PhantomJS integration module for NodeJS http://amirraminfar.com/phantomjs-node/

phantomjs nodejs phantomjs-process javascript

Amir Raminfar Upgrades jest — Latest commit 1e0e9d1 15 days ago

.github	Updates template to have env info	a year ago
decls	Adds flow check style to everything (#557)	5 months ago
examples	Adds more examples for render	a month ago
src	Updates libs	2 months ago
.babelrc	Uses babel-preset-env instead and target node 6.9	3 months ago
.editorconfig	Cleans package json, uses yarn cache. (#582)	4 months ago
.eslintignore	Adds flow check style to everything (#557)	5 months ago
.eslintrc.json	Make eslint strict	4 months ago
.flowconfig	Adds flow check style to everything (#557)	5 months ago
.gitignore	Adds license	a year ago
.npmignore	Adds flow check style to everything (#557)	5 months ago
.travis.yml	Uses babel-preset-env instead and target node 6.9	3 months ago
LICENSE.md	Update and rename LICENSE to LICENSE.md	a year ago
README.md	Updates README	a month ago
gulpfile.babel.js	Cleans package json, uses yarn cache. (#582)	4 months ago
package.json	Upgrades jest	15 days ago
yarn.lock	Upgrades jest	15 days ago

README.md

phantom - Fast NodeJS API for PhantomJS

npm install phantom
3 dependencies version 4.0.0
253 dependents updated 2 months ago
43 ★ 92,994 downloads in the last month
download rank: top 1% of 405,000 packages

npm v4.0.0 | downloads 99k/month | build passing | dependencies up to date | node >=6.9.0

Super easy to use

```
const phantom = require('phantom');

(async function() {
    const instance = await phantom.create();
    const page = await instance.createPage();
    await page.on("onResourceRequested", function(requestData) {
        console.info('Requesting', requestData.url)
    });

    const status = await page.open('https://stackoverflow.com/');
    console.log(status);

    const content = await page.property('content');
    console.log(content);

    await instance.exit();
}());
```

Using Node v7+ you can run the above example with `node --harmony-async-await file.js`

See examples folder for more ways to use this module.

Installation

Node v6.x and later

Latest version of phnatom does **require Node v6.x and later**. You can install with

```
$ npm install phantom --save
```

Node v5.x

To use version 3.x you need to have at least Node v5+. You can install it using

```
$ npm install phantom@3 --save
```

Versions *older* than 5.x, install with

```
$ npm install phantom@2 --save
```

How does it work?

v1.0.x used to leverage `dnode` to communicate between nodejs and phantomjs. This approach raised a lot of security restrictions and did not work well when using `cluster` or `pm2`.

v2.0.x has been completely rewritten to use `sysin` and `sysout` pipes to communicate with the phantomjs process. It works out of the box with `cluster` and `pm2`. If you want to see the messages that are sent try adding `DEBUG=true` to your execution, ie. `DEBUG=true node path/to/test.js`. The new code is much cleaner and simpler. PhantomJS is started with a shim which proxies all messages to the `page` or `phantom` object.

Migrating from 2.x

Going forward, version phantom@3 will only support Node v5 and above. This adds the extra benefit of less code and faster performance.

Migrating from 1.0.x

Version 2.0.x is not backward compatible with previous versions. Most notability, method calls do not take a callback function anymore. Since `node` supports `Promise`, each of the methods return a promise. Instead of writing `page.open(url, function(){})` you would have to write `page.open(url).then(function(){})`.

The API is much more consistent now. All properties can be read with `page.property(key)` and settings can be read with `page.setting(key)`. See below for more example.

`phantom` object API

`phantom#create`

To create a new instance of `phantom` use `phantom.create()` which returns a `Promise` which should resolve with a `phantom` object. If you want add parameters to the phantomjs process you can do so by doing:

```
var phantom = require('phantom');
phantom.create(['--ignore-ssl-errors=yes', '--load-images=no']).then(...)
```

You can also explicitly set :

- The phantomjs path to use
- A logger object
- A log level if no logger was specified

by passing them in config object:

```
var phantom = require('phantom');
phantom.create([], {
    phantomPath: '/path/to/phantomjs',
    logger: yourCustomLogger,
    logLevel: 'debug',
}).then(...)
```

The `logger` parameter should be a `logger` object containing your logging functions. The `logLevel` parameter should be log level like `"warn"` or `"debug"` (It uses the same log levels as `npm`), and will be ignored if `logger` is set. Have a look at the `logger` property below for more information about these two parameters.

`phantom#createPage`

To create a new `page`, you have to call `createPage()`:

```
var sitepage = null;
var phInstance = null;
phantom.create()
    .then(instance => {
        phInstance = instance;
        return instance.createPage();
    })
    .then(page => {
    // use page
    })
    .catch(error => {
        console.log(error);
        phInstance.exit();
    });
```

`phantom#exit`

Sends an exit call to phantomjs process.

Make sure to call it on the phantom instance to kill the phantomjs process. Otherwise, the process will never exit.

`phantom#kill`

Kills the underlying phantomjs process (by sending `SIGKILL` to it).

It may be a good idea to register handlers to `SIGTERM` and `SIGINT` signals with `#kill()`.

However, be aware that phantomjs process will get detached (and thus won't exit) if node process that spawned it receives `SIGKILL`!

`phantom#logger`

The property containing the winston `logger` used by a `phantom` instance. You may change parameters like verbosity or redirect messages to a file with it.

You can also use your own logger by providing it to the `create` method. The `logger` object can contain four functions : `debug`, `info`, `warn` and `error`. If one of them is empty, its output will be discarded.

Here are two ways of handling it :

```
/* Set the log level to 'error' at creation, and use the default logger  */
phantom.create([], { logLevel: 'error' }).then(function(ph) {
    // use ph
});

/* Set a custom logger object directly in the create call. Note that `info` is not provided here and so
var log = console.log;
var nolog = function() {};
phantom.create([], { logger: { warn: log, debug: nolog, error: log } }).then(function(ph) {
    // use ph
});
```

`page` object API

The `page` object that is returned with `#createPage` is a proxy that sends all methods to `phantom`. Most method calls should be identical to PhantomJS API. You must remember that each method returns a `Promise`.

`page#setting`

`page.settings` can be accessed via `page.setting(key)` or set via `page.setting(key, value)`. Here is an example to read `javascriptEnabled` property.

```
page.setting('javascriptEnabled').then(function(value){
    expect(value).toEqual(true);
});
```

`page#property`

Page properties can be read using the `#property(key)` method.

```
page.property('plainText').then(function(content) {
  console.log(content);
});
```

Page properties can be set using the `#property(key, value)` method.

```
page.property('viewportSize', {width: 800, height: 600}).then(function() {
});
```

When setting values, using `then()` is optional. But beware that the next method to phantom will block until it is ready to accept a new message.

You can set events using `#property()` because they are property members of `page`.

```
page.property('onResourceRequested', function(requestData, networkRequest) {
    console.log(requestData.url);
});
```

It is important to understand that the function above executes in the PhantomJS process. PhantomJS does not share any memory or variables with node. So using closures in javascript to share any variables outside of the function is not possible. Variables can be passed to `#property` instead. So for example, let's say you wanted to pass `process.env.DEBUG` to `onResourceRequested` method above. You could do this by:

```
page.property('onResourceRequested', function(requestData, networkRequest, debug) {
    if(debug){
      // do something with it
    }
}, process.env.DEBUG);
```

Even if it is possible to set the events using this way, we recommend you use `#on()` for events (see below).

You can return data to NodeJS by using `#createOutObject()`. This is a special object that let's you write data in PhantomJS and read it in NodeJS. Using the example above, data can be read by doing:

```
var outObj = phInstance.createOutObject();
outObj.urls = [];
page.property('onResourceRequested', function(requestData, networkRequest, out) {
    out.urls.push(requestData.url);
}, outObj);

// after call to page.open()
outObj.property('urls').then(function(urls){
   console.log(urls);
});
```

`page#on`

By using `on(event, [runOnPhantom=false],listener, args*)`, you can listen to the events the page emits.

```
var urls = [];

page.on('onResourceRequested', function (requestData, networkRequest) {
    urls.push(requestData.url); // this would push the url into the urls array above
    networkRequest.abort(); // This will fail, because the params are a serialized version of what was p
});

page.load('http://google.com');
```

As you see, using on you have access to the closure variables and all the node goodness using this function ans in contrast of setting and event with property, you can set as many events as you want.

If you want to register a listener to run in phantomjs runtime (and thus, be able to cancel the request lets say), you can make it by passing the optional param `runOnPhantom` as `true`;

```
var urls = [];

page.on('onResourceRequested', true, function (requestData, networkRequest) {
    urls.push(requestData.url); // now this wont work, because this function would execute in phantom ru
    networkRequest.abort(); // This would work, because you are accessing to the non serialized networkR
});

page.load('http://google.com');
```

The same as in property, you can pass additional params to the function in the same way, and even use the object created by `#createOutObject()`.

You cannot use `#property()` and `#on()` at the same time, because it would conflict. Property just sets the function in phantomjs, while `#on()` manages the event in a different way.

`page#off`

`#off(event)` is usefull to remove all the event listeners set by `#on()` for ans specific event.

`page#evaluate`

Using `#evaluate()` is similar to passing a function above. For example, to return HTML of an element you can do:

```
page.evaluate(function() {
    return document.getElementById('foo').innerHTML;
}).then(function(html){
    console.log(html);
});
```

`page#evaluateAsync`

Same as `#evaluate()`, but function will be executed asynchronously and there is no return value. You can specify delay of execution.

```
page.evaluateAsync(function(apiUrl) {
    $.ajax({url: apiUrl, success: function() {}});
}, 0, "http://mytestapi.com")
```

`page#evaluateJavaScript`

Evaluate a function contained in a string. It is similar to `#evaluate()`, but the function can't take any arguments. This example does the same thing as the example of `#evaluate()`:

```
page.evaluateJavaScript('function() { return document.getElementById(\'foo\').innerHTML; }').then(functi
    console.log(html);
});
```

`page#switchToFrame`

Switch to the frame specified by a frame name or a frame position:

```
page.switchToFrame(framePositionOrName).then(function() {
    // now the context of `page` will be the iframe if frame name or position exists
});
```

`page#switchToMainFrame`

Switch to the main frame of the page:

```
page.switchToMainFrame().then(function() {
    // now the context of `page` will the main frame
});
```

`page#uploadFile`

A file can be inserted into file input fields using the `#uploadFile(selector, file)` method.

```
page.uploadFile('#selector', '/path/to/file').then(function() {

});
```

Advanced

Methods below are for advanced users. Most people won't need these methods.

`page#defineMethod`

A method can be defined using the `#defineMethod(name, definition)` method.

```
page.defineMethod('getZoomFactor', function() {
    return this.zoomFactor;
});
```

`page#invokeAsyncMethod`

An asynchronous method can be invoked using the `#invokeAsyncMethod(method, arg1, arg2, arg3...)` method.

```
page.invokeAsyncMethod('open', 'http://phantomjs.org/').then(function(status) {
    console.log(status);
});
```

`page#invokeMethod`

A method can be invoked using the `#invokeMethod(method, arg1, arg2, arg3...)` method.

```
page.invokeMethod('evaluate', function() {
    return document.title;
}).then(function(title) {
    console.log(title);
});
```

```
page.invokeMethod('evaluate', function(selector) {
    return document.querySelector(selector) !== null;
}, '#element').then(function(exists) {
    console.log(exists);
});
```

Pooling

Creating new phantom instances with `phantom.create()` can be slow. If you are frequently creating new instances and destroying them, as a result of HTTP requests for example, it might be worth creating a pool of instances that are re-used.

See the phantom-pool module for more info.

Tests

To run the test suite, first install the dependencies, then run `npm test`:

```
$ npm install
$ npm test
```

Contributing

This package is under development. Pull requests are welcomed. Please make sure tests are added for new functionalities and that your build does pass in TravisCI.

People

The current lead maintainer is Amir Raminfar

List of all contributors

License

ISC