# promise

```
promise是一个对象，对象和函数的区别就是对象可以保存状态，函数不可以（闭包除外）
并未剥夺函数return的能力，因此无需层层传递callback，进行回调获取数据
代码风格，容易理解，便于维护
多个异步等待合并便于解决
```

```
new Promise(
  function (resolve, reject) {
    // 一段耗时的异步操作
    resolve('成功') // 数据处理完成
    // reject('失败') // 数据处理出错
  }
).then(
  (res) => {console.log(res)},  // 成功
  (err) => {console.log(err)} // 失败
)

promise有三个状态：
1、pending[待定]初始状态
2、fulfilled[实现]操作成功
3、rejected[被否决]操作失败
当promise状态发生改变，就会触发then()里的响应函数处理后续步骤；
```

构造一个Promise实例需要给Promise构造函数传入一个函数。传入的函数需要有两个形参，两个形参都是function类型的参数。分别是resolve和reject。

·Promise上还有then方法，then 方法就是用来指定Promise 对象的状态改变时确定执行的操作，resolve 时执行第一个函数（onFulfilled），reject时执行第二个函数（onRejected）

·当状态变为resolve时便不能再变为reject，反之同理。

```
function Promise(executor){ //executor执行器
    let self = this;
    self.status = 'pending'; //等待态
    self.value  = undefined; // 表示当前成功的值
    self.reason = undefined; // 表示是失败的值
    function resolve(value){ // 成功的方法

        if(self.status === 'pending'){

            self.status = 'resolved';

            self.value = value;

        }

    }

    function reject(reason){ //失败的方法

        if(self.status === 'pending'){
            self.status = 'rejected';

            self.reason = reason;

        }

    }

    executor(resolve,reject);

}

Promise.prototype.then = function(onFufiled,onRejected){

    let self = this;

    if(self.status === 'resolved'){

        onFufiled(self.value);

    }

    if(self.status === 'rejected'){
        onRejected(self.reason);
    }
}
module.exports = Promise;
```

```
var Promise = (function() {
    function Promise(resolver) {
        if (typeof resolver !== 'function') { //resolver必须是函数
            throw new TypeError('Promise resolver ' + resolver + ' is not a
function')
        }
        if (!(this instanceof Promise)) return new Promise(resolver)

        var self = this //保存this
        self.callbacks = [] //保存onResolve和onReject函数集合
        self.status = 'pending' //当前状态

        function resolve(value) {
            setTimeout(function() { //异步调用
                if (self.status !== 'pending') {
                    return
                }
                self.status = 'resolved' //修改状态
                self.data = value

                for (var i = 0; i < self.callbacks.length; i++) {
                    self.callbacks[i].onResolved(value)
                }
            })
        }
        function reject(reason) {
            setTimeout(function(){ //异步调用
                if (self.status !== 'pending') {
                    return
                }
                self.status = 'rejected' //修改状态
                self.data = reason

                for (var i = 0; i < self.callbacks.length; i++) {
                    self.callbacks[i].onRejected(reason)
                }
            })
        }

        try{
            resolver(resolve, reject) //执行resolver函数
        } catch(e) {
            reject(e)
        }
    }

    function resolvePromise(promise, x, resolve, reject) {
        var then
        var thenCalledOrThrow = false

        if (promise === x) {
            return reject(new TypeError('Chaining cycle detected for promise!'))
        }

        if ((x !== null) && ((typeof x === 'object') || (typeof x ===
'function'))) {
            try {
                then = x.then
                if (typeof then === 'function') {
                    then.call(x, function rs(y) {
                        if (thenCalledOrThrow) return
                        thenCalledOrThrow = true
                        return resolvePromise(promise, y, resolve, reject)
                    }, function rj(r) {
                        if (thenCalledOrThrow) return
                        thenCalledOrThrow = true
                        return reject(r)
                    })
                } else {
                    return resolve(x)
                }
            } catch(e) {
                if (thenCalledOrThrow) return
                thenCalledOrThrow = true
                return reject(e)
            }
        } else {
            return resolve(x)
        }
    }

    Promise.prototype.then = function(onResolved, onRejected) {
        //健壮性处理，处理点击穿透
        onResolved = typeof onResolved === 'function' ? onResolved : function(v)
{return v}
        onRejected = typeof onRejected === 'function' ? onRejected : function(r)
{throw r}
        var self = this
        var promise2

        //promise状态为resolved
        if (self.status === 'resolved') {
            return promise2 = new Promise(function(resolve, reject) {
                setTimeout(function() {
                    try {
                        //调用then方法的onResolved回调
                        var x = onResolved(self.data)
                        //根据x的值修改promise2的状态
                        resolvePromise(promise2, x, resolve, reject)
                    } catch(e) {
                        //promise2状态变为rejected
                        return reject(e)
                    }
                })
            })
        }

        //promise状态为rejected
        if (self.status === 'rejected') {
            return promise2 = new Promise(function(resolve, reject) {
                setTimeout(function() {
                    try {
                        //调用then方法的onReject回调
                        var x = onRejected(self.data)
                        //根据x的值修改promise2的状态
                        resolvePromise(promise2, x, resolve, reject)
                    } catch(e) {
                        //promise2状态变为rejected
                        return reject(e)
                    }
                })
            })
        }

        //promise状态为pending
        //需要等待promise的状态改变
        if (self.status === 'pending') {
            return promise2 = new Promise(function(resolve, reject) {
                self.callbacks.push({
                    onResolved: function(value) {
                        try {
                            //调用then方法的onResolved回调
                            var x = onResolved(value)
                            //根据x的值修改promise2的状态
                            resolvePromise(promise2, x, resolve, reject)
                        } catch(e) {
                            //promise2状态变为rejected
                            return reject(e)
                        }
                    },
                    onRejected: function(reason) {
```

```
                    try {
                        //调用then方法的onResolved回调
                        var x = onRejected(reason)
                        //根据x的值修改promise2的状态
                        resolvePromise(promise2, x, resolve, reject)
                    } catch(e) {
                        //promise2状态变为rejected
                        return reject(e)
                    }
                }
            })
        })
    }
}

//获取当前Promise传递的值
Promise.prototype.valueOf = function() {
    return this.data
}

//由then方法实现catch方法
Promise.prototype.catch = function(onRejected) {
    return this.then(null, onRejected)
}

//finally方法
Promise.prototype.finally = function(fn) {
    return this.then(function(v){
        setTimeout(fn)
        return v
    }, function(r){
        setTimeout(fn)
        throw r
    })
}

Promise.prototype.spread = function(fn, onRejected) {
    return this.then(function(values) {
        return fn.apply(null, values)
    }, onRejected)
}

Promise.prototype.inject = function(fn, onRejected) {
    return this.then(function(v) {
        return fn.apply(null, fn.toString().match(/\((.*?)\)/)
[1].split(',').map(function(key){
            return v[key];
        }))
    }, onRejected)
}

Promise.prototype.delay = function(duration) {
    return this.then(function(value) {
        return new Promise(function(resolve, reject) {
            setTimeout(function() {
                resolve(value)
            }, duration)
        })
        }, function(reason) {
            return new Promise(function(resolve, reject) {
                setTimeout(function() {
                    reject(reason)
                }, duration)
            })
        })
    }

    Promise.all = function(promises) {
        return new Promise(function(resolve, reject) {
            var resolvedCounter = 0
            var promiseNum = promises.length
            var resolvedValues = new Array(promiseNum)
            for (var i = 0; i < promiseNum; i++) {
                (function(i) {
                    Promise.resolve(promises[i]).then(function(value) {
                        resolvedCounter++
                        resolvedValues[i] = value
                        if (resolvedCounter == promiseNum) {
                            return resolve(resolvedValues)
                        }
                    }, function(reason) {
                        return reject(reason)
                    })
                })(i)
            }
        })
    }

    Promise.race = function(promises) {
        return new Promise(function(resolve, reject) {
            for (var i = 0; i < promises.length; i++) {
                Promise.resolve(promises[i]).then(function(value) {
                    return resolve(value)
                }, function(reason) {
                    return reject(reason)
                })
            }
        })
    }

    Promise.resolve = function(value) {
        var promise = new Promise(function(resolve, reject) {
            resolvePromise(promise, value, resolve, reject)
        })
        return promise
    }

    Promise.reject = function(reason) {
        return new Promise(function(resolve, reject) {
            reject(reason)
        })
    }

    Promise.fcall = function(fn){
        // 虽然fn可以接收到上一层then里传来的参数，但是其实是undefined，所以跟没有是一样
的，因为resolve没参数啊
```

```
        return Promise.resolve().then(fn)
    }

    Promise.done = Promise.stop = function(){
        return new Promise(function(){})
    }

    Promise.deferred = Promise.defer = function() {
        var dfd = {}
        dfd.promise = new Promise(function(resolve, reject) {
            dfd.resolve = resolve
            dfd.reject = reject
        })
        return dfd
    }

    try { // CommonJS compliance
        module.exports = Promise
    } catch(e) {}

    return Promise
})()
```