# 1.Promise解决什么问题，解决js的回到地狱问题

|  |
| --- |
|  |

# 2.Promise其实是一个类，你需要使用new Promise来创建对象

# 3.Promise的基本使用。

## 1.我们新建一个文件夹promise，在里面新建一个promise\_basic.js，然后新建一个文件夹，里面放一个test.txt文件

|  |
| --- |
|  |

## 2.promise\_basic.js的内容如下

|  |
| --- |
| //Promise的基本使用  let fs = require('fs')  let path = require('path')  let filePath = path.resolve(\_\_dirname,"files","test.txt") //路径拼接  // 使用Promise  let p = new Promise((resolve,reject)=>{      fs.readFile(filePath,"utf8",(err,data)=>{      if(err){          reject(err)      }      resolve(data)    })  })  p.then(data=>{      console.log('data',data);  }).catch(err=>{      console.log(err);  }) |

## 注意：创建一个Promise对象需要传递两个函数resolve和reject，resolve是用来处理任务执行成功后的数据，reject函数用来处理任务失败时的错误数据，使用时，用then来获取成功的数据和catch来输出错误的原因，如上面的例子，如果fs读取的文件不存在，就会删除一个文件不存在错误，文件存在，就能够读取文件的内容并且输出。

|  |
| --- |
|  |
|  |

## 注意：then方法可以链式调用，可以一直使用

|  |
| --- |
| //Promise的基本使用  let fs = require('fs')  let path = require('path')  let filePath = path.resolve(\_\_dirname,"files","test.txt") //路径拼接  // 使用Promise  let p = new Promise((resolve,reject)=>{      fs.readFile(filePath,"utf8",(err,data)=>{      if(err){          reject(err)      }      resolve(data)    })  })  //写法1  // p.then(data=>{  //     console.log('data',data);  // }).catch(err=>{  //     console.log(err);  // })  //写法2  p.then((data)=>{      console.log("data:",data);  },(err)=>{      console.log("error:",err);  }).then(()=>{      console.log("the second then func...");  }) |

## 效果

|  |
| --- |
|  |

## 注意：不管第一个then执行哪一个函数，第二个then都会执行

|  |
| --- |
|  |

## 可以在第一个then函数里面返回一个值，然后就可以在第二个then里面接收这个值

## 2-promise\_then\_chain.js

|  |
| --- |
| //Promise的基本使用  let fs = require('fs')  let path = require('path')  let filePath = path.resolve(\_\_dirname,"files","test.txt") //路径拼接  // 使用Promise  let p = new Promise((resolve,reject)=>{      fs.readFile(filePath,"utf8",(err,data)=>{      if(err){          reject(err)      }      resolve(data)    })  })  //写法1  // p.then(data=>{  //     console.log('data',data);  // }).catch(err=>{  //     console.log(err);  // })  //写法2  p.then((data)=>{      console.log("data:",data);      return "file data:" + data  },(err)=>{      console.log("error:",err);  }).then((data)=>{      console.log("the second then func...");      console.log("data from the first then func:",data);  }) |
|  |

## 即使第一个then调用失败的函数，它也可以由返回值、如下面我们把文件夹写错制作一个错误

|  |
| --- |
| //Promise的基本使用  let fs = require('fs')  let path = require('path')  let filePath = path.resolve(\_\_dirname,"files","test1.txt") //路径拼接  // 使用Promise  let p = new Promise((resolve,reject)=>{      fs.readFile(filePath,"utf8",(err,data)=>{      if(err){          reject(err)      }      resolve(data)    })  })  //写法1  // p.then(data=>{  //     console.log('data',data);  // }).catch(err=>{  //     console.log(err);  // })  //写法2  p.then((data)=>{      console.log("data:",data);      return "file data:" + data  },(err)=>{      console.log("error:",err);      console.log("==================================");      return {          result:'failed',          error:err      }  }).then((data)=>{      console.log("the second then func...");      console.log("data from the first then func:",data);  }) |

## 效果：

|  |
| --- |
|  |

## 注意：如果一个then函数里面返回一个Promise对象，下一个then获取到的不是这个Promise对象，而是它的resolve函数的参数

|  |
| --- |
| //Promise的基本使用  let fs = require('fs')  let path = require('path')  let filePath = path.resolve(\_\_dirname,"files","test.txt") //路径拼接  // 使用Promise  let p = new Promise((resolve,reject)=>{      fs.readFile(filePath,"utf8",(err,data)=>{      if(err){          reject(err)      }      resolve(data)    })  })  //写法1  // p.then(data=>{  //     console.log('data',data);  // }).catch(err=>{  //     console.log(err);  // })  //写法2  // p.then((data)=>{  //     console.log("data:",data);  //     return "file data:" + data  // },(err)=>{  //     console.log("error:",err);  //     console.log("==================================");  //     return {  //         result:'failed',  //         error:err  //     }  // }).then((data)=>{  //     console.log("the second then func...");  //     console.log("data from the first then func:",data);  // })  //then里面返回Promise对象的情况  p.then((data)=>{      console.log("data:",data);      return p  },(err)=>{      console.log("error:",err);      console.log("==================================");      return {          result:'failed',          error:err      }  }).then((data)=>{      console.log("the second then func...");      console.log("data from the first then func:",data);  }) |

## 效果：

|  |
| --- |
|  |

# 4.Promise的函数

## 1.catch和finally

|  |
| --- |
|  |

### 注意：finally函数无论上面执行是成功还是失败都会执行

## 2. Promise.all(),这个方法的参数是一个由n个Promise对象的数组，只有当所有的Promise对象都执行成功它才会成功，这时候then方法里面的data获取到的是一个数组，里面由每一个Promise对象返回的数据，我们把连接文件内容的例子代码修改为all方法来实现代码如下

|  |
| --- |
| let fs = require('fs')  let path = require('path')  let util = require('util')  let filePath1 = path.join(\_\_dirname,"files/client.txt")  let filePath2 = path.join(\_\_dirname,"files/secret.txt")  let filePath3 = path.join(\_\_dirname,"files/info.txt")  const readFile = util.promisify(fs.readFile) //util.promisify可以把一个异步函数变为一个返回Promise的函数  let p1 = readFile(filePath1,"utf-8")  let p2 = readFile(filePath2,"utf-8")  let p3 = readFile(filePath3,"utf-8")  Promise.all([p1,p2,p3]).then(data=>{      //这里的data是一个数组，包含每一个Promise对象的数据      // console.log(data);      console.log(data.join('\r\n'));  }) |

### 效果如下

|  |
| --- |
|  |

### 注意：all方法中只要数组中由一个Promise对象执行失败，整个all方法就失败，返回的是失败能够Promise对象的error，如我们把上面的文件路径的一个搞错，结果如下

|  |
| --- |
|  |

### 可见此时只有错误数据，没有正确的数据

## 3.Promise.race(),这个方法接收一个Promise对象数组，它的返回值是随机的，就看哪个Promise对象先执行完毕，就才有他的结果不管是成功还是失败

### 示例

|  |
| --- |
|  |
| let fs = require('fs')  let path = require('path')  let util = require('util')  let filePath1 = path.join(\_\_dirname,"files/client.txt")  let filePath2 = path.join(\_\_dirname,"files/secret.txt")  let filePath3 = path.join(\_\_dirname,"files/info.txt")  const readFile = util.promisify(fs.readFile) //util.promisify可以把一个异步函数变为一个返回Promise的函数  let p1 = readFile(filePath1,"utf-8")  let p2 = readFile(filePath2,"utf-8")  let p3 = readFile(filePath3,"utf-8")  Promise.race([p1,p2,p3]).then(data=>{ //race方法只要由一个Promise对象执行完成，就会结束      //这里的data包含执行完成那个个Promise对象的数据      console.log(data);  }) |

## 效果

|  |
| --- |
|  |

### all方法和race方法的区别：1.all返回的数据是一个数组，即使他只有一条数据，race返回的是一条数据而不是一个数组，2.all方法会等待到所有Promise对象执行完毕才会由结果，race方法只有由一个Promise对象执行完成，不管成功还是失败都马上又结果