## Модули и помощники

Codeception использует модульную архитектуру чтобы создать комфортное тестовое окружение для каждого написанного вами набора тестов.  
Модули позволяют выбрать действия (actions) и утверждения (assertions), которые будут выполнены в тестах.

Все действия и утверждения, которые будут выполнены "объектом-парнем" объявлены в модулях. Это может выглядеть будто Codeception ограничивает вас в тестировании, но это не правда. Вы можете расширить набор тестов своими собственными действиями и утверждениями, описав их в своем модуле.

Давайте посмотрим на этот тест.

<?php

$I = new TestGuy($scenario);

$I->amOnPage('/');

$I->see('Hello');

$I->seeInDatabase('users', array('id' => 1));

$I->seeFileFound('running.lock');

?>

Тест может оперировать различными сущностями: страница может быть загружена с помощью модуля Symfony1, база данных проверена с помощью модуля Db, а наличие файла может быть с помощью модуля Filesystem.

Модули привязан к классам "Парней" c помощью файла настроек тестового набора.  
Например, в tests/functional.suite.yml мы можем увидеть.

class\_name: TestGuy

modules:

enabled: [Symfony1, Db, Filesystem]

У класса TestGuy есть методы объявленные в модулях. На самом деле, класс не содержит все эти методы, но он выступает в качестве прокси для них. Он знает какой модуль выполняет данное действие и передает ему параметры. Чтобы ваша IDE могла видеть все методы класса TestGuy используйте команду build. Команда генерирует определение класса TestGuy, копируя сигнатуры методов из настроенных модулей.

### Стандартные модули

### https://github.com/Codeception/Codeception/tree/master/docs/modules

### Помощники

Codeception не ограничивает вас в использовании модулей только из основного репозитория. Несомненно, в вашем проекте вы можете захотеть добавить собственные действия в тестовые наборы. Запустив команду bootstrap, Codeception сгенерирует для вас три заготовленных модуля, по одному для каждого набора. Эти модули называются 'Помощниками', их можно найти по следующему пути tests/helpers.

Это хорошая идея определять недостающие действия или команды утверждения (assertion) в помощниках.

Скажем, мы собираемся расширить класс TestHelper. По умолчанию он связан с классом TestGuy и набором функциональных тестов.

Что касается действий, все очень просто. Каждое действие, которое вы определяете является публичной функцией. Опишите публичную функцию, запустите команду build и вы увидите, что добавлена новая функция в класс TestGuy. Примечание: публичные методы с префиксом \_ рассматриваются как скрытые и не будут добавлены в класс вашего "Парня".

С утверждениями немного сложнее. В первую очередь, рекомендуется добавлять префикс see или donSee для всех утверждающих действий. В философии Codeception, все тесты выполняются людьми, например "парнями". Ожидаемый результат, который они видят (или не видят) это то, что мы используем для утверждения.

Называйте ваши утверждения подобным образом:

<?php

seePageReloaded();

seeClassIsLoaded($classname);

dontSeeUserExist($user);

?>

И затем используйте их в своих тестах:

<?php

$I = new TestGuy($scenario);

$I->seePageReloaded();

$I->seeClassIsLoaded('TestGuy');

$I->dontSeeUserExist($user);

?>

Каждая see или dontSee функция требует по крайней мере одно утверждение. Codeception использует утверждения из PHPUnit.

Вы можете объявлять утверждения используя asertXXX методы модуля.  
Codeception использует утверждения из PHPUnit. Таким образом, в случае когда вам не хватает некоторых утверждений, вы можете использовать статичные методы PHPUnit из класса PHPUnit\_Framework\_Assert.

<?php

function seeClassExist($class)

{

$this->assertTrue(class\_exists($class));

// или

\PHPUnit\_Framework\_Assert::assertTrue(class\_exists($class));

}

?>

В ваших помощниках вы можете использовать эти утверждения:

<?php

function seeCanCheckEverything($thing)

{

$this->assertTrue(isset($thing), "this thing is set");

$this->assertFalse(empty($any), "this thing is not empty");

$this->assertNotNull($thing, "this thing is not null");

$this->assertContains("world", $thing, "this thing contains 'world'");

$this->assertNotContains("bye", $thing, "this thing doesn`t contain 'bye'");

$this->assertEquals("hello world", $thing, "this thing is 'Hello world'!");

}

?>

Просто введите $this->assert чтобы увидеть их все.

Так же, у каждого модуля есть специальные методы $this->assert и $this->assertNot. Оба метода принимают одинаковые параметры и полезны если вам нужно объявить как положительное так и отрицательное утверждение в вашем модуле. Эти функции принимают массив как параметр, где первое значение массива это название функции утверждения из PHPUnit.

<?php

$this->assert(array('Equals',$int,3));

$this->assertNot(array('internalType',$int,'bool'));

$this->assert(array('Contains', array(3,5,9), 3));

?>

Давайте посмотрим как определить оба see и dontSee действия без дублирования кода.

assert($this->proceedSeeClassExist($class));

}

public function dontSeeClassExist($class)

{

$this->assertNot($this->proceedSeeClassExist($class));

}

protected function proceedSeeClassExist($class)

{

return array('True',get\_class($class));

}

?>

Для dontSeeClassExist, будет вызван assertFalse.

#### Разрешение конфликтов

Что случится если у вас есть два модуля, которые содержат действия с одинаковыми названиями?  
Codeception позволяет вам переопределять действия меняя порядок модулей.  
Действие из второго модуля будет загружено, а действие из первого модуля будет проигнорировано.  
Порядок модулей описывается в файле настроек тестового набора.

#### Взаимодействие Модулей

Возможно, что вы захотите получить доступ к внутренним свойствам или функциям из другого модуля. Например, для вашего модуля понадобилось соединение из модуля Doctrine или браузер из модуля Symfony.

Модули могут взаимодействовать с другими модулями через метод getModule. Пожалуйста, обратите внимание, что этот метод может выбросить исключение если требуемый модуль не был загружен.

Давайте представим, что мы пишем модуль, который переподключается к базе данных. Предположим он использует значение свойства dbh из модуля Db.

<?php

function reconnectToDatabase() {

$dbh = $this->getModule('Db')->dbh;

$dbh->close();

$dbh->open();

}

?>

Используя функцию getModule вы получаете доступ ко всем публичным методам и свойствам запрошенного модуля.  
Свойство dbh было специально объявлено как публичное чтобы быть доступным для других модулей.  
Эта техника так может быть полезна если вам нужно выполнить последовательность действий взятых из других модулей.  
Например:

<?php

function seeConfigFilesCreated()

{

$filesystem = $this->getModule('Filesystem');

$filesystem->seeFileFound('codeception.yml');

$filesystem->openFile('codeception.yml');

$filesystem->seeInFile('paths');

}

?>

#### Неопределенные действия в помощниках

В случаи когда у вас есть действие, которое еще не определено, вы можете автоматически создать метод-заглушку для него в соответствующем помощнике. Чтобы это сделать, вы можете использовать команду analyse, которая сканирует все тесты и ищет несуществующие действия в любом подключенном модуле.

Таким образом, вы можете доверить написание тестов технически не подкованным людям или специалистам по качеству. В случае если им будет недоставать некоторых действий они объявят их в тесте.

<?php

$I->doManyCoolThings();

?>

Запустив команду analyze у вас спросят, хотите ли вы добавить doManyCoolThings в текущий Помощник.

#### Хуки

Каждый модуль может обрабатывать события из запущенного теста. Модуль может быть выполнен до начала теста или после его завершения. Это может быть полезно для действий начальной загрузки или завершающих действий.  
Вы так же можете определить специальное поведение для ситуации когда тест провален. Это может быть полезно отладке.  
Например, модуль PhpBrowser сохраняет текущую страницу в директорию log если тест провален.

Все хуки определены в \Codeception\Module и перечислены здесь. Вы вольны переопределить их в своем модуле.

Пожалуйста, обратите внимание методы с префиксом \_ не добавлены в класс "Парня". Это позволяет объявлять публичные методы, но использовать только для внутренних целей.

#### Отладка

Как мы уже упоминали, хук \_failed может помочь в отладке проваленного теста. У вас есть возможность сохранить текущее состояние тестов и показать их пользователю.

Вы не ограничены этим. Каждый модуль может выводить внутренние значения, которые могут быть полезны при отладке.  
Например, модуль PhpBrowser выводит http код ответа и текущий адрес каждый раз когда он переходит на новую страницу.  
Таким образом, модули не являются "черными ящиками". Они пытаются показать вам, что происходит в время теста. Это делает отладку ваших тестов менее болезненной.

Чтобы вывести дополнительную информацию, используйте debug и debugSection методы модуля.  
Вот пример того, как это работает для PhpBrowser:

<?php

$this->debug('Request ('.$method.'): '.$uri.' '. json\_encode($params));

$browser->request($method, $uri, $params);

$this->debug('Response code: '.$this->session->getStatusCode());

?>

Этот тест запущенный с помощью модуля PhpBrowser в режиме отладке, выведет что-то похожее на это:

I click "All pages"

\* Request (GET) http://localhost/pages {}

\* Response code: 200

#### Настройка

Модули могут быть настроены в файле настроек тестового набора или глобально в codeception.yml.  
Обязательные параметры должны быть определены в $requiredFields свойстве класса модуля. Вот как это сделано в модуле Db.

<?php

class Db extends \Codeception\Module {

protected $requiredFields = array('dsn', 'user', 'password');

?>

Следующий раз вы когда запустите набор тестов без установки эти значений будет выброшено исключение.

Для необязательных параметров, вы должны установить значения по умолчанию. Свойство $config используется чтобы определить необязательные параметры, а так же их значения. В модуле Selenium мы используем адрес и порт сервера по умолчанию.

<?php

class Selenium extends \Codeception\Util\MinkJS

{

protected $requiredFields = array('browser', 'url');

protected $config = array('host' => '127.0.0.1', 'port' => '4444');

}

?>

Параметры адрес и порт могут быть переопределены в настройках набора тестов. Значения устанавливаются в секции modules:config файла настроек.

modules:

enabled:

- Selenium

- Db

config:

Selenium:

url: 'http://mysite.com/'

browser: 'firefox'

Db:

cleanup: false

repopulate: false

Обязательные и необязательные параметры могут доступны через свойство $config. Используйте $this->config['parameter'] чтобы получить значение параметра.

#### Динамическая настройка

начиная с версии 1.6.2

Если вы хотите переопределить настройки модуля во время исполнения, вы можете использовать \_reconfigure метод модуля.  
Вы можете вызвать его из класса помощника и передать в него все поля, которые хотите изменить.

<?php

$this->getModule('Selenium2')->\_reconfigure(array('browser' => 'chrome'));

?>

В конце теста все ваши изменения откатятся до значений из файла настроек.