Node.js Core Modules

David Z. Han

Chapter 1

Assertion Testing

1.1 Introduction

This module is used for writing unit tests for your applications, you can access it with `require('assert')`.

1.2 assert.fail

Syntax:

```
assert.fail(actual, expected, message, operator)
```

where

- `actual`: the actual value returned by your function
- `expected`: the value you expect your function returns
- `message`: the message you want to display when an exception is thrown
- `operator`: the operator is used to separate the values for `actual` and `expected` when `message` is falsy.

This function throws an exception that displays the message your provided via parameter `message` or the values for `actual` and `expected` separated by the delimiter you provide via parameter `separator`.

Three Arguments

Let's start with an example with three arguments: `actual`, `expected` and `message`, which is also used most often in practice. It is shown in *Listing 1.1*.

```
1 var assert = require('assert');
2
3 assert.fail(1, 2, '1 is not equal to 2');
```

Listing 1.1: *An exception is thrown with customized message*

Running the code in *Listing 1.1* produces the log as shown in *Listing 1.2*.

```
1
2 assert.js:92
3   throw new assert.AssertionError({
4         ^
5 AssertionError: 1 is not equal to 2
6     at Object.<anonymous> (/node/src/assert/001_fail_with_message.js:3:8)
7     at Module._compile (module.js:456:26)
8     at Object.Module._extensions..js (module.js:474:10)
9     at Module.load (module.js:356:32)
10    at Function.Module._load (module.js:312:12)
11    at Function.Module.runMain (module.js:497:10)
12    at startup (node.js:119:16)
13    at node.js:906:3
```

Listing 1.2: *The error messages of running script in Listing 1.1*

A few takeaways from the error messages in *Listing 1.2*.

- the exception thrown by `assert.fail` is an instance of `assert.AssertionError`, which inherits from Javascript's `Error` object.
- the error message `"1 is not equal to 2"` is the customized message provided by you.

We can also put the function call into a `try...catch` statement as shown in *Listing 1.3*.

```
1 var assert = require('assert');
2
3 try {
4     assert.fail(1, 2, '1 is not equal to 2');
5 } catch (e) {
6     console.log(e.message);
7 }
```

Listing 1.3: *An exception is thrown and caught*

Four Arguments

The fourth argument `operator` will be used to separate the the actual value and expected value when the third argument `message` is falsy. We add the fourth argument as shown in *Listing 1.4*.

```
1 var assert = require('assert');
2
3 assert.fail(1, 2, '1 is not equal to 2', '==');
```

Listing 1.4: *An exception is thrown with truthy message*

Running the code in *Listing 1.4* produces the error stack trace as shown in *Listing 1.5*.

```
1
2 assert.js:92
3   throw new assert.AssertionError({
4         ^
5 AssertionError: 1 is not equal to 2
```

```
    at Object.<anonymous> (/node/src/assert/001_fail_truthy_message.js:3:8)
    at Module._compile (module.js:456:26)
    at Object.Module._extensions..js (module.js:474:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Function.Module.runMain (module.js:497:10)
    at startup (node.js:119:16)
    at node.js:906:3
```

Listing 1.5: *The error messages of running script in Listing 1.4*

An astute reader will notice that the error message in *Listing 1.5* is the same as the one in *Listing 1.2*, which indicates that the fourth argument `operator` is not used when the third argument `message` is truthy. Let's see what will happen if the third argument is falsy. The code snippet in *Listing 1.6* shows the changes.

```
var assert = require('assert');

assert.fail(1, 2, '', '==');
```

Listing 1.6: *An exception is thrown with falsy message*

The error message in *Listing 1.5* shows that the the `operator` we provided is used to separate the actual value and the expected value.

```

assert.js:92
  throw new assert.AssertionError({
        ^
AssertionError: 1 == 2
    at Object.<anonymous> (/node/src/assert/001_fail_empty_message.js:3:8)
    at Module._compile (module.js:456:26)
    at Object.Module._extensions..js (module.js:474:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Function.Module.runMain (module.js:497:10)
    at startup (node.js:119:16)
    at node.js:906:3
```

Listing 1.7: *The error messages of running script in Listing 1.6*

1.3 assert.ok

Syntax:

```
assert.ok(value, [message])
```

where

- `value`: check whether the value is truthy
- `message`: the message you want to display when an exception is thrown

This function is equivalent to function `assert.equal(!!value, true, message);` in Section 1.5.

One Argument

The script in *Listing 1.8* shows two simple examples of using this function. The error message in *Listing 1.11* shows operator "==" is used to separate the actual value and the expected value.

```
1 var assert = require('assert');
2
3 assert.ok(1);
4 assert.ok(0);
```

Listing 1.8: *One is Okay and the other is not Okay*

```
1
2 assert.js:92
3   throw new assert.AssertionError({
4         ^
5 AssertionError: 0 is not truthy
6     at Object.<anonymous> (/node/src/assert/002_ok_with_message.js:4:8)
7     at Module._compile (module.js:456:26)
8     at Object.Module._extensions..js (module.js:474:10)
9     at Module.load (module.js:356:32)
10    at Function.Module._load (module.js:312:12)
11    at Function.Module.runMain (module.js:497:10)
12    at startup (node.js:119:16)
13    at node.js:906:3
```

Listing 1.9: *The error messages of running script in Listing 1.8*

Two Arguments

The script in *Listing 1.10* shows two examples with customized message. The customized message shows up in the error log in *Listing 1.11*.

```
1 var assert = require('assert');
2
3 assert.ok(1, '1 is truthy');
4 assert.ok(0, '0 is not truthy');
```

Listing 1.10: *One is Okay and the other is not Okay: with optional truthy message*

```
1
2 assert.js:92
3   throw new assert.AssertionError({
4         ^
5 AssertionError: 0 is not truthy
6     at Object.<anonymous> (/node/src/assert/002_ok_with_message.js:4:8)
7     at Module._compile (module.js:456:26)
8     at Object.Module._extensions..js (module.js:474:10)
9     at Module.load (module.js:356:32)
10    at Function.Module._load (module.js:312:12)
11    at Function.Module.runMain (module.js:497:10)
12    at startup (node.js:119:16)
13    at node.js:906:3
```

Listing 1.11: *The error messages of running script in Listing 1.10*

1.4 assert

Syntax:

```
assert(value, [message])
```

where

- `value`: check whether the value is truthy
- `message`: the message you want to display when an exception is thrown

This function is exactly the same as function `assert.ok`. In fact, in the source code for module `assert`, you can see the line below:

```
var assert = module.exports = ok;
```

1.5 assert.equal

Syntax:

```
assert.equal(actual, expected, [message])
```

where

- `actual`: the actual value returned by your function
- `expected`: the value you expect your function returns
- `message`: the message you want to display when an exception is thrown

The `assert.equal` method is implemented in Node.js as follows:

```
1 assert.equal = function equal(actual, expected, message) {
2     if (actual != expected) {
3         fail(actual, expected, message, '==', assert.equal);
4     }
5 };
```

Listing 1.12: *The implementation of `assert.equal` in Node.js*

It is clear to see in *Listing 1.12* that `assert.equal` performs shallow, coercive equality test with the equal comparison operator (==). Let's verify this with a few examples.

```
1 var assert = require('assert');
2
3 assert.equal(3, 3, '3 is equal to 3');
4 assert.equal(3, 3.0, '3 is equal to 3.0');
5 assert.equal('3', 3, '"3" is equal to 3');
6 assert.equal('abc', 'abc');
7
8 var obj = {a: 1};
9 assert.equal(obj, obj);
```

Listing 1.13: *Passing examples of `assert.equal`*

All examples in *Listing 1.13* pass. *Listing 1.14* provides a few failing examples. You may wonder why the assertion in Line 4 fails. Though the two objects have the same properties and values, they refer to two physically different objects. In another word, `assert.equal` checks whether `actual` and `expected` points two the same object when both of them are objects.

```
1 var assert = require('assert');
2
3 assert.equal(3, 2, '3 is equal to 2');
4 assert.equal({a: 1}, {a: 1});
```

Listing 1.14: *Failing examples of* `assert.equal`

1.6 assert.notEqual

Syntax:

```
assert.notEqual(actual, expected, [message])
```

where

- `actual`: the actual value returned by your function
- `expected`: the value you expect your function returns
- `message`: the message you want to display when an exception is thrown

Chapter 2

Child Process

2.1 Introduction

2.2 child.pid

This property is an integer, indicating the process identifier (normally referred to as the PID) of a child process. This PID is used to uniquely identify a process.

```
1 var spawn = require('child_process').spawn;
2 var ls    = spawn('ls', ['-l']);
3
4 console.log('Spawned child pid: ' + ls.pid);
```

Listing 2.1: *The PID of a child process is written to the console*

Line 4 in Listing **??** prints the process ID to the console.

2.3 child.kill

We can use this function to send a signal to a child process. A default signal `SIGTERM` will be sent if no argument is provided as shown in Listing **??**. When running the above code, it is verified that the process is terminated by signal `SIGTERM`.

```
1 var spawn = require('child_process').spawn;
2 var grep = spawn('grep', ['ssh']);
3
4 grep.on('close', function (code, signal) {
5     console.log('The child process is killed after receiving signal ' + signal);
6 });
7
8 grep.kill();
```

Listing 2.2: *A child process is terminated by the default signal `SIGTERM`*

Now we send a signal `SIGHUP` to the child process as shown in Listing **??**. When running the above code, it is verified that the process is terminated by signal `SIGHUP`.

```
1 var spawn = require('child_process').spawn;
```

```
var grep = spawn('grep', ['ssh']);

grep.on('close', function (code, signal) {
    console.log('The child process is killed after receiving signal ' + signal);
});

grep.kill('SIGHUP');
```

Listing 2.3: *A child process is terminated by signal `SIGHUP`*

What if sending an invalid signal to a child process? A signal `NOOP` is sent to a child process as demonstrated in Listing **??**. When running the above code, it is verified that an error is thrown and it complains about the unknown signal `NOOP`.

```
var spawn = require('child_process').spawn;
var grep = spawn('grep', ['ssh']);

grep.on('close', function (code, signal) {
    console.log('The child process is killed after receiving signal ' + signal);
});

grep.kill('NOOP');
```

Listing 2.4: *An invalid signal `NOOP` is sent to a child process*

Finally if `child.kill()` is not called and a child process exits gracefully, a `null` signal is expected. It is verified by the code in Listing **??**.

```
var spawn = require('child_process').spawn;
var ls = spawn('ls', ['-l']);

ls.on('close', function (code, signal) {
    console.log('The child process gracefully exits with signal ' + signal);
});
```

Listing 2.5: *No signal is sent to a child process and it exits cleanly*

2.4 child_process.spawn

This function allows us to launch a new process with the given command. The code in Listing **??** lists all files under the current directory and prints the file list to the console.

```
var spawn = require('child_process').spawn;
var ls = spawn('ls', ['-lh', '.']);

ls.stdout.on('data', function (data) {
    console.log('stdout: ' + data);
});

ls.on('close', function (code) {
    console.log('child process exited with code ' + code);
});
```

Listing 2.6: *Spawn a new process to list files under the current directory*

Chapter 3

Cluster

3.1 Introduction

Chapter 4

Events

4.1 Introduction

Chapter 5

File System

5.1 Introduction