

jQuery

- 今天我们继续来聊 jQuery

发送 ajax 请求

- 发送 get 请求

```
1. // 直接使用 $.get 方法来发送一个请求
2. /*
3.   参数一： 请求地址
4.   参数二： 请求时携带的参数
5.   参数三： 请求成功的回调
6.   参数四： 返回的数据类型
7. */
8. $.get('./ajax.php', { id: 10 }, function (res) { console.log(res) }, 'json')
```

- 发送 post 请求

```
1. // 直接使用 $.post 方法来发送一个请求
2. /*
3.   参数一： 请求地址
4.   参数二： 请求时携带的参数
5.   参数三： 请求成功的回调
6.   参数四： 返回的数据类型
7. */
8. $.post('./ajax.php', { id: 10 }, function (res) { console.log(res) }, 'json')
```

- 综合发送 ajax 请求

```
1. // 使用 $.ajax 方法
2. // 只接受一个参数，是一个对象，这个对象对当前的请求进行所有的配置
3. $.ajax({
4.   url: './ajax', // 必填，请求的地址
5.   type: 'GET', // 选填，请求方式，默认是 GET（忽略大小写）
6.   data: {}, // 选填，发送请求是携带的参数
7.   dataType: 'json', // 选填，期望返回值的数据类型，默认是 string
8.   async: true, // 选填，是否异步，默认是 true
9.   success () {}, // 选填，成功的回调函数
10.  error () {}, // 选填，失败的回调函数
```

```
11.  cache: true, // 选填, 是否缓存, 默认是 true
12.  context: div, // 选填, 回调函数中的 this 指向, 默认是 ajax 对象
13.  status: {}, // 选填, 根据对应的状态码进行函数执行
14.  timeout: 1000, // 选填, 超时事件
15.  })
```

- 发送一个 jsonp 请求

```
1.  // 使用 $.ajax 方法也可以发送 jsonp 请求
2.  // 只不过 dataType 要写成 jsonp
3.  $.ajax({
4.    url: './jsonp.php',
5.    dataType: 'jsonp',
6.    data: { name: 'Jack', age: 18 },
7.    success (res) {
8.      console.log(res)
9.    },
10.   jsonp: 'cb', // jsonp 请求的时候回调函数的 key
11.   jsonpCallback: 'fn' // jsonp 请求的时候回调函数的名称
12. })
```

全局 ajax 函数

- 全局的 `ajax` 函数我们也叫做 `ajax` 的钩子函数
- 也就是在一个 `ajax` 的整个过程中的某一个阶段执行的函数
- 而且每一个 `ajax` 请求都会触发

ajaxStart

- 任意一个请求在 **开始** 的时候就会触发这个函数

```
1.  $(window).ajaxStart(function () {
2.    console.log('有一个请求开始了')
3.  })
```

ajaxSend

- 任意一个请求在 **准备 send 之前** 会触发这个函数

```
1.  $(window).ajaxSend(function () {
2.    console.log('有一个要发送出去了')
```

```
3.  })
```

ajaxSuccess

- 任意一个请求在 **成功** 的时候就会触发这个函数

```
1.  $(window).ajaxSuccess(function () {  
2.      console.log('有一个请求成功了')  
3.  })
```

ajaxError

- 任意一个请求在 **失败** 的时候就会触发这个函数

```
1.  $(window).ajaxError(function () {  
2.      console.log('有一个请求失败了')  
3.  })
```

ajaxComplete

- 任意一个请求在 **完成** 的时候就会触发这个函数

```
1.  $(window).ajaxComplete(function () {  
2.      console.log('有一个请求完成了')  
3.  })
```

ajaxStop

- 任意一个请求在 **结束** 的时候就会触发这个函数

```
1.  $(window).ajaxStop(function () {  
2.      console.log('有一个请求结束了')  
3.  })
```

jQuery 的多库共存

- 我们一直在使用 `jQuery`，都没有什么问题
- 但是如果有一天，我们需要引入一个别的插件或者库的时候
- 人家也向外暴露的是 `$` 获取 `jQuery`

- 那么，我们的 `jQuery` 就不能用了
- 那么这个时候， `jQuery` 为我们提供了一个多库并存的方法

```
1. // 这个方法可以交还 jQuery 命名的控制权
2. jQuery.noConflict()
3.
4. // 上面代码执行完毕以后 $ 这个变量就不能用了
5. // 但是 jQuery 可以使用
6. console.log($) // undefined
7. console.log(jQuery) // 可以使用
```

- 完全交出控制权

```
1. // 这个方法可以交并且传递一个 true 的时候，会完全交出控制权
2. jQuery.noConflict(true)
3.
4. // 上面代码执行完毕以后 $ 这个变量就不能用了
5. // jQuery 这个变量也不能用了
6. console.log($) // undefined
7. console.log(jQuery) // undefined
```

- 更换控制权

```
1. // 可以用一个变量来接受返回值，这个变量就是新的控制权
2. var aa = jQuery.noConflict(true)
3.
4. // 接下来就可以把 aa 当作 jQuery 向外暴露的接口使用了
5. aa('div').click(function () { console.log('我被点击了') })
```

jQuery 的插件扩展

- `jQuery` 确实很好很强大
- 但是也有一些方法是他没有的，我们的业务需求中有的时候会遇到一些它里面没有的方法
- 那么我们就可以给他扩展一些方法

扩展给他自己本身

- 扩展给自己本身使用 `jQuery.extend` 这个方法
- 扩展完后的内容只能用 `$` 或者 `jQuery` 来调用

1. // 扩展 jQuery 本身，将 jQuery 的 prototype 对象与 jQuery 的静态方法一起暴露到全局，并返回这个对象以便链式调用的方式

```
1. // jQuery.extend 接受一个参数，是一个对象，对象里面是我们扩展的万法
2. jQuery.extend({
3.   max: function (...n) { return Math.max.apply(null, n) },
4.   min: function (...n) { return Math.min.apply(null, n) }
5. })
```

- 扩展完毕我们就可以使用了

```
1. const max = $.max(4, 5, 3, 2, 6, 1)
2. console.log(max) // 6
3. const min = $.min(4, 5, 3, 2, 6, 1)
4. console.log(min) // 1
```

扩展给元素集

- 扩展完毕以后给元素的集合使用
- 也就是我们用 `$('li')` 这样的选择器获取到的元素集合来使用
- 使用 `jQuery.fn.extend()` 方法来扩展

```
1. // jQuery.fn.extend() 接受一个参数，是一个对象，对象里面是我们扩展的方法
2. jQuery.fn.extend({
3.   checked: function () {
4.     // return 关键字是为了保证链式编程
5.     // 后面的代码才是业务逻辑
6.     return this.each(function() { this.checked = true })
7.   }
8. })
```

- 扩展完毕我们就可以使用了

```
1. // 靠元素集合来调用
2. $('input[type=checkbox]').checked()
3. // 执行完毕之后，所有的 复选框 就都是选中状态了
```