框架简介

YYUC-PHP 框架简介

YYUC-PHP 框架（本站简称 YY 框架）是一个面向自由开发者的框架，只需一个人就可以轻松搞定一个动态网站，或者是小型的信息系统建设。当然这并不是说 YY 框架只适用于小型系统，应用于多人开发的大型项目它同样极具优势。YY 框架的设计初衷就是敏捷开发快速建站，由于是完全开源和免费的，很显然它将是小型企业和个人网络开发者的首选。

YY 框架是一个免费开源的，快速、简单的 PHP 开发框架，我们采取大道至简的原则，并不严格遵循面向对象的开发模式，追求极致的开发效率和运行效率才是我们的初衷。YY 框架遵循 Apache2 开源协议发布，社区的个人和团队为其贡献力量，代码的贡献，审阅，测试每一个步骤都力求完美。

使用 YY 框架，你可以更方便和快捷的开发和部署应用。不仅仅是网络应用，企业级应用开发更是可以从 YY 框架的简单、快速、稳定、高效的特性中受益。

你可以免费使用 YY 框架，遵从 Apache2 开源协议我们允许把你基于 YY 框架开发开源或商业产品并将之发布或销售。

优势所在

Asp，J2ee，.net，php，Ruby on Rails……，到底要选择什么？在从事网络开发的几年里，以上的几种语言的项目都参与设计或开发过。我想通过以下几点简单的筛选帮您找到答案：

1、先进性：很显然 asp 已经不再适应这一要求了。微软也不再主推它，没有先进的程序设计思想的支撑，没有面向对象的优势我想除了上世纪 90 年代开始开发的程序员，已经没有人再会选择他。

2、学习成本：j2ee 的开发首先要掌握 java 的基本程序设计基础，面向对象的思想，mvc 框架结构，开源或者官方的框架平台，ssh(j2ee 开发必修)，ejb，rest 等等等等，百家齐放的框架，累的程序员不亦乐乎。

3、开放性(跨平台)：.net 是需要运行在其平台之上的，复杂的配置选项，常常弄得你不知道到底是业务优先还是配置优先了，当然.net 的最大瓶颈还不在此，他所运行的 Windows 环境的服务效能是无法和 linux 平台相比的。

4、敏捷性：敏捷开发无非强调开发的效率，以最低的人力投入完成设计开发、实现业务需求。曾经看到过处理一个数据表的 crud 操作的代码量的统计调查，java 和 C#大约在 600 行左右(包括配置文件)，php(无框架) 90 行左右，Ruby on Rails 60 行左右。

5、流行性：Ruby on Rails 在国外已经非常流行了，但是我想开发过 rails 的程序员都有这么一个感受，一个功能 Coding 的时间是一个月，那其中的 5 天你要花在环境的搭建和相关插件的安装上。3 天的时间要花在服务器环境的配置上。同样 Rails 和 J2ee 一样很难独立运行在虚拟主机上，对于自由开发者服务器成本也是一笔不小的开支。

6、执行效率：j2ee 更适合大型的企业级开发，因为基本上是局域网，服务器内存是 10G 级别的，所以我们的代码可以一堆再堆，反正不用考虑性能和带宽。放在互联网上 j2ee 就显的那么臃肿和笨重。坦率的讲，这一点上.net 的优势倒也是很明显的，但是谁又会那么笨的首选 Windows 服务器呢。RoR 和 php 都是解释执行的，没有编译这一步骤，可能效率上会有一定影响，但是您不要忘了，他们的底层都是 C，大量的系统函数和扩展插件都是 C 解释编写的，在这种高度封装和扩展下，真正您自己的代码才有多少呢？php 的最大优势是程序不常驻内存，虽然不能像 java 那样轻松的使用像 application 这样的全局对象了，但是 php 的服务内存是不会增长的，请求执行之后就会释放掉，像 j2ee 如果程序设计的不周全，Web Server 内存就会越来越大，直至当机！

YY 框架是一个高效稳定的 PHP 框架，轻量级是她最大的优势。熟悉 Ruby On Rails 的开发人员都知道约定优于配置的原则，这一原则简化了程序开发者的大量配置工作和框架思考。YY 框架很大程度上借鉴并优化了 Rails 的这一设计思想。毫不夸张的讲有了 YY 框架其他的 PHP 框架就显得有些失色。要么全面而笨重，要么简单而不稳定，要么配置复杂难于学习和理解。

很高兴的告诉你，恭喜你找到了 YY 框架，高效的开发会让你将网络程序设计视为生活的一种享受。

特点及性能

YY 框架借鉴了国内外很多优秀的框架和模式，并在这些设计思想的基础上进行了优化和改进，遵循开发运行一切从简的原则，用强大的框架核心支撑基于 YY 框架的业务代码，一部分重要的特性和优点列举如下：

智能路由 MVC 架构：

YY 框架采用智能路由架构模式。利用这一的匹配模式，开发人员可以方便的设计系统 URL 规则，不再需要定义单一的控制器类，而是采用传统的目录结构处理用户请求，与传统的 php 开发做到了很好的技术过度。 项目层次直观明了。

预编译机制：

独有的模式转换功能，一键切换开发模式和生产模式。根据不同的模式采取不同的编译方案。特有的 JS，CSS 等文本文件的压缩机制和 GZip 压缩输出机制，最大效率的利用网络带宽。

自动类库导入：

YY 框架的所有插件库均采用动态自动加载的方案进行按需加载，开发者无需手动引入，提高开发效率。

ORM 和验证统一：

简洁轻巧的 ORM 实现，配合简单的 CURD 操作接口让开发效率得到极大的提高，更有前后台统一验证规则，让开发人员无需重复劳动，JS 验证和 PHP 入库验证无缝衔接。

优化查询语言：

内建丰富的查询机制，包括组合查询、快捷查询、复合查询、区间查询、统计查询、定位查询、多表查询、子查询、动态查询和原生查询，让你的数据查询简洁高效。

动态和静态模型互补：

无需创建任何对应的模型类，轻松完成 CURD 操作。对于负载的 DB 模型处理，开发这可以采用自动生成的静态模型，进行虚拟字段的填充与回写，数据有效性的验证，模型功能的扩展等等。

高效模板引擎：

YY 框架自建的模板编译执行引擎，最大限度的贴合原生 php 写法，让初学者更容易上手，高级开发人员更灵活应用。

前后台统一 AJAX 规则：

前后台 ajax 互动高度统一，三行 js 代码和一行 php 代码就可以完成 ajax 异步请求操作，极大的提高了开发效率和用户体验。

国际化支持：

先进易读的 yaml 标记语言，用户语言自动检测和默认指定。全站国际化和部分国际化并存，缓存国际化识别引擎为开发者提供极速的多语言网站开发效率。

缓存机制：

系统支持跨服务器缓存，数据检测机制缓存，时间过期机制缓存，基本文件缓存等多种缓存规则，最大限度的利用缓存功能减少服务器开销。

多网站并存机制：

即使你用的是一个虚拟主机，在服务商不允许架设多个网站，YY 框架可以通过内置的路由规则为你解决这一问题。

多库并存机制：

YY 框架集成多库集成调用方式，内置了分布式数据库的支持可以通过程序切换数据库连接，不人为指定的情况下自动实现主从式数据库的读写分离等（注意：主从数据库的数据同步工作不在框架实现，需要数据库考虑自身的同步或者复制机制。）。

简明教程

基础准备

不懂 php 开发环境搭建的可以去网上搜索下。相信学习这个框架的童鞋们也都不是菜鸟级的了。 我习惯开发中直接用实际要发布的域名来做测试，这样将来发布后会省去一些不必要的麻烦。 假如你的程序将来要发布到域名 www.test.com 上，这里简单介绍一下方法：

1、修改本机 host ，把你要测试开发的网络地址指向本地：

用文本编辑器打开：C:\Windows\System32\drivers\etc\host 文件

最后一行添加：

```
127.0.0.1 www.test.com
```

这样从你本机的浏览器请求网址 www.test.com 都会被解析到你的本机 ip：127.0.0.1 上。

2、打开 apache 的 Virtual Hosts 配置文件 建立添加虚拟网站文件映射。

```
<VirtualHost *:80>
    ServerAdmin mqqkobe@163.com
    DocumentRoot "D:/php/test/pub"
    ServerName www.test.com
    ErrorLog "logs/dummy-host.somenet.com-error.log"
    CustomLog "logs/dummy-host.somenet.com-access.log" common
<Directory "D:/php/test/pub">
        Options FollowSymLinks
        AllowOverride All
```

```
        Order allow,deny
        Allow from all
</Directory>
</VirtualHost>
```

3、根据上面的配置可以看出，需要把工程 **test** 的开发目录拷贝到 D:/php 下

4、默认情况下把框架的参考系统源码目录也拷贝到 D:/php 下

开发工具

这里推荐 eclipse 的 php 开发工具（PDT），笔者是从 java 开始接触程序开发的所以推荐 eclipse 下载地址：http://eclipse.org/pdt/downloads/ 一般下载 all in one 就好。

本人本地配的域名是框架网站的主域名：

```
127.0.0.1 www.yyuc.net
```

特别说明:

1、因为示例中配置的域名是 www.yyuc.net，所以下文所提到 www.yyuc.net 的地方你都需要自换成自己的域名。

2、为了让你更了解 YY 框架的原理和构成，我们先介绍几个简单的示例再介绍开发管理中心的使用，虽然开发管理中心会帮助我们省去一些敲代码的工作量，但是作为初学者，还是建议你一步一步的往下看。

Nginx 的配置：

linux 下我们更常用 nginx 来代替 apache 完成页面请求转发的工作，下面是在一个简单的 nginx 配置示例：

```
user  www www;

worker_processes 1;

error_log  /home/wwwlogs/nginx_error.log  crit;

pid        /usr/local/nginx/logs/nginx.pid;

#Specifies the value for maximum file descriptors that can be opened by this process.
worker_rlimit_nofile 51200;

events
```

```
    {
        use epoll;
        worker_connections 51200;
    }

http
    {
        include       mime.types;
        default_type  application/octet-stream;

        server_names_hash_bucket_size 128;
        client_header_buffer_size 32k;
        large_client_header_buffers 4 32k;
        client_max_body_size 50m;

        sendfile on;
        tcp_nopush     on;

        keepalive_timeout 60;

        tcp_nodelay on;

        fastcgi_connect_timeout 300;
        fastcgi_send_timeout 300;
        fastcgi_read_timeout 300;
        fastcgi_buffer_size 64k;
        fastcgi_buffers 4 64k;
        fastcgi_busy_buffers_size 128k;
        fastcgi_temp_file_write_size 256k;

        gzip on;
        gzip_min_length  1k;
        gzip_buffers     4 16k;
        gzip_http_version 1.0;
        gzip_comp_level 2;
        gzip_types       text/plain application/x-javascript text/css application/xml;
        gzip_vary on;

        #limit_zone  crawler  $binary_remote_addr  10m;

        #log format
        log_format  access  '$remote_addr - $remote_user [$time_local] "$request" '
             '$status $body_bytes_sent "$http_referer" '
             '"$http_user_agent" $http_x_forwarded_for';
```

```
server
    {
        listen          80;
        server_name  www.yyuc.net;
        index  index.html  index.htm  index.php;
        root   /home/test/pub;

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

        location ~ .*\.(php|php5)?$
            {
                try_files $uri =404;
                fastcgi_pass   unix:/tmp/php-cgi.sock;
                fastcgi_index  index.php;
                include  fcgi.conf;
            }

        location /status {
            stub_status on;
            access_log    off;
        }

        location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
            {
                expires        30d;
            }

        location ~ .*\.(js|css)?$
            {
                expires        12h;
            }

        access_log   /home/wwwlogs/access.log   access;
    }
}
```

hello world

功能需求：

输入地址 http://www.yyuc.net/demo/hello.html，页面显示 hello wolrd 字符。

通过阅读和学习通用简单路由，你会知道这个请求页面的控制器文件是：

```
1. controller/demo/hello.php。
```

在 controller 文件夹下建立 demo 目录和 hello.php 文件。

方式 1：

编辑 hello.php 代码如下：

```
1. <?php
2. Page::ignore_view();
3. Response::write("hello  world");
4. ?>
```

其中 page 类是对页面的一个封装类，里面有一系列的静态参数和方法供控制器直接修改和调用。

page::$need_view 默认为 true，执行完这个 php 文件之后框架会继续加载它对应的视图文件来执行，Page::ignore_view()将其设为 false 则执行完 php 文件后就不再寻找视图文件了。

Response::write 方法是向客户端进行文本输出，执行后立即退出脚本。

方式 2：

hello.php 文件不写任何代码，可以建立空文件：controller/demo/hello.php。

建立文件：view/default/demo/hello.html

hello.html 内容为：

```
1. <h1>hello  World</h1>
```

由此可见，如果没有执行 Page::ignore_view()，框架执行了 hello.php 文件之后，控制器会自动寻找视图文件 hello.html 文件加载执行。

方式 3：

修改配置文件 **conf.php** 将$auto_find_view 改为 true。

```
/**是否开启无控制器时自动寻找对应视图~默认:false*/
public static $auto_find_view = true;
```

无需创建控制器文件直接建立文件：view/default/demo/hello.html

内容为：

```
<h1>hello World</h1>
```

配置数据库

配置数据库连接

这只是一个标准示例，实际开发中并不一定一定按照示例的方式进行。 框架的主配置文件是 /yyuc/conf.php，它是一个被封装好的静类文件，有关数据库的配置如下：

```
/**数据库地址~*/
public static $db_host = "localhost";
/**数据库端口~*/
public static $db_port = "3306";
/**数据库名~*/
public static $db_dbname = "test";
/**数据库用户名~*/
public static $db_username = "root";
/**数据库密码~*/
public static $db_password = "";
/**数据库表前缀~*/
public static $db_tablePrefix = "qq_";
```

建立数据库和表

可以通过自己常用的 mysql 管理工具完成这一工作。这里我们建立的表名称是：qq_notes

DDl 语句如下：

```
CREATE TABLE `qq_notes` (
  `id` int(11) NOT NULL auto_increment COMMENT '主键',
  `author` varchar(255) default NULL COMMENT '作者',
  `theme` enum('Arts','Emotion','Humanities','Technology') default NULL COMMENT '主题:艺术,情感,人文,科技',
  `title` varchar(255) default '新建题目' COMMENT '标题',
  `content` text COMMENT '内容',
```

```
    `bepublished` tinyint(1) default NULL COMMENT '是否发布',
    `postdate` int(11) default NULL COMMENT '提交时间',
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

关于数据表的创建有以下几点说明：

1、像 Rails 一样,如果要通过面向对象的方式操作，YY 框架要求如果表要依据 Model 类操作必须有一个名为"id”的 int 类型的自增主键。

2、建议所有字段都要有注记，一是易于表的维护和管理，二是在自动代码生成过程中减少生成后的代码的后期修改量，实现根据注记内容动态修改字段描述的功能。

3、对于布尔类型，用 tinyint(1)表示，0 代表：否，1 代表：是。

下面两条可以根据开发者的喜好采用：

1、对于日期类型和日期时间类型，用 int(9)表示，php 开发中因为 time()和 date()方法的存在大多数开发者喜欢用数据库的 int 类型表示时间而放弃了 date 和 datetime。

2、根据喜好可以对于枚举类型，通常页面上会用下拉框或单选按钮的形式与其关联，所以在枚举字段的注记中采用",”号隔开的方式对每一个枚举项进行描述，":”号之前的文本代表该字段的描述。这样在代码生成后这些注释会自动关联到下拉框的 text 上。如果只是有注解而没有针对每一项的描述，则每项的值和 text 是一样的。

以上的数据库设计规则是 YY 框架约定的，当你按照这些规则设计数据表时，你的开发工作量将大大降低

访问路径设计

访问设计

要完成基本的增删改查工作，通常要有以下几个页面：

1. 1) 单个项目的新增页面
2. 2) 多个项目的列表页面
3. 3) 单个项目的详细信息页面
4. 4) 单个项目的修改页面

这里按照管理我们的 URL 设计如下

单个项目的新增页面 ---> www.yyuc.net/notes/creat.html

多个项目的列表页面---> www.yyuc.net/notes/index.html (也可已访问：www.yyuc.net/notes/)

单个项目的详细信息页面---> www.yyuc.net/notes/show-?.html(其中的问号代表项目 ID)

单个项目的修改页面--->www.yyuc.net/notes/edit-?.html(其中的问号代表项目 ID)

开发顺序

我们按照：新增，列表，详细，修改的顺序进行逐个页面的程序开发。

新增页面展示

创建控制器文件：controller/notes/creat.php 对应新增的控制器。

creat.php 内容如下：

```
<?php
$note = new Model('notes');
?>
```

这句话的意思是依照表：qq_notes 创建模型文件。

为什么传入的参数中"qq_"没有写呢，因为为了一库多用，我们在数据库配置中将"数据库表前缀"设置为了:qq_。那么对于程序开发来说，所有的对表 qq_notes 的描述都要用到他除去前缀之后的名字：notes。

创建视图文件：view/default/notes/creat.html 对应新增的视图。

creat.html 内容如下：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试</title>
</head>
<body>
<form action="" method="post">
标题：{$note->text('title')}<br/>
作者：{$note->text('author')}
主题：{$note->select('theme')}<br/>
发表时间：{$note->date('postdate')}
是否发布:{$note->checkbox('bepublished')}<br/>
内容：<br/>
{$note->textarea('content')}<br/>
<button type="submit">提交</button>
</form>
</body>
</html>
```

好了，新增页面的展示工作就完成了，可能你会对页面上的这些标签感到奇怪，甚至会想："哎呀，又是该死的视图标签，要学这个框架就要掌握这些烦人的标签了"。

不急让我们先访问以下这个页面，之后你就会发现，YY 框架的标签是这么的易学和神奇。

输入网址：http://www.yyuc.net/notes/creat.html，我们可以看到页面展示出来。

因为标题(title)字段的默认值是:"新建标题"，所以该项内容会自动填充，让我们来测试以下,我们把数据库中**主题（theme）**字段的默认值修改为："Humanities"：

刷新下页面你会发现**主题**字段的默认值随之改变了。

关于模板标签的说明：

对于在控制器中定义的模型变量(如上例中的"$note")，可以在对应的视图中展示针对不同字段的相应的标签：

表单项	对应 html	备注
all	<textarea></textarea>...	模型所有非空属性的表单集合(隐藏的)
text	____	文本输入
password		密码框
email	<input type="email"/>	邮件输入框(html5)
range	<input type="range"/>	程度选择框(html5)
hidden	<input type="hidden"/>	隐藏标签
textarea	<textarea></textarea>	文本框
checkbox	☐	多选按钮
select	<select>...</select>	下拉框

表单项	对应 html	备注
radio	☐	单选按钮
date	____	时间选择
datetime	____	时间日期选择
texteditor	<textarea></textarea>	富文本编辑器
color	____	颜色选择框
vercode	____	验证码输入框
upload	<input type="file"/>	文件上传

因为只是基本的增删改查功能的实现，并不涉及到自定义的数据和页面样式的问题，所以详细的使用方法参考模型类 API。

鼠标点击"发表时间"文本框时，会有日期选择框弹出，这是因为框架会对每个页面自动引入 Jquery 库和基于 Jquery 的框架适配器。视图页面中你不需要添加一行 js 代码就能实现这一功能。

你可能还会有疑问，前面提到的数据库设计中的注记在这个页面中并没有体现出来，好吧，我们把 creat.html 的内容改成这个样子：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试</title>
</head>
<body>
<form action="" method="post">
{$note->lable('title')}：{$note->text('title')}<br/>
{$note->lable('author')}：{$note->text('author')}
{$note->lable('theme')}：{$note->select('theme')}<br/>
{$note->lable('postdate')}：{$note->date('postdate')}
{$note->lable('bepublished')}:{$note->checkbox('bepublished')}<br/>
{$note->lable('content')}：<br/>
```

```
{$note->texteditor('content')}<br/>
<button type="submit">提交</button>
</form>
</body>
</html>
```

刷新页面显示效果和之前的是一样的。

新增信息保存

修改控制器文件：controller/notes/creat.php 。

```
<?php
$note = new Model('notes');
if(Request::post()){
    //如果有 post 信息 则认为是新增后的 Form 提交
    //单纯的 post 信息判断是不安全的 因为没有具体的字段要求和判断所以可以这样写
    $note->load_from_post();
    $note->save();
}
?>
```

访问：http://www.yyuc.net/notes/creat.html，页面上填入一些测试数据点击"提交" 。

这样各个字段的信息就会自动的保存到数据库中了。

信息列表展示

用户访问时默认展示列表页，所以我们建立的列表页面的控制器名称为 index.php。

创建控制器文件：controller/notes/index.php 对应列表展示的控制器。

index.php 内容如下：

```
<?php
$note = new Model('notes');
$notes = $note->list_all();
?>
```

创建视图文件：view/default/notes/index.html 对应列表展示的视图。

index.html 内容如下：

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试-列表</title>
<style type="text/css">
</style>
</head>
<body>
<table>
<tr>
<th>{$note->lable('title')}</th>
<th>{$note->lable('author')}</th>
<th>{$note->lable('theme')}</th>
<th>{$note->lable('postdate')}</th>
</tr>
{loop $notes as $n}
<tr>
<td>{$n->title}</td>
<td>{$n->author}</td>
<td>{$n->field_text('theme')}</td>
<td>{date('Y-m-d',$n->postdate)}</td>
</tr>
{/loop}
<tr></tr>
</table>
</body>
</html>
```

对于 **theme** 字段，因为存储为枚举类型，而枚举的值不是最终要显示的值，所以调用 field_text 方法展示要调用的文本。

浏览器输入：http://www.yyuc.net/notes/,一个信息列表页面就展现在你面前了。

此时，我们再修改下新增页面，使新增完成后自动跳转到列表页

creat.php 内容改为：

```
<?php
$note = new Model('notes');
if(Request::post()){
    //如果有 post 信息 则认为是新增后的 Form 提交
    //单纯的 post 信息判断是不安全的 因为没有具体的字段要求和判断所以可以这样写
    $note->load_from_post();
    $note->save();
    Redirect::to('index');
```

```
}
?>
```

通过调用框架的 Redirect::to 方法而不是直接的视图引用，可以有效的屏蔽了 F5 刷新引起的重复提交问题。

信息详细页面

创建控制器文件：controller/notes/show.php 对应详细信息展示的控制器。

show.php 内容如下：

```
<?php
$note = new Model('notes');
$note->find(get(1));
?>
```

创建视图文件：view/default/notes/show.html 对应详细信息展示的视图。

show.html 内容如下：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试-详细信息</title>
<style type="text/css">
</style>
</head>
<body>
<form action="" method="post">
{$note->lable('title')}：{$note->title}<br/>
{$note->lable('author')}：{$note->author}<br/>
{$note->lable('theme')}：{$note->field_text('theme')}<br/>
{$note->lable('postdate')}：{date('Y-m-d',$note->postdate)}<br/>
{$note->lable('bepublished')}:{$note->field_text('bepublished')}<br/>
{$note->lable('content')}：<br/>
{$note->content}<br/>
</form>
</body>
</html>
```

浏览器输入：http://www.yyuc.net/notes/show-1.html

信息删除

修改列表展示页面，让每条信息点击后进入相应的详细页面，并添加删除按钮和编辑按钮。

index.html 内容改为：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试-列表</title>
<style type="text/css">
</style>
</head>
<body>
<table>
    <tr>
        <th>{$note->lable('title')}</th>
        <th>{$note->lable('author')}</th>
        <th>{$note->lable('theme')}</th>
        <th>{$note->lable('postdate')}</th>
        <th>删除</th>
        <th>编辑</th>
    </tr>
    {loop $notes as $n}
    <tr>
        <td><a href="show-{$n->id}.html">{$n->title}</a></td>
        <td>{$n->author}</td>
        <td>{$n->field_text('theme')}</td>
        <td>{date('Y-m-d',$n->postdate)}</td>
        <td><a href="javascript:;" onclick="if(confirm('确定要删除吗？')){ goto('delete-{$n->id}.html');}">删除</a></td>
        <td><a href="edit-{$n->id}.html">编辑</a></td>
    </tr>
    {/loop}
    <tr></tr>
</table>
</body>
</html>
```

关于模板标签的说明：

1、 对于视图模板中的 JS 方法，因为同样是要包含大括号"{}”的，所以在模板解释中会被解析器误认为是 PHP 模板标签，解决的方法是对于 JS 方法”{”之后要紧跟空格或者回车，这样解释器就不会将其解释成 php 语言。

2、 因为 IE 的某些问题，当要用到 JS 跳转时要调用框架中的 goto 方法，请不要采用传统的 location.href= ? 的方式。这样控制器中的 Redirect::back()方法才会生效。

创建控制器文件：controller/notes/delete.php 对应信息删除的控制器。

delete.php 内容为：

```
<?php
if (isset($_GET[1])){
    //指定要操作的模型 id 删除之
    $note = new Model('notes');
    $note->id($_GET[1]);
    $note->remove();
}
//返回请求前的页面
Redirect::back();
?>
```

信息修改页面

创建控制器文件：controller/notes/edit.php 对应每条数据的修改页面。

edit.php 代码如下：

```
<?php
$note = new Model('notes');
if(get()){
    //如果存在 get 提交的信息
    $note->find(get(1));
}
?>
```

创建视图文件：view/default/notes/edit.html 对应修改展示的视图。

edit.html 内容如下：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8"http-equiv="content-type">
```

```
<title>CRUD 测试</title>
<style type="text/css">
</style>
</head>
<body>
<form action="creat.html"method="post">
{$note->lable('title')}：{$note->text('title')}<br/>
{$note->lable('author')}：{$note->text('author')}
{$note->lable('theme')}：{$note->select('theme')}<br/>
{$note->lable('postdate')}：{$note->date('postdate')}
{$note->lable('bepublished')}:{$note->checkbox('bepublished')}<br/>
{$note->lable('content')}：<br/>
{$note->texteditor('content')}<br/>
<button type="submit">提交</button>
{$note->hidden('id')}<br/>
</form>
</body>
</html>
```

浏览器输入：http://www.yyuc.net/notes/edit-1.html，进行相应的数据修改，点击"提交" ，此条信息就会被更新。

你会发现 edit.html 和 creat.html 基本一致，有以下两处不同：

1、creat.html 的 form 的 action 为空这样默认是当前页面，也就是 creat.html。而同样的 edit.html 的 form 的 action 也为 creat.html。也就是说两个页面提交的信息都是通过 creat.php 处理的。$note->save();既可以新增，也可以更新，框架是通过判断模型中是否定义了主键 ID 来区分的。

2、edit.html 比 creat.html 多了一行代码{$note->hidden('id')}这是隐藏的数据的主键 id 的提交信息。其实在 creat.html 中也是可以加上{$note->hidden('id')}这句的，因为进入视图 creat.html 之前$note 的 id 并未被赋值，自然提交请求之后，控制器还是会新增一条数据的。 这样两个视图就变的一模一样。索性，我们删掉 creat.html。然后将 creat.php 的代码改为：

```
<?php
$note = new Model('notes');
if(Request::post()){
    //如果有 post 信息 则为新增后的 Form 提交
    $note->load_from_post();
    $note->save();
    Redirect::to('index');
}
Page::view('edit');
?>
```

Page::view('edit')的意思是说，这个控制器的视图改为同级目录下的 edit.html。

不过，实际的开发中，两个视图总会有些差别的，所以为了减少代码量和通用性就需要将通用的部分抽离出来作为单独的模板供其他视图引用。

视图模板的引用

读到这里你会发现所有页面并没有导航条，这样只能靠直接输入地址栏或者回退按钮来做页面跳转，让我来做一个通用的导航条模板让所有视图都引用他，这样各页面的跳转就容易多了。 创建视图文件：view/default/notes/navigation.html 对应导航条视图。

navigation.html 内容如下：

```
<span>
<a href=" /notes/">首页(列表页)</a>|
<a href="creat.html">新增页面</a>|
<a href="javascript:;" onclick="history.go(-1)">后退</a>
</span>
<br/>
```

在所有视图文件的”body”标签下方都引入此视图，引用代码是：

```
{T navigation}
```

这样所有页面之前就跳转自如了。 详细了解各种路径的引用方式请参阅视图引用的说明。

分页功能的实现

YY 框架的分页原理借鉴了 Rails 框架的 kaminari 插件，即方便调用又不失个性化。详细的调用规则请参见分页类的 API。下面做一个简单的分页展示。

index.php 内容修改如下：

```
<?php
$note = new Model('notes');
$pagination = new Pagination(8, 7);
$notes = $pagination->model_list($note);
?>
```

Pagination(8, 7)的构造方法的意思是每页显示 8 条记录，每页允许出现的最多分页页面的链接数为 7。

model_list 方法传入要被分页的模型，模型将查询的工作交由分页控制类执行。

index.html 内容修改如下：

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type">
<title>CRUD 测试-列表</title>
<style type="text/css">
</style>
</head>
<body>
{T navigation}
<table>
    <tr>
        <th>{$note->lable('title')}</th>
        <th>{$note->lable('author')}</th>
        <th>{$note->lable('theme')}</th>
        <th>{$note->lable('postdate')}</th>
        <th>删除</th>
        <th>编辑</th>
    </tr>
    {loop $notes as $n}
    <tr>
        <td><a href="show-{$n->id}.html">{$n->title}</a></td>
        <td>{$n->author}</td>
        <td>{$n->field_text('theme')}</td>
        <td>{date('Y-m-d',$n->postdate)}</td>
        <td><a href="javascript:;" onclick="if(confirm('确定要删除吗？')){ goto('delete-{$n->id}.html');}">删除</a></td>
        <td><a href="edit-{$n->id}.html">编辑</a></td>
    </tr>
    {/loop}
    <tr></tr>
</table>
<center>
{P}
</center>
</body>
</html>
```

{P}标签是通用分页标签的标识。 更多关于分页的详细信息，请参阅视图中的分页

多插入几条数据，浏览器访问 http://www.yyuc.net/notes/你会看到分页效果，如此的简单不是吗？

url 路由规则

通用简单路由

YY 框架遵循简单路由约定，按照约定实现页面路由控制非常简单。

本节通过几个简单的示例让你你很快了解到 YY 框架的默认 URl 简单路由规则。

假设你网站的域名是 http://www.yyuc.net：

访问地址	控制器	视图
http://www.yyuc.net	controller/index.php	view/**default**/index.html

1、这个请求指向网站首页，它会跳转到你在 conf.php 文件中中配置的首页面。不特殊指定则会按照上表中的方式跳转。
2、其中的 **default** 是你在/yyuc/conf.php 中指定的模板库，你也可以指定到其他的视图模版文件夹，这样你只需要改动下配置 你的网站展现就会焕然一新。

访问地址	控制器	视图
http://www.yyuc.net/login.html	controller/login.php	view/default/login.html

也可以在控制器中自由指定视图如： 让此次请求跳转到 **view/default/test.html** 视图等等...

访问地址	控制器	视图
http://www.yyuc.net/user/list.html	controller/user/list.php	view/default/user/list.html

访问地址	控制器	视图
http://www.yyuc.net/user/list-ad.html	controller/user/list.php	view/default/user/list.html

要说明的是： http://www.yyuc.net/user/list-ad.html 相当于通常意义上的 http://www.yyuc.net/user/list.html?1=ad 在/yyuc/controller/user/list.php 中你可以通过 Request::get(1) 得到它。 如： $name = Request::get(1); 则 $name 的值为:"ad"。

访问地址	控制器	视图
http://www.yyuc.net/user/list-ad-qq.html	controller/user/list.php	view/default/user/list.html

和上一路径类似： http://www.yyuc.net/user/list-ad.html 相当于通常意义上的 http://www.yyuc.net/user/list.html?1=ad&2=qq
在/yyuc/controller/user/list.php 中你可以通过如： $name0 = Request::get(0); $name1 = Request::get(1); $name2 = Request::get(2); 则$name0 的值为:"list" $name1 的值为:"ad"，$name2 的值为:"qq"。
其他类似的请求参数以此类推。例如：/list-ad-qq-name-yyuc.html。

分页以"_"作为分割参数

访问地址	控制器	视图
http://www.yyuc.net/user/list-ad-qq_1.html	controller/user/list.php	view/default/user/list.html

注意 http://www.yyuc.net/user/list-ad-qq.html 等同于 http://www.yyuc.net/user/list-ad-qq_1.html 即为第 1 页，http://www.yyuc.net/user/list-ad-qq_2.html 则为第 2 页。

路径补充

形如 http://www.yyuc.net/user/list.html?name=aaa 的请求在后台通过 Request::get('name')方法也是可以获取到值的，如何传参开发中根据实际情况而定，只是出于 SEO 的友好考虑我们建议用"_"标识 get 请求参数。

自定义路由

若想实现复杂的路由，你需要掌握重写路由的方式。 在 YY 框架中我们提倡使用简单路由进行开发。

简单路由会大大减少开发过程中的配置工作量，代码易读且易于维护。 但是有的时候为了减少代码量，你可能需要把很多请求都指向一个控制器，或者干脆所有请求都指向一个特定的控制器，这个时候你就需要掌握自定义路由了。

YY 框架中自定义路由是非常简单的，只需修改配置文件中的路由配置就好，下面详细的为你介绍。

在框架内部定义了一个特殊的路由替换解析器以满足自定义路由的需求：

绝对路由规则

在修改 Conf 类文件的静态变量 routing，例如改为：

```
/**一般的绝对路由规则*/
public static $routing = array(
    'admin/'=>'system/users/admin/',
    'user'=>'system/users/user',
);
```

那么在路由请求中：

用户访问：http://www.yyuc.net/admin/ 等同于访问 http://www.yyuc.net/system/users/admin/

用户访问：http://www.yyuc.net/user.html 等同于访问
http://www.yyuc.net/system/users/user.html

前置路由规则

在修改 Conf 类文件的静态变量 routing，例如改为：

```
/**一般的前置路由规则*/
public static $routing_bef = array(
    'admin/'=>'system/users/admin/',
    'user'=>'system/users/user',
);
```

那么所有 URL 地址中以 admin/开头的请求都会自动跳转到以 system/users/admin/ 开头的请求之中。

用户访问：http://www.yyuc.net/admin/add.html 等同于访问
http://www.yyuc.net/system/users/admin/add.html

用户访问：http://www.yyuc.net/useradmin-3.html?abc=bca 等同于访问
http://www.yyuc.net/system/users/useradmin-3.html?abc=bca

正则表达式的路由规则

有的时候你要匹配的路由内容可能不只是单单的固定记录，这个时候你就要用到正则表达式了：

```
/**正则表达式的路由规则*/
public static $routing_reg = array(
    //新闻入口
    '/news\/20\d{2}\/index(_\d+)?$/' => 'news/index',
    '/news\/20\d{2}\/\d+\/\S*$/' => 'news/show',
    '/news\/index(_\d+)?$/' => 'news/index',
```

```
7.  );
```

以上的这段规则是网站新闻入口的规则方式:

用户访问：/news/2012/ 会跳转到 news/index.html

用户访问：/news/2012/index_5.html 会跳转到 news/index_5.html

用户访问：/news/2012/35/abc.html 会跳转到 news/show.html

用户访问：/news/ 会跳转到 news/index.html

注意：
路由匹配的顺序是自上而下的：
先进行一般路由匹配，匹配不通过时如果开启了无控制器自动寻找视图功能则先寻找匹配的视图。
如果仍然无法进行匹配再进行绝对路由规则匹配、前置路由规则和正则表达式的路由规则的匹配，直到找到匹配的控制器。
经过以上步骤还是无法找到匹配的控制器的话，则会跳转的 404 页面。

模型

模型简介

对于一个基于 MVC 模式的框架，模型类是必不可少的，YY 框架的模型分为两种，简单模型(SampleModel)和数据库(Model)模型。

简单模型是对任何数据类型的一个统一封装。我们主张任何数据类型都被封装在模型之中，这样有利于数据的统一校验维护和管理。

数据库模型继承自简单模型，在简单模型的基础上实现了针对数据库表的实例化操作，让开发者对数据库的操作更加便捷。

对于复杂的模型操作，如实现类似 Active Record 的模型操作，开发这可以创建自己的模型类，继承简单模型(SampleModel)或数据库(Model)模型。 在原有模型基础上实现方法的重写和新增。

简单模型

使用入门

假设你要实现一个前台的 form 表单向后台提交数据，提交的数据你并不需要保存在数据库之中。这时候你就可以使用 SampleModel 了。

看下面的例子：

控制器 mmsz.php:

```
1.  //简单模型记录前台密码
2.  $m = new SampleModel('password');
3.
4.  if($m->try_post()){//密码被传入
5.
6.      if($m->new != $m->again){
7.          Session::once('cperr','两次密码不一致！');
8.          return;
9.      }
10.     $admin = new Model('admin');
11.     $admin->id = Session::get('uid');
12.     $admin->password = $m->old;
13.     if(!$admin->is_real(array('id','password'))){
14.         Session::once('cperr','原始密码不正确！');
15.         return;
16.     }
17.     $admin->password = $m->new;
18.     if($admin->save()){
19.         Session::once('cpsuc','密码修改成功！');
20.         $m = new SampleModel('password');
21.     }
22. }
```

视图内容 mmsz.html:

```
1.  <div id="content" style="margin: 0px 0px 0px 0px;">
2.          <!-- content / right -->
3.          <div id="right" style="margin: 0px 0px 0px 0px;">
4.              <!-- table -->
5.              <div class="box" style="margin: 0px 0px 0px 0px;">
6.                  <!-- box / title -->
7.                  <div class="title">
```

```
                    <h5>
                    <a href='javascript:;'>密码修改</a>
                    </h5>
                    <div class="search">
                    <button type="button" onclick="goto_back();">返回</button>
                    </div>
                </div>
                <form action="mmsz.html" method="post"> {tk()}

                <h4 style="color: red">{Session::flush('cperr')}</h4>
                <h4 style="color: blue">{Session::flush('cpsuc')}</h4>
                <table style="width: 100%" width="100%">
                    <tr>
                    <td style="width: 150px;">
                    原始密码:
                    </td>
                    <td>
                    {$m->password('old','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
                    <td style="width: 150px;">
                    新密码:
                    </td>
                    <td>
                    {$m->password('new','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
                    <td style="width: 150px;">
                    密码确认:
                    </td>
                    <td>
                    {$m->password('again','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
                    <td colspan="2" align="center" style="border: none;">
                    <button type="submit" style="float:inherit; display: block;">数据提
交</button>
                    </td>
                    </tr>
                    </table>
            </form>
```

```
51.                 </div>
52.             </div>
53.             <!-- end content / right -->
54.         </div>
```

上例控制器中的前 10 行是简单模型的初始化和信息提交。验证成功才保存到数据库表 admin 中。

admin 表的操作方法请参阅下一节的数据库模型

方法列表

__construct

```
1.  SampleModel::__construct(string $tablename, string $postid, boolean $isorgin)
2.  构造函数
3.
4.  Parameters:
5.  string $tablename 虚拟表名
6.  string $postid 表单提交的区分 ID
7.  boolean $isorgin 是否是通用模型类
```

$postid 用来区分相同表名的模型，区分$postid 之后同一个页面就可以提交多个相同类型的模型了。

all

```
1.  SampleModel::all(string $names)
2.  所有有效的数据都输出 hidden 标签
3.  这是一种页面参数整体传递的简便方式
4.  为了便于灵活覆盖，建议将其放在 Form 的最顶端(tk 方法之后)
5.  把页面信息放在页面中是很不安全的，所以强烈建议只在新增时使用此方法
6.
7.  Parameters:
8.  string $names 字段名称数组
9.  Returns:
10.     string 标签 html 字串
```

elid

```
1.  SampleModel::elid(string $name)
2.  获得页面标签的 id
3.  Parameters:
4.
5.  string $name 字段名称 标签 name
6.  Returns:
7.     string 标签 id
```

elname

```
SampleModel::elname(string $name)
获得页面标签的 name
如果开启了表单令牌此处获得的 name 是经过框架加密的(防止恶意信息提交)

Parameters:

string $name 字段名称 标签 name
Returns:
  string 标签 id
```

field_form_name

```
SampleModel::field_form_name(string $name)
取得属性再 Form 中的 name(未经加密的)

Parameters:
string $name 字段名称 标签 name
Returns:
  string
```

load_from_get

```
SampleModel::load_from_get()
根据 get 请求内容填充这个 Model
这个方法通常用在信息检索页面的批量属性提交
切不可用此方法得来的数据进行 CUD 操作！

Returns:
  Model 模型本身
```

load_from_post

```
SampleModel::load_from_post()
根据 post 请求内容填充这个 Model
这是表单字段自动提交的最常用方法

Returns:
  Model 模型本身
```

try_get

```
SampleModel::try_get()
试探行的填充这个 model 如果能填充则采用 post 填充并返回：true 否则返回：false

```

```
4.  Returns:
5.      boolean
```

try_post

```
1.  SampleModel::try_post()
2.  试探行的填充这个 model 如果能填充则采用 post 填充并返回：true 否则返回：false
3.
4.  Returns:
5.      boolean
```

数据库模型

数据库模型简介

YY 框架的数据库模型是一种 ORM 的提现方式。首先阅读这一小节之前请确保已经阅读过简单模型。

因为数据库模型继承自简单模型，所以简单模型的用法对数据库模型统统适用。

继续上一节中的例子：

控制器 mmsz.php 改为:

```
1.  //简单模型记录前台密码
2.  $admin = new Model('admin');
3.  $admin->id = Session::get('uid');
4.  if($admin->try_post()){//密码被传入
5.
6.      if($admin->new != $admin->again){
7.          Session::once('cperr','两次密码不一致！');
8.          return;
9.      }
10.     if(!$admin->is_real(array('id','password'))){
11.         Session::once('cperr','原始密码不正确！');
12.         return;
13.     }
14.     $admin->password = $admin->new;
15.     if($admin->save()){
16.         Session::once('cpsuc','密码修改成功！');
17.     }
18. }
```

视图内容 mmsz.html:

```
<div id="content" style="margin: 0px 0px 0px 0px;">
        <!-- content / right -->
        <div id="right" style="margin: 0px 0px 0px 0px;">
            <!-- table -->
            <div class="box" style="margin: 0px 0px 0px 0px;">
                <!-- box / title -->
                <div class="title">
                    <h5>
                    <a href='javascript:;'>密码修改</a>
                    </h5>
                    <div class="search">
                    <button type="button" onclick="goto_back();">返回</button>
                    </div>
                </div>
                <form action="mmsz.html" method="post"> {tk()}

                <h4 style="color: red">{Session::flush('cperr')}</h4>
                <h4 style="color: blue">{Session::flush('cpsuc')}</h4>
                <table style="width: 100%" width="100%">
                    <tr>
                    <td style="width: 150px;">
                    原始密码:
                    </td>
                    <td>
                    {$m->password('password','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
                    <td style="width: 150px;">
                    新密码:
                    </td>
                    <td>
                    {$m->password('new','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
                    <td style="width: 150px;">
                    密码确认:
                    </td>
                    <td>
                    {$m->password('again','class="formmid"')}
                    </td>
                    </tr>
                    <tr>
```

```
                    <td colspan="2" align="center" style="border: none;">
                    <button type="submit" style="float:inherit; display: block;">数据提交</button>
                    </td>
                    </tr>
                    </table>
                </form>
                </div>
            </div>
            <!-- end content / right -->
        </div>
```

上例控制器中的操作全部是由数据库模型完成的，其中 new 和 again 字段在正式的数据库表结构中并不存在， 不用担心在执行 **save()**方法时框架会自动过滤掉数据库中不存在的字段，把变化的字段自动更新到数据库中。

save()既可以新增也可以更新，判断的依据是模型的 id 属性，不存在 id 时为新增，存在 id 时为更新。

方法列表

下面列举的方法是数据库模型中特有的，简单模型中列举过的方法此处就不再列举了。

avg

```
Model::avg(string $field, miexd $condition, miexd $pam)
查询并返回某字段的平均值
如果$condition 为数组则根据数组条件返回符合结果的列表
如果$condition 是字串则必须是 where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语句
如果不传入条件则根据 where 方法 的预设参数查询,如果 where 未被调用过则列出所有

Parameters:
string $field 要查询 的字段
miexd $condition 条件字符串或条件数组
miexd $pam 参数数组

Returns:
    integer 平均值
```

count

```
Model::count(string $field)
计算行数

Parameters:
```

```
5.  string $field 统计的参数(*)
6.
7.  Returns:
8.    integer 计数
```

delete

```
1.  Model::delete(mixed $condition, mixed $pam)
2.  批量删除数据
3.  如果不传入条件则自动将这个 Model 的除 id 之外的其他字段属性作为条件
4.
5.  Parameters:
6.  mixed $condition 条件数组或字串
7.  mixed $pam 参数数组
8.
9.  Returns:
10.   mixed 删除成功返回删除的条数 失败返回 null
```

field

```
1.  Model::field(string $select)
2.  设置查询字段 如:"id,name"
3.
4.  Parameters:
5.  string $select 要检索的字段
6.
7.  Returns:
8.    Model 模型本身
```

find

```
1.  Model::find(mixed $id, array $pam)
2.  根据 id 或者数组条件填充这个 model
3.  示例: find(5) 或 find(array('name'=>'mqq','sex'=>'man'))
4.
5.  Parameters:
6.  mixed $id 主键或条件数组
7.  array $pam 参数值的数组
8.
9.  Returns:
10.   Model 模型本身
```

get_a_clone

```
1.  Model::get_a_clone()
```

```
2.  获得一个和此模型一模一样的克隆
3.  不克隆 ID 信息
4.
5.  Returns:
6.    Model  新的模型
```

get_model_array

```
1.  Model::get_model_array(string|array  $fields)
2.  获得模型属性信息的数组形式
3.  只包含数据库中已有的字段  不包含 ID 信息(特殊$fields 指定除外)
4.
5.  Parameters:
6.  string|array  $fields  需要特定指定的字段
7.  Returns:
8.    array  模型的信息数组
```

get_model_array_with_id

```
1.  Model::get_model_array_with_id()
2.  获得模型属性信息的数组形式
3.  只包含数据库中已有的字段  包含 ID 信息
4.
5.  Returns:
6.    array  模型的信息数组
```

has

```
1.  Model::has(miexd  $condition,  miexd  $pam)
2.  查询是否含有符合条件的数据
3.  如果$condition 为数组则根据数组条件返回符合结果的列表
4.  如果$condition 是字串则必须是  where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语
    句
5.  如果不传入条件则根据 where 方法  的预设参数查询,如果 where 未被调用过则列出所有
6.
7.  Parameters:
8.  miexd  $condition  条件字符串或条件数组
9.  miexd  $pam  参数数组
10.
11. Returns:
12.   boolean
```

has_id

```
1.  Model::has_id()
```

```
2.  判断该模型是否含有 ID
3.  查看数据库中是否有独立的一条数据与 model 对应
4.
5.  Returns:
6.    boolean
```

id

```
1.  Model::id(string  $id)
2.  设置这个 Model 的标识 id
3.  只是设置主键字段,不执行实际的 DB 查询操作
4.  一般在更新或删除之前调用
5.
6.  Parameters:
7.
8.  string  $id  主键
9.  Returns:
10.   Model  模型本身
```

is_real

```
1.  Model::is_real(string|array  $fields,  boolean  $fillme)
2.  判断当前 model 是否在数据库中存在真实的对应
3.
4.  Parameters:
5.
6.  string|array  $fields  需要特定指定的字段
7.
8.  boolean  $fillme  是否填充当前 model  默认:true
9.  Returns:
10.   boolean
```

lable

```
1.  Model::lable(string  $field)
2.  获得某一字段的 Lable
3.  默认为数据库中定义的字段注释
4.
5.  Parameters:
6.
7.  string  $field  字段名称
8.  Returns:
9.    string  字段描述
```

limit

```
Model::limit(string $limit)
设置查询区间

Parameters:

string $limit 区间
Returns:
  Model 模型本身
```

list_all

```
Model::list_all(mixed $condition, array $pam)
查询并返回模型结果集
如果$condition 为数组则根据数组条件返回符合结果的列表
如果$condition 是字串则必须是 where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语句
如果不传入条件则根据 where 方法 的预设参数查询,如果 where 未被调用过则列出所有

Parameters:
mixed $condition 条件字符串或条件数组
array $pam 参数数组

Returns:
  array Model 实体的集合
```

list_all 方法是列表页面展现前，最常用的查询方式

list_all_array

```
Model::list_all_array(mixed $condition, array $pam)
查询并返回数组结果集
如果$condition 为数组则根据数组条件返回符合结果的列表
如果$condition 是字串则必须是 where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语句
如果不传入条件则根据 where 方法 的预设参数查询,如果 where 未被调用过则列出所有

Parameters:
mixed $condition 条件字符串或条件数组
array $pam 参数数组

Returns:
  array 字符下标的数组集合
```

map_array

```
Model::map_array(string $field1, string $field2, array $res_arr)
```

```
2.  将数据表的的两个字段的对应数据转换为键值数组形式
3.
4.  Parameters:
5.  string  $field1  key
6.  string  $field2  value
7.  array  $res_arr  默认预置数组
8.
9.  Returns:
10.    array  键值数组
```

map_array_kmap

```
1.  Model::map_array_kmap(string  $field1,  array  $farray)
2.  将数据表的的一个字段的值和多个字段的键值对对应的数据转换为键值数组-Map 的形式
3.
4.  Parameters:
5.
6.  string  $field1  key
7.
8.  array  $farray  要填充到 Map 的 Array(二级键值)
9.  Returns:
10.    array  一键多值数组
```

max

```
1.  Model::max(string  $field,  miexd  $condition,  miexd  $pam)
2.  查询并返回某字段的最大值
3.  如果$condition 为数组则根据数组条件返回符合结果的列表
4.  如果$condition 是字串则必须是  where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语
    句
5.  如果不传入条件则根据 where 方法  的预设参数查询,如果 where 未被调用过则列出所有
6.
7.  Parameters:
8.  string  $field  要查询  的字段
9.  miexd  $condition  条件字符串或条件数组
10.  miexd  $pam  参数数组
11.
12.  Returns:
13.    integer  最大值
```

min

```
1.  Model::min(string  $field,  miexd  $condition,  miexd  $pam)
2.  查询并返回某字段的最小值
3.  如果$condition 为数组则根据数组条件返回符合结果的列表
```

```
4.  如果$condition 是字串则必须是 where 语句之后的字串，亦可通过?和$pam 数组组合成 SQL 语句
5.  如果不传入条件则根据 where 方法 的预设参数查询,如果 where 未被调用过则列出所有
6.
7.  Parameters:
8.  string $field 要查询 的字段
9.  miexd $condition 条件字符串或条件数组
10. miexd $pam 参数数组
11.
12. Returns:
13.    integer 最小值
```

order

```
1.  Model::order(string $order)
2.  设置查询排序
3.
4.  Parameters:
5.
6.  string $order 排序
7.  Returns:
8.     Model 模型本身
```

remove

```
1.  Model::remove()
2.  删除本条信息
3.
4.  Returns:
5.     mixed 删除成功返回 1 失败返回 null
```

save

```
1.  Model::save()
2.  保存或更新此条信息
3.
4.  Returns:
5.     mixed
6.  验证失败返回 false
7.  存储失败返回 null
8.  存储成功返回本身
```

sum

```
1.  Model::sum(string $field)
```

```
2.  计算某一字段的和
3.
4.  Parameters:
5.  string  $field  参数数组
6.
7.  Returns:
8.    integer  计数
```

type

```
1.  Model::type(string  $field)
2.  获得某一字段的数据类型
3.
4.  Parameters:
5.  string  $field  字段名称
6.
7.  Returns:
8.    string  字段描述
```

update

```
1.  Model::update(array  $condition,  array  $data)
2.  批量更新信息如果不传入数据$data 且存在 id 则$condition 相当于$data 并依据 ID 进行$conditi
    on 数据更新
3.  如果不传入数据$data 且不存在 id 自动将这个 Model 的除 id 之外的其他字段属性作为更新数据
4.
5.  Parameters:
6.  array  $condition  条件数组
7.  array  $data  更新的数据数组
8.
9.  Returns:
10.   boolean  是否更新成功
```

where

```
1.  Model::where(mixed  $condition,  array  $pam)
2.  传入要查询的条件
3.
4.  Parameters:
5.  mixed  $condition  条件字符串或条件数组
6.  array  $pam  参数值的数组
7.
8.  Returns:
9.    Model  模型本身
```

自定义模型

YY 框架的模型既不是传统的基于数据库的 POJO 类也不是旨在操纵数据库的 Active Record 类。 YY 框架的模型是基于两种的结合，这种结合最大限度的发挥了 Model 层的自由度

我们来看一个例子：

```
class User extends Model {
    //此处定义虚拟字段
    //public $virtual_field = null;

    /**
    * 数据入库之前的合法性验证
    */
    public function validate(){
        //验证示例
        //$this->val_eamil('email');
        //或：$this->val_eamil('email','邮箱地址不正确');
        $this->val_eamil('email','电子邮箱格式不正确！');
        $this->val_min_length('un', 3,'帐号长度不能小于 3 个字符！');
        $this->val_max_length('un', 50,'帐号长度不能大于于 50 个字符！');
        $this->val_max_length('pwd', 50,'密码长度不能大于于 50 个字符！');
        $this->val_min_length('pwd', 3,'密码长度不能小于 3 个字符！');
        $this->val_unique('un','该用户名已经被注册！');
        $this->val_url('net','网址信息不正确！');
        $this->val_tel('phone','电话号码格式不正确！');
    }

    /**
    * 根据数据库的数据进行虚拟字段的填充
    */
    public function fill_virtual_field(){
        //虚拟字段填充示例
        //$this->virtual_field = $this->id.'_'.$this->name;

    }

    /**
    * 根据模型中的虚拟字段回填数据库字段数据
    */
    public function fill_entity_field(){
        //回填示例
        //$names = explode('_', $this->virtual_field);
```

```
37.             //$this->name = $names[1];
38.
39.         }
40.
41.         /**
42.         * 读取 Model 类的访问地址,有些模型数据的访问地址不止一个,需自行扩展
43.         */
44.         public function path(){
45.             //组合示例
46.             //$path = '/'.$this->theme.'/'.date('Y-m-d', $this->posttime).'/'.$this->id.'.html';
47.             //return $path;
48.
49.         }
50.
51. }
```

这是一个典型的自定义模型的例子。

虚拟字段是指数据库中不存在的字段，但是为了方便的数据的显示和汇总人为在模型中增加了这些字段。例如表单页面上可能会有年、月、日三个字段内容，实际的数据库中只有一个日期字段，这个时候利用虚拟字段功能实现数据操作层面的透明处理。

必须重写的方法

validate

这个方法是模型类的验证信息注册，方法内部依次填写各个字段所满足的格式要求。 详细的验证规则和使用方式请参阅模型数据校验

fill_virtual_field

根据数据库的数据进行虚拟字段的填充，这个方法会在模型数据初始完成后由框架自动调用。 方法中的内容是要明确的告诉框架每个虚拟字段的具体生成方式。

fill_entity_field

根据模型中的虚拟字段回填数据库字段数据，这个方法是在模型的数据即将保存(更新)到数据库前由框架自动调用。 方法中的内容旨在告诉框架如何把虚拟字段的内容转化为集体的数据库字段。

注意：以上三个方法只有在继承数据库模型时才是必须重写的，继承简单模型时 validate 方法可以根据实际需要进行重写。

表单项

YY 框架的表单提交遵循简易原则，下面是一个简单的 form 表单示例：

```
<form action="new.html" method="post">{tk()}
<div class="form">
{$m->hidden('id')}
<div class="fields">
    <div class="field field-first">
        <div class="label">
            <label for="input-large">IP 地址:</label>
        </div>
        <div class="input" id="selarea">
        {$m->text('ip','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">面板用户名:</label>
        </div>
        <div class="input">
        {$m->text('mbyhm','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">面板密码:</label>
        </div>
        <div class="input">
        {$m->text('mbmm','class="medium"')}
        </div>
    </div>
        <div class="field">
        <div class="label">
            <label for="input-small">购买日期:</label>
        </div>
        <div class="input">
        {$m->date('gmrq','class="medium"')}
        </div>
    </div>
        <div class="field">
        <div class="label">
            <label for="input-small">用户到期日:</label>
```

```
        </div>
        <div class="input">
        {$m->date('yhdqr','class="medium"')}
        </div>
    </div>
        <div class="field">
        <div class="label">
            <label for="input-small">服务器到期日:</label>
        </div>
        <div class="input">
        {$m->date('fwqdqr','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">系统用户名:</label>
        </div>
        <div class="input">
        {$m->text('xtyhm','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">系统密码:</label>
        </div>
        <div class="input">
        {$m->text('xtmm','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">淘宝用户名:</label>
        </div>
        <div class="input">
        {$m->text('tbyhm','class="medium"')}
        </div>
    </div>
    <div class="field">
        <div class="label">
            <label for="input-small">QQ 号码:</label>
        </div>
        <div class="input">
        {$m->text('qq','class="medium"')}
        </div>
```

```
84.     </div>
85.     <div class="field">
86.         <div class="label">
87.             <label for="input-small">联系邮箱:</label>
88.         </div>
89.         <div class="input">
90.         {$m->text('lxyx','class="medium"')}
91.         </div>
92.     </div>
93.     <div class="field">
94.         <div class="label">
95.             <label for="input-small">服务器类型:</label>
96.         </div>
97.         <div class="input">
98.         {$m->text('fwqlx','class="medium"')}
99.         </div>
100.    </div>
101.    <div class="buttons">
102.        <button type="submit">数据提交</button>
103.    </div>
104.</div>
105.</div>
106.</form>
```

视图中的**$m** 即为控制器中定义的数据库模型。表单项的第二个参数是 html 元素的其他属性。可以是数组如：$m->text('fwqlx',array("class"=>"medium","length"=>"2"))也可以是字符串的形式如：$m->text('fwqlx','"class"="medium" "length"="2")

其中的 **tk()**方法是开启表单令牌的声明，声明之后 form 中表单项自动开启加密，这样做可以有效的放置恶意数据的提交。 与此同时针对表单的重复提交也会被屏蔽。

注意：表单中的 **tk()**方法必须紧跟 form 的起始标签之后，这样之后定义的表单项才会被加密。为了系统安全我们建议表单的提交都开启令牌支持。

表单项

表单项	对应 html	备注
all	<textarea></textarea>...	模型所有非空属性的表单集合(隐藏的)
text	____	文本输入

表单项	对应 html	备注
password		密码框
email	<input type="email"/>	邮件输入框(html5)
range	<input type="range"/>	程度选择框(html5)
hidden	<input type="hidden"/>	隐藏标签
textarea	<textarea></textarea>	文本框
checkbox	☐	多选按钮
select	<select>...</select>	下拉框
radio	☐	单选按钮
date	____	时间选择
datetime	____	时间日期选择
texteditor	<textarea></textarea>	富文本编辑器
color	____	颜色选择框
vercode	____	验证码输入框
upload	<input type="file"/>	文件上传

数据校验

YY 框架把一切数据的校验全部集成在模型之中，这样可以有效的做到逻辑规则和限定规则的分离。 数据校验的实现是通过继承模型类并实现 validate 方法实现的。

集成在模型中的数据校验有一个更加出色的功能，那就是前后台校验的统一。 一旦在 validate 中定义好字段的校验规则，前台页面的的 js 适配器会自动针对校验规则实现 js 数据验证方法。

注意：好的开发人员不会轻易相信来自前台的任何数据，所以即使有 JS 校验后台的数据校验也是必须的。

通用校验方法

val_chinese

```
1.  SampleModel::val_chinese(string $field, string $errmsg)
2.  字段是否为中文的验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_eamil

```
1.  SampleModel::val_eamil(string $field, string $errmsg)
2.  字段 Email 格式合法性验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_english

```
1.  SampleModel::val_english(string $field, string $errmsg)
2.  字段是否为英文和数字的验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_integer

```
1.  SampleModel::val_integer(string $field, string $errmsg)
2.  字段是否为整数的验证
```

```
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_max_length

```
1.  SampleModel::val_max_length(string $field, integer $max, string $errmsg)
2.  字段最大长度的验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  integer $max 允许的最大长度
7.  string $errmsg 错误信息 {0}替换为$max
```

val_min_length

```
1.  SampleModel::val_min_length(string $field, integer $min, string $errmsg)
2.  字段最小长度的验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  integer $min 允许的最小长度
7.  string $errmsg 错误信息{0}替换为$min
```

val_notnull

```
1.  SampleModel::val_notnull(string $field, string $errmsg)
2.  字段非空验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_number

```
1.  SampleModel::val_number(string $field, string $errmsg)
2.  字段是否为数字的验证
3.
4.  Parameters:
5.  string $field 字段名称
6.  string $errmsg 错误信息
```

val_tel

```
1.  SampleModel::val_tel(string $field, string $errmsg)
```

```
2.  字段国内电话号码格式合法性验证
3.
4.  Parameters:
5.  string  $field  字段名称
6.  string  $errmsg  错误信息
```

val_url

```
1.  SampleModel::val_url(string  $field,  string  $errmsg)
2.  字段 url 合法性验证
3.
4.  Parameters:
5.  string  $field  字段名称
6.  string  $errmsg  错误信息
```

val_reg

```
1.  SampleModel::val_reg(string  $field,  string  $reg,  string  $errmsg,  string  $regjs)
2.  字段的自定义正则表达式验证
3.
4.  Parameters:
5.  string  $field  字段名称
6.  string  $reg  正则表达式
7.  string  $errmsg  错误信息
8.  string  $regjs  字段的前台 js 信息的验证规则
```

val_unique

```
1.  Model::val_unique(string  $field,  string  $errmsg)
2.  字段的 唯一性验证
3.
4.  Parameters:
5.  string  $field  字段名称
6.  string  $errmsg  错误信息
```

注意&提醒：

1、**val_unique** 方法只能在自定义的**数据库模型**中使用，因为这个方法的验证是需要经过数据检索的，简单模型没有这个验证功能。

2、val_unique 验证设置之后无论模型的新增或更新都会触发这个验证，数据新增时会对全表数据进行唯一性验证，更新时会对除了当前数据之外的其他数据进行唯一性验证。

2、除了 **val_reg** 方法外，其他参数的错误信息除了直接传入外还可以在国际化配置文件里配置中文的配置文件名称为 **zh-cn_validate.php**。

zh-cn_validate.php 文件内容：

```php
<?php
return array(
    'email' => '邮箱格式不正确',
    'unique' => '信息重复',
    'url' => '错误的网络地址',
    'number' => '此项必须为数字',
    'integer' => '此项必须为整数',
    'chinese' => '此项必须为中文和数字',
    'english' => '此项必须为英文和数字',
    'short' => '长度不能少于{0}位',
    'long' => '长度不能大于{0}位',
);
```

自定义验证规则

开发者要实现复杂的数据校验可以通过自定义正则表达式(val_reg)实现，但是某些情况下，出于一些特殊的需求可能单纯的正则表达式无法完成数据的有效性验证。 这个时候我们可以在 validate 方法中通过自定义验证逻辑来实现复杂验证。

方法实例：

```php
/**
* 数据入库之前的合法性验证
*/
public function validate(){
    $this->val_eamil('email','电子邮箱格式不正确！');
    $this->val_min_length('un', 3,'帐号长度不能小于 3 个字符！');
    $this->val_max_length('un', 50,'帐号长度不能大于于 50 个字符！');
    $this->val_max_length('pwd', 50,'密码长度不能大于于 50 个字符！');
    $this->val_min_length('pwd', 3,'密码长度不能小于 3 个字符！');
    $this->val_unique('un','该用户名已经被注册！');
    $this->val_url('net','网址信息不正确！');
    $this->val_tel('phone','电话号码格式不正确！');

    //下面为自定义的逻辑验证

    if($this->nickname == 'yyuc' && $this->nickname == 'laravel'){
        $this->set_err_msg('nickname','请不要使用我们钟爱的昵称');
    }
}
```

注意：通过 **set_err_msg** 方法自定义的错误校验结果，只能在后台获取；无法同步到前台页面通过框架的 JS 适配器完成校验。所以前台的校验脚本需要开发者自己完成。

校验结果的获取

后台获取

在自定义的数据库模型中校验是在 save 方法中自动被触发的。如果信息校验失败，save 方法返回 **false** 退出，并不会把信息写入到数据库中。

在自定义的简单模型中，需要开发者调用 validate 方法完成信息字段的校验，当然在自定义的数据库模型中也是可以直接调用 validate 方法的。

无论是哪种方式调用 validate 方法，如果校验失败，错误信息都会被存储到模型中，我们可以通过以下几个方法获取。

errors

```
SampleModel::errors()
获得模型的验证错误信息数组
这是一个以字段名称为数组下标的二维数组没有则返回空数组

Returns:
    array 错误信息数组
```

field_error

```
SampleModel::field_error(string $field)
获得某一字段的验证错误信息数组
返回该字段的验证错误信息,没有则返回空数组

Parameters:
string $field 字段名称

Returns:
    array 错误信息数组
```

field_errors

```
SampleModel::field_errors(string $field)
获得某一字段的验证错误信息字串(','分隔)
返回该字段的验证错误信息,没有则返回空串

Parameters:
```

```
6.  string $field 字段名称
7.
8.  Returns:
9.    string 错误信息
```

前台获取

前台表单项的验证错误信息可以通过 2 种方法获取：

1、单项获取：

```
1.  //弹出表单项校验的所有错误信息
2.  $('#theid').tovalidate(function(errs){
3.      if(errs === false){
4.          alert('这个字段不准为空');
5.      }else if(errs !== true){
6.          for(var i=0; i<errs.length;i++){
7.              alert(errs[i]);
8.          }
9.      }
10. });
```

2、整个 Form 表单错误信息获取

```
1.  $('form').validate(function(m){
2.      if(m.length>0){
3.          for(var i=0;i<m.length;i++){
4.              //m[i].e 为验证错误的表单元素
5.              //m[i].m 该单元的错误信息（非空错误为 false，其他为信息数组）
6.
7.              $(m[i].e).after('<span class="error">'+(m[i].m===false?'该项内容不能为空':m
    [i].m.join(','))+'</span>');
8.          }
9.      }else{
10.         $('form').submit();
11.     }
12. });
```

提示：如果你读不懂上面两段错误信息获取的代码，请学习使用 Jquery。

视图

视图简介

默认情况下 YY 框架的视图文件是和控制器文件是一一对应的，YY 框架也有一套便捷的视图文件解释器，生产模式下视图文件被采取预编译的方式包含到项目中，并不像有些框架每次请求都调用编译函数进行编译。 所以在系统效率上 YY 框架的视图机制没有任何额外的开销。

YY 框架视图采用{和}符号作为区别 Html 代码和标签代码的分割，也就是说学习 YY 框架的视图非常的容易。

请不要觉得视图标签的学习是一件很烦人的事情，有的人甚至认为 php 就是标签语言，额外的视图标签是多此一举的。 其实不然，通过对 YY 框架的视图学习，你将会发现学会并使用视图标签将大大简化你在项目中的开发量。

视图文件夹的目录结构

打开视图模板文件夹你会发现这这样的目录结构：

```
#分页视图
view/default/@pagination
#存放前台访问的资源文件的文件夹
view/default/@style
#存放 css 样式文件的文件夹
view/default/@style/css
#存放 js 脚本文件的文件夹
view/default/@style/js
#存放多媒体文件的文件夹
view/default/@style/media
#存放动画的文件夹
view/default/@style/media/animations
#存放图片的文件夹
view/default/@style/media/images
#存放声音文件的文件夹
view/default/@style/media/sounds
#存放视频文件的文件夹
view/default/@style/media/videos
#存放外包裹文件的文件夹
view/default/@wrap
```

注意：开发模式下@style 文件夹下的文件会即时同步到网站根目录，也就是/pub 文件夹下。生产模式下@style 下的文本文件(js,css)会被压缩存放到 pub 文件夹下，最大限度的保证系统访问速度。

视图基本标签

标签常量

YY 框架的标签中有 8 个标签常量：

```
/**JS 文件调用路径*/
$JS = Page::asset('js/');
/**CSS 文件调用路径*/
$CSS = Page::asset('css/');
/**图片文件调用路径*/
$IMG = Page::asset('media/images/');
/**动画文件调用路径*/
$ANI = Page::asset('media/animations/');
/**视频文件调用路径*/
$VID = Page::asset('media/videos/');
/**声音文件调用路径*/
$SOU = Page::asset('media/sounds/');
//国际化页面文字
/**通用页面国际化信息*/
$COM = YYUC::i18n();
/**具体页面国际化*/
$TXT = YYUC::i18n_page_init(Page::$my_view);
```

资源定位

其中前 6 个是资源文件的引用，通常在视图文件中你可以这样写：

```
<script src="{$JS}bootstrap.min.js"></script>
<link rel="shortcut icon" type="image.png" href="{$IMG}favicon.png">
```

很显然这 6 个标签是对资源文件的绝对引用，有了他们你就没有必要为烦人的资源定位而苦恼了。

注意： css 文件中同样存在图片资源定位的问题，因为 css 文件不是视图文件无法实现预编译，所以 css 文件中我们采取一种特殊的图片定位符 **img@**来定位图片文件。

css 中图片的定位示例：

```
.well {
  min-height: 20px;
  padding: 19px;
```

```
    margin-bottom: 20px;
    background-color: #f5f5f5;
    background: #f5f5f5 url(img@bg-secondary.png) repeat-x top left;
    border: 1px solid #eee;
    border: 1px solid rgba(0, 0, 0, 0.05);
    -webkit-border-radius: 4px;
    -moz-border-radius: 4px;
    border-radius: 4px;
    -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
    -moz-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
    box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
}
```

国际化文本引入

$COM 和$TXT 是针对国际化网站的文本引入常量，两个常量中分别保存了全局的和针对当前页面的文本信息，他们通过如下的方式调用：

```
#此处{$TXT['un']}的值为"用户名"，而如果是英文浏览器访问的话这个值会变成"username"
<label for="username">{$TXT['un']}:</label>

#此处的$COM 和$TXT 实现的效果相同，只是$COM 在任何视图中所访问的值都是相同的，而$TXT 是针对每个视图单独定义的
<span>版权所有{$COM['copyright']}</span>
```

更过关于国际化的知识请参阅国际化一节

逻辑标签

输出

控制器中定义的变量视图中可以直接进行输出

变量输出：

```
{$var}
```

函数执行并输出：

```
{date()}
```

特殊字符转义输出：

```
#这是防止跨站脚本攻击的有效输出方式
```

```
2.  {h  $var}
```

转义后的特殊字符原格式输出

```
1.  #与{h  $var}的作用正好相反
2.  {c  $var}
```

回车转换为 html 换行并输出：

```
1.  {n  $var}
```

回车转换为 html 换行并进行 html 字符安全转义输出：

```
1.  {nh  $var}
```

逻辑

php 脚本执行：

```
1.  {_  $var  =1}
```

判断脚本：

```
1.  {if  $var==1}
2.  ...
3.  {elseif   $var==2}
4.  ...
5.  {else}
6.  ...
7.  {/if}
```

循环脚本：

```
1.  #普通循环
2.  {loop  $arr  as  $key=>$value}
3.  {/loop}
4.
5.  {for  $i=0;$i<10;$i++}
6.  {/for}
7.
8.  {while  $i<10}
9.  {/while}
10.
11. #只循环前 20 次：
12. {20  loop  $arr  as  $value}
13. {/loop}
```

```
14.
15. #设置循环计数器，其中$i 从 1 开始递增
16. {loop@i $arr as $value}
17. <span>{$i}</span>
18. {/loop}
```

提示：如果想在模板文件中直接输出{，请在{之后加上空格或者之前加上！形如：**{空格** 或者 **!{**。

视图引用

YY 框架的视图引入有两种方式，一种是引入后统一编译(T 标签)，另一种是先执行在直接引入结果(I 标签)。

T 标签

T 标签是模板包含标签，其参数路径规则和 Page::view 方法一致，T 标签会把引入的视图文件直接包含到当前视图文件中。

如在 view/default/notes/index.html 文件中

{T list} 所包含的视图文件是：view/default/notes/list.html

{T /list} 所包含的视图文件是：view/default/list.html

I 标签

I 标签是引用包含标签，其参数路径规则和 Redirect::to 方法一致，I 标签会把引入的 URL 地址包含的内容直接展现在所属文件中。

如在 view/default/notes/index.html 文件中

{I list.htm} 会动态的替换为：http://www.yyuc.net/notes/list.htm 的 HTML 代码内容

{I list} 会动态的替换为：http://www.yyuc.net/notes/list.html 的 HTML 代码内容

{T /list}会动态的替换为：http://www.yyuc.net/list.html 的 HTML 代码内容

{T http://www.baidu.com }会动态的替换为：http://www.baidu.com 的 HTML 代码内容

特别说明：

1、调用 I 标签的控制器和视图一般是不要开启缓存的，因为如果开启缓存后目标页面发生了变化，调用页面不会对应的变化。 2、一般不推荐类似{I http://www.baidu.com }，因为调用的是别人网站的内容，很大程度上读取速度决定了网页的展现速度，一旦目标主机 Down 掉，本身的页面展现也会收到影响。

视图包裹

借鉴了 Rails 的思想，视图文件的相互引用方式分为内部包含和外部包裹两种方式。内部包含标签是{T ...}或者{I ...}。 外部包裹的用法是在视图文件夹下的@wrap 文件夹下建立对应的目录和文件，同等目录的视图文件就会自动被对应规则包裹起来。

视图包裹实例

建立包裹文件，view/default/@wrap/user/login/wrap.html

内容为：

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>{$title}</title>
</head>
<body>
<ul>
{loop $list as li}
....
{/loop}
<li>
</ul>
<div>
@YYUC-WRAP
</div>
</body>
</html>
```

建立包裹文件，view/default/@wrap/user/wrap.html

内容为：

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>全局包裹</title>
</head>
<body>
@YYUC-WRAP
```

```
</body>
</html>
```

建立视图文件：view/default/user/login/login.html

内容为：

```
<lable>用户名:</lable>...
<lable>密码:</lable>...
```

建立视图文件：view/default/user/login/success.html

```
<h2>登录成功</h2>
```

框架在进行视图编译时会进行逐级的包裹匹配，如编译文件 **view/default/login/user/login.html** 或文件 **view/default/user/login/success.html** 时首先寻找包裹文件 **view/default/@wrap/user/login/wrap.html** 是否存在。 如果存在则直接进行编译，不存在继续查找上级包裹文件 **view/default/@wrap/user/wrap.html** 是否存在，如不存在则继续查找直到查找到包裹目录顶级为止。

如上述文件所示，编译文件 **view/default/login/user/login.html** 时，相当于直接编译如下内容:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>{$title}</title>
</head>
<body>
<ul>
{loop $list as li}
....
{/loop}
<li>
</ul>
<div>
<lable>用户名:</lable>...
<lable>密码:</lable>...
</div>
</body>
</html>
```

提示： 包裹文件中的**@YYUC-WRAP** 标签为统一包裹替换符号。

视图包裹通常应用在框架结构一致的目录中，这样开发人员只需要关注视图中变化的部分就好。

分页

YY 框架的分页原理借鉴了 Rails 框架的 kaminari 插件，即方便调用又不失个性化，可以通过调用 Pagination 类方便的实现分页。

构造方法

控制器中的分页初始化：

```
1.  $pagination = new Pagination(20,9,true);
```

分页类初始化时接受 3 个参数：

1、每页条数 $numperpage

2、每页展现的链接数目 $linknum

3、当只有一页是是否仍要显示分页链接(默认 true) $showifone

初始化分页类之后，再调用查询代理方法，分页类中有两个代理分页查询的方法：model_list 和 sql_list。

model_list 传入已经设置了相关条件的模型，查询完成后返回包含模型的数组。

sql_list 直接传入 sql 语句，这种方式适合多表关联查询的情况返回二维数组结果集。

分页样式(P 标签)

框架视图中通过 P 标签来调用分页模块，在需要展现分页的地方放置标签代码：{P}即可。

{P}是{P default}的简写形式。default 是默认分页样式文件夹的名称。在视图文件夹下的@pagination 目录下。

分页样式文件夹下有 7 个样式文件,每个文件代表分页组建的一个部分: 1、common.html 普通页码 html 代码 2、current.html 当前页面的 html 代码 3、first.html 第一页 html 代码 4、gap.html 省略符 html 代码 5、last.html 最后一页 html 代码 6、next.html 下一页 html 代码 7、prev.html 上一页 html 代码

通过修改这些面的代码就可以实现分页样式的修改，如果网站中有多种分页样式，可以将 default 文件夹复制出来修改，如文件夹名称改为 bluepage。 视图模版中调用分页的代码写为: {P bluepage}即可。

控制器

YYUC 控制器简介

控制器是应用程序的心脏，因为它们决定如何处理 HTTP 请求

YY 框架的控制器和其他所有框架的控制器均不相同，其他框架的一个控制器就是一个类文件，而歪歪框架的控制器是实际存在的一个 php 过程脚本。 之所以没有沿用其他框架采用类文件的形式，是因为通常的一个 URL 请求用到的只是控制器类的一个方法，而无端加载其他方法到系统内存中显然是要有额外开销的。 最重要的一点是，类中方法的变量要直接被视图模板调用是不可能的(通常情况是要先把变量注册到视图模版中)，单独的几次请求可能看不出什么，但是当成千上万个并发到来时，类中变量转移到试图变量的开销就不容忽视了。

假设这个 URL:

```
1.  example.com/blog/
```

在上面的 URL 中，YY 框架将尝试寻找并装载一个名为 controller/blog/index.php 的控制器。 当控制器存在时，它将被装载。更多请参考 URL 规则

提示： 控制器名称和实际的 URL 请求是大小写敏感的，实际的开发过程中要格外注意。

控制器说明:

控制器是整个系统的核心，文档中介绍的方法几乎都是在控制器中调用的。

页面控制类 Page

Page 类是控制器的页面操作辅助类，它负责页面的访问控制，视图控制，缓存配置和资源获取。

Page 类对外方法

Page::view

```
1.  Page::view(string $viewpam)
2.  设置对应的 view 视图视图的相对路径
3.  根目录定位请开头补“/”
4.
5.  Parameters:
```

```
6.  string  $viewpam  视图名称
```

Page::ignore_view

```
1.  Page::ignore_view()
2.  忽略此次请求的视图(控制器执行之后直接退出)
```

Page::access_control

```
1.  Page::access_control(string  $fun)
2.  页面访问控制函数
3.  必须在控制器首行加入
4.
5.  Parameters:
6.  string  $fun  验证函数名称  定义在  /fun/access_validations.php 下的 php 函数
```

提示：通常只有在页面缓存和权限校验同时需要的时候才在控制器首行调用 access_control 函数，实际的开发中此函数的调用情况很少。

Page::cache_normal

```
1.  Page::cache_normal()
2.  页面缓存方式设置为常规缓存
```

Page::cache_time

```
1.  Page::cache_time(integer  $time)
2.  页面缓存方式设置为时间缓存
3.
4.  Parameters:
5.  integer  $time  缓存保留时间(单位：小时)
```

Page::cache_dbs

```
1.  Page::cache_dbs(string  $dbs)
2.  页面缓存方式设置为库表检测
3.
4.  Parameters:
5.  string  $dbs  检测的数据表，请以逗号分开各个表名如：'users,events'
```

缓存

YYUC 缓存简介

YY 框架的页面缓存分为常规缓存、基于时间的缓存和基于库表变动的缓存三类。

常规缓存

这是一种基于 Web 服务器的缓存模式，缓存文件一旦生成再次访问此 URL 时 Web 服务器将不会再把请求分发到 PHP 解释器处理，而是直接将缓存文件输出。 这是一种极其高效的缓存模式，缺点是缓存一旦生成后需要另外有进程更新和维护缓存。这种缓存适用于文章管理系统或新闻博客系统等静态展示较多的系统

注意： 需要进行常规缓存的 URL 页面请求我们建议以**.htm** 作为请求后缀，而不是普通的**.html** 作为后缀结尾。

假定 url 请求为：http://www.yyuc.net/demo/hello.htm。

首先创建控制器 controller/demo/hello.php 和视图文件 view/default/demo/hello.html

控制器内容为：

```
1.  //设为常规缓存
2.  page::cache_normal();
3.  $time = date('Y-m-d H:i:s');
```

视图内容为：

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5.  <title>缓存测试</title>
6.  </head>
7.  <body>
8.  <h1>{$time}</h1>
9.  </body>
10. </html>
```

访问页面 http://www.yyuc.net/demo/hello.htm 你会看到缓存输出的结果。

特别提醒：

1、所有页面缓存只有在生产模式下才能开启，所以请将配置文件中的**$is_developing** 设置为 **false**。

2、页面缓存只缓存不带 **?** 号传参的缓存数据。如 **news-2-a_3.htm** 可以被缓存，而 **new.htm?a=2&p=3** 是不能被缓存的。

3、因为是网络服务器直取数据而不经过 php 解释器所以常规缓存是不支持国际化的，也就是说常规缓存一旦形成其页面语言不会改变。所以常规缓存只建议在单语言系统中使用。

基于时间的缓存

基于时间的缓存，顾名思义设置缓存的更新时间，一段时间之后缓存自动失效新的缓存将生成，这类缓存通常适用于门户网站的主页。

将上例中的控制器改为

```
1.  //基于时间的缓存
2.  page::cache_time(0.001);
3.  $time  =  date('Y-m-d  H:i:s');
```

访问页面 http://www.yyuc.net/demo/hello.html 你会看到缓存输出的结果。

提醒：cache_time 方法传入值单位是小时，所以此处传入 0.001 即为 3.6 秒。

基于库表变动的缓存

YY 框架内部有检测库表更新状态的回调函数，一旦数据库的某张表涉及到了 CUD(插入、更新、删除)操作，那么这张表所对应的某些页面缓存也会被更新。

将上例中的控制器改为

```
1.  //设为库表变动缓存
2.  page::cache_dbs('news','users');
3.  $time  =  date('Y-m-d  H:i:s');
```

访问页面 http://www.yyuc.net/demo/hello.html 你会看到缓存输出的结果。

一旦 news 或者是 users 中的某一张表发生变化缓存就将被更新。

页面缓存

YY 框架的页面缓存分为常规缓存、基于时间的缓存和基于库表变动的缓存三类。

常规缓存

这是一种基于 Web 服务器的缓存模式，缓存文件一旦生成再次访问此 URL 时 Web 服务器将不会再把请求分发到 PHP 解释器处理，而是直接将缓存文件输出。 这是一种极其高效的缓存模式，缺点是缓存一旦生成后需要另外有进程更新和维护缓存。这种缓存适用于文章管理系统或新闻博客系统等静态展示较多的系统

注意： 需要进行常规缓存的 URL 页面请求我们建议以**.htm** 作为请求后缀，而不是普通的**.html** 作为后缀结尾。

假定 url 请求为：http://www.yyuc.net/demo/hello.htm。

首先创建控制器 controller/demo/hello.php 和视图文件 view/default/demo/hello.html

控制器内容为：

```
1.  //设为常规缓存
2.  page::cache_normal();
3.  $time = date('Y-m-d H:i:s');
```

视图内容为：

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5.  <title>缓存测试</title>
6.  </head>
7.  <body>
8.  <h1>{$time}</h1>
9.  </body>
10. </html>
```

访问页面 http://www.yyuc.net/demo/hello.htm 你会看到缓存输出的结果。

特别提醒：

1、所有页面缓存只有在生产模式下才能开启，所以请将配置文件中的**$is_developing** 设置为 **false**。

2、页面缓存只缓存不带 **?** 号传参的缓存数据。如 **news-2-a_3.htm** 可以被缓存，而 **new.htm?a=2&p=3** 是不能被缓存的。

3、因为是网络服务器直取数据而不经过 php 解释器所以常规缓存是不支持国际化的，也就是说常规缓存一旦形成其页面语言不会改变。所以常规缓存只建议在单语言系统中使用。

基于时间的缓存

基于时间的缓存，顾名思义设置缓存的更新时间，一段时间之后缓存自动失效新的缓存将生成，这类缓存通常适用于门户网站的主页。

将上例中的控制器改为

```
1.  //基于时间的缓存
2.  page::cache_time(0.001);
3.  $time = date('Y-m-d H:i:s');
```

访问页面 http://www.yyuc.net/demo/hello.html 你会看到缓存输出的结果。

提醒：cache_time 方法传入值单位是小时，所以此处传入 0.001 即为 3.6 秒。

基于库表变动的缓存

YY 框架内部有检测库表更新状态的回调函数，一旦数据库的某张表涉及到了 CUD(插入、更新、删除)操作，那么这张表所对应的某些页面缓存也会被更新。

将上例中的控制器改为

```
1.  //设为库表变动缓存
2.  page::cache_dbs('news','users');
3.  $time = date('Y-m-d H:i:s');
```

访问页面 http://www.yyuc.net/demo/hello.html 你会看到缓存输出的结果。

一旦 news 或者是 users 中的某一张表发生变化缓存就将被更新。

程序缓存

程序缓存是开发着在实际的代码中调用的缓存技术。

有关缓存的配置文件

```
1.  /**全局变量适配器~(可选项:file,memcached)这是多用户共用缓存的好方式，大型系统不推荐 file 模式，默认:file*/
2.  public static $cache_adapter = 'file';
3.
4.  /**MemCached 连接信息~*/
5.  public static $memcached = array(
6.          'cache1'=>array('127.0.0.1','11211'),
```

```
            'cache2'=>array('127.0.0.1','11211')
);
```

YY 框架默认提供文件系统和 memcached 两种缓存方式，其中 memcached 方式需要额外的配置 memcached 的服务器信息以及开启 php 的 memcache 支持。

提示：其他的一些流行的缓存技术如 APC, mongodb, redis 等在框架插件都有相应的支持，开发中可以根据系统的实际情况选取。
默认情况下 YY 框架的缓存都是存储在文件系统中，这种方法虽然简便但是当达到 IO 瓶颈时系统响应就会大受影响(并发少于 1000 的话文件缓存还是可取的)。

缓存的常用方法

Cache::set

```
Cache::set(string $k, mixed $v, string $time)
保存缓存数据

Parameters:
string $k 键
mixed $v 值
string $time 保存持续时间(单位:秒,0 为永远)
```

Cache::get

```
Cache::get(string $k)
获得缓存的值

Parameters:

string $k 键
Returns:
mixed 值
```

Cache::remove

```
Cache::remove(string $k)
删除某个缓存值

Parameters:
string $k 键
```

Cache::has

```
Cache::has(string $k)
```

```
2.  判断是否存在某个缓存值
3.
4.  Parameters:
5.
6.  string  $k 键
7.  Returns:
8.  boolean
```

Cache::forever

```
1.  Cache::forever(string  $k,  mixed  $v)
2.  永久要保存的缓存数据
3.  这个方法通过文件形式存储缓存
4.
5.  Parameters:
6.  string  $k 键
7.  mixed  $v 值
```

$v 为 null 时则为读取永久缓存

Cache::forget

```
1.  Cache::forget(string  $k)
2.  删除某个永久缓存
3.
4.  Parameters:
5.  string  $k 键
```

Session 和 Cookie

Session

框架中对 session 的操作封装在 Session 类中，我们可以通过设置 **conf.php** 中的配置项设置 session 的适配器。

```
1.  /**会话适配器~(可选项:session,cache)session:系统默认的 Session 机制，cache:运用系统缓存保存 session*/
2.  public  static  $session_adapter  =  "session";
```

关于缓存请参阅 YYUC 缓存

同样我们可以在配置文件中定义 session 的过期时间：

```
/**Session 过期时间~设置访问客户端的 Session 过期时间(注意：这只是 Session 的标记删除时间，并不保证这段时间后一定会被回收，单位：分钟)，默认:60*/
public static $session_time = 60;
```

操作方法：

set

```
Session::set(string $k, string $v)
Session 设置

Parameters:
string $k Session 键
string $v Session 值 传入 null 则为清空 Session
```

get

```
Session::get(string $k)
安全取得 Session 内容

Parameters:
string $k Session 键

Returns:
  string Session 内容
```

has

```
Session::has(string $k)
判断 Session 是否含有指定内容

Parameters:
string $k Session 参数

Returns:
  boolean
```

remove

```
Session::remove(string $k)
删除指定 Session 内容

Parameters:
string $k Session 键
```

once

```
Session::once(string $k, string $v)
一次性 Session 显示信息存入

Parameters:
string $k Session 键
string $v Session 值 传入 null 则为清空 Session
```

flush

```
Session::flush(string $k)
取得 Session 一次性显示内容

Parameters:
string $k Session 参数

Returns:
  string Session 内容
```

hold

```
Session::hold(string $k)
判断 Session 是否含有一次性显示内容

Parameters:
string $k Session 参数

Returns:
  boolean
```

clear

```
Session::clear()
清空 Session
```

Cookie

框架中对 cookie 的操作封装在 Cookie 类中。

操作方法：

set

```
Cookie::set(string $k, string $v, boolean $httponly, integer $time, string $domain, strin
g $path, boolean $secure)
```

```
2.  Cookie 设置 默认过期时间是一个月,范围是整个 domain
3.
4.  Parameters:
5.  string $k Cookie 键
6.  string $v Cookie 值 传入 null 则为清空 Cookie
7.  boolean $httponly 是否是禁止客户端操作的
8.  integer $time 过期时间
9.  string $domain 适用范围
10. string $path 使用路径
11. boolean $secure 是否是安全(https)的
```

get

```
1.  Cookie::get(string $k)
2.  安全取得 Cookie 内容
3.
4.  Parameters:
5.  string $k cookie 参数
6.
7.  Returns:
8.    string cookie 内容
```

has

```
1.  Cookie::has(string $k)
2.  判断是否含有指定的 Cookie 内容
3.
4.  Parameters:
5.  string $k Cookie 键
6.
7.  Returns:
8.    boolean
```

remove

```
1.  Cookie::remove(string $k)
2.  删除指定的 Cookie 内容
3.
4.  Parameters:
5.  string $k Cookie 键
```

clear

```
1.  Session::clear()
2.  清空 Cookie
```

请求和响应(Request&Response)

Request

YY 框架中对请求的操作封装在 Request 类中。

操作方法：

get

```
Request::get(string $pam, mixed $default)
获取 get 方式提交的参数
参数为 null 则为判断是否有 get 提交

Parameters:
string $pam 参数名称
mixed $default 如果不存在请求内容则为传入的默认值，此值可以为闭包函数

Returns:
  string 参数值
```

post

```
Request::post(string $pam, mixed $default)
获取 post 方式提交的参数
参数为 null 则为判断是否有 post 提交

Parameters:
string $pam 参数名称
mixed $default 如果不存在请求内容则为传入的默认值，此值可以为闭包函数

Returns:
  mixed 参数值 字串或数组
```

obtain

```
Request::obtain(string $pam, mixed $default)
获取提交的参数(包含 get,post 等方式)
参数为 null 则为判断是否有请求

Parameters:
string $pam 参数名称
mixed $default 如果不存在请求内容则为传入的默认值，此值可以为闭包函数
```

```
8.
9.  Returns:
10.    mixed  参数值  字串或数组
```

ip

```
1.  Request::ip()
2.  获得客户端 IP(真实的 IP 地址)
3.
4.  Returns:
5.    string  客户端 IP 地址
```

is_proxy

```
1.  Request::is_proxy()
2.  判断用户是否通过代理访问
3.  对于超匿名代理无法判断出来
4.  如果网站本身采用了 cdn 加速等功能的话正常用户会被误判断成代理访问的。
5.
6.  Returns:
7.    string  客户端 IP 地址
```

page

```
1.  Request::page()
2.  URL 请求中含有分页参数表示的页数
3.  没有分页参数则返回 1
4.
5.  Returns:
6.    integer  页数
```

例如：请求为 http://www.yyuc.net/news/2012/index_3html,则：Request::page() 的值为 **3**。

part

```
1.  Request::part(integer  $index)
2.  层级式 URL 解析时的各级名称
3.  级别索引 从 0 开始
4.
5.  Parameters:
6.
7.  integer  $index  索引
8.  Returns:
9.    string  URL 中的路径名称
```

例如：请求为 http://www.yyuc.net/linux/install/sendmail.html,则：

$num = Request::part(); $num 的值为 3。

$link0 = Request::part(0); $link0 的值为 linux。

$link1 = Request::part(1); $link1 的值为 install。

$link2 = Request::part(2); $link2 的值为 sendmail。

注意：Request 的操作是针对用户请求的，即使你做了自定义路由，part 方法也不会受影响。如把 /linux/install/sendmail.html 映射到/linux/sendmail.php，Request::part(1)仍然为 **install**。

part 方法是自定义路由中控制器常常调用的方法，通常情况下自定义路由中需要把用户的真实请求的 URL 段作为判断和查询的依据。详细请参阅自定路由

is_normal_cache

```
Request::is_normal_cache()
判断当前请求是不是常规缓存的请求

Returns:
    string  URL
```

有关缓存的详细介绍请参阅页面缓存

url

```
Request::url()
获得用户请求的真实 URL
不带请求后缀

Returns:
    string  URL
```

和 part 方法一样，这个方法也是用户请求的真实体现。

part_control

```
Request::part_control(integer  $index)
控制器层级式 URL 解析时的各级名称
级别索引 从 0 开始

Parameters:
integer  $index  索引
Returns:
    string  URL 中的路径名称
```

part_control 方法和 **part** 方法不同，part_control 是针对当前实际执行的控制器而言的。

json

```
Request::json()
获得 ajax 请求的 JSON 数据 并转换为 PHP 数组
客户端请求的参数名称必须是:'data'

```

```
5.  Returns:
6.      array 请求数组
```

Response

框架中对响应的操作封装在 Response 类中。

操作方法：

write

```
1.  Response::write(string $str, Mime $mime)
2.  文本输出 ,输出后退出此次请求
3.
4.  Parameters:
5.  string $str 要输出的字串
6.  Mime $mime 要输出 mime 类型默认为：html
```

Mime 类中定义了常用的 Mime 类型，他们是：

```
1.  public static $hqx = 'application/mac-binhex40';
2.  public static $cpt = 'application/mac-compactpro';
3.  public static $csv = array('text/x-comma-separated-values', 'text/comma-separated-valu
    es', 'application/octet-stream');
4.  public static $bin = 'application/macbinary';
5.  public static $dms = 'application/octet-stream';
6.  public static $lha = 'application/octet-stream';
7.  public static $lzh = 'application/octet-stream';
8.  public static $exe = array('application/octet-stream', 'application/x-msdownload');
9.  public static $class = 'application/octet-stream';
10. public static $psd = 'application/x-photoshop';
11. public static $so = 'application/octet-stream';
12. public static $sea = 'application/octet-stream';
13. public static $dll = 'application/octet-stream';
14. public static $oda = 'application/oda';
15. public static $pdf = array('application/pdf', 'application/x-download');
16. public static $ai = 'application/postscript';
17. public static $eps = 'application/postscript';
18. public static $ps = 'application/postscript';
19. public static $smi = 'application/smil';
20. public static $smil = 'application/smil';
21. public static $mif = 'application/vnd.mif';
```

```
22. public static $xls = array('application/excel', 'application/vnd.ms-excel', 'application/msexcel');
23. public static $ppt = array('application/powerpoint', 'application/vnd.ms-powerpoint');
24. public static $wbxml = 'application/wbxml';
25. public static $wmlc = 'application/wmlc';
26. public static $dcr = 'application/x-director';
27. public static $dir = 'application/x-director';
28. public static $dxr = 'application/x-director';
29. public static $dvi = 'application/x-dvi';
30. public static $gtar = 'application/x-gtar';
31. public static $gz = 'application/x-gzip';
32. public static $php = array('application/x-httpd-php', 'text/x-php');
33. public static $php4 = 'application/x-httpd-php';
34. public static $php3 = 'application/x-httpd-php';
35. public static $phtml = 'application/x-httpd-php';
36. public static $phps = 'application/x-httpd-php-source';
37. public static $js = 'application/x-javascript';
38. public static $swf = 'application/x-shockwave-flash';
39. public static $sit = 'application/x-stuffit';
40. public static $tar = 'application/x-tar';
41. public static $tgz = array('application/x-tar', 'application/x-gzip-compressed');
42. public static $xhtml = 'application/xhtml+xml';
43. public static $xht = 'application/xhtml+xml';
44. public static $zip = array('application/x-zip', 'application/zip', 'application/x-zip-compressed');
45. public static $mid = 'audio/midi';
46. public static $midi = 'audio/midi';
47. public static $mpga = 'audio/mpeg';
48. public static $mp2 = 'audio/mpeg';
49. public static $mp3 = array('audio/mpeg', 'audio/mpg', 'audio/mpeg3', 'audio/mp3');
50. public static $aif = 'audio/x-aiff';
51. public static $aiff = 'audio/x-aiff';
52. public static $aifc = 'audio/x-aiff';
53. public static $ram = 'audio/x-pn-realaudio';
54. public static $rm = 'audio/x-pn-realaudio';
55. public static $rpm = 'audio/x-pn-realaudio-plugin';
56. public static $ra = 'audio/x-realaudio';
57. public static $rv = 'video/vnd.rn-realvideo';
58. public static $wav = 'audio/x-wav';
59. public static $bmp = 'image/bmp';
60. public static $gif = 'image/gif';
61. public static $jpeg = array('image/jpeg', 'image/pjpeg');
62. public static $jpg = array('image/jpeg', 'image/pjpeg');
63. public static $jpe = array('image/jpeg', 'image/pjpeg');
```

```
64. public static $png = 'image/png';
65. public static $tiff = 'image/tiff';
66. public static $tif = 'image/tiff';
67. public static $css = 'text/css';
68. public static $html = 'text/html';
69. public static $htm = 'text/html';
70. public static $shtml = 'text/html';
71. public static $txt = 'text/plain';
72. public static $text = 'text/plain';
73. public static $log = array('text/plain', 'text/x-log');
74. public static $rtx = 'text/richtext';
75. public static $rtf = 'text/rtf';
76. public static $xml = 'text/xml';
77. public static $xsl = 'text/xml';
78. public static $mpeg = 'video/mpeg';
79. public static $mpg = 'video/mpeg';
80. public static $mpe = 'video/mpeg';
81. public static $qt = 'video/quicktime';
82. public static $mov = 'video/quicktime';
83. public static $avi = 'video/x-msvideo';
84. public static $movie = 'video/x-sgi-movie';
85. public static $doc = 'application/msword';
86. public static $docx = 'application/vnd.openxmlformats-officedocument.wordprocessing
    ml.document';
87. public static $xlsx = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sh
    eet';
88. public static $word = array('application/msword', 'application/octet-stream');
89. public static $xl = 'application/excel';
90. public static $eml = 'message/rfc822';
91. public static $json = array('application/json', 'text/json');
```

json

```
1.  Response::json($arr)
2.  json 输出 ,输出后退出此次请求
3.
4.  Parameters:
5.  array  $arr  要输出的数组
```

exejs

```
1.  Response::exejs(string  $jscode)
2.  客户端直接执行 JS 语句!!!此方法只在加载了视图文件的情况下可用
3.  所添加的 JS 代码会在页面前期初始化工作完成之后执行
4.
```

```
5.  Parameters:
6.  string  $jscode  要执行的 JS 脚本
```

download

```
1.  Response::download(string  $filename,  mixed  $content,  Mime  $mime)
2.  准备为客户端进行文件下载
3.
4.  Parameters:
5.  string  $filename  客户端下载的文件名
6.  mixed  $content  客户端下载的文件内容，如不传入则把这次请求的实际相应内容作为文件内容
7.  Mime  $mime  要输出 mime 类型默认为：APPLICATION
```

权限和验证

权限授予与认证

YY 框架中对认证和权限的操作封装在 Auth 类中。我们引入用户和组的概念来管理系统用户。

Auth 类的操作方法：

权限声明：

im_admin

```
1.  Auth::im_admin(mixed  $adminlevel)
2.  管理员声明
3.
4.  Parameters:
5.  mixed  $adminlevel  可以不填，管理员级别(string|array)
```

声明当前登录用户是管理员，可以调用 Auth::im_admin()。
如果管理员分为多种类型可以调用 Auth::im_admin('home')，Auth::im_admin('office')。由传入的**组**即可区分声明是家庭管理员还是办公管理员。
同一个登录用户可以兼具不同组的管理权限，所以如果同时调用 Auth::im_admin('home')和 Auth::im_admin('office')，则当前用户既是家庭管理员又是办公管理员。
你也可以采取传入数组的形式，这样更加简单：Auth::im_admin(array('home','office'))。

im_user

```
1.  Auth::im_user($userlevel,  string  $userkey)
```

```
用户声明

Parameters:
mixed  $userlevel 用户级别(string|array)
string  $userkey  群组标识
```

和 **im_admin** 用法相同，此方法用来声明当前用户的权限级别。

其中$userkey 是群组标识，用来传入用户所属的群组，$userlevel 用来标识用户所在组内的级别。

注意： im_admin 只是 im_user 的一种特殊形式，即所属组为**管理员组**。**im_user** 也可以不传任何参数直接调用，用来声明当前用户为普通用户：Auth::im_user()。

权限判断：

is_admin

```
Auth::is_admin(mixed  $adminlevel)
判断用户是否是管理员

Parameters:

mixed  $adminlevel  管理员级别(string|array)
Returns:
    boolean
```

如：该用户曾经声明 Auth::im_admin('home')，则 Auth::is_admin('home')返回 **true**,这里要注意的是 Auth::is_admin()同样返回 **true**。

is_user

```
Auth::is_user(mixed  $userlevel,  string  $userkey)
判断用户是否登录

Parameters:
mixed  $userlevel  用户级别(string|array)
string  $userkey  用户群组
Returns:
    boolean
```

如：该用户曾经声明 Auth::im_user('1')，则 Auth::is_user('1')返回 **true**,这里要注意的是 Auth::is_user()同样返回 **true**。

权限注销：

权限的注销和权限声明在调用方法以及传入参数上一致的。

im_notadmin

```
Auth::im_notadmin(mixed $adminlevel)
管理员注销声明

Parameters:
mixed $adminlevel 可以不填，管理员级别组(string|array)
```

im_notuser

```
Auth::im_notuser($userlevel, string $userkey)
用户注销声明

Parameters:
mixed $userlevel 用户级别(string|array)
string $userkey 用户群组
```

访问控制

很显然，对于需要用户身份验证的系统，每个页面都加入验证脚本是十分愚蠢的。同样的即使每个页面都用类似 **include** 的方式引入验证规则也是十分麻烦的事情。

因此我们提出访问控制的概念，让开发者根据用户请求的 URL 信息验证当前用户是否有权限访问。验证的方法名为：**access_validations**，通常情况下我们把它定义在 fun/access_validations.php 文件中。

验证示例：

```
/**
 * 此方法是框架访问权限验证的钩子方法<br/>
 * $uri 为请求的 URL 的相对路径<br/>
 * 如:http://www.yyuc.com/admin/set/index.html 则$uri 为:admin/set/index<br/>
 * $uri 为实际的控制器路径，而并非用户真实的请求路径(开启自定义路由的情况下两者并不相同)
 * @param $uri
 */
function access_validations($uri){
    //网站管理员验证
    if(String::start_with($uri, 'admin/') && $uri != 'admin/login' && !Auth::is_user()){
        Redirect::to('/admin/login');
    }

    if(String::start_with($uri, 'superadmin/')){
            Redirect::to_404();
    }
}
```

这里要特别注意的是参数$uri 是**控制器路径**而不是用户实际请求的路径。

数据监控

出于对数据安全性的考虑，YY 框架提供了一种对数据操作的监控机制。同样也是运用了类似钩子的回调机制完成的。

这个监控方法的名称为 **db_validations**，通常情况下我们也把也它定义在 fun/access_validations.php 文件中。

函数示例：

```
/**
 * 数据执行校验
 *
 * @param DBDes $dbdes
 */
function db_validations($dbdes){
    //操作表为 user、操作为非查询操作，操作用户并不是管理员
    if($dbdes->table == 'user' &&$dbdes->operate !='R' && !Auth::is_admin() ){
        log::warn('非法修改 user 表，视图的修改方式是：'.$dbdes->operate.'，执行的 SQL 是：'.$dbdes->sql.'入侵 IP 为:'.Request::ip());
        die('非法操作');
    }
}
```

其中 DBDes 为数据库操作描述类。

DBDes 属性介绍：

```
$real_table
真实表名(包含表前缀)
Type
  string

$table
表名(不含表前缀)
Type
  string

$model
执行当前操作的数据模型，如果不是有模型操作而是直接的原生 SQL 此项为 null
Type
  Model

```

```
$sql
所执行的 SQL
Type
    string


$operate
所执行的操作 'C' 'R' 'U' 'D'
Type
    string
```

国际化

YYUC 国际化功能介绍

YY 框架的国际化功能和其他框架一样，都是保存在配置文件之中的，首先要在配置文件 **conf.php** 中定义默认的语言。

默认语言为中文简体:

```
/**国际化~默认的网站国际化支持如 zh-cn:中文简体,zh-tw:中文繁体,en:英语...*/
public static $default_i18n = "zh-cn";
```

假设我们的系统需要支持中文简体和英文两种语言，后面的内容为具体的配置步骤。

全局文本配置

在 **i18n** 文件夹下建立 zh-cn.php 和 en.php 文件

zh-cn.php：

```
return array(
    'blue' => '蓝色',
    'green' => '绿色',
    'brown' => '棕色',
    'purple' => '姿色',
    'red' => '红色',
    'greyblue' => '灰蓝色',
    'news' => array(
            'book'=> '书',
            'paper'=> '纸',
```

```
        )
);
```

en.php：

```
return array(
    'blue' => 'blue',
    'green' => 'green',
    'brown' => 'brown',
    'purple' => 'purple',
    'red' => 'red',
    'greyblue' => 'greyblue',
    'news' => array(
            'book'=> 'book',
            'paper'=> 'paper',
        )
);
```

控制器中的文本读取：

```
$blue = i18n('.blue');
$book = i18n('.news.book');
```

视图文件中显示:

```
<span>{$COM['blue']}</span>
<span>{$COM['news']['book']}</span>

<!-- 也可以用控制器中的取得方式 -->
<span>{i18n('.blue')}</span>
<span>{i18n('.news.book')}</span>
```

全局文本配置信息过多时我们可以分文件存储：

```
在**i18n**文件夹下建立 zh-cn_color.php 和 en_color.php 文件
```

zh-cn_color.php：

```
return array(
    'blue' => '蓝色',
    'green' => '绿色',
    'brown' => '棕色',
    'purple' => '姿色',
    'red' => '红色',
    'greyblue' => '灰蓝色',
    'news' => array(
```

```
                'book'=> '书',
                'paper'=> '纸',
            )
);
```

en_color.php：

```
return array(
    'blue' => 'blue',
    'green' => 'green',
    'brown' => 'brown',
    'purple' => 'purple',
    'red' => 'red',
    'greyblue' => 'greyblue',
    'news' => array(
                'book'=> 'book',
                'paper'=> 'paper',
            )
);
```

控制器中的文本读取：

```
$blue = i18n('color.blue');
$book = i18n('color.news.book');
```

视图文件中显示:

```
<!-- 在视图文件中只有通过和控制器相同的方式获取 -->
<span>{i18n('color.blue')}</span>
<span>{i18n('color.news.book')}</span>
```

说明：$COM 方式是相对$TXT 方式而言的，$COM 是全局国际化定义文本，任何视图文件都可以直接调用，**$TXT** 是针对多级视图国际化文本的分文件存储方式。

国际化分文件存储

对于全站国际化的系统，如果把视图中所出现的文字全部放在一个国际化文件中，如放到 zh-cn.php 中，这个文件将会特别大，每次请求都加载这个国际化文件的话系统执行效率将会大大降低。

因此 YY 框架允许国际化文件的分文件存储。

示例

建立国际化文件：i18n/zh-cn/login.php 内容为：

```
return array(
    title => '全站登录',
);
```

建立国际化文件：i18n/en/login.php 内容为：

```
return array(
    title => 'all-login',
);
```

建立国际化文件：i18n/zh-cn/user/login.php 内容为：

```
return array(
    title => '用户登录',
);
```

建立国际化文件：i18n/en/user/login.php 内容为：

```
return array(
    title => 'user-login',
);
```

以上国际化文件是针对特定视图的，只有在特定视图中才能调用。

建立视图文件 view/default/login.html:

```
<h1>{$TXT['title']}</h1>
```

英文地域的浏览器请求结果为：**all-login**。中文地域的浏览器请求结果为：**全站登录**。

建立视图文件 view/default/user/login.html:

```
<h1>{$TXT['title']}</h1>
```

英文地域的浏览器请求结果为：**user-login**。中文地域的浏览器请求结果为：**用户登录**。

数据库操作

数据库功能介绍

目前 YY 框架的关系型数据库只支持 mysql 一种，近期我们不会推出基于其他类型数据库的适配器。

数据库配置:

数据的各项配置在配置文件 **conf.php** 中定义：

```
1.  ////////////////////数据库配置 编码数据库编码为 utf-8
2.  /**数据库地址~*/
3.  public static $db_host = "localhost";
4.  /**数据库端口~*/
5.  public static $db_port = "3306";
6.  /**数据库名~*/
7.  public static $db_dbname = "yyuc";
8.  /**数据库用户名~*/
9.  public static $db_username = "root";
10. /**数据库密码~*/
11. public static $db_password = "root";
12. /**数据库表前缀~为了区分同一个数据库中不同框架的数据表,虚拟主机环境下我们建议设置此项。
    */
13. public static $db_tablePrefix = "";
```

以上为基本的数据库配置信息，有的时候出于性能考虑你可能会用到主从数据库或多库数据库，那么我们还需要下面的配置信息：

```
1.  /**DB 连接的扩展信息~一般用于大数据量时的 主从数据库和多库数据库*/
2.  public static $db_boys = array(
3.      'db1'=>array('host'=>'localhost1','port'=>'3306','dbname'=>'sino','username'=>'root
    ','password'=>'root'),
4.      'db2'=>array('host'=>'localhost1','port'=>'3307','dbname'=>'sino','username'=>'root
    ','password'=>'root'),
5.      'db3'=>array('host'=>'localhost2','port'=>'3306','dbname'=>'sino','username'=>'root
    ','password'=>'root'),
6.      'db4'=>array('host'=>'localhost3','port'=>'3306','dbname'=>'sino','username'=>'root
    ','password'=>'root')
7.  );
```

到这里你可能还是弄不明白如何使用这些数据库连接信息，不急，接着往下看。

数据库连接的获取

get_db

```
1.  DB::get_db(string $conn_name)
2.  单列模式获得数据库连接类
3.  如果人为指定连接名称则 DB 类会按照多库规则处理,不会自动切换数据库连接
4.
```

```
5.
6.  Parameters:
7.  string  $conn_name  多库并存的情况下指明要获得连接的名称  默认为 true
8.  传入 false 则锁定到主数据库
9.  传入 null 切换主数据库恢复到主从模式
10. 传入 true 时随机切换到从属数据库
11.
12. Returns:
13.    DB
```

通常情况下不建议直接调用 get_db 获得数据连接进行 CUD(非查询)操作，而是通过数据库模型类进行操纵。通过数据库模型类操作的 CUD 信息都会通过默认连接(非 db_boys 连接)保存到数据库中。

具体指定使用某一个连接：

```
1.  $db = DB::get_db('db1');
```

这样无论进行何种操作(包括 CUD 操作)都会使用 db_boys 中指定的 db1 信息进行连接，直至再次调用 **get_db** 方法或**请求退出。**

指定使用主数据库连接：

```
1.  $db = DB::get_db('false');
```

这样无论进行何种操作(包括 R 操作)都会使用主数据库信息进行连接，直至再次调用 **get_db** 方法或**请求退出。**

如果主数据库和 db_boys 均配置了连接信息时，系统默认使用主从数据库模式，通过数据库模型调用的有关数据操作规则如下：
CUD(非查询)操作使用主数据库连接，**R**(查询)操作使用 db_boys 的一条随机信息连接。
注意： 主从数据下务必做好各数据库细信息的同步。

原生操作

数据库的原生 SQL 操作最常用的有两个方法：**query** 和 **execute**。

query

```
1.  DB::query(string  $sql,  array  $pam)
2.  普通 SQL 查询
3.
4.  Parameters:
5.  string  $sql  要执行查询的 SQL 语句参数请用"?"代替
6.  array  $pam  数字下标的数组，数组项依次替换 SQL 中的"?"参数
```

```
7.
8.  Returns:
9.      array 字符下标的数组集合
```

实例：

```
1.  $db = DB::get_db();
2.  //普通 SQL
3.  $res = $db->query("select * from a, b where a.id<3 and a.name=b.name and b.sex='男'");
4.  //参数查询
5.  $res = $db->query("select * from a, b where a.id<? and a.name=b.name and b.sex=?",array('3','男'));
```

结果的形式类似于：array(array('id'=>'1','sex'=>'女'),array('id'=>'2','sex'=>'男'))。

execute

```
1.  DB::execute(string $sql, array $pam)
2.  执行无结果查询(增，删，改)
3.
4.  Parameters:
5.  string $sql 要执行查询的 SQL 语句参数请用"?"代替
6.  array $pam 数字下标的数组，数组项依次替换 SQL 中的"?"参数
```

execute 的参数传入方式和 **query** 相同。

其他方法

begin_transaction

```
1.  DB::begin_transaction()
2.  开启事务
```

rollback

```
1.  DB::rollback()
2.  事务回滚
```

commit

```
1.  DB::commit()
2.  事务提交
```

list_fields

```
1.  DB::list_fields(表名 $table)
```

```
2.  获得数据库表字段名称一维数组
3.  成功获得后将存入 静态变量中 作为数据缓冲
4.
5.  Parameters:
6.  string $table 表名
7.
8.  Returns:
9.    array 数据表的字段名称集合
```

list_tables

```
1.  DB::list_tables()
2.  获得数据库表名称集合
3.
4.  Returns:
5.    array 数据库表名称集合
```

version

```
1.  DB::version()
2.  获得数据库版本
3.
4.  Returns:
5.    string 版本号
```

数据库模型

数据库模型的相关 DB 操作方法请参阅数据库模型。

参数条件

数据库模型支持复杂的参数条件拼接，大多数情况下这些参数拼接条件就足以完成常用的查询需求。

简单匹配：

```
1.  $u = new Model('user');
2.  $u->where(array('name'=>'张三'));
```

匹配的条件为：

```
1.  where name = '张三'
```

对比匹配：

```
$u = new Model('user');
$u->where(array('name'=>'张三','age@<'=>30,'age@>='=>20));
```

匹配的条件为：

```
where name = '张三' and age <30 and age >=20
```

相似度匹配：

```
$u = new Model('user');
$u->where(array('name'=>'张三','age@<'=>30,'age@>='=>20,'local1@~'=>'北京','local2@|~'=>'北','local3@~|'=>'京'));
```

匹配的条件为：

```
where name = '张三' and age <30 and age >=20 and local1 like '%北京%' and local2 like '北%' and local3 like '%京'
```

空匹配：

```
$u = new Model('user');
$u->where(array('local1@~'=>'北京','local2@|~'=>'北','local3@~|'=>'京','part'=>null,'level'=>true));
```

匹配的条件为：

```
where  local1 like '%北京%' and local2 like '北%' and local3 like '%京' and part is null and level is not null
```

范围匹配：

```
$u = new Model('user');
$u->where(array('part'=>null,'level'=>true,'height'=> array('1.5','1.6','1.7')));
```

匹配的条件为：

```
where   part is null and level is not null and (height = '1.5' or height = '1.6' or height = '1.7')
```

或者匹配：

```
$u = new Model('user');
```

```
2.  $u->where(array('height'=> array('1.5','1.6','1.7'), 'or' => array('id'=>'2','id'=>'3')));
```

匹配的条件为：

```
1.  where  (height = '1.5' or height = '1.6' or height = '1.7') and (id='2' or id='3')
```

直接或者匹配

```
1.  $u = new Model('user');
2.  $u->where(array('or' => array('id'=>'2','id'=>'3')));
```

匹配的条件为：

```
1.  where  id='2' or id='3'
```

当然如果以上规则不好理解你可以使用常规方式，例如：

$u->where(" (height = ? or height = ? or height = ?) and (id=? or id=?)",array('1.5','1.6','1.7','2','3'));

版权申明

发布本资料须遵守开放出版许可协议 1.0 或者更新版本。