一、概述

- 同步模式与异步模式
- 事件循环与消息队列
- 异步编程的几种方式
- Promise异步方案、宏任务/微任务队列
- Generator异步方案、Async/Await语法糖

二、同步模式 Synchronous

三、异步模式Asynchronous

```
console.log('global begin'); //1

setTimeout(function timer1 () {
 console.log('timer1 invoke'); //4
}, 1800)

setTimeout(function timer2 () {
 console.log('timer2 invoke'); //3

  setTimeout(function inner () {
  console.log('inner invoke'); //5
  }, 1000)
}, 1000)

console.log('global end') //2
```

四、回调函数

回调函数可以理解为一件你想要做的事情

五、Promise概述 - 一种更优的异步编程统一方案

六、Promise基本用法

```
const promise = new Promise(function (resolve, reject) {
 //这里用于‘兑现’承诺
 resolve(100) // 承诺达成
 reject(new Error('promise rejected')) // 承诺失败
})

promise.then(function (value) {
 console.log('resolved', value)
}, function (error) {
  console.log('rejected', error)
 })

console.log('end')
```

七、Promise 使用案例

```
// Promise 方式的AJAX
function ajax (url) {
 return new Promise(function (resolve,reject) {
 var xhr = new XMLHttpRequest()
 xhr.open('GET', url)
 xhr.responseType = 'json'
 xhr.onload = function () {
 if (this.status === 200) {
 resolve(this.response)
  } else {
  reject(new Error(this.statusText))
  }
  }
  xhr.send()
```

```
 })
}

ajax('/api/users.json').then(function (res) {
 console.log(res)
}, function (error) {
 console.log(error)
})
```

八、Promise 常见误区

九、Promise 链式调用

```
ajax('/api/users.json')
 .then(function (value) {
 console.log(111)
 return ajax('/api/users.json')
 })
 .then(function (value) {
 console.log(222)
 })
 .then(function (value) {
 console.log(333)
 return 'foo'
 })
 .then(function (value) {
 console.log(444)
 console.log(value) // foo
 })

//Promise对象的then方法会返回一个全新的Promise对象
//后面的then方法就是为上一个then返回Promise注册回调
// 前面then方法中回调函数的返回值会作为后面then方法回调的参数
// 如果回调中返回的是Promise，那后面then方法的回调会等待它的结束
```

十、Promise 异常处理

```
ajax('/api/users111.json')
 .then(function onFulfilled (value) {
 console.log('onFulfilled', value)
 })
 .catch(function onRejected (error) {
 console.log('onRejected', error)
 })

ajax('/api/users111.json')
  .then(function onFulfilled (value) {
  console.log('onFulfilled', value)
  })
  .then(undefined, function onRejected (error) {
  console.log('onRejected', error)
  })
```

十一、Promise 静态方法

```
Promise.resolve('foo')
 .then(function (value) {
 console.log(value)
 })

new Promise(function (resolve, reject) {
 resolve('foo')
})

var promise = ajax('/api/users.json');
var promise2 = Promise.resolve(promise);
console.log(promise === promise2)

Promise.resolve({
```

```
 then:function (onFulfilled, onRejected) {
 onFulfilled('foo');
 }
})
.then(function (value) {
 console.log(value)
})
```

十二、Promise 并行执行

```
//Promise.all
ajax('/api/urls.json')
 .then(value => {
 const urls = Object.values(value);
 const tasks = urls.map(url => ajax(url))
 return Promise.all(tasks)
 })
 .then(values => {
 console.log(values)
 })

const request = ajax('/api/posts.json')
const timeout = new Promise((resolve, reject) => {
 setTimeout(() => reject(new Error('timeout')), 500)
})

//Promise.race
Promise.race([
 request,
 timeout
])
.then(value => {
 console.log(value)
})
.catch(error => {
```

```
 console.log(error)
})
```

十三、Promise 执行时许/宏任务vs.微任务

```
// 微任务
console.log('global start'); //1
setTimeout(() => {
 console.log('setTimeout') //4 以宏任务形式回到回调队列的末尾（嗯，完全没听懂）
},0)
Promise.resolve()
 .then(() => {
 console.log('promise') //3 微任务
 })
console.log('global end') //2
```

十四、Generator异步方案(上)

```
function * foo () {
 console.log('start')

 try {
 const res = yield 'foo'
 console.log(res)
 } catch (e) {
 console.log(e)
 }
}

const generator = foo ()

const result = generator.next()
console.log(result)

```

```
//generator.next('bar')
generator.throw(new Error('Generator error'))
```

十五、Generator异步方案(中)

```
function * main () {
 const users = yield ajax('/api/users.json')
 console.log(users)

 const posts = yield ajax('/api/post.json')
 console.log(posts)

 const urls = yield ajax('/api/urls11.json')
 console.log(urls)
} catch (e) {
  console.log(e)
}

const g = main()

const result = g.next()

result.value.then(data => {
  const result2 = g.next(data)

  if (result2.done) return
  result2.value .then(data => {
  const result3 = g.next(data)

  if (result3.done) return
  result3.value.then(data => {
  g.next(data)
  })
  })
})
```

十六、Generator异步方案(下)

```
function * main () {
 const users = yield ajax('/api/users.json')
 console.log(users)

 const posts = yield ajax('/api/post.json')
 console.log(posts)
}

const g = main()

function handleResult (result) {
 if (result.done) return // 生成器函数结束
 result.value.then(data => {
 handleResult(g.next(data))
 }, error => {
 g.throw(error)
 })
}
```

十七、Async / Await 语法糖

```
async function main () {
 const users = await yield ajax('/api/users.json')
 console.log(users)

 const posts = await yield ajax('/api/post.json')
 console.log(posts)

 const urls = await yield ajax('/api/urls11.json')
 console.log(urls)
} catch (e) {
 console.log(e)
```

```
12 }
13
14 const promise = main()
15 promise.then(() => {
16  console.log('all completed');
17 })
```