kohana 可能将它们作为异常抛出，而使得你的应用程序没有用。

一个可用的干净版本可以在 http://sourceforge.net/scm/?type=svn&group_id=139736 上找到

```
svn co https://xajax.svn.sourceforge.net/svnroot/xajax xajax
```

复制到 Kohana 目录结构

*复制 xajax/xajax_core, xajax_controls 和 xajax_plugins 目录(从你解压或 unzip 获得)到 modules/vendors 目录。那么你的目录结构看起来像这样。

```
application
modules
  vendors
    xajax
      xajax_core
      xajax_controls
      xajax_plugins
system
```

*复制 xajax/xajax_js 到你的 javascript 目录

```
application
media
  js
    xajax_js
modules
system
```

如果你从 sourceforge.net 解压出最新的代码，整合 xajax(0.5 后)

整合到 Kohana

* application/bootstrap.php

```
...
  Kohana::modules(array(
      ...
      'vendors'       => MODPATH.'vendors',       // 增加 vendors 目录
      ...
      ));
...
```

* application/classes/controller/base.php（假设这是你所有控制器的父类控制器）

```
abstract class Controller_Base extends Controller_Template {
    ...
        protected $ajax_functions;   // ajax 函数数组
    ...

    /**
     * After 函数
     */
    public function after()
    {
        if ($this->auto_render)
        {
            ...
            $this->_initialize_ajax_functions();   // 调用 ajax 初始化方法
            ...
        }
        parent::after();
```

```
        }

        ...

        /**
         * 增加一个能从视图中被调用的 ajax 函数 调用
         */
        protected function add_ajax_function($function)
        {
            if( !is_array($this->ajax_functions) ) $this->ajax_functions = array();
            array_push($this->ajax_functions, $function);
        }

        /**
         * 如果需要，初始化 xajax 对象
         */
        private function _initialize_ajax_functions()
        {
            if( isset($this->ajax_functions) && count($this->ajax_functions)>0 )
            {
                require_once Kohana::find_file('xajax_core','xajax.inc'); // 你将你的 xajax 目录放在哪
里
                $xajax = new xajax();
                foreach($this->ajax_functions as $function)
                {
                    $xajax->register(XAJAX_FUNCTION, array($function, &$this,
'ajax_'.$function));
                }
                $xajax->processRequest();
                // 注意你的 xajax_js 位于何处
                $this->template->ajax_functions = View::factory('template/div/ajax')
                                                    ->set('ajax_functions',
$xajax->getJavascript(Url::base() . 'media/js'));
            }
        }

    }
```

* application/views/template/div/ajax.php

```
    <?php
    echo $ajax_functions;
    ?>
```

* application/views/template/layout/base.php

```
<html>
<head>
    <title><?php echo $title ?></title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <!—这里放 css 文件 ... -->
    <!—这里放 js 文件 ... -->
    <?php echo isset($ajax_functions)?$ajax_functions:'' ?>
</head>
<body>

<?php echo $content ?>

</body>
</html>
```

如何在控制器和视图中使用

现在从任何控制器中使用任何 xajax 函数。你只需要做下面几步

* application/classes/controller/user.php

```
class Controller_User extends Controller_Base
{

    ...

    public $template = 'template/layout/base';

    ...

    public function action_index()
    {
        $this->template->title = 'User Home Page';
        $this->template->content = View::factory('user/index');
        $this->add_ajax_function("test_call");
    }

    ...

    // 前缀 ajax 调用 ajax_ (这只是我使用的用来使操作方便)
    // 如果你想改变，你可以在这里改  _initialize_ajax_functions()
    public function ajax_test_call()
    {
        $response = new xajaxResponse();
```

```
            $message = "TEST CALL TEXT";
            $response->alert($message);
            return $response;
        }

        ...

    } // 结束
```

* application/views/user/index.php

```
    <a href="#" onclick="xajax_test_call();return false;">Test Ajax Alert</a>
```

待办

* 更新传递的参数到远程调用

整合 Xajax(0.5 后)

如果你下载的是稳定版 0.5，请使用整合 Xajax 上的指示

在http://sourceforge.net/scm/?type=svn&group_id=139736 上最新的代码使用插件和最

终类，所以他们不能被继承。

整合到 Kohana

* application/bootstrap.php

```
...
Kohana::modules(array(
	...
	'vendors'	=> MODPATH.'vendors',		// 增加 vendors 目录
	...
	));
...
```

* application/classes/controller/base.php（假设这是你所有控制器的父类控制器）

```
abstract class Controller_Base extends Controller_Template {
	...
		protected $ajax_functions;	// ajax 函数数组
	...

	/**
	 * After
	 */
	public function after()
	{
		if ($this->auto_render)
		{
			...
			$this->_initialize_ajax_functions();	// 调用 ajax 初始化方法
			...
		}
		parent::after();
	}

	...
```

```
        /**
         * 增加一个可以从视图被调用的 ajax 函数调用
         */
        protected function add_ajax_function($function)
        {
            if( !is_array($this->ajax_functions) ) $this->ajax_functions = array();
            array_push($this->ajax_functions, $function);
        }

        /**
         * 如果需要，初始化 xajax 对象
         */
        private function _initialize_ajax_functions()
        {
            if( isset($this->ajax_functions) && count($this->ajax_functions)>0 )
            {
                require_once Kohana::find_file('xajax_core','xajax.inc'); //你将你的xajax目录放在哪里
                $xajax = new xajax();
                $xajax->configure('javascript URI', '/media/js');
                // 如果你有有插件，在这里加载它们 (根除)
                // 所有的 Kohana 自动加载规则和以往一样工作
                // $plugin = new Velocimedia_Xajax_Response_Plugin;
                foreach($this->ajax_functions as $function)
                {
                    $xajax->register(XAJAX_FUNCTION, array($function, &$this,
'ajax_'.$function));
                }
                $xajax->processRequest();
                //注意你的 xajax_js 位于何处
                $this->template->ajax_functions = View::factory('template/div/ajax')
                                                    ->set('ajax_functions',
$xajax->getJavascript());
            }
        }

    }
```

* application/views/template/div/ajax.php

```
    <?php
    echo $ajax_functions;
    ?>
```

* application/views/template/layout/base.php

```
<html>
<head>
    <title><?php echo $title ?></title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <!—这里放 css 文件 ... -->
    <!—这里放 js 文件 ... -->
    <?php echo isset($ajax_functions)?$ajax_functions:'' ?>
</head>
<body>

<?php echo $content ?>

</body>
</html>
```

如何在控制器和视图里使用

现在从任何控制器中使用任何 xajax 函数。你只需要做下面几步

* application/classes/controller/user.php

```
class Controller_User extends Controller_Base
{

    ...

    public $template = 'template/layout/base';

    ...

    public function action_index()
    {
        $this->template->title = 'User Home Page';
        $this->template->content = View::factory('user/index');
        $this->add_ajax_function("test_call");
    }

    ...

    // 前缀 ajax 调用 ajax_ (这只是我使用的用来使操作方便)
    // 如果你想改变，你可以在这里改  _initialize_ajax_functions()
    public function ajax_test_call()
    {
        $response = new xajaxResponse();
```

```
            $response->alert("This is working now!");
            //  调用插件的示例
            //$response->Velocimedia_Xajax_Response_Plugin->add_error_message("Show using ajax notification");
            return $response;
        }

        ...

    } //结束
```

* application/views/user/index.php

```
    <a href="#" onclick="xajax_test_call();return false;">Test Ajax Alert</a>
```

待办

* 更新传递的参数到远程调用