模块化进化史教程

1. 全局function模式

- module1.js

```
//数据
let data = 'atguigu.com'

//操作数据的函数
function foo() {
  console.log(`foo() ${data}`)
}
function bar() {
  console.log(`bar() ${data}`)
}
```

- module2.js

```
let data2 = 'other data'

function foo() {  //与另一个模块中的函数冲突了
  console.log(`foo() ${data2}`)
}
```

- test1.html

```
<script type="text/javascript" src="module1.js"></script>
<script type="text/javascript" src="module2.js"></script>
<script type="text/javascript">

  let data = "修改后的数据"
  foo()
  bar()
</script>
```

- 说明:
  - 全局函数模式: 将不同的功能封装成不同的全局函数
  - 问题: Global被污染了, 很容易引起命名冲突

2. namespace模式

- module1.js

```
let myModule = {
  data: 'atguigu.com',
  foo() {
    console.log(`foo() ${this.data}`)
  },
  bar() {
    console.log(`bar() ${this.data}`)
  }
}
```

- module2.js

```
let myModule2 = {
  data: 'atguigu.com2222',
  foo() {
    console.log(`foo() ${this.data}`)
  },
  bar() {
    console.log(`bar() ${this.data}`)
  }
}
```

- test2.html

```
<script type="text/javascript" src="module2.js"></script>
<script type="text/javascript" src="module22.js"></script>
<script type="text/javascript">
  myModule.foo()
  myModule.bar()

  myModule2.foo()
  myModule2.bar()

  myModule.data = 'other data' //能直接修改模块内部的数据
  myModule.foo()

</script>
```

- 说明
    - namespace模式: 简单对象封装
    - 作用: 减少了全局变量
    - 问题: 不安全

3. IIFE模式

- module3.js

```
(function (window) {
  //数据
  let data = 'atguigu.com'

  //操作数据的函数
  function foo() { //用于暴露有函数
    console.log(`foo() ${data}`)
  }

  function bar() {//用于暴露有函数
    console.log(`bar() ${data}`)
    otherFun() //内部调用
  }

  function otherFun() { //内部私有的函数
    console.log('otherFun()')
  }

  //暴露行为
  window.myModule = {foo, bar}
})(window)
```

- test3.html

```
<script type="text/javascript" src="module3.js"></script>
<script type="text/javascript">
  myModule.foo()
  myModule.bar()
  //myModule.otherFun()  //myModule.otherFun is not a
function
  console.log(myModule.data) //undefined 不能访问模块内部数据
  myModule.data = 'xxxx' //不是修改的模块内部的data
  myModule.foo() //没有改变

</script>
```

- 说明:
  - IIFE模式: 匿名函数自调用(闭包)
  - IIFE : immediately-invoked function expression(立即调用函数表达式)
  - 作用: 数据是私有的, 外部只能通过暴露的方法操作
  - 问题: 如果当前这个模块依赖另一个模块怎么办?

4. IIFE模式增强

- 引入jquery到项目中
- module4.js

```
(function (window, $) {
  //数据
  let data = 'atguigu.com'

  //操作数据的函数
  function foo() { //用于暴露有函数
    console.log(`foo() ${data}`)
    $('body').css('background', 'red')
  }

  function bar() {//用于暴露有函数
    console.log(`bar() ${data}`)
    otherFun() //内部调用
  }

  function otherFun() { //内部私有的函数
    console.log('otherFun()')
  }
```

```
  //暴露行为
  window.myModule = {foo, bar}
})(window, jQuery)
```

- test4.html

```
<script type="text/javascript" src="jquery-1.10.1.js">
</script>
<script type="text/javascript" src="module4.js"></script>
<script type="text/javascript">
  myModule.foo()
</script>
```

- 说明
    - IIFE模式增强 : 引入依赖
    - 这就是现代模块实现的基石

5. 页面加载多个js的问题

- 页面:

```
<script type="text/javascript" src="module1.js"></script>
<script type="text/javascript" src="module2.js"></script>
<script type="text/javascript" src="module3.js"></script>
<script type="text/javascript" src="module4.js"></script>
```

- 说明
    - 一个页面需要引入多个js文件
    - 问题:
        - 请求过多
        - 依赖模糊
        - 难以维护
    - 这些问题可以通过现代模块化编码和项目构建来解决