مرجع و راهنمای عملی

فارسی

yiiframework

ترجمه و تألیف: علی برجیان

تابستان ۱۳۹۰

www.CoolHost.ir طرح جلد و نشر (مخصوص کتابخوان)

این کتاب برای استفاده در اینترنت و به صورت رایگان ارائه شده است.
هرگونه استفاده از مطالب با ذکر منبع بلامانع است.

m. متد ها و متغیرهای تعریف شده در class CActiveRecordMetaData موجود در فایل CActiveRecord
n. دسترسی به مدل از طریق کنترلر
o. نکات لازم در طراحی مدل

۵. ویو ها

a. معرفی ویو ها
b. ارسال پارامترها و مقادیر آنها از کنترلر به ویو
c. فراخوانی یک ویو در ویو دیگر
d. دسترسی به کنترلر از داخل ویو
e. دسترسی به ویو از طریق کنترلر
f. دسترسی به خروجی مدل در ویو
g. نکات لازم در طراحی view
h. معرفی قالب ها
i. قرار دادن یک layout درون layout دیگر
j. Partial view
k. System View
l. Widget
m. ساخت یک Widget دلخواه
n. Customizing Widgets Globally
o. Portlet
p. CActiveDataProvider
q. کلاس CHtml
r. Theming

۶. بانک های اطلاعاتی

a. انواع روش کار با بانک اطلاعاتی
b. اتصال به بانک اطلاعاتی
c. DAO
    i. اجرای دستورات SQL
    ii. فراخوانی نتایج
    iii. نحوه کار با تراکنش ها
    iv. انقیاد پارامتر ها
    v. انقیاد فیلدها
    vi. استفاده از Table Prefix
d. Query Builder
    i. مزایای روش Query Builder
    ii. ایجاد دستورات
    iii. ایجاد پرس و جو ها
    iv. اجرای پرس و جو ها
    v. به دست آوردن عبارت SQL ایجاد شده
    vi. به کار بردن دستورات جایگزین
    vii. کار با چند پرس و جو
    viii. پرس و جو های کار با داده
    ix. پرس و جو های کار با ساختار
e. Active Record
    i. اتصال به بانک اطلاعاتی

ii. تعریف AR Class
iii. ساخت رکورها
iv. دسترسی به مقدار یک فیلد از رکورد
v. استفاده از توابع هاست
vi. به روز رسانی رکوردها خواندن رکورد ها
vii. حذف رکورد ها
viii. اعتبار سنجی داده ها
ix. چند متد مفید در کلاس مدل
x. به کارگیری تراکنش ها در AR
xi. Named Scopes

f. بانک اطلاعاتی رابطه ای
i. نحوه تعریف یک رابطه در مدل
ii. اجرای پرس و جو های رابطه ای
iii. اجرای پرس و جو های رابطه ای بدون استفاده از مدل های داری رابطه
iv. گزینه های پرس و جو های رابطه ای
v. فیلد های با نام تکراری
vi. گزینه های پرس و جوی رابطه ای پویا
vii. کارایی پرس و جو های در رابطه
viii. پرس و جو های استاتیک
ix. پرس و جو های رابطه ای با Named Scopes
x. پرس و جو های رابطه ای با through

g. مهاجرات بانک اطلاعاتی

۷. ماژول ها و کامپوننت ها

a. معرفی ماژول
b. مزایای استفاده از ماژول
c. ساخت یک ماژول جدید
d. نحوه استفاده از ماژول
e. معرفی کامپوننت ها
f. نحوه دسترسی به یک Component
g. انواع Component
h. تعریف واستفاده از یک کامپوننت
i. کامپوننت های پیش فرض

۸. تشخیص هویت کاربر

a. تشخیص هویت کاربر و حدود دسترسی

۹. فرم ها

a. معرفی فرم ها
b. نحوه ایجاد فرم جدید
c. یک فرم و چند مدل

۱۰. Caching

a. معرفی caching
b. Data Caching
c. وابستگی کش
d. Fragment caching
e. گزینه های کش
f. وابستگی داده ها

g. تنوع داده ها
h. درخواست نوع
i. کش کردن تو در تو
j. Page Caching
k. محتوای پویا

۱۱.توسعه yii

a. معرفی توسعه ها
b. استفاده از توسعه ها
c. ایجاد توسعه ها
d. ساخت توسعه های گوناگون
e. استفاده از کتابخانه های rd-Party۳

۱۲.تست

a. انواع روش تست
b. TDD
c. تست Bootstrap

۱۳.مدیریت آدرسها

a. ایجاد URL ها
b. استفاده از پارامترهای نام گزاری شده
c. مخفی کردن index.php

۱۴.مثال ها

۱۵.ضمیمه ها

سخن مولف :

هر روز شاهد پیشرفتهای جدیدی در حوزه فناوری اطلاعات می باشیم. همزمان با توسعه سخت افزار ها و تولید قطعات با قابلیت های پیشرفته، نرم افزار ها نیز به صورت موازی پیشرفت می کنند. نرم افزار هایی که هر کدام فضاهای جدیدی را در اختیار کاربران خود قرار می دهند و در این میان سهم برنامه نویسان و طراحان وب به عنوان تولید کنندگان نرم افزار بسیار بیشتر است.

فریم ورک Yii یک بستر نرم افزاری با سرعت و بازدهی بسیار بالا است که به وسیله زبان PHP توسعه یافته است و توانایی تولید انواع نرم افزارهای تحت وب با مقیاس های گوناگون را دارا می باشد.

هدف از تالیف این کتاب کمک به پیشرفت برنامه نویسان فارسی زبانی است که به تازگی کار با این نرم افزا را آغاز نموده اند و قصد آشنایی با این محیط نرم افزاری را دارند همچنین سعی شده است تا حد امکان جزئیات مطالب نیز لحاظ شود تا کتاب به صورت یک مرجع برای برنامه نویسان تبدیل شود.

این کتاب شامل ترجمه هایی است که عینا از راهنمای عملی فریم ورک در سایت رسمی و کتاب های منتشر شده در این مورد برگردان شده است همچنین راهنمای کدهای داخل فایلهای فریم ورک نیز مورد توجه قرار گرفته است و در موارد اندکی تجربیات و مثالهای عملی مولف آورده شده است.

تا این تاریخ که نسخه ۱.۱.۸ فریم ورک ارائه شده سعی شده است مطالب به صورت یکپارچه و منظم ارائه شود و کلیه امکانات نسخه های قبلی پوشش داده شود.

مطمئنا این کتاب نیز دارای اشکالاتی می باشد که با راهنمایی و توجه شما در نسخه های بعدی اصلاح خواهد شد. در صورت داشتن هرگونه سوال، نظر و یا پیشنهاد لطفا از طریق آدرس ایمیل web.ebox@gmail.com آن را با مولف در میان بگذارید.

امیدوارم این کتاب بتواند گامی هرچند کوچک در راه ارتقای علمی برنامه نویسان فارسی زبان برداشته باشد.

علی برجیان

شهریور ۱۳۹۰

Yii Framework چیست؟

یکی از خواسته های برنامه نویسان و طراحان سایت های اینترنتی انتخابیکPlatform مناسب برای پیاده سازی سایت ها، برنامه های تحت وب و پورتال ها می باشد. شاید برای کاربران ویندوز و عموما برنامه نویسان ASP.NET گزینه های زیادی برای انتخاب مطرح نباشد و MS dotNet Framework به عنوان Platform اصلی مورد استفاده قرار گیرد. اما برای برنامه نویسان جامعه متن باز به خصوص برنامه نویسان PHP انتخاب یک Platform مناسب مقداری پیچیده است. پیدا کردن Platform مناسب برای برنامه نویسانی که سطح متوسط برنامه نویسی PHP را پشت سر گذاشته اند و قصد ورود به فضاهای جدید با قابلیت های بیشتر را دارند امری مهم به شمار می رود.شاید نام بسیاری از این Platform ها را شنیده باشید که البته تعداد آنها کم هم نیستند مانند :ZooP - Zend - Prado - CakePHP

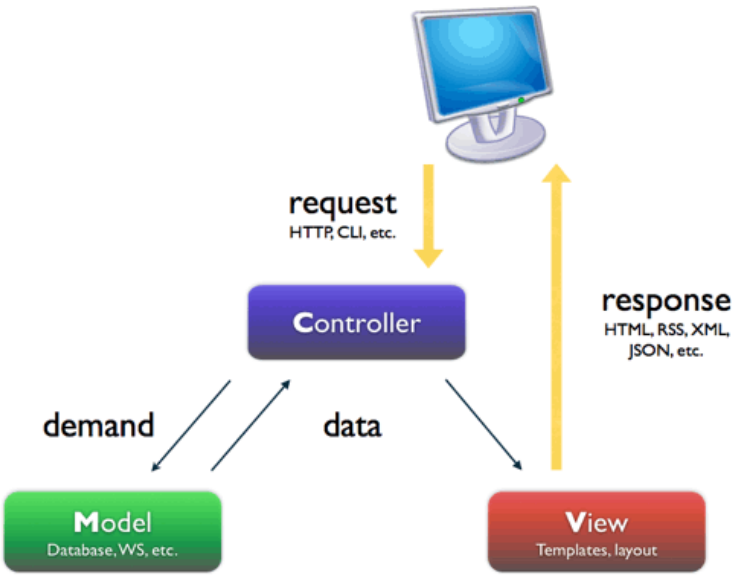
فریم ورک Yii (خوانده می شود ای مانند حرف E کشیده یا تلفظ Yee) نیز همانند سایر این برنامه ها یک Platform برای استفاده برنامه نویسان PHP است که در سال ۲۰۰۸ توسط تعدادی از برنامه نویسان مجرب و حرفه ای که مدت زمان زیادی را بر روی پروژه های توسعه PHP Frameworks (مثل پروژه Prado) صرف کرده بودند عرضه شد. این پروژه هر چند عمر طولانی ندارد ولی به قدری موثر واقع شده که در مدت کوتاهی توانسته است نظر بسیاری از توسعه دهندگان وب را به خود جلب نماید. Yii بسیاری از قابلیت های پروژه موفق Prado را به ارث برده است. پروژه Yii توسط شرکت Yii Software LLC یا YiiSoft ارائه شده است. Yii Frameworkیک نرم افزار رایگان می باشد که گواهینامه BSD License را نیز اخذ نموده است.

برخی از ویژگی های Yii Framework

Yii تنها از PHP نسخه ۵ و بالاتر پشتیبانی می کند. در مقایسه با فریم ورکهایی مانند CakePHP که از PHP 4نیز پشتیبانی می کنند این مسئله یک کمبود به نظر میرسد ولی همیشه اینطور نیست. فریم

ورکهایی مانند CakePHP برای ایجاد تطابق با نسخه های ۴ ناچارند که خیلی از قابلیت های نسخه ۵ را نادیده بگیرند یا به گونه ای آنها را تغییر دهند که باعث ایجاد تطابق گردد ولی Yii Framework با تمرکز بر روی نسخه ۵ PHP از همه قابلیت های آن استفاده می کند و این مسئله در سال ۲۰۱۱ که اکثر میزبانها از PHP 5 حمایت می کنند نه تنها یک کمبود نیست بلکه یک مزیت به شمار می رود.

*در ساخت فریم ورک Yii از الگوی طراحی MVC استفاده شده است.*MVC مدلی است که در مهندسی نرم افزار معرفی می شود و در این مدل اجزای برنامه به سه قسمت اصلی کنترل کننده –مدل– دیدگاه۱تقسیممیشوندوجریانداده‌هادرآنبرقرارمیشود. Yii تمامی قواعد MVC را رعایت کرده است الگوی کاری MVC در شکل زیر آمده است:



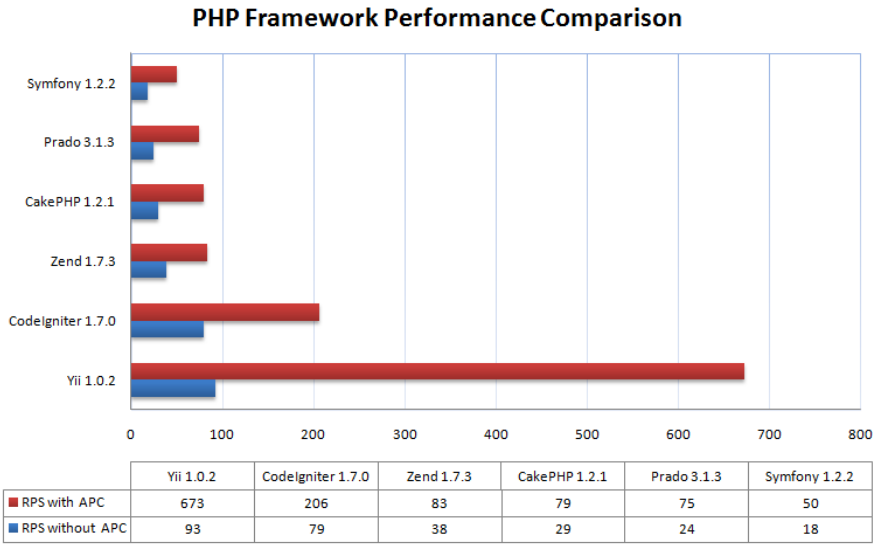
یک محیط کاملا شی گرا. در Yii شی گرایی به عنوان یک اصل اولیه رعایت شده است و دسترسی به همه چیز حتی دسترسی به جداول بانک اطلاعاتی و روابط و ... نیز در قالب شی گرایی تعریف می شود که این مسئله در مقابل فریم ورکهایی مثل CakePHP که همچنان از دسترسی های رابطه ای پشتیبانی می کنند یک مزیت عمده است.

*از قابلیت برنامه نویسی generic حمایت می کند.*generic یک قابلیت برنامه نویسی است که هدف از آن پشتیبانی از انواع داده های ورودی بدون قید محدودیت نوع است و به عنوان یکی از روشها و تکنیک های نوین برنامه نویسی استفاده می شود.

هماهنگی با JQuery. JQuery یک Platform براساس زبان JavaScript است که قابلیت های فوق العاده بصری را در محیط کاری کاربر ایجاد می کند. Yii Framework با این Platform هماهنگی و ارتباط مستقیمی را ایجاد می کند و امکان استفاده از قابلیت های AJAX را فراهم می کند.

*از ORM پشتیبانی می کند.*Object Relational Mapping روشی برای تطابق با شی گرایی است. از آنجایی که بانک اطلاعاتی MySQL یک بانک اطلاعاتی رابطه ای است لذا در روش ORM یک بانک اطلاعاتی مجازی از بانک اصلی MySQL ساخته می شود و عملیات بر روی آن با استفاده از روش شی گرا انجام می شود و سپس این بانک اطلاعاتی مجازی بر روی بانک اصلی Update می شود. برای برنامه نویسان .NET. این موضوع آشنا است و در ADO.NET مفهوم بانک اطلاعاتی مجازی در قالب Dataset ارائه می شود.

استفاده از حالت Caching پیشرفته .Yii از یک حالت Caching بسیار پیشرفته برای کار با بانک اطلاعاتی استفاده می کند که یک مزیت عمده است و باعث سبک شدن فضای کاری و افزایش بسیار زیاد بازدهی میشود. به نموداری که معرف بازدی Yii Framework در مقایسه با چند فریم ورک معروف است توجه کنید :



	Yii 1.0.2	CodeIgniter 1.7.0	Zend 1.7.3	CakePHP 1.2.1	Prado 3.1.3	Symfony 1.2.2
RPS with APC	673	206	83	79	75	50
RPS without APC	93	79	38	29	24	18

مشاهده می شود میزان بازدهی آزمایش شده برای Yii بسیار بیشتر از رقبای قدیمی خود است.

چند مورد از ویژگی های دیگر :

- پشتیبانی از AJAX
- پشتیبانی از بانک اطلاعاتی چند گانه MultipleDB
- پشتیبانی از Templates
- پشتیبانی از Modules
- پشتیبانی ازAuthentication
- پشتیبانی از Validation

پیش نیازهای قبل از شروع کار

برای شروع کار با Yii کاربر باید با اصول اولیه برنامه نویسی آشنایی داشته باشد. همچنین اصول اولیه طراحی وب و آشنایی با دستورات PHP ضروری است. کاربر باید با مفاهیم شی گرایی و همچنین بانک های اطلاعاتی آشنا باشد. تعدادی از پیش نیاز ها در زیر آمده است:

- XHTML
- CSS
- JavaScript & JQuery
- PHP
- Object Oriented Programming
- SQL Language & PhpMyAdmin

چه برنامه هایی را با Yii می توان تولید کرد

فریم ورک Yii مناسب برای طراحی سایت ها و برنامه های تحت وب با ترافیک بالا است. سایت های مرتبط با تجارت الکترونیک، پورتال ها، فروم ها و سیستم های مدیریت محتوا از جمله برنامه هایی هستند

که می توان توسط Yii آنها را پیاده سازی نمود.امنیت، سرعت و کارایی از جمله مهمترین مشخصه های برنامه های تولید شده توسط Yii هستند.Yii یک فرم ورک مناسب برای پیاده سازی سایت های حرفه ای و پیشرفته با میزان کارایی بالا است و قابلیت های گسترش زیادی را در اختیار کاربران خود قرار می دهد.

نصب Yii Framework

نصب فریم ورک شامل مراحل زیر است :

اگر قصد نصب فریم ورک بر روی localhost را داریم باید host server مناسب را نصب نماییم. پیشنهاد می شود که از نرم افزار XAMP Serverاستفاده نمایید هر چند می توان از برنامه های دیگر همانند Wamp Server نیز استفاده کرد. نکته مهم این است که هاست مورد استفاده از PHP نسخه ۵.۱یا بالاتر پشتیبانی نماید.آخرین نسخه فریم ورک را از قسمت downloads سایت www.yiiframework.com دریافت نمایید. این فایل در قالب tar یا zip می باشد. این فایل را از حالت فشره خارج می کنیم. و در یک پوشه با عنوان yii قرار میدهیم.پوشه yii ایجاد شده را در مسیر localhost قرار می دهیم. اگر از XAMP Server استفاده می کنیم این مسیر چیزی مانند C:\xampp\htdocs است و اگر از Wamp Server استفاده می کنیم C:\wamp\www می باشد.

در سایر هاست ها باید پوشه مورد نظر را پیدا کنیم.نکته : قرار دادن پوشه فریم ورک در این مسیر الزامی نمی باشد و بعدا می توان این مسیر را به هر جای دیگر تغییر داد.در این مرحله Localhost Server را Start می کنیم و سرویس های Apache و MySQL را راه اندازی می کنیم و سپس در مرورگر خود آدرس زیر را باز می کنیم :

http://localhost/yii/requirements/index.php

در این صفحه نیازمندی های مورد نیاز نشان داده شده است. گزینه های Failed شده باید بررسی و سرویس های مرتبط با آنها فعال گردند تا از حالت Failed خارج شوند. دو سرویس PDO extension و PDO SQLite extension برای شروع به کار با Yii ضروری هستند و باید فعال باشند.

Name	Result	Required By	Memo
PHP version	Passed	Yii Framework	PHP 5.1.0 or higher is required.
$_SERVER variable	Passed	Yii Framework	
Reflection extension	Passed	Yii Framework	
PCRE extension	Passed	Yii Framework	
SPL extension	Passed	Yii Framework	
DOM extension	Passed	CHtmlPurifier, CWsdlGenerator	
PDO extension	Passed	All DB-related classes	
PDO SQLite extension	Passed	All DB-related classes	This is required if you are using SQLite database.
PDO MySQL extension	Passed	All DB-related classes	This is required if you are using MySQL database.
PDO PostgreSQL extension	Warning	All DB-related classes	This is required if you are using PostgreSQL database.
Memcache extension	Warning	CMemCache	
APC extension	Warning	CApcCache	
Mcrypt extension	Passed	CSecurityManager	This is required by encrypt and decrypt methods.
SOAP extension	Passed	CWebService, CWebServiceAction	
GD extension with FreeType support	Passed	CCaptchaAction	

passed failed warning

در این مرحله باید مسیر های سیستم را اصلاح نماییم برای این کار در ویندوز در صفحه MyComputer کلیک راست کرده و گزینه Properties را انتخاب می کنیم و سپس در قسمت Advanced Settings در قسمت Advanced وارد می شویم و گزینه Environment Variables را انتخاب می کنیم و در قسمی System Variables گزینه Path را انتخاب کرده و مسیر قرار گیری فریم ورک و همچنین مسیر قرار پیری فایل php.exe که در Localhost Server قرار دارد را اضافه می کنیم مانند زیر :

C:\xampp\htdocs\yii\framework;C:\xampp\php

حال سیستم را یک بار راه اندازی مجدد میکنیم.

ساخت اولین پروژه

پس از نصب فریم ورک برای ساخت اولین پروژه به شکل زیر عمی می کنیم . ابتدا در ویندوز در قسمت run عبارت cmd را تایپ می کنیم و وارد Command Prompt میشویم.سپس با دستور cd وارد مسیر web root می شویم مثلا اگر از XAMP Server استفاده می کنیم :

Cd C:\xampp\htdocs

سپس از ابزار yiic برای ساخت پروژه مورد نظر استفاده می کنیم :

yiic webapp demo

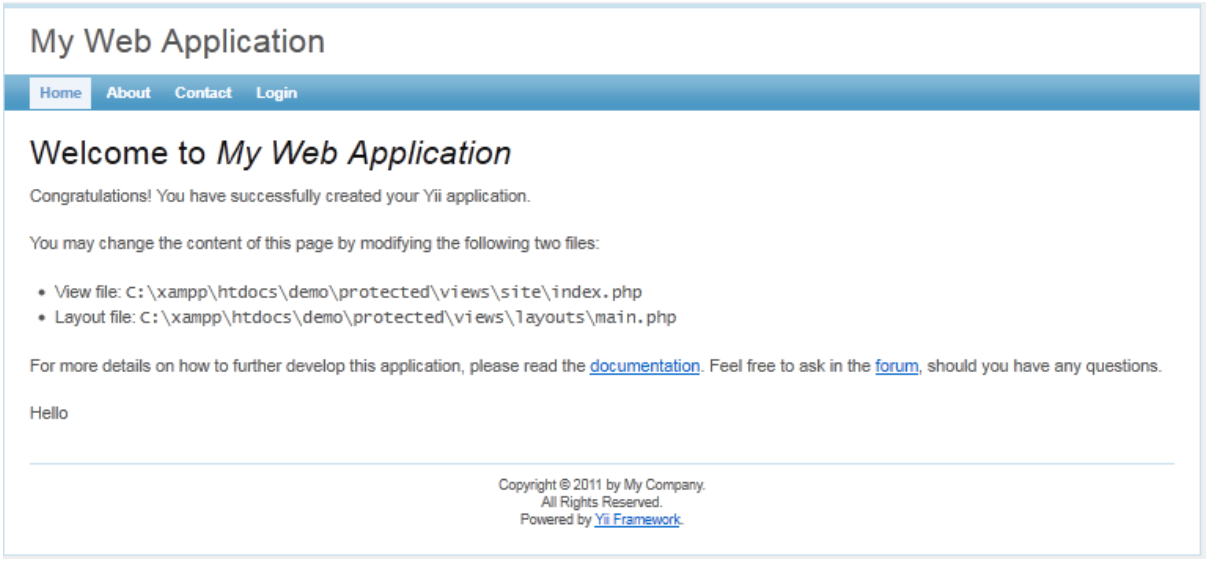
سپس یک سوال پرسیده می شود که y یا yes را وارد می کنیم

Create a Web application demo? [Yes|No] yes

.....

این کار باعث ساخت یک پوشه با نامdemo در مسیر جاری webroot می شود. برای مشاهده پروژه ساخته شده مسیر زیر را در مررگر خود وارد می کینم :

http://localhost/demo/index.php



پروژه ساخته شده دارای ۴ صفحه homepage, about page, contact page login page است. به این پروژه ساخته شده skeleton می گویند که شامل این ۴ صفحه پیش فرض و ابتدایی است و بعدا صفحات و محتویات آن توسط ما ویرایش می شود.

نکته: مسیر پوشه yii framework باید حتما در path قرار داده شده باشد در غیر این صورت برنامه yiic اجرا نخواهد شد و برای اجرای yiic باید مسیر فایل php.exe نیز در path موجود باشد.

ابزار خط فرمانyiic

پس از نصب Yii Framework برای ایجاد یک پروژه جدید به شکل زیر عمل می کنیم .به سطر فرمان رفته و دستور زیر را وارد می کنیم :

Yiic webapp [path/] project-name

مثال:

Yiic webapp c:\xampp\htdocs\project1

ساختار کامل دستور yiic به شکل زیر است:

yiic <command-name> [parameters...]

The following commands are available:

- message

- migrate

- shell

- webapp

سوییچ Message

yiic message <config-file>

ترجمه یک متن به آرایه قابل فهم توسط PHP و ذخیره آن در یک فایل config

سوییچ Migrate

yiic migrate [action] [parameter]

جهت استفاده از قابلیت database migrations به کار می رود.

سوییچ Shell

yiic shell [entry-script | config-file]

این دستور جهت کار بر یک پروژه ایجاد شده درخط فرمان به کار می رود.این سرویس همچنینابزارهایی جهت تولید خودکار کنترلرها،مدل ها و ویوها را دارد.توصیه می شود کهاین دستوررا از داخل دایرکتوری اصلی برنامه اجرا نمایید.پارامترهای entry-script و config-file اختیاری هستند و مسیرفایل پیکربندیبرای برنامه را نشان می دهند و اگر این مسیر آورده نشود به طور پیش فرض فایل 'index.php' موجود در همین دایرکتوری را نشان می دهد.

به وسیله سوییچshellمی توان تمامی عملکرد gii را پیاده سازی نمود

```
yiic shell

 - controller
 - crud
 - form
 - help
 - model
 - module
```

سوییچ Webapp

yiic webapp <app-path>

این دستور جهت ایجاد یک پروژه جدید در مسیر و شاخه مرود نظر استفاده می شود.app-path حتما مورد نیازاست و آن شاخه ای است که در آن پروژه جدید ایجاد خواهد شد و اگر در هم که آن دایرکتوری وجود ندارد ، آن را ایجاد خواهد کرد. باید مطمئن شوید که دایرکتوری توسط کاربران وب سایت قابل دسترسی باشد.

پوشه ها و فایلهای ساخته شده :

index.php	فایل entry script
index-test.php	فایل entry script برای functional tests
assets/	فایلهای مرجع و مورد نیاز که به طور خودکار تولید می شوند. در صورت حذف محتویات این پوشه مجددا تولید می شوند ولی خود پوشه نباید حذف شود
css/	شامل CSS های اصلی برنامه
images/	شاملimageهای اصلی برنامه
themes/	های برنامهقالبشامل
protected/	

شامل پوشه ها و فایل های اصلی مورد نیاز پروژه که به آن Application Base Directory می گویند. مسیر این پوشه از طریق نام مستعار application قابل دسترسی است. کلیه محتوای این پوشه باید از دسترس کاربران محفوظ بماند.

yiic	yiic command line script برای Unix/Linux
yiic.bat	yiic command line script برای Windows
yiic.php	yiic command line PHP script

commands/ شامل دستورات yiic

shell/ شامل دستورات 'yiic shell'

components/ شامل کامپوننت های برنامه

Controller.php the base class for all controller classes

UserIdentity.php the 'UserIdentity' class used for authentication

config/ فایل تنظیمات برنامه

console.php the console application configuration

main.php the Web application configuration

test.php the configuration for the functional tests

controllers/ شامل فایلهای کلاس کنترلر

SiteController.php the default controller class

data/ شامل فایلهای مرتبط با بانک اطلاعاتی

schema.mysql.sql the DB schema for the sample MySQL database

schema.sqlite.sql the DB schema for the sample SQLite database

testdrive.db the sample SQLite database file

extensions/

شامل برنامه ها و اضافات third-party است.

messages/ شامل پیامهای ترجمه شده برای چند زبانی

models/ شامل فایلهای کلاس مدل

LoginForm.php the form model for 'login' action

ContactForm.php the form model for 'contact' action

runtime/

شامل فایلهای موقت و کمکی ساخته شده در حین اجرا است که به طور اختصاصی برای سرویس دهنده وب باید قابل دسترسی باشد.

tests/ شامل اسکریپت های تست برنامه

views/ شامل ویو های برنامه

layouts/ شامل قالب های برنامه

main.php the base layout shared by all pages

column1.php the layout for pages using a single column

column2.php the layout for pages using two columns

site/ containing view files for the 'site' controller

pages/ containing "static" pages

about.php the view for the "about" page

contact.php the view for 'contact' action

error.php the view for 'error' action (displaying external errors)

index.php the view for 'index' action

login.php the view for 'login' action

modules/

شامل ماژول های برنامه است و هر ماژول تحت یک پوشه مجزا قرار می گیرد

محیط مناسب برای کد نویسی

برای کار با پروژه Yii می توان از هر نوع IDE مرتبط با PHP استفاده نمود و یا حتی ویرایشگرهای متنی مثل notepad یا ++notepad مفید خواهند بود. اما ابزارهای حرفه ای طراحی مثل Eclipse با PDT که همان PHP Dev Tools است مناسب تر می باشند. همچنین برنامه netbeans محیطی مناسب است. از دیگر ابزار ها همچون Microsoft Visual Studio و یا MS Expression Web نیز می توان استفاده نمود

مراحل ده گانه تولید برنامه :

۱- ساخت اسکلت برنامه توسط yiic

۲- اعمال تنظیمات در فایل پیکره بندی config

۳- ساخت مدل های مورد نیاز توسط ابزار gii

۴- ساخت کنترلرهای برنامه توسط ابزار gii

۵- نوشتن کد های مورد نیاز در کنترلر و ایجاد اکشن ها و نوشتن کدهای مورد نیاز در مدل و ویوها

۶- ایجاد فیلرهای مورد نیاز برای محدود سازی دسترسی ها

۷- ایجاد theme های مورد نیاز در صورت لزوم

۸- ایجاد ترجمه های مورد نیاز پروژه

۹- انجام تنظیمات caching

۱۰- اصلاحات و tune up نهایی قبل از انتشار

چهار قسمت اصلی هر برنامه :

هر پروژه ایجاد شده می تواند در چهار قسمت زیر تقسیم بندی شود.

۱- Front end : یک قسمت public که کاربر نهایی (end user) مستقیما با آن ارتباط برقرار می کند مانند bootstrap یا یک فرم ویو و ..

۲- Back end : یک قسمت Protected که کاربر نهایی به آن دسترسی ندارد و تنها سرویس دهنده وب به آن دسترسیی دارد.

۳- Console : یک برنامه که تحت CLI (Command Line Interface) اجرا می شود و جهت انجام برخی اعمال همانند پشتیبانی و مدیریت پروژه اصلی به کار می رود.

۴- Web API : قسمتی از برنامه که مسئول مار کردن با third-party ها می باشد.

MVC و اجرای برنامه

MVC چیست ؟

MVC حرف ابتدای سه کلمه Model View Controller است که معرف یک مدل پیاده سازی نرم افزاری است و در مهندسی نرم افزار مطرح می شود. هدف ازاین مدل جداسازی قسمت های مختلف برنامه از یکدیگر است به گونه ای که جریان داده ای بین آنها حفظ شود. این مدل شامل ۳ قسمت اصلی می باشد که عبارتند از :

۱- مدل : معرف داده ها و اطلاعات می باشد و همچنین قوانین و مقررات مربوط به نوع داده ها.

۲- ویو : شامل اجزای واسط کاربر مانند جعبه های ورودی، فرم ها و سایر اجزای ورودی و خروجی می باشد.

۳- کنترلر : جریان داده ها بین کنترلر و ویو را هدایت می کند.

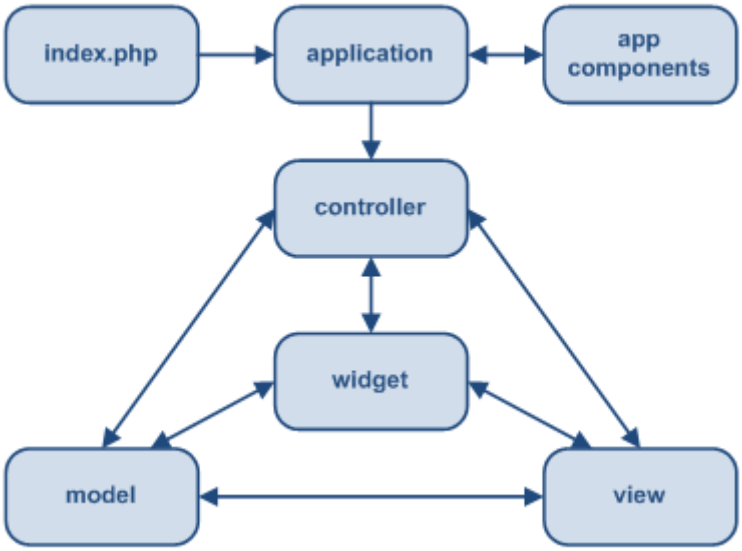
مزایای اصلی استفاده از مدل MVCعبارتند از ۱- قابلیت استفاده مجدد و ۲- جداسازی کدها و محتواها

: Application

Yii در کنار مدل MVC یک مفهوم دیگری به نام Application را نیز معرفی می کند. که به آن front-controller می گویند. Application درخواستهای کاربر را گرفته و آن را برای کنترلر ارسال می نماید. Application کلاسی از نوع singleton است و در هر کجای پروژه می توان از طریق دستور Yii::app() به آن دسترسی پیدا کرد. کلاس singleton کلاسی است که تنها یک شی از روی آن ساخته می شود و هیچ کلاس دیگری از روی آن ساخته نخواهد شد. لذا در هر پروژه تنها میتوان یک نمونه از Application داشته باشیم که در آن هم یک شی ساخته شده از روی کلاس Application است. یک نمونه از Application در حافظه RAM تولید شده و به عنوان یک شی مورد استفاده قرار می گیرد و پس از اتمام برنامه از حافظه پاک می شود. دسترسی به Application از طریق Yii::app() :

```
// مشاهده مقادیر
echo  Yii::app()->defaultController;
```

```
echo Yii::app()->theme;
// تغییر به مقادیر
Yii::app()->name = "My Project Name";
```



هر Applicationتوسط کلاس CWebApplication ساخته می شود که در مسیر yii\framework\web\CWebApplication.php قرار دارد.مسیر قرارگیری yii.php به شکل زیر تعیین می شود:

```
$yii=dirname(__FILE__).'/../yii/framework/yii.php';
```

پس application نمونه ای ساخته شده از روی کلاس yii است که تنظیمات config در آن اعمال شده است.پوشه Protected محل قرار گیری فایلها و پوشه های اصلی برنامه است که به آن پوشه Application نیز می گویند و باید از دسترس دیگران محافظت شود لذا یک فایل مخفی با نام .htaccess ایجاد کرده و محتوای آن را برابر deny from all قرار می دهیم.آدرس دسترسی به آن :

```
dirname(__FILE__).'/protected
```

: Entry Script

Entry Script به فایل index.php موجود در شاخه اصلی گفته می شود که شی اصلی برنامه یعنی Application را ایجاد می کند. به فایل index.php اصلی Entry Script یا اسکریپت ورودی گفته می

شود.در این فایل تنظیمات دیگری نیز قرار داده می شود. محتوای این فایل به طور پیش فرض به شکل زیر است.

```
// change the following paths if necessary
$yii=dirname(__FILE__).'/../yii/framework/yii.php';
$config=dirname(__FILE__).'/protected/config/main.php';
// remove the following lines when in production mode
defined('YII_DEBUG') or define('YII_DEBUG',true);
// specify how many levels of call stack should be shown in
each log message
defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);
require_once($yii);
Yii::createWebApplication($config)->run();
```

$yii : مسیر قرار گیری yii.php را مشخص می کند که به آن Bootstrap گفته می شود.

$config : مسیر قرار گیری فایل Application یا همان config file را مشخص می کند.

defined('YII_DEBUG') or define('YII_DEBUG',true); در حالت debug mode از این کد استفاده می شود و هنگام publish سایت که به آن Production mode می گوییم دیگر نیازی به این کد نیست.

defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3); اگر حالت logging فعال باشد مشخص می کند که هر چند سطح خطا باید در یک پیام نشان داده شود.

require_once($yii); فایل yii.php را فراخوانی می کند.

Yii::createWebApplication($config)->run(); توسط این دستور یک نمونه از کلاس Yiiساخته می شود و پاراlترهای موجود در فایل config در آن اعمال می شوند و سپس اجرا می شود.

: Bootstrap

اسکریپت bootstrap همان فایل yiic.php موجود در Framework است که کلیه درخواستهای کاربر را در دریافت می کند و برای کنترلر مربوطه ارسال می کند.Bootstrap تنها اسکریپتی است که کاربر سایت می تواند به آن دسترسی مستقیم داشته باشد. با هر درخواست کاربر bootstrat اجرا می شود و بقیه کارها را دنبال می کند.

Config :

فایل Config فایلی است که تنظیمات کلی پروژه در آن قرار دارد.مسیر فایل config را در فایل bootstrap به شکل زیر تعیین می کنیم :

```
$config=dirname(__FILE__).'/protected/config/main.php';
```

هر application ساخته شده توسط bootstrap در واقع نمونه ای از فایل yiiBase.php موجود در framework است که تنظیمات config در آن اعمال شده است:

```
Yii::createWebApplication($config)->run();
```

Config در واقع آرایه ای است که مقادیر و تنظیمات پروژه در آن قابل تعریف است.اگر فایل config خیلی پیچیده و طولانی باشد می توان آن را در چند فایل جدا تقسیم کرد و نهایتا همه را به یک نقطه مشترک پیوند داد. config می تواند در فایل protected/config/main.php ذخیره شده باشد و یا در یک فایل دیگر که برای امنیت بیشتر در هنگام Production می توان آن را در فایل دیگری قرار داد و مسیر Config را در index.php اصلی برنامه اصلاح نمود.

مسیر پیش فرض config

```
protected/config/main.php';
```

مثالی از نحوه انجام تنظیمات در config

```
array(
'name'=>'Yii Framework',
'defaultController'=>'site',
)
```

محتوای کلی config تنها یک آرایه است به شکل :

```
<?php
return array(...);
?>
```

در قسمت ... آرایه های داخلی برای تنظیمات مختلف تعریف می شوند.

نکته : اگر با اشکالی در مورد تابع date_default_timezone_set در حین اجرای برنامه مواجه شویم هم می توان فایل php.ini را تغییر داد که روش چندان مناسبی نیست و هم می توان تنظیم زیر را را در فایل configدر همان ابتدا ی فایل وارد کرد و مشکل حل خواهد شد.

```
'timeZone'=>"Asia/Tehran',
```

انواع حالت اجرای برنامه :

برنامه Yii می تواند در دو حالت متفاوت اجرا شود :

Debug Mode – 1

اجرای برنامه در حالت ساخت که با ثبت رویدادهای زیاد و نشان دادن خطاها همراه است و برای حالت برنامه نویسی مناسب است و کارایی کمتری دارد. این حالت را Development Mode نیز می گوییم.

Production Mode - 2

اجرای برنامه کامل شده با کارایی بالا و حالت های خطایابی و ثبت رویدادهای کمتر.

برای تعیین Mode اجرای برنامه در فایل index.php کد زیر را ویرایش می کنیم :

برای حالت Debug Mode کد زیر را اضافه کرده و برای حالت Production Mode این کد را حذف می کنیم.

```
defined('YII_DEBUG') or define('YII_DEBUG',true);
```

مراحل ساخت برنامه :

ساخت skeleton توسط فرمان yiic

انجام تنظیمات اولیه برنامه در فایل config و ساخت component های مورد نیاز در برنامه

ساخت مدل ها توسط ابزار gii

ساخت کنترلر های مورد نیاز توسط gii

ساخت اکشن ها و ویو های مربوطه توسط Crud در gii

ساخت فیلترهای مورد نیاز برای اکشن ها

ساخت theme های مورد نیاز در صورت لزوم

ساخت پیامهای ترجمه شده در صورتی که سایت نیاز به قابلیت چند زبانی داشته باشد.

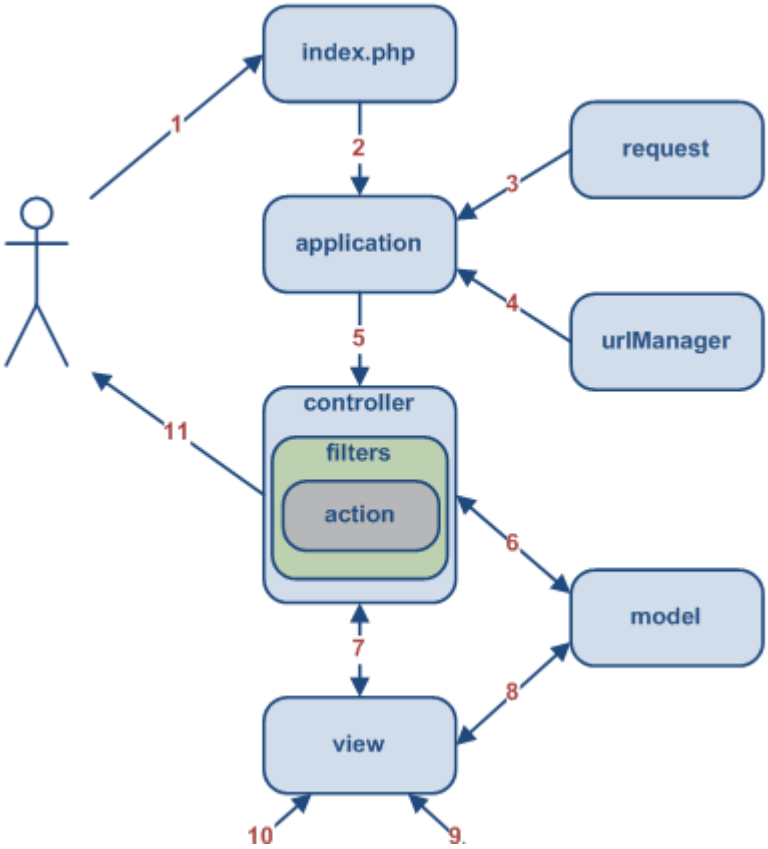
پیاده سازی تکنیک های caching در صورت نیاز.

Tune up نهایی و انتشار سایت

نکته : برای هر کدام از مراحل ده گانه بالا می توان test را فعال و پیاده سازی نمود.

مراحل اجرای یک برنامه :

جریان کاری در Yii به شکل زیر می باشد :



کاربر درخواستی را به شکل زیر وارد می کند :

http://www.example.com/index.php?r=post/show&id=1

سرویس دهنده درخواست را به واسطه اسکریپت bootstrap که همان فایل index.php است دریافت می کند.

اسکریپت bootstrap یک نمونه از Application را با اعمال تنظیمات مشخص شده در فایل config ایجاد کرده و اجرا می کند. این کار در فایل index.php توسط فرمان زیر انجام می شود :

```
Yii::createWebApplication($config)->run();
```

application درخواستهای کاربر را توسط یک component به نام request component دریافت می نماید.

سپس application توسط urlmanager component آدرس داده شده را تحلیل کرده و controller و action مورد درخواست را مشخص می نماید.

در مثال بالا کنترلر post نام دارد که کلاس postController موجود در فایل postController.php اشاره دارد. و همچنین اکشن مورد نظر show نام دارد که به متد actionShow در کلاس کنترلر اشاره دارد. و پارامتر ارسالی نیز id با مقدار ۱ است.

سپس یک نمونه از کنترلر postController توسط application ساخته شده و اجرا می گردد. در این کنترلر فیلتر های مورد نظر اعمال شده و در صورتی که فیلترها اجازه دهند متد actionShow اجرا می شود. فیلترها می توانند access control, benchmarking و یا غیره باشند.

یک اکشن مدلی با نام Post را فراخوانی می کند که یک رکورد از جدول بانک اطلاعاتی است که id آن برابر ۱ است.

اکشن یک ویو به نام show را render (فراخوانی و اجرا) می کند و همچنین از مدل Post کمک می گیرد.

ویو show اطلاعات مورد نظر را از مدل Post خوانده و نمایش می دهد.

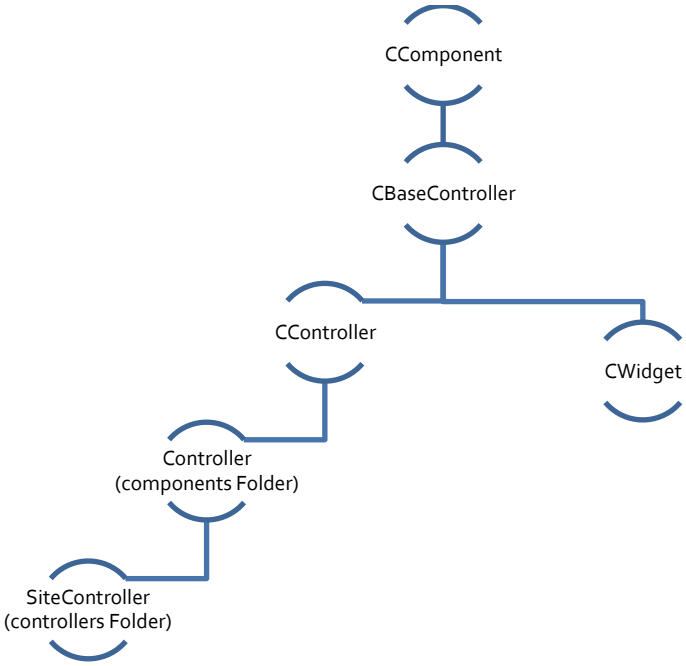
ویو تعدادی از widget ها را برای نمایش اطلاعات به کار می گیرد.

محتویات ویو داخل یک layout قرار می گیرند تا یک صفحه کامل وب تشکیل گردد.

اکشن عملیات خود را به پایان می رساند و صفحه ساخته شده را برای کاربر نمایش می دهد.

کنترلر

نمودار ارث بری های کنترلرها :



معرفی کنترلر :

فایل کنترلر ها در شاخه protected/controllers قرار می گیرند. در ابتدای شروع برنامه یک نمونه از کنترلر توسط Bootstrap یعنی فایل yiiBase.php ساخته و سپس اجرا می شود. هر کنترلر دارای یک نام منحصر به فرد یا ControllerID می باشد. کنترلر شامل تعدادی از اکشن ها می باشد که بر اساس درخواستهای کاربر کار خاصی را انجام می دهند. کنترلر ارطریق اکشن ها، جریان داده بین مدل ها و ویوها را مدیریت می کند. Action در واقع متد موجود در کنترلر است که حتما این متد با کلمه action شروع میشود.کاربر در فراخوانی هایش کلمه action را ذکر نمی کند مثل :

www.hostname.com/index.php?r=site/view

در این مثال کنترلر site و اکشنview فراخوانی میشوند ولی نام متد در اصل actionView است که کلمه action آن ذکر نمی گردد و این باعث افزایش امنیت نیز می شود. زیرا که کاربر مستقیما نام فایل را متوجه نمی شود. نحوه تعریف یک اکشن جدید :

```
class UpdateAction extends CAction
{
    public function run()
    {
        // place the action logic here
    }
}
```

اکشن می تواند پارامترهایی را هم داشته باشد که این پارامترها خود کار می باشند و مقدار آنها توسط $_GET از کاربر دریافت می شوند.

پارامترهای خودکار اکشن

یک متد اکشن می تواند پارامتهایی را برای آن تعریف نمود که مقادیر آنها را از طریق $_GET از کاربر دریافت نماید.در حالتی که نخواهیم از پارامترها خودکار استفاده نماییم اگر یک اکشن با نام create ایجاد نماییم که در کنترلر PostController تعریف شده باشد و دارای دو پارامتر category و languageباشد که مقادیر آنها را از ورودی بگیرد کد نویسی طولانی زیر را خواهیم داشت :

```
class PostController extends CController
{
    public function actionCreate()
    {
        if(isset($_GET['category']))
            $category=(int)$_GET['category'];
        else
            throw new CHttpException(404,'invalid request');

        if(isset($_GET['language']))
            $language=$_GET['language'];
        else
            $language='en';

        // ... fun code starts here ...
```

```
    }
}
```

حال با استفاده از پارامترهای خودکار مجددا همین کنترلر را تعریف می نماییم. که دارای کد نویسی ساده به شکل زیر می باشد.

```
class PostController extends CController
{
public function actionCreate($category, $language='en')
    {
        $category=(int)$category;

        // ... fun code starts here ...
    }
}
```

تعداد پارامترها و نام آنها باید دقیقا با آن چیزی که کاربر وارد می کند تطابق داشته باشد برای کد بالا درخواستی مثل کد زیر قابل قبول است:

www.hostname.com/index.php?r=Post/categoory=1&language=fa

اگر کاربر درخواستی بدهد که از این الگو یا الگوهای مورد قبول این الگو پیروی نکند یک خطای ۴۰۰ رخ می دهد. پارامتر language بالا دارای یک مقدار پیش فرض است $language='en' و در صورتی که کاربر این پارامتر را ارسال نکند خطایی رخ نمی دهد و مقدار پیش فرض برای آن در نظر گرفته می شود. اما چون category مقدار پیش فرض ندارد درخواستی که پارامتر category در آن نباشد باعث بروز خطا خواهد شد.

```
class PostController extends CController
{
public function actionCreate(array $categories)
    {
        // Yii will make sure $categories be an array
    }
}
```

در این حالت کلمه کلیدی arrayباید نوشته شود. کاربر می تواند با $_GET['categories'] کار کند که یک رشته عادی است ولی میتوان آن را به یک آرایه تبدیل کرد و از عناصر آن آرایه استفاده نمود.

وقتی که کاربر یک اکشن مثل XYZ را درخواست می کند، کنترلر یکی از کارهای زیر را انجام می دهد :

Method-based action : یعنی اینکه متد actionXYZ را در صورت وجود فراخوانی می کند.

Class-based action : یعنی اینکه یک نمونه از کلاس XYZ را در صورت موجود بودن کلاس در action class map ایجاد می کند. و سپس اکشن را فراخوانی می کند.

Call missingAction : یعنی اینکه به طور پیش فرض یک خطای HTTP ۴۰۴ را تولید می کند.

اگر کاربر هیچ اکشن مشخصی را درخواست ندهد اکشن defaultAction اجرا خواهد شد.defaultAction را به عنوان یک متغیر می توان در ابتدای کلاس کنترلر تعیین نمود.

کنترلر ممکن است که بخواهد قبل و یا بعد از اجرای یک اکشن فیلترهایی را اجرا کند.

پارامترهای عمومی قابل تعریف در کنترلر :

تعریف قالب :public $layout

می توان قالب پیش فرض برای استفاده در ویوهای مورد استفاده در این کنترلر را تعیین نمود که به طور پیش فرض main می باشد. اگر مقدار آن false تعریف شود هیچ قالبی استفاده نمی شود. مثال :

```
public $layout='//layouts/mylayout;
```

اگر کنترلر در یک ماژول قرار داشته باشد می توان برای تعیین قالب از دستور CWebModule::layout module layout استفاده می شود.

تعریف اکشن پیش فرض : public $defaultAction='index'

اکشن پیش فرض برای اجرا مشخص می شود. در اینصورت اگر در، درخواست کاربر اکشن مشخص نشود این اکشن پیش فرض اجرا می شود. البته از ابتدا مقدار آن 'index' استو باعث اجرای اکشن actionIndex می شود.مثال :

```
public $defaultAction='myAction';
```

متدهای قابل استفاده در کنترلر :

```
public function init()
```

کنترلر را مقدار دهی اولیه می کند. این متد توسط application و قبل از شروع اجرای کنترلر اجرا می شود. ممکن است شما این متد را override کنید تا بتوانید تنظیمات پیش از اجرای کنترلر را اعمال نمایید.

public function filters()

این متد تنظیمات فیلتر را بر می گرداند. این متد آرایه ای از مقادیر فیلتر را بر می گرداند که هر کدام از عناصر آرایه مربوط به هر یک از فیلتر ها هستند. کنترلر ممکن است که بخواهد قبل و یا بعد از اجرای یک اکشن فیلترهایی را اجرا کند.

فیلتر ها می توانند قبل و یا بعد از درخواست و یا پاسخ به کاربر اجرا شوند و ممکن است جلوی اجرای یک اکشن را بگیرند. در صورت لزوم فیلتر ها می توانند در یک ترتیب خاص اجرا شوند. در اینصورت اگر در هر مرجله ای از اجرا هر یک از فیلتر ها مقدار true را بر گرداند بقیه فیلترها و اکشن ها اجرا نخواهند شد.

فیلترها می توانند به صورت یک شی ساخته شده از روی یک کلاس مجزا باشند و یا به صورت متدهای تعریف شده در کلاس کنترلر. فیلتر ها با override کردن متد filters به شکل زیر اجرا می شوند. مثال :

```
<pre>
  array(
      'accessControl - login',
      'ajaxOnly + search',
      array(
          'COutputCache + list',
          'duration'=>300,
      ),
  )
</pre>
```

در مثال بالا ۳ فیلتر تعریف شده است که عبارتند از accessControl, ajaxOnly, COutputCache دو فیلتر اول یعنی accessControl, ajaxOnly, بر اساس متد ساخته شده اند. که متد آنها در کلاس CController تعریف شده است که به فیلتر کردن متد ها در کلاس کنترلر اشاره می کنند.

در حالی که سومین فیلتر از نوع شی است که کلاس آن system.web.widgets.COutputCache می باشد و پارامتر duration آن برابر مقدار ۳۰۰ قرار گرفته است. مثال دیگری به شکل زیر است :

```
class PostController extends CController
{
    ......
    public function filters()
    {
        return array(
            'postOnly + edit, create',
            array(
```

```
                'application.filters.PerformanceFilter - edit,
create',
                'unit'=>'second',
            ),
        );
    }
}
```

کد بالا دو فیلتر را تعریف می نماید.

متد فیلتر postOnly و کلاس PerformanceFilter– مسیر قرار گیری این کلاس در مسیر application.filters.PerformanceFilter است که کلاس فیلتر مورد نظر در فایل PerformanceFilter.php قرار دارد. که دارای یک property با نام unit است که مقدار second برای آن ارسال می شود.

عملگر + : مشخص می کند که این فیلتر برای کدامیک از اکشن ها اجرا شود. در مثال بالا اکشن postOnly باید برای اکشن های edit و create اجرا شود.

عملگر - : مشخص می کند که این فیلتر باید برای کدامیک از اکشن ها اجرا نشود. در مثال بالا فیلتر PerformanceFilter برای همه اکشن ها به جز edit و create باید اجرا شود.

نکته : اگر – یا + در فیلتر مشخص نشود آن فیلتر برای همه اکشن ها اجرا می شود.

برای فیلترهایی که بر اساس متد ساخته شده اند یک متد با نام filterXYZ به صورت filterXYZ($filterChain) تعریف می شود که نام این فیلتر XYZ است.

نکته : داخل متد فیلتر باید کد filterChain->run$() حتما نوشته شود تا زنجیره اجرایی ادامه پیدا کند وگرنه زنجیره اجرادر همین نقطه پایان می پذیرد.

نکته : اگر مقدار برگشتی یک متد فیلتر false باشد اکشن های مربوطه اجرا نخواهند شد.

فیلترها می توانند به گونه ای تعریف شوند که تنها زمانی اجرا شوند که یک اکشن خاص اجرا می شود. برای فیلترهای بر اساس متد این کار به وسیله عملگرهای + و – در تعریف فیلتر انجام می شود. عملگر + مشخص می کند که فیلتر تنها زمانی اجرا شود که یک فیلتر به خصوص فراخوانی شود در حالی که عملگر – به این معنی است که فیلتر تنها زمانی اجرا می شود که این اکشن در میان اکشن های درخواست قرار ندارد.برای اکشن های بر اساس شی عملگر + و – نام کلاس را دنبال می کنند.

فیلتر ها به دو دسته تقسیم می شوند :

۱- inline filter : که فیلترهای بر اسا متد هستند. و الگوی تعریف آنها به شکل زیر است:

```
FilterName[ +|- Action1, Action2, ...]
```

که عملگرهای + و – مشخص می کنند که کدام اکشن باید/نباید فیلتر گذاری شود.

۲- class-based filter : که فیلتر مربوطه به وسیله یک شی ساخته شده از روی کلاس تعریف می شود. این کلاس از کلاس Cfilter ارث بری می کند. مثال :

```
class PerformanceFilter extends CFilter
{
    protected function preFilter($filterChain)
    {
// logic being applied before the action is executed
return true; // false if the action should not be executed
    }

    protected function postFilter($filterChain)
    {
// logic being applied after the action is executed
    }
}
```

نحوه تعریف اینگونه فیلتر به شکل زیر است :

```
<pre>
 array(
      'FilterClass[ +|- Action1, Action2, ...]',
      'name1'=>'value1',
      'name2'=>'value2',
      ...
  )
</pre>
```

'name1'=>'value1' در این ساختار مقادیر property های فیلتر را مشخص می کنند.

نکته : برای ارث بری فیلترها از یکدیگر یک کلاس فرزند باید با کلاس والد خود ادغام شود که این کار می تواند توسط توابعی مثل array_merge انجام گیرد.

```
public function actions()
```

این متد لیستی از اکشن های موجود در کلاس خارجی را بر می گرداند. این متد شامل آرایه ای است که عناصر آن اکشن ها و کلاسهای آنها را مشخص می کنند. مثال :

```
'edit'=>'application.controllers.article.EditArticle'
```

در این مثال اکشنی با نام edit در کلاسی به مسیر 'application.controllers.article.EditArticle' فراخوانی می شود و از این به بعد در این کلاس قابل فراخوانی و استفاده است.همچنین می توان پارامترهایی را نیز به اکشن فراخوانی شده ارسال نمو. مثال :

```
    <pre>
      return array(
          'action1'=>'path.to.Action1Class',
          'action2'=>array(
              'class'=>'path.to.Action2Class',
              'property1'=>'value1',
              'property2'=>'value2',
          ),
      );
    </pre>
```

در مثال بالا action2 تعریف می شود که مسیر آن 'path.to.Action2Class' است و دو property نیز همراه با فراخوانی ارسال می شود.

یکی دیگر از کاربردهای این متد این است که اگر یک کلاس کنترلر از کلاس کنترلر ما مشتق شود توسط این متد می تواند اکشن های کلاس والد خود را فراخوتنی نماید.

نکته : برای ارث بری اکشن ها از یکدیگر یک کلاس فرزند باید با کلاس والد خود ادغام شود که این کار می تواند توسط توابعی مثل array_merge انجام گیرد.

شما همچنین می توانید اکشن ها را از یک action providerمثل CWidget::actions فراخوانی نمایید. مثال :

```
<pre>
return array(
...other actions...
// import actions declared in ProviderClass::actions()
// the action IDs will be prefixed with 'pro.'
'pro.'=>'path.to.ProviderClass',
// similar as above except that the imported actions are
values
'pro2.'=>array(
'class'=>'path.to.ProviderClass',
```

```
'action1'=>array(
'property1'=>'value1',
),
'action2'=>array(
'property2'=>'value2',
),
),
 )
</pre>
```

در مثال بالا ما action providers را از سایر تعاریف اکشن جدا کرده ایم برای action providers ها باید برای تعریف از یک نقطه استفاده نماییم بنابر این مثلا pro2.action1 به عنوان action1 شناخته می شود که در ProviderClass تعریف شده است.

public function behaviors()

لیستی از رفتارها را که کنترلر باید از خود نشان دهد بر می گرداند. این متد شامل آرایه ای است که در آن نام رفتار و مقدار آن مشخص می شود مثل name=>behavior. هر رفتار می تواند یک رشته معرف نوع رفتار کلاس باشد یا یک آرایه با ساختار زیر داشته باشد :

```
    <pre>
     'behaviorName'=>array(
         'class'=>'path.to.BehaviorClass',
         'property1'=>'value1',
         'property2'=>'value2',
     )
    </pre>
```

در مثال بالا 'behaviorName' یک رفتار است که در کلاسی با آدرس 'path.to.BehaviorClass' معرفی می شود و دو پارامتر با مقادیر مشخص شده را دریافت می کند.

توجه کنید که کلاس behavior بایستی از واسط IBehavior استفاده کند و یا از کلاس CBehavior ارث بری نماید. رفتارهای تعریف شده در کنترلر در زمان ساخت نمونه کنترلر توسط application به کلاس کنترلر ملحق می شوند.

توضیحات تکمیلی در مورد رفتارها را می توان در راهنمای شی گرایی در PHP و یا در کلاس CComponent مشاهده نمود.

public function accessRules()

این متد فیلترهای دسترسی کاربر به اکشن های کنترلر را مشخص می کند. به عنوان مثال مشخص می کند که کدام کاربر مجاز به اجرای کدام اکشن می باشد یا نمی باشد.

برای این که هر کاربری نتواند با وارد کردن هر آدرسی وارد آن صفحه شود از این قسمت استفاده می شود. مثلا کاربری که وارد سایت نشده یعنی login نکرده نمی تواند کنترلر Post را اجرا کند.

برای هر یک یا چند اکشن می توان یک آرایه جدا تشکیل داد و نحوه دسترسی کاربرانی که می توانند از آن استفاده کنند را مشخص کرد.

از کاراکتر * برای معرفی کردن همه کاربران و از کاراکتر @ برای معرفی کاربرانی که وارد سایت شده اند استفاده می کنیم همچنین از علامی ؟ برای کاربران ناشناس استفاده می شود. هر مدخل آرایه برای این متد به شکل زیر تعریف می شود :

```
array('deny or allow',
'actions'=>array('action1','action2',…),
     'users'=>array('@ or *')
),
```

ابزار gii در استفاده از ابزار Crud generator باعث می شود که مدخلی با شکل زیر ایجاد می کند که برای اجرای اکشن های کاربر باید آن را اصلاح نمود وگرنه اکشن های کاربر اجرا نخواهند شد.

```
array('deny',  // deny all users
     'users'=>array('*'),
),
```

در این اکشن همچنین می توان پارامترهای دیگری مثل Ip, Roles, Controllers, Verbs, Expression و غیره را نیز استفاده کرد به عنوان مثال از Ips می توان استفاده کرد تا کاربرانی با Ip مشخص اجازه استفاده از سایت را داشته باشند یا نداشته باشند. برای مدیریت گروهها نیز از Roles استفاده می شود که برای توضیحات بیشتر به ویکی پدیا بخش RBAC مراجعه نمایید.

روش دیگر این کار تعریف یک شرط است مثل:

```
array('allow',
          'actions'=>array('admin'),
          'expression'=> 'Yii::app()->user->group == 3',
     ),
```

public function run($actionID)

این متد نام یک اکشن را گرفته و آن را اجرا می کند. فیلترهای اکشن مورد نظر نیز اجرا خواهند شد. اگر اکشن مورد نظر پیدا نشد و یا نام اکشن صحیح وارد نشده باشد یک CHttpException رخ می دهد.

public function runActionWithFilters($action,$filters)

این متد یک فیلتر را با یک فیلتر مشخص شده اجرا می کند. یک زنجیره فیلتر در این مرحله ایجاد شده و سپس اکشن مورد نظر اجرا می شود.

public function runAction($action)

این متد یک اکشن را پس از اعمال تمامی فیلترها مرتبط با این اکشن اجرا می کند.

public function createAction($actionID)

این متد یک نمونه از اکشن معرفی شده را تولید می کند. این اکشن می تواند یک اکشن inline یعنی تعریف شده در داخل همین کلاس کنترلر باشد و یا یک اکشن object یعنی تعریف شده در کلاس خارجی و سپس نمونه سازی شده باشد.

public function missingAction($actionID)

این متد بررسی می کند که آیا اکشن معرفی شده موجود می باشد یا خیر و در صورتی که موجود نباشد یک خطا صادر می کند.

public function getRoute()

این متد رشته تقاضای جاری به شکل module ID, controller ID and action ID را بر می گرداند.

public function getAction()

این متد نام اکشن فعال را بر می گرداند. در صورتی که هیچ اکشن فعالی وجود نداشته باشد مقدار null را بر می گرداند.

public function getId()

این متد نام کنترلر جاری را بر می گرداند.

public function getUniqueId()

این متد نیز نام کنترلر جاری را برمی گرداند منتهی اگر این کنترلر داخل ماژولی باشد نام آن ماژول را نیز بر می گرداند. به صورت ModuleID.ControllerID

public function getModule()

این متد نام ماژولی که کنترلر در آن قرار گرفته را بر می گرداند. در صورتی که هیچ ماژولی موجود نباشد مقدار null را بر می گرداند.

public function getViewPath()

این متد مسیر قرار گیری ویوهای مورد استفاده در این کنترلر را مشخص می کند. مسیر پیش فرض protected/views/ControllerID می باشد که می توان آن را تغییر داد. کلاسهای فرزند این کلاس کنترلر می توانند این متد را override کنند و مسیر دلخواه خود را برای قرار دادن فایلهای ویو قرار دهند. اگر کنترلر جاری در یک ماژول قرار گرفته باشد مسیری که این متد بر می گرداند شامل نام آن ماژول نیز خواهد بود.

public function getClips()

این متد لیستی از کلیپ های مپروژه را بر می گرداند. یک کلیپ قسمتس از نتایج به وجود آمده توسط پروژه است که می تواند تحت قالب یک نام مشخص قرار گیرد و در جاهای مختلف برنامه فراخوانی شود.

public function forward($route,$exit=true)

این متد درخواست کاربر را توسط اکشن دیگری در کنترلر اجرا می کند. این کار شبیه عملکرد دستور redirect است با این تفاوت که با اجرای این متد URL موجود در مرورگر کاربر تغییری نمی کند. در بسیاری از موارد بهتر است که از دستور URL به کای استفاده از این متد استفاده کرد. پارامتر route مسیر اکشنی که باید اجرا شود را مشخص می کند. این می تواند تنها شامل نام آن اکشن باشد که در همین کلاس

کنترلر قرار گرفته و یا یک مسیر کامل شامل سایر اجزا مانند ماژول ها باشد. پارامتر exit یک مقدار true/false را می گیرد و اگر true باشد یعنی برنامه پس از اجرای این متد پایان پذیرد.

```
public function processOutput($output)
```

این متد خروجی های تولید شده توسط render را پردازش می کند. این متد در پایان اجرای متد render و متد renderText اجرا می شود.اگر scripts و یا dynamic contents برای این صفحه مشخص شده آنها نیز به صفحه اضافه می شوند و وظیفه اصلی این متئ انجام همین کار است.

```
public function render($view,$data=null,$return=false)
```

این متد یک ویو را با قالب آن اجرا می کند. این متد ابتدا renderPartial را فراخوانی می کند تا ویو را به تنهایی اجرا کرده باشد و سپس قالب را اجرا کرده و نتایج را در هم قرار می دهد تا یک صفحه کامل شکل گیرد. در قالب مورد نظر هر جا کد content$فراخوانی شود در همانجا محتوای ویو نشان داده می شود. در انتها این متد processOutput را فراخوانی می کند تا اگر scripts و یا dynamic contents برای این صفحه مشخص شده آنها نیز به صفحه اضافه شوند. به طور پیش فرض قالب مورد استفاده protected/views/layouts/main.php می باشد که می توان آن را با تغییر مقدار متغیر layout در کلاس کنترلر تغییر داد. پارامتر result نتیجه اجرا را بر می گرداند.

نحوه اسال پارامترها از کنترلر به ویو به شکل ;('go',array('id'=>'3'))$this->render می باشد. در این کد پارامتر id با مقدار ۳ برای ویو go ارسال می شود در ویو می توان به شکل ;$idپارامتر ارسالی را دریافت و استفاده نمو. نکته ای که باید دقت کرد این است که پارامتر ها فقط به وسیله آرایه بایستی که ارسال شوند.

انئاع کنترل بر روی صفحه :

```
$this->render('demo',array(loc => $id));
```

اجرای ویوی demo و ارسال پارامتر loc با مقدار $id برای آن. این کار باعث ساخت صفحه و نمایش صفحه به همراه قالب صفحه خواهد شد.

```
$this->renderParitial('demo',array(loc => $id));
```

اجرای ویوی demo و ارسال پارامتر loc با مقدار $id برای آن. این کار باعث ساخت صفحه و نمایش صفحه بدون قالب صفحه خواهد شد.

```
$this->redirect(Yii::app()->homeUrl);
```

این دستور باعث انتقال به صفحه ای با Urlمشخص خواهد شد.

```
$this->refresh();
```

این دستور باعث اجرای مجدد همین صفحه خواهد شد.

protected function beforeRender($view)

این متد قبل از اجرای متد render اجرا می شود. شما می توانید این متد را override کنید تا قبل از اجرای یک ویو محاسبات یا پردازشهای خاصی انجام شود. پارامتر $view نام ویویی است که اجرا می شود.

protected function afterRender($view, &$output)

این متد بعد از اجرای متد render اجرا می شود. این متد حتما قبل از متد processOutput اجرا می شود. . شما می توانید این متد را override کنید تا بعد از اجرای یک ویو محاسبات یا پردازشهای خاصی انجام شود.پارامتر $view نام ویویی است که اجرا می شود. پارامتر $output نتیجه اجرای ویو است.

public function renderText($text,$return=false)

این متد یک نوشته text را نشان می دهد. این نوشته در قالب جاری پروژ] قرار می گیرد و نشان داده می شود یعنی قالب سایت هم در آن لحاظ می شود. پارامتر text متن مورد نظر است. پارامتر result نتیجه اجرا را بر می گرداند.

```
public function
renderPartial($view,$data=null,$return=false,$processOutput=false)
```

این متد یک ویو را اجرا می کند. تفاوت این متد با متد render در این است که متد renderPartial قالبی را برای صفحه اعمال نمی کند ولی متد render قالب را نیز اعمال می کند. همچنین از این متد برای نمایش پاسخ با استفاده از AJAX نیز استفاده می شود. پارامتر $view نام ویویی است که باید اجرا شود. پارامتر $data مقادیی که برای اجرا به صورت پارامتر ارسال شده اند را مشخص می کند. پارامتر $return مشخص می کند که نتایج حاصله باید برای کاربر نشان داده شود یا خیر. پارامتر processOutput $ مشخص می کند که در اجرا باید نتایج متد processOutput هم لحاظ شود یا خیر.

```
public function createUrl($route,$params=array(),$ampersand='&')
```

این متد یک relative URL را برای اکشن موجود در این کنترلر تعریف می کند. پارامتر $route آدرس مورد نظر را مشخص می کند. این آدرس باید دارای ساختار ControllerID/ActionID باشد. اگر ControllerID مشخص نباشد کنترلر جاری به عنوان کنترلر قرار می گیرد. اگر این آدرس خالی باشد آدرس اکشن جاری

در نظر گرفته می شود. اگر نخواهیم ماژولی برای این آدرس در نظر گرفته شود باید آدرس با یک علامت / شروع شود. پارامتر $params پارامترهایی که توسط GTT قرار است دریافت شود را مشخص می کند که دارای ساختار name=>value می باشد که name و value باید هر دو URL-encoded باشند. اگر name کاراکتر # تعریف شده باشد value مربوطه به عنوان یک anchor در صفحه مشخص می شود. پارامتر $ampersand پارامترهای name-value را در URL از یکدیگر تفکیک می کند.

```
public function
createAbsoluteUrl($route,$params=array(),$schema='',$ampersand='&')
```

این متد یک absolute URL را برای اکشن تعریف شده در کنترلر مشخص می کند. پارامترهای مشترک آن همانند متد قبلی می باشند. پارامتر $schema مشخص کننده نوه درخواست مانند http یا https و غیره است که اگر خالی باشد مقدار موجود در درخواست جاری درنظر گرفته می شود.

```
public function getPageTitle()
```

این متد عنوان صفحه را بر مر گرداند که به طور پیش فرض شامل نام کنترلر و نام اکشن اجرا شده است.

```
public function setPageTitle($value)
```

این متد عنوان صفحه را تعیین می کند.

```
public function redirect($url,$terminate=true,$statusCode=302)
```

این متد باعث می شود که مرور گر به آدرس URL تعیین شده برود. که مسیر آن به شکل controller/action است. پارامتر $url آدرس مورد نظر را مشخص می کند. اگر آدرس ترکیبی باشد در قالب یک ارایه تعریف می شود که عنصر اول آن آدرس URL را مشخص می کند و بقیه عناصر ارایه پارامترهایی را که به صورت name-value در GET مورد استفاده قرار دارند مشخص می کند. پارامتر $terminate مشخص می کند که پس از اجرای URL مورد نظر برنامه پایان پذیرد یا ادامه پیدا کند. پارامتر $statusCode مشخص کننده وضعیت HTTP است که به طور پیش فرض برابر ۳۰۲ قرار گرفته است. برای توضیحات بیشتر می توان به www.w3.org/Protocols/rfc2616/rfc2616-sec10.html مراجعه کرد.

```
public function refresh($terminate=true,$anchor='')
```

این متد باعث refresh شدن صفحه جاری می شود. پارامتر $terminate شبیه پارامتر متد قبلی است و پارامتر $anchor مشخص می کند که صفحه پس از بارگزاری مجدد به کدام anchor پرش نماید. مقدار پیش فرض آن خالی است. نکته : اگر قصد استفاده از anchor را دارید حتما باید نام آن با علامت # شروع شود.

public function recordCachingAction($context,$method,$params)

یک فراخوانی به متد یا method call را وقتی که یک output cache فعال است ثبت می کند. وقتی که محتوا در خروجی نشان داده شود متد ذخیره شده فراخوانی می شود.

public function getCachingStack($createIfNull=true)

اگر مقدار پارامتر createIfNull برابر true برای کش کردن یک stack را در صورتی که قبلا ایجاد نشده باشد ایجاد می کند.

public function isCachingStackEmpty()

مشخص می کند که آیا caching stack خالی است یا خیر.

protected function beforeAction($action)

این متد دقیقا قبل از اینکه یک اکشن فراخوانی و اجرا شود اجرا می شود البته بعد از اجرای تمامی فیلترهای وابسته. شما می توانید این متد را برای انجام کارهای لازم قبل از اجرای اکشن override کنید. پارامتر $action به اکشن مورد نظر اشاره دارد.

protected function afterAction($action)

این متد درست بعد از اجرای اکشن اجرا می شود.

public function filterPostOnly($filterChain)

متد فیلتر برای فیلتر postOnly

public function filterAjaxOnly($filterChain)

متد فیلتر برای فیلتر ajaxOnly

public function filterAccessControl($filterChain)

متد فیلتر برای فیلتر accessControl

public function paginate($itemCount,$pageSize=null,$pageVar=null)

این متد اطلاعات مربوط به pagination را تولید می کند. از طریق این متد می توان اندازه یک صفحه را مشخص کرد. شما می توانید مستقیما از طریق استفاده از new Cpagination یک شی pagination جدید را تولید نمایید. پارامتر $itemCount تعداد عناصر در هر صفحه را مشخص می کند. پارامتر $pageSize اندازه

صفحه را مشخص می کند که برای تعیین مقادیر آن و سایر پارامترها می توانید توضیحات مربوط به Cpagination را ملاحظه نمایید.

```
public function getPageState($name,$defaultValue=null)
```

این متد و سایر متهای مرتبط حالت صفحه را مشخص می کنند. یک حالت صفحه یک مقدار است که از طریق درخواست نوع POST در همین صفحه ارسال می شود.

```
public function clearPageStates()
```

این متد کلیه حالات صفحه را که لحاظ شده اند حذف می کند.

متد تعریف شده loadModel

دسترسی به مدل ها به دو شکل امکان پذیر است:

۱- نیاز به ساخت رکوردی جدید داریم مثلا برای عملیات ایجاد یا حذف که به شکل زیر عمک می کنیم :

```
$model=new MyModel();
```

۲- می خواهیم با رکورد جاری کار کنیم مثلا برای عملیات update که در اینصورت باید از متد کمکی loadModel استفاده نماییم. مثال :

```
$model=$this->loadModel($id);
```

این متد در کنترلر به شکل زیر تعریف می شود :

```
     public function loadModel($id)
     {
          $model=Website::model()->findByPk((int)$id);
          if($model===null)
               throw new CHttpException(404,'The requested
page does not exist.');
          return $model;
     }
```

مدیریت درخواستهای کاربر

درخواستهای کاربر توسط دو component مدیریت می شوند :

componentCUrlManager

componentCHttpRequest

استفاده از پارامترهای ورودی در اکشن :

```
Public function actionExample()
 {
  If (isset($_GET[‘a’]) $a = $_GET[‘a’];
 Else die (“invalid parameter a”);
 }

Public function actionExample($a)       // function Override
{
 Echo $a;
}
```

Action می تواند به عنوان پارامتر یک آرایه را هم قبول کند.

```
Public function actionExample(array $a)
```

نحوه استفاده از پارامترهای ورودی کاربر در کنترلر :(URL parameter -> Controller)

```
     public function actionGo()
     {
     if (isset($_GET['id']))
      {
       $id = $_GET['id'];
      }
     else
     {
      die ("Invalid id parameter");
     }
```

اگر در خواستی به شکل زیر داده شود :

```
http://hostname.com/index.php?r=site/Go/id/2
```

آنگاه مقدار متغیر $id برابر ۲ قرار می گیرد و اگر اکشن Go بدون پارامتر فراخوانی شود پیغام خطایی ظاهر می شود.

نکاتی در مورد کنترلرها :

۱- کنترلر شامل کدهایی است که در آنها اطلاعات end user مورد نیاز است مثل کدهای شامل $_GET و$_POST$ این نوع کدها فقط در کنترلر قابل تعریف می باشند.

۲- یک کنترلر می تواند یک مدل جدید را بسازد و استفاده نماید.

۳- کنترلر ها نباید شامل کدهای SQL باشد.

۴- کنترلرها نباید شامل کدهای HTML باشند.

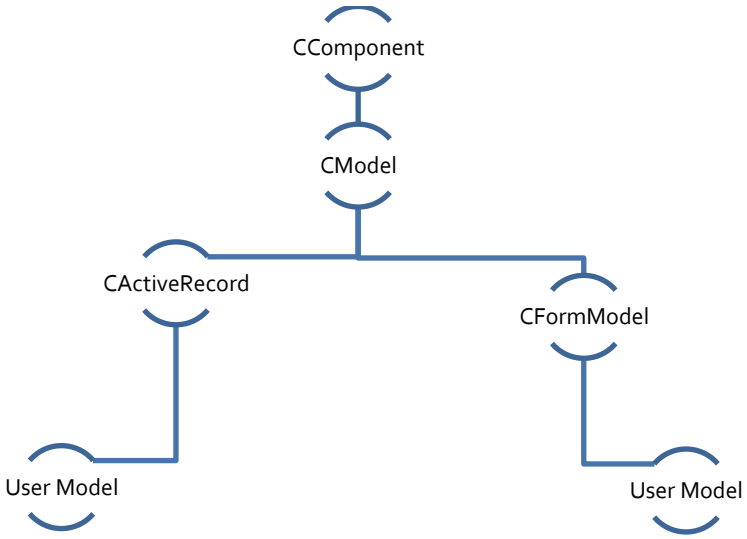
۵- در کنترلرها باید فقط از کدهای PHP استفاده کرد.

۶- کلاس کنترلر باید تا حد امکان کم حجم باشد و شامل کدهای کوتاه ولی موثر باشد. کنترلر ها معمولا شامل کدهای بسیار کمی هستند.

۷- عملیات دریافت داده های کاربر و دریافت تقاضاها و سپس پردازش و نهایتا ارسال اطلاعات و ارسال آنها به بخش های مختلف وظیفه کنترلر است.

مدل ها

نمودار ارث بری های مدل :



یک مدل معرف یک رکورد از بانک اطلاعاتی و یا اطلاعات یک فرم وارد شده توسط کاربر است.کلیه کلاسهای مدل ازروی کلاس والد CModel و یا زیر کلاسهای آن ساخته می شوند.CModel کلاس پایه برای استفاده از ویژگی های data model objects می باشد. هر کدام از فیلدهای بانک اطلاعاتی معرفی شده در مدل را یک attribute می گویند. هر attribute می تواند شامل یک مقدار value و یک برچسب label باشد.یک مدل می تواند بین چند قسمت از برنامه به طور مشترک استفاده شود.

محل قرار گیری فایل مدل ها : protected \ models

آدرس مستعار برای دسترسی به مدل ها : application.modles

انواع مدل

مدل ها به دو شکل موجود می باشند. یکی FormModel ها و دیگری ActiveRecord که هر دو از کلاس والد CModel ارث بری می کنند.

۱) FormModel : یک مدل داده که اطلاعات ورودی کاربر در قالب فرم های HTML را دریافت می کند. و این اطلاعات تنها در حافظه سیستم قرار می گیرند و نه در داخل بانک اطلاعاتی. برای دسترسی به فیلدهای فرم باید برای هر کدام یک متغییر با همان نام تعریف نماییم. این نوع مدل از کلاس CFormModel ارث بری می کند. نمونه بارز این نوع مدل فرم login است که تنها اطلاعات را از ورودی توسط فرم دریافت می کند و آن را تجزیه و تحلیل می نماید ولی چیزی را در بانک اطلاعاتی ذخیره نمی کند. مثال :

```
class LoginForm extends CFormModel
```

```
{
	public $username;
	public $password;
	public $rememberMe;
```

دسترسی به ورودی های فرم آن از طریق همین متغیرهایی است که در ابتدا تعریف کرده ایم.اگر شما می خواهید با داده های وارد شده کاربر کار کنید مثل Login باید از FormModel استفاده نمایید.

۲) ActiveRecord : یک مدل داده که اطلاعات را مستقیما داخل بانک اطلاعاتی ذخیری می نماید و معرف فیلد ها و رابطه های جدول بانک اطلاعاتی می باشدو نیازی به تعریف متغیر جداگانه نمی باشد. مثل فرم register که عملیات دریافت و ذخیره مستقیم اطلاعات داخل بانک اطلاعاتی را انجام می دهد. (Active Record (ARیک طراحی شی گرا برای دسترسی به اجزای بانک اطلاعاتی در قالب شی است. الگوی active record ارایه کننده یک مدل بر اساس Object-Relational Mapping (ORM) می باشد.کلیه کلاسهای این مدل از کلاس CActiveRecord ارث بری می کنند. کلاسCActiveRecordکلاس پایه برای کلیه کلاسهایی است که بانک های اطلاعاتی رابطه ای را پیاده سازی می کنند. یک فرم AR دارای ساختار اصلی زیر است

```
	public static function model($className=__CLASS__)
	{
		return parent::model($className);
	}

	public function tableName()
	{
		return '{{table1}}';
	}
```

این متد دارای دو متد اصلی است که یکی از آنها متد model است که یک متد استاتیک است و از طریق آن می توان به اجزای مدل دسترسی پیدا کرد و دیگری متد tableName است که نام جدول را بر می گرداند. وجود این دو متد در ActiveRecord ضروری است.اگر شما می خواهید که داده ها را در بانک اطلاعاتی ذخیره کنید مثل Registeration باید از Active Record استفاده کنید.

نکته : ابزار gii در قسمت Form generator فقط فرم از نوع ActiveRecord ایجاد می کند. و Formodel ها را باید به طور دستی ایجاد نماییم.

متد های تعریف شده در CModel و قابل استفاده در کلیه کلاسهای مدل :

```
abstract public function attributeNames();
```

این متد لیستی از نامهای attribute های مورد استفاده در مدل را بر می گرداند.

```
public function rules()
```

این متد قوانین اعتبار سنجی را برای attribute های فرم انجام می دهد.خروجی متد فوق آرایه ای از نتایج به دست آمده توسط متد اعتبار سنجی است. می توان این متد را override نمود و قوانین اعتبار سنجی را مشخص کرد.

نکته : اعتبارسنجی ها را فقط باید برای فیلدهایی نوشت که مقدار آنها توسط کاربر از ورودی دریافت می شود و مثلا اگر فیلدی مثل CurrentTime موجود باشد که مقدار آن توسط سیستم پر می شود نیازی به اعتبار سنجی آن نیست.هر قانون اعتبار سنجی در قالب یک آرایه با ساختار زیر مشخص می شود :

```
<pre>

array('attribute list', 'validator name', 'on'=>'scenario
name', ...validation parameters...)

</pre>
```

attribute list مشخص کننده نام attribute ها است که باید توسط کاما از یکدیگر جدا شوند. validator name مشخص کننده نوع عملیات اعتبار سنجی است که باید انجام شود.مثال :

```
array('username, password', 'required')
```

در مثال فوق username و password را attribute می گویند و required یک Validator می باشد.

انواع Validator

۱- می تواند نام یک کلاس اعتبار سنجی باشد که وقتی آن کلاس فراخوانی شد یک شی از روی آن کلاس ساخته شده و عملیات اعتبار سنجی توسط آن شی انجام می شود.

۲- می تواند به جای نام کلاس اعتبار سنجی نام یک alias یعنی نام مستعار برای آن کلاس اعتبار سنجی باشد مثل required

alias های معروف برای اعتبار سنجی :

```
boolean: alias of CBooleanValidator, ensuring the attribute has
a value that is either CBooleanValidator::trueValue or
CBooleanValidator::falseValue.
captcha: alias of CCaptchaValidator, ensuring the attribute is
equal to the verification code displayed in a CAPTCHA.
compare: alias of CCompareValidator, ensuring the attribute is
equal to another attribute or constant.
email: alias of CEmailValidator, ensuring the attribute is a
valid email address.
default: alias of CDefaultValueValidator, assigning a default
value to the specified attributes.
exist: alias of CExistValidator, ensuring the attribute value
can be found in the specified table column.
file: alias of CFileValidator, ensuring the attribute contains
the name of an uploaded file.
filter: alias of CFilterValidator, transforming the attribute
with a filter.
in: alias of CRangeValidator, ensuring the data is among a pre-
specified list of values.
length: alias of CStringValidator, ensuring the length of the
data is within certain range.
match: alias of CRegularExpressionValidator, ensuring the data
matches a regular expression.
numerical: alias of CNumberValidator, ensuring the data is a
valid number.
required: alias of CRequiredValidator, ensuring the attribute is
not empty.
type: alias of CTypeValidator, ensuring the attribute is of
specific data type.
unique: alias of CUniqueValidator, ensuring the data is unique
in a database table column.
url: alias of CUrlValidator, ensuring the data is a valid URL.
```

۳- می تواند یک متد باشد مثل متد authenticate

یک متد اعتبار سنجی به شکل زیر تعریف می شود :

```
<pre>

// $params refers to validation parameters given in the rule

function validatorName($attribute,$params)

</pre>
```

چند مثال از اعتبار سنجی ها :

وارد کردن مقدار در فیلد attribute ضروری است

```
array('username', 'required'),
```

فیلد username باید بین ۳ تا ۱۲ کاراکتر داشته باشد

```
array('username', 'length', 'min'=>3, 'max'=>12),
```

در سناریو register فیلدpassword1 باید با password2 مقدارشان یکسان باشد

```
array('password1','compare','compareAttribute'=>'password2', 'on'=>'register'),
```

در سناریو login باید متد attribute اجرا شود و فیلد password را اعتبار سنجی کند

```
array('password', 'authenticate', 'on'=>'login'),
```

ورودی باید ۰ یا ۱ باشد

```
array('check', 'boolean'),
```

ورودی باید ایمیل باشد

```
array('email', 'email')
```

ورودی باید آدرس URL باشد

```
array('url', 'url'),
```

ورودی باید طبق یک الگوی تعریف شده باشد

```
Array('tags','match','pattern'=>'/^[\w\s,]+$',
```

مقایسه دو ورودی مثل بررسی یکسان بودن password1 وpassword2

در این مثال دو فیلد یکی به نام password۱ و دیگری فیلد password2 است که باید یکسان باشند. البته باید در مدل حتما public $password2; نوشته شود و همچنین باید در Rules نوع آن safe در نظر گرفته شود.

```
array('password1', 'compare', compareAttribute'=>'password2'),
```

نحوه تعریف و استفاده از Capcha

Capcha یک حالت امنیتی است که یک کد تصادفی تولید می کند و از کاربر می خواهد این کد را خوانده و در فیلد مربوطه در فرم وارد کند.

برای این کار ابتدا در متد action در کنترلر کد زیر را وارد می کنیم:

```
     public function actions()
     {
          return array(
          // captcha action renders the CAPTCHA image
displayed on the contact page
               'captcha'=>array(
                    'class'=>'CCaptchaAction',
                    'backColor'=>0xFFFFFF,
               ),
          );
     }
```

در قسمت backColor می توان رنگ پس زمینه را مشخص کرد همچنین ویژگی های دیگری نیز دارد که می توان از روی دستور العمل تعیین کرد.

حال در مدل در قسمت اعتبار سنجی فیلدی که می خواهیم انجام شود به شکل زیر استفاده می کنیم:

```
array('verifyCode', 'captcha',
'allowEmpty'=>!CCaptcha::checkRequirements()),
```

و در مدل برای نمایش capcha به شکل زیر عمل می کنیم :

```
<?php if(CCaptcha::checkRequirements()): ?>
<div class="row">
     <?php echo $form->labelEx($model,'verifyCode'); ?>
     <div>
     <?php $this->widget('CCaptcha'); ?>
     <?php echo $form->textField($model,'verifyCode'); ?>
     </div>
     <div class="hint">Please enter the letters as they are
shown in the image above.
     <br/>Letters are not case-sensitive.</div>
</div>
<?php endif; ?>
```

سناریو ها یا سیاست های اعتبار سنجی :

از آنجایی که بعضی از اعتبار سنجی ها در بعضی از قسمت ها به کار می روند و در جای دیگر لزومی به اجرای آنها نمی باشد می توان مشخص کرد که کدام اعتبار سنجی در کجا به کار رود. به عنوان مثال :

```
public function rules()
{
    return array(
        array('username, password', 'required'),
        array('password_repeat', 'required',
'on'=>'register'),
        array('password', 'compare', 'on'=>'register'),
    );
}
```

کد اول اعتبار سنجی بالا همه جا اجرا می شود ولی دو کد بعدی فقط در زمان register اجرا می شوند. و این به معنی تجمیع سیاستها در یک قسمت است. و جلوی کد نویسی های تکراری را می گیرد.

می توان مشخص نمود که عملیات اعتبار سنجی برای کدام سناریو اجرا شود و اگر مشخص نکنیم اعتبار سنجی برای همه سناریو ها اجرا خواهد شد.

Safe Attributesچیست ؟

به Attributeهایی که وجود دارند ولی باید از دید کابر مخفی باشدunsafe گفته می شود مثل Attributesمربوط به کلید اصلی زیرا اگر کلید اصلی در فرم نشان داده شود (که لازم نیست) این باعث می شود که یک هکر بتواند به سایر فیلدها هم دسترسی پیدا کند. بنابر این یک Attributesunsafr یک Attributes است که از دید کاربر مخفی می شود. سایر Attribute ها از نوع safe بوده و برای کاربر نشان داده می شوند.

هر فیلدی که در اعتبار سنجی در یک سناریو قرار بگیرد در همان سناریو safe استمثال :

```
array('username, password', 'required', 'on'=>'login,
register'),
array('email', 'required', 'on'=>'register'),
```

در مثال بالا attributes های username و password در سناریو loginو register تعریف شده اند لذا در فرم های با سناریو login وregister نمایش داده خواهند شد. ولی attributes با نام email تنها در سناریو register تعریف شده و در فرم های با همین سناریو safe است و نشان داده می شود و در فرم با سناریو login به صورت unsafe است و نشان داده نمی شود.

ساخت یک نمونه از مدل در کنترلر با سناریو login

```
// in login scenario
$model=new User('login');
if(isset($_POST['User']))
    $model->attributes=$_POST['User'];
```

ساخت یک نمونه از مدل در کنترلر با سناریو register

```
// in register scenario
$model=new User('register');
if(isset($_POST['User']))
    $model->attributes=$_POST['User'];
```

گاهی لازم است که ما اعلام کنیم که یک attribute حالت safe است حتی اگر در validation rules ما نیامده باشد برای این کار به شکل زیر عمل می کنیم :

```
array('content', 'safe')
```

و به همین ترتیب برای unsafe بودن به شکل زیر عمل می کنید :

```
array('permission', 'unsafe')
```

نکته : قوانین اعتبار سنجی تنها برای attribute هایی استفاده می شوند که مقادیرش را از کاربر می گیرند و برای attribute هایی که نیاز به ورود داده ها از کاربر ندارند نیازی به تعریف rules نیست.

پیامهای خطای صادر شده :

وقتی عملیات اعتبار سنجی اجرا شد پیامهای خطای صادر شده توسط CModel::getErrors() و یا CModel::getError(). قابل دسترسی هستند.

CModel::getError(). فقط اولین پیام خطای صادر شده را بر می گرداند.

CModel::getErrors() . همه پیامهای خطای صادر شده را بر می گرداند.

تعیین پیام خطا :

می توان پیغام خطای مناسب را برای هر rule مشخص نمود که این کار بوسیله مقداردهی message انجام می شود : مثال

```
array('name', 'required', 'message'=>'{attribute} is required')
```

در مثال بالا {attribute}برچسب فیلد جاری را نشان می دهد.

```
array('family', 'required', 'message'=>'Family is required')
```

public function behaviors()

در این متد لیستی از رفتارها که این مدل باید از خود نشان دهد مشخص می شود. خروجی این متد آرایه ای از behavior configurations می باشد. هر behavior configurations می تواند یک رشته تعریف کننده رفتار باشد و یا آرایه ای با ساختار زیر یاشد :

```
<pre>
  'behaviorName'=>array(
      'class'=>'path.to.BehaviorClass',
      'property1'=>'value1',
      'property2'=>'value2',
  )
</pre>
```

کلاس behavior باید از واسط IBehavior استفاده کند و یا از کلاس CBehavior ارث بری نماید. رفتارها هنگام ساخت یک نمونه از مدل به آن نمونه الحاق می شوند.برای مطالعه کامل رفتارها می توان راهنمای شی گرایی در PHP را مطالعه کرد و یا به راهنمای CComponent مراجعه نمود.

public function attributeLabels()

این متد برچسب attribute ها را بر می گرداند. هر attribute دارای یک نام و یک برچسب و یک مقدار می باشد که این متد تنها برچسب آن attribute را بر می گرداند.از آنجایی که تمامی قسمتهایی که کاربر در آنها در فرم اطلاعات را وارد می کنند دارای یک عنوان است لذا می توان عنوان هر فیلد وردی را در مدل مشخص نمود. هر چند می توان این کار را در خود فرم یا در ویو مورد نظر قرار داد ولی این قابلیت برای مدل لحاظ شده است. تمامی عنواوین باید در متد attributeLabels() موجود در مدل مشخص شوند. مثال :

```
public function attributeLabels()
{
     return array(
          'username' => 'Username',
          'password' => 'Password',
     );
}
```

برچسب یا عنوان ورودی username برابر Username و بر چسب ورودی password برابر Password قرار گرفته است. به طور پیش فرض همانطور که مشاهده می شود نام هر attribute با برچسب آن یکی در نظر گرفته می شود ولی در برچسب حرف اول حرف بزرگ انگلیسی است.

protected function afterConstruct()

این متد پس از ساخت یک نمونه از روی مدل بلافاصله اجرا می شود.

protected function beforeValidate()

این متد درست قبلاجرای متد اعتبار سنجی اجرا می شود.

protected function afterValidate()

این متد بعد از اجرای متد اعتبار سنجی بلافاصله اجرا می شود

public function getAttributeLabel($attribute)

این متد نام یک attribute را گرفته و برچسب آن را بر می گرداند.

public function hasErrors($attribute=null)

این متد مشخص می کند که نتیجه اعتبار سنجی برای این attribute خطایی را به دنبال دارد یا خیر.

public function getErrors($attribute=null)

این متد تمامی پیامهای خطای صادر شده را برای یک attributeبه صورت یکجا نشان می دهد.

public function getError($attribute)

این متد اولین پیام خطای صادر شده را برای یک attribute نشان می دهد.

public function addError($attribute,$error)

این متد پیام خطای جدیدی را به یک attribute اضافه می کند.

public function addErrors($errors)

این متد لیستی از پیامهای خطا را به شکل attributeName->ErrorMessage برای attribute ها ایجاد می کند.

public function clearErrors($attribute=null)

این متد تمامی پیامهای خطای صادر شده برای یک یا همه Attribute ها را پاک می کند.

public function getAttributes($names=null)

این متد مقادیر value های موجود در تمامی Attribute ها و یا یک attribute مشخص را در قالب یک آرایه بر می گرداند. اگر $names=null تعریف شود برای همه برگردانده می شوند.

public function setAttributes($values,$safeOnly=true)

مقادیر تمامی attribute ها را به طور یکجا تعیین می کند که به این روش massive assignmet میگویند و باعث افزایش امنیت می گردد.

نکته : در massive assignment تنها مقادیر attribute هایی کپی می شود و قرار می گیرد که از نوع safe هستند و غیر safe ها کپی نمی شوند و این باعث افزایش میزان امنیت می شود.

پارامتر $safeOnly مشخص می کند که عملیات انتساب آیا باید برای همه attribute ها انجام شود یا تنها برای safe attribute ها انجام شود.

public function unsetAttributes($names=null)

این متد مقادیر Attribute ها را خالی می کند.

public function getScenario()

این متد نام سناریویی را که این مدل در آن کار می کند را مشخص می کند. سناریو مشخص می کند که عملیات اعتبار سنجی به چه شکلی انجام شود و کدام attribute ها می توانند به صورت یکجا یا massive مقدار دهی شوند. به اینصورت که هر کلاس می تواند در یک سناریو تعریف شود مثل :

```
class Users extends CActiveRecord('register')
```

این مدل کلاس در سناریو register تعریف شده است. حال در متد rule هر کدام از attribute ها که در این سناریو تعریف شوند به صورت massive مقدار دهی می شوند. مثل اعتبار سنجی زیر :

```
array('password', 'compare', 'on'=>'register'),
```

public function setScenario($value)

سناریو مربوط به این مدل را تعریف می کند.

public function getSafeAttributeNames()

لیستی از attribute ها که در این مدل safe هستند و به صورت massive مقدار دهی می شوند را مشخص می کند.

متد های تعریف شده در CActiveRecordو قابل استفاده در کلاسهای مدل نوع CActiveRecord :

نکته : متد های تعریف شده در CFormModel تنها به متد سازنده و چند متد داخلی محدود می شود. لذا در این قسمت به توضیح متدهای پرکاربرد در CactiveRecord می پردازیم.

متغیر CDbConnection برای تمامی کلاسهای active record معرف ارتباط بانک اطلاعاتی یا database connection می باشد که داخل متغیر db قرار می گیرد.

```
public static $db;
```

برای توضیحات بیشتر در این مورد به getDbConnection رجوع کنید.

```
public function init()
```

این متد زمانی اجرا می شود که یک نمونه از روی مدل ساخته شود.

```
public function
getRelated($name,$refresh=false,$params=array())
```

این متد رکوردهای در رابطه را بر می گرداند. اگر رابطه ما از نوع HAS_ONE و یا BELONGS_TO باشد این متد یک شی را بر می گرداند و اگر این رابطه موجود نباشد مقدار null را بر می گرداند. اگر رابطه از نوع HAS_MANY و یا MANY_MANY باشد این متد آرایه ای از اشیا را بر می گرداند.

```
public function hasRelated($name)
```

این متد مشخص می کند که آیا اشیا در رابطه که نام آنها توسط پارامتر $name مشخص شده است بارگزاری شده اند یا خیر.

```
public function getDbCriteria($createIfNull=true)
```

این متد کلیه criteria های تعریف شده برای این مدل را بر می گرداند. این متد با متد scope در رابطه است.

```
public function setDbCriteria($criteria)
```

این متد criteria ها را برای این مدا مشخص می کند.

```
public function defaultScope()
```

این متد نام پیش فرض scope را مشخص می کند که باید برای تمامی query های این مدل لحاظ شود. البته این متد تنها برای query های SELECT عمل می کند. برای یک مدل می توان یک حوزه دسترسی از بانک اطلاعاتی را مشخص نمود که توسط آن دسترسی کاربر به یک فضای مشخص از بانک اطلاعاتی محدود شود و این کار توسط تعیین criteria ها در متد scope انجام می شود.

public function resetScope()

این متد تمامی scope ها و criteria های تعریف شده و همچنین defaultScope را غیر فعال و می کند.

public static function model($className=__CLASS__)

این متد یک نمونه static از این کلاس را بر می گرداند. کلیه کلاسهایی که قصد کار کردن با AR را دارند باید این کلاس را به شکل زیر override نمایند :

```
<pre>
  public static function model($className=__CLASS__)
  {
      return parent::model($className);
  }
</pre>
```

public function getMetaData()

این متد meta-data های این کلاس مدل را بر می گرداند.

public function refreshMetaData()

این متد meta-data های این کلاس مدل را refresh می کند. این متد زمانی مفید است که table schema تغییر کرده باشد و شما بخواهید آخرین تغییرات را اعمال کنید.

public function tableName()

این متد نام جدول مورد استفاده در بانک اطلاعاتی را بر می گرداند. به طور پیش فرض این متد نام کلاس را به عنوان نام جدول بر می گرداند.

public function primaryKey()

این متد فیلد اصلی از جدول مورد استفاده در بانک اطلاعاتی را بر می گرداند. اگر کلید اصلی تنها روی یک فیلد تعریف شده باشد این متد نام آن فیلد را بر می گرداند و اگر کلید اصلی روی چند رکورد اعمال شده باشد این متد آرایه ای را بر میگرداندکه شامل نام تمامی آن فیلدها است.

```
public function relations()
```

ابزار gii به طور خودکار رابطه های موجود بین جدلول را در model وارد می کند. نکته مهم این است که در رابطه های باید از Foreign Key استفاده شده باشد.یک بانک اطلاعاتی می تواند از جندین نوع موتور مختلف استفاده کند که به آنها Storage Engine می گوییم. برای نمایش آن در PhpMyAdmin در جدول مورد نظر وارد شده و در قسمت Operations می توانیم وارد شویم. حال در قسمت Storage Engine می توان موتور مورد نظر را انتخاب کرد. ند موتور معروف :

MyISAM : موتوری ساده، با امکانات کمتر، پیش فرض، قابلیت کار با Foreign Key ها را ندارد

InnoDB : موتوری پیشرفته تر با قابلیت های زیاد تر ، امکان تعریف کلید های خارجی برای جداول و کار با Transactions و سایر امکانات بانک اطلاعاتی.

برای اعلام related objects ها این متد باید override شود. در کل چهار نوع رابطه بین active record object ها می تواند تعریف شود که عبارتند از :

1- BELONGS_TO: e.g. a member belongs to a team
2- HAS_ONE: e.g. a member has at most one profile
3- HAS_MANY: e.g. a team has many members
4- MANY_MANY: e.g. a member has many skills and a skill belongs to a member

علاوه بر چهار نوع رابطه بالا یک نوع رابطه دیگر با نام STAT نیز مطرح می شود که می توان از آن استفاده کرد و برای اجرای statistical query ها و یا aggregational query ها (پرس و جو های تجمیعی) مورد استفاده قرار می گیرد.این نوع رابطه اطلاعات تجمیعی در مورد اشیا در رابطه را بر می گرداند. مثلا تعداد پیامها برای هر مطلب پست شده و یا معدل بازدید از یک محصول.

هر کدام از رابطه های تعریف شده در این متد در قالب زیر تعریف می شوند :

```
<pre>

'varName'=>array('relationType', 'className', 'foreign_key',
...additional options)
```

```
</pre>
```

پارامتر varName به نام variabl و یا property مربوطه اشاره دارد و پارامتر relationType مشخص کننده نوع رابطه است که انواع آن در بالا آمد و پارامتر className به active record class اشاره دارد که رابطه در آن تعریف می شود و پارامتر foreign_key به کلید خارجی اشاره دارد که رابطه دو فیلد را برقرار می کند.

نکته : اگر کلید خارجی ترکیبی از چند فیلد باشد باید آن فیلدها همگی آورده شده و با علامت کاما از یکدیگر جدا شوند. و برای رابطه های MANY_MANY ارتباط دو جدول باید مانند join_table('fk1, fk2') تعریف شود. پارامترهای اضافی باید مانند الگوی name-value تعریف شوند که می توان برای توضیحات بیشتر به راهنمای متد relations مراجعه نمود.

مثال :

```
<pre>
return array(
    'author'=>array(self::BELONGS_TO, 'User', 'author_id'),
    'comments'=>array(self::HAS_MANY, 'Comment', 'post_id',
'with'=>'author', 'order'=>'create_time DESC'),
'tags'=>array(self::MANY_MANY, 'Tag', 'post_tag(post_id,
tag_id)', 'order'=>'name'),
);
</pre>
```

public function scopes()

این متد اعلان scope مورد نظر را بر می گرداند. یک scope یک query criteria می باشد که می تواند با scope دیگر به صورت زنجیره وار در آید. این متد توسط کلاسهای فرزند که از نوع AR هستند ovrride می شود. به عنوان مثال کد زیر دو scope با نامهای recently وpublished را تعریف می کند :

```
    <pre>
      return array(
          'published'=>array(
                'condition'=>'status=1',
          ),
          'recently'=>array(
                'order'=>'create_time DESC',
                'limit'=>5,
          ),
      );
    </pre>
```

اگر این scope در مدل Post تعریف شود ما می توانیم query های زیر را داشته باشیم :

```
<pre>
 $posts=Post::model()->published()->findAll();
 $posts=Post::model()->published()->recently()->findAll();
 $posts=Post::model()->published()->with('comments')-
>findAll();
</pre>
```

توجه کنیئ که آخرین query تعریف شده از نوع رابطه ای می باشد.

public function attributeNames()

این متد نام تمامی attribute های این مدل را بر می گرداند. این می تواند تمامی نام تمامی فیلدهای جدول استفاده شده در این کلاس AR رابر گرداند.

public function getAttributeLabel($attribute)

این متد برچسب attribute مورد نظر را بر می گرداند.

public function getDbConnection()

این متد Database Connection مورد استفاده توسط این AR را بر می گرداند. به طور پیش فرض db component به عنوان Database Connection مورد استفاده قرار می گیرد. و اگر شما قصد دارید با connection دیگری کار کنید می توانید از این متد استفاده کنید.

class CBelongsToRelationextends CActiveRelation

این کلاس پارامترهای مربوط به رابطه نوع BELONGS_TO را بر می گرداند.

class CHasOneRelationextends CActiveRelation

این کلاس پارامترهای مربوط به رابطه نوع HAS_ONE را بر می گرداند.

class CHasManyRelation extends CActiveRelation

این کلاس پارامترهای مربوط به رابطه نوع HAS_MANY را بر می گرداند.

class CManyManyRelation extends CHasManyRelation

این کلاس پارامترهای مربوط به رابطه نوع MANY_MANY را بر می گرداند.

متد ها و متغیرهای تعریف شده در class CActiveRecordMetaData موجود در فایل CActiveRecord :

```
public $tableSchema  // the table schema information
public $columns   // array table columns
public $relations=array()// array list of relations
public $attributeDefaults=array()// array attribute default values
```

public function addRelation($name,$config)

این متد یک رابطه را ایجاد می کند. $config یک آرایه با سه عنصر است که عبارتند از relation type و active record class و foreign key.

public function hasRelation($name)

این متد بررسی می کند که آیا رابطه ای با نام مشخص شده وجود دارد یا خیر.

public function removeRelation($name)

این متد یک رابطه را که توسط نام آن مشخص شده است حذف می کند.

public function getActiveRelation($name)

این متد اعلان (declaration) رابطه ای با نام مشخص شده را بر می گرداند.

public function getTableSchema()

این متد metadata مربوط به جدولی که این AR متعلق به آن است را بر می گرداند.

public function getCommandBuilder()

این متد command builder مربوز به این AR را بر می گرداند.

public function hasAttribute($name)

این متد بررسی می کند که آیا این جدول attribute با نام ذکر شده دارد یا خیر.

public function getAttribute($name)

این متد مقدار attribute با نام داده شده را بر می گرداند. اگر هیچ رکوردی پیدا نشود مقدار پیش فرض تعریف شده برای این attribute را بر می گرداند. اگر هم که این رکورد نتیجه یک query باشد و تاکنون query بار گزاری و مقدار دهی نشده باشد مقدار null را بر می گرداند.

نکته : شما همچنین می توانید از طریق $this->AttributeName به مقدار موجود در Attribute دسترسی پیدا کنید.

public function setAttribute($name,$value)

این متد مقدار attribute مشخص شده را تعیین می کند. شما همچنین می توانید با استفاده از دستور زیر این کار را انجام دهید:

```
$this->AttributeName = newValue;
```

public function getAttributes($names=true)

این متد مقدار تمامی Attribute ها را بر می گرداند. اما مقادیر اشیا در رابطه را بر نمی گرداند. اگر مقدار پارامتر آن true باشد که پیش فرض هم هست همه مقادیر attribute ها بر گردانده می شود که شامل آنهایی که هنوز از بانک اطلاعاتی بارگزاری نشده اند نیز می شود و مقدار آنها null برگردانده می شود و اگر مقدار آن false باشد تنها attribute هایی برگردانده می شوند که از بانک اطلاعاتی مقدارشان بارگزاری شده باشد.

public function save($runValidation=true,$attributes=null)

این متد رکورد جاری را ذخیره می کند. این متد رکورد جدیدی را به جدول اضافه می کند اگر پارامتر isNewRecord آن برابر true باشد. در غیر این صورت این متد عملیات update رکورد را انجام می دهد.

نکته : همواره عملیات اعتبار سنجی قبل از عملیات Save انجام می گیرد. و اگر اعتبارسنجی با شکست همراه شود عملیات ذخیره سازی انجام نمی گیرد. شما می توانید از متد ()getErrors برای پیدا کردن خطاهای اعتبارسنجی استفاده نمایید.اگر رکورد جدید اضافه شود پارامتر isNewRecord آن برابر false می شود و پارامتر scenario آن برابر update قرار می گیرد و اگر کلیدی از جدول دارای خاصیت auto-incremental باشد یکی به آن اضافه می شود.

public function getIsNewRecord()

اگر رکورد جدیدی قرار باشد Save شود این متد مقدار true را بر می گرداند.

public function onBeforeSave($event)

این متد دقیقا قبل از اجرای متد save اجرا می شود. و بعد از اجرای متد اعتبار سنجی. این متد توسط متد beforeSave فراخوانی می شود.

public function onAfterSave($event)

این متد درست بعد از اجرای متد save اجرا می شود. این متد توسط متد afterSave فراخوانی می شود.

public function onBeforeDelete($event)

این متد دقیقا قبل از اجرای متد delete اجرا می شود.این متد توسط متد beforeDelete فراخوانی می شود.

public function onAfterDelete($event)

این متد درست بعد از اجرای متد delete اجرا می شود.این متد توسط متد afterDelete فراخوانی می شود.

public function onBeforeFind($event)

این متد دقیقا قبل از اجرای یک فرمان find call اجرا می شود.در این رویداد خاصیت CModelEvent::criteria محتوی query criteria می باشد که به صورت پارامتر به متدهای find ارسال می شود.

public function onAfterFind($event)

این متد درست بعد از اجرای یک فرمان find call اجرا می شود.

protected function beforeSave()

این متد دقیقا قبل از اجرای متد save اجرا می شود. و بعد از اجرای متد اعتبار سنجی.

protected function afterSave()

این متد بعد از اجرای متد save که به صورت موفقیت آمیز اجرا شده است اجرا می شود.

protected function beforeDelete()

این متد دقیقا قبل از اجرای متد delete اجرا می شود.

protected function afterDelete()

این متد درست بعد از اجرای متد delete اجرا می شود.

protected function beforeFind()

این متد دقیقا قبل از اجرای یک فرمان find call اجرا می شود. انواع متد جستجو عبارتند از :

Find و findAll و findByPk و findAllByPk و findByAttributes و findAllByAttributes

protected function afterFind()

این متد پس از نمونه سازی از روی هر رکورد توسط یک متد جستجو اجرا می شود.

public function insert($attributes=null)

این متد یک رکورد جدید را در جدول ذخیره می کند. متد اعتبار سنجی برای این متد اجرا نمی شود. شما می توانید با استفاده از متد validate اعتبارسنجی را فراخوانی نمایید.

public function update($attributes=null)

این متد رکورد ارائه شده توسط این AR را به روز رسانی می کند. . متد اعتبار سنجی برای این متد اجرا نمی شود. شما می توانید با استفاده از متد validate اعتبارسنجی را فراخوانی نمایید.

public function saveAttributes($attributes)

این متد لیستی از attribute های انتخاب شده را ذخیره می کند. برخلاف متد save این متد تنها attribute ها را ذخیره می کند و متدهای beforeSave و afterSave را فراخوانی نمی کند. همچنین هیچ گونه متد فیلتر یا اعتبار سنجی را اجرا نمی کند. بنابر این این متد را برای درخواستهای غیر مشخص و نا امن اجرا نکنید. شما می توانید از کدهای جایگزین زیر استفاده نمایید :

```
<pre>
$postRecord=Post::model()->findByPk($postID);
$postRecord->attributes=$_POST['post'];
$postRecord->save();
</pre>
```

public function delete()

این متد رکورد مرتبط با این AR را حذف می کند.

public function refresh()

این متد رکورد جاری را با آخرین مقادیری که در آن قرار داشته است به روزرسانی می کند.

public function equals($record)

این متد رکورد جاری در AR را با یک رکورد دیگر مقایسه می کند.

public function getPrimaryKey()

این متد مقدار کلید اصلی را برای رکورد جاری جدول بر می گرداند.

```
public function setPrimaryKey($value)
```

این متد مقدار کلید اصلی را برای رکورد جاری تعیین می کند.

```
public function getOldPrimaryKey()
```

مقدار قبلی کلید اصلی را بر می گرداند و اگر قبلا مقداری نداشته است null را بر می گرداند.

```
public function setOldPrimaryKey($value)
```

مقدار قبلی کلید اصلی را تعیین می کند.

```
private function query($criteria,$all=false)
```

این متد بر اساس criteria داده شده یک query ایجاد می کند.

```
public function applyScopes(&$criteria)
```

این متد query scope ها را برای criteria داده شده اعمال می کند.

```
public function getTableAlias($quote=false,
$checkScopes=true)
```

این متد alias های این جدول را برای استفاده در متدهای جستجو بر می گرداند.

```
public function setTableAlias($alias)
```

این متد alias های این جدول را تعیین می کند.

```
public function find($condition='',$params=array())
```

این متد یک رکورد را در AR جاری بر اساس شرایط تعیین شده پیدا می کند.

```
public function
findAll($condition='',$params=array())
```

این متد تمامی رکوردهای موجود در این AR را بر می گرداند.

```
public function
findByPk($pk,$condition='',$params=array())
```

این متد یک رکورد موجود در این AR را بر اساس کلید اصلی اعلام شده توسط پارامتر pk پیدا می کند.

```
public function
findAllByPk($pk,$condition='',$params=array())
```

این متد تمامی رکوردهای موجود در این AR که کلید اصلی آنها مشخص شده است را بر می گرداند.

```
public function
findByAttributes($attributes,$condition='',$params=array())
```

این متد یک رکورد جاری را در این AR با attribute های اعلام شده و مقادیر آنها پیدا می کند.

```
public function
findAllByAttributes($attributes,$condition='',$params=array())
```

این متد همه رکوردهایی را پیدا می کند که دارای یک مقدار مشخص برای attribute اعلام شده باشند.

```
public function findBySql($sql,$params=array())
```

این متد یک رکورد موجود را بر اساس یک عبارت SQLجستجو می کند و بر می گرداند. عبارت sql در پارامتر $sql قرار می گیرد.

```
public function findAllBySql($sql,$params=array())
```

این متد تمامی پارامترهایی را که عبارت Sqlمشخص می کند بر می گردانند.

```
public function count($condition='',$params=array())
```

این متد تعداد رکوردهایی را که عبارت conditionمشخص می کند بر می گرداند.

```
public function
countByAttributes($attributes,$condition='',$params=array())
```

این متد تعداد رکوردهایی را که توسط attribute ها و مقادیر آن مشخص شده است را بر می گرداند.

```
public function countBySql($sql,$params=array())
```

این متد تعداد رکوردهایی را که عبارت sql مشخص می کند بر می گرداند.

```
public function exists($condition='',$params=array())
```

این متد بررسی می کند که آیا رکوردی با شرایط تعیین شده موجود هست یا خیر.

```
public function with()
```

این مند مشخص می کند که کدام شی در رابطه باید بار گزاری شود. این متد تعداد متغیری از پارامترها را دریافت می کند که هر پارامتر نام یک رابطه و یا نام یک فرزند رابطه را مشخص می کند. به عنوان مثال :

```
<pre>
// find all posts together with their author and comments
 Post::model()->with('author','comments')->findAll();
 // find all posts together with their author and the author's
profile
 Post::model()->with('author','author.profile')->findAll();
</pre>
```

رابطه باید در متد relations تعریف شده باشد. به طور پیش فرض پارامترها از تعاریف موجود در متد relation بارگزاری می شوند اما برای سفارشی کردن این متد باید آرایه ای از پارامترها را به متد with ارسال نماییم. که شامل نام رابطه و مقدار آن برابر پرس و جوی تعریف شده توسط رابطه است. به عنوام مثال :

```
<pre>
 Post::model()->with(array(
    'author'=>array('select'=>'id, name'),
    'comments'=>array('condition'=>'approved=1',
'order'=>'create_time'),
 ))->findAll();
</pre>
```

public function together()

این متد مقدار پارامتر CDbCriteria::together را برابر true قرار می دهد که برای توضیحات بیشتر می توان به توضیحات CDbCriteria::together مراجعه کرد.

public function updateByPk($pk,$attributes,$condition='',$params=array())

این متد رکوردهای مشخص شده توسط کلید اصلی را به روز رسانی می کند. برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد. نکته : attribute ها در این متد اعتبار سنجی نمی شوند.

public function updateAll($attributes,$condition='',$params=array())

این متد رکوردهایی را که توسط condition مشخص می شوند به روزرسانی می کند. . برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد. نکته : attribute ها در این متد اعتبار سنجی نمی شوند.

```
public function
updateCounters($counters,$condition='',$params=array())
```

این متد یک یا چند فیلد شمارنده را به روزرسانی می کند که توسط condition مشخص شده باشند. برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد.

```
public function
deleteByPk($pk,$condition='',$params=array())
```

این متد رکورهایی را که توسط کلید اصلی مشخص شده اند حذف می کند. . برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد.

```
public function
deleteAll($condition='',$params=array())
```

این متد رکورهایی را که توسط condition مشخص شده اند حذف می کند. برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد.

```
public function
deleteAllByAttributes($attributes,$condition='',$params=array())
```

این متد رکورهایی را که توسط attribute مشخص شده اند حذف می کند. برای توضیحات بیشتر در مورد پارامترهای این متد می توان به متد find مراجعه کرد.

```
protected function instantiate($attributes)
```

این متد یک AR را نمونه سازی می کند. شما می توانید از این متد مثلا برای ساخت یک رکورد جدید بر اساس مقدار یک فیلد مشخص استفاده کنید.

```
class CBaseActiveRelation extends CComponent
```

این کلاس یک کلاس پایه برای همه کلاسهای AR است. که شامل متغیرهای قابل استفاده زیر است :

public $name; // نام شی در رابطه

public $className; // نام کلاس

public $foreignKey; // نام کلید خارجی رابطه

public $select='*'; // نام فیلد های جدول SELECT

public $condition=''; // مشخصات فیلتر WHERE

public $params=array(); // مشخصات پارامترها

public $group=''; // مشخصات گروه ها GROUP BY

public $join=''; // مشخصاتJOIN الحاق

public $having=''; // مشخصات شمول HAVING

public $order=''; // مشخصات ترتیب ORDER BY

public function addRelation($name,$config)

این متد یک رابطه جدید را ایجاد می کند.

public function hasRelation($name)

این متد مشخص می کند که آیا رابطه ای با این نام وجود دارد یا خیر.

دسترسی به مدل از طریق کنترلر :

هر مدل فرم در یک اکشن مرتبط در کنترلر ساخته می شود. و در واقع در این اکشن است که یک نمونه از روی مدل ساخته و اجرا می شود. برای تعریف اکشن مرتبط با مدل فرم به کدهای زیر نیاز داریم :

```
$model=new ProductModel('register');

if(isset($_POST[ProductModel]))
{
    $model->attributes=$_POST[ProductModel];
}

$model=new ProductModel('register');
```

این کد یک شی با نام $model از روی ProductModelایجاد می کند و اعلام می دارد که از این مدل ما برای حالت Active Record استفاده می کنیم. یعنی برای عملیات ثبت داده ها در بانک اطلاعاتی. این شی ساخته شده تنها یک فرم خالی است و هیچ داده ای در آن قرار ندارد. نکته قابل توجه تعیین سناریو این شی است که اعلام می دارد از این مدل ما برای ثبت داده ها با سناریو register استفاده می کنیم. تعیین سناریو را می توان برای شی ساخته شده به شکل زیر نیز تعیین نمود :

```
$model=new User'register';
```

```
$model->scenario='register';

if(isset($_POST[ProductModel]))
```

این کد بررسی می کند که آیا اطلاعات درون فرم وارد شده است یا خیر. در صورتی که داده های داخل فرم به درستی وارد نشده باشند و اعتبارسنجی با موفقیت انجام نشده باشد این بلوک if اجرا نخواهد شد.

```
$model->attributes=$_POST[ProductModel];
```

این کد عملیات Massive Assignment را انجام می دهد که به واسطه آن تمامی اطلاعات وارد شده در فرم به طور یک جا (و نه تک به تک) وارد شی مدل می شود.و این وارد کردن یک جا باعث افزایش ضریب امنیت می شود. کد بالا معادل کد زیر است :

```
foreach($_POST['LoginForm'] as $name=>$value)
{
    if($name is a safe attribute)
        $model->$name=$value;
}
```

مدل ها در کنترلر هر جا که لازم باشد ساخته می شوند و متد مورد نظر آنها اجرا می شود. مثال زیر قسمتی از کد موجود در کنترلر است با فرض این که یک مدل با نام Users قبلا تعریف شده باشد و متدی با عنوان Create در آن باشد :

```
public function actionCreate()
{
$model=new Users;
     if(isset($_POST['Users']))
     {
     $model->attributes=$_POST['Users'];
     if($model->save())
     $this->redirect(array('view','id'=>$model->username));
     }
     $this->render('create',array(
     'model'=>$model,
));
}
```

ارسال پارامترها برای $model توسط دستور ;$model->attributes=$_POST['Users'] انجام می شود که به آن massive assignment گفته می شود زیرا به طور همزمان همه attribute های مورد نیاز مدل را توسط مقادیر وارد شده توسط فرم پر می کند.

صداکردن متد مورد نظر در مدل توسط کنترلر :

```
$this->render('create',array(
```

```
'model'=>$model,
));
```

این کد متد Create موجود در مدل نمونه برداری شده را اجرا می کند. و پارامتر model را با مقادیر $model که قبلا پر شده بود برای این متد ارسال می کند. البته متد create در محتوای داخلی فایل Users.php که کلاس مدل اصلی است مشاهده نمی شود و چون کلاس Users از کلاس CActiveRecord ارث بری می کند متد Create در کلاس CActiveRecord موجود می باشد. به کد موجود در کلاس Users توجه کنید :

```
class Users extends CActiveRecord
```

دسترسی به فیلدهای جدول در داخل مدل از طریق $this->FieldName و خارج از آن اگر $model یک نمومه از مدل باشد به صورت $model->FieldName خواهد بود

نکات لازم در طراحی مدل :

۱- شامل property هایی است که داده ها را توصیف می کنند.

۲- شامل متدهای Validation Rules هستند.

۳- ممکن است شامل کدهایی برای دستکاری داده ها باشند مثل Search ویا هر نوع عملیات دیگر بر روی داده ها در مدل انجام می شود.

۴- نباید شامل کدهایی باشد که در آنها اطلاعات end user مورد نیاز است مثل کدهای شامل $_GET و$_POST بلکه این نوع کدها فقط در کنترلر قابل تعریف می باشند.

۵- نباید شامل متدها و کد هایی باشند که مستقیما توسط کاربران مورد استاده قرار می گیرد. مثلا متدی که در یک فرم کاربر مستقیما فراخوانی می شود.

۶- می توان از از کدهای PHP و SQL استفاده کرد.

۷- نباید در آن از کدهای HTML در آن استفاده کرد چون مدل حالت توصیفی ندارد و نمایش ها فقط از طریق view انجام می شوند.

۸- مدل ها معمولا چاق هستند و شامل کدهای زیادی می شوند.

۹- کلیه عملیات کار و پردازش داده های بانک اطلاعاتی تنها باید در مدل انجام شود.

ویو ها

معرفی ویو ها :

نام یک ویو با نام فایل آن یکسان است مثلا ویویی که در فایل edit.php قرار دارد با نام edit شناخته می شود. به ازای هر فایل کنترلر یک پوشه با همان نام ایجاد می شود مثلا کنترلر siteController.php دارای یک پوشه با عنوان site در دایرکتوری view است. می توان به ازای هر اکشن در کنترلر یک فایل ویو داشته باشیم.

برای فراخوانی یک ویو در کنترلر از دستور CController::render() استفاده می کنیم. مثال :

```
$this->render('edit');
```

در داخل ویو ما می توانیم با کلمه کلیدی $this به کنترلری که این ویو را فراخوانی و اجرا کرده است دسترسی پیدا کنیم. مثلا در ویو ما توسط دستور $this->propertyName می توانیم مقدار propertyName را که در کنترلر اعلام شده دسترسی پیدا کنیم.

ارسال پارامترها و مقادیر آنها از کنترلر به ویو :

```
$this->render('edit', array(
    'var1'=>$value1,
    'var2'=>$value2,
));
```

سپس در ویو می توان به این پارامترها دسترسی پیدا کرد. به عنوان مثال در مثال بالا در ویو از طریق **$var1** و **$var2.** مقادیر آنها فراخوانی می شود.

فراخوانی یک ویو در ویو دیگر

```
<?php $this->renderPartial('_footer'); ?>
```

از این روش همچنین برای اجرای همزمان چند ویو در یک صفحه استفاده می شود.

دسترسی به کنترلر از داخل ویو :

```
echo $this->propertyname;
```

ارسال پارامتر از کنترلر به ویو

```
$theTime = date("D M j G:i:s T Y");
     $this->render('helloWorld',array('time'=>$theTime));
```

دریافت پارامتر ارسالی از کنترلر توسط ویو

```
     <h3><?php echo $time; ?></h3>
```

در یکویو $this به کنترلرجاری وصاحباینویواشارهدارد.

دسترسی به ویو از طریق کنترلر :

```
$this->render('edit');
$this->render ('edit', array('a'=>'1' , 'b'=>'2'));
```

در مثال بالا در ویو می توان از طریق $a , $b به مقادیر آنها که ۱و ۲ است دسترسی پیدا کرد.

دسترسی به خروجی مدل در ویو :

در مدل داریم :

```
class TblUser extends CActiveRecord
{
     public function first()
     { return  1; }
```

در ویو داریم :

```
echo TblUser::model()->first();
```

نکات لازم در طراحی view :

۱- کار view صرفا نمایش اطاعات است و کلیه محاسبات در کنترلر انجام می گیرد.

۲- ممکن است در ویو از property ها و متدهای مدل ها و کنترلر ها استفاده کند ولی باید این کار جهت نمایش اطلاعات باشد و نه انجام محاسبات.

۳- نباید شامل کدهایی باشد که در آنها اطلاعات end user مورد نیاز است مثل کدهای شامل $_GET و$_POST بلکه این نوع کدها فقط در کنترلر قابل تعریف می باشند.

۴- می توان از از کدهای PHP و HTML استفاده کرد.

۵- نباید در آن از کدهای SQL در آن استفاده کرد و کار با داده ها ها فقط از طریق مدل انجام می شوند.

۶- ویو ها باید تا حد امکان ساده و شامل کدهای کم و موثر باشند.

۷- کلیه عملیات مربوط به نمایش اطلاعات را انجام می دهد.

۸- باید ویو تا حد امکان ساده و قابل فهم باشد.

معرفی قالب ها :

یک layout برای قالب بندی کلی و تشکیل یک صفحه کامل وب استفاده می شود که مقادیر ثابتی مثل سرصفحه و ته صفحه را در آن قرار داد. مثال :

```
......header here......
<?php echo $content; ?>
......footer here......
```

هر جا <?phpecho$content; ?> نوشته شود محتوای ویو در آن قرار می گیرد.

به طور پیش فرض protected/views/layouts/main.php به عنوان layout انتخاب می شود مگر اینکه layout دیگری تعریف شود. تغییر layout به دو شکل قابل انجام است :

از طریق config به شکل CWebApplication::layout

۱- از طریق Controller به شکل CController::layout

اگر بخواهیم یک ویو اجرا شود و هیچ layout در آن اعمال نشود به جای دستور render از دستور renderPartial() در کنترلر استفاده می کنیم.

Layout مورد استفاده در برنامه در کنترلر تعریف می شود که می توان آن را تغییر داد :

```
public $layout='//layouts/column2';
```

در layout دیستور زیر باعث درج ویو ها داخل فایل layout می شود

```
<?php echo $content; ?>
```

جهت تعیین layout پیش فرض می توان در فایل Application دستور زیر را وارد کرد:

```
Layout -> 'LayoutName';
```

نکته :

در فایل protected\components\controller.php یک خط فرمان به شکل زیر وجود دارد :

```
    public $layout='//layouts/column1';
```

که قالب پیش فرض برنامه را مشخص می کند (برای کنترل هایی که مجددا قالب آنها مقدار دهی نشده باشد) که می توان آن را اصلاح کرد.

نکته :

در ابتدای هر فایل کنترلر نیز به طور اختصاصی ممکن است قالب تعریف شده باشد مثل :

```
public $layout='//layouts/column2';
```

قرار دادن یک layout درون layout دیگر :

فرض کنیک یک layout به نام main.php داشته باشیم و یک layout به نام footer.php_ که می خواهیم footer_ را درون main قرار دهیم بدین منظور در main هر جا بخواهیم footer_ نشان داده شود کد زیر را می نویسیم :

```
<?php $this->beginContent('//layouts/_footer'); ?>
        <?php echo $content; ?>
<?php $this->endContent(); ?>
```

Partial view

Partial view یا ویو های جزئی ویو هایی هستند که جهت نمایش در سایر ویوها استفاده می شوند و به طور مستقل بهتر است نمایش داده نشوند. نام فایل آنها با کاراکتر _ شروع میشود مثل view.php_ که داخل index.php مربوط به ویو فراخوانی و نشان داده می شود.

فراخوانی یک ویو در ویو دیگر

```
<?php $this->renderPartial('_footer'); ?>
```

System View

System View ها جهت نمایش error ها و ثبت رویداد ها به کار می روند.

عملکرد System View :

۱) Display errors
۲) Logging information

System View ها ویو هایی هستند که مثلا در هنگام بروز خطا نشان داده می شوند و در مسیر yii/framework/views قرار دارند ولی می توان در مسیر protected/views/system ویو هایی با همان نام مشابه مثل error۴۰۴ ایجاد کرد و در واقع ویو قبلی را با ویو کاربر Overload کرد.

نام ویوهای سیستمی به شکل errorXXX می باشد مثل error400

Widget

ویجت برای نمایش ساخت یافته مقادیر به کار می رود مثل ساخت یک gridview

یک ویجت خود می تواند شامل یک یا چند ویو باشد.از آنجایی که ویجت ها در ویو فراخوانی و استفاده می شودند لذا $this در یک ویجت به ویو حامل ویجت اشاره دارد و نه به کنترلر فراخواننده ویو حامل ویجت.
ویجت های مختلفی برای انجام کارهای مختلف وجود دارد که از یک MaskText گرفته تا DetailView و غیره می توانند باشند.

ساخت یک Widget دلخواه

ابتدا در پوشه components وارد شده و یک فیل با نام face.php ایجاد می کنیم.سپس محتوای آن را به شکل زیر تایپ می کنیم :

```
<?php
class face extends CWidget {
    public $params = array();
    public function run() {
         $this->render('faceView');
    }
}
?>
```

حال یک پوشه با نام views در داخل پوشه components ایجاد می کنیم و داخل آن فایل faceView.php را ایجاد می کنیم و محتوای آن را به شکل زیر قرار می دهیم:

```
<?php
        echo $this->params['param1'];
    ?>
```

حال در ویویی اصلی برنامه که می خواهیم این محتوا چاپ شود کد زیر را وارد می کنیم :

```
<?php $this->widget('application.components.face', array(
   'params' => array(
   'param1' => 'Hello',
  ),
)); ?>
```

از widget ها معمولا برای نمایش ها استفاده می شود. مثل نمایش جداول و لیست ها و ... .

Widget ها معمولا داخل ویو ها قرار می گیرند تا چیز خاصی را نشان دهند مثل یک تقویم یا ساعت یا ... در واقعwidget ها شبیه کامپوننت های از پیش تعریف شده هستند که در ویو ها می توان از آنها استفاده کرد و شبیه Toolbox برای Yii Framework عمل می کنند.

نحوه استفاده از widget با قابلیت درج محتوا در آن :

```
<?php $this->beginWidget('path.to.WidgetClass'); ?>
...body content that may be captured by the widget...
<?php $this->endWidget(); ?>
```

ویا روش دوم بدون نیاز به محتوا

```
<?php $this->widget('path.to.WidgetClass'); ?>
```

فراخوانی widget به همراه تعریف و ارسال پارامترها :

```
<?php $this->widget("MyWidget",array('mask'=>'12' ));  ?>
<?php
$this->widget('CMaskedTextField',array(
    'mask'=>'99/99/9999'
));
?>
```

انواع widget :

۱) Built-in
۲) User-defined

کلیه ویگت ها از کلاس CWidget ارث بری می کنند.برای تعریف یک ویگت جدید توسط کاربر به شکل زیر عمل می کنیم. و باید متدهای init و run آن را override نماییم.

```
class MyWidget extends CWidget
{
    public function init()
    {
        // this method is called by CController::beginWidget()
    }

    public function run()
    {
        // this method is called by CController::endWidget()
    }
}
```

همانند کنترلر ها ویگت ها نیز می توانند ویوهای خود را داشته باشند.به طور پیش فرض ویوهای ویگت داخل پوشه با نام view ذخیره می شوند که این پوشه داخل پوشه ای است که کلاس ویگت در آن تعریف شده است. این ویو ها توسط فراخوانی به شکل CWidget::render() اجرا یا همان render خواهند شد همانند همان روشی که برای فراخوانی یک ویو در کنترلر استفاده می شود. تنها تفاوت آن در این است که هیچ layout برای این ویو ها اعمال نخواهد شد. همچنین $this دخل ویو به ویگت جاری اشاره می کند و نه به کنترلری که آن ویو و ویگت را فراخوانی کرده است.

Widget

A widget should extend from CWidget or its child classes.

The easiest way of creating a new widget is extending an existing widget and overriding its methods or changing its default property values. For example, if you want to use a nicer CSS style for CTabView, you could configure its CTabView::cssFile property when using the widget. You can also extend CTabView as follows so that you no longer need to configure the property when using the widget.

```
class MyTabView extends CTabView
```

```
{
    public function init()
    {
        if($this->cssFile===null)
        {

$file=dirname(__FILE__).DIRECTORY_SEPARATOR.'tabview.css';
            $this->cssFile=Yii::app()->getAssetManager()-
>publish($file);
        }
        parent::init();
    }
}
```

In the above, we override the CWidget::init method and assign to CTabView::cssFile the URL to our new default CSS style if the property is not set. We put the new CSS style file under the same directory containing the MyTabView class file so that they can be packaged as an extension. Because the CSS style file is not Web accessible, we need to publish as an asset.

To create a new widget from scratch, we mainly need to implement two methods: CWidget::init and CWidget::run. The first method is called when we use $this->beginWidget to insert a widget in a view, and the second method is called when we call $this->endWidget. If we want to capture and process the content displayed between these two method invocations, we can start output buffering in CWidget::init and retrieve the buffered output in CWidget::run for further processing.

A widget often involves including CSS, JavaScript or other resource files in the page that uses the widget. We call these files *assets* because they stay together with the widget class file and are usually not accessible by Web users. In order to make these files Web accessible, we need to publish them using CWebApplication::assetManager, as shown in the above code snippet. Besides, if we want to include a CSS or JavaScript file in the current page, we need to register it using CClientScript:

```
class MyWidget extends CWidget
{
    protected function registerClientScript()
    {
        // ...publish CSS or JavaScript file here...
        $cs=Yii::app()->clientScript;
        $cs->registerCssFile($cssFile);
        $cs->registerScriptFile($jsFile);
```

```
    }
}
```

A widget may also have its own view files. If so, create a directory named views under the directory containing the widget class file, and put all the view files there. In the widget class, in order to render a widget view, use $this->render('ViewName'), which is similar to what we do in a controller.

4.

To create a new widget from scratch, we mainly need to implement two methods: CWidget::init and CWidget::run. The _rst method is called when we use $this->beginWidget to
insert a widget in a view, and the second method is called when we call $this->endWidget.
If we want to capture and process the content displayed between these two method invocations, we can start output bu_ering in CWidget::init and retrieve the bu_ered output
in CWidget::run for further processing.
A widget often involves including CSS, JavaScript or other resource _les in the page that
uses the widget. We call these _les assets because they stay together with the widget
class _le and are usually not accessible by Web users. In order to make these _les Web
accessible, we need to publish them using CWebApplication::assetManager, as shown in
the above code snippet. Besides, if we want to include a CSS or JavaScript _le in the
current page, we need to register it using CClientScript:

```
class MyWidget extends CWidget
f
protected function registerClientScript()
f
// ...publish CSS or JavaScript file here...
$cs=Yii::app()->clientScript;
$cs->registerCssFile($cssFile);
$cs->registerScriptFile($jsFile);
g
```



```
g
```

A widget may also have its own view _les. If so, create a directory named views under the
directory containing the widget class _le, and put all the view _les there. In the widget
class, in order to render a widget view, use $this->render('ViewName'), which is similar

to what we do in a controller.

4. Customizing Widgets Globally ¶

Note: this feature has been available since version 1.1.3.

When using a widget provided by third party or Yii, we often need to customize it for specific needs. For example, we may want to change the value of CLinkPager::maxButtonCount from 10 (default) to 5. We can accomplish this by passing the initial property values when calling CBaseController::widget to create a widget. However, it becomes troublesome to do so if we have to repeat the same customization in every place we use CLinkPager.

```
$this->widget('CLinkPager', array(
    'pages'=>$pagination,
    'maxButtonCount'=>5,
    'cssFile'=>false,
));
```

Using the global widget customization feature, we only need to specify these initial values in a single place, i.e., the application configuration. This makes the customization of widgets more manageable.

To use the global widget customization feature, we need to configure the widgetFactory as follows:

```
return array(
    'components'=>array(
        'widgetFactory'=>array(
            'widgets'=>array(
                'CLinkPager'=>array(
                    'maxButtonCount'=>5,
                    'cssFile'=>false,
                ),
                'CJuiDatePicker'=>array(
                    'language'=>'ru',
                ),
            ),
        ),
    ),
);
```

In the above, we specify the global widget customization for both CLinkPager and CJuiDatePicker widgets by configuring the CWidgetFactory::widgets property. Note that the global customization for each widget is represented as a key-value pair in the array, where the key refers to the wiget class name while the value specifies the initial property value array.

Now, whenever we create a CLinkPager widget in a view, the above property values will be assigned to the widget, and we only need to write the following code in the view to create the widget:

```
$this->widget('CLinkPager', array(
    'pages'=>$pagination,
));
```

We can still override the initial property values when necessary. For example, if in some view we want to set **maxButtonCount** to be 2, we can do the following:

```
$this->widget('CLinkPager', array(
    'pages'=>$pagination,
    'maxButtonCount'=>2,
));
```

Portlet

Portletنوعی ویگت است و بر ای نمایش اطلاعاتی مانند منو ها که ثابت تر هستند به کار می رود.

CPortlet کلاس پایه برای ویگت های Portlet است.

یک Portlet یک بخش از محتوا را نمایش میدهد. عموما معرف نوار کناری در یک صفحه وب است. برای فراخوانی یک CPortlet به شکل زیر عمل می کنیم : مثلا :

```
<?php $this->beginWidget('zii.widgets.CPortlet'); ?>
    ...insert content here...
<?php $this->endWidget(); ?>
```

یک Portlet همچنین یک گزینه اختیاری به نام title نیز می تواند داشته باشد.همچنین توسط تعریف renderDecoration می توان به گزینه های بیشتری از حالت نمایش Portletدسترسی پیدا کرد.

Property Details

contentCssClass property
public string **$contentCssClass**;

the CSS class for the content container tag. Defaults to 'portlet-content'.

decorationCssClass property
public string **$decorationCssClass**;

the CSS class for the decoration container tag. Defaults to 'portlet-decoration'.

hideOnEmpty property (available since v1.1.4)
public boolean **$hideOnEmpty**;

whether to hide the portlet when the body content is empty. Defaults to true.

htmlOptions property
public array **$htmlOptions**;

the HTML attributes for the portlet container tag.

tagName property
public string **$tagName**;

the tag name for the portlet container tag. Defaults to 'div'.

title property
public string **$title**;

the title of the portlet. Defaults to null. When this is not set, Decoration will not be displayed. Note that the title will not be HTML-encoded when rendering.

titleCssClass property
public string **$titleCssClass**;

the CSS class for the portlet title tag. Defaults to 'portlet-title'.

Method Details

init() method
public void **init**()
Source Code:framework/zii/widgets/CPortlet.php#76 (**show**)

```
public function init()
{
    ob_start();
    ob_implicit_flush(false);

    $this->htmlOptions['id']=$this->getId();
```

```
    echo CHtml::openTag($this->tagName,$this->htmlOptions)."\n";
    $this->renderDecoration();
    echo "<div class=\"{$this->contentCssClass}\">\n";

    $this->_openTag=ob_get_contents();
    ob_clean();
}
```

Initializes the widget. This renders the open tags needed by the portlet. It also renders the decoration, if any.

renderContent() method

protected void **renderContent**()

Source Code:framework/zii/widgets/CPortlet.php#123 (**show**)

```
protected function renderContent()
{
}
```

Renders the content of the portlet. Child classes should override this method to render the actual content.

renderDecoration() method

protected void **renderDecoration**()

Source Code:framework/zii/widgets/CPortlet.php#109 (**show**)

```
protected function renderDecoration()
{
    if($this->title!==null)
    {
        echo "<div class=\"{$this->decorationCssClass}\">\n";
        echo "<div class=\"{$this->titleCssClass}\">{$this->title}</div>\n";
        echo "</div>\n";
    }
}
```

Renders the decoration for the portlet. The default implementation will render the title if it is set.

run() method

public void **run**()

CActiveDataProvider

هر گونه دسترسی به داده ها باید توسط CActiveDataProvider مدیریت شود. در هر فراخوانی و ... مثلا در یک ListView یا GridView باید ابتدا CActiveDataProvider تنظیم شود. مثلا در actionIndex که باعث اجرای ویو view_ می شود که به شکل یک ListView نمایش داده می شود داریم :

```
public function actionIndex()
     {
          $dataProvider=new CActiveDataProvider('Website',
array(
          'criteria'=>array(

     'condition'=>'visit_count > 1',
                              'order'=>'visit_count DESC',
                              ),

            'pagination'=>array(
                                'pageSize'=>2,),)
           );

          $this->render('index',array(
               'dataProvider'=>$dataProvider,
          ));
     }
```

مثال دیگری به شکل زیر است :

```
$dataProvider=new CActiveDataProvider('Post', array(
    'criteria'=>array(
        'condition'=>'status=1',
        'order'=>'create_time DESC',
        'with'=>array('author'),
    ),
    'pagination'=>array(
        'pageSize'=>20,
    ),
));
// $dataProvider->getData() will return a list of Post objects
```

بنابرای کلاس CActiveDataProvider برای ما اهمیت زیادی دارد و متدهای آن به شکل زیر است :

Public Properties

Property	Type	Description	Defined By
criteria	CDbCriteria	Returns the query criteria.	CActiveDataProvider
data	array	Returns the data items currently available.	CDataProvider
id	string	Returns the ID that uniquely identifies the data provider.	CDataProvider
itemCount	integer	Returns the number of data items in the current page.	CDataProvider
keyAttribute	string	the name of key attribute for modelClass.	CActiveDataProvider
keys	array	Returns the key values associated with the data items.	CDataProvider
model	CActiveRecord	the AR finder instance (eg `Post::model()`).	CActiveDataProvider
modelClass	string	the primary ActiveRecord class name.	CActiveDataProvider
pagination	CPagination	Returns the pagination object.	CDataProvider
sort	CSort	Returns the sorting object.	CActiveDataProvider
totalItemCount	integer	Returns the total number of data items.	CDataProvider

Public Methods

Method	Description	Defined By
__call()	Calls the named method which is not a class method.	CComponent
__construct()	Constructor.	CActiveDataProvider
__get()	Returns a property value, an event handler list or a behavior based on its name.	CComponent
__isset()	Checks if a property value is null.	CComponent
__set()	Sets value of a component property.	CComponent
__unset()	Sets a component property to be null.	CComponent
asa()	Returns the named behavior object.	CComponent
attachBehavior()	Attaches a behavior to this component.	CComponent
attachBehaviors()	Attaches a list of behaviors to the component.	CComponent
attachEventHandler()	Attaches an event handler to an event.	CComponent
canGetProperty()	Determines whether a property can be read.	CComponent
canSetProperty()	Determines whether a property can be set.	CComponent
detachBehavior()	Detaches a behavior from the component.	CComponent
detachBehaviors()	Detaches all behaviors from the component.	CComponent
detachEventHandler()	Detaches an existing event handler.	CComponent
disableBehavior()	Disables an attached behavior.	CComponent
disableBehaviors()	Disables all behaviors attached to this component.	CComponent
enableBehavior()	Enables an attached behavior.	CComponent
enableBehaviors()	Enables all behaviors attached to this component.	CComponent
evaluateExpression()	Evaluates a PHP expression or callback under the context of this component.	CComponent
getCriteria()	Returns the query criteria.	CActiveDataProvider

getData()	Returns the data items currently available.	CDataProvider
getEventHandlers()	Returns the list of attached event handlers for an event.	CComponent
getId()	Returns the ID that uniquely identifies the data provider.	CDataProvider
getItemCount()	Returns the number of data items in the current page.	CDataProvider
getKeys()	Returns the key values associated with the data items.	CDataProvider
getPagination()	Returns the pagination object.	CDataProvider
getSort()	Returns the sorting object.	CActiveDataProvider
getTotalItemCount()	Returns the total number of data items.	CDataProvider
hasEvent()	Determines whether an event is defined.	CComponent
hasEventHandler()	Checks whether the named event has attached handlers.	CComponent
hasProperty()	Determines whether a property is defined.	CComponent
raiseEvent()	Raises an event.	CComponent
setCriteria()	Sets the query criteria.	CActiveDataProvider
setData()	Sets the data items for this provider.	CDataProvider
setId()	Sets the provider ID.	CDataProvider
setKeys()	Sets the data item keys for this provider.	CDataProvider
setPagination()	Sets the pagination for this data provider.	CDataProvider
setSort()	Sets the sorting for this data provider.	CDataProvider
setTotalItemCount()	Sets the total number of data items.	CDataProvider

Protected Methods

Hide inherited methods

Method	Description	Defined By
calculateTotalItemCount()	Calculates the total number of data items.	CActiveDataProvider
fetchData()	Fetches the data from the persistent data storage.	CActiveDataProvider
fetchKeys()	Fetches the data item keys from the persistent data storage.	CActiveDataProvider

کلاس CHtml

این کلاس یک کلاس استاتیک است یعنی مستقیما و بدون ساخت نمونه از روی آن استفاده می شود و جهت ساخت اجزای HTML در ویو به کار می رود.

علت استفاده :

```
<a href="/demo/index.php?r=message/goodbye">Goodbye!</a>
```

کد بالا را در نظر بگیرید. این کد با HTML معمولی نوشته شده است و یک Url ثابت را باز می کند. حال اگر به هر دلیلی مثل تغییر نحوه نمایش آدرس های سایت برای بهبود SEO یا هر علت دیگری این نحوه نمایش تغییر نماید دیگر این لینک باز نخواهد شد. لذا از کد زیر استفاده می کنیم :

```
<?php echo CHtml::link("Goodbye",array('message/goodbye'));
?>
```

این کد هم همان کار قبلی را انجام می دهد ولی با روش CHtml نوشته شده است. این کد در مسیر جاری سایت (هر چه باشد) کنترلر message و اکشن goodbye را بارگزاری می کند.

Public Properties

Property	Type	Description
afterRequiredLabel	string	the HTML code to be appended to the required label.
beforeRequiredLabel	string	the HTML code to be prepended to the required label.
count	integer	the counter for generating automatic input field names.
errorCss	string	the CSS class for highlighting error inputs.
errorMessageCss	string	the CSS class for displaying error messages (see error).
errorSummaryCss	string	the CSS class for displaying error summaries (see errorSummary).
requiredCss	string	the CSS class for required labels.

Public Properties

Property	Type	Description
afterRequiredLabel	string	the HTML code to be appended to the required label.
beforeRequiredLabel	string	the HTML code to be prepended to the required label.
count	integer	the counter for generating automatic input field names.
errorCss	string	the CSS class for highlighting error inputs.
errorMessageCss	string	the CSS class for displaying error messages (see error).
errorSummaryCss	string	the CSS class for displaying error summaries (see errorSummary).
requiredCss	string	the CSS class for required labels.

Public Methods

Method	Description
activeCheckBox()	Generates a check box for a model attribute.
activeCheckBoxList()	Generates a check box list for a model attribute.
activeDropDownList()	Generates a drop down list for a model attribute.
activeFileField()	Generates a file input for a model attribute.
activeHiddenField()	Generates a hidden input for a model attribute.
activeId()	Generates input field ID for a model attribute.
activeLabel()	Generates a label tag for a model attribute.

activeLabelEx()	Generates a label tag for a model attribute.
activeListBox()	Generates a list box for a model attribute.
activeName()	Generates input field name for a model attribute.
activePasswordField()	Generates a password field input for a model attribute.
activeRadioButton()	Generates a radio button for a model attribute.
activeRadioButtonList()	Generates a radio button list for a model attribute.
activeTextArea()	Generates a text area input for a model attribute.
activeTextField()	Generates a text field input for a model attribute.
ajax()	Generates the JavaScript that initiates an AJAX request.
ajaxButton()	Generates a push button that can initiate AJAX requests.
ajaxLink()	Generates a link that can initiate AJAX requests.
ajaxSubmitButton()	Generates a push button that can submit the current form in POST method.
asset()	Generates the URL for the published assets.
beginForm()	Generates an opening form tag.
button()	Generates a button.
cdata()	Encloses the given string within a CDATA tag.
checkBox()	Generates a check box.
checkBoxList()	Generates a check box list.
closeTag()	Generates a close HTML element.
css()	Encloses the given CSS content with a CSS tag.
cssFile()	Links to the specified CSS file.
dropDownList()	Generates a drop down list.
encode()	Encodes special characters into HTML entities.
encodeArray()	Encodes special characters in an array of strings into HTML entities.
endForm()	Generates a closing form tag.
error()	Displays the first validation error for a model attribute.
errorSummary()	Displays a summary of validation errors for one or several models.
fileField()	Generates a file input.
form()	Generates an opening form tag.
getActiveId()	Returns the element ID that is used by methods such as activeTextField.
getIdByName()	Generates a valid HTML ID based on name.
hiddenField()	Generates a hidden input.
htmlButton()	Generates a button using HTML button tag.
image()	Generates an image tag.
imageButton()	Generates an image submit button.
label()	Generates a label tag.
link()	Generates a hyperlink tag.
linkButton()	Generates a link submit button.
linkTag()	Generates a link tag that can be inserted in the head section of HTML page.
listBox()	Generates a list box.
listData()	Generates the data suitable for list-based HTML elements.
listOptions()	Generates the list options.
mailto()	Generates a mailto link.
metaTag()	Generates a meta tag that can be inserted in the head section of HTML page.
normalizeUrl()	Normalizes the input parameter to be a valid URL.

openTag()	Generates an open HTML element.
pageStateField()	Generates a hidden field for storing persistent page states.
passwordField()	Generates a password field input.
radioButton()	Generates a radio button.
radioButtonList()	Generates a radio button list.
refresh()	Registers a 'refresh' meta tag.
renderAttributes()	Renders the HTML tag attributes.
resetButton()	Generates a reset button.
resolveName()	Generates input name for a model attribute.
resolveNameID()	Generates input name and ID for a model attribute.
resolveValue()	Evaluates the attribute value of the model.
script()	Encloses the given JavaScript within a script tag.
scriptFile()	Includes a JavaScript file.
statefulForm()	Generates a stateful form tag.
submitButton()	Generates a submit button.
tag()	Generates an HTML element.
textArea()	Generates a text area input.
textField()	Generates a text field input.
value()	Evaluates the value of the specified attribute for the given model.

Protected Methods

Method	Description
activeInputField()	Generates an input HTML tag for a model attribute.
addErrorCss()	Appends errorCss to the 'class' attribute.
clientChange()	Generates the JavaScript with the specified client changes.
inputField()	Generates an input HTML tag.

Theming

در Yii هر theme نشانگر یک قالب و طرح خاص برای صفحه است به طوری که با تغییر یک theme کل ظاهر سایت متفاوت می شود. Theme ها در پوشه ای **WebRoot/themes** قرار دارند. هر پوشه Theme می تواند شامل کیله ویوها، طرح ها، CSS ها، فایلهای اسکریپت و ... در مورد Theme باشد.

نام هر Theme با نام پوشه اصلی آن یکسان است.

نحوه استفاده از یک Theme

برای این کار در فایل config مقدار خاصیت Theme را برابر نام Theme مورد نظر قرار می دهیم. این کار می تواند در ابتدای فایل config در زیر تعریف نام سایت قرار گیرد. مثال :

```
return array)
     'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR,'..'.
     'name' => 'My Web Site',
     'timezone' => 'Asia/Tehran,'
     'theme'=>'myTheme,'
```

نکته : اسم Theme باید با رعایت حروف کوچک و بزرگ وارد شود چون خاصیت Theme به کوچک و بزرگ بودن حروف حساس است.

به شکل زیر می توان Theme جاری را پیدا کرد که اگر هیچ Theme در نظر گرفته نشده باشد مقدار null را بر می گرداند.

```
Yii::app()->theme
```

ساخت یک Theme

باید تمامی View ها و Layout هایی که در پوشه Protected قرا دارند با همان نام و شکل منتقل شوند. اگر در برنامه View1 صدا زده شود ابتدا در Theme به دنبال آن می گردد و اگر در پوشه Theme وجود نداشت در مسیر Protected یعنی مسیر پیش فرض به دنبال آن می گردد.

نکته : در بسیاری از موارد ما نیاز داریم که به یک فایل یا پوشه درون پوشه Theme اشاره کنیم. مثلا اگر بخواهیم یک تصویر موجود در پوشه Theme را فراخوانی کنیم. برای انجام این کار به شکل زیر می توان آدرس دهی را انجام داد :

```
Yii::app()->theme->baseUrl . '/images/FileName.gif'
```

Yii::app()->theme->baseUrl مسیر پوشه مربوط به Theme جاری است.

پوشه ها و ساختار مربوط به دو Theme با نام basic و fancy به شکل زیر است :

```
WebRoot/
    assets
    protected/
        .htaccess
        components/
        controllers/
        models/
        views/
            layouts/
                main.php
            site/
                index.php
    themes/
        basic/
            views/
                .htaccess
                layouts/
                    main.php
                site/
                    index.php
        fancy/
            views/
                .htaccess
                layouts/
                    main.php
                site/
                    index.php
```

اگر در فایل تنظیمات داشته باشیم :

```
return array(
    'theme'=>'basic',
    ......
);
```

آنگاه Theme با نام basic فعال می شود که پوشه ها و فایلهای مربوطه آن در مسیر زیر قرار دارد :

```
themes/basic/views/layouts
```

اگر فایل مربوطه پیدا نشد آن را در protected/views جستجئ می کند.

Theme های ویگت

برای یک ویگت با نام Foo می توانیم یک ویو جداگانه طراحی نماییم که در قسمت توضیحات ویگت آمده است. اما اگر بخواهیم آن را در Theme خود قرار دهیم مثلا اگز فایلی با نام xyz.php مورد نیاز ویگت باشد در Theme مربوطه که basic نام دارد آن را در پوشه زیر قرار می دهیم :

```
themes/basic/views/Foo/xyz.php
```

Skin (پوسته)

هر ویگت می تواند دارای چندین ظاهر و شکل متفاوت باشد که به دلخواه می توان آن را تغییر داد. مثلا یک **CLinkPager** می تواند دارای چندین ظاهر متفاوت با رنگ و فونت و شکل متفاوت باشد که به هر کدام از آنها skin گفته می شود.

برای استفاده از skin ابتدا باید در فایل config این خاصیت را به شکل زیر فعال نماییم :

```
return array(
    'components'=>array(
        'widgetFactory'=>array(
            'enableSkin'=>true,
        ),
    ),
);
```

سپس فایلهای skin مورد نظر را در مسیر زیر قرار داد :

```
protected/views/skins
```

حال فایل CLinkPager.php را در مسیر protected/views/skins ایجاد می کنیم که محتوای آن به شکل زیر است :

```
<?php
return array(
    'default'=>array(
        'nextPageLabel'=>'&gt;&gt;',
        'prevPageLabel'=>'&lt;&lt;',
    ),
    'classic'=>array(
        'header'=>'',
        'maxButtonCount'=>5,
    ),
);
```

در مثال بالا دو skin ایجاد شده که با نام default و classic شناخته می شوند. حال برای استفاده از skin مورد نظر در تعریف ویگت خود داریم :

```
<?php $this->widget('CLinkPager'); ?>

<?php $this->widget('CLinkPager', array('skin'=>'classic'));
?>
```

بانک های اطلاعاتی

Yii از PDO که مخفف PHP Data Objects که یکی از سرویسهای بانک اطلاعاتی است استفاده می نماید این سرویسها باید در میزبان مانند Apache فعال گردند. لیست این سرویسها به شکل زیر می باشد :

PDO extension

PDO SQLite extension

PDO MySQL extension

PDO PostgreSQL extension

انواع روش کار با بانک اطلاعاتی :

در yii به سه روش امکان کار با بانک اطلاعاتی وجود دارد :

۱- DAO :

Yii Data Access Objects که مخفف آن DAO است امکان دسترسی به انواع سیستم مدیریت بانک اطلاعاتی یا DBMS را فراهم می کند.

۲- Query Builder :

Query Builder سرویسی شی گرا برای دسترسی به پرس و جو های SQL است که از نظر امنیتی باعث کاهش خطرات حمله از نوع SQL injection می شوند.

۳- AR :

Active Record و یا AR از Object-Relational Mapping و یا ORM استفاده می کند که برنامه نویسی بانک اطلاعاتی را بدون نیاز به پرس و جو های SQL به صورت بسیار ساده در می آورد. در این مدل یک جدول در قالب یک کلاس و یک رکورد آن نمونه ای از آن کلاس معرفی می شود.

اتصال به بانک اطلاعاتی :

از آنجایی که اولین فدم در کار با بانک اطلاعاتی ایجاد یک ارتباط است در این قسمت نحوه اتصال را بررسی می کنیم که مورد نیاز در تمام روشها است. برای برقراری ارتباط با بانک اطلاعاتی باید در فایل config تنظیمات مربوطه را انجام داد. قالب کلی این ارتباط به شکل زیر تعریف می شود :

```
$connection=new CDbConnection($dsn,$username,$password);
// establish connection. You may try...catch possible
exceptions
$connection->active=true;
......
$connection->active=false;  // close connection
```

نوع DSN به PDO database driver بستگی دارد که می تواند یکی از مقادیر زیر باشد :

- SQLite: `sqlite:/path/to/dbfile`
- MySQL: `mysql:host=localhost;dbname=testdb`
- PostgreSQL: `pgsql:host=localhost;port=5432;dbname=testdb`
- SQL Server: `mssql:host=localhost;dbname=testdb`
- Oracle: `oci:dbname=//localhost:1521/testdb`

یک نمونه از آن به شکل زیر است :

```
'db'=>array(
     'connectionString' => 'mysql:host=localhost;dbname=db1,
     'emulatePrepare' => true,
     'username' => 'root',
     'password' => '',
     'charset' => 'utf8',
     'tablePrefix' => 'tbl_',
     ),
```

نکته : تنها بانک های اطلاعاتی زیر پشتیبانی می شوند :

- MySQL 4.1 or later
- PostgreSQL 7.3 or later
- SQLite 2 and 3

- Microsoft SQL Server 2000 or later
- Oracle

نکته : در هر کجای برنامه می توان از طریق Yii::app()->db به DB connection دسترسی پیدا کرد.

Data Access Objects (DAO)

DAO یک روش دسترسی به بانک اطلاعاتی با استفاده از generic API می باشد. که قابلیت کار با انواع DBMS را دارد. DAO از چهار کلاس اصلی زیر تشکیل شده است :

- CDbConnection: represents a connection to a database.
- CDbCommand: represents an SQL statement to execute against a database.
- CDbDataReader: represents a forward-only stream of rows from a query result set.
- CDbTransaction: represents a DB transaction.

اجرای دستورات SQL

کلیه دستورات کار با داده ها از طریق دستورات sql به طور مستقیم مورد استفاده قرار می گیرد. دستورات توسط کلاس CDbCommand اجرا می شوند. نحوه تعریف یک دستور به شکل زیر است :

```
$command=$connection->createCommand($sql);
```

نحوه اجرای دستورات نیز به شکل زیر است :

```
$rowCount=$command->execute();   // execute the non-query SQL

$dataReader=$command->query();   // execute a query SQL

$rows=$command->queryAll();      // query and return all rows
                                                of result

$row=$command->queryRow();       // query and return the first
                                                row of result

$column=$command->queryColumn(); // query and return the first
                                                column of result
```

```
$value=$command->queryScalar();  // query and return the first
                                    field in the first row
```

نکته : از دستورات فوق در روش Query Builder نیز استفاده می شود.

فراخوانی نتایج

پس از اجرای یک دستور SQL نحوه دریافت نتایج با استفاده از کلاس CDbDataReader به شکل زیر است :

```
$dataReader=$command->query();
// calling read() repeatedly until it returns false
while(($row=$dataReader->read())!==false) { ... }
// using foreach to traverse through every row of data
foreach($dataReader as $row) { ... }
// retrieving all rows at once in a single array
$rows=$dataReader->readAll();
```

نحوه کار با تراکنش ها

تراکنش اجرای یک query است که می تواند شامل چندید دستور sql باشد. تراکنش مرحله به مرحله اجرا می شود و تا انتها اگر همه مراحل به درستی اجرا شد اعمال می شود واگر حتی یک دستور با خطا مواجه شود کل تراکنش لغو می شود. اجرای تراکنش ها توسط کلاس CDbTransaction انجام می شود. یک تراکنش به شکل زیر عمل می کند :

- تراکنش را شروع می کند
- دستورات sql را یکی یکی و پشت سر هم اجرا می کند. هیچ نتیجه ای از انجام هر کدام از این مراحل قابل مشاهده نیست.
- اگر تراکنش تا انتها به درستی انجام شود تنظیمات را که شامل ایجاد/ به روز رسانی و .. است به طور یکجا اجرا می کند و نتیجه نهایی قابل ملاحظه است.
- اگر حتی یکی از دستورات query با خطا مواجه شود کل تراکنش لغو خواهد شد.

مثال :

```
$transaction=$connection->beginTransaction();
```

```
try
{
$connection->createCommand($sql1)->execute();
$connection->createCommand($sql2)->execute();
//.... other SQL executions
$transaction->commit();
}
catch(Exception $e) // an exception is raised if a query fails
{
$transaction->rollBack();
}
```

انقیاد پارامتر ها :

برای اینکه از حملات SQL injection جلوگیری شود و همچنین برای افزایش کارایی اجرای دستورات Sql می توانید از پارامترهای کمکی که به آنها جا نگهدار یا placeholders هم گفته می شود به جای پارامتر های اصلی استفاده نمایید. مثال :

```
// an SQL with two placeholders ":username" and ":email"

$sql="INSERT INTO tbl_user (username, email)
VALUES(:username,:email)";
$command=$connection->createCommand($sql);

// replace the placeholder ":username" with the actual
username value

$command->bindParam(":username",$username,PDO::PARAM_STR);

// replace the placeholder ":email" with the actual email
value

$command->bindParam(":email",$email,PDO::PARAM_STR);
$command->execute();

// insert another row with a new set of parameters
$command->bindParam(":username",$username2,PDO::PARAM_STR);
$command->bindParam(":email",$email2,PDO::PARAM_STR);
$command->execute();
```

انقیاد فیلدها :

```
$sql="SELECT username, email FROM tbl_user";
$dataReader=$connection->createCommand($sql)->query();
// bind the 1st column (username) with the $username variable
$dataReader->bindColumn(1,$username);
// bind the 2nd column (email) with the $email variable
$dataReader->bindColumn(2,$email);
while($dataReader->read()!==false)
{
// $username and $email contain the username and email in
the current row
}
```

استفاده از Table Prefix

می توان برای جداول یک بانک اطلاعاتی یک پیش نام مشخص کرد و در ادامه کار بدون نیاز به رجوع به پیش نام از جدول استفاده نمود. در ابزار gii در هنگام ساخت مدل می توان پیش نام جداول مثلا _tbl را مشخص نمود.

```
$sql='SELECT * FROM {{user}}';
$users=$connection->createCommand($sql)->queryAll();
```

Query Builder

این روش یک روش شی گرا برای دسترسی به عناصر بانک اطلاعاتی می باشد. به عنوان مثال یک پرس و جوی SQL در این روش به شکل زیر نوشته می شود :

```
$user = Yii::app()->db->createCommand()
->select('id, username, profile')
->from('tbl_user u')
->join('tbl_profile p', 'u.id=p.user_id')
->where('id=:id', array(':id'=>$id))
->queryRow();
```

مزایای روش Query Builder

- پیاده سازی پرس و جو های پیچیده
- خطر حمله نوع SQL injection را کاهش می دهد
- نام جداول و فیلدها را به گونه ای مشخص می کند که امکان تداخل نامها وجود نداشته باشد.
- جهت انتقال بانک اطلاعاتی و تغییر سیستم تطابق وجود دارد و مشکلی ایجاد نمی کند.

ایجاد دستورات :

```
$command = Yii::app()->db->createCommand();
```

ایجاد پرس و جو ها :

Select

```
// SELECT *
select()
// SELECT `id`, `username`
select('id, username')
// SELECT `tbl_user`.`id`, `username` AS `name`
select('tbl_user.id, username as name')
// SELECT `id`, `username`
select(array('id', 'username'))
// SELECT `id`, count(*) as num
select(array('id', 'count(*) as num'))
```

selectDistinct

```
SELECT DISTINCT `id`, `username`
```

From

```
// FROM `tbl_user`
from('tbl_user')
```

```
            // FROM `tbl_user` `u`, `public`.`tbl_profile` `p`
                    from('tbl_user u, public.tbl_profile p')
                           // FROM `tbl_user`, `tbl_profile`
                      from(array('tbl_user', 'tbl_profile'))
             // FROM `tbl_user`, (select * from tbl_profile) p
      from(array('tbl_user', '(select * from tbl_profile) p'))
```

where

```
                                        // WHERE id=1 or id=2
                                        where('id=1 or id=2')
                                  // WHERE id=:id1 or id=:id2
     where('id=:id1 or id=:id2', array(':id1'=>1, ':id2'=>2))
                                        // WHERE id=1 OR id=2
                            where(array('or', 'id=1', 'id=2'))
                           // WHERE id=1 AND (type=2 OR type=3)
  where(array('and', 'id=1', array('or', 'type=2', 'type=3')))
                                     // WHERE `id` IN (1, 2)
                           where(array('in', 'id', array(1, 2))
                                     // WHERE `id` NOT IN (1, 2)
                       where(array('not in', 'id', array(1,2)))
                                // WHERE `name` LIKE '%Qiang%'
                         where(array('like', 'name', '%Qiang%'))
          // WHERE `name` LIKE '%Qiang' AND `name` LIKE '%Xue'
         where(array('like', 'name', array('%Qiang', '%Xue')))
           // WHERE `name` LIKE '%Qiang' OR `name` LIKE '%Xue'
      where(array('or like', 'name', array('%Qiang', '%Xue')))
                           // WHERE `name` NOT LIKE '%Qiang%'
                    where(array('not like', 'name', '%Qiang%'))
  // WHERE `name` NOT LIKE '%Qiang%' OR `name` NOT LIKE '%Xue%'
where(array('or not like', 'name', array('%Qiang%', '%Xue%')))
```

order

```
                                  // ORDER BY `name`, `id` DESC
                                        order('name, id desc')
                        // ORDER BY `tbl_profile`.`name`, `id` DESC
                         order(array('tbl_profile.name', 'id desc')
```

limit and offset

```
                                                  // LIMIT 10
                                                    limit(10)
```

```
// LIMIT 10 OFFSET 20
limit(10, 20)
// OFFSET 20
offset(20)
```

join and its variants

```
// JOIN `tbl_profile` ON user_id=id
join('tbl_profile', 'user_id=id')
// LEFT JOIN `pub`.`tbl_profile` `p` ON p.user_id=id AND
type=:type
leftJoin('pub.tbl_profile p', 'p.user_id=id AND type=:type',
array(':type'=>1))
```

group

```
// GROUP BY `name`, `id`
group('name, id')
// GROUP BY `tbl_profile`.`name`, `id`
group(array('tbl_profile.name', 'id')
```

having

```
// HAVING id=1 or id=2
having('id=1 or id=2')
// HAVING id=1 OR id=2
having(array('or', 'id=1', 'id=2'))
```

union

```
// UNION (select * from tbl_profile)
union('select * from tbl_profile')
```

اجرای پرس و جو ها

پس از نوشتن پرس و جوی مورد نظر به روشی که در DAO برای اجرای پرس و جو ها اشاره شد می توان پرس و جو را اجرا نمود. مثال :

```
$users = Yii::app()->db->createCommand()
->select('*')
->from('tbl_user')
```

```
->queryAll();
```

به دست آوردن عبارت SQL ایجاد شده :

توسط متد text می توان عبارت Sql ایجاد شده را به دست آورد. مثال :

```
$sql = Yii::app()->db->createCommand()
->select('*')
->from('tbl_user')
->text;
```

به کار بردن دستورات جایگزین :

می توان بعضی از دستورات را برای تطابق با شی گرایی به شکل دیگری نیز نوشت. مثال :

```
$command->select(array('id', 'username'));
$command->select = array('id', 'username');
```

و یا مثالی برای ایجاد یک پس و جو :

```
$row = Yii::app()->db->createCommand(array(
'select' => array('id', 'username'),
'from' => 'tbl_user',
'where' => 'id=:id',
'params' => array(':id'=>1),
))->queryRow();
```

کار با چند پرس و جو :

می توان بر روی چند پرس و جو با یک مدخل مثل $command کار کرد. و در هر جایی با ()reset نتایج قبلی را پاک کرد. مثال:

```
$command = Yii::app()->createCommand();
$users = $command->select('*')->from('tbl_users')->queryAll();
$command->reset();  // clean up the previous query
$posts = $command->select('*')->from('tbl_posts')->queryAll();
```

پرس و جو های کار با داده :

Insert

```
                    // build and execute the following SQL:
// INSERT INTO `tbl_user` (`name`, `email`) VALUES (:name,
                                                    :email)
                        $command->insert('tbl_user', array(
                                                'name'=>'Tester',
                                'email'=>'tester@example.com',
                                                ));
```

Update

```
                    // build and execute the following SQL:
          // UPDATE `tbl_user` SET `name`=:name WHERE id=:id
                        $command->update('tbl_user', array(
                                                'name'=>'Tester',
                                ), 'id=:id', array(':id'=>1));
```

Delete

```
                    // build and execute the following SQL:
                     // DELETE FROM `tbl_user` WHERE id=:id
      $command->delete('tbl_user', 'id=:id', array(':id'=>1));
```

پرس و جو های کار با ساختار :

نوع داده های مورد استفاده در Query Builder :

- **pk**: a generic primary key type, will be converted into **int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY** for MySQL;
- **string**: string type, will be converted into **varchar(255)** for MySQL;
- **text**: text type (long string), will be converted into **text** for MySQL;
- **integer**: integer type, will be converted into **int(11)** for MySQL;
- **float**: floating number type, will be converted into **float** for MySQL;
- **decimal**: decimal number type, will be converted into **decimal** for MySQL;
- **datetime**: datetime type, will be converted into **datetime** for MySQL;
- **timestamp**: timestamp type, will be converted into **timestamp** for MySQL;
- **time**: time type, will be converted into **time** for MySQL;

- **`date`**: date type, will be converted into **`date`** for MySQL;
- **`binary`**: binary data type, will be converted into **`blob`** for MySQL;
- **`boolean`**: boolean type, will be converted into **`tinyint(1)`** for MySQL.

createTable

```
// CREATE TABLE `tbl_user` (
//     `id` int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
//     `username` varchar(255) NOT NULL,
//     `location` point
// ) ENGINE=InnoDB
createTable('tbl_user', array(
    'id' => 'pk',
    'username' => 'string NOT NULL',
    'location' => 'point',
), 'ENGINE=InnoDB')
```

renameTable

```
// RENAME TABLE `tbl_users` TO `tbl_user`
renameTable('tbl_users', 'tbl_user')
```

dropTable

```
// DROP TABLE `tbl_user`
dropTable('tbl_user')
```

truncateTable

```
// TRUNCATE TABLE `tbl_user`
truncateTable('tbl_user')
```

addColumn

```
// ALTER TABLE `tbl_user` ADD `email` varchar(255) NOT NULL
addColumn('tbl_user', 'email', 'string NOT NULL')
```

dropColumn

```
// ALTER TABLE `tbl_user` DROP COLUMN `location`
dropColumn('tbl_user', 'location')
```

renameColumn

```
            // ALTER TABLE `tbl_users` CHANGE `name` `username`
                                              varchar(255) NOT NULL
                    renameColumn('tbl_user', 'name', 'username')
```

alterColumn

```
          // ALTER TABLE `tbl_user` CHANGE `username` `username`
                                              varchar(255) NOT NULL
          alterColumn('tbl_user', 'username', 'string NOT NULL')
```

addForeignKey

```
                    // ALTER TABLE `tbl_profile` ADD CONSTRAINT
                                                `fk_profile_user_id`
        // FOREIGN KEY (`user_id`) REFERENCES `tbl_user` (`id`)
                             // ON DELETE CASCADE ON UPDATE CASCADE
addForeignKey('fk_profile_user_id', 'tbl_profile', 'user_id',
                          'tbl_user', 'id', 'CASCADE', 'CASCADE')
```

dropForeignKey

```
                  // ALTER TABLE `tbl_profile` DROP FOREIGN KEY
                                                `fk_profile_user_id`
             dropForeignKey('fk_profile_user_id', 'tbl_profile')
```

createIndex

```
      // CREATE INDEX `idx_username` ON `tbl_user` (`username`)
                          createIndex('idx_username', 'tbl_user')
```

dropIndex

```
                       // DROP INDEX `idx_username` ON `tbl_user`
                            dropIndex('idx_username', 'tbl_user')
```

Active Record

AR روشی برای کار با بانک اطاعاتی است که در آن نیازی به استفاده از کدهای SQL نمی باشد و کاملا مبتنی بر شی گرایی است. کلیه کدها داخل اکشن کنترلر نوشته می شوند و تنظیمات در مدل انجام می شود. مثلا فراخوانی یک مدل ساخته شده در کنترلر می تواند به شکل زیر باشد :

```
$models = Enquiry::model()->findAllByAttributes(
                                                array(),
"enquiry_status < 5 AND enquiry_date >=:enquiry_date",
                array('enquiry_date'=>date('Y-m-d H:i:s')))
                                                );
```

۱- اتصال به بانک اطلاعاتی : طبق توضیح قبل

۲- تعریف AR Class

برای ادرمه کار ما نیاز به یک مدل از نوع AR داریم هر کلای AR معرف یک جدول از بانک اطلاعتی می باشد و هر شی یا نمونه ساخته شده از روی این کلاس معرف یک رکورد از بانک اطلاعاتی است. کدهای زیر حداقل کد مورد نیاز برای ساخت کلاس مدل tbl_post می باشد :

```
class Post extends CActiveRecord
                                                {
public static function model($className=__CLASS__)
                                                {
return parent::model($className);
                                                }

public function tableName()
                                                {
return 'tbl_post';
                                                }
                                                }
```

از آنجایی که تمامی مدل های ما در چندین فایل جداگانه تعریف می شوند می توان در فایل config به شکل زیر دایرکتوری protected/modelsرا import کرد تا دسترسی به فایلهای آن به صورت یکجا و راحت باشد :

```
return array(
'import'=>array(
'application.models.*',
```

```
),
);
```

به طور پیش فرض نام کلاس مدل با نام جدول بانک اطلاعاتی یکسان است.

برای تعریف نام جدول با استفاده از prefix در کلاس مدل به شکل زیر عمل می کنیم :

```
public function tableName()
{
return '{{post}}';
}
```

ایم کد باعث تعریف جدولی با نام post می شود که پیش نام آن هر چه باشد لحاظ می شود مثلا اگر پیش نام آن _tbl تعریف شده باشد نام اصلی tbl_post تعریف می شود و در آینده می توان پیش نام جدول را تغییر داد بدون نیاز به تغییر کدهای کلاس مدل.

هر جدولی که با آن کار می کنیم باید به طور پیش فرض دارای یک کلید اصلی باشد و اگر اینطور نیست باید از طریق متد زیر یک فیلد را به عنوان کلید اصلی تعریف نماییم. نکته : اگر جدول از قبل دارای کلید اصلی باشد نیازی به این کد نیست :

```
public function primaryKey()
{
return 'id';
// For composite primary key, return an array like the
following
// return array('pk1', 'pk2');
}
```

۳- ساخت رکورها

برای هر کاری ابتدا باید یک نمونه از کلاس مدل ساخته شود که این نمونه معادل یک رکورد است این کار به شکل زیر انجام می شود :

```
$post=new Post;
```

کد فوق یک شی با نام $post ایجاد می کند که از روی کلاس Post ایجاد می شود. Post نام کلاس مدل است که با نام جدول یعنی جدول Post یکسان است.

نحوه ساخت یک رکورد جدید ، مقدار دهی به فیلد های آن و ذخیره نهایی :

```
$post=new Post;
$post->title='sample post';
```

```
$post->content='content for the sample post';
$post->create_time=time();
$post->save();
```

نکته : برای کلید اصلی که به صورت خودکار مقدار آن اضافه می شود نیازی به نوشتن آن فیلد نیست و پس از ایجاد هر رکورد مقدار فیلد کلید اصلی به طور خودکار اضافه می شود.

دسترسی به مقدار یک فیلد از رکورد :

برای دسترسی به مقدار یک فیلد از رکورد به شکل زیر می توان عمل کرد :

```
echo $post->title;
```

این مثال باعث می شود که مقدار موجود در فیلد title از رکورد جاری چاپ شود. همچنین برای مقدار دهی آن به شکل زیر می توان عمل کرد.

```
$post->title = 'hello';
```

برای پر کردن رکورد جاری با مقادیر وارد شده توسط فرم نیز به شکل زیر عمل می کنیم :

```
// assume $_POST['Post'] is an array of column values indexed
by column names
$post->attributes=$_POST['Post'];
```

استفاده از توابع هاست :

هر هاست مانند MySQL دارای یک سری توابع پیش فرض هستند که می توان از آنها استفاده نمود به عنوان مثال برای استفاده از تابع درونی MySQL NOW() موجود در MySQL در مقدار دهی یک فیلد با استفاده از تابع CDbExpressionبه شکل زیر عمل می کنیم :

```
$post=new Post;
$post->create_time=new CDbExpression('NOW()');
// $post->create_time='NOW()'; will not work because
// 'NOW()' will be treated as a string
$post->save();
```

نکته : بسیاری از مواقع نیاز داریم که بدانیم چه دستوری در برنامه اجرا شد که این کار را می توان به راحتی با استفاده از فعال کردن خاصیت logging در برنامه انجام داد و به فایل log مراجعه نمود.

۴- خواندن رکورد ها

برای خواندن رکوردهای یک جدول باید از یکی از متدهای جستجوس زیر استفاده کرد :

```
// find the first row satisfying the specified condition
$post=Post::model()->find($condition,$params);

// find the row with the specified primary key
$post=Post::model()->findByPk($postID,$condition,$params);

// find the row with the specified attribute values
$post=Post::model()->findByAttributes($attributes,$condition,$params);

// find the first row using the specified SQL statement
$post=Post::model()->findBySql($sql,$params);
```

مثال :

```
public function actionX()
{

$model = Website::model()->findbyPk(3);

$this->renderText($model->title);

}
```

نکته : متد model به صورت استاتیک است و در تمامی جستجوها ضروری است. برای هر کلاس به شکل $MyModel::model()-> شروع جستجو است. این جستجو یک رکورد را بر می گرداند که به صورت شی گرا می توان به آن دسترسی پیدا کرد.

پس از اینکه توسط یک تابع جستجو رکورد مورد نظر پیدا شد به شکل زیر می توان به فیلدهای آن به راحتی دسترسی پیدا کرد :

```
echo $post->title;
```

متد find در صورتی که هیچ رکوردی را پیدا نکند ومقدار null را بر می گرداند.

پارامتر $condition

این پارامتر رشته ای است که معادل دستور WHERE در SQL عمل می کند

پارامتر $params

این پارامتر شامل آرایه از پارامترها است که مقادیر آنها به جانگهدارهای تعریف شده در رشته $condition اشاره می کنند. مثال :

```
// find the row with postID=10
$post=Post::model()->find('postID=:postID',
array(':postID'=>10));
```

همچنین می توان از پارامتر $condition برای ساخت پرس و جو های پیشرفته تر استفاده نمود. این پارامتر می تواند نمونه ای از کلاس CdbCriteria باشد که این کار به ما اجازه می دهد پرس و جو هایی بنویسیم که بیش از دستور WHERE توانایی دارند. مثال :

```
$criteria=new CDbCriteria;
$criteria->select='title';  // only select the 'title' column
$criteria->condition='postID=:postID';
$criteria->params=array(':postID'=>10);
$post=Post::model()->find($criteria); // $params is not needed
```

نکته : وقتی از روش بالا استفاده می شود نیازی به استفاده از پارامتر $condition نمی باشد زیرا که این پارامتر در تعریف نمونه CdbCriteria استفاده و تعریف می شود.

کد بالا را می توان به شکل زیر ساده تر نوشت که با قبلی یکی است :

```
$post=Post::model()->find(array(
'select'=>'title',
'condition'=>'postID=:postID',
'params'=>array(':postID'=>10),
));
```

وقتی که بخواهیم جستجوی ما بیش از یک نتیجه در بر داشته باشد (البته در صورت وجود) باید از متدهای زیر به جای متدهای بالا استفاده کرد که تنها یک رکورد را بر می گرداندند :

```
// find all rows satisfying the specified condition
$posts=Post::model()->findAll($condition,$params);

// find all rows with the specified primary keys
$posts=Post::model()-
>findAllByPk($postIDs,$condition,$params);
```

```
// find all rows with the specified attribute values
$posts=Post::model()->findAllByAttributes($attributes,$condition,$params);

// find all rows using the specified SQL statement
$posts=Post::model()->findAllBySql($sql,$params);
```

نکته : در استفاده از این متدها اگر هیچ نتیجه (رکوردی) یافت نشود یک آرایه خالی بر گردانده می شود بر عکس متدهای قبلی که null را بر می گرداندند.

علاوه بر متدهای فوق چند متد کمکی زیر نیز می تواند استفاده شود :

```
// get the number of rows satisfying the specified condition
$n=Post::model()->count($condition,$params);
// get the number of rows using the specified SQL statement
$n=Post::model()->countBySql($sql,$params);
// check if there is at least a row satisfying the specified condition
$exists=Post::model()->exists($condition,$params);
```

۵- به روز رسانی رکوردها :

پس از ساخت یک نمونه از کلاس که معرف یک رکورد است می توان با استفاده از متد Save عملیات به روز رسانی را انجام داد مثال :

```
$post=Post::model()->findByPk(10);
$post->title='new post title';
$post->save(); // save the change to database
```

اگر یک رکورد جدید با کلمه کلیدی new ایجاد شده باشد متد save باعث ذخیره شدن آن رکورد می شود و اگر این رکورد توسط یکی از متدهای find فراخوانی شده باشد متد save باعث ذخیره آن می شود.

روش دیگری نیز برای به روز رسانی رکوردها بدون فراخوانی آنها وجود دارد که به شکل زیر است :

```
// update the rows matching the specified condition
Post::model()->updateAll($attributes,$condition,$params);
// update the rows matching the specified condition and primary key(s)
```

```
Post::model()->updateByPk($pk,$attributes,$condition,$params);
// update counter columns in the rows satisfying the specified conditions
  Post::model()->updateCounters($counters,$condition,$params);
```

$attributes آرایه ای از مقادیر فیلد ها است.

$counters آرایه ای از مقادیر که مقدارشان به طور خودکار اضافه می شود.

سایر پارامتر ها قبلا توضیح داده شده است.

۶- حذف رکورد ها :

همانند روش به روز رسانی این عملیات نیز به دو روش قابل انجام است.

روش اول :

```
$post=Post::model()->findByPk(10); // assuming there is a post whose ID is 10
    $post->delete(); // delete the row from the database table
```

روش دوم :

```
            // delete the rows matching the specified condition
                  Post::model()->deleteAll($condition,$params);
       // delete the rows matching the specified condition and primary key(s)
             Post::model()->deleteByPk($pk,$condition,$params);
```

۷- اعتبار سنجی داده ها :

در بسیاری از موارد قبل از انجام یک عملیات می خواهیم ببینیم که آیا اعتبارسنجی مناسب در مورد داده ها انجام گرفته ایت یا خیر. AR به طور خودکار هنگام اجرای متد save متد rules را در مدل برای اعتبار سنجی اجرا می کند و می توانیم در کنترلر اجرای متد را به شکل زیر مدیریت کنیم که باعث می شود در صورت بروز خطا به کنترل برنامه دسترسی داشته باشیم :

```
                                        if($post->save())
                                        {
        // data is valid and is successfully inserted/updated
                                        }
                                      else
                                        {
```

```
        // data is invalid. call getErrors() to retrieve error
                                                        messages
                                                              }
```

۸- چند متد مفید در کلاس مدل :

از متدهای زیر می توان در کلاس مدل استفاده نمود :

- beforeValidate and afterValidate: these are invoked before and after validation is performed.
- beforeSave and afterSave: these are invoked before and after saving an AR instance.
- beforeDelete and afterDelete: these are invoked before and after an AR instance is deleted.
- afterConstruct: this is invoked for every AR instance created using the **new** operator.
- beforeFind: this is invoked before an AR finder is used to perform a query (e.g. **find()**, **findAll()**). This has been available since version 1.0.9.
- afterFind: this is invoked after every AR instance created as a result of query.

۹- به کارگیری تراکنش ها در AR

هر نمونه ساخته شده از روی AR دارای یک property با نام dbConnection می باشد که نمونه ساخته شده ای از روی کلا CDbConnection است . ما می توانیم برای کار با تراکنش ها از transaction در صورت نیاز استفاده کنیم. مثال :

```
                                                $model=Post::model();
        $transaction=$model->dbConnection->beginTransaction();
                                                                try
                                                                  {
    // find and save are two steps which may be intervened by
                                                      another request
      // we therefore use a transaction to ensure consistency
                                                      and integrity
                                          $post=$model->findByPk(10);
                                  $post->title='new post title';
                                                  $post->save();
                                        $transaction->commit();
                                                                  }
                                             catch(Exception $e)
                                                                  {
                                      $transaction->rollBack();
                                                                  }
```

Named Scopes – ۱۰

هر Named Scope نشانگر یک پرس و جوی دارای نام است که می تواند با یک پرس وجوی دیگر ترکیب شده و در یک پرس و جوی AR عمل کند.

این ایده از Ruby On Rails گرفته شده است. هر *named scope معرف یک پرس و جوی SQL است.*

مثال :

```
class Post extends CActiveRecord
{
    ......
    public function scopes()
    {
        return array(
            'published'=>array(
                'condition'=>'status=1',
            ),
            'recently'=>array(
                'order'=>'create_time DESC',
                'limit'=>5,
            ),
        );
    }
}
```

حال می توان به نتیجه پرس و جو به شکل زیر دسترسی پیدا کرد :

```
$posts=Post::model()->published()->recently()->findAll();
```

و همچنین :

```
Post::model()->published()->recently()->delete();
```

همچنین می توان از روش ارسال پارامترها استفاده کرد. مثال :

```
public function recently($limit=5)
{
    $this->getDbCriteria()->mergeWith(array(
        'order'=>'create_time DESC',
        'limit'=>$limit,
    ));
    return $this;
}
```

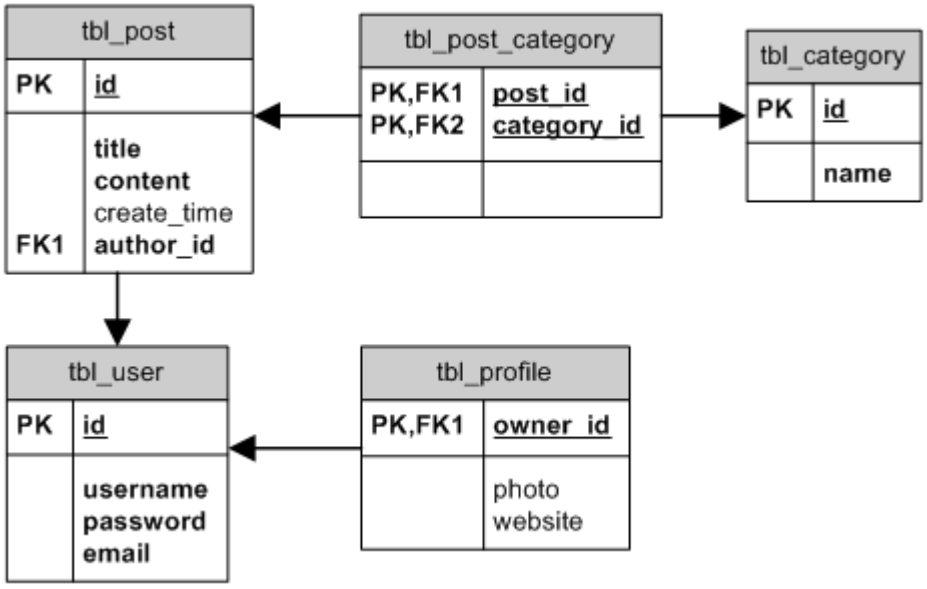
نحوه فراخوانی :

```
$posts=Post::model()->published()->recently(3)->findAll();
```

Active Record رابطه ای :

در مطالب قبلی نحوه دریافت اطلاعات از یک مدل AR را بررسی کردیم در این قسمت به نحوه بازیابی اطلاعات از یک بانک اطلاعاتی رابطه ای خواهیم پرداخت.

برای بررسی این مطلب مثال زیر را در نظر بگیرید :



انواع رابطه :

۱-۱ : مثل **tbl_user** و **tbl_profile**

1-Many : مثل **tbl_post** و **tbl_user**

Many – Many : مثل **tbl_post** و **tbl_category**

انواع حالت تعریف رابطه در مدل :

BELONGS_TO : اگر رابطه بین جدول A و B از نوع ۱-چند باشد می توانیم بگوییم : B belongs to A مثال **Post** belongs to **User**

HAS_MANY : اگر رابطه بین جدول A و B از نوع ۱-چند باشد می توانیم بگوییم : A has many B مثال **User** has many **Post**

HAS_ONE : این یک حالت خاص از HAS_MANY است که در آن A حداکثر شامل یک B باشد. مثال **User** has at most one **Profile**

MANY_MANY : این حالت مربوط به رابطه چند-چند است. اکثر بانک های اطلاعاتی رابطه چند-چند را پشتیبانی نمی کنند و باید آن را به دو رابطه چند-۱ و ۱-چند تقسیم کرد در مثال فوق جدول tbl_post_category همین کار را برای ما انجام می دهد. مثال **Category** has many **Post** و **Post** belongs to many **Category**

نحوه تعریف یک رابطه در مدل :

تعریف یک رابطه در یک مدل دارای الگوی زیر است :

```
'VarName'=>array('RelationType', 'ClassName', 'ForeignKey',
...additional options)
```

که در آن VarName نام رابطه است و **RelationType نوع رابطه است که در قسمت قبلی توضیح آن داده شد و می تواند یکی از مقادیر زیر باشد :**

self::BELONGS_TO, **self::HAS_ONE**, **self::HAS_MANY** and **self::MANY_MANY**;

ClassName نام کلاس AR در رابطه با این کلاس AR است.

ForeignKey نام کلید یا کلیدهای خارجی در رابطه است.

به عنوان مثال رابطه بین جداول User و Post به شکل زیر تعریف می شود : برای Post

```
class Post extends CActiveRecord
{
    ......

    public function relations()
    {
        return array(
            'author'=>array(self::BELONGS_TO, 'User',
'author_id'),
            'categories'=>array(self::MANY_MANY, 'Category',
                'tbl_post_category(post_id, category_id)'),
        );
    }
}
```

و برای User

```
class User extends CActiveRecord
{
    ......

    public function relations()
    {
        return array(
            'posts'=>array(self::HAS_MANY, 'Post',
'author_id'),
            'profile'=>array(self::HAS_ONE, 'Profile',
'owner_id'),
        );
    }
}
```

نکته : یک کلید خارجی ممکن است ترکیبی باشد یعنی متشکل از دو یا چند فیلد باشد در این صورت باید آن فیلد ها را با کاما از هم جدا کرد به عنوان مثال در رابطه چند-چند بالا بین جداول **categories** و **post** **کلید به صورت زیر مشخص می شود:**

```
tbl_post_category(post_id, category_id).
```

اجرای پرس و جو های رابطه ای :

روش Lazy Loading :

این روش را روش بارگزاری تنبل می گویند.

```
// retrieve the post whose ID is 10
$post=Post::model()->findByPk(10);
// retrieve the post's author: a relational query will be
performed here
$author=$post->author;
```

نکته : اگر در این رابطه نوشته شده هیچ مقداری موجود نباشد یعنی رابطه به جایی اشاره کند که مقداری برای آن وجود ندارد یک مقدار null و یا یک مقدار empty برگردانده می شود. برای **BELONGS_TO** و **HAS_ONE** **مقدار null برگردانده می شود. برای HAS_MANY** و **MANY_MANY** مقدار empty برگردانده می شود. دقت کنید که **HAS_MANY** و **MANY_MANY** **یک آرایه ای از اشیاء را بر می گردانند بنابر این در این رابطه ها ممکن است با اشکال** "Trying to get property of non-object" بر خورد نمایید.

مشکل روش .Lazy Loading در این است که عملیات بازیابی تنها برای یک رکورد انجام می گیرد و اگر بخواهیم بر روی چند رکورد عملیات همزمان انجام شود باید چندین پرس و جوی جدا بنویسیم.

روش Eager Loading :

این روش را روش بارگزاری مشتاقانه می گویند.

مثال ۱: (این کار در روش بارگزاری تنبل هم امکان پذیر است)

```
$posts=Post::model()->with('author')->findByPk(1);
```

مثال۲:

```
$posts=Post::model()->with('author')->findAll();
```

نکته : در روش تنبل نمی توان چندید رکورد را مانند مثال۲ به طور یکجا فراخوانی کرد.

مثال :

```
$posts=Post::model()->with('author')->findAll();
```

مثال بالا رکوردهایی از مدل Post را بر می گرداند به طوری که فیلد **author** **در هر Post با مقدار معادل آن در جدول User از قبل مرتبط شده است.**

نکته : می توانیم همزمان نام چند رابطه را بنویسیم و مقادیر آنها را به طور همزمان بازیابی نماییم. مثال :

```
$posts=Post::model()->with('author','categories')->findAll();
```

همچنین می توان به صورت یک لیست از اسامی در رابطه آن را تعریف نمود :

```
$posts=Post::model()->with(
    'author.profile',
    'author.posts',
    'categories')->findAll();
```

همچنین می توانیم از CDbCriteria::with استفاده نماییم :

```
$criteria=new CDbCriteria;
$criteria->with=array(
    'author.profile',
    'author.posts',
    'categories',
);
$posts=Post::model()->findAll($criteria);
```

و یا :

```
$posts=Post::model()->findAll(array(
    'with'=>array(
        'author.profile',
        'author.posts',
        'categories',
    )
));
```

اجرای پرس و جو های رابطه ای بدون استفاده از مدل های داری رابطه :

به عنوان مثال ما می خواهیم تمامی user هایی را پیدا کنیم که که post هایی را published کرده اند و ما علاقه ای نداریم که بدانیم خود پست ها چه هستند. در این صورت به شکل زیر می نویسیم :

```
$users=User::model()->with(array(
    'posts'=>array(
        // we don't want to select posts
        'select'=>false,
        // but want to get only users with published posts
        'joinType'=>'INNER JOIN',
        'condition'=>'posts.published=1',
    ),
))->findAll();
```

گزینه های پرس و جو های رابطه ای :

تعریف یک رابطه به شکل زیر مطرح شد :

```
'VarName'=>array('RelationType', 'ClassName', 'ForeignKey',
...additional options)
```

در آن گزینه های اضافی یا additional options می تواند به شکل زیر تعریف شوند :

- **select**: a list of columns to be selected for the related AR class. It defaults to '*', meaning all columns. Column names referenced in this option should be disambiguated.
- **condition**: the **WHERE** clause. It defaults to empty. Column names referenced in this option should be disambiguated.
- **params**: the parameters to be bound to the generated SQL statement. This should be given as an array of name-value pairs.
- **on**: the **ON** clause. The condition specified here will be appended to the joining condition using the **AND** operator. Column names referenced in this option should be disambiguated. This option does not apply to **MANY_MANY** relations.
- **order**: the **ORDER BY** clause. It defaults to empty. Column names referenced in this option should be disambiguated.
- **with**: a list of child related objects that should be loaded together with this object. Be aware that using this option inappropriately may form an infinite relation loop.
- **joinType**: type of join for this relationship. It defaults to **LEFT OUTER JOIN**.
- **alias**: the alias for the table associated with this relationship. It defaults to null, meaning the table alias is the same as the relation name.

- **together**: whether the table associated with this relationship should be forced to join together with the primary table and other tables. This option is only meaningful for **HAS_MANY** and **MANY_MANY** relations. If this option is set false, the table associated with the **HAS_MANY** or **MANY_MANY** relation will be joined with the primary table in a separate SQL query, which may improve the overall query performance since less duplicated data is returned. If this option is set true, the associated table will always be joined with the primary table in a single SQL query, even if the primary table is paginated. If this option is not set, the associated table will be joined with the primary table in a single SQL query only when the primary table is not paginated. For more details, see the section "Relational Query Performance".
- **join**: the extra **JOIN** clause. It defaults to empty. This option has been available since version 1.1.3.
- **group**: the **GROUP BY** clause. It defaults to empty. Column names referenced in this option should be disambiguated.
- **having**: the **HAVING** clause. It defaults to empty. Column names referenced in this option should be disambiguated.
- **index**: the name of the column whose values should be used as keys of the array that stores related objects. Without setting this option, an related object array would use zero-based integer index. This option can only be set for **HAS_MANY** and **MANY_MANY** relations.

همچنین گزینه های زیر جهت بارگزاری تنبل می تواند استفاده شود :

- **limit**: limit of the rows to be selected. This option does NOT apply to **BELONGS_TO** relation.
- **offset**: offset of the rows to be selected. This option does NOT apply to **BELONGS_TO** relation.
- **through**: name of the model's relation that will be used as a bridge when getting related data. Can be set only for **HAS_ONE** and **HAS_MANY**. This option has been available since version 1.1.7.

مثال :

در مثال زیر رابطه Post با گزینه های اضافی اصلاح شده است

```
class User extends CActiveRecord
{
    public function relations()
    {
        return array(
            'posts'=>array(self::HAS_MANY, 'Post',
'author_id',
                           'order'=>'posts.create_time DESC',
                           'with'=>'categories'),
```

```
            'profile'=>array(self::HAS_ONE, 'Profile',
'owner_id'),
        );
    }
}
```

حال اگر ما به **$author->posts** **دسترسی پیدا کنیم ما پست های** author را خواهیم داشت که بر اساس زمان ایجادشان به صورت نزولی مرتب شده اند.

فیلد های با نام تکراری :

اگر یک فیلد با نام یکسان در چند جدول تکرار شده باشد برای اینکه تداخل نامها پیش نیاید باید آن نام را توسط نام جدول معادلش تعریف کرد که این کار از تداخل نامها جلوگیری می کند. مثال :

```
$posts=Post::model()->with('comments')->findAll(array(
    'order'=>'t.create_time, comments.create_time'
));
```

گزینه های پرس و جوی رابطه ای پویا :

در مدل user بالا اگر بخواهیم از روش بارگزاری مشتاقانه استفاده نماییم تا بتوانیم پست های مرتبط با یک author خاص را بر گردانیم که به طور صعودی مرتب شده باشند (و این در حالی است که در تعریف رابطه در مدل این ترتیب به شکل نزولی نعریف ده است) می توانیم به شکل زیر عمل کنیم که در واقع گزینه های جدید بر روی گزینه های قبلی overwrite می شوند:

```
User::model()->with(array(
    'posts'=>array('order'=>'posts.create_time ASC'),
    'profile',
))->findAll();
```

این روش همچنین برای بارگزاری تنبل نیز استفاده می شود. مثلا در مثال زیر پست های کاربر که مقدار status آن ۱ است برگردانده می شوند :

```
$user=User::model()->findByPk(1);
$posts=$user->posts(array('condition'=>'status=1'));
```

کارایی پرس و جو های در رابطه :

همانطوری که گفته شد روش بارگزاری مشتاقانه زمانی مفید است که بخواهیم چندید رکرود در رابطه را به طور یکجا برگردانیم. انجام چنین کاری اگر می تواند با تعریف فیلترهایی در تعریف رابطه در مدل نیز انجام گیرد که هر دو روش دارای محاسن و معایبی هستند. مثال :

روش اول (در تعریف رابطه)

```
public function relations()
{
    return array(
        'comments' => array(self::HAS_MANY, 'Comment',
'post_id', 'together'=>false),
    );
}
```

روش دوم (در خود رابطه مشتاقانه)

```
$posts = Post::model()-
>with(array('comments'=>array('together'=>false)))->findAll();
```

پرس و جو های استاتیک :

مواردی مانند شمارش تعداد رکوردهای خاص یک جدول در رابطه و کارهایی از این دست توسط پرسو جو های استاتیک قابل انجام است. این عملیات تنها بر روی مدل های رابطه ای نوع ۱-چند و چند-چند قابل انجام است. مثال :

```
class Post extends CActiveRecord
{
    public function relations()
    {
        return array(
            'commentCount'=>array(self::STAT, 'Comment',
'post_id'),
            'categoryCount'=>array(self::STAT, 'Category',
'post_category(post_id, category_id)'),
        );
    }
```

```
}
```

در مثال بالا دو پرس و جوی استاتیک وجود دارند :

```
commentCount : تعداد comment های یک  post را شمارش می کند.
categoryCount: تعداد categories را که یک post به آن تعلق دارند را مشخص می کند.
```

```
با دستور بالا ما می توانیم به comment های یک post توسط دستور زیر دسترسی پیدا کنیم که یک روش تنبل است :
$post->commentCount
```

همچنین توسط روش مشتاقانه می توانیم به شکل زیر داشته باشیم :

```
$posts=Post::model()->with('commentCount', 'categoryCount')->findAll();
```

نکته : به طور پیش فرض یک پرس و جوی استاتیک عمل شمارش تعداد رکوردها یا همان COINT را انجام می دهد.

ما می توانیم در پرس و جویمان از گزینه های اضافی تری نیز استفاده کنیم که خلاصه آن در زیر آمده است :

- **select**: the statistical expression. Defaults to **COUNT(*)**, meaning the count of child objects.
- **defaultValue**: the value to be assigned to those records that do not receive a statistical query result. For example, if a post does not have any comments, its **commentCount** would receive this value. The default value for this option is 0.
- **condition**: the **WHERE** clause. It defaults to empty.
- **params**: the parameters to be bound to the generated SQL statement. This should be given as an array of name-value pairs.
- **order**: the **ORDER BY** clause. It defaults to empty.
- **group**: the **GROUP BY** clause. It defaults to empty.
- **having**: the **HAVING** clause. It defaults to empty.

پرس و جو های رابطه ای با Named Scopes

پرس و جو های رابطه ای همچنین می توانند در ترکیب با named scopes ها به کار روند. این کار به دو شکل انجام می شود. در روش اول named scopes ها در مدل اصلی اجرا می شوند

```
$posts=Post::model()->published()->recently()-
>with('comments')->findAll();
```

در روش دوم named scopes ها در مدل رابطه ای اجرا می شوند.

```
$posts=Post::model()->with('comments:recently:approved')-
>findAll();
// or since 1.1.7
$posts=Post::model()->with(array(
    'comments'=>array(
        'scopes'=>array('recently','approved')
    ),
))->findAll();
// or since 1.1.7
$posts=Post::model()->findAll(array(
    'with'=>array(
        'comments'=>array(
            'scopes'=>array('recently','approved')
        ),
    ),
));
```

همچنین Named scopes ها می توانند با with استفاده شوند. در مثال زیر اگر ما به `$user->posts` **دسترسی پیدا کنیم تمامی** comments های مربوط به post را نشان می دهد.

```
class User extends CActiveRecord
{
    public function relations()
    {
        return array(
            'posts'=>array(self::HAS_MANY, 'Post',
'author_id',
                'with'=>array(
                    'comments'=>array(
```

```
                        'scopes'=>'approved'
                    ),
                ),
            ),
        );
    }
}
```

شما همچنین می توانید پارامترهایی را نیز برای named scopes های رابطه ای ارسال نمایید. مثلا اگر گزینه ای به نام rated در post وجود داشته باشد داریم :

```
$users=User::model()->findAll(array(
    'with'=>array(
        'posts'=>array(
            'scopes'=>array(
                'rated'=>5,
            ),
        ),
    ),
));
```

پرس و جو های رابطه ای با through

استفاده از through به شکل زیر می باشد :

```
'relationName'=>array('relationType', 'ClassName',
'foreign_key', 'through'=>'throughRelationName', ...additional
options)
```

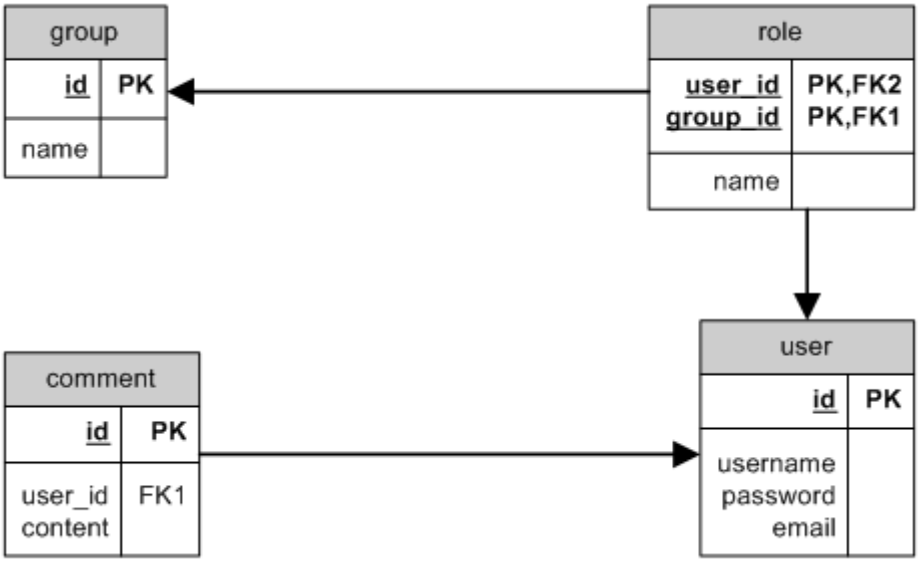
که در آن **foreign_key به نام کلیدی اشاره دارد که :**

۱- در جدولی تعریف شده است که در **throughRelationName مشخص می شود**

۲- **به جدولی اشدره دارد که داده های مدل ClassName را در خود نگه می دارد.**

Through برای هر دو نوع رابطه **HAS_ONE** و **HAS_MANY می تواند به کار می رود.**

HAS_MANY through



مثالی از HAS_MANY با through می تواند به دست آوردن کاربرانی باشد که در یک گروه قرار دارند وقتی که کاربران توسط role هایی گروه بندی شده اند.

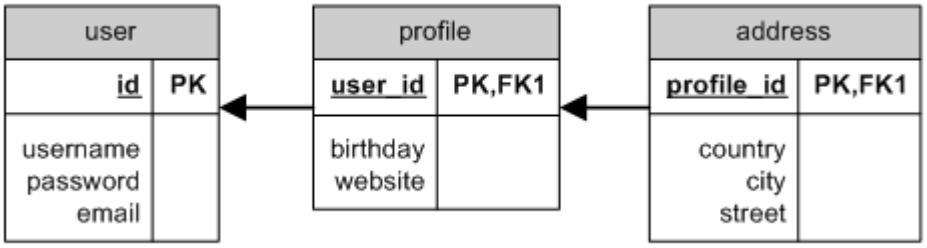
یک مثال پیچیده تر به شکل زیر است :

```
class Group extends CActiveRecord
{
   ...
   public function relations()
   {
       return array(
           'roles'=>array(self::HAS_MANY,'Role','group_id'),

'users'=>array(self::HAS_MANY,'User','user_id','through'=>'rol
es'),

'comments'=>array(self::HAS_MANY,'Comment','user_id','through'
=>'users'),
       );
   }
}
```

مثالهای کاربردی :

```
// get all groups with all corresponding users
$groups=Group::model()->with('users')->findAll();

// get all groups with all corresponding users and roles
$groups=Group::model()->with('roles','users')->findAll();

// get all users and roles where group ID is 1
$group=Group::model()->findByPk(1);
$users=$group->users;
$roles=$group->roles;

// get all comments where group ID is 1
$group=Group::model()->findByPk(1);
$comments=$group->comments;
```

HAS_ONE through



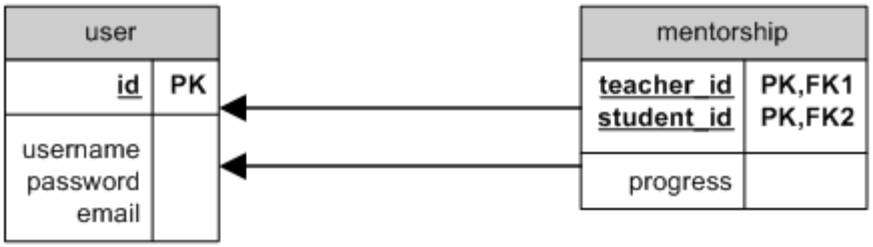
یک مثال از HAS_ONE با through به دست آوردن آدرس کاربرانی است که آدرس آنها در profile آنها تعریف شده است :

```
class User extends CActiveRecord
{
   ...
   public function relations()
   {
       return array(

'profile'=>array(self::HAS_ONE,'Profile','user_id'),

'address'=>array(self::HAS_ONE,'Address','profile_id','through
'=>'profile'),
       );
   }
}
```

مثال کاربردی :

```
// get address of a user whose ID is 1
$user=User::model()->findByPk(1);
$address=$user->address;
```

through on self



مثال :

```
class User extends CActiveRecord
{
   ...
   public function relations()
   {
       return array(

'mentorships'=>array(self::HAS_MANY,'Mentorship','teacher_id',
'joinType'=>'INNER JOIN'),

'students'=>array(self::HAS_MANY,'User','student_id','through'
=>'mentorships','joinType'=>'INNER JOIN'),
       );
   }
}
```

مثال کاربردی :

```
// get all students taught by teacher whose ID is 1
$teacher=User::model()->findByPk(1);
$students=$teacher->students;
```

نکته : در نوشتن کلیه پرس و جو های رابطه ای حفظ فاصله بین پارامترهای دستورات بسیار مهم است.

مهاجرات بانک اطلاعاتی

در بسیاری از موارد ممکن است در حین اجرای پروژ تغییری در ساختار بانک اطلاعاتی اصلی ایجاد شود مثلا یک index به یک جدول اضافه شود یا یک فیلد از یک جدول حذف شود برای دنبال کردن این تغییرات مفهوم مهاجرت در Yii مطرح می شود.

مراحل انجام مهاجرت :

1. Tim creates a new migration (e.g. create a new table)
2. Tim commits the new migration into source control system (e.g. SVN, GIT)
3. Doug updates from source control system and receives the new migration
4. Doug applies the migration to his local development database

ماژول ها و کامپوننت ها

معرفی ماژول

یک ماژول یک نمونه کامل از پروژه است که شامل کنترلر ها ویو ها و مدل ها و سایر اجزای یک پروژه کامل می باشد.دسترسی به ماژول :

```
http://hosname.com/index.php/ModuleID/ControllerID/Action
```

تفاوت ماژول با یک پروژه واقعی در این است که ماژول به تنهایی نمی تواند اجرا شود و باید حتما داخل یک پروژه واقعی فراخوانی شود.

مزایای استفاده از ماژول :

در پروژه های خیلی بزرگ که می توان هر بخش را تحت قالب یک ماژول پیاده سازی کرد و هر بخش توسط یک نفر اداره و پیاده سازی می شود. پس قابلیت کار تیمی را افزایش می دهد.همچنین می توان از یک ماژول ساخته شده در چندین پروژه استفاده کرد که باعث افزایش قابلیت استفاده مجدد می شود.یک ماژول می تواند شامل کلیه کنترلرها، ویوها، مدل ها و سایر اجزای بک پروژه واقعی باشد.خیلی از اجزای برنامه مثل مدیریت حسابها، مدیریت پیامها، فروم سایت، وبلاگ سایت و ... را می توان در قالب یک ماژول پیاده سازی نمود تا در آینده در دیگر پروژه های می توان از آنها استفاده نمود.

ساخت یک ماژول جدید :

یک ماژول باید شامل یک کلاس ماژول باشد که از کلاس CWebModule ارث بری می کند.

نام کلاس ماژول از قالب `ucfirst($id).'Module'` استفاده می کند مه در آن `$id` با `ModuleID` اشاره دارد. کلاس ماژول محلی برای ذخیره اطلاعاتی است که در کل ماژول مشترکا مورد استفاده قرار می گیرد.

به عنوان مثال ما می توانیم از CWebModule::params برای ذخیره پارامترهای مورد استفاده در ماژول استفاده کرد.و همچنین می توان از CWebModule::components برای اشتراک گذاری application components در سطح ماژول استفاده کرد.برای ساخت ماژول جدید می توان از ابزار module generator موجود در giiاستفاده نمود و یا از command line ابزار yiic استفاده نمود.

ساختار فایلها و پوشه های ایجاد شده برای ماژول forum به شکل زیر می باشد:

```
forum/
ForumModule.php          the module class file
components/              containing reusable user components
views/                   containing view files for widgets
controllers/             containing controller class files
DefaultController.php    the default controller class file
extensions/              containing third-party extensions
models/                  containing model class files
```

views/ containing controller view and layout files

layouts/ containing layout view files

default/ containing view files for DefaultController

index.php the index view file

نحوه استفاده از ماژول

برای استفاده از ماژول ابتدا باید پوشه ماژول را در protected\models\ قرار داد . سپس باید در فایل config ماژول موجود را اعلان نمود. به عنوان مثال برای فراخوانی ماژول forum بالا باید در فایل config مقادیر زیر را وارد کرد :

```
returnarray(
    ......
'modules'=>array('forum',...),
    ......
);
```

نحوه فراخوانی یک ماژول:

http://www.example.com/index.php?r=forum/post/create

یک ماژول می تواند شامل ماژول های داخلی نیز باشد. و یک یا چند ماژول داخل یک ماژول دیگر قرار گیرد که نحوه دسترسی به آن ماژول داخلی به شکل زیر می باشد:

```
ParentModuleID/childModuleID/controllerID/actionID
```

معرفی کامپوننت ها

همه اجزای برنامه از یک request ساده گرفته تا login همه از Components استفاده می کنند.

نحوه دسترسی به یک Component :

```
Pattern :   Yii::app()->ControllerId
Sample:  Yii::app()->cahche
```

اگر دسترسی به یک Component امکان پذیر باشد مقدار برگشتی NULL خواهد بود.

Component ها فقط زمانی در برنامه ساخته می شموند که به آنها نیاز باشد هرچند در برنامه ممکن است چندین Component تعریف شده باشد. و این مسئله به طور قابل ملاحظه ای کارایی Yii را افزایش می دهد.

انواع Component :

Core Application ComponentYiis

User Defined Component

به طور پیش فرض نام فایل کلاس با نام خود کلاس یکسان است.

نحوه دسترسی به یک کلاس از طریق alias امکان پذیر است.

هر Component از کلاس CComponent و یا کلاس های مشتق آن ساخته می شود.

اجزای کامپوننت

به طور کلی یک کامپوننت مانند هر کلاس دیگری می تواند شامل اجزای Property, Behaviors, Event باشد.

کلیه کدها مربوطه در راهنمای شی گرایی PHP موجود می باشند. از تعریف کلاس و رویداد و ...

تعریف واستفاده از یک کامپوننت

برای استفاده از یک component می توان به شکل زیر عمل کرد :

ساخت component

ابتدا باید وارد پوشه C:\xampp\htdocs\demo\protected\controllers شویم و یک فایل کلاس ایجاد نماییم :

فایلی به نام demo.php ایجاد می نماییم. باید توجه کرد که نام کلاس ببا نام فایل باید یکی باشد

محتوای کلاس را به شکل زیر وارد می کنیم :

```
<?php
     class demo
          {
               public function fun1()
```

```
                {
                    return 1;
                }
            }
?>
```

حال مثلا می خواهیم در کنترلرمان از این کلاس استفاده نماییم که نحوه استفاده از آن می تواند به شکل زیر باشد که با استفاده از نمونه سازی انجام می شود :

```
    $class1 = new demo();
    echo  $class1->fun1();
```

این کار باعث نمایش ۱ در خروجی می شود.

سایر کنترلر ها را نیز می توان به همین شکل ایجاد کرد.

روش دسترسی مستقیم به کلاس و بدون ساختن نمونه نیز به شکل زیر است :

```
echo Demo::fun1();
```

کامپوننت های پیش فرض

کامپوننت های زیر توسط CWebApplication. به طور پیش فرض تعریف شده هستند:

assetManager: CAssetManager - manages the publishing of private asset files.

authManager: CAuthManager - manages role-based access control (RBAC).

cache: CCache - provides data caching functionality. Note, you must specify the actual class (e.g. CMemCache, CDbCache). Otherwise, null will be returned when you access this component.

clientScript: CClientScript - manages client scripts (javascripts and CSS).

coreMessages: CPhpMessageSource - provides translated core messages used by Yii framework.

db: CDbConnection - provides the database connection. Note, you must configure its connectionString property in order to use this component.

errorHandler: CErrorHandler - handles uncaught PHP errors and exceptions.

format: CFormatter - formats data values for display purpose. This has been available since version 1.1.0.

messages: CPhpMessageSource - provides translated messages used by Yii application.

request: CHttpRequest - provides information related with user requests.

securityManager: CSecurityManager - provides security-related services, such as hashing, encryption.

session: CHttpSession - provides session-related functionalities.

statePersister: CStatePersister - provides global state persistence method.

urlManager: CUrlManager - provides URL parsing and creation functionality.

user: CWebUser - represents the identity information of the current user.

themeManager: CThemeManager - manages themes.

تشخیص هویت کاربر و حدود دسترسی

تشخیص هویت کاربر توسط کلاس CWebUser انجام می شود و کلیه اطلاعات کاربر مثل نام کاربر، وضعیت کاربر و ... را در Session هایی ذخیره می کند. نحوه دسترسی به این session ها به شکل زیر می باشد :

Yii::app()->user

برای اینکه مشخصات کاربر از بانک اطلاعاتی MySQL خوانده شود کد زیر را داری :

```
<?php
class UserIdentity extends CUserIdentity
{
    private $_id;
    public function authenticate()
    {
        $record=TblUser::model()-
>findByAttributes(array('username'=>$this->username));
        if($record===null)
```

```
            $this->errorCode=self::ERROR_USERNAME_INVALID;
        else if($record->password!==($this->password))
            $this->errorCode=self::ERROR_PASSWORD_INVALID;
        else
        {
            $this->_id=$record->id;
            $this->setState('id', $record->id);
               $this->setState('group', $record->group);


            $this->errorCode=self::ERROR_NONE;
        }
        return !$this->errorCode;
    }

    public function getId()
    {
        return $this->_id;
    }
}
?>
```

چند نکته مهم :

نام مدل باید مثل TblUser وارد شده باشد.

این جدول باید دارای فیلد username و password باشد.

می توان مقادیری را در همین ابتدا به صورت Session ذخیره کنیم و در طول برنامه از آنها استفاده کنیم مثل id و group در مثال بالا. که نحوه دسترسی به آنها در برنامه هم مثل کد زیر است :

```
echo Yii::app()->user->id;

echo Yii::app()->user->group;
```

معرفی فرم ها

فرم ها جهت دریافت اطلاعات کاربر از ورودی مورد استفاده قرار می گیرند و در پوشه ویو ها ذخیره می شوند.

نحوه ایجاد فرم جدید

چندین روش برای ایجاد فرم وجود دارد که ابزار gii به طور خودکار فرم های اولیه را ایجاد می کند.

روش اول : ساده ترین روش – مورد استفاده توسط gii

```
<div class="form">
<?php echo CHtml::beginForm(); ?>

<?php echo CHtml::errorSummary($model); ?>

<div class="row">
    <?php echo CHtml::activeLabel($model,'username'); ?>
    <?php echo CHtml::activeTextField($model,'username') ?>
</div>

<div class="row">
   <?php echo CHtml::activeLabel($model,'password'); ?>
   <?php echo CHtml::activePasswordField($model,'password') ?>
</div>

<div class="row submit">
   <?php echo CHtml::submitButton('Login'); ?>
</div>

<?php echo CHtml::endForm(); ?>
</div><!-- form -->
```

برای تعیین css های فرم از blueprint و کلاسهای مبوط به form استفاده شده است.

روش دوم : استفاده از ویجت CActiveForm

```
<div class="form">
<?php $form=$this->beginWidget('CActiveForm'); ?>

    <?php echo $form->errorSummary($model); ?>

    <div class="row">
        <?php echo $form->label($model,'username'); ?>
        <?php echo $form->textField($model,'username') ?>
```

```
    </div>

    <div class="row">
        <?php echo $form->label($model,'password'); ?>
        <?php echo $form->passwordField($model,'password') ?>
    </div>

    <div class="row submit">
        <?php echo CHtml::submitButton('Login'); ?>
    </div>

<?php $this->endWidget(); ?>
</div><!-- form -->
```

روش سوم : استفاده از Form Builder

از آنجایی که روشهای قبلی باعث نوشتن کدهای زیاد می شود روش Form Builder به عنوان جدید ترین روش که روشی برای کد نویسی ساده تر و کمتر است مورد استفاده قرار می گیرد.

یک فرم و چند مدل :

گاهی لازم است اطلاعات مربوط به چند مدل از طریق یک فرم دریافت و ارسال شود که به آن روش Tabular Input گفته می شود. برای استفاده از این روش در کنترلر ابتدا اکشن را به شکل زیر تعریف و یا اصلاح می نماییم :

```
public function actionBatchUpdate()
{
    // retrieve items to be updated in a batch mode
    // assuming each item is of model class 'Item'
    $items=$this->getItemsToUpdate();
    if(isset($_POST['Item']))
    {
        $valid=true;
        foreach($items as $i=>$item)
        {
            if(isset($_POST['Item'][$i]))
                $item->attributes=$_POST['Item'][$i];
            $valid=$item->validate() && $valid;
        }
        if($valid)  // all items are valid
            // ...do something here
    }
```

```
    // displays the view to collect tabular input
    $this->render('batchUpdate',array('items'=>$items));
}
```

حال فرم مورد نظر را به شکل زیر تعریف می کنیم :

```
<div class="form">
<?php echo CHtml::beginForm(); ?>
<table>
<tr><th>Name</th><th>Price</th><th>Count</th><th>Description</
th></tr>
<?php foreach($items as $i=>$item): ?>
<tr>
<td><?php echo CHtml::activeTextField($item,"[$i]name");
?></td>
<td><?php echo CHtml::activeTextField($item,"[$i]price");
?></td>
<td><?php echo CHtml::activeTextField($item,"[$i]count");
?></td>
<td><?php echo CHtml::activeTextArea($item,"[$i]description");
?></td>
</tr>
<?php endforeach; ?>
</table>

<?php echo CHtml::submitButton('Save'); ?>
<?php echo CHtml::endForm(); ?>
</div><!-- form -->
```

نکته : توجه کنید که ما در این مثال به جای name از [$i]name استفاده می کنیم.

Caching

معرفی caching

کش کردن راهی است برای افزایش بازدهی سایتهای اینترنتی توسط ذخیره موقت داده های استاتیک که به طور مکرر مورد استفاده قرار می گیرند. این داده ها در نهانگاه کش ذخیره می شوند.

برای شروع کار باید در فایل config تنظیمات مربوط به component کش را اضافه کرد. مثال :

```
array(
    ......
'components'=>array(
        ......
'cache'=>array(
'class'=>'system.caching.CMemCache',
'servers'=>array(
array('host'=>'server1', 'port'=>11211, 'weight'=>60),
array('host'=>'server2', 'port'=>11211, 'weight'=>40),
),
),
),
);
```

این مثال از دو سرور Server1 و Server2 جهت ذخیره اطلاعات مورد استفاده قرار می گیرد.

در هنگام اجرای برنامه توسط via Yii::app()->cache می توان به محتویات حافظه نهانگاه دسترسی پیدا کرد.

در Yii کامپوننت های مختلفی جهت کار با کش آماده شده است که در زیر توضیح مختصر هر کدام را می بینیم :

- CMemCache: uses PHP memcache extension.
- CApcCache: uses PHP APC extension.
- CXCache: uses PHP XCache extension. Note, this has been available since version 1.0.1.
- CEAcceleratorCache: uses PHP EAccelerator extension.
- CDbCache: uses a database table to store cached data. By default, it will create and use a SQLite3 database under the runtime directory. You can explicitly specify a database for it to use by setting its connectionID property.
- CZendDataCache: uses Zend Data Cache as the underlying caching medium. Note, this has been available since version 1.0.4.
- CFileCache: uses files to store cached data. This is particular suitable to cache large chunk of data (such as pages). Note that this has been available since version 1.0.6.
- CDummyCache: presents dummy cache that does no caching at all. The purpose of this component is to simplify the code that needs to check the availability of cache. For example, during development or if the server doesn't have actual cache support, we can use this cache component.

When an actual cache support is enabled, we can switch to use the corresponding cache component. In both cases, we can use the same code `Yii::app()->cache->get($key)` to attempt retrieving a piece of data without worrying that `Yii::app()->cache` might be `null`. This component has been available since version 1.0.5.

کلیه کامپوننت های فوق از کلاس CCache ارث بری می کنند.

کش کردن در سطوح مختلفی قابل انجام است :

۱- پایین ترین سطح ذخیره یک داده ساده مثل محتوای یک متغیر ساده است که به آن *data caching* می گویند.
۲- سطح متوسط مربوط به ذخیره یک فریم یا یک قسمت از صفحه در قالب یک ویو است که به آن Fragment caching می گویند.
۳- بالاترین سطح مربوط به ذخیره یک صفحه کامل است.

نکته : نباید از حافظه کش برای ذخیره داده های مهم مثل Session ها استفاده کرد زیرا حافظه نهان یک محل موقت و نا مطمئن است.

Data Caching

پایین ترین سطح ذخیره یک داده ساده مثل محتوای یک متغیر ساده است که به آن*data caching* می گویند. کلاس پایه Ccache که والد بقیه کامپوننت های کش است از دو متد مهم پشتیبانی می کند این دو متد get و set می باشند.

برای ذخیره `$value` در حافظه نهان ما ابتدا یک ID یکتا را در نظر می گیریم و توسط متد set آن را در حافظه نهان ذخیره می کنیم . مثال :

```
Yii::app()->cache->set($id, $value);
```

داده ها در حافظه نهان باقی می مانند تا زمانی که بر اثر یک سیاست کش مثل حذف کش و یا پر شدن کش اطلاعات حذف شوند همچنین می توان زمانی را مشخص کرد که پس از گذشت آن زمان اطلاعات کش پاک می شوند. این زمان برحسب ثانیه است.مثال :

```
// keep the value in cache for at most 30 seconds
Yii::app()->cache->set($id, $value, 30);
```

برای بازگرداندن اطلاعات از متد gget و شناسه یکتای آن داده در کش یعنی $idاستفاده می کنیم. اطلاعات کش را می توان در یک متغیر بازیابی کرد. اگر مقدار برگشتی false باشد یعنی آن داده در حافظه کش وجود ندارد. مثالی از بازیابی داده ها به شکل زیر است :

```
$value=Yii::app()->cache->get($id);
if($value===false)
{
// regenerate $value because it is not found in cache
```

```
// and save it in cache for later use:
// Yii::app()->cache->set($id,$value);
}
```

نکته : $id باید همواره یکتا باشد. البته نیازی نیست که $id در تمامی برنامه هایی (سایت ها و صفحات) که باز هستند یکتا باشد زیرا سیستم کش به اندازه کافی هوشمند عمل می کند که تشخیص دهد از کدام مقادیر در کش استفاده کند.

برای حذف یک مقدار از حافظه نهان از متد delete() استفاده می کنیم. و برای حذف کامل حافظه کش از متد flush() استفاده می شود هنگام استفاده از این متد باید مواظب با شیم زیرا این متد کلیه داده های موجود در کش از جمله داده های مربوط به برنامه های دیگر را نیز حذف می کند.

از آنجایی که Ccache از ArrayAccess استفاده می کند لذا میتوان به شکل زیر نیز به محتویات آن دسترسی پیدا کرد :

```
$cache=Yii::app()->cache;
$cache['var1']=$value1;  // equivalent to: $cache->set('var1',$value1);
$value2=$cache['var2'];  // equivalent to: $value2=$cache->get('var2');
```

وابستگی کش

فرض کنیم بخواهیم محتویات یک فایل را کش کنیم حال اگر محتویات فایل برای بار اول در کش قرار گیرد و سپس محتویات فایل تغییر کند دیگر نمی توان به داده های موجود در کش اطمینان کرد برای رفع این مشکل از کلاس CcacheDependency و یا فرزندان این کلاس استفاده می کنیم. مثال :

```
// the value will expire in 30 seconds
// it may also be invalidated earlier if the dependent file is changed
Yii::app()->cache->set($id, $value, 30, newCFileCacheDependency('FileName'));
```

حال به همان روش قبلی می توان توسط متد get اطلاعات کش را بازیابی کرد با این تفاوت که اگر محتویات اصلی تغییر کند محتویات کش نیز تغییر پیدا می کند.

در زیر خلاصه روشهای کار با dependency را ملاحظه می کنید :

- CFileCacheDependency: the dependency is changed if the file's last modification time is changed.
- CDirectoryCacheDependency: the dependency is changed if any of the files under the directory and its subdirectories is changed.
- CDbCacheDependency: the dependency is changed if the query result of the specified SQL statement is changed.
- CGlobalStateCacheDependency: the dependency is changed if the value of the specified global state is changed. A global state is a variable that is persistent across multiple requests and multiple sessions in an application. It is defined via CApplication::setGlobalState().
- CChainedCacheDependency: the dependency is changed if any of the dependencies on the chain is changed.

- CExpressionDependency: the dependency is changed if the result of the specified PHP expression is changed. This class has been available since version 1.0.4.

Fragment caching

برای ذخیره یک فریم از داده ها مثلا گزارش سالیانه از آمار هواشناسی و ... می توان از این روش استفاده کرد. مثال :

```
...otherHTMLcontent...
<?phpif($this->beginCache($id)){?>
...content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

گزینه های کش :

تعیین مدت زمان قرار گیری داده ها در کش

```
...otherHTMLcontent...
<?phpif($this->beginCache($id, array('duration'=>3600))){?>
...content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

نکته : اگر مدت زمان مشخص نشده باشد به صورت پیش فرض ۶۰ می باشد یعنی بعد از ۶۰ ثانیه محتویات آن پاک می شوند.

وابستگی داده ها :

همانند وابستگی داده ها در data caching می توان وابستگی داده ها را در fragment caching نیز مشخص نمود. مثال :

```
...otherHTMLcontent...
<?phpif($this->beginCache($id, array('dependency'=>array(
'class'=>'system.caching.dependencies.CDbCacheDependency',
'sql'=>'SELECT MAX(lastModified) FROM Post')))){?>
...content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

تنوع داده ها :

در بعضی از موارد ممکن است بخواهیم کش کردن داده ها را به گونه ای انجام دهیم که تفاوت داده ها در یک یا چند فاکتور مشخص گردد مثلا در ذخیره پروفایل کاربر داده ها توسط ID یکتا کاربران از هم

متمایز می شوند که می توان آن تنوع داده را در هنگام BeginCache مشخص کرد. انواع حالتهای آن به شکل زیر است :

- varyByRoute: by setting this option to true, the cached content will be variated according to route. Therefore, each combination of the requested controller and action will have a separate cached content.
- varyBySession: by setting this option to true, we can make the cached content to be variated according to session IDs. Therefore, each user session may see different content and they are all served from cache.
- varyByParam: by setting this option to an array of names, we can make the cached content to be variated according to the values of the specified GET parameters. For example, if a page displays the content of a post according to the `id` GET parameter, we can specify varyByParam to be `array('id')` so that we can cache the content for each post. Without such variation, we would only be able to cache a single post.
- varyByExpression: by setting this option to a PHP expression, we can make the cached content to be variated according to the result of this PHP expression. This option has been available since version 1.0.4.

درخواست نوع :

می توان مشخص نمود که چه نوع داده ای کش گردد مثلا فرمی که از نوع POST است و یا از نوع GET است کش گردد. مثال :

```
...otherHTMLcontent...
<?phpif($this->beginCache($id, array('requestTypes'=>array('GET')))){?>
...content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

کش کردن تو در تو :

می توان کش کردن را به صورت تو در تو مشخص کرد. مثال :

```
...otherHTMLcontent...
<?phpif($this->beginCache($id1)){?>
...outer content to be cached...
<?phpif($this->beginCache($id2)){?>
    ...inner content to be cached...
<?php$this->endCache(); }?>
...outer content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

Page Caching

برای کش کردن یک صفحه به طور کامل می توان از اکشن فیلتر استفاده کرد. مثال :

```
publicfunctionfilters()
{
returnarray(
array(
'COutputCache',
'duration'=>100,
'varyByParam'=>array('id'),
),
);
}
```

نکته : ما می توانیم از CoutputCache به عنوان یک فیلتر استفاده کنیم زیرا این کامپوننت از CfilterWidget ارث بری می کند. این یعنی اینکه CoutputCache هم یک فیلتر است و هم یک ویگت.

محتوای پویا :

کش کردن محتوا معمولا جهت داده های استاتیک انجام می شود مثلا یک صفحه Help را در نظر بگیرید که کلیه محتوای آن استاتیک است البته ممکن است کاربر با نام کاربری وارد شده باشد که در این صورت کلیه محتوای این صفحه استاتیک است به جز نام کاربری که متغیر است و هر کاربری که وارد شود نام کاربری او را می نویسد. حال برای کش کردن این صفحه به دو روش می توان عمل کرد:

روش اول : تقسیم کردن صفحه به چند fragment مجزا و کش کردن جداگانه آنها.

روش دوم : استفاده از محتوای پویا (روش بهتر)

در روش محتوای پویا ما CController::renderDynamic را هر جا که بخواهیم فراخوانی می کنیم.

```
...otherHTMLcontent...
<?phpif($this->beginCache($id)){?>
...fragment content to be cached...
<?php$this->renderDynamic($callback); ?>
...fragment content to be cached...
<?php$this->endCache(); }?>
...otherHTMLcontent...
```

در مثال بالا `$callback` به PHP callback معتبر اشاره دارد که آن می توان یک رشته باشد که به نام یک متد در کلاس کنترلر جاری اشاره دارد. این همچنین می تواند به آرایه ای اشاره کند کهبه متد کلاس اشاره دارد. هر پارامتر اضافی در renderDynamic به callback ارسال می شود. Callback باید محتوای پویا را برگرداند به جای آن که آن را نشان دهد.

توسعه هایYii

معرفی توسعه ها :

اگر هر کدی نوشته شود که بعدا بتوان از آن در جای دیگر استفاده کرد به آن کد یک *extension یا همان توسعه می گویند*.یک توسعه می تواند به یکی از اشکال زیر باشد :

- application component
- behavior
- widget
- controller
- action
- filter
- console command
- validator: a validator is a component class extending CValidator.
- helper: a helper is a class with only static methods. It is like global functions using the class name as their namespace.
- module: a module is a self-contained software unit that consists of models, views, controllers and other supporting components. In many aspects, a module resembles to an application. The main difference is that a module is inside an application. For example, we could have a module that provides user management functionalities.

همچنین هر کامپوننت دیگری که نوشته شده باشد و بتوان از آن در برنامه استفاده کرد می تواند در تعریف فوق قرار گیرد.

استفاده از توسعه ها :

استفاده از توسعه ها معمولا در سه مرحله زیر انجام می گیر :

۱- دانلود یا دریافت توسعه (از آدرس http://www.yiiframework.com/extensions)
۲- از حالت فشرده خارج کردن بسته توسعه در مسیر extensions/xyz که xyz در آن ناو توسعه است.
۳- Import کردن و تنظیم کردن توسعه در پروژه و استفاده از آن.

توسعه های مختلف دارای تنظیمات و نحوه استفاده متفاوت هستند. در زیر به چند نمونه از مهمترین آنها اشاره می کنیم.

Zii Extensions

برخی از توسعه ها مربوط به خود پدید آورندگان yii می باشد که به صورت پی فرض در فریم ورک وجود دارند و به آنها zii گفته می شود و در پوشه ای با همین نام قرار دارند. نحوه فراخوانی این توسعه ها در پروژه به وسیله `zii.path.to.ClassName` است. مثلا اگر بخواهیم توسعه `CGridView` را در پروژ] استفاده نماییم به شکل زیر عمل مل کنیم :

```
$this->widget('zii.widgets.grid.CGridView', array(
'dataProvider'=>$dataProvider,
));
```

Application Component

برای استفاده از یک application component ابتدا باید application configuration را تنظیم نماییم که این کار به شکل زیر انجام می شود :

```
returnarray(
// 'preload'=>array('xyz',...),
'components'=>array(
'xyz'=>array(
'class'=>'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
),
// other component configurations
),
);
```

سپس در هر کجای برنامه توسط Yii::app()->xyz می توان به آن دسترسی پیدا کرد. این کامپونت به صورت lazily created می باشد. یعنی تنها زمانی ساخته خواهد شد که به آن نیاز پیدا کنیم مگر اینکه آن را به صورت preload تعریف کرده باشیم که در این صورت در هنگام اجرای برنامه ساخته می شود.

Behavior

Behavior ها در هر نوع از کامپوننت قابل استفاده می باشند. استفاده از behavior در دو مرحله انجام می شود. در مرحله اول behavior به یک کامپوننت الحاق داده می شود و در مرحله دوم behavior توسط آن کامپوننت فراخوانی می شود. به عنوان مثال :

```
// $name uniquely identifies the behavior in the component
$component->attachBehavior($name,$behavior);
// test() is a method of $behavior
$component->test();
```

همچنین می توان به جای استفاده از attachBehavio مستقیما behavior را به کامپوننت الحاق نمود. مثال :

```
returnarray(
```

```
'components'=>array(
'db'=>array(
'class'=>'CDbConnection',
'behaviors'=>array(
'xyz'=>array(
'class'=>'ext.xyz.XyzBehavior',
'property1'=>'value1',
'property2'=>'value2',
),
),
),
//....
),
);
```

در مثال بالا behavior **xyz** به application component **db** الحاق داده شده است. این کار قابل انجام است زیرا که CApplicationComponent دارای یک خاصیت به نام **behaviors**است.

برای کلاسهای CController, CFormModel و CActiveRecord که اغلب نیاز به افزودن توسعه ها دارند الحاق توسعه توسط override کزدن متد **behaviors()** آنها قابل انجام است و کلاس به صورت خودکار کلیه **behavior** های لازم را الحاق می کند. مثال :

```
publicfunctionbehaviors()
{
returnarray(
'xyz'=>array(
'class'=>'ext.xyz.XyzBehavior',
'property1'=>'value1',
'property2'=>'value2',
),
);
}
```

Widget

ویگت ها معمولا در ویو ها به کار می روند. کلاس ویگت **XyzClass** متعلق به extension **xyz** داده شده است که ما می توانیم آن را در ویو به شکل زیر به کار ببریم :

```
// widget that does not need body content
<?php$this->widget('ext.xyz.XyzClass', array(
'property1'=>'value1',
'property2'=>'value2')); ?>

// widget that can contain body content
```

```
<?php$this->beginWidget('ext.xyz.XyzClass', array(
'property1'=>'value1',
'property2'=>'value2')); ?>

...bodycontentofthewidget...

<?php$this->endWidget(); ?>
```

Action

اکشن ها در کنترلر ها به کار می روند تا به یک درخواست کاربر واکنش نشان دهند. کلاس اکشن XyzClass متعلق به xyz extension داده شده است در کلاس کنترلر مان می توانیم توسط override کردن CController::actions به آن دسترسی پیدا کنیم. مثال :

```
classTestControllerextendsCController
{
publicfunctionactions()
{
returnarray(
'xyz'=>array(
'class'=>'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
),
// other actions
);
}
}
```

حال این اکشن توسط test/xyz قابل استفاده و دسترسی است.

Filter

فیلتر ها هم توسط کنترلر مورد استفاده قرار می گیرند. کلاس فیلتر XyzClass متعلق به xyz extension داده شده است.در کلاس کنترلر مان می توانیم توسط override کردن CController::filters به آن دسترسی پیدا کنیم. مثال :

```
classTestControllerextendsCController
{
publicfunctionfilters()
{
returnarray(
array(
```

```
'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
),
// other filters
);
}
}
```

Controller

یک کنترلر شامل تعدادی از اکشن ها می باشد. برای استفاده از یک توسعه کنترلر باید خاصیت CWebApplication::controllerMap را در application configuration: تنظیم نماییم :

```
returnarray(
'controllerMap'=>array(
'xyz'=>array(
'class'=>'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
),
// other controllers
),
);
```

حال یک اکشن مانند a توسط xyz/a قابل دسترسی است.

Validator

یک Validator عموما در کلاس مدل مورد استفاده قرار می گیرد کلاس validator با نام XyzClass داده شده است ما می توانیم توسط override کردن متد CModel::rules در کلاس مدل به آن دسترسی پیدا کنیم. مثال :

```
classMyModelextendsCActiveRecord// or CFormModel
{
publicfunctionrules()
{
returnarray(
array(
'attr1, attr2',
'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
```

```
),
// other validation rules
);
}
}
```

Console Command

یک دستور خط فرمان جهت بهبود و کمک به دستور yiic به کار می رود. یک دستور خط فرمان با عنوان XyzClass متعلق به توسعه xyz داده شده است ما می توانیم به آن از طریق تنظیمات مربوط به console application دسترسی پیدا کنیم.
مثال :

```
returnarray(
'commandMap'=>array(
'xyz'=>array(
'class'=>'ext.xyz.XyzClass',
'property1'=>'value1',
'property2'=>'value2',
),
// other commands
),
);
```

حال می توان از yiic با دستور کمکیxyz استفاده کرد.

نکته : تنظیمات مربوط به خط فرمان در یک فایل config متفاوت که در آدرس protected/config/console.php موجود می باشد قرار دارند که با config اصلی پروژه متفاوت است.

Generic Component

قبل از استفاده از یک کامپوننت Generic باید آن کلاس را به شکل زیر در پروژه وارد کنیم :

```
Yii::import('ext.xyz.XyzClass');
```

حال می توانیم یک نمونه از روی آن کلاس یا کلاسهای فرزند آن ساخته (ساخت شی) و از آن استفاده کنیم.

ایجاد توسعه ها

نکاتی برای ایجاد توسعه جدید :

- یک توسعه جدید باید حداقل وابستگی به منابع دیگر را داشته باشد.
- کلیه فایلها و پوشه های مربوط به یک توسعه باید داخل یک پوشه اصلی با نام توسعه باشند.
- کلاسهای یک توسعه باید با یک پیش نام شروع شوند تا از تداخل نامها با کلاسهای اصلی برنامه جلوگیری شود.
- توضیحات کدها باید جامع و کافی باشد تا اگر فرد دیگری خواست آن را تغییر دهد یا به روز رسانی کند بتواند.
- یک توسعه می تواند تحت یک لیسانس قرار گیرد مثل لیسانس BSD, MIT و غیره

ساخت توسعه های گوناگون :

Application Component

یک کامپوننت از این نوع می تواند از IapplicationComponent ارث بری کند و یا از کلاس CApplicationComponent ارث بری نماید.

متد اصلی مورد نیاز برای آن IApplicationComponent::init می باشد. که متد اولیه است و تنظیمات ابتدایی در آن قرار دارد.

Behavior

برای ساخت آن می توان از Ibehavior ارث بری نماید ولی Yii به طور پیش فرض یک کلاس پایه با نام CBehaviorتعریف کرده است که کلیه موارد مورد نیاز را در خود دارد.

مثال :

```
classTimestampBehaviorextendsCActiveRecordBehavior
{
publicfunctionbeforeSave($event)
{
if($this->owner->isNewRecord)
$this->owner->create_time=time();
else
$this->owner->update_time=time();
}
}
```

Widget

یک ویگت از کلاس CWidget و یا زیر کلاسهای آن ارث بری می کند. یک مثال از ارث بری از زیر کلاسها مانند :

```
classMyTabViewextendsCTabView
```

```
{
publicfunctioninit()
{
if($this->cssFile===null)
{
$file=dirname(__FILE__).DIRECTORY_SEPARATOR.'tabview.css';
$this->cssFile=Yii::app()->getAssetManager()->publish($file);
}
parent::init();
}
}
```

و یا ساخت یک ویگت کاملا جدید مثل :

```
classMyWidgetextendsCWidget
{
protectedfunctionregisterClientScript()
{
// ...publish CSS or JavaScript file here...
$cs=Yii::app()->clientScript;
$cs->registerCssFile($cssFile);
$cs->registerScriptFile($jsFile);
}
}
```

نکته : سایر توسعه ها مانند اکشن ها ، فیلتر ها ، کنترلرها و ... در جای خود توضیح داده شده اند.

استفاده از کتابخانه های 3rd-Party

در زیر مثالی از نحوه استفاده از کامپوننت Zend_Search_Luceneمربوط به کتابخانه Zend Framework در Yii را مورد بررسی قرار می دهیم.

ابتدا توسعه مورد نظر را دریافت کرده و آن را در مسیر protected/vendors/Zend/Search از حالت فشرده خارج می کنیم. توجه کنید باید حتما فایل Lucene.php در این پوشه قرار داشته باشد.

در مرحله بعدی باید در ابتای یک فایل کلاس کنترلر کدهای زیر را می نویسیم :

```
Yii::import('application.vendors.*');
require_once('Zend/Search/Lucene.php');
```

کدهای بالا فایل Lucene.php را در برنامه import می کند.

پس از انجام مرحله فوق می توان توسط کد زیر در یک اکشن از توسعه مورد نظر استفاده کرد :

```
$lucene=newZend_Search_Lucene($pathOfIndex);
$hits=$lucene->find(strtolower($keyword));
```

تست

انواع روش تست :

۱- *unit testing* : بررسی می کند که آیا یک واحد کد به درستی عمل می کند یا خیر. در برنامه نویسی شی گرا یک واحد معمولا به یک کلاس اطلاق می شود. از این روش تست معمولا کسانی استفاده می کنند که کلاس ها را ایجاد می کنند.
۲- *functional testing* : بررسی می کند که آیا یک بخش از برنامه (مثلا قسمت ارسال پست ها در وبلاگ) به درستی عمل می کند یا خیر. از این روش معمولا توسعه دهندگان پروژه استفاده می کنند.

test-driven development (TDD)

مراحل TDD به شکل زیر است :

۱- ایجاد یک تست جدید برای یک ویژگی از برنامه. اجرای این تست و بررسی اینکه آیا با خطا مواجه می شود یا خیر.
۲- اجرای همگی تست ها و بررسی اینکه آیا تست جدید با خطا مواجه می شود یا خیر.
۳- اصلاح کدها تا اینکه خطای تست جدید از بین برود.
۴- اجرای همگی تست ها و بررسی صحت اجرای تمامی تست ها.
۵- Refactor کردن کدهای نوشته شده جدید و بررسی اینکه همچنان تست ها بدون خطا اجرا می شوند.

برای اجرای تست ها ما به برنامه های PHPUnit و Selenium Remote Control نیاز داریم.

وقتی توسط yiic webapp یک پروژه جدید ایجاد می شود فایلهای زیر نیز در پروژه ایجاد می شوند که مربوط به تست برنامه است:

```
testdrive/
   protected/                  containing protected application
files
      tests/                   containing tests for the
application
         fixtures/             containing database fixtures
         functional/           containing functional tests
         unit/                 containing unit tests
         report/               containing coverage reports
         bootstrap.php         the script executed at the very
beginning
         phpunit.xml           the PHPUnit configuration file
         WebTestCase.php       the base class for Web-based
functional tests
```

برای اجرای تست ها می توان کدهای زیر را نوشت :

```
% cd testdrive/protected/tests
% phpunit functional/PostTest.php    // executes an individual
test
% phpunit --verbose functional       // executes all tests
under 'functional'
% phpunit --coverage-html ./report unit
```

دستور آخر کلیه تست های داخل پوشه unit را اجرا می کند و گزارشی را در پوشه report ایجاد می کند.

نکته : برای اجرای تست ها حتما باید xdebug extension نصب و اجرا شده باشد تا بتواند گزارش ها را ایجاد نماید.

تست Bootstrap

محتوای این فایل به شکل زیر است :

```php
$yiit='path/to/yii/framework/yiit.php';
$config=dirname(__FILE__).'/../config/test.php';
require_once($yiit);
require_once(dirname(__FILE__).'/WebTestCase.php');
Yii::createWebApplication($config);
```

محتوای فایل protected\config\test.php به شکل زیر می باشد :

```
returnCMap::mergeArray(
require(dirname(__FILE__).'/main.php'),
array(
'components'=>array(
'fixture'=>array(
'class'=>'system.test.CDbFixtureManager',
),
/* uncomment the following to provide test database connection
            'db'=>array(
                'connectionString'=>'DSN for test database',
            ),
            */
),
)
);
```

اگر برنامه ما شامل یک یانک اطلاعاتی نیز باشد باید در کد بالا قسمت *'db'=>array* را از حالت توضیح خارج کرده و *'connectionString'* برنامه را وارد نماییم.

مدیریت آدرسها

ایجاد URL ها :

یک آدرس جدید می تواند به شکل زیر تعریف شود :

```
$url=$this->createUrl($route,$params);
```

که در آن $this به کنترلر جاری اشاره دارد و $route به مسیر مورد نظر اشاره می کند و $params شامل لیستی از پارامترهای نوع GET هستند که توسط ورودی از کاربر دریافت می شوند.

به طور پیش فرض آدرسها دارای ساختاری مانند زیر می باشند :

```
http://serverhost/index.php?r=post/read&id=100
```

این روش آدرس دهی کاربرپسند نیست و در آن حروف و کلمات نا آشنایی به کار رفته است. مثلا حرف r که مخفف route می باشد. برای رفع مشکل فوق ما باید urlManager را تنظیم نماییم به گونه ای که آدرسهای تولید شده مورد پسند کاربر واقع شوند.

پس از انجام این کار ما آدرسهای بهتری خواهیم داشت مثلا آدرس /post/100 به جای آدرس index.php/post/read/id/100/ مورد استفاده خواهد بود.

نحوه انجام این کار استفاده از تنظیمات داخل فایل config به شکل زیر است :

```
array(
......
'components'=>array(
......
'urlManager'=>array(
'urlFormat'=>'path',
'rules'=>array(
'pattern1'=>'route1',
'pattern2'=>'route2',
'pattern3'=>'route3',
),
),
),
                                                                    );
```

در آرایه rules ما مشخص می کنیم که چه route باید به چه pattern ترجمه شود.

در شکل کاملتر و پیشرفته تر یک مسیر می تواند به شکل زیر تعریف شود :

```
'pattern1'=>array('route1', 'urlSuffix'=>'.xml',
'caseSensitive'=>false)
```

urlSuxادامه های آدرس را مشخص می کند که به طور پیش فرض null است. به عنوان مثال ما می توانیم داشته باشم post/100.html/که به جای post/100/ به کار می رود و شاید برای ما بیشتر به یک آدرس اینترنتی شبیه باشد و یا بیشتر آن را بپسندیم. البته برای این کار باید مقدار urlSux را برابر مقدار مورد نظر قرار دهیم.

caseSensitive مشخص می کند که آیا این آدرس به کوچک و بزرگ بودن حروف حساس است که به طور پیش فرض برابر null است.

.

مخفی کردن index.php

برای اینکه در آدرس دهی های ما نیازی به نوشتن index,php نداشته باشیم باید تنظیماتی را بر روی سرور Apache انجام دهیم. می توانیم این تنظیمات را در فایل htaccess. خود نیز قرار دهیم این فایل باید در مسیر اصلی در کنار index.phpاصلی قرار داشته باشد. محتوای آن به شکل زیر است :

```
RewriteEngine on

# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# otherwise forward it to index.php
RewriteRule . index.php
```

حال باید در فایل config مقدار ویژگی showScriptName را در کامپوننت urlManager برابر false قرار دهیم.

حال ما می توانیم برای دسترسی به $this->createUrl('post/read',array('id'=>100)) از آدرس post/100/ استفاده نماییم.

مثال۱ : ساخت برنامه Login

ساخت database با نام db_demo

Import کردن schema زیر در آن:

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";

CREATE TABLE IF NOT EXISTS `tbl_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(128) COLLATE utf8_persian_ci NOT NULL,
  `password` varchar(128) COLLATE utf8_persian_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_persian_ci
AUTO_INCREMENT=24 ;

INSERT INTO `tbl_user` (`id`, `username`, `password`) VALUES
(22, 'ali', '123'),
(23, 'hasan', '456');
```

باعث ایجاد جدولی به شکل زیر می شود :

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	id	int(11)			No	None	auto_increment
☐	username	varchar(128)	utf8_persian_ci		No	None	
☐	password	varchar(128)	utf8_persian_ci		No	None	

			id	username	password
☐	✎	✕	22	ali	123
☐	✎	✕	23	hasan	456

ساخت یک پروژه جدید yii با نام demo با دستور yiic webapp demo

فعال کردن gii در فایل config

تعیین بانک اطلاعاتی mysql با تنظیمات زیر :

```
        'db'=>array(
                'connectionString' =>
'mysql:host=localhost;dbname=db_demo',
                'emulatePrepare' => true,
                'username' => 'root',
                'password' => '',
                'charset' => 'utf8',
        ),
```

ورود به Model generator و ساخت مدل زیر :

Table Prefix

[empty]

Table Name *

tbl_user

Model Class *

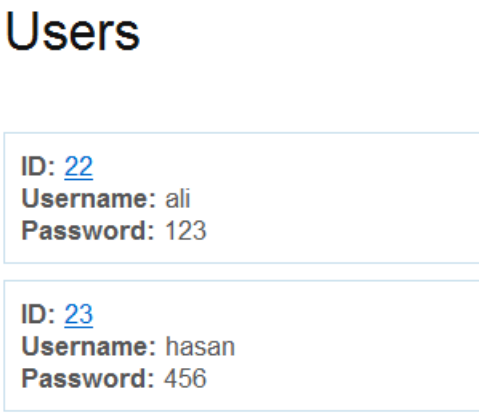
User

Base Class *

CActiveRecord

ورود به Crud generator و ساخت کنترلر زیر :

مشاهده آدرس http://localhost/demo/index.php?r=user و مشاهده جدول



رفتن به مسیر C:\xampp\htdocs\demo\protected\components و ویرایش فایل UserIdentity.php به شکل زیر :

```
<?php
class UserIdentity extends CUserIdentity
{
    private $_id;
    public function authenticate()
    {
        $username=strtolower($this->username);
        $user=User::model()-
>find('LOWER(username)=?',array($username));
        if($user===null)
            $this->errorCode=self::ERROR_USERNAME_INVALID;
        else if(!$user->validatePassword($this->password))
            $this->errorCode=self::ERROR_PASSWORD_INVALID;
        else
        {
            $this->_id=$user->id;
            $this->username=$user->username;
            $this->errorCode=self::ERROR_NONE;
```

```
        }
        return $this->errorCode==self::ERROR_NONE;
    }
    public function getId()    // overrided
    {         return $this->_id;     }
}
```

رفتن به مسیر C:\xampp\htdocs\demo\protected\models و ویرایش فایل Use.php و اضافه کردن متد زیر به آن :

```
public function validatePassword($password)
    {
        return ($this->password == $password);
    }
```

باز کردن آدرس زیر و ورود به سایت با حساب ali/123

http://localhost/demo/index.php?r=site/login

اگر بخواهیم اکشن login را برای کنترلر Users فعال کنیم باید به شکل زیر عمل کنم :

به آدرس C:\xampp\htdocs\demo\protected\views\site رفته و فایل login.php را کپی می کنیم. سپس به آدرس C:\xampp\htdocs\demo\protected\views\user رفته و آن را paste می کنیم.

به فایل کنترلر SiteController.php رفته و اکشن login را کپی می کنیم و آن را در انتهای فایل کنترلر user یعنی فایل UserController.php قرار می دهیم.

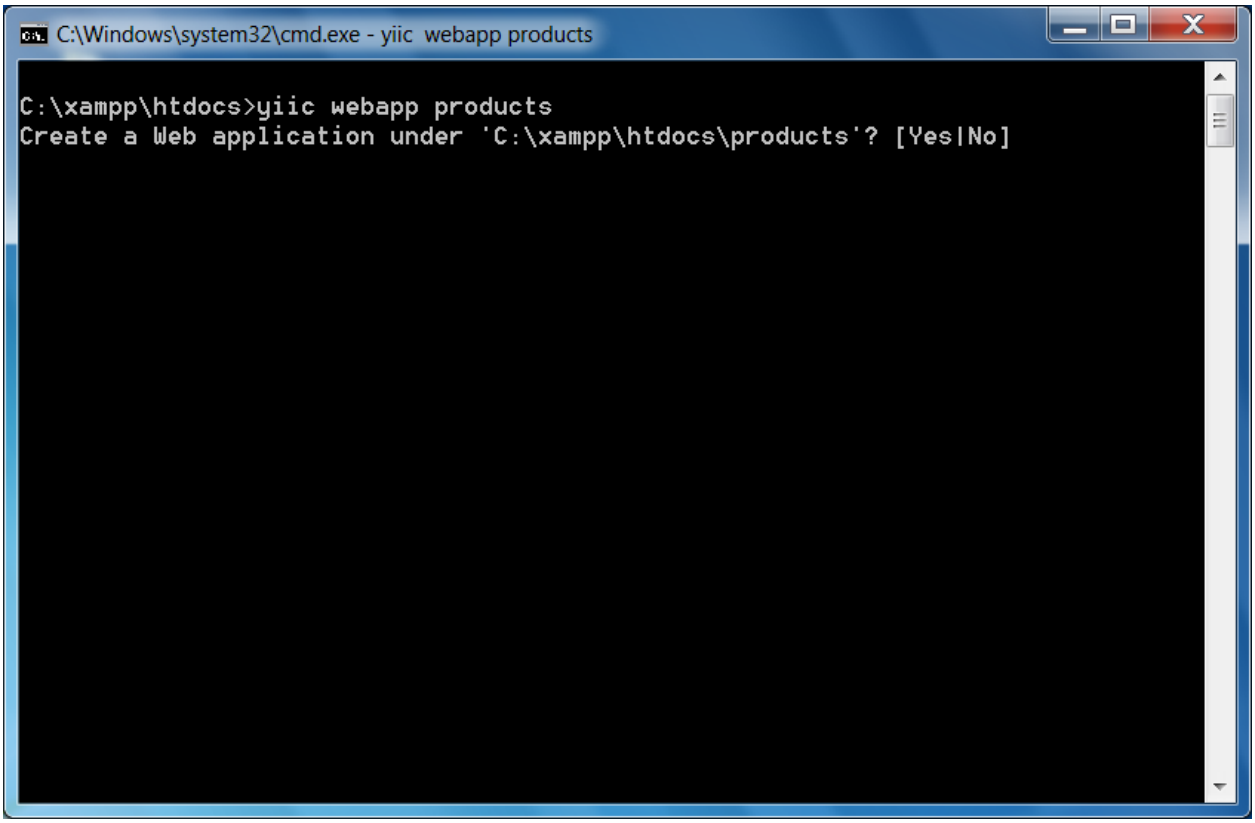
حتما باید در فایل کنترلر UserController.php متد accessRules را ویرایش نماییم و اجازه دسترسی را یه همه بدهیم یا کلا این متد را حذف نماییم.

حال با وارد کردن آدرس زیر می توانیم به صفحه ورود کاربر توسط کنترلر User برویم:

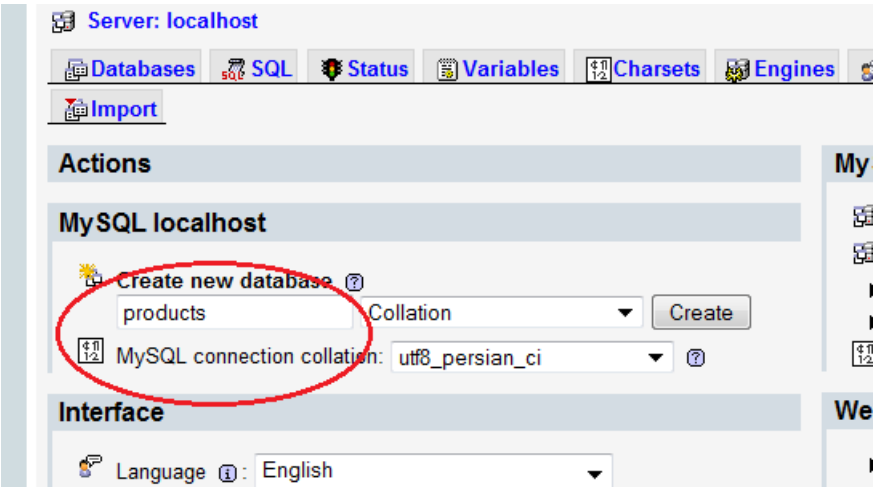
http://localhost/demo/index.php?r=User/login

مثال۲ : ساخت برنامه Register

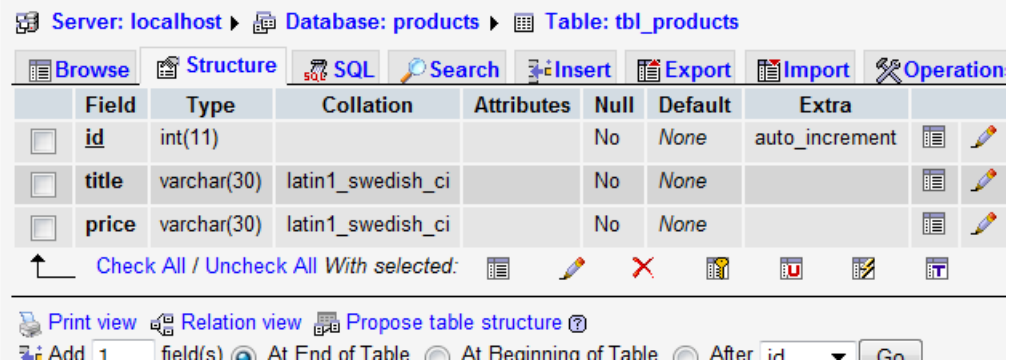
ساخت پروژه جدید با نام products



ساخت بانک اطلاعاتی با نام products توسط PhpMyAdmin



ساخت یک جدول با نام tbl_products

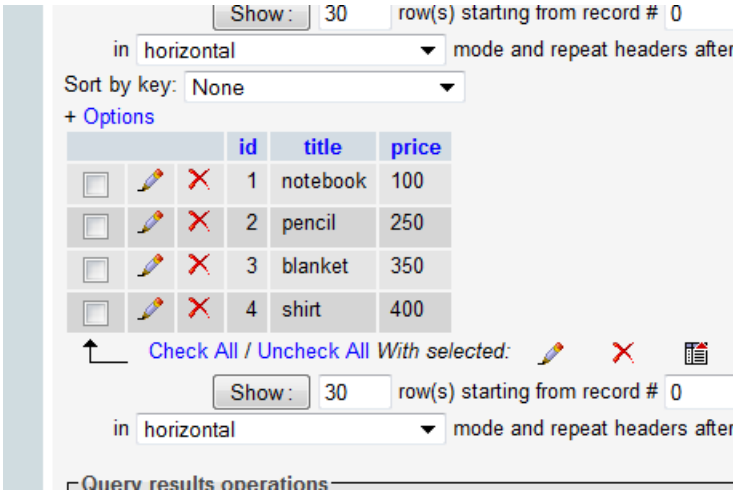


کد SQL ساخت این جدول به شکل زیر است

```
CREATE TABLE IF NOT EXISTS `tbl_products` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(30) NOT NULL,
  `price` varchar(30) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=8 ;
```

می توان این کد را مستقیما نیز در PhpMyAdmin وارد نمود تا جدول ایجاد شود.

وارد کردن مقادیر پیش فرض در جدول



ویرایش فایل products\protected\config\main.php

```
    'modules'=>array(
```

```
        'gii'=>array(
            'class'=>'system.gii.GiiModule',
            'password'=>'123',
            // If removed, Gii defaults to localhost only.
Edit carefully to taste.
            'ipFilters'=>array('127.0.0.1','::1'),
        ),
    ),
```

و همچنین :

```
        /*
        'db'=>array(
            'connectionString' =>
'sqlite:'.dirname(__FILE__).'/../data/testdrive.db',
        ),
        */
        'db'=>array(
            'connectionString' =>
'mysql:host=localhost;dbname=products',
            'emulatePrepare' => true,
            'username' => 'root',
            'password' => '',
            'charset' => 'utf8',
        ),
```

باز کردن آدرس http://localhost/products/index.php?r=gii

وارد کردن رمز ۱۲۳ و ورود

ساخت مدل

Table Prefix

[empty]

Table Name *

tbl_products

Model Class *

ProductsModel

Base Class *

CActiveRecord

Model Path *

application.models

Code Template *

default (C:\xampp\htdocs\yii\framework\gii\generators\model\templates\default)

Preview Generate

Code File	Generate
models\ProductsModel.php	new ☑

ساخت کنترلر و ویو ها

Model Class *

ProductsModel

Controller ID *

products

Base Controller Class *

Controller

Code Template *

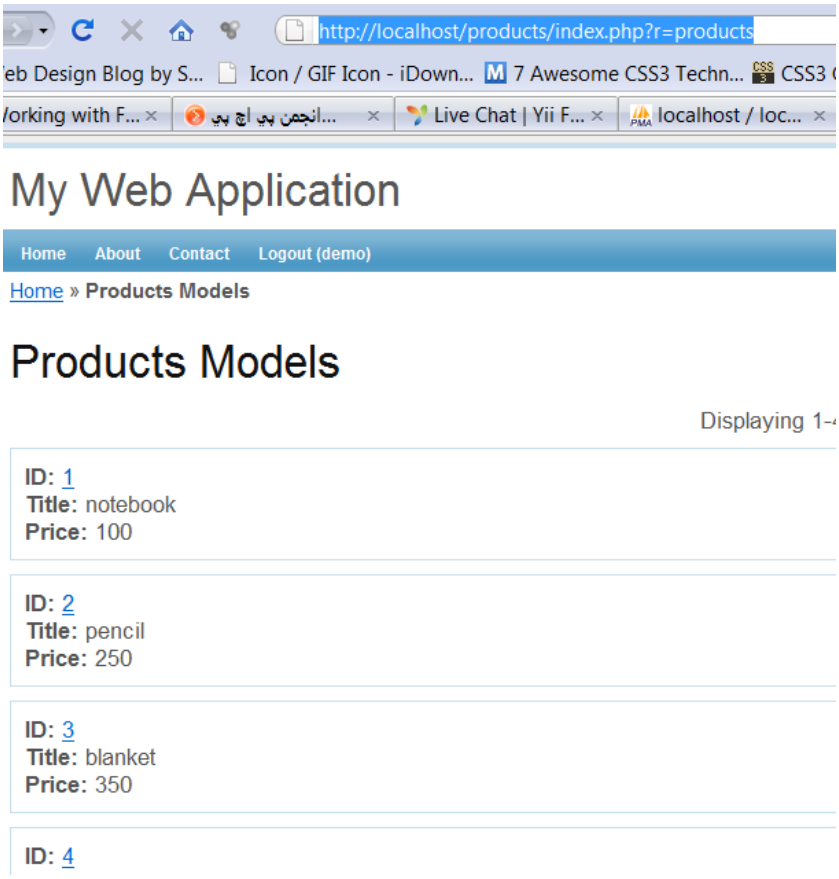
default (C:\xampp\htdocs\yii\framework\gii\generators\crud\templates\default)

Preview Generate

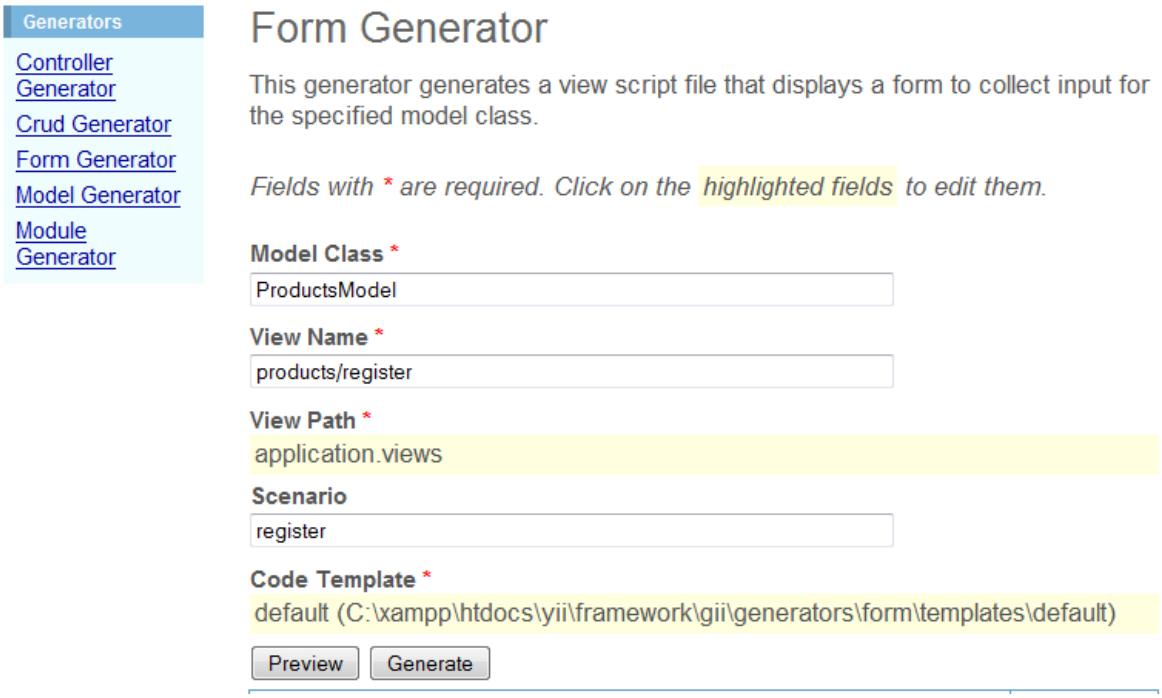
Code File	Generate ☑
controllers\ProductsController.php	new ☑
views\products_form.php	new ☑
views\products_search.php	new ☑
views\products_view.php	new ☑
views\products\admin.php	new ☑

بررسی صحت برقراری ارتباط با بانک اطلاعاتی توسط بازکردن آدرس زیر

http://localhost/products/index.php?r=products



ساخت فرم جدید برای ثبت کالای جدید با نام register



پس از زدن کلید generate کد زیر ظاهر می شود که آن را کپی می کنیم :

Preview

The form has been generated successfully.

You may add the following code in an appropriate controller class to invoke the view:

```php
<?php
public function actionRegister()
{
    $model=new ProductsModel('register');

    // uncomment the following code to enable ajax-based validation
    /*
    if(isset($_POST['ajax']) && $_POST['ajax']==='products-model-register-
form')
    {
        echo CActiveForm::validate($model);
        Yii::app()->end();
    }
    */
```

عمل کپی :

```php
<?php
public function actionRegister()
{
    $model=new ProductsModel('register');

    // uncomment the following code to enable ajax-based validation
    /*
    if(isset($_POST['ajax']) && $_POST['ajax']==='products-model-register-
form')
    {
        echo CActiveForm::validate($model);
        Yii::app()->end();
    }
    */

    if(isset($_POST['ProductsModel']))
    {
        $model->attributes=$_POST['ProductsModel'];
        if($model->validate())
        {
            // form inputs are valid, do something here
            return;
        }
    }
    $this->render('register',array('model'=>$model));
}
```

کد کنترلر را در مسیر products\protected\controllers\ProductsController.php ویرایش می کنیم. و کد بالا را در آن paste می کنیم.

در همین فایل کنترلرمتد accessRules را اصلاح می کنیم :

```
array('deny',  // deny all users
'allow'=>array('*'),
    ),
```

کد های اکشن register را که paste کرده بودیم به شکل زیر اصلاح می کنیم :

```
public function actionRegister()
{
$model=new ProductsModel('register');
if(isset($_POST['ProductsModel']))
{
    $model->attributes=$_POST['ProductsModel'];
    if($model->save()) {$this->redirect(array('view','id'=>$model->id));}
}
$this->render('register',array('model'=>$model));
}
```

توضیح کدها :

اولا تمامی کارهای قبلی به صورت خودکار توسط Gii انجام شد . یعنی تمامی فایلهای مدل – کنترلر – ویوها – فرم و کدهای مربوط به ساخت اکشن فرم و ...همگی به طور خودکار تولید شد. در این مرحله کاربر باید کدهای متد Register را خود ویرایش نماید که توضیح آنها در زیر آمده است.

```
public function actionRegister()
```

کد مربوط به شروع متد register

```
$model=new ProductsModel('register');
```

این کد یک نمونه از مدل ProductsModel میسازد و سیاست این فرم را برابر register قرار می دهد. یعنی همان چیزی که در حین ساخت فرم به عنوان سناریو مشخص نمودیم.

البته می توان سناریو را حذف نمود و در اجرای صحیح پروژه مشکلی ایجاد نخواهد شد. یعنی به شکل زیر:

```
  $model=new ProductsModel();
```

دستور بعدی :

```
    if(isset($_POST['ProductsModel']))
      {    //start if block
```

در این شرط ما بررسی می کنیم که آیا محتوای فرم توسط کاربر پر شده است یا خیر و بلوک if زمانی اجرا می شود که کاربر اطلاعات صحیح و کامل را داخل فرم وارد کرده باشد وگرنه کد بعد از if اجرا می شود که کد زیر است.

```
    }    //end if block
    $this->render('register',array('model'=>$model));
```

و به واسطه این دستور ویو register نمایش داده می شود که در واقع همان فایل فرم ما بود که آن را توسط gii ساخته بودیم.

ملاحظه می گردد که با باز کردن آدرس http://localhost/products/index.php?r=products/Register کابر فرم ورود اطلاعات را مشاهده می کند و این به آن معنی است که بلوک if اجرا نشده و ویو فرم register که خارج از if است اجرا شده. لذا می توان کد بالا را به شکل زیر هم نوشت :

```
    if(isset($_POST['ProductsModel']))
    {
        ---block if  codes ---
    }
else
 {      $this->render('register',array('model'=>$model));    }
```

دستور بعدی :

```
        $model->attributes=$_POST['ProductsModel'];
```

این دستور باعث می شود که کلیه مقادیر وارد شده در فرم به صورت یکجا (massive) وارد شی $model می شوند. وارد کردن مقادیر به صورت یکجا برای موارد امنیتی بسیار پر کاربرد می باشد. و جلوی دسترسی کاربر به تک تک مقادیر فیلد را می گیرد که درجلوگیری از هک کردن فرم ها بسیار موثر است.

کد بعدی :

```
if($model->save()) {$this->redirect(array('view','id'=>$model->id));}
```

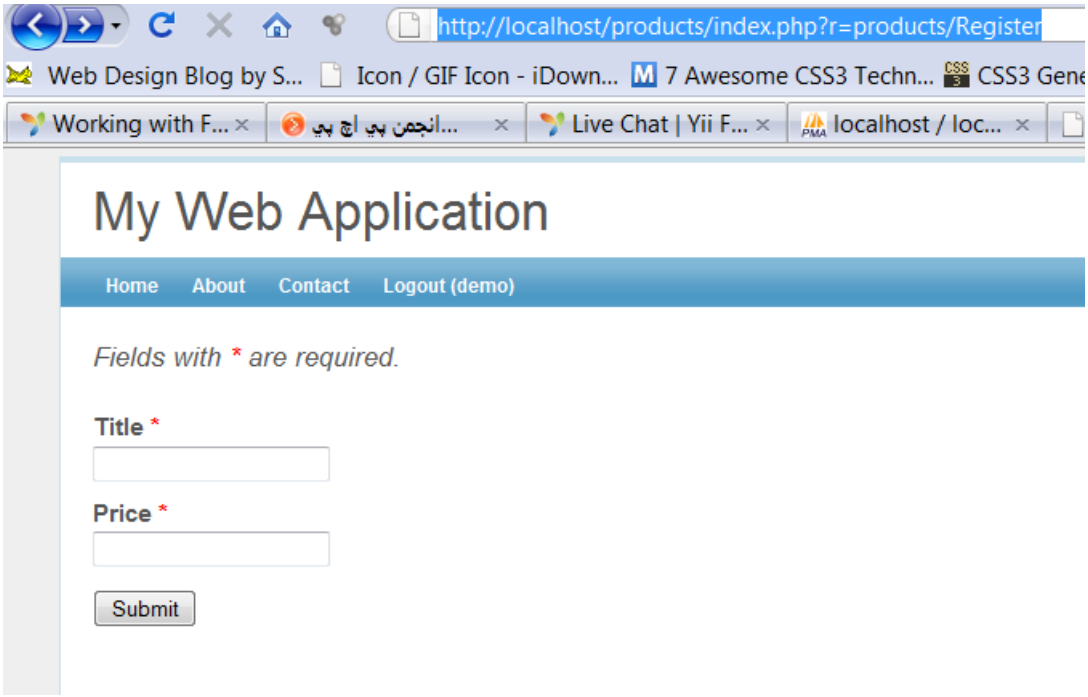
اصلی ترین کد شاید کد بالا باشد. که متد save را در $model اجرا می کند. متد save از کلاس CModel به کلاس RegisterModel ما به ارث رسیده است پس نباید دنبال آن در فایل مدل خودمان بگردیم. این دستور بررسی می کند که متد save برای $model اجرا شود و اگر به درستی اجرا شد آنگاه ویو view را به کاربر

نشان می دهد. در ضمن پارامتر id را نیز برای این ویو می فرستد که فقط مشخصات همین رکورد با همین id نشان داده شود و نه دیگر رکوردها. می توان برای این if یکelse هم نوشت که در صورت ناموفق بودن عمل save یک پیغام خطای مناسب را نمایش دهد.

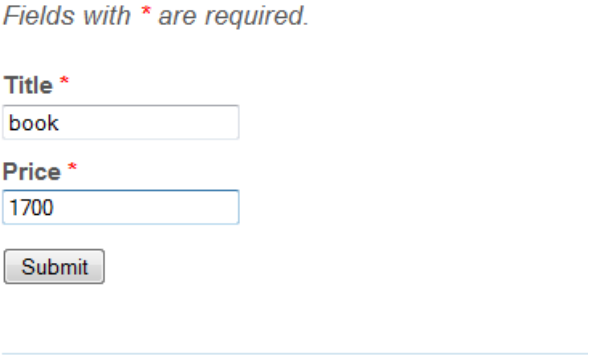
اجرای پروژه. حال که همه چیز درست انجام شد آدرس زیر را در مرورگر باز می کنیم :

http://localhost/products/index.php?r=products/Register

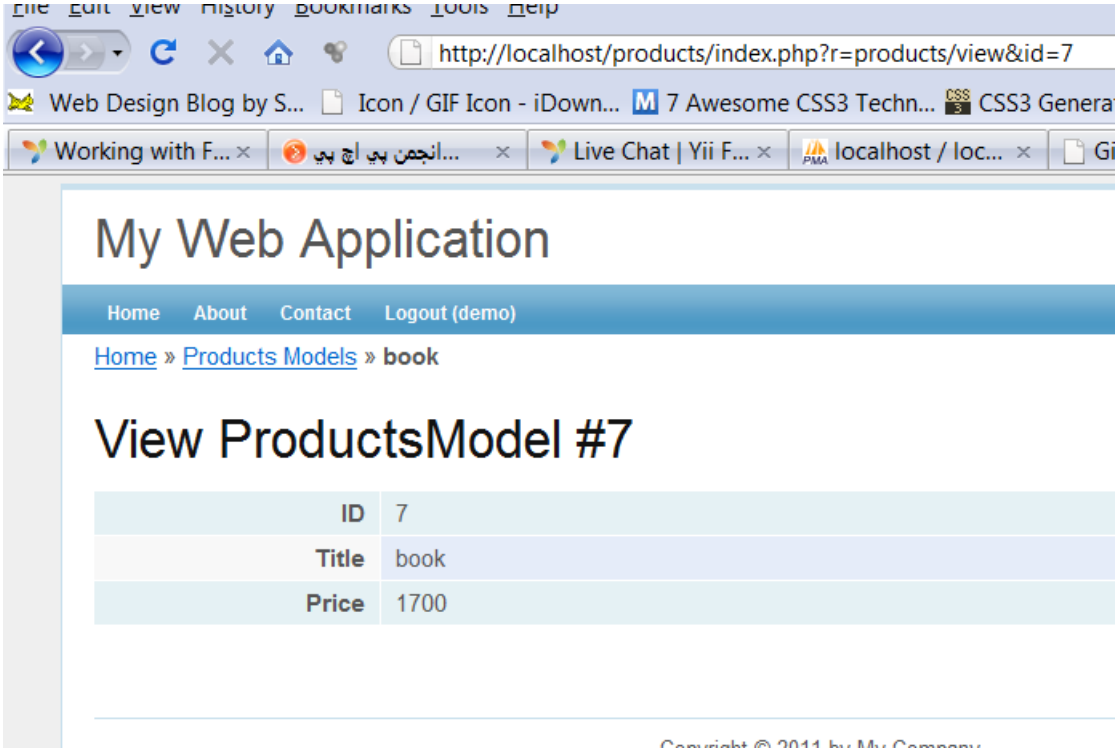
این آدرس متد register در کنترلر products را اجرا می نماید که نتیجه آن باید مانند شکل زیر باشد :



حال فرم را با دو مقدار امتحان می کنیم مثلا :



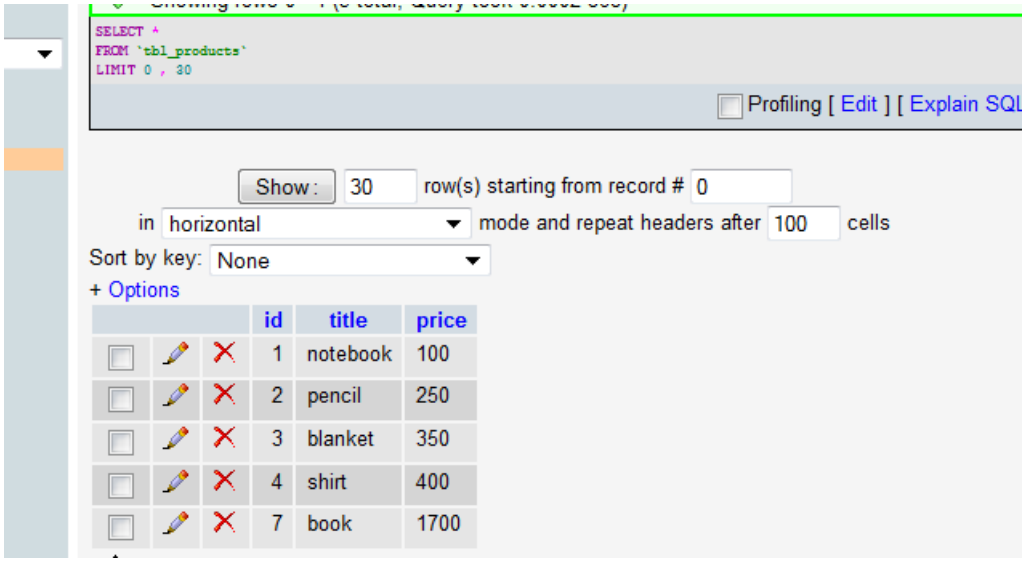
پس از زدن دکمه submit باید به صفحه زیر منتقل شویم :



مشاهده می شود که چگونه پس از render شدن view پارامترid و مقدار آن هم برای این صفحه ارسال شده است.

http://localhost/products/index.php?r=products/view&id=7

برای بررسی صحت ثبت رکورد وارد شده در PhpMyAdmin محتوای جدول را نگاه می کنیم :



مشاهده می شود که آخرین رکورد همان رکوردی است که ما وارد کرده ایم.

نکته : فیلد id که در جدول به کار رفته بود به دلیل این که کلید اصلی جدول بود در فرم ما نمایش داده نشد لذا د تولید فرم باید دقت کرد که فیلد کلید اصلی در فرم ظاهر نمی شود.

ضمیمه ها

رمزگذاری

در بسیاری از موارد نیاز به رمز گذاری وجود دارد. به عنوام مثال زمانی که می خواهیم رمز ورود را در بانک اطلاعاتی ثبت نماییم. یک روش رمز گذاری MD5 نام دارد که یک روش one-way به معنی یک طرفه است یعنی امکان رمز گشایی آن به طور مستقیم وجود ندارد. برای ثبت رمز ورود کد گذاری شده در بانک اطلاعاتی به شکل زیر عمل می کنیم :

```
Public function beforeValidate() {
     $this->password = $this->encript($this->password);
}
Public function encrypt($value) {
     Return MD5($value);
}
```

پس از استفاده از این روش مثلا در فرم به روزرسانی نمی توان رمز ورود را مشاهده و تغییر داد هرچند می توان از آن برای مقایسه صحیح بودن رمز ورود وارد شده استفاده کرد.

فعال سازی کوکی های مرورگر

در صورتی که کوکی های مرورگر کاربر فعال باشند مقادیر وارد شده در فیلد هایی که کاربر پر می کند در حافظه باقی می مانند. از این روش مثلا در قسمت Remember Me برای مشخصات ورود کاربر می توان بهره گرفت. اگر فایل config را مشاده کنیم با دستور زیر مواجه می شویم :

```
     // application components
     'components'=>array(
          'user'=>array(
               // enable cookie-based authentication
               'allowAutoLogin'=>true,
          ),
```

که توسط آن ذخیره سازی مقادیر با true فعال یا با false غیر فعال می شوند. حال در مدل ورود کاربر کل زیر مشاهده می شود که شامل تنظیمات پیشرفته تری مثل مدت زمان باقی ماندن کوکی ها وجود دارد :

```
if($this->_identity->errorCode===UserIdentity::ERROR_NONE)
          {
$duration=$this->rememberMe ? 3600*24*30 : 0; // 30 days
Yii::app()->user->login($this->_identity,$duration);
                return true;
          }
```

فضانام ها و آدرس ها

مسیر های مستعار

در خیلی از جاهای برنامه که نیاز به آدرس جایی داریم می توانی از Path Alias استفاده کنیم.مثال :

به جای Protected/Controller/Post/Update

از این آدرس استفاده می کنیم : application.Controller.Post.Update

در آدرس دهی Alias به جای / از علامت . استفاده می شود.

در هر جای از برنامه می توان به وسیله YiiBase::getPathOfAlias() آدرس alias را به آدرس واقعی آن تبدیل کرد.

به عنوان مثال system.web.CController به yii/framework/web/CController ترجمه می شود.به وسیله YiiBase::setPathOfAlias() نیز می توان یک root alias جدید را برای برنامه مشخص کرد. در ماژول ها root alias به آدرس همین ماژول اشاره می کند و نه به آدرس Application

چند کلمه کلیدی زیر که می تواد در مسیرهای Alias از آنها استفاده نمود.

```
System   ->  Yii Framework Directory
Zii -> Zii Lirary Path
Application -> base directory = path to protected folder
Webroot -> entry scripts directory
Ext -> third-party extensions directory
```

Import کردن کلاس ها :

تنها کلاسهای خارجی که خارج از component ها هستند و توسط Yii پیش تعریف نیستند را باید import کرد. و سایر کلاسها نیازی به import کردن ندارند.

```
Yii::import("system.web.CController");
```

Import معادل include , require نیست هرچند کار مشابهی را انجام می دهد. سرعت import بیشتر است و بهرهبری بیشتری دارد.

Import directory

می توان مسیر یک دایرکتوری را import کرد تا هر زمان که خواستیم کلاسی از آن دایرکتوری را فراخوانی و import کنیم تنها با صدا زدن نام آن کلاس این امر امکان پذیر شود.

```
Yii::import("system.web.* ");
```

همه کتابخانه های Yii با حرف C شروع می شوند که ابتدای کلمه Class است.بهتر است کلاسهای کاربر با حرفی غیر از C شروع شوند تا تداخلی ایجاد نشود.

استفاده از Class Map

می توان تعدادی از کلاسها را از قبل تعریف کرد و دیگر نیازی به import ندارد و در هر جای برنامه می توان از آنها استفاده کرد. براس انجام این کار باید کلاسهای مورد استفاده قبل از CWebApplication::run() کلاسها را تعربف کرد.مثال :

```
Yii::$classMap=array(
    'ClassName1' => 'path/to/ClassName1.php',
    'ClassName2' => 'path/to/ClassName2.php',
    ......
);
```

Current Path:

Sample : C:\xampp\htdocs\y1\protected\views\site

Getter: echo dirname(__FILE__);

Setter: It returns the current path where this function calls.

Base Directory:

Sample : C:\xampp\htdocs\y1\protected

Getter : echo Yii::app()->basePath; / anywhere

Setter : Config File (Application) : 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..'

یک مسیر به طور کلی به شکل زیر تعریف می شود :

```
http://hostname.com/index.php?r=ModuleID/
ControllerID/actionID/Parm1/Value1/Parm2/Value2/...
```

در آدرس بالا حرف r معرف کلمه route است.

اگر برنامه شامل هیچ ماژولی نباشد می توان ModuleID را ننوشت.با اعمال تنظیمات در Application برنامه و برای SEO بهتر میتوان مسیر بالا را به شکل زیر اصلاح نمود :

```
http://hostname.com/index.php/
ModuleID/ControllerID/actionID/Parm1/Value1/Parm2/Value2/...
```

الگو های نام گذاری

الگوی حروف کوچک و بزرگ برای درخواست به شکل زیر است:

http://hostname/index.php?r=ControllerID/ActionID

درخواست به حروف کوچک و بزرگ حساس می باشد.

نامگذاری متغیرها – توابع – کلاسها به شکل camelCase انجام می شود. مثال $basePath, runController(), LinkPager

حرف اول متغیرها کوچک باشد.

کلاسها حرف اول بزرگ و برای کلاسهای داخلی با حرف C کلاسهای کاربر یک حرف به غیر از C

برای کلاسها و اعضای private حتما حرف اول با _ شروع شود مثل view_

نام فایل کنترلرها باید شامل کلمه Controller باشد مثل siteController.php ولی دسترسی به این کلاس بدو کلمه Controller است مثل : index.php?r=site که باعث ایجاد امنیت بیشتر می شود زیرا که نام اصلی فایل را از کاربران مخفی نگه می دارد.

نام فایل کلاس باید با نام کلاس باید یکی باشد مثلا کلای CController در فایل CController.php قرار گیرد.

نام فایل ویو باید با نام فایل آن یکی باشد مثلا ویو edit نام کلاس آن edit و نام فایل آن edit.php.

نام جداول و فیلدهای بانک اطلاعاتی باید با حروف کوچک باشد.

جداول و فیلدهایی که چند کلمه ای هستند باید با _ از هم جدا شوند مثل tbl_mytable و یا first_name

نام جداول بهتر ست با پیشوند tbl_ شروع شود مثل tbl_table1

در نام جداول بهتر است از کلمات مفرد به جای جمع استفاده کرد مثلا به جای tbl_users از عنوان tbl_user استفاده شود. برای نام گذاری فیلد ها هم به همین شکل.

Configuration

Configuration یک آرایه است که پارامترهایی را مشخص می کند :

```
array('name'=>'My application', 'basePath'=>'./protected')
```

در هر جای برنامه به این شکل انجام می شود.

نکته : اگر پارامتری تعریف نشود مقدار پیش فرض آن در نظر گرفته می شود.

چند نام دایرکتوری مهم :

نام	Alias	Path	access
application base directory	application	WebRoot/protected	CWebApplication::basePath
WebRoot		WebRoot/protected/runtime	.CApplication::runtimePath
Extension		WebRoot/protected/extensions	CApplication::extensionPath
modules		WebRoot/protected/modules	
controllers		WebRoot/protected/controllers	CWebApplication::controller

Path			
CWebApplication::viewPath	WebRoot/protected/views		views
CController::viewPath	WebRoot/protected/views/ControllerID		Views/ControllerID
CWebApplication::layoutPath	WebRoot/protected/views/layouts		Layouts
CWebApplication::systemViewPath	WebRoot/protected/views/system		Views/system
CAssetManager::basePath	WebRoot/assets		Assets
CThemeManager::basePath	WebRoot/themes		themes

Class Map : این امکان در yii وجود دارد که یک سری از کلاسها از قبل تعریف شوند بدون اینکه در برنامه نیاز به تعریف مجدد داشته باشند که از این روش خود yii نیز استفاده می کند. مثال :

```
Yii::$classMap=array(
    'ClassName1' => 'path/to/ClassName1.php',
    'ClassName2' => 'path/to/ClassName2.php',
    ......
);
```

Automatic Code Generation

Yii دارای ابزاری به نام Gii می باشد که بوسیله آن می تواند کدها را ایجاد نماید. این ابزار تحت وب می باشد و از طریق مرورگر قابل اجراست. Gii ابزاری است که تحت یک ماژول ارائه شده است و برای استفاده از آن باید ابتدا در فایل config تنظیمات زیر را داشته باشیم :

```
return array(
    ......
    'modules'=>array(
        'gii'=>array(
            'class'=>'system.gii.GiiModule',
            'password'=>'pick up a password here',
            // 'ipFilters'=>array(...a list of IPs...),
            // 'newFileMode'=>0666,
            // 'newDirMode'=>0777,
        ),
    ),
```

```
);
```

نکته : کدهای فعال سازی Gii به طور خودکار در ابتدای ساخت پروژه وجود دارند ولی به صورت Comment می باشند که برای استفاده از آن باید این بلوک کد را از حالت توضیحات خارج نمود.

برای ایجاد محافظت در استفاده از Giid یک رمز ورود قرار داده شده است.

به طور پیش فرض Gii برای استفاده بر روی Localhost ایجاد شده است و اگر بخواهیم آن را در جای دیگر استفاده کنیم باید پارامتر GiiModule::ipFilters را برابر IP مورد نظر تعریف نماییم.

از آنجایی که ابزار Gii فایلها و پوشه هایی را ایجاد می نماید باید مجوز استفاده از فضای سرور داده شده باشد که توسط دو پارامتر [GiiModule::newFileMode] و[GiiModule::newDirMode] این کار انجام می شود.

نکته : از ابزار Gii تنها در زمان طراحی سایت باید استفاده شود چون که این ابزار جهت توسعه است. لذا در هنگام راه اندازی نهایی سایت باید دسترسی به gii غیر فعال گردد.

پس از انجام تنظیمات توسط آدرس http://hostname/path/to/index.php?r=gii می توان به Gii دسترسی پیدا کرد.

نکته : می توان Gii را به طور دلخواه سفارشی کرد تا کد هایی که تولید می کند دارای ظاهر یا ستختاری به دلخواه ما باشند. حتی می توان آن را به گونه ای گسترش داد تا کامپوننت ها و سایر اجزای مورد نیاز ما را به شکل دلخواه تولید نماید.

جلسه ها

تعریف یک Session

```
Yii::app()->session->add('item','value');
```

فراخوانی یک Session

```
echo   Yii::app()->session['item']
```

فایل Controller.php

این فایل در مسیر زیر قرار دارد :

```
\protected\components
```

همه کنترلرهای پروژه از این کلاس ارث بری می کنند بنابر این محل مناسبی برای تعریف متغیر های مهم و ثوابت است.

در این فایل به طور پیش فرض سه متغیر تعریف شده اند که عبارتند از :

```
public $layout='//layouts/column1';
```

قالب پیش فرض را برای همه کنترلر ها تعریف می کند.

```
public $menu=array;()
```

متغیر $menu را تعریف می کند که برای تعریف منو ها در ویو به کار می رود.

```
public $breadcrumbs=array;()
```

متغیری برای تعریف مسیر جاری تعریف می کند.

مشکل TimeZone

در بعضی سیستم عامل ها ممکن است تنظیمات زمانی متفاوت باشد که برای حل آن در فایل config به شکل زیر کدها را اضافه می کنیم :

```
<?php
return array(
     'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
     'timezone'=>'Asia/Tehran',
```

چند زبانی

برای اضافه کردن یک زبان ابتدا پوشه مربوط به آن زبان را از داخل framework به داخل پوشه protected\messages\ کپی می کنیم مثل پوشه de یا ar که همگی در مسیر framework\messages\ قرار دارند. حال در فایل config کد زیر را اضافه می کنیم :

```
<?php
return array(
     'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
     'language' => 'de',
```

همچنین

```
 //application components
'components'=>array(
'coreMessages'=>array'(basePath'=>'protected/messages,) ) ,'
```

پارامترهای برنامه

می توان پارامتر هایی را در فایل config تعریف نموده و در طول برنامه از آنها استفاده کرد مثل آدرس ایمیل مدیر به شکل زیر :

```
     'params'=>array(
          'adminEmail'=>'web.ebox@gmail.com',
     ),
```

حال در هر جای برنامه می توان توسط کد زیر به پارامتر مورد نظر دسترسی پیدا کنیم :

```
echo Yii::app()->params['paramNam’];
```

منابع :

- Agile Web Application Development with Yii1.1 and PHP5
- yii-1-1-application-development-cookbook
- The Definitive Guide to Yii
- آموزش های آقای لری اولمن
- آموزش ساخت وبلاگ