Plugin Development

making reusable libraries

Goals of Talk

- Familiarize you with CakePHP's Plugin system
- How plugins work and how to build one
- How to design plugins to be reusable

Who am I

- Mark Story - from Canada
- CakePHP Core member since May 08
- Author of DebugKit
- 1.5 years CakePHP experience
- 4 years PHP experience

What is a plugin?

- Plugins are ‘mini’ applications or bundles of functionality.
- Generic enough to be reused.
- Focused on a particular task or area of functionality.

Introduction to Plugins

- History of plugins
  - Introduced in CakePHP 1.0
  - Plugin integration expanded in 1.1.x.x series.
  - Not all Cake objects were accessible, problems with webservices and plugins.
  - No css, js or images.

Plugins Today

- Plugin system rebuilt in 1.2
- All Cake objects can be accessed from main app. Or from other plugins.
- Plugins can have vendors, css, js and images!

Mini Applications?

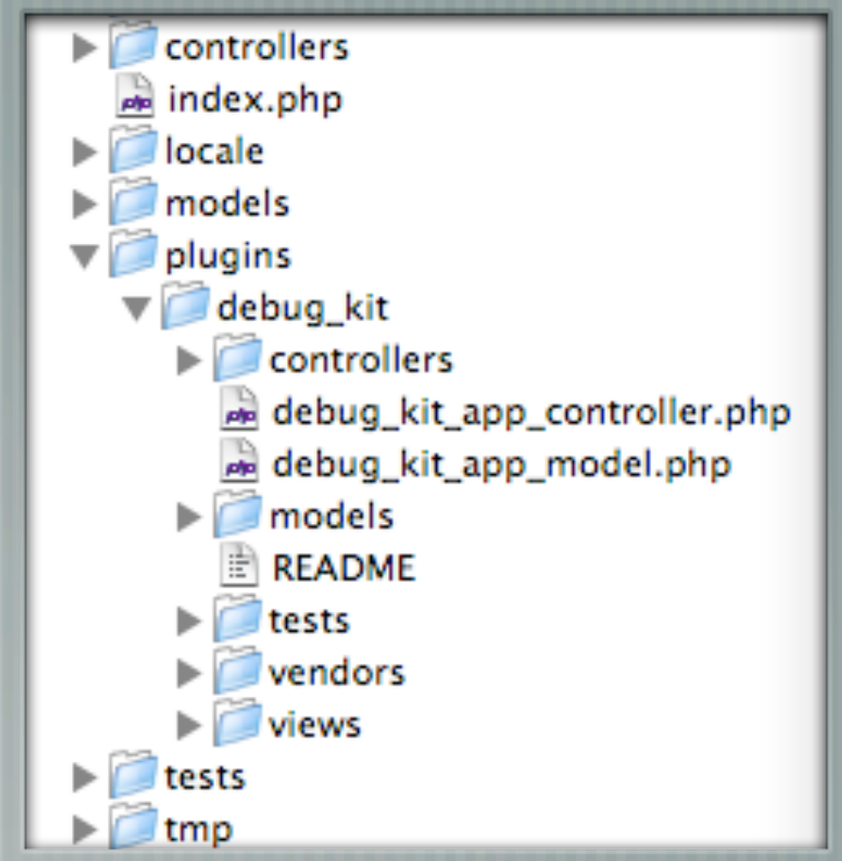
- Not always complete applications, but a set of tasks or related objects that can be reused in many places.
- For example a blog/Mini CMS, login system, debug kit, photo sharing, ACL management, graph generation.
- Basically anything that you would need in more than one application and is a generic enough set of functions to easily be reused.

Why bother with plugins?

- Save you time and money in the long term.
- Easier to share. Plugins create easy to install bricks of functionality making them ideal to share.
- If well designed they can be used over and over again, to make duct tape applications.

File Structure of a plugin

- All plugins in `app/plugins`
- plugin_name contains plugin files.
- plugins have very similar file layout to app.



Plugin Naming Conventions

- Plugin directory is lower cased & underscored
- Plugin name is CamelCased.
- Common to have plugin name as a prefix for all classes to avoid namespace conflicts with app. Keep prefix's short.

Plugin Naming Conventions

Example Graphs plugin.

- app/plugins/graphs
- GraphsAppModel - graphs_app_model.php
- GraphsAppController - graphs_app_controller.php

Plugin Naming Conventions

- Prefixing classes
  - Graph model
    - GraphsGraph - graphs_graph.php
  - Graphs controller
  - GraphsGraphsController - graphs_graphs_controller.php

Plugin Naming Conventions

- All plugins need `plugin_name_app_controller.php` and `plugin_name_app_model.php`
- These classes should inherit from `AppController` and `AppModel` respectively.

Plugin Routes

Plugin key

- plugin key lets you specify that the link/route is a plugin route.
- `array('plugin' => 'graphs', 'controller' => 'graphs', 'action' => 'index')`

Plugin Tests & Fixtures

- Plugins can have and should have their own tests.
- Test cases work exactly like app tests.
- Fixtures used in tests need a PluginName prefix
  - ie. `var $fixtures = array('debugKit.post');`

Practical Plugins

Loading and using plugin files.

Loading Plugin files

Using App::import()

```
App::import('Model', 'Blog.Post');

App::import('Component', 'DebugKit.Toolbar');

App::import('Behavior', 'Blog.Taggable');

App::import('View', 'DebugKit.Debug');

App::import('Helper', 'Graphs.FusionChart');

App::import('Vendor', 'DebugKit.FireCake');
```

Loading Plugin files

Using class properties

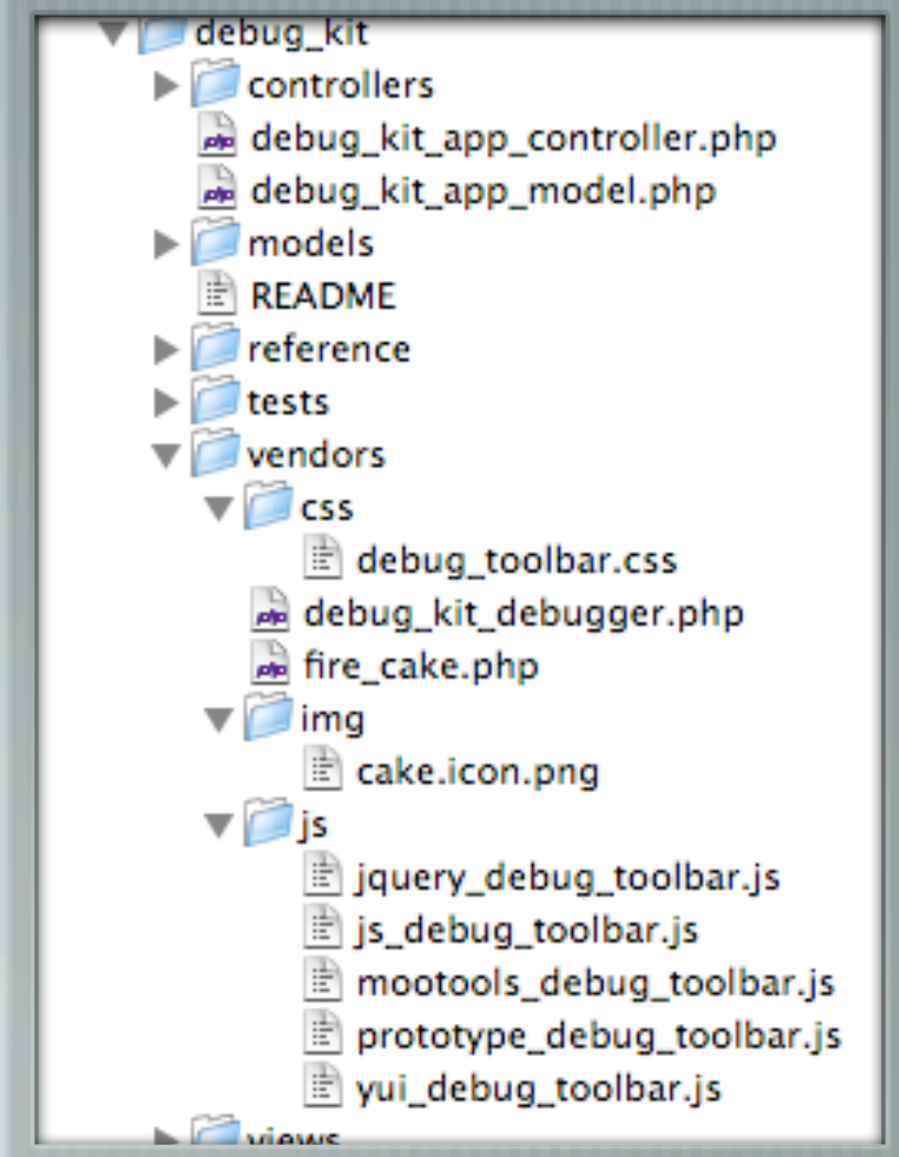
```
var $helpers = array('Html', 'Graphs.FusionChart');

var $actsAs = array('Blog.Taggable', 'Containable');

var $components = array('DebugKit.Toolbar');

var $view = 'Graphs.Csv';
```

Loading Plugin files

- Plugin.Class is used everywhere.
- Inside a plugin you do not need to use pluginName.class It is assumed you want the plugin class.
- However, you can use Plugin.Class to refer to another plugin!

CSS, JS and Images oh my!

- All plugin assets must be in `plugin_name/vendors`
- paths to plugin assets are slightly different.



Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin asset paths.

```
//link to an image.
$html->image('/debug_kit/img/cake_icon.png', array('alt' => 'icon of power!'));

//link to a css file.
$html->css('/debug_kit/css/debug_toolbar');

//link to a js file.
$javascript->link('/debug_kit/js/js_debug_toolbar');
```

Plugin Shells

- Plugins can have shells!
- Place plugin shells in `plugin_name/vendors/shells`
- Access plugin shells via `cake shellName`. Naming conflicts can be resolved by using `pluginName.shellName`

Interplugin communication

- Communication between plugins can be done with `requestAction()`
- Not always the best option. But it gets the job done.

Squeezing requestAction()

- If you need to use requestAction() between plugins, remember to use array() urls and not string urls.
- Array urls skip all the route parsing steps.

Squeezing requestAction()

```
$posts = $this->requestAction('/blog/posts/get_recent');

$posts = $this->requestAction(array(
    'plugin' => 'blog',
    'controller' => 'posts',
    'action' => 'get_recent'
));
```

I hate pink.

I hate pink.

- So you've found an amazing plugin
- Only problem is all the views are pink & black.
- Or the markup has a serious case of 'divitis'.

Replacing Plugin Views

- By adding the same named directory to app/views you can override/replace some or all of the view files for a plugin.
- You can also use ThemeView. Its one stop shopping for view replacement.

DebugKit

A case study in plugin development.

Debug Kit Background

- Was initially planned as set of enhancements to core Debugger.
- Debug Kit was designed as a CakePHP counterpart to the symfony / django debug tools.

Transition to a plugin

- When planning out the enhancements to Debugger it became clear that not all the planned code would fit inside Debugger.
- Providing it as a set of loose classes would make it difficult to use and install.
- By creating a plugin, it became a single package that people could easily use.

Benefits of Plugin

The benefits of creating the DebugKit as a plugin have been numerous.

- Far more power.
- Planned features added as DebugKitDebugger.
- Far easier for others to use.

DebugKit Design

- DebugKit has no controllers or models.
- All functionality is provided through Component, View, Helpers and Vendors files.
- Vendor files provide Custom Debugger as well as FirePHP support.

DebugKit Design

- Plugin is easy to use, and unobtrusive to primary application.
- Can be easily attached to any application.
- Leverages plugin assets to provide js, css and images.

DebugKit Features

- Adds many features to the host application seamlessly and without disturbing main app.
- DebugKit is extensible, custom panels, and toolbar helper back ends can be added.

Designing plugins

The Basics

- What is it going to do?
- How is it going to provide its functionality?
- What configuration will be required?
- For internal use only? or for wider distribution?

Configuration & Setup

- Configuration is a necessary evil.
- Most plugins will require some amount of work to setup.
  Can build an installer to minimize pain factor.
- Most of the time an informative README will suffice.
- Remember that your users are other developers.

Configuration

Routing

- include custom routing in plugin/config/routes.php. This makes them easy to find and easy to include.
- There is no core method to load plugin routes. However a simple `include()` works in 90% of cases. Again documentation is key.

Configuration

- Configure settings.
  - Good practice to use `Configure::write()`/`Configure::read()`
  - Nest your settings under pluginName.
    - e.x. UserManager.UserClass

Configuration

- If you have a lot of config settings, try storing them in config/bootstrap.php so they can easily be included.
- Less is more. Use the fewest number of configuration settings possible.
- Let your plugin API be your configuration.

Distributing plugins

- If you choose to distribute your work, some thing to consider.
  - SQL and required database schema.
  - Documentation.
  - Hosting / SCM.
  - Licensing.

Distributing plugin SQL

SQL dump

- Simple and easy to use.
- Tied to one database server in many cases.
- Migrations can be painful.

Distributing plugin SQL

Cake Schema

- Database independent.
- Easy to do migrations.
- Requires shell access, or custom install/upgrade functions.

Documentation

- Need for documentation changes depending on how your plugin works or interacts with primary app.
- Good doc blocks are always a good place to start.
- Wiki or blog articles.

Hosting / SCM

- Every good plugin needs a good home. Luckily there are plenty of options.
  - http://thechaw.com
  - http://github.com
  - http://cakeforge.org
  - http://beanstalk.org

Licensing

Licensing

- Open Source license like MIT / New BSD or LGPL will keep users the happiest.
- More restrictive licenses are an option as well. But they change how you host and distribute it as well.

Challenges of plugins

- Biggest challenge is keeping them specific yet generic.
- Can be difficult to strike a balance between general solution and a specific implementation.
- Keeping everyone happy is hard.
- Focus on one thing, and do it well.

Thank you

Thank you for coming and listening.

Questions?