Promise

什么是Promise

Promise 是异步编程的一种解决方案，ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象

Promise用处

异步执行

传统的方式——回调

```
let img = new Image();

img.src = 'src';

img.onload = () => {
    //执行回调
};
```

```
setTimeout(() => {
    //
}, 500)
```

a -> b -> c -> d

a, b, c, d -> e　　loading图片

Promise更强大

Promise基本用法和特点

1. Promise建立后立即执行

```
const p = new Promise(() => {
    console.log(1);
});

//1
```

Promise基本用法和特点

2. 有三种状态：
   pending（进行中）、
   fulfilled（已成功)、
   rejected（已失败）。

```
const p = new Promise((resolve, reject) => {
    if (3 > 2) {
        resolve(true); //pending -> fulfilled
    } else {
        reject(false); //pending -> rejected
    }
});


p
.then((mes) => {
    console.log(mes) //true
}, (mes) => {

})
```

Promise实例生成以后，可以用then方法
分别指定resolved状态和rejected状态的回调函数。

Promise基本用法和特点

3.对象的状态不受外界影响。（异步操作的结果，可以决定当前是哪一种状态）

```
const p = new Promise((resolve, reject) => {
    let img = new Image();

    img.onload = () => {
        resolve();
    };

    img.onerror = () => {
        reject();
    };
});
```

Promise基本用法和特点

4.一旦状态改变，就不会再变，任何时候都可以得到这个结果。

```
const p = new Promise((resolve, reject) => {
    resolve('成功');
    throw new Error('失败');
});



p
.then((mes) => {
    console.log(mes) //成功
}, (mes) => {

})
```

```
const p = new Promise((resolve, reject) => {
    resolve('成功');
});



document.onclick = () => {
    p
    .then((mes) => {
        console.log(mes) //成功
    }, (mes) => {
        console.log(mes)
    })

}
```

Promise.prototype.catch

```
const p = new Promise((resolve, reject) => {
    throw new Error('失败');
});



p
.then((mes) => {
    console.log(mes);
}, (mes) => {
    console.log(mes);//失败
})
```

```
const p = new Promise((resolve, reject) => {
    throw new Error('失败');
});



p
.then((mes) => {
    console.log(mes);
})
.catch((mes) => {
    console.log(mes);//失败
})
```

```
const p = new Promise((resolve, reject) => {
    resolve('成功');
});



p
.then((mes) => {
    console.log(mes);//成功
    throw new Error('失败');
})
.catch((mes) => {
    console.log(mes);//失败
})
```

Promise.prototype.finally

```
const p = new Promise((resolve, reject) => {
    throw new Error('图片加载失败');

});

p
.then((mes) => {
    console.log(mes);//图片加载成功
})
.catch((mes) => {
    console.log(mes);//图片加载失败
})
.finally(() => {//不接受参数

});
```

Promise.prototype.all

```
const p0 = new Promise((resolve, reject) => {
    resolve('resolve 0');

});
const p1 = new Promise((resolve, reject) => {
    resolve('resolve 1');

});
const p2 = new Promise((resolve, reject) => {
    resolve('resolve 2');

});

const p = Promise.all([p0, p1, p2]);


p
.then((mes) => {
    console.log(mes);//["resolve 0", "resolve 1", "resolve 2"]
})
.catch((mes) => {
    console.log(mes);
})
.finally(() => {

});
```

Promise.prototype.all

```
let loadingFun = (arr) => {
    let loadNum = 0;
    let loadImg = (url) => {
        return new Promise((resolve, reject) => {
            let img = new Image();
            img.onload = resolve;
            img.onerror = reject;
            img.src = url;
        });
    }


    let pArr = [];
    arr.forEach(url => {
        let p = loadImg(url)
            .then(() => {
                loadNum += 1;
                document.querySelector('.load-num').innerHTML(Math.floor(loadNum / arr.length * 100) + "%");
            })
            .catch(() => {

            });
        pArr.push(p);
    });
    //全部加载结束

    Promise.all(pArr)
        .then(() => {

        });
}

loadingFun(imgArr);
```

Promise.prototype.race

```
const p0 = new Promise((resolve, reject) => {
    resolve('成功');
});
const p1 = new Promise((resolve, reject) => {
    resolve('成功');
});
const p2 = new Promise((resolve, reject) => {
    reject('失败');
});

const p = Promise.race([p0, p1, p2]);



p
.then((mes) => {
    console.log(mes);//成功

})
.catch((mes) => {
    console.log(mes);
})
```

Promise.prototype.resolve

```
//参数是一个 Promise 实例
const l = new Promise((resolve, reject) => {
    resolve('resolve l');

});

const p = Promise.resolve(l);


p
.then((mes) => {
    console.log(mes); // resolve l
})
.catch((mes) => {
    console.log(mes);
})
.finally(() => {

});
```

```
//参数是一个 thenable 对象
let thenable = {
    then(resolve, reject) {
        resolve('resolve thenable');
    }
}

const p = Promise.resolve(thenable);


p
.then((mes) => {
    console.log(mes); // resolve thenable
})
.catch((mes) => {
    console.log(mes);
})
.finally(() => {

});
```

```
//参数不是具有then方法的对象，或不是对象
const p = Promise.resolve('hello');


p
.then((mes) => {
    console.log(mes); // hello
})
.catch((mes) => {
    console.log(mes);
})
.finally(() => {

});
```

end