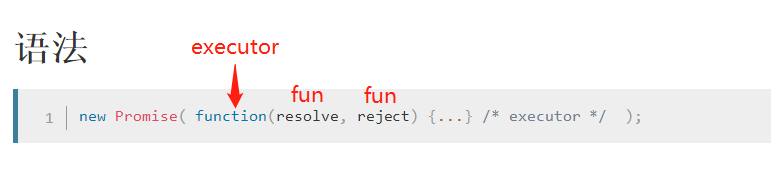
# Promise对象生存手册-入门篇



resolve和reject这对孪生兄弟的作用就是更改Promise对象的状态（fulfilled）（rejected）

触发resolve要手动，触发rejected可以直接抛出一个错误触发。

什么时候调用executor函数？答：Promise构造函数执行时，等价于executor函数。

promise的返回值 和 executor的返回值的区别是：

## 描述

## ****代理能力：Promise**** 对象是一个代理对象（或者说代理一个值），在Promise对象创建时被代理的对象可能是不确定的（pending状态），即不确定执行成功或失败。

**异步能力：Promise** 对象允许你为异步操作的成功和失败分别回调调用(即绑定)相应的处理方法（handlers）。这么一来你就不需要阻塞等待它的返回，所以**Promise** 对象是一个能代表未来出现的结果的对象。

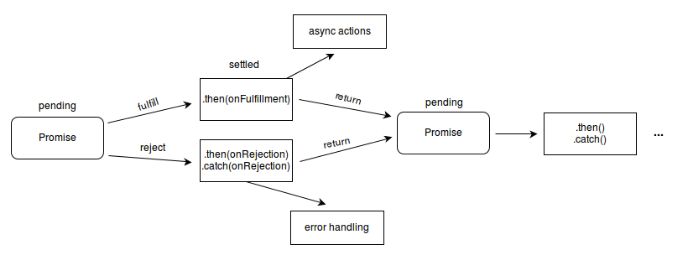
pending 状态的 Promise 对象要么变为fulfilled 状态然后再传递一个相应的状态处理方法（如），要么变为失败状态（rejected）但只会传递失败信息而不是方法。

无论是哪种状态，Promise 对象的 then 方法绑定的处理方法（handlers 回调函数）就会被调用（then方法包含两个参数：onfulfilled 和 onrejected，它们都是 Function 类型。当Promise状态为fulfilled时，调用 then 的 onfulfilled 方法，

Promise then(function onfulfilled ,function onrejected)

当Promise状态为rejected时，调用 then 的 onrejected 方法， 所以在异步操作的完成和绑定处理方法之间不存在竞争）。

由于[Promise.prototype.then](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/then) 和  [Promise.prototype.catch](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/catch) 方法返回的又是一个promise 对象， 所以，从某种意义上实现链式调用。



状态概念解析：resolved状态=Settled状态=fulfilled状态 +rejected状态

## 属性

1. **Promise.length**

**记录着构造器参数的数目，promise中的构造器总是一个（难道还有多个构造函数这种说法？）**

1. **[Promise.prototype](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/prototype)**

Promise 构造器的原型

## 方法

**[Promise.all(iterable)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/all)**

这个方法返回一个新的promise对象。

这个新的promise对象有一个触发成功条件：在iterable参数对象里所有的promise对象都成功时，一旦有任何一个iterable里面的promise对象失败则立即触发该promise对象的失败。这个新的promise对象在触发成功状态以后，成功回调返回什么呢？返回一个数组。

有这么一个数组：包含iterable里所有promise返回值（自己设置，如布尔值）

数组里面的顺序跟iterable的顺序保持一致；Iterable是一个对象，包含着好多promise对象。

如果这个新的promise对象触发了失败状态，它会把iterable里第一个触发失败的promise对象的错误信息作为它的失败错误信息。

Promise.all方法常被用于处理多个promise对象的状态集合。

**[Promise.race(iterable)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/race)**

当iterable参数里的任意一个子promise被成功或失败后，父promise马上也会用子promise的成功返回值或失败详情作为参数调用父promise绑定的相应句柄，并返回父promise对象。

**所以这个race是什么作用？？**

**[Promise.reject(reason)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/reject)**

**reject**返回一个状态为失败的Promise对象，并将给定的失败信息传递给对应的处理方法，传递给谁呢？

**[Promise.resolve(value)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/resolve)**

**resolve**返回一个Promise对象，一个状态由value决定的Promise对象。

Value是promise对象：如果value是thenable(即，带有then方法的对象，也就是，不一定只有promise对象才有then)，那么所决定返回的Promise对象的最终状态由then方法执行决定；

Value不是promise对象：否则的话(该value为空，基本类型或者不带then方法的对象),返回的Promise对象状态为fulfilled，并且将该value传递给对应的then方法。

通常用这个方法来判断是不是promise对象。

## Promise 原型

属性

**Promise.prototype.constructor**

**返回Promise的实例函数，Promise的默认实例函数就是Promise**

方法——主要体现了新旧promise继承机制

**[Promise.prototype.catch(onRejected)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/catch" \o "catch() 方法返回一个Promise，并且处理拒绝的情况。它的行为与调用Promise.prototype.then(undefined, onRejected) 相同。 (事实上, calling obj.catch(onRejected) 内部calls obj.then(undefined, onRejected)).)**

在当前 promise对象中添加一个拒绝(rejection) 回调函数, 再将这个已经添加拒绝回调函数的新的promise返回。所以这时候就出现了旧promise与新promise的关系：

1. 要么是**onRejected回调**函数被调用 新promise的完成结果是resolve(**onRejected()**) 以**onRejected**的返回值来resolve—— “成功调用就hook了resolve,体现了catch的概念”
2. 要么是**onRejected回调**函数没有调用+旧的promise 进入fulfilled：新promise的完成结果是旧的promise的完成结果（即reject()） ——“无用就默认以旧promise reject”

结论：成功就要catch，不成就理会了。

**[Promise.prototype.then(onFulfilled, onRejected)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/then" \o "then() 方法返回一个  Promise 。它最多需要有两个参数：Promise 的成功和失败情况的回调函数。)**

返回以个新promise：

新promise.解决(fulfillment)回调=**onFulfilled**

新promise.拒绝(rejection)回调=**onRejected**

结论：**Promise.prototype.then**直接在老promise基础上定制好它的成功回调、失败回调 作为新的promise返回。

**[Promise.prototype.finally(onFinally)](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise/finally" \o "返回一个设置了 finally 回调函数的Promise对象。)**

返回以个新promise：

**onFinally**回调会在promise运行完毕后被调用，无论当前promise的状态是完成(fulfilled)还是失败(rejected)

结论：**Promise.prototype.then**直接在老promise基础上定制一个只要调用完毕就调用的hook.