NodeJS 中的异步

尚春

marketplace.500px.com/photos/139175555

提纲

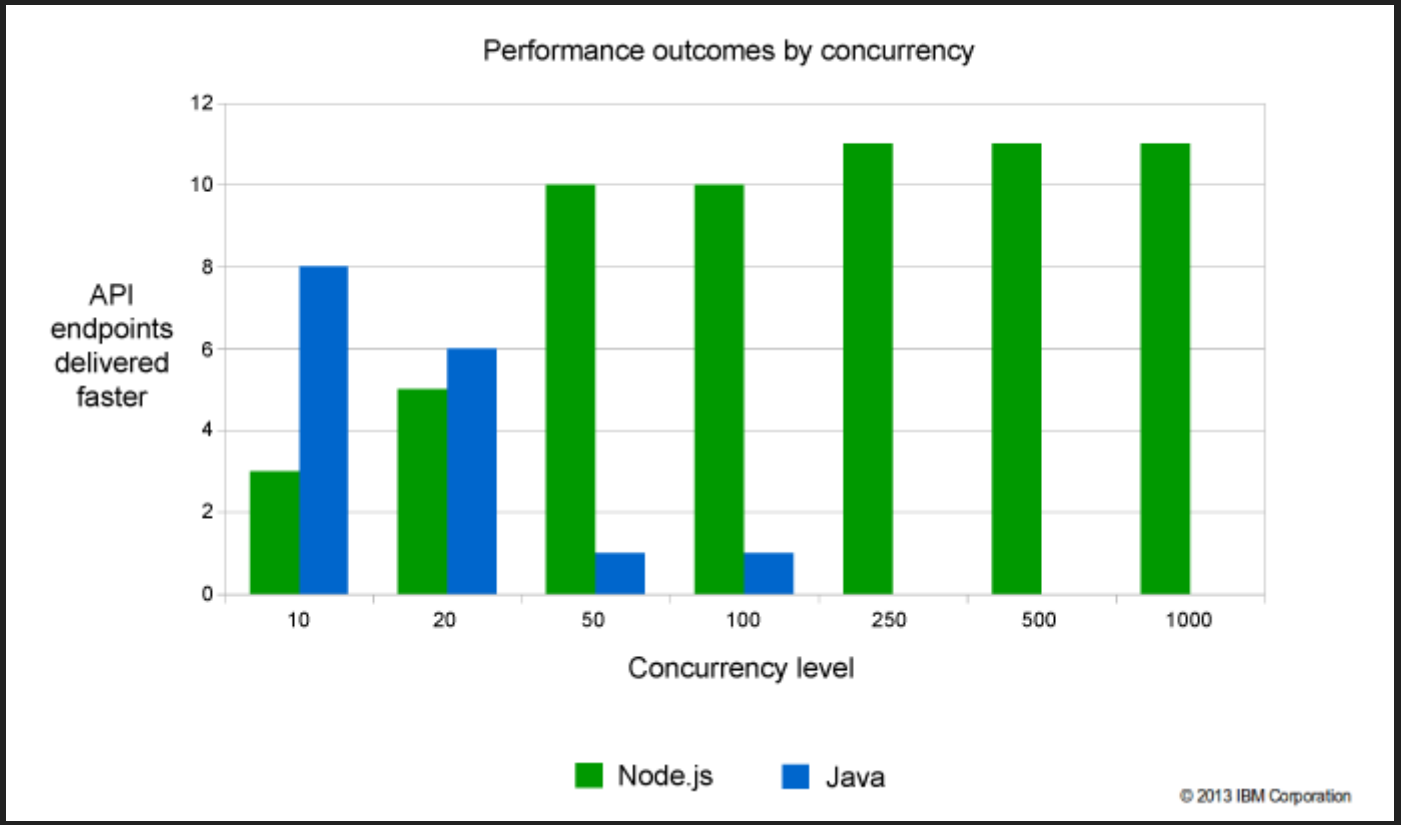
- 为异步而生
- Event Loop
- 异步模式
  - Callback
  - Promise
  - Generator
  - Async/Await

为异步而生

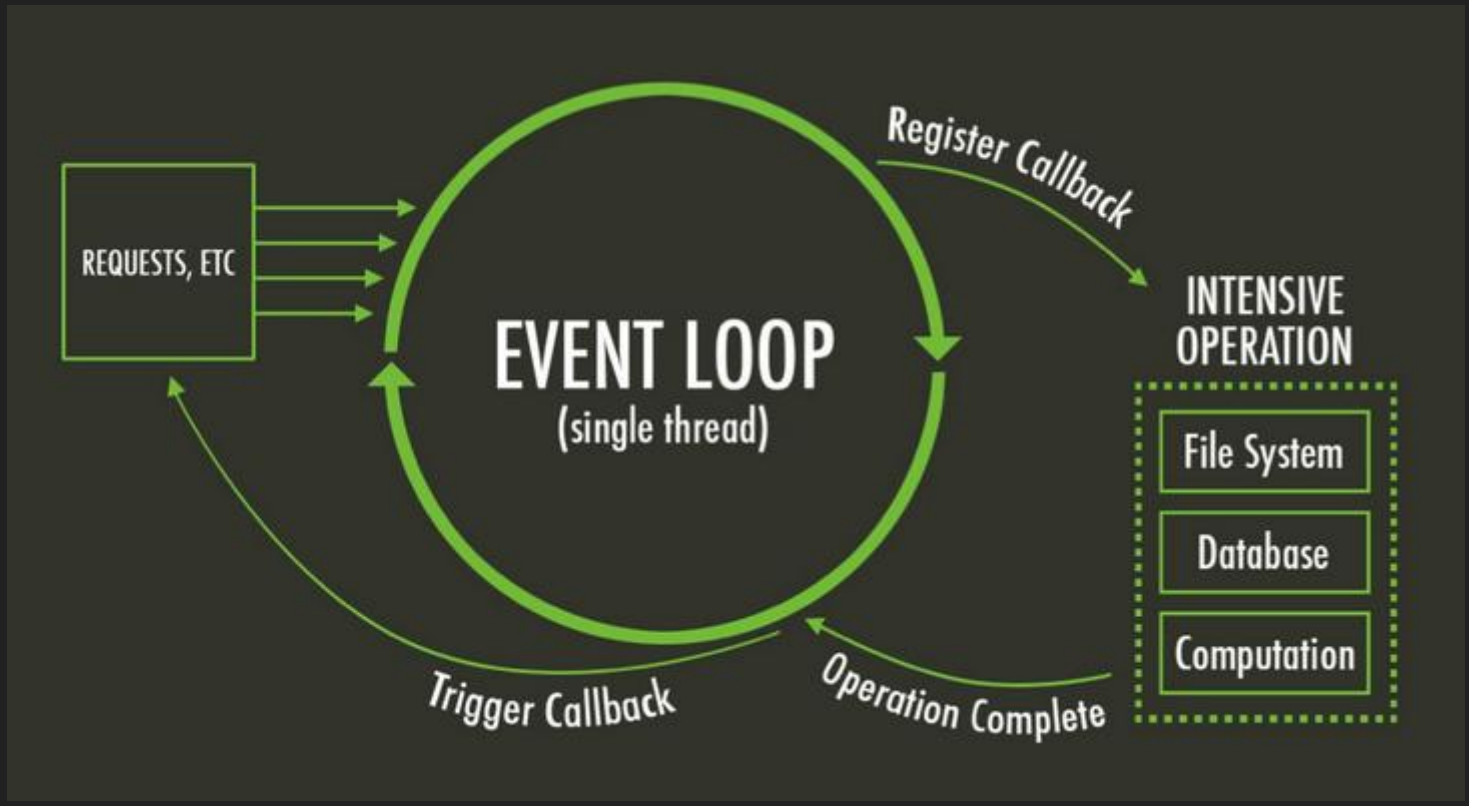
```
// NodeJS
var request = require('request');

request('http://www.10101111.com', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body);
  }
});

console.log('do other stuff');
```

```
// GoLang
response, _ := http.Get("http://www.10101111.com")
defer response.Body.Close()
body, _ := ioutil.ReadAll(response.Body)
fmt.Println(string(body))

fmt.Println("do other stuff")
```

异步带来的


Performance outcomes by concurrency
API endpoints delivered faster
12
10
8
6
4
2
0
10
20
50
100
250
500
1000
Concurrency level
Node.js
Java
© 2013 IBM Corporation

Event Loop



Demo by Philip Roberts

问题

找出给定目录中最大的文件

A: Callback

```
function findMaxSizeFile(dir, cb) {
  fs.readdir(dir, function (er, files) {
    if (er) return cb(er);
    var counter = files.length;
    var errored = false;
    var stats = [];

    files.forEach(function (file, index) {
      fs.stat(path.join(dir,file), function (er, stat) {
        if (errored) return;
        if (er) {
          errored = true;
          return cb(er);
        }
        stats[index] = stat;

        if (--counter == 0) {
          var largest = stats
            .filter(function (stat) { return stat.isFile(); })
            .reduce(function (prev, next) {
              if (prev.size > next.size) return prev;
              return next;
            })
          cb(null, files[stats.indexOf(largest)]);
        }
      });
    });
  });
};
```

小结

优势

- 简单
- 符合 Node 中多数 API 设计

不足

- 难以控制嵌套层级，常常遭遇 Callback Hell
- 需要自行控制回调协同
- 错误处理较为复杂

B: Callback with Async

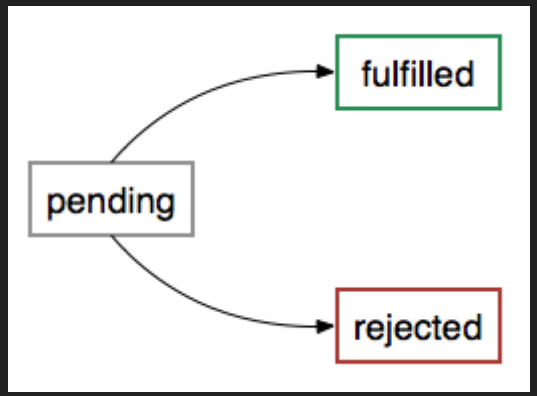
```
function findMaxSizeFile(dir, cb) {
  async.waterfall([
    function (next) {
      fs.readdir(dir, next)
    },
    function (files, next) {
      var paths = files.map(function (file) {
        return path.join(dir, file);
      });
      async.map(paths, fs.stat, function (err, stats) {
        next(err, files, stats);
      });
    },
    function (files, stats, next) {
      var largest = stats
        .filter(function (stat) { return stat.isFile(); })
        .reduce(function (prev, next) {
          if (prev.size > next.size) return prev;
          return next;
        });
      next(null, files[stats.indexOf(largest)]);
    }
  ], cb);
}
```

小结

- 弥补了原生 Callback 的主要问题
- 新的问题是：
  - Async 是一个第三方库，API 有记忆成本
  - 对外仍然通过 Callback，嵌套不可避免

C: Promise

A Promise represents a single asynchronous operation that hasn't completed yet, but is expected in the future.



Promise API

```
new Promise(function (resolve, reject) {})

Promise.all(iterable)
Promise.race(iterable)

Promise.resolve(value)
Promise.reject(reason)

Promise.prototype.then(onFulfilled, onRejected)
Promise.prototype.catch(onRejected)
```

Chaining & Error Bubbling

```
getTweetsFor('domenic')
  .then(function (tweets) {
    var shortUrls = parseTweetsForUrls(tweets);
    var mostRecentShortUrl = shortUrls[0];
    return expandUrlUsingTwitterApi(mostRecentShortUrl);
  })
  .then(httpGet)
  .then(
    function (responseBody) {
      console.log('Most recent link text:', responseBody);
    },
    function (error) {
      console.error('Error with the twitterverse:', error);
    }
  );
```

The Point of Promise

对比下同步代码：

```
try {
  var tweets = getTweetsFor("domenic");
  var shortUrls = parseTweetsForUrls(tweets);
  var mostRecentShortUrl = shortUrls[0];
  var responseBody = httpGet(expandUrlUsingTwitterApi(mostRecentShortUrl));
  console.log("Most recent link text:", responseBody);
} catch (error) {
  console.error("Error with the twitterverse: ", error);
}
```

Promise 获得了类似同步代码的顺序执行和错误处理能力

第三方库 Bluebird

```
var Promise = require('bluebird');

// 集合
Promise.map(iterable, mapper)
Promise.filter(iterable, filterer)
Promise.reduce(iterable, reducer, initialValue)

// 原型
Promise.prototype.finally(onFulfilled, onRejected)
Promise.prototype.spread(onFulfilled)

// 和 Generator 结合
Promise.coroutine(generatorFunction)
```

Promise 的解法

```
var denodeify = require('denodeify');
var fsReaddir = denodeify(fs.readdir);
var fsStat = denodeify(fs.stat);

function findMaxSizeFile(dir) {
  return fsReaddir(dir)
    .then(function (files) {
      var promises = files.map(function (file) {
        return fsStat(path.join(dir, file))
      })
      return Promise.all(promises).then(function (stats) {
        return { files: files, stats: stats };
      });
    })
    .then(function (data) {
      var largest = data.stats
        .filter(function (stat) { return stat.isFile(); })
        .reduce(function (prev, next) {
          if (prev.size > next.size) return prev;
          return next;
        });
      return data.files[data.stats.indexOf(largest)]
    });
}
```

小结

优势

- 为异步而设计
- 原生
- 可以获得类似同步代码的能力

不足

- 写法上和同步代码仍有差距

D: Generator

Generators are functions which can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances.

```
function* idMaker() {
  var index = 0;
  while (index < 3) {
    yield index++;
  }
}

var gen = idMaker();

console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
console.log(gen.next().value); // undefined
```

Generator 的解法

```
var co = require('co');
var thunkify = require('thunkify');
var readdir = thunkify(fs.readdir);
var stat = thunkify(fs.stat);

function findMaxSizeFile(dir) {
  return co(function* () {
    var files = yield readdir(dir);
    var stats = yield files.map(function (file) {
      return stat(path.join(dir, file));
    });
    var largest = stats
      .filter(function (stat) { return stat.isFile(); })
      .reduce(function (prev, next) {
        if (prev.size > next.size) return prev;
        return next;
      });
    return files[stats.indexOf(largest)];
  });
}
```

小结

优势

- 和同步代码几乎一致

不足

- 一般依赖第三方库，例如 co
- 设计本意是解决枚举问题，Hack 的味道比较浓

E: Async/Await

异步编程的终极方案

The language-level model for writing asynchronous code in ECMAScript.

```
const f = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(123);
    }, 2000);
  });
};

const testAsync = async () => {
  const t = await f();
  console.log(t);
};

testAsync();
```

Async/Await 的解法

```
const denodeify = require('denodeify');
const fsReaddir = denodeify(fs.readdir);
const fsStat = denodeify(fs.stat);

const findMaxSizeFile = async dir => {
  const files = await fsReaddir(dir);
  const stats = [];
  for (const file of files) {
    const stat = await fsStat(path.join(dir, file));
    stats.push({ file: file, stat: stat });
  }
  const largest = stats
    .filter(stat => stat.stat.isFile())
    .reduce((prev, next) => {
      if (prev.stat.size > next.stat.size) return prev;
      return next;
    });
  return largest.file;
};
```

小结

优势

- 基于 Promise 和 Generator，原语级支持
- 比 Promise 更简单，比 Generator 更面向异步

不足

- 尚在 Stage 3，即将成为 ES2017 标准，目前需要借助 Babel 等转译

参考

- Managing Node.js Callback Hell with Promises, Generators and Other Approaches
- You're Missing the Point of Promises
- Promisejs.org
- Proposal: Async Functions for ECMAScript

Thanks