link: null
title: 珠峰架构师成长计划
description: TDD指的是Test Drive Development，很明显的意思是测试驱动开发，也就是说我们可以从测试的角度来检验整个项目。大概的流程是先针对每个功能点抽象出接口代码，然后编写单元测试代码，接下来实现接口，运行单元测试代码，循环此过程，直到整个单元测试都通过。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=47 sentences=135, words=938

1. 单元测试

TDD指的是Test Drive Development，很明显的意思是测试驱动开发，也就是说我们可以从测试的角度来检验整个项目。大概的流程是先针对每个功能点抽象出接口代码，然后编写单元测试代码，接下来实现接口，运行单元测试代码，循环此过程，直到整个单元测试都通过。

- BDD指的是Behavior Drive Development，也就是行为驱动开发。
- 在TDD中，我们并不能完全保证根据设计所编写的测试就是用户所期望的功能。
- BDD将这一部分简单和自然化，用自然语言来描述，让开发、测试、BA以及客户都能在这个基础上达成一致。

2. 工具

```
const { expect } = require('chai');

describe('zftest', function () {
    it('#1', function () {
        expect(1 + 1).to.be.equal(2);
    })
    it('#2', function (done) {
        setTimeout(function () {
            expect(1 + 1).to.be.equal(2);
            done();
        }, 1000);
    })
    it('#3', function () {
        return new Promise(function (resolve) {
            setTimeout(function () {
                expect(1 + 1).to.be.equal(2);
                resolve();
            }, 3000);
        });
    });
});
```

```
const express = require('express');
const app = express();
app.get('/', function (req, res) {
    res.status(200).json({ name: 'zfpx' });
});
app.listen(8080);
module.exports = app;
```

```
const app = require('../src/app');
const request = require('supertest');
describe('app', function () {
    it('/', function (done) {
        request(app)
            .get('/')
            .expect('Content-Type', /json/)
            .expect('Content-Length', "15")
            .expect(200)
            .end(done);
    });
});
```

3. egg.js

- 约定test目录为存放所有的测试脚本的目录
- 测试脚本文件统一按 ${filename}.test.js 命名，必须以 .test.js 作为文件后缀。

```
{
  "scripts": {
    "test": "egg-bin test"
  }
}
```

```
describe('exec order', () => {
    before(() => console.log(1));
    before(() => console.log(2));
    after(() => console.log(6));
    beforeEach(() => console.log(3));
    afterEach(() => console.log(5));
    it('should work', () => console.log(4));
});
```

```
it('promise 200', () => {
    return app.httpRequest()
        .get('/')
        .expect(200)
});

it('callback 200', done => {
    app.httpRequest()
        .get('/')
        .expect(200, done);
});

it('done 200', done => {
    app.httpRequest()
        .get('/')
        .expect(200)
        .end(done)
});

it('await 200', async () => {
    await app.httpRequest()
        .get('/')
        .expect(200);
});
```

```
const { app, mock, assert } = require('egg-mock/bootstrap');
describe('test/controller/home.test.js', function () {
    describe('GET /', () => {
        it('should 200 and get body', () => {
            return app.httpRequest()
                .get('/')
                .expect(200)
                .expect('hello')
        })

        it('should 200 and get reqeust body', () => {
            app.mockCsrf();
            return app.httpRequest()
                .post('/post')
                .type('form')
                .send({ name: 'zfpx' })
                .expect(200)
                .expect({ name: 'zfpx' })
        })
    });
});
```

Service 相对于 Controller 来说，测试起来会更加简单， 我们只需要先创建一个 ctx，然后通过 ctx.service.${serviceName} 拿到 Service 实例， 然后调用 Service 方法即可。 \app\service\user.js

```
const { Service } = require('egg');
class UserService extends Service {
    async create(user) {
        return await this.ctx.model.User.create(user);
    }
    async get(username) {
        return await this.ctx.model.User.findOne({ username });
    }
}
module.exports = UserService;
```

\test\service\user.test.js

```
const { app, assert } = require('egg-mock/bootstrap');
describe('get name', () => {
    it('create user', async () => {
        const ctx = app.mockContext();
        const doc = await ctx.service.user.create({ username: 'zfpx', password: '123456', email: '1@qq.com' });
        assert(doc);
        assert(doc.username == 'zfpx');
    });
    it('get user', async () => {
        const ctx = app.mockContext();
        const doc = await ctx.service.user.get('zfpx');
        assert(doc);
        assert(doc.username == 'zfpx');
    });
});
```

app\extend\application.js

```
module.exports = {
    get name() {
        return 'app-name';
    }
}
```

\test\extend\application.test.js

```
const { app, assert } = require('egg-mock/bootstrap');
describe('test app', () => {
    it('test app', () => {
        assert(app.name == 'app-name');
    });
});
```

app\extend\context.js

```
module.exports = {
    get isXHR() {
        return this.get('X-Requested-With') == 'XMLHttpRequest';
    }
}
```

\test\extend\context.test.js

```
const { app, assert } = require('egg-mock/bootstrap');
describe('test context', () => {
    it('XHR is true', () => {
        const ctx = app.mockContext({
            headers: {
                "X-Requested-With": "XMLHttpRequest"
            }
        });
        assert(ctx.isXHR == true);
    });

    it('XHR is true', () => {
        const ctx = app.mockContext({
            headers: {
                "X-Requested-With": "SuperAgent"
            }
        });
        assert(ctx.isXHR == false);
    });
});
```

app\extend\request.js

```
module.exports = {
    get isChrome() {
        return this.get('User-Agent').toLowerCase().includes('chrome');
    }
}
```

\test\extend\request.test.js

```
const { app, assert } = require('egg-mock/bootstrap');
describe('test context', () => {
    it('XHR is true', () => {
        const ctx = app.mockContext({
            headers: {
                "X-Requested-With": "XMLHttpRequest"
            }
        });
        assert(ctx.isXHR == true);
    });

    it('XHR is true', () => {
        const ctx = app.mockContext({
            headers: {
                "X-Requested-With": "SuperAgent"
            }
        });
        assert(ctx.isXHR == false);
    });
});
```

app\extend\response.js

```
module.exports = {
    get isSuccess() {
        return this.status == 200;
    }
}
```

\test\extend\response.test.js

```
const { app, assert } = require('egg-mock/bootstrap');
describe('test app', () => {
    it('response isSuccess is true', () => {
        const ctx = app.mockContext();
        ctx.status = 200;
        assert(ctx.response.isSuccess == true);
    });

    it('response isSuccess is false', () => {
        const ctx = app.mockContext();
        ctx.status = 404;
        assert(ctx.response.isSuccess == false);
    });
});
```

app\extend\helper.js

```
module.exports = {
    money(val) {
        const lang = this.ctx.get('Accept-Language');
        if (lang.includes('zh-CN')) {
            return `￥ ${val}`;
        } else {
            return `$ ${val}`;
        }
    }
}
```

\test\extend\helper.test.js

```javascript
const { app, assert } = require('egg-mock/bootstrap');
describe('test helper', () => {
    it('should RMB', () => {
        const ctx = app.mockContext({
            headers: {
                "Accept-Language": "zh-CN"
            }
        });
        assert(ctx.helper.money(100) == '￥ 100');
    });

    it('should Dollar', () => {
        const ctx = app.mockContext();
        assert(ctx.helper.money(100) == '$ 100');
    });
});
```