**JS执行——Promise**

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理且更强大。它最早由社区提出并实现，ES6将其写进了语言标准，统一了用法，并原生提供了Promise对象。

**特点**

1. 对象的状态不受外界影响 （3种状态）
   * Pending状态（进行中）
   * Fulfilled状态（已成功）
   * Rejected状态（已失败）
2. 一旦状态改变就不会再变 （两种状态改变：成功或失败）
   * Pending -> Fulfilled
   * Pending -> Rejected

**用法**

**创建Promise实例**

var promise = new Promise(function(resolve, reject){

// ... some code

if (/\* 异步操作成功 \*/) {

resolve(value);

} else {

reject(error);

}

})

  Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。它们是两个函数，由JavaScript引擎提供，不用自己部署。  
  resolve作用是将Promise对象状态由“未完成”变为“成功”，也就是Pending -> Fulfilled，在异步操作成功时调用，并将异步操作的结果作为参数传递出去；而reject函数则是将Promise对象状态由“未完成”变为“失败”，也就是Pending -> Rejected，在异步操作失败时调用，并将异步操作的结果作为参数传递出去。

**then**

  Promise实例生成后，可用then方法分别指定两种状态回调参数。then 方法可以接受两个回调函数作为参数：

1. Promise对象状态改为Resolved时调用 （必选）
2. Promise对象状态改为Rejected时调用 （可选）

**基本用法示例**

function sleep(ms) {

return new Promise(function(resolve, reject) {

setTimeout(resolve, ms);

})

}

sleep(500).then( ()=> console.log("finished"));

  这段代码定义了一个函数sleep，调用后，等待了指定参数(500)毫秒后执行then中的函数。值得注意的是，Promise新建后就会立即执行。

**执行顺序**

  接下来我们探究一下它的执行顺序，看以下代码：

let promise = new Promise(function(resolve, reject){

console.log("AAA");

resolve()

});

promise.then(() => console.log("BBB"));

console.log("CCC")

// AAA

// CCC

// BBB

  执行后，我们发现输出顺序总是 AAA -> CCC -> BBB。表明，在Promise新建后会立即执行，所以首先输出 AAA。然后，then方法指定的回调函数将在当前脚本所有同步任务执行完后才会执行，所以BBB 最后输出。

**与定时器混用**

  首先看一个实例：

let promise = new Promise(function(resolve, reject){

console.log("1");

resolve();

});

setTimeout(()=>console.log("2"), 0);

promise.then(() => console.log("3"));

console.log("4");

  可以看到，结果输出顺序总是：1 -> 4 -> 3 -> 2。1与4的顺序不必再说，而2与3先输出Promise的then，而后输出定时器任务。原因则是Promise属于JavaScript引擎内部任务，而setTimeout则是浏览器API，而引擎内部任务优先级高于浏览器API任务，所以有此结果。

**拓展 async/await**

**async**

  顾名思义，异步。async函数对 Generator 函数的改进，async 函数必定返回 Promise，我们把所有返回 Promise 的函数都可以认为是异步函数。特点体现在以下四点：

* 内置执行器
* 更好的语义
* 更广的适用性
* 返回值是 Promise

**await**

  顾名思义，等待。正常情况下，await命令后面是一个 Promise 对象，返回该对象的结果。如果不是 Promise 对象，就直接返回对应的值。另一种情况是，await命令后面是一个thenable对象（即定义then方法的对象），那么await会将其等同于 Promise 对象。

**混合使用**

  先看示例：

function sleep(ms) {

return new Promise(function(resolve, reject) {

setTimeout(resolve,ms);

})

}

async function handle(){

console.log("AAA")

await sleep(5000)

console.log("BBB")

}

handle();

// AAA

// BBB (5000ms后)

  我们定义函数sleep，返回一个Promise。然后在handle函数前加上async关键词，这样就定义了一个async函数。在该函数中，利用await来等待一个Promise。