Promise with Bluebird

Kieve Chua

- Performance
- Basic & Chaining
- Collection
- Error handling
- Future

Flow Control Library

- Callback
  - Async - https://github.com/caolan/async
  - Step - https://github.com/creationix/step
- Promises
  - Q - https://github.com/kriskowal/q
  - When - https://github.com/cujojs/when
- Generator
  - Co - https://github.com/visionmedia/co
  - Task.js - http://taskjs.org/

What’s promise?

http://promises-aplus.github.io/promises-spec/

Promises are about making asynchronous code retain most of the lost properties of synchronous code such as flat indentation and one exception channel. - Petka Antonov

Bluebird

A full featured promise library with unmatched performance.

Source: https://github.com/petkaantonov/bluebird

Author: Petka Antonov

P/S: Don’t pronounce it in Hokkien :P

Performance, why it’s matter?

Benchmark

results for 10000 parallel executions, 1 ms per I/O op

file	time(ms)	memory(MB)
• callbacks-baseline.js	172	34.69
• promises-bluebird-generator.js	234	39.63
• promises-bluebird.js	312	48.68
• callbacks-caolan-async-waterfall.js	609	75.96
• promises-cujojs-when.js	4820	285.99
• promises-kriskowal-q.js	23322	713.96

Getting started

```
1   var Promise = require('bluebird');
2
3   new Promise(function (resolve, reject) {
4       Http.get(function (error, result) {
5           if (error) {
6               reject(error);
7           }
8
9           if (result) {
10              // Do something
11              resolve(result);
12          }
13      });
14  })
15  .then(
16      function (result) {
17          // Handle result
18      },
19      function (error) {
20          // Handle error
21      }
22  )
23  .catch(function (error) {
24      // Unified error handler
25  });
```

Deferred

```
1   function someFunction() {
2       var defer = Promise.defer();
3
4       setTimeout(function(){
5           defer.resolve('Done');
6       }, 1000);
7
8       return defer.promise;
9   }
10
11  someFunction.then(function (result) {
12      console.log(result);
13  });
```

Working with Nodejs

```
1  var fs = Promise.promisifyAll(require("fs"));
2
3  fs.readFileAsync('./package.json', 'utf-8')
4  .then(function (result) {
5      console.log(result);
6  });
```

- Does not overwritten existing methods.
- Create new function with prefix,
  e.g. fs.<originalName>Async

Wrapping or Promisification?

- If handling none Nodejs's callback
  - new Promise(function () {});
  - Promise.defer();
- Else
  - Promise.promisify(Function nodeFunction [, dynamic receiver])
  - Promise.promisifyAll(Object target)

Chaining

```
1  gatherNodejsProgrammer()
2  .then(function (programmers) {
3      return injectPromise(programmers);
4  })
5  .then(function (awesomeProgrammers) {
6      res.json(awesomeProgrammers);
7  });
```

Return promise in .then will pass it to next .then

Chaining Cont.

```
1   fs.readFileAsync('./package.json', 'utf-8')
2   .then(function (result) {
3       doSomehing(result);
4
5       return [1, 2, 3];
6   })
7   .then(function (result) {
8       console.log(result);
9       // [1, 2, 3]
10  });
```

Any object returned in .then will be convert into promise.
Even return a function.

```
User.withFacebookLogin()
// Get user with email
.then(function (users) {
    return _.filter(users, function (user) {
        if (!_.isEmpty(user.facebook.email)) {
            return true;
        }
    });
})
// Sent marketing campaign
.then(function (users) {
    Queue.push('marketingEmail', users);

    return Admin.inchargeOfMarketing();
})
// Alert marketing department of this week campaign
.then(function (admins) {
    Queue.push('marketingEmail', admins);

    res.json(200);
});

```

Run in parallel

```
1   var carPromise = Car.find();
2   var bikePromise = Bike.find();
3
4   Promise.all([carPromise, bikePromise])
5   .then(function (vehicles) {
6       // Do something
7   });
8
9   // or
10
11  Promise.join(carPromise, bikePromise)
12  .then(function (vehicles) {
13      // Do something
14  });
```

Data handling

- Map - format data
- Reduce - combine data
- Filter - filtering
- Zip - merge 2 arrays
- mergeAll - merge multiple result

Collection

```
Promise.reduce(
    [
        'file1.txt',
        'file2.txt',
        'file3.txt'
    ],
    function(total, fileName) {
        return fs.readFileAsync(fileName, 'utf8')
            .then(function(contents) {
                return total + parseInt(contents, 10);
            });
    },
    0
)
.then(function(total) {
    //Total is 30
});
```

```
var Promise = require('bluebird');
var fs = Promise.promisifyAll(require('fs'));

// Get current directory file/directory list
fs.readdirAsync('./')
// Reformat data to return file type only
.map(function (file) {
    var filePartial = file.split('.');

    return filePartial.length > 1 ? filePartial[filePartial.length - 1] : null;
})
// Filter out directory
.filter(function (fileType) {
    return fileType;
})
// Calculate occurence of each file types
.reduce(function (total, current) {
    if (typeof(total) === 'string') {
        total = {};
    }

    total[current] = total[current] ? total[current] + 1 : 1;

    return total;
})
.then(function (total) {
    console.log(total);
    // {
    //     js: 6,
    //     json: 1
    // }
});
```

Error handling

If error is handler here

It won't propagate to this function

```
1   fs.readdirAsync(directory)
2   .then(
3       function (result) {},
4       function (error) {
5           // Error handling
6       }
7   )
8   .catch(function (error) {
9       // Error handling
10  });
```

Specific Error handling

Unified error catching

Error handling

```
1   fs.readdirAsync(directory)
2   .then(function (result) {})
3   .error(function (result) {})
4   .catch(function (result) {})
```

- .error - Catch error from explicit rejections, e.g. reject('Error message')

- .catch - Catch all type of exceptions.

Error handling

```
1   new Promise(function (resolve, reject) {
2       var notFunction = 'Not function';
3
4       notFunction();
5   })
6   .catch(TypeError, function (error) {
7       console.log(error);
8   })
9   .catch(function (error) {
10      console.log(error);
11  });
```

Custom error?

```
1   function MyCustomError(message) {
2       this.message = message;
3       this.name = "MyCustomError";
4       Error.captureStackTrace(this, MyCustomError);
5   }
6
7   MyCustomError.prototype = Object.create(Error.prototype);
8   MyCustomError.prototype.constructor = MyCustomError;
9
10  Promise.resolve()
11  .then(function(){
12      throw new MyCustomError();
13  })
14  .catch(MyCustomError, function(e){
15      //will end up here now
16  });
```

Debug

Promise.longStackTraces();

- Cannot be disabled after being enabled
- BEWARE, long stack traces imply a substantial performance penalty, around 4-5x for throughput and 0.5x for latency.

Writing module

Mikeal said it: If you write a library based on promises, nobody is going to use it.

```
1   module.exports = function (data, next) {
2       return fs.readdirAsync(data).nodeify(next);
3   };
4
```

Best of both world

```
1   // Promise
2   stats(['./promise', './async'])
3   .then(function (result) {
4       console.log(result);
5   });
6
7   // Callback
8   stats('./promise', function (error, result) {
9       console.log(result);
10  });
```

Testing

```
describe('Test async', function () {
    it('does something asynchronous', function (done) {
        getSomePromise()
        .then(function (value) {
            value.should.equal('foo');
        })
        .then(function () {
            done();
        });
    });
});
```

https://github.com/domenic/mocha-as-promised

Generator

```
1   var getData = Promise.coroutine(function* (urlA, urlB) {
2       var resultA = yield http.getAsync(urlA);
3       var resultB = yield http.getAsync(urlB);
4   });
```

References

- http://jhusain.github.io/learnrx/
- http://spion.github.io/posts/analysis-generators-and-other-async-patterns-node.html
- http://spion.github.io/posts/why-i-am-switching-to-promises.html
- http://callbackhell.com/
- http://www.devthought.com/2011/12/22/a-string-is-not-an-error/

Contact me

- Twitter: @KieveChua
- Github: kievechua