# Promise

Promise 对象是一个代理对象（代理一个值），被代理的值在Promise对象创建时可能是未知的。它允许你为异步操作的成功和失败分别绑定相应的处理方法（handlers）。 这让异步方法可以像同步方法那样返回值，但并不是立即返回最终执行结果，而是一个能代表未来出现的结果的promise对象

一个 Promise就3种状态:  
pending: 初始状态，既不是成功，也不是失败状态。  
fulfilled: 意味着操作成功完成。  
rejected: 意味着操作失败。

## 关于promise对象

创建一个promise实例，然后直接打印会报错

<**script**>  
 **var** pro = **new** Promise();  
 console.log(pro);  
</**script**>

原因就是当创建一个promise实例的时候，需要在里面放一个函数：

<**script**>  
 **var *pro*** = **new *Promise***(() => {});  
 ***console***.log(***pro***); *// 打印promise实例*</**script**>

----------------------

如果在函数里面，写一些代码，会发现内部代码立即执行了

<**script**>  
 **var *pro*** = **new *Promise***(() => {  
 ***console***.log(**"实例化..."**); *// 立即执行* });  
</**script**>

----------------

在promise的函数中传入两个参数：resolve和reject ，表示成功的状态和失败的状态：

<**script**>  
 **var *pro*** = **new *Promise***((resolve, reject) => {  
 ***console***.log(**"实例化..."**);  
 *// resolve() // "fulfilled"* reject() *// "rejected"* });  
 ***console***.log(***pro***);  
</**script**>

------------------

一旦进入到成功或失败的状态，就无法再改变状态了：

<**script**>  
 **var *pro*** = **new *Promise***((resolve, reject) => {  
 ***console***.log(**"实例化..."**);  
 resolve() *// "fulfilled" 进入到成功状态，reject就没有用了* reject() *// 不会执行* });  
 ***console***.log(***pro***);  
</**script**>

----------------------

当异步代码执行成功时，我们应该调用resolve(...), resolve会把实参传给then；   
当异步代码失败时应该调用reject(...) ， reject会把实参传给catch

<**script**>  
 **var *pro*** = **new *Promise***((resolve, reject) => {  
 **var** xhr = **new *XMLHttpRequest***();  
 xhr.**onreadystatechange** = **function** () {  
 **if** (xhr.**readyState** == 4) {  
 **if** (xhr.**status** == 200) {  
 resolve(**'欧耶，成功了:'** + xhr.**responseText**)  
 } **else** {  
 reject(**'哥们，失败了:'** + xhr.**statusText**)  
 }  
 }  
 }  
 xhr.open(**'get'**, **'a.txt'**, **true**);  
 xhr.send(**null**);  
 });  
 ***pro***.then((successMessage) => {  
 *// successMessage的值是上面调用resolve(...)方法传入的实参  
 // successMessage参数不一定非要是字符串类型* ***console***.log(**'then:'** + successMessage);  
 });  
 ***pro***.catch((error) => {  
 ***console***.log(error);  
 })  
</**script**>

---------------------

then方法执行完之后有一个返回值，返回的是一个新的Promise对象，所以可以使用链式方法继续调用then方法：

<**script**>  
 **var *pro1*** = **new *Promise***((resolve, reject) => {  
 resolve(**'成功1'**)  
 });  
 **var *pro2*** = ***pro1***.then((value) => {  
 ***console***.log(value); *// 成功1  
 // return 111* **return new *Promise***((resolve, reject) => {  
 *// resolve('成功2')* reject(**"失败2"**)  
 });  
 });  
 ***pro2***.then((value) => {  
 ***console***.log(**'then2：'** + value); *// then2：成功2* });  
 ***pro2***.catch((error) => {  
 ***console***.log(error);  
 });  
 ***console***.log(***pro2***);  
  
</**script**>

--------------------

catch 除了可以接收reject信息，还可以拦截错误信息，捕获前面链式调用中的错误信息:

<**script**>  
 **var *pro1*** = **new *Promise***((resolve, reject) => {  
 resolve(**'成功1'**)  
 });  
 **var *pro2*** = ***pro1***.then((value) => {  
 ***console***.log(1);  
 })  
 .then((value) => {  
 ***console***.log(2);  
 a *// 这里写个a就报错了* })  
 .then((value) => {  
 ***console***.log(3);  
 })  
 .catch((err) => {  
 ***console***.log(err);  
 })  
 .finally(() => {  
 *// 不过是then还是catch， 我都会执行* ***console***.log(**'执行完了！'**);  
 })  
  
 ***console***.log(**'啦啦啦~'**);  
</**script**>

finally方法不管前面是走的then还是catch 都会执行。