JS - Code Coverage

Sudheer

Java Script Unit Test Framework / Test Runner

- Some of the popular java script test frameworks,
    - Jasmine
    - Mocha
    - Karma
    - QUnit
    - YUITest

Java Script Code Coverage Tools

- The following are the popular code coverage tools available in the market.

- Blanketjs

- Istanbul

- JSCover

blanket.js

A seamless JavaScript code coverage library that works both in-browser and with nodejs.

FYI: Please note that this repo is not actively maintained

If you're looking for a more active project for JavaScript code coverage, I recommend Istanbul.

Ref: https://github.com/alex-seville/blanket#blanketjs

Philosophy

Blanket.js is a code coverage tool for javascript that aims to be:

- Easy to install
- Easy to configure
- Easy to use
- Easy to understand

Blanket.js can be run seamlessly or can be customised for your needs.

Download & Install

Download

The only file you need is blanket.js. Download the library and reference it in your test runner.

```
<script src="blanket.js"></script>
```

Configure

Add a data-cover attribute to the script reference of each file you want covered

```
<script src="my-library.js" data-cover></script>
```

Install

- Run your test as usual, and the coverage details will automatically be reported.
    - Install from npm
        - npm install blanket
    - Make sure you require Blanket before you require or run any of the code you want covered
        - require("blanket")({ /* optional options */ }), require("src/myscripttotest");
- Run your tests using mocha and take advantage of the json-cov and html-cov reporters to output the coverage results.

Reference

- http://blanketjs.org/
- https://github.com/alex-seville/blanket#philosophy
- https://github.com/addyosmani/backbone-koans-qunit

Istanbul

Istanbul - a JS code coverage tool written in JS

It computes statement, line, function and branch coverage

code coverage metrics

- **Statements**: How many of the statements in you code are executed.
- **Branches**: Conditional statements create branches of code which may not be executed (e.g. `if/else`). This metric tells you how many of your branches have been executed.
- **Functions**: The proportion of the functions you have defined which have been called.
- **Lines**: The proportion of lines of code which have been executed.

Install npm install -g istanbul

Features

- All-javascript instrumentation library that tracks statement, branch, and function coverage.
- Module loader hooks to instrument code on the fly
- Command line tools to run node unit tests "with coverage turned on" and no cooperation whatsoever from the test runner
- Multiple report formats: HTML, LCOV, Cobertura and more.
- Ability to use as middleware when serving JS files that need to be tested on the browser.
- Can be used on the command line as well as a library
- Based on the awesome `esprima` parser and the equally awesome `escodegen` code generator
- Well-tested on node (prev, current and next versions) and the browser (instrumentation library only)

Getting Sample JS code

- Example code for running code coverage is below
  - https://github.com/dwyl/learn-istanbul
- cd <go to the path where .js file exists>
- run any of the following commands to generate coverage
  - npm run coverage
  - npm test
  - istanbul cover test.js

Package.JSON

- All npm packages contain a file, usually in the project root, called package.json.
- this file holds various metadata relevant to the project.
- This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.
- It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package.
- The package.json file is normally located at the root directory of a Node.js project.
- **Reference**: https://docs.npmjs.com/files/package.json

```
{
  "name": "learn-mocha",
  "repository": {
    "type": "git",
    "url": "https://github.com/nelsonic/learn-mocha.git"
  },
  "description": "Simple mocha.js tutorial including istanbul test coverage",
  "version": "1.0.0",
  "devDependencies": {
    "mocha": "^2.2.5",
    "istanbul":"^0.3.17",
    "codeclimate-test-reporter": "^0.0.4"
  },
  "scripts": {
    "test": "/usr/local/lib/node_modules/mocha/bin/mocha --recursive",
    "coverage":"istanbul cover /usr/local/lib/node_modules/mocha/bin/_mocha -- -R spec --recursive",
    "codeclimate":
"CODECLIMATE_REPO_TOKEN=8138682931f34a05f1829d6c49d791a097e820489d6f98d3d0c0d829c5612585 ./node_modules/
codeclimate-test-reporter/bin/codeclimate.js < ./coverage/lcov.info"
  },
  "author": "nelsonic <nodecoder@gmail.com> (https://github.com/nelsonic)",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/docids/learn-mocha/issues"
  }
}
```

Reference

- https://github.com/gotwarlost/istanbul
- https://github.com/dwyl/learn-istanbul
- https://www.npmjs.com/package/istanbul
- https://medium.com/@the1mills/front-end-javascript-test-coverage-with-istanbul-selenium-4b2be44e3e98

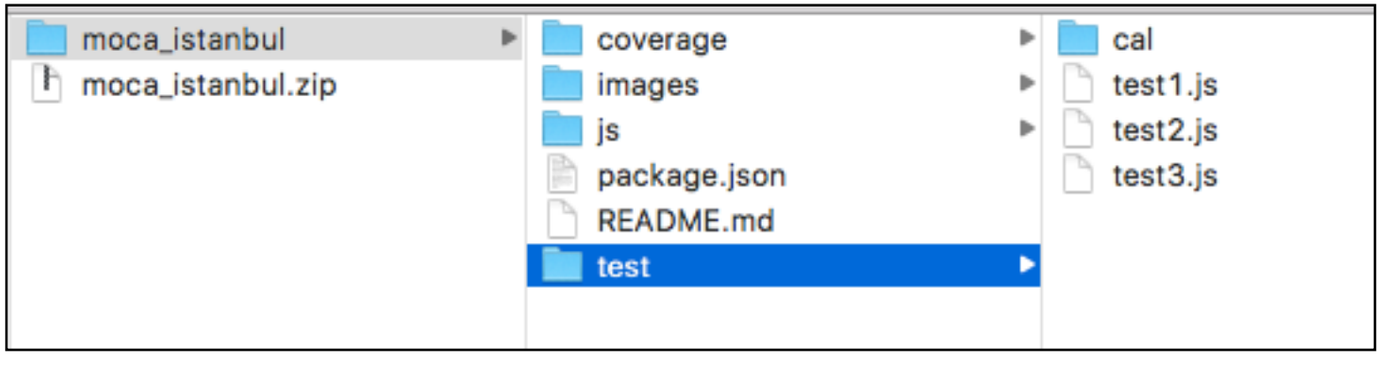
Mocha with Istanbul

Getting JS Code coverage for the unit tests written in MOCHA js unit test framework

npm install --global mocha

npm install -g istanbul

- go to project folder where it contains ‘test’ folder
- cd learn-mocha-master
- mocha (run mocha in terminal, which finds ’test’ folder)
- istanbul cover _mocha -- -R spec (for run & get coverage)
- From Package.json
  - npm test
  - npm run converage



Assertions can be used with mocha

- Mocha allows you to use any assertion library you wish.
- we're using Node.js' built-in assert module–but generally.
    - should.js - BDD style shown throughout these docs
    - expect.js - expect() style assertions
    - chai - expect(), assert() and should-style assertions
    - better-assert - C-style self-documenting assert()
    - unexpected - "the extensible BDD assertion toolkit"

Mocha commands

Command 1: get list of reporters available

$ mocha — reporters

```
dot - dot matrix
doc – html documentation
spec – hierarchical spec list
json – single json object
progress – progress bar
list – spec-style listing
tap – test-anything-protocol
landing – unicode landing strip
xunit – xunit reporter
min – minimal reporter (great with --watch)
json-stream – newline delimited json events
markdown – markdown documentation (github flavour)
nyan – nyan cat!
```

Command 2: prints report in html format

$ mocha -R doc

Command 3: redirect output to a file

```
$ mocha -R xunit --reporter-options output=filename.xml
```

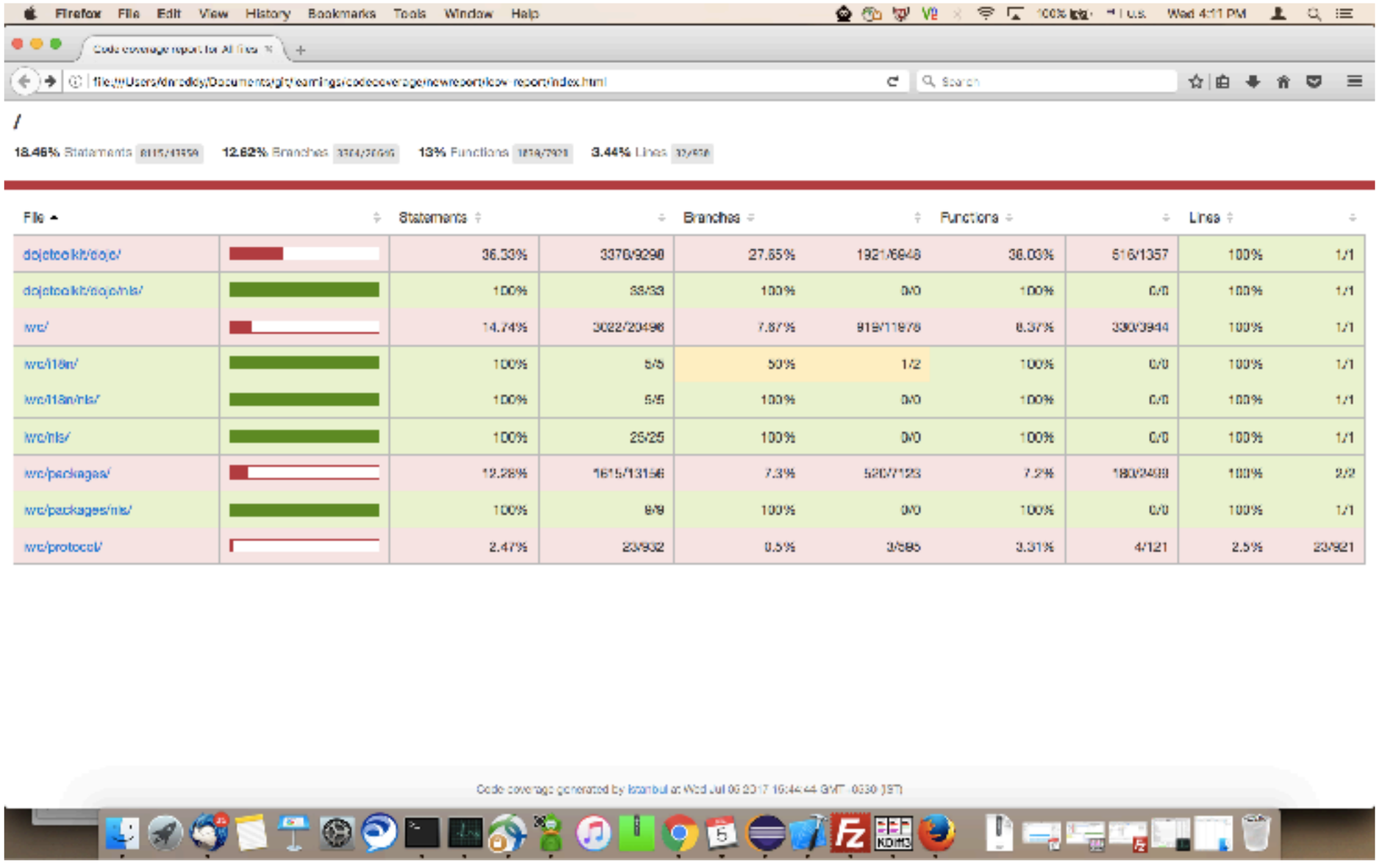
Selenium With Istanbul

- Instrument java script code.
    - istanbul instrument iwc_static/js --output iwc_static/js_ins --embed-source true (or)
    - istanbul instrument js/ --output js_ins/ -x '**/dojotoolkit/**' --embed-source true —complete-copy=true
    - istanbul instrument iwc_static/js/ --output iwc_static/js_ins/ --embed-source true --config=js/config.yml
    - you can download the js files from linux setup, do the instrumentation as shown in above command, replace the instrumented js files in linux setup (its better to remove the files and copy to the linux setup rather just replace)
- Make sure the application is accessing the instrumented java script code (if you done the above setup correctly, it will taken care automatically)
- From selenium automation code, access the application and run the tests.
- Collect the code coverage after every test run, the following code to be innovate for collecting code coverage to file system.
    - Make sure the .json file should be different while collecting the code coverage, other wise older coverage details will not be present.

```
JavascriptExecutor js = (JavascriptExecutor) driver;
Object coverageObject = js.executeScript("return JSON.stringify(window.__coverage__);”);
FileWriter file = new FileWriter("/report/coverage.json")) // needs to be called coverage*.json
file.write(coverageObject.toString());
```

- After all your tests run , execute the below command where your coverage*.json files are available to generate HTML Report
    - istanbul report --root report/ --dir newreport/
- Alternative, To get js code coverage with selenium code written in javascript https://github.com/gotwarlost/istanbul-middleware

JS Code coverage report taken for Convergence for Few scenarios

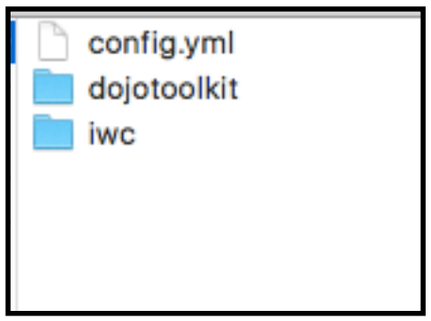


Selenium code for getting JS Code Coverage

```
public class CodeCoverage
{
	WebDriver driver;

	@Test
	public void test1() throws ClientProtocolException, IOException, InterruptedException
	{
		System.setProperty("webdriver.gecko.driver", "/Users/dnreddy/Documents/selenium/driver/geckodriver");
		ProfilesIni profile = new ProfilesIni();
		FirefoxProfile myprofile = profile.getProfile("selenium");//location of your new firefox profile
		driver = new FirefoxDriver(myprofile);
		driver.get("https://blr2222220.idc.oracle.com/iwc");

		driver.findElement(By.id("username")).clear();
		driver.findElement(By.id("username")).sendKeys("admin");
		driver.findElement(By.id("password")).clear();
		driver.findElement(By.id("password")).sendKeys("adminpass");

		driver.findElement(By.id("signin")).click();
		Thread.sleep(20*1000);

		driver.findElement(By.linkText("Sign out")).click();
		Thread.sleep(10*1000);
	}

	@BeforeMethod
	public void tearDown() throws IOException
	{
		JavascriptExecutor js = (JavascriptExecutor) driver;
		Object coverageObject = js.executeScript("return JSON.stringify(window.__coverage__);");
		FileWriter file = new FileWriter("/Users/dnreddy/Documents/git/learnings/codecoverage/jsonreport/coverage3.json");
		file.write(coverageObject.toString());
		file.flush();
	}
}
```

config.yml

```
verbose: true
instrumentation:
    root: .
    extensions:
        - .js
    default-excludes: true
    excludes: ['**/dojotoolkit/**']
    embed-source: true
    variable: __coverage__
    compact: true
    preserve-comments: false
    complete-copy: true
    save-baseline: false
    baseline-file: ./coverage/coverage-baseline.json
    include-all-sources: false
    include-pid: false
    es-modules: false
reporting:
    print: summary
    reports:
        - lcov
    dir: ./coverage
    watermarks:
        statements: [50, 80]
        lines: [50, 80]
        functions: [50, 80]
        branches: [50, 80]
    report-config:
        clover: {file: clover.xml}
        cobertura: {file: cobertura-coverage.xml}
        json: {file: coverage-final.json}
        json-summary: {file: coverage-summary.json}
        lcovonly: {file: lcov.info}
        teamcity: {file: null, blockName: Code Coverage Summary}
        text: {file: null, maxCols: 0}
        text-lcov: {file: lcov.info}
        text-summary: {file: null}
hooks:
    hook-run-in-context: false
    post-require-hook: null
    handle-sigint: false
check:
    global:
        statements: 0
        lines: 0
        branches: 0
        functions: 0
        excludes: []
    each:
        statements: 0
        lines: 0
        branches: 0
        functions: 0
        excludes: []
```



The Config.yml should be placed in the root of the source tree.