欢迎转载 留个这里的链接就成，任何建议请发至 zhengkanjie @ gmail.com，你的建议是我的动力， 谢谢

基本概念

Section1 简介

本章节篇幅很短，简单的介绍一下 MVC 概念以及在 cake 中是如何实现的。如果你对 MVC 模式很陌生，这个章节将会非常适合你。我们由常规的 MVC 概念开始我们的讨论，然后用我们的方式在 CakePHP 中使用 MVC 模式，并且展示几个简单的示例。

Section 2 MVC 模式

[TODO 考虑到 MVC 模式是必需过桥技能，所以建议还是先看一下相关文章，原文很简单，最后补上望海涵]

Section 3 Cake 目录结构一览

当你解压缩 Cake 后，会发现有 3 个主目录

- app
- cake
- vendors

cake 目录下是核心 lib，通常情况下你不要接触它们。

app 目录下是你的程序文件所存放的位置。将 app 目录和 cake 目录分开，从而可以使多个应用共享一个 cake 类库。同样的也使得 CakePHP 的升级变得很容易：你只需要下载最新的版本并且覆盖原来的类库就可以了。不用担心会覆盖掉你的应用程序文件。

你可以使用 vendors 目录来存放一些第三方类库。后面的章节将会讲到更多关于 vendors 的内容，一个最基本的概念就是你可以非常方便的通过 vendor()方法来调用 vendor 类库。

让我们来看一下完整的目录结构：

```
/app
    /config              - 配置文件目录，包括 Database, ACL 等

    /controllers       - Controllers 文件
        /components   - Components 文件
```

```
    /index.php          - 允许你将 app 目录部署为 DocumentRoot (译注：参见 Apache 相关配置)

    /models             - Model 文件

    /plugins            - Plugins 文件

    /tmp                - Cache 和日志存放处

    /vendors            - 你的应用中使用到的第三方类库

    /views              - 视图文件
        /elements       - 视图元素文件
        /errors         - 自定义错误页面
        /helpers        - Helpers 文件
        /layouts        - 页面布局文件
        /pages          - 静态页面文件

    /webroot            - web 根目录
        /css
        /files
        /img
        /js

/cake                   - 核心类库，请不要随意修改任何文件，除非你确信你有这个能力

index.php

/vendors                - 服务器端的第三方类库
```

安装 CakePHP

Section 1 简介

现在你已经了解了每一件事情，包括系统的结构和整个类库的目的，或者说你略过了上述章节，因为你对这些资料并不感兴趣只是希望快点开始工作。Anyway 你已经摩拳擦掌准备完毕了。

这个章节来介绍了哪些是必须被安装到服务器上的，不同的配置方案，如何下载并安装 CakePHP，访问默认页面，以及一些 troubleshooting 以备一些意外情况的发生。

Section 2 安装必需

[TODO 一些译者认为比较基础的内容暂时优先级设为低 完美版释出时会补上]

Section 3 安装 CakePHP

[TODO]

Section 4 设置 CakePHP

第一种设置方案仅推荐使用在开发环境下，因为它不是最安全的方案。第二种方案考虑了更多的安全性问题因而可以放心的使用在生产环境下。

注意：/app/tmp 目录必须具有写权限

开发环境设置

对于开发环境来说，我们可以把整个 Cake 目录放置在 DocumentRoot 下：

```
/wwwroot
    /cake
        /app
        /cake
        /vendors
        .htaccess
        index.php
```

这样的设置方案下，你的 URL 会如下这般(假设你使用了 mod_rewrite)：
www.example.com/cake/controllerName/actionName/param1/param2

生产环境设置

使用生产环境配置，你必须拥有修改 Web Server DocumentRoot 的权限。那样的话，整个域就如同一个 CakePHP 应用一般。

生产环境配置使用如下目录结构

```
../path_to_cake_install
    /app
        /config
        /controllers
        /models
        /plugins
        /tmp
        /vendors
        /views
        /webroot <-- 这将是你新的 DocumentRoot
```

```
        .htaccess
        index.php
    /cake
    /vendors
    .htaccess
    index.php
```

建议修改 Apache 的配置文件如下：
DocumentRoot /path_to_cake/app/webroot

这样的配置下，你的 URL 将会如下这般：
http://www.example.com/controllerName/actionName/param1/param2

高级设置

在有些情况下，你希望能够将 Cake 应用的目录放在磁盘不同目录下。这可能是因为虚拟主机的限制，或者你希望多个 app 应用能够共享同一个 Cake Lib。

对于一个 Cake 应用，有 3 个主要组成部分：

1. CakePHP 核心 Lib - /cake 目录下
2. 你的应用的代码（例如：controllers, models, layouts and views） - /app 目录下
3. 你的应用 web 目录下的文件 （例如 图片 js 脚本以及 css 样式文件等） - /app/webroot 目录下

这 3 个目录都可以放置在磁盘的任意位置，但是 webroot 目录必须是 web server 可以访问的。你甚至可以把 webroot 目录移出 app 目录，只要你告诉 Cake 你把它们放到哪里去了。

你需要修改/app/webroot/index.php 来完成配置（和 Cake 一起分发）。你需要修改 3 个常量：ROOT, APP_DIR, and CAKE_CORE_INCLUDE_PATH。

1. ROOT 为包含 app 目录的根路径
2. APP_DIR app 目录路径
3. CAKE_CORE_INCLUDE_PATH Cake 核心 Lib 目录

你的应用 web 目录下的文件 （例如 图片 js 脚本以及 css 样式文件等） - /app/webroot 目录下

这是范例：

```
/app/webroot/index.php (partial, comments removed)
if (!defined('ROOT'))
{
    define('ROOT', dirname(dirname(dirname(__FILE__))));
}

if (!defined('APP_DIR'))
{
    define ('APP_DIR', basename(dirname(dirname(__FILE__))));
}

if (!defined('CAKE_CORE_INCLUDE_PATH'))
{
    define('CAKE_CORE_INCLUDE_PATH', ROOT);
}
```

下面通过一个具体的例子来解释我的配置

1. 我希望 Cake Lib 能够被共享，所以放在 user/lib/cake 下 （译注：作者都是 linux 配置，windows 环境下随便放放吧）

2. Cake webroot 目录为/var/www/mysite/。

3. 程序文件目录为/home/me/mysite。

下面为具体的目录结构，不再赘述

```
/home
    /me
        /mysite                     <-- Used to be /cake_install/app
            /config
```

```
                /controllers
                /models
                /plugins
                /tmp
                /vendors
                /views
                index.php
/var
    /www
        /mysite                 <-- Used to be /cake_install/app/webroot
            /css
            /files
            /img
            /js
            .htaccess
            css.php
            favicon.ico
            index.php
/usr
    /lib
        /cake                   <-- Used to be /cake_install/cake
            /cake
                /config
                /docs
                /libs
                /scripts
                app_controller.php
                app_model.php
                basics.php
                bootstrap.php
                dispatcher.php
            /vendors
```

我按照上面的目录结构修改/var/www/mysite/index.php 如下：

我建议使用'DS'常量代替路径中的斜杠。这样会保证你不会写错导致找不到文件。（考虑跨平台）

```
1. if (!defined('ROOT'))
2. {
3.     define('ROOT', DS.'home'.DS.'me');
4. }
5.
6. if (!defined('APP_DIR'))
```

```
{
    define ('APP_DIR', 'mysite');
}

if (!defined('CAKE_CORE_INCLUDE_PATH'))
{
    define('CAKE_CORE_INCLUDE_PATH', DS.'usr'.DS.'lib'.DS.'cake');
}

```

Section 5 配置 Apache 的 mod_rewrite

CakePHP 和 mod_rewrite 一同工作时会是最佳体验，我们发现有部分用户在他们的系统上不能让每件事情都很好的运作起来。所以我罗列了一些可能会有帮助的信息：

1. 首先确认.htaccess 覆写是被允许的：在 apache 配置文件中检查<Directory>配置下的 AllowOverride 是不是设置为 All。

2. 确认你修改的是 server 的配置文件而不是当前用户或者特定站点的配置文件。

3. 有的时候，可能你得到的 CakePHP 副本缺少了必须的.htaccess 文件。因为有的操作系统会将以.开头的文件视作隐藏文件，因而不会拷贝它们。确定你的 CakePHP 副本是我们网站上下载或者是我们的 SVN repository 上 checkout 的。

4. 确认你是否成功加载了 mod_rewrite！察看配置文件中是否有'LoadModule rewrite_module libexec/httpd/mod_rewrite.so' 和 'AddModule mod_rewrite.c'。

5. 如果你把 Cake 安装在一个用户目录下的话(http://example.com/~myusername/)，你需要修改根目录中的.htaccess 文件，加上一行"RewriteBase /~myusername/"。

6. 如果你发现 URL 后面加上了冗长乏味的 sessionID 的话 (http://example.com/posts/?CAKEPHP=4kgj577sgabvnmhjgkdiuy1956if6ska)

> 你需要加上一行"php_flag session.trans_id off"就 ok 了。

Section 6 确认 Cake 可以出炉了

好吧，让我们来看看这个 baby 吧。鉴于你选择的配置方式，你可以通过 http://www.example.com or http://www.example.com/ 来访问它。你会看到 CakePHP 的默认主页，和一个简短的通知消息，描述了你当前的数据库连接状态。

祝贺你！你已经完成了开始第一个 CakeApp 的准备工作。

配置 CakePHP

Section 1 数据库配置

app/config/database.php 这个文件包含了所有的数据库配置。新安装的 Cake 是没有 datebase.php 这个文件的，你需要从 database.php.default 复制一份过来并重新命名，内容如下：

```
1. app/config/database.php
2. var $default = array('driver'    => 'mysql',
3.                      'connect'   => 'mysql_connect',
4.                      'host'      => 'localhost',
5.                      'login'     => 'user',
6.                      'password' => 'password',
7.                      'database' => 'project_name',
8.                      'prefix'    => '');
9.
10. 和大多数 php 程序一样，修改一下你的配置就 ok 了。
11.
```

关于前缀要提醒一下：这里定义的前缀将会用在任何一处由 Cake 发起的对数据表的 SQL 操作中。你在这里定义过后不能再在其它地方重新定义它。假设你的服务器只给了你一个单独的 database，它同样可以让你遵循 Cake 的表名约定。注意：对于 HABTM(has-and-belongs-to-many)，也就是多对多关联表，你只需要添加一次前缀 prefix_apples_bananas，而不是 prefix_apples_prefix_bananas。

CakePHP 支持下列数据库驱动：

- mysql
- postgres
- sqlite
- pear-drivername (so you might enter pear-mysql, for example)
- adodb-drivername

$default 连接中的'connect'可以指定这个连接是否是 persistent 连接。参考 database.php.default 中的注释来决定采用何种方式。

Rails 两大原则之一，就是约定大于配置，所以你定义的数据表名必须遵循下面的命名约定：

1. 表名为英语单词的复数形式，例如"users", "authors"或者"articles"。注意与之对应的 model 对象则使用单数形式做名字。

2. 表字段必须有一个主键字段叫'id'。（相信用过 ORM 工具的都明白个中道理吧）

3. 如果你使用外键关联表，外键字段必须定义成这个样子：'article_id'。即关联表名的单数形式加上下划线然后 id。其实也很好理解，并不需要特别记忆吧。

4. 如果你定义了 'created' 'modified'这样的 auditor 字段的话，Cake 会自动在操作的时候去填充。(译注：rails 果然周全，hibernate 里面的 interceptor 就那么容易的做到了)

你应该注意到，同样的还有一个$test 的数据库连接配置项，这是考虑到一般开发数据库和生产数据库通常并不是同一个，所以你可以在程序通过 var $useDbConfig = 'test'来切换数据库连接设置。

在你的 model 对象中，你可以通过这样的方式来添加任意多个附加数据库连接。

Section 2 全局配置

CakePHP 的全局配置文件为/app/config/core.php。尽管我们非常讨厌配置文件，但是还是有些配置需要去做。文件中的注释对每一个配置项都有很详尽的说明。

DEBUG：这个选项定义了应用程序的 Debug 级别。设置为一个非零的数值 Cake 会输出所有的 pr()和 debug()函数调用，并且在 Forward 之前会有提示信息。设置为 2 或者是更高的数值，则会在页面底部输出 sql 语句。
在 debug 模式下（DEBUG 被设为 1 或者更高），Cake 会输出错误自定义错误页面，例如 "Missing Controller","Missing Action"等等。在生产环境下（DEBUG 被设为 0），Cake 输出 "Not Found"页面，你可以自行修改该页面(app/views/errors/error404.thtml)。

CAKE_SESSION_COOKIE: 这个设置可以设定你所希望使用的 cookie 名字。

CAKE_SECURITY：改变这个值标示了你所希望采用 Session 校验级别。Cake 会根据你设定的值来执行 Session 超时，生成新的 session id，删除历史 session file 等操作。可以选择的值如下所示：

1. high：最高安全级别，10 分钟没有活动的 session 被置为超时，同时每一次请求都将生成新的 session id
2. medium: 20 分钟没有活动的 session 会被置为超时
3. low: 30 分钟没有活动的 session 会被置为超时

CAKE_SESSION_SAVE：设置你的 session 数据储存策略：

1. cake：Session 被保存在 Cake 安装目录下的 tmp 目录
2. php: Session 被保存在 php.ini 文件设置的储存路径下
3. database: Session 保存在$default 数据库连接所指定的库中

Section 3 路由(Route)配置

"Routing"是一个纯 PHP，类似 mod_rewrite 的机制，来实现 URL 和 controller/action/params 的自动映射。通过这个机制 Cake 可以拥有简洁且可配置的 URL 并且可以摆脱对 mod_rewrite 的依赖。当然使用 mod_rewrite 会让你的浏览器地址栏变得更加简洁。

Routes（路由）是独立的 URL 和指定的 controller 和 action 映射规则。路由规则在 app/config/routes.php 里面配置。示例如下：

```
Route Pattern
<?php
$Route->connect (
    'URL',
    array('controller'=>'controllername',
    'action'=>'actionname', 'firstparam')
);
```

```
?>

    1.
        URL 是一个正则表达式，表示你希望采用的 URL 格式
    2.
        controllername 是希望调用的 controller 的名字

    3.
        actionname 是希望调用的 controller 的 action 名字

    4.
        firstparam 是 action 的第一个参数名
```

第一个参数之后的所有参数作为 parameters 被传递到 action。

下面是一个简单的示例，所有的 url 都被定向到 BlogController。默认的 action 为 BlogController::index()。

```
Route Example
<?php
$Route->connect ('/blog/:action/*', array('controller'=>'Blog', 'action'=>'index'));
?>
A URL like /blog/history/05/june can then be handled like this:
Route Handling in a Controller
<?php

class BlogController extends AppController
{
  function history($year, $month=null)
  {
    // .. Display appropriate content
  }
}
?>
```

URL 中的'history'通过 Blog's route 中的:action 匹配。URL 中通过*匹配的内容都被当作 parameters 传递到:action 对应的方法中，就像这里的$year 和$month。如果 URL 是/blog/history/05，则 history() 方法只会得到一个参数，05。

下面这个例子是 CakePHP 默认设置的一个 route 来为 PagesController::display('home')配置路由。Home 是 Cake 的默认首页视图，你可以在这个位置找到并修改它/app/views/pages/home.thtml。

```
1. Setting the Default Route
2. <?php
3. $Route->connect ('/', array('controller'=>'Pages', 'action'=>'display', 'home'));
4. ?>
5.
```

Section 4 高级路由配置：Admin 路由与 Webservice

在/app/config/core.php 中还有那么一些高级选项可以供你使用，你可以利用它们使你的 URL 精巧地适应绝大多数情况。

第一个介绍的选项是管理员路由(admin routing)。假设你有两个 Controller，ProductsController 和 NewsController，你肯定希望能有一些特别的 URL 给具有管理权限的用户来访问 Controller 中一些特别的 action。为了保持 URL 的优雅和可读性，有些程序员相比 /products/adminAdd 和 /news/adminPost 更喜欢 /admin/products/add 和 /admin/news/post。

激活这个功能，首先你需要把/app/config/core.php 中 CAKE_ADMIN 的注释去掉。CAKE_ADMIN 的默认值是'admin'，不过你可以修改为任何你喜欢的名字。但是你要记住这个名字，因为你需要在每一个 admin action 名字前面加上这个前缀，格式为 admin_actionName()。

```
/admin/products/add          CAKE_ADMIN = 'admin'
                              name of action in ProductsController = 'admin_add()'

/superuser/news/post         CAKE_ADMIN = 'superuser'
                              name of action in NewsController = 'superuser_post()'
```

```
/admin/posts/delete             CAKE_ADMIN = 'admin'
                                  name of action in PostsController = 'admin_delete()'
```

使用 admin route 可以让你的业务逻辑非常井井有条。
注意！！并不是说使用了 admin route 同时 Cake 也替你完成了安全和认证，这些还是需要你自己来完成的。

类似的，你也能很快的为 webservice 建立路由。你需要将某个 action 暴露为 webservice 接口吗？首先，将/app/config/core.php 中的 WEBSERVICES 选项设为'on'。和先前的 admin 一样，打开这个选项可以很方便的让带有如下前缀的 action 自动暴露为 webservice 接口。

- rss
- xml
- rest
- soap
- xmlrpc

这样做的结果就是同一个 action，拥有了两种不同的视图，一个是标准的 HTML 视图，一个是 webservice 接口。因此你可以非常简单的让你的程序拥有 webservice 的能力。

举个例子，我的程序中有这么一个逻辑，有这么一个 action 可以告诉用户当前办公室里有多少人正在通话。我已经有了一个 HTML 视图，不过我希望再提供一个 XML 格式的接口供我的桌面 Widget(译注：就是类似 yahoo widget 一类的桌面小贴士)或者是掌上设备来使用。首先，我打开了 webservice 路由功能：

```
1. /app/config/core.php (partial)
2. /**
3.  *  The define below is used to turn cake built webservices
4.  *  on or off. Default setting is off.
5.  */
6.     define('WEBSERVICES', 'on');
7.
```

然后我在 controller 中构建代码如下：

```
1. <?php
2. class PhonesController extends AppController
3. {
```

```
4.    function doWhosOnline()
5.    {
6.        // this action is where we do all the work of seeing who's on the phone...
7.
8.        // If I wanted this action to be available via Cake's xml webservices route,
9.        // I'd need to include a view at /app/views/posts/xml/do_whos_online.thtml.
10.         // Note: the default view used here is at /app/views/layouts/xml/default.thtml.
11.
12.       // If a user requests /phones/doWhosOnline, they will get an HTML version.
13.       // If a user requests /xml/phones/doWhosOnline, they will get the XML version.
14.   }
15. }
16. ?>
17.
```

Section 5 自定义反射配置（可选）

Cake 的命名约束真的是非常不错，你可以命名 model 为 Box，controller 为 Boxes，于是所有的问题都解决了。但是考虑到有的情况下（特别是非英语国家）这样的命名约束不如你意的话，你可以通过自定义反射配置来满足你的需求。

/app/config/inflections.php，这里定义了 Cake 的一些变量，你可以来调整名字单复数约定等。你需要详细阅读文件中的注释，并且最好具备正则表达式知识，否则请误随便修改该配置文件。

Scaffolding 脚手架

Section 1 Cool！Cake 的脚手架

Cake 的脚手架真的非常酷，以至于你会希望将他用到你的生产环境下。我们也认为它非常的酷，但是请意识到脚手架再好也还是脚手架。它只是一大捧素材供你在项目起始阶段能够迅速的搭起整个项目的架子。所以，当你发现你需要实现自己的业务逻辑的时候就是该拆掉脚手架的时候了。

对于 web 应用项目，由脚手架开始项目是非常不错的办法。早期的数据库设计通常是经常会变动的。有这么一种趋势，web 开发人员非常厌恶创建一下根本不会实际使用的页面。为了减少开发人员的这种压力，Cake 内置了脚手架功能。脚手架会根据你当前的表结构创建基础的 CRUD 页面。在你的 controller 中使用脚手架只需要一行代码，加入一个变量就 ok。

```
1. <?php
2. class CategoriesController extends AppController
```

```
{
    var $scaffold;
}
?>

```

有一个非常重要的事情：脚手架中，要求每一个外键字段都必须以 name_id 这样的形式命名。因此，当你多级目录结构的时候，你可能会有一个字段叫 parent_id，这样的时候你就不能如此命名了，因为根据约定，它会去映射 parent 这个 model，而实际上是不存在这个 model 的，所以最好命名为 parentid。但是如果你是有一个外键关联另一个 model 的话（例如：titles 表中有 category_id 作为外键），你首先要确保正确的处理关联对象(参见 正确理解关联关系， 6.2)，在 show/edit/new 这 3 个视图里面会自动生成一个下拉框来展示/选择 category 信息。可以通过设置 model 中的 $displayField 变量来取舍那些外键字段是需要显示的。具体示例如下：

```
<?php
class Title extends AppModel
{
    var $name = 'Title';

    var $displayField = 'title';
}
?>

```

Section 2 自定义脚手架视图

如果你希望脚手架的页面有一些自定义的元素，你可以自己修改模版。我们仍然不建议你在生产环境下使用脚手架视图，不过一定的自定义将会对于原形构建有这非常大的帮助。

如果你不希望修改默认提供的脚手架视图，你可以自己提供这些模版：

自定义单个 Controller 默认脚手架视图

PostsController 的自定义脚手架视图放置位置如下：

/app/views/posts/scaffold/index.scaffold.thtml
/app/views/posts/scaffold/show.scaffold.thtml
/app/views/posts/scaffold/edit.scaffold.thtml
/app/views/posts/scaffold/new.scaffold.thtml

自定义全局脚手架默认视图

全局默认脚手架视图放置位置如下：

/app/views/scaffold/index.scaffold.thtml
/app/views/scaffold/show.scaffold.thtml
/app/views/scaffold/edit.scaffold.thtml
/app/views/scaffold/new.scaffold.thtml

当你发现你需要实现自己的业务逻辑的时候也就是该拆除脚手教的时候。
Cake 中 code generator 有一个非常有用的功能：Bake。Bake 可以为你的应用生成一个基础的架子供你填充业务逻辑。

Models

Section 1 What is a model?

什么是 model 呢？model 就是 MVC 模式中的 M。
Model 用来做什么呢？它将领域逻辑从表现层和独立的业务逻辑中剥离出来。

Model 通常是数据库的访问介质，而且更明确的指定于某一张表（译注：在 rails 框架中 model 扮演着 active record 的角色，因此同时承担着 DAO 对象的职责）。默认情况下，model 使用的表名为 model 名字的复数形式，比如'User'这个 model 对应的表名为'users'。Model 可以包含数据校验规则，关联关系以及针对这张表的业务逻辑。下面展示了 User model 在 cake 中的具体定义：

```
1. Example User Model, saved in /app/models/user.php
2. <?php
3.
4. //AppModel gives you all of Cake's Model functionality
5.
6. class User extends AppModel
7. {
8.     // Its always good practice to include this variable.
9.     var $name = 'User';
```

```
10.
11.     // This is used for validation, see Chapter 11.
12.     var $validate = array();
13.
14.     // You can also define associations.
15.     // See section 6.3 for more information.
16.
17.     var $hasMany = array('Image' =>
18.                 array('className' => 'Image')
19.                 );
20.
21.     // You can also include your own functions:
22.     function makeInactive($uid)
23.     {
24.         //Put your own logic here...
25.     }
26. }
27.
28. ?>
29.
```

（译注：关于 Model 应该是贫血型，失学型还是充血型的论战持续至今尚无明显得胜的一方，本文中的这句话：Model 将领域逻辑从表现层和独立的业务逻辑中剥离出来 说得还比较清晰的，起码在 cake 里面他就是这么遵循的）

Section 2 Model Functions

从 PHP 的角度来看 Model，只是一个继承了 AppModel 的类。AppModel 类在/cake 目录中已经定义了，如果你希望创建一个自己的 AppModel 基类，你可以放在 app/app_model.php 位置。AppModel 中定义的方法将被所有 model 所继承并共享。而 AppModel 则继承于 Cake Lib 中的 Model 类，文件为 cake/libs/model.php。

本章将会讨论到大量已经在 Cake Model 类中定义的方法，所以这里提醒，http://api.cakephp.org 将是您非常重要的参考。

用户定义的 Model 函数
下面是一个用户定义的 Model 方法，是帖子的隐藏/显示方法：

```
1. Example Model Functions
```

```php
<?php
class Post extends AppModel
{
    var $name = 'Post';

    function hide ($id=null)
    {
        if ($id)
        {
            $this->id = $id;
            $this->saveField('hidden', '1');
        }
    }

    function unhide ($id=null)
    {
        if ($id)
        {
            $this->id = $id;
            $this->saveField('hidden', '0');
        }
    }
}
?>
```

获取你需要的数据

下面是 Model 中一些获取你需要的数据的标准方法：

```
findAll
    string $conditions
    array $fields
    string $order
    int $limit
    int $page
    int $recursive
```

[$conditions Type: string] 检索条件,就是 sql 中 where 子句，就像这样 $conditions = "race = 'wookie' AND thermal_detonators > 3"。

[$fields Type: array] 检索属性，就是投影，指定所有你希望返回的属性。（译注：这里我没有使用字段这个亲 SQL 的名字而是用了属性这个亲对象的名字，我相信很多 PHPer 更熟悉基于 Sql 和 DTO 的开发模式，但是引入 Model 的一个初衷就是将关系数据库转换为对象操作，所以投影查询应该理解为检索对象的某些属性）

[$order Type: string] 排序属性 指定了 oder by 的属性名 [TODO：check whether the multiple order field be supported]

[$limit Type: int] 结果集数据条目限制

[$page Type: int] 结果集分页 index，默认为返回第一页

[$recursive Type: int] 递归 当设为大于 1 的整数时，将返回该 model 关联的其他对象。（译注：如果你使用过类似于 Hibernate 这样的 ORM 工具的话，不难理解这个属性就是 LazyLoad 属性，但也不完全等同，数字的值代表级联查询的层次数，例如这样，user.country.city.address.id）

```
find
    string $conditions
    array $fields
    string $order
    int $recursive
```

find 方法和 findAll 方法的区别在于，findAll 方法返回所有符合的结果集，find 方法只返回 list 中的第一个结果。

```
findAllBy<fieldName>
    string $value
```

这是一个非常有魔力的方法，你可以看成是一个快速按照某个属性检索数据的方法，下面是在 controller 中的一段示例代码：

```
1. $this->Post->findByTitle('My First Blog Post');
2. $this->Author->findByLastName('Rogers');
3. $this->Property->findAllByState('AZ');
4. $this->Specimen->findAllByKingdom('Animalia');
5.
```

返回类型和 find() findAll()一样，是一个 array。

```
findNeighbours
        string $conditions
        array $field
        string $value
```

这个方法首先是通过 conditions 过滤获取结果集，然后根据 field=value 查找符合的对象（仅包含 $field 中指定的属性），最终返回该对象以及该对象的上一个对象和下一个对象。这在诸如相册这样的应用场景中将会非常有作用，你可以方便的获取当前相片，上一张和下一张这样的结果集合。注意：该方法只能作用于数值型和日期时间型属性。下面是示例代码：

```
1. class ImagesController extends AppController
2. {
3.     function view($id)
4.     {
5.         // Say we want to show the image...
6.
7.         $this->set('image', $this->Image->find("id = $id");
8.
9.         // But we also want the previous and next images...
10.
11.          $this->set('neighbours', $this->Image->findNeighbours(null, 'id', $id);
12.
13.     }
14. }
15.
```

我们拿到了包含一个完整$image['Image']对象的 array，以及$neighbours['prev']['Image']['id']和 $neighbours['next']['Image']['id']。

```
field
    string $name
    string $conditions
    string $order
```

返回 conditions 过滤后按 order 排序的第一个结果中的 name 属性值，返回类型为 string。

```
findCount
    string $conditions
```

返回 conditions 过滤后结果集的数量。即 select count .....

```
generateList
    string $conditions
    string $order
    int       $limit
    string $keyPath
    string $valuePath
```

generateList 方法可以非常快捷的获取一个 key-value 这样的 list，其实就是其他语言中的 map 类型，在 web 领域中，下拉框可能将是该方法最大的受益者。$conditions， $order 和 $limit 这 3 个参数的用法和 findAll()没有区别，$keyPath 和 $valuePath 是 model 中用来填充结果集中 key 和 value 的属性。举个例子，在权限操作中，角色的定义往往是 id - name 这样的组合，参见下面的示例代码：

```
$this->set(
    'Roles',
    $this->Role->generateList(null, 'role_name ASC', null, '{n}.Role.id', '{n}.Role.role_name')
);

//This would return something like:
array(
    '1' => 'Account Manager',
    '2' => 'Account Viewer',
     '3' => 'System Manager',
     '4' => 'Site Visitor'
);
```

```
read
    string $fields
    string $id
```

根据$fields 中的属性名从当前对象中读出对应的值，或者是$id 指定的对象中读取。注意：read() 方法仅级联读取该对象的一级关联对象，不管 model 中的$recursive 的值为多少。如果你是希望获取所有级联属性的话，请使用 find()或者 findAll()方法。

```
query
    string $query

execute
    string $query
```

有的时候希望执行自定义的 sql 语句或者是出于性能上的考虑要优化 sql 语句，则可以使用 query(string $query)和 execute(string $query)方法，这两个方法不同的地方在于 execute 方法无返回值，适用于执行脚本而不是查询结果集。

```
1. Custom Sql Calls with query()
2. <?php
3. class Post extends AppModel
4. {
5.     var $name = 'Post';
6.
7.     function posterFirstName()
8.     {
9.         $ret = $this->query("SELECT first_name FROM posters_table
10.                              WHERE poster_id = 1");
11.         $firstName = $ret[0]['first_name'];
12.         return $firstName;
13.     }
14. }
15. ?>
16.
```

复合查询条件（使用 array）

绝大多数的 Model 查询方法中都会通过各种方式来传递一组条件。最简单的方式莫过于使用一个 SQL 中的 where 子句，不过如果你希望获得更多的控制，你可以使用 array。使用 rray 的好处是代码会非常的清晰和易读，并且可以比较方便的构造查询。而且这样的语法可以保证你的查询组成元素（属性，值，运算符等）被拆分为精巧可控的几部分。这可以保证 Cake 能够生成绝大部分非常有效率的查询语句，并且避免了 SQL 语法上易犯的错误。

先看看非常基础的基于 array 的查询：

简单查询条件 array 示例：
$conditions = array("Post.title" => "This is a post");

这个结构恐怕是不言自明了：该查询返回所有 title 为"This is a post"的 Post 对象。注意，只使用"title"作为查询的属性名也是可以的，但是在构造查询的时候，使用[ModelName.FieldName]这样的写法，一者可以让你的代码更加清晰，二者可以防止命名冲突，所以请遵循这个 Good Practice。那其他的检索形式如何呢？同样很简单，比如我们想检索所有 title 不是"This is a post"的 Post 对象：

array("Post.title" => "<> This is a post")

唯一增加的就是'<>'这个表达式。Cake 能够解析所有的合法 SQL 比较运算符，包括 LIKE, BETWEEN 或者是正则表达式，注意，运算符和值或者表达式之间需要有一个空格。唯一不同的操作是 IN 操作。我们看下具体的例子：

array("Post.title" => array("First post", "Second post", "Third post"))

为 conditions 增加更多 filter 就像往 array 中添加一对 key-value 那样简单：

```
array
(
    "Post.title"    => array("First post", "Second post", "Third post"),
    "Post.created" => "> " . date('Y-m-d', strtotime("-2 weeks"))
)

```

默认情况下，Cake 使用'AND'来连接多个条件；这意味着，上面的示例代码检索结果为过去 2 周内 title 为"First post", "Second post"或者"Third post"的记录。当然我们也同样可以使用其他的逻辑运算符来连接查询条件：

```
array
("or" =>
    array
    (
        "Post.title" => array("First post", "Second post", "Third post"),
        "Post.created" => "> " . date('Y-m-d', strtotime("-2 weeks"))
    )
)

```

Cake 可以使用任何 SQL 语句中允许的逻辑运算符，如 AND, OR, NOT, XOR 等等，并且大小写不敏感（Conditions 可以无限嵌套，但我并不推荐使用这样的 magic code）。Posts 和 Authors 两个对象间分别是 hasMany/belongsTo 的关系(熟悉 Hibernate 的同学或许更喜欢一对多 多对一的叫法)。假设你希望检索所有过去两周内发布的 posts 或者是包含有指定关键字的 posts，并且你希望限定作者为 Bob，让我们看如下代码：

```
array
("Author.name" => "Bob", "or" => array
    (
        "Post.title" => "LIKE %magic%",
        "Post.created" => "> " . date('Y-m-d', strtotime("-2 weeks")
    )
)

```

保存数据

当需要保存 model 对象时（译注：或者使用持久化这个词更贴切），你需要提供如下形式的数据给 sava()方法：

```
Array
(
    [ModelName] => Array
        (
            [fieldname1] => 'value'
            [fieldname2] => 'value'
        )
)

```

为了能够让数据通过这样的形式提交给 controller，最方便的办法就是使用 HTML Helper 来完成这个任务，因为 HTML Helper 会为提交的 Form 封装成 Cake 希望的这种形式。当然你可以不使用，只要页面上的元素的 name 被设置成 date[Modelname][fieldname]形式就 ok 了。不过我还是要说，$html->input('Model/fieldname')是最方便的办法。
（译注：OGNL 的 php 版，太棒了，我的意见是如果不嫌麻烦的话尽量使用类似 OGNL 的做法，因为 tag 产生的页面在设计期是无法预览的，我相信 web 应用前台页面的设计的复杂性并不亚于后台业务逻辑）

从页面 Form 中提交的数据自动被格式化并且注入到 controller 中的$this->data 变量，所以保存从 web 页面提交的数据只是举手之劳。下面我们来看一段示例代码：

```
function edit($id)
{

    //Note: The property model is automatically loaded for us at $this->Property.

    // Check to see if we have form data...
    if (emptyempty($this->data))
    {
        $this->Property->id = $id;
          $this->data = $this->Property->read();//populate the form fields with the current row
    }
    else
    {
        // Here's where we try to save our data. Automagic validation checking
        if ($this->Property->save($this->data['Property']))
        {
            //Flash a message and redirect.
            $this->flash('Your information has been saved.',
```

```
19.                     '/properties/view/'.$this->data['Property']['id'], 2);
20.         }
21.         //if some fields are invalid or save fails the form will render
22.     }
23. }
24.
```

注意保存数据前会触发 model 的校验机制，更多关于数据校验的内容请参见 Chapter 12。如果你不希望进行数据校验，可以使用 save($date, false)。

其他一些有用的数据保存函数

del
string $id
boolean $cascade

删除指定 id 的 model 对象，或者当前 model 对象。
如果$cascade 设为 true，则会级联删除当前 model 所关联的所有 model 中，设为'dependent'的 model 对象。删除成功返回 true

saveField
string $name
string $value

保存单个属性值。

getLastInsertId

返回最后当前 ID 属性的 nextValue。
[ToDo 扩展支持多种主键生成策略 例如 sequence UUID]

Model 中的回调函数

我们在 model 中增加了一些回调函数以帮助你在 model 操作前后能够织入一些业务逻辑（原文为 sneak in，借用了 AOP 中的织入一词，因为从操作来看这些回调函数等同于 AOP 中的 advice）。
为了获得这样的能力，需要使用 model 中的一些参数并且在你的 model 中覆写这些方法。

beforeFind
    string $conditions

beforeFind()回调函数是在 model 的 find 方法执行前的前置操作。你可以加入任何检索前的业务逻辑。你覆写该方法只要保证在前置操作成功后返回 true 来执行真正的 find 方法，返回 false 中断 find 方法就可以了。（译注：在一些复杂场景中，需多次持久化的情况下请慎用）

afterFind
    array $results

使用 afterFind 回调函数能够更改 find 方法的返回结果集，或者在检索动作完成后加上一些业务逻辑。该函数的参数$results 为经过回调函数处理以后的 find 检索结果集。

beforeValidate

beforeValidate 回调函数能够在 model 校验数据之前更改 model 中的一些数据。同样也可以用来在 model 校验之前加入更为复杂的额外校验规则。和 beforeFind 一样，必须保证返回 true 来调用真正的操作，返回 false 来中断校验乃至 save 操作。

beforeSave

和先前介绍的回调函数类似，在校验完成之后，保存动作之前加入额外的处理（如果校验失败是不会触发该回调函数的）。返回 true 或者 false，不再赘述。
一个比较常见的 beforeSave 的应用场景就是在保存动作之前格式化日期属性以适应不同的数据库：

```
// Date/time fields created by HTML Helper:
// This code would be seen in a view

$html->dayOptionTag('Event/start');
$html->monthOptionTag('Event/start');
$html->yearOptionTag('Event/start');
$html->hourOptionTag('Event/start');
$html->minuteOptionTag('Event/start');

/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
```

```
11.
12. // Model callback functions used to stitch date
13. // data together for storage
14. // This code would be seen in the Event model:
15.
16. function beforeSave()
17. {
18.     $this->data['Event']['start'] = $this->_getDate('Event', 'start');
19.
20.     return true;
21. }
22.
23. function _getDate($model, $field)
24. {
25.     return date('Y-m-d H:i:s', mktime(
26.         intval($this->data[$model][$field . '_hour']),
27.         intval($this->data[$model][$field . '_min']),
28.         null,
29.         intval($this->data[$model][$field . '_month']),
30.         intval($this->data[$model][$field . '_day']),
31.         intval($this->data[$model][$field . '_year'])));
32. }
33.
34.
```

afterSave

保存完成后执行的动作。[ToDo 如果保存出错是否会触发？]

beforeDelete

afterDelete

不需要多说了吧，删除操作前后的回调函数。

Section 3 Model 中的变量

当你创建一个新的 model 的时候，有很多特殊的变量可以设置，从而是 model 具备 Cake 赋予的相应操作。

$primaryKey
如果主键字段不为'id'，COC 无法发挥的时候，你可以通过该变量来指定主键字段名字。

$recursive
这个我们上面已经介绍过了，指定了 model 级联关联对象的深度。

想象这样一个场景，Group 下的 User 下面还有各自的 Articles。

```
$recursive = 0 Cake 只会检索 Group 对象
$recursive = 1 Cake 会检索包含 User 对象的 Group 对象
$recursive = 2 Cake 会检索完成的包含 Article 的 Group 对象
```

$transactional
决定 Model 是否允许事务处理，true 或者 false。注意，仅在支持事务的数据库上有效。

$useTable
和$primaryKey 一样，如果你的表名不能和 model 匹配的话，而且你也不想或者不能修改表名，则可以通过该变量来指定数据表名。

$validate
该变量为一个 array，表示一组校验规则，详细请参见 Chapter 12。

$useDbConfig
还记得我们在配制一章介绍的如何配置数据库连接变量吗？可以通过该变量非常方便的切换数据库连接，默认的是什么？猜一下，当然就是'default'，rails 就是让你忘却那些难以记忆的配置。

Section 4 [重头戏]关联对象

简介

CakePHP 中提供的一个非常重要的功能就是关联数据表间的映射。在 CakePHP 中，关联表通过 association（关联）来处理。关联是这些逻辑上相关的单元间的胶水一般。

CakePHP 中一共有 4 种类型的关联：

```
hasOne

hasMany

belongsTo
```

hasAndBelongsToMany

一旦 model 间的关联关系被定义，Cake 能够自动检索你当前操作的 model 中包含的关联对象。也就是将基于关系数据库的数据映射为基于对象的领域模型。举个例子，Post 和 Author 之间是 hasMany(一对多)关系，当我们在 controller 中通过$this->Post->findAll()来检索所有的 Post 对象时，也会同时把所有关联的 Author 对象检索出来。

遵循 CakePHP 的命名约定是正确定义关联关系的有利保证（参见附录 B）。如果你使用了 CakePHP 的命名约定，可以通过脚手架来可视化你的数据，因为脚手架会自动侦测并使用你定义的关联关系，这样也能某种程度上供你检查是否定义正确。当然，你也可以完全不使用命名约定来定义关联，不过我们稍候再介绍关于这种定义。现在我们仅关注使用命名约定的情况下的关联定义。命名约定中我们需要考虑的有这 3 个内容，外键，model 名字，表名。

这里先简单的介绍一下关于这 3 者的要求，更为详细的请查看附录：

外键：[单数形式的 model 名字]_id 假设在"authors"表中有 Post 的外键关联，则外键字段名字应该为 "post_id"。
表名：[复数形式的 model 名字] 表名必须为 model 名字的复数形式，例如："posts" "authors"。
Model 的名字：[驼峰法命名 单数形式]。

CakePHP 的脚手架希望你的关联定义的顺序和表中的外键字段的顺序是一致的。所以如果我有一个 Article 对象[belongsTo(属于)]另外 3 个对象（Author Editor Publisher）的话，我需要 3 个外键 author_id, editor_id, publisher_id。脚手架要求你 model 中对应的关联和数据库中的列顺序保持一致。

为了更好的描述关联对象是如何运作的，让我们继续以 Blog 为应用场景来介绍。假设我们现在需要为 Blog 系统创建一个简单的用户管理模块。我们假设我们不需要跟踪用户情况，但是我们希望每个用户都一个个人记录（用户 [hasOne] 个人记录）。用户可以创建多条评论记录（用户 [hasMany] 评论记录）。同样的，我们的文章会被分配到多个 tag，同时每个 tag 都包含多篇文章，也就是多对多关系 （文章 [hasAndBelongsToMany] Tag）。

hasOne 关联的定义与查询

假设你已经准备好了 User 和 Profile 两个 model，让我们来定义他们之间的关联。hasOne 关联的定义是通过在 model 中增加一个 array 来实现的。下面是示例代码：

```
1. /app/models/user.php hasOne
2. <?php
3. class User extends AppModel
4. {
5.     var $name = 'User';
6.     var $hasOne = array('Profile' =>
```

```
                array('className'    => 'Profile',
                    'conditions'    => '',
                    'order'        => '',
                     'dependent'    =>  true,
                     'foreignKey'    => 'user_id'
                )
            );
}
?>

```

$hasOne 变量是一个 array，Cake 通过该变量来构建 User 与 Profile 之间的关联。我们来看每一个元素代表的意义：

- className (required)：关联对象的类名，上面代码中我们设为'Profile'表示关联的是 Profile 对象。
- conditions：关联对象的选择条件，（译注：类似 hibernate 中的 formula）。具体到我们的例子来看，假设我们仅关联 Profile 的 header color 为绿色的文件记录，我们可以这样定义 conditions，"Profile.header_color = 'green'"。
- order：关联对象的排序方式。假设你希望关联的对象是经过排序的，你可以为 order 赋值，就如同 SQL 中的 order by 子句："Profile.name ASC"。
- dependent：这是个布尔值，如果为 true，父对象删除时会级联删除关联子对象。在我们的 Blog 中，如果"Bob"这个用户被删除了，则关联的 Profile 都会被删除。类似一个外键约束。
- foreignKey：指向关联 Model 的外键字段名。仅在你不遵循 Cake 的命名约定时需要设置。

现在，现在当我们使用 find() findAll()检索 User 对象时，你会发现关联的 Profile 对象也被检索回来，非常的方便：

```
$user = $this->User->read(null, '25');
print_r($user);

//output:

Array
(
    [User] => Array
        (
            [id] => 25
            [first_name] => John
            [last_name] => Anderson
```

```
            [username] => psychic
            [password] => c4k3roxx
        )

    [Profile] => Array
        (
            [id] => 4
            [name] => Cool Blue
            [header_color] => aquamarine
            [user_id] = 25
        )
)

```

belongsTo 关联的定义与使用

现在 User 对象能够得到对应的 Profile 对象，当然我们也应该为 Profile 对象定义一个关联使之能够获取它的所有者，也就是对应的 User 对象。在 Cake 中，我们使用 belongsTo 关联来实现：

```
/app/models/profile.php belongsTo
<?php
class Profile extends AppModel
{
    var $name = 'Profile';
    var $belongsTo = array('User' =>
                        array('className'   => 'User',
                            'conditions' => '',
                            'order'      => '',
                              'foreignKey' => 'user_id'
                          )
                    );
}
?>

```

和 hasOne 关联一样，belongsTo 也是一个 array 变量，你可以通过设置其中的值来具体定义 belongsTo 关联：

- className (required)：关联对象的类名，这里我们关联的是 User 对象，所以应该是'User'。
- conditions: SQL 条件子句以限定关联的对象，假定我们只允许 Profile 关联状态为 active 的用户，我们可以这样写："User.active = '1'"，当然也可以是类似的其它条件。
- order：关联对象的排序子句，假如你希望关联的对象经过排序，你可以类似"User.last_name ASC"这样来定义。
- foreignKey：关联对象所对应的外键字段名，仅在你不遵循 Cake 的命名约定时需要设置。

现在当我们使用 find() findAll()来检索 Profile 对象时，会发现关联的 User 对象也一同被检索回来。

```
$profile = $this->Profile->read(null, '4');
print_r($profile);

//output:

Array
(

    [Profile] => Array
        (
            [id] => 4
            [name] => Cool Blue
            [header_color] => aquamarine
            [user_id] = 25
        )

    [User] => Array
        (
            [id] => 25
            [first_name] => John
            [last_name] => Anderson
            [username] => psychic
            [password] => c4k3roxx
        )
)

```

hasMany 关联的定义与查询

我们已经为 User 和 Profile 对象建立起了双向关联，那让我们开始为 User 和 Comment 对象之间建立关联吧，先看下面的示例代码：

```
1. /app/models/user.php hasMany
2. <?php
3. class User extends AppModel
4. {
5.     var $name = 'User';
6.     var $hasMany = array('Comment' =>
7.                         array('className'     => 'Comment',
8.                               'conditions'    => 'Comment.moderated = 1',
9.                               'order'         => 'Comment.created DESC',
10.                               'limit'         => '5',
11.                               'foreignKey'    => 'user_id',
12.                               'dependent'     => true,
13.                               'exclusive'     => false,
14.                               'finderQuery'   => ''
15.                          )
16.                 );
17.
18.     // Here's the hasOne relationship we defined earlier...
19.     var $hasOne = array('Profile' =>
20.                         array('className'     => 'Profile',
21.                               'conditions'    => '',
22.                               'order'         => '',
23.                               'dependent'     =>  true,
24.                               'foreignKey'    => 'user_id'
25.                         )
26.                 );
27. }
28. ?>
29.
```

$hasMany array 用来定义 User 包含多条 Comment 这样的关联关系。还是老样子，介绍一下包含的 key，但是一些和之前同样含义的 key 我将不再赘述详细。

- className (required)：关联对象类名。
- conditions：关联对象限定条件。
- order：关联对象排序子句。
- limit：因为是一对多关系，所以可以通过 limit 来限定检索的关联对象数量。比如我们可以只关联 5 条评论记录。
- foreignKey：外键字段名。仅当不遵循命名约定时起用。
- dependent：是否级联删除。（该动作可能会造成数据的误删除，请谨慎设定）
- exclusive：如果设为 true，所有的关联对象将在一句 sql 中删除，model 的 beforeDelete 回调函数不会被执行。但是如果没有复杂的逻辑在级联删除中，这样的设定会带来性能上的优势。（译注：Cake 的确方便，但是使用时一定要记住控制 sql 语句发送数量）
- finderQuery：定义一句完整的 sql 语句来检索关联对象，能够对关联规则进行最大程度上的控制。当关联关系特别复杂的时候，比如 one table - many model one model - many table 的情况下，Cake 无法准确的替你完成映射动作，需要你自己来完成这个艰巨的任务。

现在看一下如何在检索 user 对象的时候一并读回 comment 对象集合

```
1. $user = $this->User->read(null, '25');
2. print_r($user);
3.
4. //output:
5.
6. Array
7. (
8.     [User] => Array
9.         (
10.             [id] => 25
11.             [first_name] => John
12.             [last_name] => Anderson
13.             [username] => psychic
14.             [password] => c4k3roxx
15.         )
16.
17.     [Profile] => Array
18.         (
19.             [id] => 4
20.             [name] => Cool Blue
```

```
            [header_color] => aquamarine
            [user_id] = 25
        )

    [Comment] => Array
        (
            [0] => Array
                (
                    [id] => 247
                    [user_id] => 25
                    [body] => The hasMany assocation is nice to have.
                )

            [1] => Array
                (
                    [id] => 256
                    [user_id] => 25
                    [body] => The hasMany assocation is really nice to have.
                )

            [2] => Array
                (
                    [id] => 269
                    [user_id] => 25
                    [body] => The hasMany assocation is really, really nice to have.
                )

            [3] => Array
                (
                    [id] => 285
                    [user_id] => 25
                    [body] => The hasMany assocation is extremely nice to have.
                )

            [4] => Array
                (
                    [id] => 286
                    [user_id] => 25
                    [body] => The hasMany assocation is super nice to have.
                )

        )
)
```

你同样可以为 Comment 加上关联 User 对象的 belongsTo 关联，但是在文档中就不再详细描述了。

hasAndBelongsToMany 关联的定义与查询

我相信你已经掌握了简单的关联定义，让我们来看最后一个，也是最为复杂的关联关系：hasAndBelongsToMany(HABTM)。这个关联会让你头大的，不过也是最有用的。（译注：我倒认为应该数据库设计上尽量的避免出现大量的多对多关联，有的时候多对多关联可以比较简单拆分为两个一对多关联。）HABTM 关联也就是 3 张表的关联关系，关系数据库中应该说只存在多对一外键关联，所以如果要做多对多关联必然需要一张关联表来保存关联关系。

hasMany 和 hasAndBelongsToMany 的不同处在于，hasMany 关联所关联的对象只会属于本对象，不会同时属于其他对象。但是 HABTM 不同，所关联的对象同时会被其他对象所关联持有。比如 Post 和 Tag 之间的关联就是这种关系，一篇日志可以属于多个不同的 Tag，一个 Tag 也会包含多篇不同的日志。

为了实现多对多关联，首先要建立那张关联关系表（参照表）。除了"tags" "posts"表以外，根据 Cake 的命名约定，关联表的名字应该是[复数形式的 model1 名字]_[复数形式的 model2 名字]，至于两个 model 谁先谁后则根据字典排序法。

下面是一些示例：
Posts and Tags: posts_tags
Monkeys and IceCubes: ice_cubes_monkeys
Categories and Articles: articles_categories

关联表至少需要两个关联对象的外键字段，例如"post_id" 和 "tag_id"。当然你也可以加入一些其他的属性。

下面是生成的数据库脚本：

Here's what the SQL dumps will look like for our Posts HABTM Tags example:

```
--
-- Table structure for table `posts`
--

CREATE TABLE `posts` (
    `id` int(10) unsigned NOT NULL auto_increment,
    `user_id` int(10) default NULL,
    `title` varchar(50) default NULL,
```

```
    `body` text,
    `created` datetime default NULL,
    `modified` datetime default NULL,
    `status` tinyint(1) NOT NULL default '0',
    PRIMARY KEY   (`id`)
) TYPE=MyISAM;

-- --------------------------------------------------------

--
-- Table structure for table `posts_tags`
--

CREATE TABLE `posts_tags` (
    `post_id` int(10) unsigned NOT NULL default '0',
    `tag_id` int(10) unsigned NOT NULL default '0',
    PRIMARY KEY   (`post_id`,`tag_id`)
) TYPE=MyISAM;

-- --------------------------------------------------------

--
-- Table structure for table `tags`
--

CREATE TABLE `tags` (
    `id` int(10) unsigned NOT NULL auto_increment,
    `tag` varchar(100) default NULL,
    PRIMARY KEY   (`id`)
) TYPE=MyISAM;With our tables set up, let's define the association in the Post model:

/app/models/post.php hasAndBelongsToMany
<?php
class Post extends AppModel
{
      var $name = 'Post';
      var $hasAndBelongsToMany = array('Tag' =>
                                       array('className'       => 'Tag',
                                             'joinTable'       => 'posts_tags',
                                             'foreignKey'      => 'post_id',
                                             'associationForeignKey'=> 'tag_id',
                                             'conditions'      => '',
                                             'order'           => '',
                                             'limit'           => '',
```

```
                'uniq'              => true,
                'finderQuery'    => '',
                'deleteQuery'    => '',
            )
            );
}
?>
```

$hasAndBelongsToMany array 是定义 HABTM 关联的变量，简单介绍一下需要定义的 key：

- className (required)：关联对象类名。
- joinTable：如果你没有遵循 Cake 的命名约定建立关联表的话，则需要设置该 key 来指定关联表。
- foreignKey：注意和 associationForeignKey 的区别，这个是定义本 model 在关联表中的外键字段。当然也是仅在你没有遵循 Cake 命名约定的时候才需要。
- associationForeignKey：关联表中指向关联对象的外键字段名。
- conditions：关联对象限定条件。
- order：关联对象排序子句。
- limit：关联对象检索数量限制。
- uniq：设为 true 的话，重复的关联对象将被过滤掉。
- finderQuery：完整的关联对象检索语句。
- deleteQuery：完整的删除关联关系的 sql 语句。当你需要自己实现删除操作的时候可以使用该值。

```
1. 最后我们来看一下代码：
2.
3. $post = $this->Post->read(null, '2');
4. print_r($post);
5.
6. //output:
7.
8. Array
9. (
10.     [Post] => Array
11.         (
12.             [id] => 2
13.             [user_id] => 25
14.             [title] => Cake Model Associations
15.             [body] => Time saving, easy, and powerful.
16.             [created] => 2006-04-15 09:33:24
17.             [modified] => 2006-04-15 09:33:24
18.             [status] => 1
19.         )
20.
21.     [Tag] => Array
```

```
        (
            [0] => Array
                (
                    [id] => 247
                    [tag] => CakePHP
                )

            [1] => Array
                (
                    [id] => 256
                    [tag] => Powerful Software
                )
        )
)

```

保存关联对象

请记住一件非常重要的事情，当保存对象时，很多时候需要同时保存关联对象，比如当我们保存 Post 对象和它关联的 Comment 对象时，我们会同时用到 Post 和 Comment 两个 model 的操作。

抽象出来说，当关联的两个对象都没有持久化（即未保存在数据库中），你需要首先持久化主对象，或者是父对象。我们通过保存 Post 和关联的一条 Comment 这个场景来具体看看是如何操作的：

```
//------------Post Comment 都没有持久化------------

/app/controllers/posts_controller.php (partial)
function add()
{
    if (!emptyempty($this->data))
    {
        //We can save the Post data:
        //it should be in $this->data['Post']

        $this->Post->save($this->data);

        //Now, we'll need to save the Comment data
        //But first, we need to know the ID for the
        //Post we just saved...
```

```
16.
17.         $post_id = $this->Post->getLastInsertId();
18.
19.         //Now we add this information to the save data
20.         //and save the comment.
21.
22.         $this->data['Comment']['post_id'] = $post_id;
23.
24.         //Because our Post hasMany Comments, we can access
25.         //the Comment model through the Post model:
26.
27.         $this->Post->Comment->save($this->data);
28.
29.     }
30. }
31.
```

换一种情形，假设为现有的一篇 Post 添加一个新的 Comment 记录，你需要知道父对象的 ID。你可以通过 URL 来传递这个参数或者使用一个 Hidden 字段来提交。

```
1. /app/controllers/posts_controller.php (partial)
2. //Here's how it would look if the URL param is used...
3. function addComment($post_id)
4. {
5.     if (!emptyempty($this->data))
6.     {
7.         //You might want to make the $post_id data more safe,
8.         //but this will suffice for a working example..
9.
10.         $this->data['Comment']['post_id'] = $post_id;
11.
12.         //Because our Post hasMany Comments, we can access
13.         //the Comment model through the Post model:
14.
15.         $this->Post->Comment->save($this->data);
16.     }
17. }
18.
```

如果你使用 hidden 字段来提交 ID 这个参数，你需要对这个隐藏元素命名（如果你使用 HtmlHelper）来正确提交：

假设日志的 ID 我们这样来命名$post['Post']['id']

<?php echo $html->hidden('Comment/post_id', array('value' => $post['Post']['id'])); ?>

这样来命名的话，Post 对象的 ID 可以通过$this->data['Comment']['post_id']来访问，同样的通过 $this->Post->Comment->save($this->data)也能非常简单的调用。

当保存多个子对象时，采用一样的方法，只需要在一个循环中调用 save()方法就可以了（但是要记住使用 Model::create()方法来初始化对象）。

小结一下，无论是 belongsTo, hasOne 还是 hasMany 关联，在保存关联子对象时候都要记住把父对象的 ID 保存在子对象中。

保存 hasAndBelongsToMany 关联对象

如果定义关联一样，最复杂的莫过于 hasAndBelongsToMany 关联，hasOne, belongsTo, hasMany 这 3 种关联只需要很简单的保存一下关联对象外键 ID 就可以了。但是 hasAndBelongsToMany 却没有那么容易了，不过我们也做了些努力，使之尽可能变得简单些。继续我们的 Blog 的例子，我们需要保存一个 Post，并且关联一些 Tag。

实际项目中你需要有一个单独的 form 来创建新的 tag 然后来关联它们，不过为了叙述简单，我们假定已经创建完毕了，只介绍如何关联它们的动作。

当我们在 Cake 中保存一个 model，页面上 tag 的名字（假设你使用了 HtmlHelper）应该是这样的格式 'Model/field_name' 。好了，让我们开始看页面代码：

```
/app/views/posts/add.thtml Form for creating posts

                <h1>Write a New Post</h1>

                    <table>

                    <tr>

                        <td>Title:</td>
```

```
                        <td><?php echo $html->input('Post/title')?></td>

                    </tr>

                    <tr>

                        <td>Body:<td>

                        <td><?php echo $html->textarea('Post/title')?></td>

                    </tr>

                    <tr>

                    <td colspan="2">

                        <?php echo $html->hidden('Post/user_id',
array('value'=>$this->controller->Session->read('User.id')))?>

                        <?php echo $html->hidden('Post/status' , array('value'=>'0'))?>

                        <?php echo $html->submit('Save Post')?>

                    </td>

                    </tr>

                    </table>
```

上述页面仅仅创建了一个 Post 记录，我们还需要加入些代码来关联 tag：

```
/app/views/posts/add.thtml (Tag association code added)
<h1>Write a New Post</h1>
    <table>
        <tr>
            <td>Title:</td>
            <td><?php echo $html->input('Post/title')?></td>
        </tr>
        <tr>
            <td>Body:</td>
            <td><?php echo $html->textarea('Post/title')?></td>
        </tr>
```

```
        <tr>
            <td>Related Tags:</td>
            <td><?php echo $html->selectTag('Tag/Tag', $tags, null, array('multiple' =>
'multiple')) ?>
            </td>
        </tr>

        <tr>
            <td colspan="2">
                <?php echo $html->hidden('Post/user_id',
array('value'=>$this->controller->Session->read('User.id')))?>
                <?php echo $html->hidden('Post/status' , array('value'=>'0'))?>
                <?php echo $html->submit('Save Post')?>
            </td>
        </tr>
</table>
```

我们在 controller 中通过调用 $this->Post->save() 来保存当前 Post 以及关联的 tag 信息，页面元素的命名必须是这样的格式 "Tag/Tag" （Cake Tag Render 之后实际的 Html Tag 名字为 'data[ModelName][ModelName][]' 这样的格式）。提交的数据必须是单个 ID，或者是一个 ID 的 array。因为我们使用了一个复选框，所以这里提交的是一个 ID 的 array。

$tags 变量是一个 array，包含了复选框所需要的 tag 的 ID 以及 Name 信息。

使用 bindModel() 和 unbindModel() 实时地改变关联关系

有的时候可能你会需要实时地，动态的改变 model 的关联关系，比如在一个异常情况下。特别是 LazyLoad 的问题，有的时候我们并需要一个完整的 model，所以我们可以使用 bindModel() 和 unbindModel()来绑定或者解除绑定关联对象。

代码说话，Start：

```
1. leader.php and follower.php
2. <?php
3.
4. class Leader extends AppModel
5. {
6.     var $name = 'Leader';
7.
8.     var $hasMany = array(
9.         'Follower' => array(
10.             'className' => 'Follower',
11.             'order'     => 'Follower.rank'
```

```
        )
    );
}

?>

<?php

class Follower extends AppModel
{
    var $name = 'Follower';
}

?>
```

上述两个 Model，在 Leader Model 中，有一个 hasMany 关联，定义了 "Leader hasMany Followers" 这样的关系。下面我们演示如何在 Controller 中动态地解除这种关联绑定。

```
leaders_controller.php (partial)
function someAction()
{
    //This fetches Leaders, and their associated Followers
    $this->Leader->findAll();

    //Let's remove the hasMany...
    $this->Leader->unbindModel(array('hasMany' => array('Follower')));

    //Now a using a find function will return Leaders, with no Followers
    $this->Leader->findAll();

    //NOTE: unbindModel only affects the very next find function.
    //注意：unbindModel 方法只作用一次，第二次 find 方法调用时则仍然是关联关系有效的
    //An additional find call will use the configured association information.

    //We've already used findAll() after unbindModel(), so this will fetch
    //Leaders with associated Followers once again...
    $this->Leader->findAll();
}
```

```
21.
```

对于其他各种关联的 unbindModel()的用法是类似的，你只需要更改名字和类型就可以了，下面介绍一些基础的 Usage：

通用的 unbindModel()

```
$this->Model->unbindModel(array('associationType' => array('associatedModelClassName')));
```

掌握了如何动态的解除绑定之后，让我们看看如何动态的绑定关联关系。

```
1. leaders_controller.php (partial)
2. funciton anotherAction()
3. {
4.     //There is no Leader hasMany Principles in the leader.php model file, so
5.     //a find here, only fetches Leaders.
6.     $this->Leader->findAll();
7.
8.     //Let's use bindModel() to add a new association to the Principle model:
9.     $this->Leader->bindModel(
10.         array('hasMany' => array(
11.                 'Principle' => array(
12.                     'className' => 'Principle'
13.                 )
14.             )
15.         )
16.     );
17.
18.     //Now that we're associated correctly, we can use a single find function
19.     //to fetch Leaders with their associated principles:
20.     $this->Leader->findAll();
21. }
22.
```

bindModel()方法不单能创建一个关联，同样可以用来动态的修改一个关联。

```
1. 下面是通常的用法：
2.
3. Generic bindModel() example
4. $this->Model->bindModel(
5.         array('associationName' => array(
6.                 'associatedModelClassName' => array(
7.                     // normal association keys go here...
8.                 )
9.             )
10.         )
11.     );
12.
```

注意：这些的前提是你的数据库表中的外键关联等已经正确设置。