一.promise中resolve的使用

```
class Service{
	constructor() {

	}
	getUser() {
		const p = new Promise(function(resolve, reject){
			setTimeout(function(){
				resolve({name:"小明", age:30});
			}, 5000)
		});
		return p;
	}
}
//当上面的resolve();函数执行后,下面的p.then(函数)中的函数才会被执行。

class Controller {
	constructor() {
		this.s1 = new Service();
	}
	showUser() {
		let p = this.s1.getUser();
		p.then(function(v){
			console.log(v);
		})
	}
}

let c1 = new Controller();
c1.showUser();
console.log("得到User")
```

执行结果为

```
得到User
new_file.html:30 {name: "小明", age: 30}
```

二.promise中reject的使用

```
class Service{
	constructor() {
	}
	getUser() {
		const p = new Promise(function(resolve, reject){
			setTimeout(function(){
				const num = Math.ceil(Math.random()*20)
				if(num<10) {
					//触发p.then中第1个参数函数的执行,代表成功
```

```
                                resolve({state:1, name:'小明', age:num});
                            }else {
                                //触发p.then中第2个参数函数参数的执行,代表失败
                                reject("数据有错");
                            }
                        })
                    });
                    return p;
                }
            }

            class Controller {
                constructor() {
                    this.s1 = new Service();
                }
                showUser() {
                    let p = this.s1.getUser();
                    p.then(
                        //由前面的resolve()函数触发,
                        function(data) {
                            console.log(data);
                        },
                        //由前面的reject()函数触发
                        function(reason) {
                            console.log(reason);
                        }
                    );
                }
            }
            let c1 = new Controller();
            c1.showUser();
            console.log("得到User")
```

三. then和catch

可以代替在then中使用两个函数的功能

```
let p1 = new Promise(function(resolve, reject){
    setTimeout(function(){
        console.log("异步执行完毕");
        const num = Math.floor(Math.random()*10);
        if(num%2==0) {
            resolve("偶数"+num+" 正确");
        }else {
            reject("奇数"+num+"错误");
        }
    },2000);
```

```
});
p1.then(function(data){//当上面执行resolve函数时加调
	console.log(data);
}).catch(function(err){//当上面执行reject函数时回调
	console.log(err);
});
```

四. then、catch、finally

finally在最后总会执行。

```
let p1 = new Promise(function(resolve, reject){
	setTimeout(function(){
		console.log("异步执行完毕");
		const num = Math.floor(Math.random()*10);
		if(num%2==0) {
			resolve("偶数"+num+" 正确");
		}else {
			reject("奇数"+num+"错误");
		}
	},2000);
});
p1.then(function(data){
	console.log(data);
}).catch(function(err){
	console.log(err);
}).finally(function(){
	console.log("最后执行....");
});

/**
异步执行完毕
奇数5错误
最后执行....
*/
```

五、then、finally

finally在最后始终会执行

```
let p1 = new Promise(function(resolve, reject){
        setTimeout(function(){
                console.log("异步执行完毕");
                const num = Math.floor(Math.random()*10);
                resolve("数字为"+num);
        })
},2000);

p1.then(function(data){
        console.log(data);
}).finally(function(){
        console.log("最后执行....");
});
```

六、Promise.all()

Promise.all()方法用于将多个 Promise 实例，包装成一个新的 Promise 实例。

```
const p = Promise.all([p1, p2, p3]);
```

p的状态由p1、p2、p3决定，分成两种情况。

（1）只有p1、p2、p3的状态都变成fulfilled，p的状态才会变成fulfilled，此时p1、p2、p3的返回值组成一个数组，传递给p的回调函数。

（2）只要p1、p2、p3之中有一个被rejected，p的状态就变成rejected，此时第一个被reject的实例的返回值，会传递给p的回调函数。

```
<script type="text/javascript">

function getPromiseArray() {
        let promises = [];
```

```
        let promises = [];
        for(let i=0; i<3; i++) {
            const p = new Promise(function(resolve,reject){
                let num = Math.floor(Math.random()*10);

                if(num>8) {
                    reject("有错"+num);
                }else {
                    resolve("对了"+num);
                }
            });
            promises.push(p)
        }
        return promises;
}

const p = Promise.all(getPromiseArray());
p.then(function(data){//接收到all中参数的所有promise的resolve函数返回的结果的数组
        console.log(data);
}).catch(function(error){
        console.log(error)
}).finally(function(){
        console.log("结束");
});
</script>
```

七、

函数使用async修饰，该函数中可以使用await修改一个函数调用语法，被调用的函数可以是一个Promise的值。.....

```
        function testAwait() {
                    return new Promise((resolve, reject)=>{
                        setTimeout(()=>{
                            console.log("返回一个promise对象");
                           resolve("返回数据")
                        }, 2000)
                    });
                }
                async function helloAsync() {
                    var a = await testAwait();
                    console.log(a);
                    console.log("helloAsync");
                }

                helloAsync();
```