Promise

Promise有两大特点：

1）对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。

（2）一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：**从pending变为fulfilled和从pending变为rejected**。**只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果，这时就称为 resolved（已定型）**。如果改变已经发生了，你再对Promise对象添加回调函数，也会立即得到这个结果。这与事件（Event）完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

一．

简单例子1：

**var *p*** =**new** *Promise*((resolve,reject) => {  
 *//做一些异步操作* ***window***.setTimeout(()=> {  
 ***console***.log(**'执行完成'**);  
 resolve(**'I am resolve'**); *//会被p.then catch到* },2000);  
});  
  
***p***.then(response => {  
 ***console***.log(response); *//I am resolve*});

简单例子2：

**var *p*** = **new** *Promise*((resolve, reject) => {  
 ***window***.setTimeout(() => {  
 ***console***.log(**'执行完成'**);  
 reject(**'I am reject'**); *//可以被then的第二个参数catch到* }, 2000);  
});  
  
***p***.then((response) => {  
 ***console***.log(**'执行不到'**);  
 ***console***.log(response);  
}, (response) => {  
 ***console***.log(**'执行到了'**);  
 ***console***.log(response); *// I am reject*});

简单例子3：

**var *p*** =**new** *Promise*((resolve,reject) => {  
 *//做一些异步操作* ***window***.setTimeout(()=> {  
 ***console***.log(**'执行完成'**);  
 reject(**'I am reject'**); *//可以被p.catch catch到* },2000);  
});  
  
***p***.catch(error => {  
 ***console***.log(error); *//I am reject*});

**简单来说：resolve会被then的第一个回调函数接收到；reject和异常(throw** **new Error()等)会被then的第二个回调函数或则catch方法接收到，如果then的第二个函数和catch方法都写了，那么只会执行then的第二个回调函数（除非then里面报错了，就会在catch里面执行）。catch还有一个作用就是抓取异常，包括then方法里面的异常， then方法指定的回调函数，如果运行中抛出错误，也会被catch方法捕获。**

二．

**Promise的出现就是为了解决链式地狱的问题。**

**例子：**

**function** *runAsync*() {  
 **return new** *Promise*((resolve,reject) => {  
 ***window***.setTimeout(() => {  
 resolve(**'我是第一个'**);  
 }, 2000);  
 });  
}  
  
**function** *runAsync2*() {  
 **return new** *Promise*((resolve,reject) => {  
 ***window***.setTimeout(() => {  
 resolve(**'我是第二个'**);  
 }, 2000);  
 });  
}  
  
**function** *runAsync3*() {  
 **return new** *Promise*((resolve,reject) => {  
 ***window***.setTimeout(() => {  
 resolve(**'我是第三个'**);  
 }, 2000);  
 });  
}  
  
*runAsync*().then(response => {  
 ***console***.log(response); *//我是第一个* **return** *runAsync2*();  
}).then(response => { *//response由上个then里面的then而来* ***console***.log(response); *//我是第二个* **return** *runAsync3*();  
}).then(response => {  
 ***console***.log(response); *//我是第三个*});

**then里面的return不一定返回promise对象，也可以直接是数据，例子：**

*runAsync*().then(response => {  
 ***console***.log(response); *//我是第一个* **return** *runAsync2*();  
}).then(response => { *//response由上个then里面的then而来* ***console***.log(response); *//我是第二个* **return** 1111; ***//也可以直接返回数据***}).then(response => {  
 ***console***.log(response); *//1111*});

三 **Promise.prototype.catch()**

catch方法是.then(null, rejection)或 .then(undefined, rejection)的别名，用于指定发生错误时的回调函数。catch不同于then的第二个回调函数，catch能够抓取then的回调函数的运行中抛出的错误，同时catch具有冒泡性质，错误会一直向后传递，直到被捕获为止，也就是说：错误总是会被下一个catch语句捕获到。例子：

**function** *test1*() {  
 **return new** *Promise*((resolve, reject) => {  
 **throw new *Error***(**'error message'**);

***console***.log(**'我是不会被执行到的'**); *//不会执行*  
 });  
}  
*test1*().catch(err => {  
 ***console***.log(err);  
});

**需要注意的是，一旦有抛出异常，异常后面的代码(属于统一区块)就不会执行了。**

**reject(new Error())和 throw new Error()**效果很类型，但是他们有他们的不同， reject(new Error())后面的代码还能够执行；但是throw new Error()后面的代码就不会执行了。

例子

**function** *test*() {  
 **return new** *Promise*((resolve, reject) => {  
 reject(**new *Error***(**'我是error'**));  
 ***console***.log(**'我还能被执行'**);  
 });  
}  
*test*().then(**null**, err => {  
 ***console***.log(err);  
});

vs

**function** *test*(){  
 **return new** *Promise*((resolve, reject) => {  
 **throw new *Error***(**'我是error'**);  
 ***console***.log(**'我不会被执行'**);  
 });  
}  
*test*().then(**null**, err => {  
 ***console***.log(err);  
});

**Promise 的状态一旦改变，就永久保持该状态，不会再变了。**

例子：

**function** *test*() {  
 **return new** *Promise*((resolve, reject) => {  
 resolve(**'aa'**);  
 ***console***.log(**'会被执行1'**);  
 reject(**'bb'**);  
 ***console***.log(**'会被执行2'**);  
 reject(**new *Error***(**'error'**));  
 ***console***.log(**'会被执行3'**);  
 **throw new *Error***(**'error'**);  
 ***console***.log(**'不会被执行'**);  
 });  
}  
*test*().then(response => {  
 ***console***.log(**'response会被执行'**);  
 ***console***.log(response); *//aa 因为Promise 的状态一旦改变，就永久保持该状态，不会再变了。*}, error => {  
 ***console***.log(**'error不会被执行'**);  
 ***console***.log(error);  
});

**Promise 对象的错误具有“冒泡”性质，会一直向后传递，直到被捕获为止。也就是说，错误总是会被下一个catch语句捕获。**

例子：

**function** *test*() {  
 **return new** *Promise*((resolve,reject) => {  
 resolve(**'jiejue'**);  
 });  
}  
*test*().then(response => {  
 ***console***.log(response);  
 **return** 1;  
}).then(response => {  
 ***console***.log(response);  
 **throw new *Error***(**'error'**); //被后面的catch方法捕获  
}).catch(err => {  
 ***console***.log(err);  
});

**一般来说，不要在then方法里面定义 Reject 状态的回调函数（即then的第二个参数），总是使用catch方法。**

Promise.all(promiseArr).then(results => {

});

例子：

***Promise***.all([*test1*(), *test2*(), *test2*()]).then(**function**(results) {  
 ***console***.log(results);  
});

用Promise.all来执行，all接收一个数组参数，**数组里面的值都是Promise对象**。这样，三个异步操作的并行执行的，等到它们都执行完后才会进到then里面。那么，三个异步操作返回的数据都在then里面，all会把所有异步操作的结果放进一个数组中传给then，就是上面的results。