Promise API

详情查看官方文档：EcmaScript 6 入门

看基本 Promise 代码结构：

```
var fs = require('fs')

// 创建 Promise 容器
// Promise 本身不是一个异步操作，但是 Promise 容器里面通常放一个 异步方法
//      Promise 容器一旦创建，就开始执行里面的代码
//      function 中的 resolve, reject 是形参，想取什么名字，就什么名字
var _promise1 = new Promise(function (resolve, reject) {
    fs.readFile('./data/a.txt', 'utf8', function (err, data) {
        if (err) {
            // 失败了，承诺容器中的任务失败了
            reject(err)
        } else {
            // 承诺容器中的任务成功了
            resolve(data)
        }
    })
})

// _promise1 就是那个 Promise 容器的实例，所以 使用的时候直接 .then() 方法
//   当 _promise1 成功了，然后（.then）,做指定的操作
//   .then 接收的 function 就是 _promise1 容器中的 resolve 函数 ：  成功信息
//   后面的function 接收的就是 _promise1 容器中的 reject 函数   ：  错误信息
_promise1
    .then(function(data) {
        console.log(data)
    },function(err) {
        console.log(err)
    })
```

多任务回调嵌套：

```
var fs = require('fs')


var _promise1 = new Promise(function (resolve, reject) {
    fs.readFile('./data/a.txt', 'utf8', function (err, data) {
        if (err) {
            reject(err)
        } else {
            resolve(data)
        }
    })
```

```
})

var _promise2 = new Promise(function (resolve, reject) {
    fs.readFile('./data/b.txt', 'utf8', function (err, data) {
        if (err) {
            reject(err)
        } else {
            resolve(data)
        }
    })
})

var _promise3 = new Promise(function (resolve, reject) {
    fs.readFile('./data/c.txt', 'utf8', function (err, data) {
        if (err) {
            reject(err)
        } else {
            resolve(data)
        }
    })
})


_promise1
    .then(function (data) {
        console.log(data)

        // 当 _promise1 读取成功的时候
        // 在这里 return 的结果就可以在后面的 then 中的 function 接收到
        // 当你在这 return 123，后面 .then 中 function 中接收的就是 123
        // 当你 return 'hello' 后面接受的就是 hello
        // 没有 return 的话， 后面接受的就是 undefined
        // 上面 return 的都是些没用的，并没有什么卵用
        // 真正有用的是： 我们可以 return 一个 Promise 对象
        // 当 return 一个 Promise 对象的时候， 后续的 then 中的方法的第一个参数会作为
_promise2 的 resolve
        // 第二个方法参数 也就是 reject
        return _promise2
    }, function (err) {
        console.log(err)
    })
    .then(function(data) {
        console.log(data)
        return _promise3
    },function(err) {
        console.log(err)
    })
    .then(function(data) {
        console.log(data)
    },function(err) {
        console.log(err)
    })
```

上面例子重复代码过多，如何封装 Promise API :

```
var fs = require('fs')

function promiseReadFile(filePath) {
    return new Promise(function(resolve,reject) {
        fs.readFile(filePath, 'utf8', function (err, data) {
            if (err) {
                reject(err)
            } else {
                resolve(data)
            }
        })
    })
}

promiseReadFile('./data/a.txt')
    .then(function(data) {
        console.log(data)
        return new promiseReadFile('./data/b.txt')
    })
    .then(function (data) {
        console.log(data)
        return new promiseReadFile('./data/c.txt')
    })
    .then(function (data) {
        console.log(data)
    })
```