Promise

概念

Promise对象用于一个异步操作的最终完成（包括成功和失败）及结果值的表示。
简单说，就是处理异步请求的。
之所以叫做Promise，就是我承诺，如果成功则怎么处理，失败则怎么处理。

```
// 语法
new Promise(
    /* executor */
    function(resolve, reject) {...}
);
```

executor

- executor 是一个带有 resolve 和 reject **两个参数**的**函数** 。
- executor 函数在Promise构造函数执行时同步执行，被传递resolve和reject函数（executor 函数在Promise构造函数返回新建对象前被调用）。
- executor 内部通常会执行一些异步操作，一旦完成，可以调用resolve函数来将promise状态改成fulfilled即完成，或者在发生错误时将它的状态改为rejected即失败。
- 如果在executor函数中抛出一个错误，那么该promise 状态为rejected。executor函数的返回值被忽略
- executor中，resolve或reject只能执行其中一个函数

Promise的状态

- pending: 初始状态，不是成功或失败状态。
- fulfilled: 意味着操作成功完成。
- rejected: 意味着操作失败。

Promise.then(onFulfilled, onRejected)
参数是2个函数，根据Promise的状态来调用不同的函数，fulfilled走onFulfilled函数，rejected走onRejected函数。
then的返回值是一个新的promise对象。调用任何一个参数后，其返回值会被新的promise对象来resolve，向后传递。

```
// 简单使用
var myPromise = new Promise((resolve, reject) => {
    resolve('hello'); // 执行，置状态为fulfilled
    console.log('~~~~~~~~~~~~~~~~');
    reject('world'); // 永远执行不到
});
```

```
console.log(myPromise);

myPromise.then(
    /*如果成功则显示结果*/
    (value) => console.log(1, myPromise, value),
    /*如果失败则显示原因*/
    (reason) => console.log(2, myPromise, reason)
);
```

catch(onRejected)

为当前Promise对象添加一个拒绝回调，返回一个新的Promise对象。onRejected函数调用其返回值会被新的Promise对象用来resolve。

```
var myPromise = new Promise((resolve, reject) => {
    //resolve('hello'); // 执行，置状态为fulfilled
    console.log('~~~~~~~~~~~~~~~~');
    reject('world'); // 可以执行了
});

console.log(myPromise);

// 链式处理
myPromise.then(
    /*如果成功则显示结果*/
    (value) => console.log(1, myPromise, value),
    /*如果失败则显示原因*/
    (reason) => console.log(2, myPromise, reason)
).then(
    function (v) {
        console.log(2.5, v);
        return Promise.reject(v + '***') //
    }
).catch(reason => {
    console.log(3, reason);
    return Promise.resolve(reason);
})
```


异步实例

```
function runAsync() {
    return new Promise(function(resolve, reject){
        // 异步操作
        setTimeout(function(){
```

```
            console.log('do sth...');
            resolve('ok...');
        }, 3000);
    });
}

// 调用
runAsync().then(value => {
    console.log(value);
    return Promise.reject(value + '*');
}).catch(reason => {
    console.log(reason);
    return Promise.resolve(reason + '*');
}).then(value => {
    console.log(value);
    console.log('END');
});

console.log('~~~~~~ FIN ~~~~~~~')
```