

ES6 promise

`promise` 主要是实现异步编程，他事先定义了两个回调函数，来分别处理两种结果对应的处理方式：

`promise` 的结构一般如下

```
let pro=new Promise(function(resolve,reject){  
    /*  
     *  
     * 待处理的数据  
     *  
     */  
    let boo;      //定义处理的逻辑  
    if(boo==true){  
        resolve('成功时返回的数据')  
    }else{  
        reject('失败返回的数据')  
    }  
})
```

`promise` 定义以后就会立刻执行,且结果不会再作改变

```
var pro=new Promise(function(resolve,reject){  
    let ran=Math.random(0,1)*100;  
    if(ran>50){  
        console.log('大于五');  
        resolve(Math.round(ran))  
    }else{  
        console.log('小于5');  
        reject(Math.round(ran))  
    }  
})
```

```

var pro=new Promise(function(resolve,reject){
  let ran=Math.random(0,1)*100;
  if(ran>50){
    console.log('大于五');
    resolve(Math.round(ran))
  }else{
    console.log('小于5');
    reject(Math.round(ran))
  }
})

```

大于五
undefined

成功后调用resolve方法: 64就是方法返回的值

```

pro
▶ Promise {<resolved>: 64}
pro
▶ Promise {<resolved>: 64}
pro
▶ Promise {<resolved>: 64}

```

然后就可以向下面这种方法来链式调用

```

> pro
< ▶ Promise {<resolved>: 64} ⓘ
  ▶ __proto__: Promise
    [[PromiseStatus]]: "resolved"
    [[PromiseValue]]: 64
> pro.then(res=>console.log(res)).catch(err=>console.log(err))
  64
                                         VM16059:1
< ▶ Promise {<resolved>: undefined}
> pro.then(res=>console.log(res)).then(()=>console.log('没有要执行的事务了')).catch(err=>console.log(err))
  64
                                         VM16074:1
  没有要执行的事务了
< ▶ Promise {<resolved>: undefined} ⓘ
  ▶ __proto__: Promise
    [[PromiseStatus]]: "resolved"
    [[PromiseValue]]: undefined
                                         VM16074:1

```

很多情况下，`promise` 的返回结果都是动态的，所以我们需要将包装一下就可以实现了，这也是最常用的使用方法：

```

function ranPro(){
  let ran=Math.random(0,1)*100;//用户逻辑部分移到外
部
  return new Promise(function(resolve,reject){
    //let ran=Math.random(0,1)*100;
    if(ran>50){
      console.log('大于五');
      resolve(Math.round(ran))
    }else{
      console.log('小于5');
    }
  })
}

```

```
        reject(Math.round(ran))
    }
})
}
```

```
> function ranPro(){
  let ran=Math.random(0,1)*100;//用户逻辑部分移到外部
  return new Promise(function(resolve,reject){
  //let ran=Math.random(0,1)*100;
  if(ran>50){
    console.log('大于五');
    resolve(Math.round(ran))
  }else{
    console.log('小于5');
    reject(Math.round(ran))
  }
})
}
undefined
> ranPro()
小于5
结果每次就变
为动态
> ► Promise {<rejected>: 44}
> ► Uncaught (in promise) 44
> ranPro()
小于5
> ► Promise {<rejected>: 40}
> ► Uncaught (in promise) 40
```

把 `promise` 对象再返回给函数，这个函数结果也变成 `promise` 对象

```
ranPro().then(res=>console.log(res)).then(()=>console.log('返回函数也可以链式调用')).catch(err=>console.log(err))
大于五
55
返回函数也可以链式调用
► Promise {<resolved>: undefined}
```

链式调用的传值dai