JavaScript Promise 对象

ECMAscript 6 原生提供了 Promise 对象。

Promise 对象代表了未来将要发生的事件，用来传递异步操作的消息。

异步编程

- fs 文件操作

```
require('fs').readFile('./index.html', (err,data)=>{})
```

- 数据库操作
- AJAX

```
  $.get('/server', (data)=>{})
```

- 定时器

```
setTimeout(()=>{}, 2000);
```

Promise 的状态特点

实例对象中的一个属性 『PromiseState』

- pending 初始状态，不是成功或失败状态。
- resolved / fullfilled 成功
- rejected 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。

- ### Promise 创建构造函数

  要想创建一个 promise 对象、可以使用 new 来调用 Promise 的构造器来进行实例化。

  下面是创建 promise 的步骤：

```
var promise = new Promise(function(resolve, reject) {
    // 异步处理
    // 处理结束后、调用resolve 或 reject
});
```

  Promise 构造函数包含一个参数和一个带有 resolve（解析）和 reject（拒绝）两个参数的回调。在回调中执行一些操作（例如异步），如果一切都正常，则调用 resolve，否则调用 reject。

```
var myFirstPromise = new Promise(function(resolve, reject){
    //当异步代码执行成功时，我们才会调用resolve(...), 当异步代码失败时就会调用
reject(...)
    //在本例中，我们使用setTimeout(...)来模拟异步代码，实际编码时可能是XHR请求或是
HTML5的一些API方法.
    setTimeout(function(){
        resolve("成功!"); //代码正常执行!
    }, 250);
});

myFirstPromise.then(function(successMessage){
    //successMessage的值是上面调用resolve(...)方法传入的值.
    //successMessage参数不一定非要是字符串类型，这里只是举个例子
    document.write("Yay! " + successMessage);
});
```

对于已经实例化过的 promise 对象可以调用 promise.then() 方法，传递 resolve 和 reject 方法作为回调。

promise.then() 是 promise 最为常用的方法。

```
promise.then(onFulfilled, onRejected)
```

promise简化了对error的处理，上面的代码我们也可以这样写：

```
promise.then(onFulfilled).catch(onRejected)
```

封装axios

```
export const get = (url,params)=>{
    params = params || {};
    return new Promise((resolve,reject)=>{
        // axiso 自带 get 和 post 方法
        $http.get(url,{
            params,
        }).then(res=>{
            if(res.data.status===0){
                resolve(res.data);
            }else{
                alert(res.data.msg)
            }
        }).catch(error=>{
            alert('网络异常');
        })
    })
}
export const post = (url,params)=>{
    params = params || {};
    return new Promise((resolve,reject)=>{
        $http.post(url,params).then(res=>{
            if(res.data.status===0){
                resolve(res.data);
            }else{
```

```
                alert(res.data.msg);
            }
        }).catch(error=>{
            alert('网络异常');
        })
    })
}
```

async

async 是 ES7 才有的与异步操作有关的关键字，和 Promise 有很大关联的。

语法

```
async function name([param[, param[, ... param]]]) { statements }
```

- name: 函数名称。
- param: 要传递给函数的参数的名称。
- statements: 函数体语句。

async 函数返回一个 Promise 对象，可以使用 then 方法添加回调函数。

```
async function helloAsync(){
    return "helloAsync";
  }

console.log(helloAsync())  // Promise {<resolved>: "helloAsync"}

helloAsync().then(v=>{
   console.log(v);         // helloAsync
})
```

async 函数中可能会有 await 表达式，async 函数执行时，如果遇到 await 就会先暂停执行，等到触发的异步操作完成后，恢复 async 函数的执行并返回解析值。

await 关键字仅在 async function 中有效。如果在 async function 函数体外使用 await，你只会得到一个语法错误。

```
function testAwait(){
   return new Promise((resolve) => {
       setTimeout(function(){
          console.log("testAwait");
          resolve();
       }, 1000);
   });
}

async function helloAsync(){
   await testAwait();
   console.log("helloAsync");
 }
helloAsync();
```

await

await 操作符用于等待一个 Promise 对象, 它只能在异步函数 async function 内部使用。

返回值

返回 Promise 对象的处理结果。如果等待的不是 Promise 对象，则返回该值本身。

如果一个 Promise 被传递给一个 await 操作符，await 将等待 Promise 正常处理完成并返回其处理结果。

```
function testAwait(){
   console.log("testAwait");
}
async function helloAsync(){
   await testAwait();
   console.log("helloAsync");
}
helloAsync();
```