JavaScript进阶（ES6）

---Promise与异步编程

河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

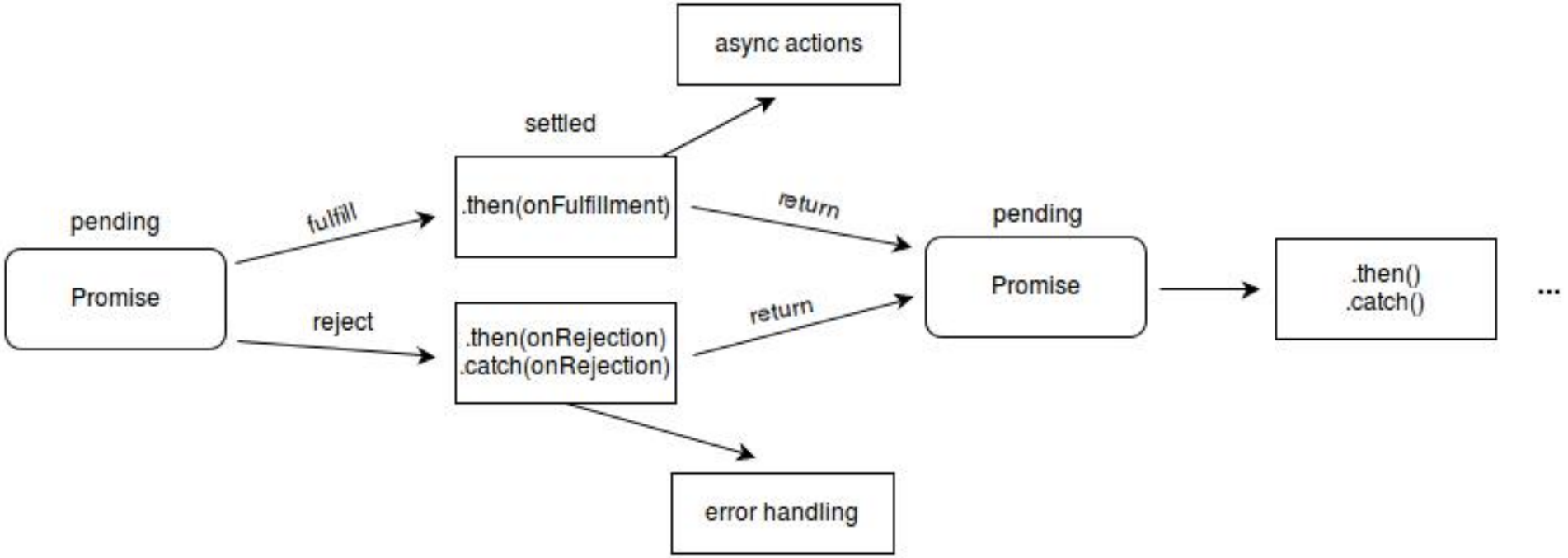
- Promise 概念及语法
- Promise 原型方法及静态方法
- Promise 综合案例

Promise概念及语法

- JS异步几种形式
  - 回调函数（简单容易理解，但耦合太紧密、多层回调时流程混乱不利于理解与维护）
  - 事件监听（可绑定多个事件相应函数，但事件驱动，会使运行流程的清晰度下降）
  - 订阅发布（观察者模式，与“事件监听”类似，通过常“消息中心”监控程序的运行）
  - Promise（以同步思维方式编写异步代码，便于对代码的理解、代码追踪及异常捕获）
- 使用Promise的含义及作用
  - Promise 对象保存着某个未来才会结束的事件（通常是一个异步操作）的结果，可获取异步消息
  - Promise 对象是一个代理对象，被代理的值在Promise对象创建时可能是未知的
  - Promise 为异步操作的两种状态成功(fulfilled)和失败(rejected)分别绑定相应的处理方法
  - Promise 让异步方法可以像同步方法那样返回值，是一个能代表未来出现的结果的promise对象
  - Promise 对象可以类比订餐时给的订餐票（有了订餐票就可以畅想之后的行为）

Promise概念及语法

- Promise中的几个基本概念
  - Promise对象中的执行器executor（有 resolve 和 reject 两个参数的函数 ，创建时会立即执行）
  - Promise对象有三种状态（pending进行中、fulfilled已成功、rejected已失败）
  - Promise对象的状态不受外界影响，只有异步操作的结果，可以决定当前应该为哪一种状态
  - 状态之间可能发生的两种状态转换（pending ->fulfilled 或者 pending ->rejected）
  - 状态发生转换后Promise 对象的 then 方法绑定的处理方法（handlers ）就会被调用



Promise概念及语法

Promise基本语法

- 通过Promise构造函数创建Promise对象，参数为一个函数对象（执行器executor）
- 执行器executor包含两个参数，即resolve 和 reject 两个函数（由引擎提供）
- 指定Promise状态转换后的then（接收两个回调函数作为参数）、catch方法、finally方法

```
let myFirstPromise = new Promise(function(resolve, reject){
    setTimeout(function(){
        resolve("成功!"); // 思考: 如果改为 reject("error");
    }, 2500);
});

myFirstPromise.then(function(successMessage){
    console.log("Hi! " + successMessage);
},function(errorMessage){
    console.log("Hi! " + errorMessage);
});
```

Promise的状态一旦改变就无法再次改变

then和catch返回值为Promise对象，可进行链式调用

参见实例demo1 Pomise基本语法

内容提纲

- Promise 概念及语法
- Promise 原型方法及静态方法
- Promise 综合案例

Promise原型方法及静态方法

- Promise原型方法（Promise.prototype.then）
  - Promise.prototype.then( )接收两个函数作为参数，分别用来响应fulfilled状态和rejected状态
  - Promise.prototype.then( )第二个参数可选，then返回的结果为Promise对象

```
function promiseTest(ms) {
    return new Promise((resolve, reject) => {
        console.log(111);
        setTimeout(resolve, ms, 'done');
        console.log(222);
    });
}
promiseTest(2000).then(
    (val) => { console.log('this is success callback:',val) },
    (err) => { console.log('this is fail callback:',err) }
)
```


参见实例demo2 Pomise原型方法then Part1

Promise原型方法及静态方法

- Promise原型方法（Promise.prototype.then）
  - Promise.prototype.then( )返回的结果为Promise对象，可以进行链式调用

```
var p = new Promise(function(resolved,rejected){
    console.log("11");
    resolved("22");//思考: rejected("22");
    console.log("33");
});
p
.then((v1)=>{console.log("44",v1);},
      (e1)=>{console.log("55",e1);})
.then((v2)=>{console.log("66",v2);},
      (e2)=>{console.log("77",e2);})
.then((v3)=>{console.log("88",v3);},
      (e2)=>{console.log("99",e3);})
```


参见实例demo2 Pomise原型方法then Part2

Promise原型方法及静态方法

- Promise原型方法（Promise.prototype.catch/finally）
  - Promise.prototype.catch()的作用是捕获Promise的错误，与then()的rejected回调作用几乎一致
  - Promise的抛错具有冒泡性质，能够不断传递，这样就能够在下一个catch()中统一处理这些错误
  - 建议不要使用then()的rejected回调，而是统一使用catch()来处理错误
  - catch()中也可以抛出错误，抛出的错误会在下一个catch中被捕获处理，因此可以再添加catch( )

```
var p2 = new Promise((resolve,reject)=>{
    console.log(111);
    //resolve(222);
    reject(new Error("222"));
    console.log(333);
});
p2
.then(()=>{console.log('444') })
.then(()=>{console.log("555");})
.catch((err)=>{console.log("666",err); }
)
```

注意：避免then中的rejected回调和catch同时使用

参见实例demo3 Pomise原型方法catch和finally

Promise原型方法及静态方法

- Promise静态方法（Promise.resolve）
  - Promise.resolve方法返回一个Promise对象（该Promise对象的状态由给定value决定）
  - 参数若是Promise对象则直接返回这个Promise对象
  - 参数若是带有then方法的对象，返回的Promise对象的最终状态由then方法执行决定
  - 参数若为空，基本类型或不带then方法的对象时，返回的Promise对象状态为fulfilled，并且将该value传递给对应的then方法
  - 通常而言，如果不知道一个值是否是Promise对象，使用Promise.resolve(value) 来返回一个Promise对象,这样就能将该value以Promise对象形式使用
- Promise静态方法（Promise.reject）
  - 方法返回一个带有拒绝原因reason参数的Promise对象

Promise原型方法及静态方法

- Promise静态方法（Promise.resolve）案例

```
Promise.resolve("Success").then(function (value) {
    console.log(value); // "Success"
}, function (value) {});
```

- Promise静态方法（Promise.reject）案例

```
var p = Promise.reject("reject reason");
p.then(
    (v)=>{console.log("v:",v)},
    (e)=>{console.log("e:",e)}
)
```


参见实例demo4 Pomise静态方法resolve和reject

Promise原型方法及静态方法

- Promise静态方法（Promise.all）
  - 这个方法返回一个新的Promise对象，Promise.all方法的参数为Promise对象数组
  - 该Promise对象在iterable参数对象里所有的Promise对象都成功的时候才会触发成功
  - 只要有任何一个iterable里面的promise对象失败则立即触发该Promise对象的失败
  - 这个新的Promise对象在触发成功状态以后，会把一个包含iterable里所有Promise返回值的数组作为成功回调的返回值，顺序跟iterable的顺序保持一致
  - 如果这个新的Promise对象触发了失败状态，它会把iterable里第一个触发失败的Promise对象的错误信息作为它的失败错误信息

- Promise静态方法（Promise.race）
  - 参数为Promise对象数组，参数里的任意一个子Promise成功或失败后，父Promise马上也会用子Promise的成功返回值或失败详情作为参数调用父Promise绑定的相应句柄，并返回该promise对象

Promise原型方法及静态方法

- Promise静态方法（Promise.all）案例

```
const p1 = new Promise((resolve, reject) => {resolve('hello');
}).then(result => result);
const p2 = new Promise((resolve, reject) => {resolve('world');//throw new Error('error'),
}).then(result => result);

Promise.all([p1, p2]) .then(result => console.log(result)).catch(e => console.log(e));
```

- Promise静态方法（Promise.race）案例

```
var p1 = new Promise((resolve,reject)=>{setTimeout(resolve,Math.random()*5000,"aaa")});
var p2 = new Promise((resolve,reject)=>{setTimeout(reject,Math.random()*5000,"bbb")});
var p3 = Promise.race([p1,p2]).then(
(val)=>{console.log("val:",val)},
(err)=>{console.log("err:",err)});
```


参见实例demo5 Pomise静态方法all和race

内容提纲

- Promise 概念及语法
- Promise 原型方法及静态方法
- Promise 综合案例

Promise综合案例

```
function loadImageAsync(url) {
  return new Promise(function (resolve, reject) {
    const image = new Image(url);
    image.onload = function () {
      image.width = image.height = 100;
      resolve(image);
    };
    image.onerror = function () {
      reject(new Error('Could not load image at ' + url));
    };
    image.src = url;
  });
}
```


更多案例参见 Demo06 Promise案例

Thank You !

河北師范大学软件学院
Software College of Hebei Normal University

Promise补充

- Promise参考链接

  https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise

  https://segmentfault.com/a/1190000010399626

  https://segmentfault.com/a/1190000007678185

- 作业（学习在线视频和Promise相关博客）

  https://www.imooc.com/learn/949

  https://www.jianshu.com/p/c98eb98bd00c

  https://segmentfault.com/a/1190000007678185