Mink Documentation

Versão 1.6

Konstantin Kudryashov (everzet)

15 February, 2017

Conteúdo

Esta é uma tradução não oficial feita e mantida por Diego Santos, portanto pode estar desatualizada. Sinta-se encorajado para me ajudar a mantê-la atualizada.

Uma das partes mais importantes da web é o navegador. Navegador é a janela onde através dela usuários web interagem com aplicações web e outros usuários. Usuários estão sempre falando com aplicações web através dos navegadores.

Então, a fim de testar se aquela nossa aplicação web se comporta corretamente, nós precisamos de uma forma para simular esta interação entre o navegador e a aplicação web em nossos testes. Nós precisamos do **Mink**.

O Mink é um controlador/emulador open source de um navegador para aplicações web, escrito em PHP 5.3.

Leia Mink de Relance para aprender mais sobre o Mink e porque você precisa dele.

CAPÍTULO 1

Instalação

Mink é uma biblioteca php 5.3 que você vai usar dentro de suas suites de teste ou projeto. Antes de você começar, garanta que você tem o PHP 5.3.1 ou superior instalado.

A forma recomendada para instalar o Mink com todas as suas dependências é através do Composer:

```
$ composer require behat/mink
```

Nota: Para instalações locais do composer você precisa dizer a ele algo como isto: `$ php composer.phar require behat/mink .` Neste caso você precisa utilizar uma chamada diferente `php composer.phar` em todos os lugares ao invés de simplesmente o comando `composer`.

Tudo será instalado dentro da pasta `vendor`.

Finalmente, inclua o script auto carregável Composer ao seu projeto:

```
require_once 'vendor/autoload.php';
```

Nota: Por padrão, o Mink será instalado sem drivers. A fim de ser capaz de utilizar divers adicionais, você deve instalá-los (através do composer). Eixija as dependencias apropriadas:

- GoutteDriver - `behat/mink-goutte-driver`
- Selenium2Driver - `behat/mink-selenium2-driver`
- BrowserKitDriver - `behat/mink-browserkit-driver`
- ZombieDriver - `behat/mink-zombie-driver`
- SeleniumDriver - `behat/mink-selenium-driver`
- SahiDriver - `behat/mink-sahi-driver`
- WUnitDriver - `behat/mink-wunit-driver`

Se você é novato ou simplesmente não sabe qual escolher, você deverá provavelmente iniciar com o GoutteDriver e o Selenium2Driver (você poderá substituí-lo depois):

CAPÍTULO 2

Guias

Aprenda Mink com as guias de tópicos:

2.1 Mink de Relance

Há um enorme número de emuladores de navegadores lá fora, como Goutte, Selenium, Sahi e outros. Todos eles fazem o mesmo trabalho, mas o fazem de diferentes maneiras. Eles se comportam diferentemente e tem muitas API's diferentes. Mas, o que é mais importante, existem na verdade 2 tipos completamente diferentes de emuladores de navegadores:

- Emuladores de navegadores Headless (sem interface gráfica)
- Controladores de navegador

O primeiro tipo de emuladores de navegadores são implementações pura e simples de especificação HTTP, como Goutte. Esses emuladores de navegadores enviam uma requisição HTTP real contra uma aplicação e analisam o conteúdo da resposta. Eles são muito simples de executar e configurar, porque este tipo de emulador pode ser escrito em qualquer linguagem de programação disponível e pode ser executado através do console em servidores sem GUI. Emuladores Headless tem vantagens e desvantagens. As vantagens são simplicidade, velocidade e habilidade para o executar sem a necessidade de um navegador real. Mas este tipo de navegador tem uma grande desvantagem, eles não suportam JS/AJAX. Então, você não pode testar suas ricas aplicações web GUI com navegadores headless.

O segundo tipo de controladores são os emuladores de navegadores. Estes emuladores visam controlar o emulador real. É isto mesmo, um programa para controlar outro programa. Controladores de Navegadores simulam as interações dos usuários no navegador e são capazes de recuperar a informação atual da atual página de navegador. Selenium e Sahi são os dois controladores de navegadores mais famosos. A principal vantagem do uso de controladores de navegadores é o suporte a interações JS/AJAX na página. A desvantagem é que tais emuladores de navegadores exigem a instalação do navegador, configurações extra e geralmente são mais lentos do que headless.

Então, a resposta fácil é escolher o melhor emulador para o seu projeto e usar a sua API para testes. Mas como nós já vimos, ambos tipos de emuladores de navegadores tem vantagens e desvantagens. Se você escolher o headless, você não será capaz de testar suas páginas JS/AJAX. E se você escolher o controlador de navegador, sua suite de testes vai se tornar muito lenta em algum ponto. Então, no mundo real nós devemos usar ambos! E é por isso que voc precisa do **Mink**.

O **Mink** remove as diferentes APIs entre diferentes emuladores de navegador fornecendo diferentes drivers (leia no capítulo Drivers) para todos os emuladores de navegador e provê uma forma fácil de controlar o navegador (Controlando o Navegador), analisar páginas (Analisando Páginas), manipular elementos da página (Manipulando Páginas) ou interagir com elas (Interagindo com Páginas).

2.2 Controlando o Navegador

No Mink, o ponto de entrada do navegador é chamado de sessão. Pense nisso como sendo a janela do navegador (alguns drivers até deixam você alternar guias!).

Primeiro, inicie sua sessão (é como abrir a guia do seu navegador). Nada pode ser feito com isto antes de iniciá-lo.

```
// Escolha um driver Mink. Veja mais sobre isto nos capítulos seguintes.
$driver = new \Behat\Mink\Driver\GoutteDriver();

$session = new \Behat\Mink\Session($driver);

// inicia a sessão
$session->start();
```

Nota: O primeiro argumento do construtor da sessão é um objeto driver. Drivers são a forma de abstração que as camadas Mink trabalham. Você irá descobrir mais sobre os drivers disponíveis em um capitulo posterior.

Cuidado: Apesar do Mink fazer o seu melhor para remover as diferenças entre os diferentes drivers, cada driver tem funcionalidades e deficiências diferentes. Veja *Suporte a recursos do driver* para ver cada funcionalidade são suportadas sobre cada driver.

2.2.1 Interação Básica no Navegador

Agora que sua sessão está iniciada, você irá querer abrir uma página com ele. Apenas depois de começar, a sessão não está em uma página qualquer (em um navegador real, você estaria na página `about:blank`), e chamando qualquer outra ação é provável que falhe.

```
$session->visit('http://meu_projeto.dev/alguma_pagina.php');
```

Nota: Mink é principalmente projetado para ser usado para testar websites. Para que você possa navegar e testar páginas de erro, o método `Session:visit` não considera códigos de status de erro como inválida. Ele *não* irá lançar uma exceção neste caso. Você irá precisar checar o código do status (ou certo texto na página) para saber se a resposta teve sucesso ou não.

2.2.2 Interagindo com a Página

A sessão dá acesso a página através do método `Session::getPage`. Isto lhe permite analisar a página e interagir com elas. Os próximos capítulos cobrem a página API com profundidade. A maioria do que você vai fazer com Mink usará este objeto, mas você pode continuar lendo para saber mais sobre Sessão.

2.2.3 Usando o Histórico do Navegador

A sessão lhe da acesso ao histórico do navegador:

```
// obter a URL da página atual:
echo $session->getCurrentUrl();

// usar controles de histórico:
```

```
$session->reload();
$session->back();
$session->forward();
```

2.2.4 Gerenciamento de Cookie

A sessão pode manipular cookies disponíveis no navegador.

```
// definir cookie:
$session->setCookie('cookie nome', 'valor');

// obter cookie:
echo $session->getCookie('cookie nome');

// deletar cookie:
$session->setCookie('cookie nome', null);
```

Nota: Com drivers que usam o JavaScript para controlar o navegador - como Sahi - que pode ser restringido a acessar/definir tudo, mas Cookies HttpOnly.

2.2.5 Código de Status de Recuperação

A sessão lhe permite recuperar o código HTTP do status da resposta:

```
echo $session->getStatusCode();
```

2.2.6 Gerenciamento de Headers

A sessão lhe permite manipular request de headers e acessar a resposta dos headers:

```
// definindo linguagem do navegador:
$session->setRequestHeader('Accept-Language', 'fr');

// recuperação da resposta do headers:
print_r($session->getResponseHeaders());
```

Nota: A manipulação de headers somente é suportada em drivers headless (como o Goutte, por exemplo). Controladores de navegadores (como o Selenium2, por exemplo) não podem acessar aquela informação.

2.2.7 Autenticação HTTP

A sessão tem um método especial para atuar na autenticação básica de HTTP:

```
$session->setBasicAuth($user, $password);
```

O método pode também ser usado para reiniciar uma autenticação prévia:

```
$session->setBasicAuth(false);
```

Nota: Autenticação HTTP automática somente é suportada em drivers headless. Porque autenticação HTTP nos navegadores exigem atuação manual do usuário, que não pode ser feita remotamente pelos controladores de navegador.

2.2.8 Avaliação do Javascript

A sessão lhe permite executar ou avaliar Javascript.

```
// Executar JS
$session->executeScript('document.body.firstChild.innerHTML = "";');

// analisar expressão JS:
echo $session->evaluateScript(
    "return 'algo a partir do navegador';"
);
```

Nota: A diferença entre estes métodos é que `Session::evaluateScript` retorna o resultado desta expressão. Quando você não precisa obter o valor do retorno, usar `Session::executeScript` é melhor.

Você pode também esperar até uma determinada expressão JS retornar um valor booleano verdadeiro ou o tempo limite ser atingido:

```
// esperar por N milissegundos ou
// até a expressão JS tornar-se verdadeira:
$session->wait(
    5000,
    "$('.suggestions-results').children().length"
);
```

Nota: O método `Session::wait` retorna `verdadeiro` quando a avaliação retorna verdadeiro. Ele irá retornar `falso` quando o timeout é alcançado.

2.2.9 Repondo a Sessão

O objetivo primário para o Mink é prover um navegação na WEB API única consistente para teste de aceite. Mas uma parte muito importante no teste é o isolamento.

O Mink provê dois métodos muito úteis para isolar testes, que podem ser usados em seus métodos de teste `destruir`:

```
// soft-reset:
$session->reset();

// hard-reset:
$session->stop();
// ou se você quiser iniciar novamente ao mesmo tempo
$session->restart();
```

Parar a sessão é a melhor maneira de reiniciar a sessão ao estado inicial. Ele irá fechar o seu navegador inteiramente. Para usar a sessão novamente, você precisa iniciar a sessão antes de qualquer outra ação. O atalho `Session::restart` permite você fazer estas 2 chamadas em uma única.

A desvantagem de fechar o navegador e iniciá-lo novamente é que leva tempo. Em muitos casos, um nível mais baixo de isolamento é o suficiente a favor de uma reinicialização rápida. O método `Session::reset` cobre este caso de uso. Ele irá tentar limpar os cookies e reiniciar o request de headers e o histórico do navegador ao limite das possibilidades do driver.

Levando tudo isto em conta, por padrão é recomendado usar `Session::reset()` e chamar `Session::stop()` quando você realmente precisar de isolamento completo.

2.3 Analisando Páginas

A maioria dos usos do Mink envolverá trabalhar com a página aberta em seu navegador. Isto é feito graças ao poderoso Elemento API. Esta API permite a análise da página (similar ao DOM em Javascript), manipular elementos da página e para interegir com eles, o qual será coberto nos próximos capítulos.

2.3.1 DocumentElement e NodeElement

O Elemento API consiste em 2 classes principais. A instância `DocumentElement` representa a página a ser exibida no navegador, enquanto a classe `NodeElement` é usada para representar qualquer elemento dentro da página. Ambas classes compartilham um conjunto comum de métodos para analisar a página (definidos em `TraversableElement`).

A instância `DocumentElement` é acessível através do método `Session::getPage`:

```
$page = $session->getPage();

// Agora você pode manipular a página.
```

Nota: A instância `DocumentElement` representa o nó `<html>` no DOM. É equivalente ao `document.documentElement` no DOM API do Javascript.

2.3.2 Métodos de Análise

Elementos tem 2 métodos principais: `ElementInterface::findAll` retorna um array de instâncias `NodeElement` correspondentes ao seletor fornecido dentro do elemento atual enquanto `ElementInterface::find` retorna a primeira combinação ou `null` quando não há ninguém.

A classe `TraversableElement` também provê um grupo de métodos atalho no topo do `find()` para torná-lo mais fácil de conseguir muitos casos de uso em comum:

`ElementInterface::has` Checa se um elemento filho combina com um dado seletor, mas sem devolvê-lo.

`TraversableElement::findById` Procura um elemento filho com um dado id.

`TraversableElement::findLink` Procura um link com um dado texto, title, id ou atributo `alt` (para imagens usadas dentro de links).

`TraversableElement::findButton` Procura por um botão com um dado texto, title, id, atributo `name` ou um atributo `alt` (para imagens usadas dentro de links).

`TraversableElement::findField` Procura por um campo (`input`, `textarea` ou `select`) com um dado label, placeholder, id ou atributo `name`.

Nota: Estes atalhos estão retornando um único elemento. Se você precisa encontrar todos correspondentes, você irá precisar usar `findAll` com o seletor named.

Analisando Aninhado

Cada método `find*()` retornará uma instância `Behat\Mink\Element\NodeElement` e `findAll()` irá retornar um array das tais instâncias. A parte divertida é que você pode fazer a mesma velha análise nos tais elementos também:

```
$formularioRegistrar = $page->find('css', 'form.registrar');

if (null === $formularioRegistrar) {
    throw new \Exception('O elemento não foi encontrado');
}

// busca algum campo DENTRO do formulário com a class="registrar"
$campo = $formularioRegistrar->findField('Email');
```

2.3.3 Seletores

Os métodos `ElementInterface::find` e `ElementInterface::findAll` suporta vários tipos de seletores para encontrar elementos.

Seletor CSS

O tipo de seletor `css` permite você usar expressões CSS para buscar por elementos na página:

```
$titulo = $page->find('css', 'h1');

$iconeBotao = $page->find('css', '.btn > .icon');
```

Seletor XPath

O seletor tipo `xpath` permite você usar queries XPath para buscar por elementos na página:

```
$ancorasSemUrl = $page->findAll('xpath', '//a[not(@href)]');
```

Cuidado: Este seletor busca por um elemento dentro do nó atual (que é `<html>` para o objeto da página). Isto significa que tentando passar o XPath de um elemento recuperado com `ElementInterface::getXpath` não irá funcionar (esta query inclui a query para o nó raiz). Para checar se um objeto elemento ainda existe na página do navegador, use `ElementInterface::isValid` como alternativa.

Seletores Named

Seletores named fornecem um conjunto de queries para necessidades comuns. Para condições baseadas no conteúdo dos elementos, o seletor named irá tentar encontrar a primeira correspondência exata. No caso de não obter uma correspondência exata será então um retorno correspondente parcial.

Para o tipo seletor named, o segundo argumento para o método `find()` é um array com 2 elementos: o nome da query que irá usar e o valor da busca nesta query:

```
$valorEscapado = $session->getSelectorsHandler()->xpathLiteral('Go to top');

$topLink = $page->find('named', array('link', $valorEscapado));
```

Cuidado: O seletor named requer um valor escapando como XPath literal. De outra forma a query XPath gerada será inválida.

As seguintes queries são suportadas pelo seletor named:

`id` Busca um elemento pelo seu id.

`id_or_name` Busca um elemento pelo seu id ou name.

`link` Busca um link pelo seu id, title, img alt, rel ou text.

`button` Busca um botão pelo seu name, id, text, img alt ou title.

`link_or_button` Busca links e botões.

`content` Busca um conteúdo especifico da página (texto).

`field` Busca um campo no formulário pelo seu id, name, label ou placeholder.

`select` Busca um campo select pelo seu id, name ou label.

`checkbox` Busca um checkbox pelo seu id, name, ou label.

`radio` Busca um radio button pelo seu id, name, ou label.

`file` Busca um file input pelo seu id, name, ou label.

`optgroup` Busca um optgroup pelo seu label.

`option` Busca um option pelo seu content ou value.

`fieldset` Busca um fieldset pelo seu id ou legend.

`table` Busca uma table pelo seu id ou caption.

Seletor Customizado

O Mink permite você registrar seus próprios tipos de seletor através da implementação da `Behat\Mink\Selector\SelectorInterface`. Isto deve então ser registrado no `SelectorsHandler` que tem registo dos seletores disponíveis.

A forma recomendada para registrar um seletor customizado é fazê-lo quando construir sua `Session`:

```
$selector = new \App\MySelector();

$handler = new \Behat\Mink\Selector\SelectorsHandler();
$handler->registerSelector('mine', $selector);

$driver = // ...

$session = new \Behat\Mink\Session($driver, $handler);
```

2.4 Manipulando Páginas

Uma vez que você pegue um elemento da página, você vai querer manipulá-lo. Você pode também interagir com a página, que é coberta no próximo capitulo.

2.4.1 Obtendo o nome da tag

O método `NodeElement::getTagName` lhe permite obter o nome da tag do elemento. Este nome é sempre retornado em letras minúsculas.

```
$el = $page->find('css', '.something');

// obter o nome da tag:
echo $el->getTagName(); // exibe 'a'
```

2.4.2 Acessando atributos HTML

A classe `NodeElement` lhe dá acesso aos atributos HTML do elemento.

`NodeElement::hasAttribute` Verifica se o elemento tem um determinado atributo.

`NodeElement::getAttribute` Obtém o valor de um atributo.

`NodeElement::hasClass` Verifica se o elemento tem uma dada classe (convenientemente envolvido em torno de `getAttribute('class')`).

```
$el = $page->find('css', '.something');

if ($el->hasAttribute('href')) {
    echo $el->getAttribute('href');
} else {
    echo 'Esta âncora não é um link. Ela não tem um href.';
}
```

2.4.3 Elemento de Conteúdo e Texto

A classe `Element` provê acesso ao conteúdo dos elementos.

`Element::getHtml` Obtém o HTML interno do elemento, por exemplo todos os filhos do elemento.

`Element::getOuterHtml` Obtém o HTML exterior do elemento, por exemplo incluindo o próprio elemento.

`Element::getText` Obtém o texto do elemento.

Nota: `getText()` irá retirar tags e caracteres não impressos fora da resposta, incluindo linhas novas. Então ele basicamente retorna o texto, que o usuário vê na página.

2.4.4 Verificando a visibilidade de um Elemento

O método `NodeElement::isVisible` permite checar se o elemento está visível.

2.4.5 Acessando o estado do formulário

A classe `NodeElement` permite o acesso ao estado de elementos do formulário:

`NodeElement::getValue` Obtém o valor do elemento. Veja *Interagindo com formulários*.

`NodeElement::isChecked` Verifica se o checkbox ou o radio button está checado.

`NodeElement::isSelected` Verifica se o elemento `<option>` está selecionado.

Métodos de atalho

A classe `TraversableElement` provê uns poucos métodos de atalho permitindo a busca de um elemento filho na página e checar o estado dele imediatamente:

`TraversableElement::hasCheckedField` Olha para uma checkbox (veja findField) e checa se ela está checada.

`TraversableElement::hasUncheckedField` Olha para um checkbox (veja findField) e checa se ele não está checado.

2.5 Interagindo com Páginas

A maioria dos usos de Mink envolverá trabalhar com páginas abertas em seu navegador. O elemento Mink API permite você interagir com elementos da página.

2.5.1 Interagindo com Links e Botões

Os métodos `NodeElement::click` e `NodeElement::press` lhe permitem clicar nos links e apertar os botões na página.

Nota: Estes métodos atualmente são equivalentes internamente (pressionar um botão envolve clicá-lo). Tendo ambos os métodos permite manter o código mais legível.

2.5.2 Interagindo com formulários

A classe `NodeElement` tem um conjunto de métodos permitindo interagir com formulários:

`NodeElement::getValue` pega o valor de um campo do formulário. O valor depende do tipo do campo:

- o valor da opção selecionada para caixas de seleção individuais (ou `null` quando nenhuma é selecionada) ;
- um array dos valores das opções selecionadas para multiplas caixas de seleção;
- o valor do campo checkbox quando checado, ou `null` quando não checado;
- o valor do radio button selecionado no grupo de radio para radio buttons;
- o valor do campo para campos de texto e textareas;
- um valor indefinido para campos de arquivos (devido as limitações do navegador).

`NodeElement::setValue` coloca o valor em um campo do formulário

- para um campo de arquivo, deve ser o caminho para o arquivo;
- para um checkbox, deve ser um booleano indicando se está checado;
- para outros campos, deve corresponder o comportamento de `getValue`.

`NodeElement::isChecked` informa se um radio button ou um checkbox está checado.

`NodeElement::isSelected` informa se um elemento `<option>` está selecionado.

`NodeElement::check` checa um campo checkbox.

`NodeElement::uncheck` descheca um campo checkbox.

`NodeElement::selectOption` seleciona uma opção em uma caixa de seleção ou em um radio group.

`NodeElement::attachFile` anexa um arquivo em uma em uma entrada de arquivo.

`NodeElement::submit` submete o formulário.

2.5.3 Interagindo com o mouse

A classe `NodeElement` oferece um conjunto de métodos que permitem interagir com o mouse:

`NodeElement::click` executa um clique no elemento.

`NodeElement::doubleClick` executa um clique duplo no elemento.

`NodeElement::rightClick` executa um clique com o botão direito no elemento.

`NodeElement::mouseOver` move o mouse para cima do elemento.

2.5.4 Interagindo com o teclado

O mink permite você interagir com o teclado graças aos métodos `NodeElement::keyDown`, `NodeElement::keyPress` e `NodeElement::keyUp`.

2.5.5 Manipulando o Foco

A classe `NodeElement` permite você dar e remover o foco em um elemento graças aos métodos `NodeElement::focus` e `NodeElement::blur`.

2.5.6 Clica e arrasta

Mink provê o clica e arrasta de um elemento para outro:

```
$arrastado = $page->find(...);
$alvo = $page->find(...);

$arrastado->dragTo($alvo);
```

2.5.7 Métodos de atalho

A classe `TraversableElement` provê alguns poucos métodos de atalho permitindo procurar um elemento filho na página e executar uma ação imediatamente:

`TraversableElement::clickLink` Olha para um link (veja findLink) e clica nele.

`TraversableElement::pressButton` Olha para um botão (veja findButton) e o pressiona.

`TraversableElement::fillField` Olha para um campo (veja findField) e coloca um valor nele.

`TraversableElement::checkField` Olha para um checkbox (veja findField) e checa ele.

`TraversableElement::uncheckField` Olha para um checkbox (veja findField) e descheca ele.

`TraversableElement::selectFieldOption` Olha para um select ou radio group (veja findField) e seleciona uma opção.

`TraversableElement::attachFileToField` Olha para um campo de arquivo (veja findField) e anexa um arquivo a ele.

Nota: Todos estes métodos de atalho lançam uma `ElementNotFoundException` no caso do elemento filho não ser encontrado.

2.6 Drivers

Como é que o Mink fornece uma API consistente para vários tipos de bibliotecas de navegadores diferentes, frequentemente escritas em diferentes linguagens? Através dos drivers! Um driver Mink é uma classe simples, que implementa `Behat\Mink\Driver\DriverInterface`. Esta interface descreve metodos pontes entre o Mink e os emuladores de navegadores reais. Ele não sabe nada sobre como inicar/parar ou analisar uma página nesse emulador de navegador em particular. Ele somente sabe que método do driver que deve chamar para fazer isso.

O Mink vem com seis drivers fora da caixa:

2.6.1 GoutteDriver

GoutteDriver fornece uma ponte para o Goutte headless browser. Goutte é um clássico php-puro headless browser, escrito pelo criador do framework Symfony Fabien Potencier.

Nota: O GoutteDriver estende-se do BrowserKitDriver para consertar um caso extremamente pequeno na implementação Goutte do Browserkit. Ele também é capaz de instanciar o cliente Goutte automaticamente.

Instalação

GoutteDriver é uma biblioteca PHP pura disponibilizada através do Composer:

```
$ composer require behat/mink-goutte-driver
```

Nota: GoutteDriver é compatível em ambos Goutte 1.x o qual confia no Guzzle 3 e Goutte 2.x o qual confia no Guzzle 4+ para a implementação HTTP adjacente.

O Composer provavelmente irá selecionar o Goutte 2.x por padrão.

Uso

Afim de falar com o Goutte, você deverá instanciar um `Behat\Mink\Driver\GoutteDriver`:

```
$driver = new \Behat\Mink\Driver\GoutteDriver();
```

Também, se você quiser configurar o Goutte com maior precisão, você poderia fazer a configuração total na mão:

```
$client = new \Goutte\Client();
// Faça mais configurações para o cliente Goutte

$driver = new \Behat\Mink\Driver\GoutteDriver($client);
```

2.6.2 BrowserKitDriver

BrowserKitDriver provê uma ponte para o componente Symfony BrowserKit. BrowserKit é um emulador de navegador fornecido pelo Symfony project.

Instalação

BrowserKitDriver é uma biblioteca PHP pura disponível através do Composer:

```
$ composer require behat/mink-browserkit-driver
```

Nota: O componente BrowserKit somente provê uma implementação abstrata. As implementações atuais são fornecidas por outros projetos, como Goutte ou o componente Symfony HttpKernel.

Se você está usando Goutte, você deve usar o especial GoutteDriver que garante completa compatibilidade com o Goutte devido um caso extremo no Goutte.

Uso

Afim de falar conversar com o BrowserKit, você deve instanciar um `Behat\Mink\Driver\BrowserKitDriver`:

```
$browserkitClient = // ...

$driver = new \Behat\Mink\Driver\BrowserKitDriver($browserkitClient);
```

2.6.3 Selenium2Driver

O Selenium2Driver fornece uma ponte para a ferramenta Selenium2 (webdriver). Se você ama o Selenium2, agora você pode usá-lo corretamente fora da caixa também.

Instalação

O Selenium2Driver está disponível através do Composer:

```
$ composer require behat/mink-selenium2-driver
```

Afim de falar com o servidor do selenium, você deve instalar e confirá-lo primeiramente:

1. Faça o Download do Servidor do Selenium do site do projeto.
2. Execute o servidor com o seguinte comando (altere a versão para o número que você baixou):

```
$ java -jar selenium-server-standalone-2.44.0.jar
```

Dica: O Selenium2Driver atualmente confia no protocolo WebDriver definido pelo Selenium2. Isto significa que é possível usá-lo em outras implementações do protocolo. Nove, porém, que outras implementações podem ter alguns bugs.

A suite de testes do driver é executada contra a implementação Phantom.js mas ainda desencadeia algumas falhas devido os bugs na sua implementação.

Uso

É isso aí, agora você pode usar o Selenium2Driver:

```
$driver = new \Behat\Mink\Driver\Selenium2Driver('firefox');
```

2.6.4 ZombieDriver

O ZombieDriver fornece uma ponte para o emulador de navegador Zombie.js. Zombie.js é um emulador de browser headless, escrito em node.js. Ele suporta todas as interações JS Selenium e Sahi funciona quase tão rápido quanto o Goutte faz. Atualmente ele é o melhor dos dois mundos, mas ainda limitado a apenas um tipo de navegador (Webkit). Também ainda é mais devagar que o Goutte e requerem que o node.js e npm estejam instalados no sistema.

Instalação

ZombieDriver está disponível através do Composer:

```
$ composer require behat/mink-zombie-driver
```

Afim de falar com um servidor zombie.js, você precisa instalar e configurar o zombie.js primeiramente:

1. Instale o node.js com as seguintes instruções do site oficial: http://nodejs.org/.
2. Instale o npm (node package manager) pela seguinte instruções do http://npmjs.org/.
3. Instale o zombie.js com npm:

```
$ npm install -g zombie
```

Depois de instalar npm e zombie.js, você precisará adicionar as bibliotecas do npm em seu `NODE_PATH`. A forma mais fácil de fazer isto é adicionar:

```
export NODE_PATH="/PATH/TO/NPM/node_modules"
```

em seu `.bashrc`.

Uso

Depois disto, você estará capaz de usar somente o ZombieDriver sem a instalação manual do servidor. O driver irá fazer tudo isto para você automaticamente:

```
$driver = new \Behat\Mink\Driver\ZombieDriver(
    new \Behat\Mink\Driver\NodeJS\Server\ZombieServer()
);
```

Se você quer mais controle durante a instalação do driver, por exemplo se você quer configurar o driver para ser inicializado no servidor em uma porta específica, use a versão mais verbosa:

```
$driver = new \Behat\Mink\Driver\ZombieDriver(
    new \Behat\Mink\Driver\Zombie\Server($host, $port, $nodeBin, $script)
);
```

Nota: `$host` simplesmente define o host em que o zombie.js será inicializado. Por padrão é `127.0.0.1`.

`$port` define uma porta zombie.js. A padrão é `8124`.

`$nodeBin` define o caminho completo do binário do node.js. Por padrão é somente `node`.

`$script` define um script node.js para inicializar um servidor zombie.js. Se você passar um `null` o script padrão será usado. Use está opção cuidadosamente!

2.6.5 SahiDriver

O SahiDriver fornece uma ponte para o controlador de navegador Sahi. Sahi é um novo controlador de navegador JS, que substituiu rapidamente a velha suite de testes Selenium. É tão mais fácil configurar e usar do que o Selenium classico. Ele tem um instalador GUI para cada sistema operacional lá fora e é capaz de controlar cada sistema navegador através de um servidor de proxy especial empacotado.

Instalação

SahiDriver está disponível através do Composer:

```
$ composer require behat/mink-sahi-driver
```

Afim de falar com um navegador real através do Sahi, você deve instalar e configurar o Sahi primeiramente:

1. Faça o download e execute o jar do Sahi do site do projeto Sahi e o execute. Ele irá executar o instalador, que irá lhe guiar através do processo de instalação.
2. Execute o proxy Sahi antes das suas suites de teste (você pode iniciar este proxy durante a inicialização do sistema):

```
cd $YOUR_PATH_TO_SAHI/bin
./sahi.sh
```

Uso

Depois de instalar o Sahi e executar o servidor de proxy Sahi, você estará hábil a o controlar com `Behat\Mink\Driver\SahiDriver`:

```
$driver = new \Behat\Mink\Driver\SahiDriver('firefox');
```

Nota: Aviso, que o primeiro argumento do `SahiDriver` sempre será o nome do navegador, suportado pelo Sahi.

Se você quiser maior controle durante a inicialização do driver, por exemplo, se você quiser configurar o driver para falar com um proxy em outra máquina, utilize a versão mais verbosa com um segundo argumento cliente:

```
$driver = new \Behat\Mink\Driver\SahiDriver(
    'firefox',
    new \Behat\SahiClient\Client(
        new \Behat\SahiClient\Connection($sid, $host, $port)
    )
);
```

Nota: `$sid` é uma sessão ID do Sahi. Ele é uma string única, usada pelo driver e o proxy Sahi afim de possibilitar a conversa com cada outro. Vôcê deve preencher isto com `null` se você quiser que o Sahi inicie seu navegador automaticamente ou com alguma string única se você quiser controlar um navegador já inicializado.

`$host` simplesmente define o host no qual o Sahi é iniciado. Por padrão é o `localhost`.

`$port` define a porta do proxy Sahi. O padrão inicial é `9999`.

2.6.6 SeleniumDriver

SeleniumDriver fornece uma ponte para a famosa ferramenta Selenium. Se você precisar do legado Selenium, você pode usá-lo direito fora da caixa nas suas suites de test Mink.

Cuidado: O protocolo SeleniumRC usado por este driver está obsoleto e não será suportado por todas as funcionalidades do Mink. Por esta razão, o SeleniumDriver está obsoleto a favor do Selenium2Driver, que é baseado no novo protocolo e é mais poderoso.

Instalação

SeleniumDriver está disponível através do Composer:

```
$ composer require behat/mink-selenium-driver
```

Afim de conversar com o servidor selenium, você deve instalar e configurá-lo primeiro:

1. Baixe o Servidor Selenium do website do projeto.
2. Execute o servidor com o seguinte comando (atualize a versão para o número que você baixou):

```
    $ java -jar selenium-server-standalone-2.44.0.jar
```

Uso

É isso aí, agora você pode usar o SeleniumDriver:

```
$client = new \Selenium\Client($host, $port);
$driver = new \Behat\Mink\Driver\SeleniumDriver(
    'firefox', 'base_url', $client
);
```

2.6.7 Suporte a recursos do driver

Apesar do Mink fazer o seu melhor em remover as diferenças entre navegador e diferentes emuladores de navegador, ele não pode fazer muito em alguns casos. Por exemplo, BrowserKitDriver não avalia JavaScript e Selenium2Driver não consegue pegar código dos status das respostas. Nestes casos, o driver sempre irá lançar uma exceção `Behat\Mink\Exception\UnsupportedDriverActionException`.

Funcionalidade	BrowserKit/Goutte	Selenium2	Zombie	Selenium	Sahi
Atravessar páginas	Sim	Sim	Sim	Sim	Sim
Manipular Formulários	Sim	Sim	Sim	Sim	Sim
Autenticação Básica HTTP	Sim	Não	Sim	Não	Não
Gestão de janelas	Não	Sim	Não	Sim	Sim
Gestão de iFrames	Não	Sim	Não	Sim	Não
Acessar cabeçalhos de solicitação	Sim	Não	Sim	Não	Não
Cabeçalhos de resposta	Sim	Não	Sim	Não	Não
Manipular Cookies	Sim	Sim	Sim	Sim	Sim
Acessar códgigo de status	Sim	Não	Sim	Não	Não
Manipular mouse	Não	Sim	Sim	Sim	Sim
Arrastar e largar	Não	Sim	Não	Sim	Sim
Ações de teclado	Não	Sim	Sim	Sim	Sim
Visualizar elementos	Não	Sim	Não	Sim	Sim
Avaliar JS	Não	Sim	Sim	Sim	Sim
Redimensionar janela	Não	Sim	Não	Não	Não
Maximizar janela	Não	Sim	Não	Sim	Não

2.7 Gerenciando Sessões

Apesar do objeto de sessão já estar utilizável o suficiente, não é tão fácil escrever um código multissessão (multidriver/multinavegador). Sim, você ouviu direito, com o Mink você pode manipular múltiplos emuladores de navegador simultaneamente com uma única API consistente:

```
// inicialização de sessões
$sessao1 = new \Behat\Mink\Session($driver1);
$sessao2 = new \Behat\Mink\Session($driver2);

// começa sessões
$sessao1->start();
$sessao2->start();

$sessao1->visit('http://my_project.dev/chat.php');
$sessao2->visit('http://my_project.dev/chat.php');
```

Cuidado: O estado de uma sessão é gerenciado atualmente pelo driver. Isto significa que cada sessão precisa usar uma instância diferente de driver.

Não é legal? Mas o Mink torna isto mais frio:

```
$mink = new \Behat\Mink\Mink();
$mink->registerSession('goutte', $goutteSession);
$mink->registerSession('sahi', $sahiSession);
$mink->setDefaultSessionName('goutte');
```

Com tal configuração, você pode falar com suas sessões pelo nome através de um único objeto contêiner:

```
$mink->getSession('goutte')->visit('http://my_project.dev/chat.php');
$mink->getSession('sahi')->visit('http://my_project.dev/chat.php');
```

Nota: O Mink sempre irá iniciar suas sessões ociosas quando necessário (na primeira chamada `getSession()`). Então, o navegador não será inicial até que você realmente precise dele!

Ou você até pode omitir o nome da sessão em casos padrão:

```
$mink->getSession()->visit('http://my_project.dev/chat.php');
```

Esta chamada é possível graças ao `$mink->setDefaultSessionName('goutte')` configurado previamente. Nós criamos a sessão padrão, que seria devolvida na chamada sem argumentos `getSession()`.

Dica: A classe `Behat\Mink\Mink` também provê uma forma fácil de redefinir ou reiniciar suas sessões iniciadas (e somente iniciadas):

```
// redefine sessões iniciadas
$mink->resetSessions();

// reinicia sessões iniciadas
$mink->restartSessions();
```

CAPÍTULO 3

Ferramentas de Teste de Integração

O mink tem integrações com muitas ferramentas de teste:

- Behat através da Behat MinkExtension
- PHPUnit através do phpunit-mink package