HANDBOOK

Alessandro Ronchi

MAGENTO

BEST PRACTICES

Magento Best Practices Handbook

A collection of practical advices to develop with Magento the right way.

Alessandro Ronchi

This book is for sale at http://leanpub.com/magebp

This version was published on 2015-09-07

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Alessandro Ronchi by spreading the word about this book on Twitter!

The suggested hashtag for this book is #magebp.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#magebp

Three things every developer should do in life: plant a tree, have a child, write a book, solve that damned off-by-one error.

Contents

Foreword

If you are reading this book the chances are that you are a developer. Or a programmer. Or a coder. Or what else?

Few of us refer to themselves as *software craftsman* or *code author* but this is likely the most correct expression representing what our role should be.

Programming is not a mechanical task: take ten programmers, ask them to develop the "Game of Life"[1] and you will probably see ten different approaches and implementations.

Being a human activity[2], programming is subjected to personal experience, attitude, smartness and circumstances.

Knowing a programming language doesn't automatically make us code authors just like knowing grammar and syntax doesn't mean we are able to write a successful novel.

Kent Beck[3] once said: *"I'm just a good programmer with great habits."* Only knowing and applying the best programming practices and adopting good habits can really make us strive to become software craftsmen.

[1] See the definition of the Game of Life on Wikipedia.

[2] I recommend reading The Art of Programming article by Erika Heidi.

[3] http://en.wikipedia.org/wiki/Kent_Beck

Preface

Magento is a huge framework and the path from learning to mastering can take a long time and require a lot of mental effort and practice.

Even after years of day-by-day work with Magento, we can realize there is always something new to learn; this is why I coined the following motto:

"There are at least two ways of developing things in Magento the best of which is usually the third."

In other words Magento offers so many ways to be extended that it's easy to stop to the first option not considering better ones.

This book was conceived to be *a light in dark places, when all other lights go out.*

In other words it aims to be a companion containing most of the discoveries, practices and habits I collected during my long journey with Magento.

It's first of all a book for myself, to learn by teaching and reinforce knowledge and to give birth to discussion with coworkers and community pals.

Michael Abrash[4] once said: *"None of us learn in a vacuum; we all stand on the shoulders of giants such as Wirth and Knuth and thousands of others. Lend your shoulders to building the future!"*

This is particularly true for us belonging to the *share generation*: thanks to the internet we have the chance to instantly access others' knowledge and the privilege and responsibility to use it to build a better future.

[4] http://en.wikipedia.org/wiki/Michael_Abrash

1. Introduction

This book will cover different specific topics about Magento architecture and development.

Some chapters are very practical and guide towards the development or usage of an effective solution to common problems, such as:

- Using the best Tools
- Working with Magento Runtime
- Logging
- Extending native Autoloading
- Deploying

Other chapters are more theoretical and cover topics such as:

- Developing effective MVC Components
- How to work with Data
- How to extend Magento

1.1 Why and How to read this book

If you are interested in getting things done the right way in Magento this book has something to offer.

To read this book a basic knowledge of Magento fundamentals is required.

I warmly suggest you to read this book in front of your PC in order to be able to put your hands on Magento code and put into practice what you read along the way.

Each chapter is focused on a single concern, meaning you are not forced to read this book cover-to-cover.

1.2 Conventions

Unless differently specified, the Magento version I always refer to in this book is Magento Community Edition 1.8 because at the time of writing it still is the version which the Certification exam is based on.

Although this book doesn't aim to be a Certification companion it will leverage your knowledge of some internal aspects of Magento thus being of some help if you are considering to take the exam.

I will use the following placeholders to refer to some values specific for any Magento installation:

- `<base_dir>` - indicates the root directory where Magento is installed;
- `<code_core>` - indicates the `<base_dir>/app/code/core` directory;
- `<code_local>` - indicates the `<base_dir>/app/code/local` directory;
- `<design_dir>` - indicates the `<base_dir>/app/design` directory where layout and template files are generally located.

1.3 Feedback

Your feedback not only is appreciated but is vital being this my first book.

Will it also be the last one depends on you, so please let me know whether you appreciated it or not, what you liked and disliked and whether there were topics you expected me to cover more in depth.

The fastest way to reach me is sending an email to aleron75@gmail.com.

2. Magento Runtime

This chapter explains different possible ways of setting up a fully initialized Magento application environment, more often referred to as *runtime*.

By showing different approaches we will have the opportunity to analyze some internal mechanisms which are at the basis of Magento dispatch flow.

2.1 A bare minimum PHP script

The fastest way of initializing a Magento runtime consists of writing a shell script like the one shown below:

```
1  <?php
2  // Prevent this script to be called via HTTP
3  if (isset($_SERVER['REQUEST_METHOD']))
4  {
5      die('Permission denied.');
6  }
7
8  // Avoid any time limit
9  set_time_limit(0);
10
11 // Avoid any memory limit
12 ini_set('memory_limit', -1);
13
14 // Include bootstrap code and Mage class
15 require_once 'app/Mage.php';
16
17 // Enable developer mode
18 Mage::setIsDeveloperMode(true);
19
20 // Set the default file creation mask
```

```
21 umask(0);
22
23 // Init application with default store
24 Mage::app();
```

The above script initializes a default store scope Magento application.

But it still lacks something, that is the initialization of additional configuration area and of proper user session.

A **configuration area** is a specific node of Magento config XML where values are looked up. Native areas are `global`, `frontend`, `adminhtml`, `admin` and `install`. The `global` area is the only one which is always loaded during application initialization. Other areas are loaded depending on the **context**; in a normal dispatch flow it is determined by the `preDispatch()` method of current controller.

In the case of a shell script we are outside of a normal dispatch flow so we have to force the initialization of proper store, configuration area and user session. These concepts are bounded and should be correctly initialized not to incur in undesired behaviour.

The following additional code can be alternatively used after the `Mage::app()` invocation to initialize an administrator or a customer app.

In case of a customer:

```
1 // Load frontend config area
2 Mage::app()->loadArea(Mage_Core_Model_App_Area::AREA_FRONTEND);
3
4 // Initialize customer session
5 $userModel = Mage::getModel('customer/customer');
6 $customer = $userModel->load(<customer_id>);
7 $customerSession = Mage::getSingleton('customer/session');
8 $customerSession->setCustomerAsLoggedIn($customer);
9 $customerSession->renewSession();
```

In case of an administrator:

```
// Init admin store
Mage::app()->setCurrentStore(Mage_Core_Model_App::ADMIN_STORE_ID);

// Load adminhtml config area
Mage::app()->loadArea(Mage_Core_Model_App_Area::AREA_ADMINHTML);

// Initialize administrator session
$userModel = Mage::getModel('admin/user');
$administrator = $userModel->load(<administrator_id>);
$adminSession = Mage::getSingleton('admin/session');
$adminSession->renewSession();
$adminSession->setUser($administrator);
$acl = Mage::getResourceModel('admin/acl')->loadAcl();
$adminSession->setAcl($acl);
```

To save time, you can access the above snippets directly from my GitHub Gist:

- Customer: [1]
- Administrator: [2]

2.2 The more structured Mage Shell script

A more structured way to initialize a Magento environment consists of writing a so called *Mage Shell* script, that is a single PHP file (let's call it `runtime.php`) inside the `<base_dir>/shell` directory whose basic structure is shown below:

[1] https://gist.github.com/aleron75/190b25ea621c14a21d6b
[2] https://gist.github.com/aleron75/9d6609e99153d19461f3

```
<?php
require_once 'abstract.php';

class Mage_Shell_Runtime extends Mage_Shell_Abstract
{
  public function run()
  {
  }
}

$shell = new Mage_Shell_Runtime();
$shell->run();
```

The above code declares a class which extends `Mage_Shell_Abstract`, implements the mandatory `run()` method and calls it on an instance of our brand new `Mage_Shell_Runtime` class.

Given we open a terminal in the `<base_dir>` directory, we can execute the `runtime.php` as shown below;

```
$ php -f shell/runtime.php
```

Executing the script won't give any output because the `run()` method defined so far does nothing at all; but at this point we have a fully functional application runtime initialized.

Note that since we are executing our code outside the normal routing dispatch flow, at this point only the global configuration area is loaded.

Before examining how to complete runtime initialization based on our needs, let's inspect the code of `Mage_Shell_Abstract` because it reveals some interesting internal aspects.

2.2.1 The store scope

As we can see in the declaration of `Mage_Shell_Abstract`'s protected variables, the application is initialized by default to bootstrap an `admin` store scope:

```php
/**
 * Initialize application with code (store, website code)
 *
 * @var string
 */
protected $_appCode     = 'admin';

/**
 * Initialize application code type (store, website, store_group)
 *
 * @var string
 */
protected $_appType     = 'store';

...

/**
 * Initialize application and parse input parameters
 *
 */
public function __construct()
{
  ...
  Mage::app($this->_appCode, $this->_appType);
  ...
}
```

Obviously we can initialize a different scope overriding the value of the protected variables but it's important to remember that the default one is `admin` especially when accessing configuration values which are affected by the scope.

2.2.2 PHP configuration options

Another interesting aspect of the `Mage_Shell_Abstract` class is the fact that at some point in the constructor, the `_applyPhpVariables()` method is called.

This method sets some PHP configuration options, which are valid only during the

script's execution, taking their value from the `.htaccess` file we can find in the Magento base directory, as shown below:

```
// file: <base_dir>/shell/abstract.php
// class: Mage_Shell_Abstract
protected function _applyPhpVariables()
{
  $htaccess = $this->_getRootPath() . '.htaccess';
  if (file_exists($htaccess)) {
    // parse htaccess file
    $data = file_get_contents($htaccess);
    $matches = array();
    preg_match_all('#^\s+?php_value\s+([a-z_]+)\s+(.+)$#siUm', $data, $match\
es, PREG_SET_ORDER);
    if ($matches) {
      foreach ($matches as $match) {
        @ini_set($match[1], str_replace("\r", '', $match[2]));
      }
    }
    preg_match_all('#^\s+?php_flag\s+([a-z_]+)\s+(.+)$#siUm', $data, $matche\
s, PREG_SET_ORDER);
    if ($matches) {
      foreach ($matches as $match) {
        @ini_set($match[1], str_replace("\r", '', $match[2]));
      }
    }
  }
}
```

This means that if we want to overwrite some configuration options, typically the `memory_limit` and `max_execution_time`, without changing the `.htaccess`, we can override the `_applyPhpVariables()` method as shown below:

```
// file: <base_dir>/shell/runtime.php
// class: Mage_Shell_Runtime
protected function _applyPhpVariables()
{
  parent::_applyPhpVariables();

  // Avoid any time limit
  set_time_limit(0);

  // Avoid any memory limit
  ini_set('memory_limit', -1);
}
```

The intuitive reason why we shouldn't change those options in the `.htaccess` file in the Magento base directory is that it would affect the entire application, not only the one bootstrapped by our shell script.

2.2.3 Input arguments

Let's conclude the analysis of the `Mage_Shell_Abstract` class with some words about input arguments, starting from the sequence of methods called by the constructor:

```
/**
 * Initialize application and parse input parameters
 *
 */
public function __construct()
{
  ...
  $this->_parseArgs();
  ...
  $this->_validate();
  ...
}
```

As its name suggests, the `_parseArgs()` is responsible for parsing input arguments, storing them in the protected `$_args` array variable; we can then access parameters by calling the public `getArg()` passing the argument name.

For example, given the command:

```
$ php shell/runtime.php flagParam -valueParam 1 -valueParamWithSpace "1 2 3"
```

the protected `$_args` array variable will be initialized as follows:

```
array (
  'flagParam' => true,
  'valueParam' => '1',
  'valueParamWithSpace' => '1 2 3',
)
```

Note that flag parameters can't be passed after value parameters and are identified by the absence of the preceding '-' character.

The `_validate()` method is the place where we should place our *logic of validation* for parameters; if, say, a parameter is mandatory, it's here that we should check if it has been passed to the shell command and, if not, exiting with an explanation message:

```
protected function _validate()
{
  parent::_validate();

  if (!$this->getArg('mandatory_parameter'))
  {
    exit("The mandatory_parameter should be specified");
  }
}
```

We should always remember to first call the `parent::_validate()` because it avoids the script to be run from a browser for security reasons as shown below:

```
// Method from Mage_Shell_Abstract
protected function _validate()
{
  if (isset($_SERVER['REQUEST_METHOD'])) {
    die('This script cannot be run from Browser. This is the shell script.');
  }
}
```

2.2.4 Additional application initialization

During a normal routing dispatch flow the application is initialized and the request is processed, determining the context in which the environment is fully initialized. But inside a shell script there isn't any request to process so the context can't be fully determined, leaving application initialization incomplete.

Actually we can write and check code inside an incompletely initialized application runtime but it is important to keep in mind that at this point something may not work as expected.

If, for example, an event observer is defined under the `frontend` area and we don't explicitly load the corresponding area part somewhere in our shell script, the event won't be dispatched and the observer will never be triggered.

Loading a specific config XML area is quite easy and the best place to do that is in the `_construct()` method:

```
/**
 * Additional initialize instruction
 *
 * @return Mage_Shell_Abstract
 */
protected function _construct()
{
  parent::_construct();
  Mage::app()->loadArea(Mage_Core_Model_App_Area::AREA_FRONTEND);
  return $this;
}
```

Again in our _construct() method we can place the code to impersonate a user which can be alternatively a customer or an administrator.

In case of a customer:

```
$userModel = Mage::getModel('customer/customer');
$customer = $userModel->load(<customer_id>);
$customerSession = Mage::getSingleton('customer/session');
$customerSession->setCustomerAsLoggedIn($customer);
$customerSession->renewSession();
```

In case of an administrator:

```
$userModel = Mage::getModel('admin/user');
$administrator = $userModel->load(<administrator_id>);
$adminSession = Mage::getSingleton('admin/session');
$adminSession->renewSession();
$adminSession->setUser($administrator);
$acl = Mage::getResourceModel('admin/acl')->loadAcl();
$adminSession->setAcl($acl);
```

Remember that outside the normal dispatch flow, being consistent with application initialization is up to us, combining proper store and configuration area with proper user session. Otherwise the result of execution can be unexpected.

2.3 The normal dispatch flow

The normal routing dispatch flow is called in action by calling a custom route, such as:

`http://<hostname>/<frontName>/<controller>/<action>`

Once done, we can count on a fully initialized application runtime environment with the default store configuration loaded and an anonymous user session.

Given that we extended a frontend controller, both global and frontend configuration areas are loaded in this case.

To impersonate a specific customer we can log in through the login form typically served by the following URL:

```
http://<hostname>/customer/account/login
```

Once logged in, our customer session will be initialized.

Impersonating an admin user requires us to define the entry point action within a controller extending `Mage_Adminhtml_Controller_Action` class and configure a custom menu item and a proper *ACL* to have access to the corresponding route.

Then we can log in the Admin Panel, identify the custom menu item, click on it and finally execute our entry point action.

At that point, we have our action executed into a fully initialized application runtime environment with the admin store configuration loaded and an admin user session.

Note that since we have extended an adminhtml controller, this time the global and adminhtml configuration areas are loaded.

Explaining the details of Magento routing dispatch flow is out of the scope of this book. To examine it in depth I suggest reading the brilliant Grokking Magento[3] book by Vinai Kopp.

2.4 Summary

We have seen that there are different ways to initialize a Magento runtime environment, from faster approaches to more structured ones.

Doing that we had the opportunity to analyze and discuss some interesting aspects of the framework, such as different config areas and user sessions.

As we will see in next chapters, knowing how Magento works internally is the real key to understand, appreciate and exploit it for the best.

[3] https://shop.vinaikopp.com/grokking-magento