JS高级 —— 异步

题目

- 什么是单线程，和异步有什么关系
- 什么是 `event-loop`
- 是否用过 `jQuery` 的 `Deferred`
- `Promise` 的基本使用和原理
- 介绍一下 `async/await` （和 `Promise` 的区别/联系）
- 总结一下当前JS的异步解决方案

什么是单线程，和异步有什么关系

> 单线程就是同时只做一件事情，原因就是避免DOM渲染的冲突，异步是一种“无奈”的解决方案，当然也面临很多问题

- 单线程：只有一个线程，只能做一件事
- 原因：避免DOM渲染的冲突
  - 浏览器需要渲染DOM
  - JS可以修改DOM
  - JS执行的时候，DOM渲染会暂停
  - 两端JS不能同时执行（都修改DOM就冲突了）
  - H5中webworker支持多线程，但是不能访问DOM
- 解决方案：异步
  - 异步的问题
    - 没按照书写顺序执行，代码可读性差
    - callback中不容易模块化

`event-loop` （事件轮询）

- 解释
  - 事件轮询，JS实现异步的具体解决方案
  - 同步代码，直接执行
  - 异步函数先放在 `异步队列` 中
  - 待同步函数执行完毕，轮询执行 `异步队列` 的函数

`jQuery` 的 `Deferred`

- jQuery 1.5的变化
  - 无法改变JS异步和单线程的本质
  - 只能从写法上杜绝 callback 这种形式
  - 它是一种语法糖，但是解耦了代码
  - 很好的体现：开放封闭原则(开放封闭原则就是软件实体应该对扩展开放，而对修改封闭)

```
var wait = function() {
  var task = function() {
    console.log('执行完成');
  }
  setTimeout(task, 2000);
```

```
}
wait();
// jQuery Deffered
var waitHandler = function() {
  var deffer = $.Deferred()
  var wait = function(deffer) {
    var task = function() {
      console.log('执行完成');
      // 成功
      deffer.resolve();
      // 失败
      // deffer.reject();
    }
    setTimeout(task, 2000);
    // 返回一个promise对象
    return deffer.promise();
  }
  return deffer(deffer);
}
var w = waitHandler();
w.then(function() {
  console.log('success 1');
}, function() {
  console.log('error 1');
}).then(function() {
  console.log('success 2');
}, function() {
  console.log('error 2');
})
```

`Promise` 的基本使用和原理

- 基本语法

```
// Promise
function loadImg(src) {
  const promise = new Promise(function(resolve, reject) {
    let img = document.createElement('img');
    img.onload = function() {
      resolve(img);
    }
    img.onerror = function() {
      reject();
    }
    img.src = src;
  })
  return promise;
}
const src = 'https://www.imooc.com/static/img/index/logo_new.png';
const result = loadImg(src)
result.then(function(img) {
  console.log(img.width);
  // 链式操作，这里要返回img，否则下一个then里img为 undefined
  return img;
}, function() {
  console.log('fail');
}).then(function(img) {
```

```
  console.log(img.height);
})
```

- 异常捕获

```
function loadImg(src) {
    const promise = new Promise(function(resolve, reject) {
        let img = document.createElement('img');
        img.onload = function() {
            resolve(img);
        }
        img.onerror = function() {
            reject();
        }
        img.src = src;
        // 抛出一个异常
        throw new Error('自定义错误')
    })
    return promise;
}
const src = 'https://www.imooc.com/static/img/index/logo_new.png';
const result = loadImg(src)
result.then(function(img) {
    console.log(img.width);
    return img;
}).then(function(img) {
    console.log(img.height);
}).catch(function(error) {
    // 统一捕获异常
    console.log(error)
})
// 捕获reject异常
// 修改代码并修改图片src为一个不存在的地址
img.onerror = function() {
    reject('图片加载失败');
}
```

- 多个串联

```
const src = 'https://www.imooc.com/static/img/index/logo_new.png';
    const src2 = 'https://www.imooc.com/static/img/common/logo.png';
    const result1 = loadImg(src)
    const result2 = loadImg(src2)
    result1.then(function(img) {
        console.log(img.width);
        return result2;
    }).then(function(img) {
        console.log(img.width);
    }).catch(function(error) {
        // 统一捕获异常
        console.log(error)
    })
```

- Promise.all 和 Promise.race

```
const src = 'https://www.imooc.com/static/img/index/logo_new.png';
 const src2 = 'https://www.imooc.com/static/img/common/logo.png';
```

```
const result1 = loadImg(src)
const result2 = loadImg(src2)
// Promise.all 和 Promise.race 都接收一个数组
// Promise.all返回一个数组结果
Promise.all([result1, result2]).then(function(datas) {
    console.log('all', datas[0])
    console.log('all', datas[1])
})
// Promise.race返回先执行完的那个数据结果
Promise.race([result1, result2]).then(function(data) {
    console.log('race', data)
})
```

- Promise标准
  - Promise 实例必须实现then这个方法
  - then()必须可以接收两个函数作为参数
  - then()返回的必须是一个Promise实例
- Promise 状态变化
  - 三种状态： `pending` `fulfilled` `rejected`
  - 初始状态是 `pending`
  - `pending-->fulfilled`,或者 `pending--> rejected`

`async/await` （ES7提案）

```
const load = async function() {
  const result1 = await loadImg(src1);
  console.log(result1);
  const result2 = await loadImg(src2);
  console.log(result2);
}
load();
```

- `then` 只是将 `callback` 拆分了
- `async/await` 是最直接的同步写法
- 用法
  - 使用 `await` ,函数必须用 `async` 标识
  - `await` 后面跟的是一个 `Promise` 实例
  - 需要 `babel-polyfill` 支持
- 特点
  - 使用了Promise的封装，并没有和Promise冲突，是和Promise的结合
  - 完全是同步的写法，再也没有回调函数
  - 改变不了JS单线程和异步的本质