20.Promise 状态特点

学习要点：

1.状态特点

2.更多方法

本节课我们来开始学习 ES6 新增的 Promise 异步通信方案的状态特点

一. 状态特点

1. 回顾上一节：Promise 解决了异步多层回调混乱，且执行顺序的问题；
2. 本节课，了解一下 Promise 对象异步操作的三种状态：

   (1) .Pending(进行中)

   (2) .Fulfilled(已成功)

   (3) .Rejected(已失败)

3. 当异步操作执行后，它得到的结果来决定其状态，其它任何操作都无法改变；
4. Promise 状态只有两种运行方式：从 Pending 到 Fulfilled 或 Rejected；
5. 而当状态已经固定后，此时就变成 Resolved(已完成)。关键字详解：请搜索；

   pending -> resolve 方法 -> fulfilled -> resolved

   pending -> reject 方法 -> rejected -> resolved

   PS：测试当前状态，在浏览器环境下比较直观直接：console.log(p1)，在不同阶段执行；

二. 更多方法

1. 上一节课，我们使用了三组 Promise 实例完成三段异步的排序输出问题；
2. Promise 提供了一个 all()方法，可以简化多个实例调用输出排序；

```
//p1,p2,p3 是三个 Promise 实例，数组元素顺序即输出顺序
let p = Promise.all([p1, p2, p3]);

//将三个 Promise 实例的回调组合成数组输出
p.then(value => {
    console.log(value);
});
```

   PS：虽然 p1,p2,p3 都是异步操作，但最终要等待所有异步完成，才可以输出；

   PS：只要 p1,p2,p3 中有一个出现了 Rejected，则会执行失败回调；

3. Promise 提供了一个 race()方法，只输出第一个改变状态的实例；

```
//p1,p2,p3 只要有一个改变状态，即回调
let p = Promise.race([p1, p2, p3]);

//所以，这里只输出 p2
p.then(value => {
```

```
    console.log(value);
});
```

4. Promise 提供了 resolve()和 reject()，直接返回一个成功或失败的实例；

```
//直接返回成功或失败的 Promise 实例
let ps = Promise.resolve('成功');
let pj = Promise.reject('失败');

ps.then(value => {
    console.log(value);
    return pj;
}).catch(reason => {
    console.log(reason);
})

//等价于
new Promise(resolve => resolve('成功'));

//最常用的场景，类型一致性
function getP() {
    if (false) {
        return new Promise(resolve => {
            resolve('异步成功');
        })
    } else {
        return 0;       //强制类型一致保证程序正确性 Promise.resolve(0)
    }
}

getP().then(value => {
    console.log(value);
});
```