# 复习

FS模块：NodeJS的核心模块之一。用于操作文件和文件夹。

创建文件： appendFile(filePath, content, callback);

删除文件： unlink(filePath, callback);

读取文件： readFile(filePath, callback);

重命名： rename(oldPath, newPath, callback);

创建文件夹： mkdir(dirPath, callback);

删除文件夹： rmdir(dirPath, callback);

读取文件夹： readdir(dirPath, callback);

判定目标状态：stat(targetPath, callback);

注：以上所有方法都有一个同名Sync方法。是同步的方法。

URL模块：NodeJS的核心模块之一。用于操作URL字符串。

解析： parse(urlStr, boolean); 用于将URL字符串解析为URL对象。

格式化：format(urlObj); 用于将URL对象转回URL字符串

QueryString模块：NodeJS的核心模块之一。用于操作QueryString字符串

解析： parse(querystring, bigSplit, smallSplit); 用于将query字符串或者类似的字符串转为对象

NodeJS获取GET请求数据：

因为GET请求的数据，携带在Query上，所以可以使用URL模块顺带处理URL时，得到Query对象。

NodeJS获取POST请求数据：

因为POST请求的数据容量很大，可能一次无法传递完毕，所以需要使用事件。

data：每一段数据发送到服务器，就会触发一次。（可能触发多次）

end：所有数据都传递完毕之后，触发一次。（只会触发一次）

静态服务器：将服务器开启之后，此时，放入的内容，无需重启服务器，就可以直接被浏览器访问到。

接口：服务器资源的一种。通常是URL，当把请求发送给URL时，后台服务器会执行一段处理代码。再根据代码的执行结果返回可能相同、可能不同的结果。

MimeType：互联网给每一种资源定义的类型。

jQuery中发送AJAX：

$.get(url, data, callback, dataType); 发送GET请求的简写方式

url: 目标地址也就是URL

data: 携带的数据 可以省略 可以是query字符串 可以是对象

callback: 回调函数 回调函数中有一个参数 表示本次响应的数据

dataType: 返回的数据类型

$.post(url, data, callback, dataType) 发送POST请求的简写方式

url: 目标地址也就是URL

data: 携带的数据 可以省略 可以是query字符串 可以是对象

callback: 回调函数 回调函数中有一个参数 表示本次响应的数据

dataType: 返回的数据类型

$.ajax(option); 发送AJAX 具体的内容由参数决定

option 是配置对象

url: 目标地址也就是URL

data: 携带的数据 可以省略 可以是query字符串 可以是对象

success: 成功回调函数 回调函数中有一个参数 表示本次响应的数据

dataType: 返回的数据类型

error: 失败回调函数

complete: 完成时的回调函数（不论成功失败）

contentType: 布尔值配置 默认为true 表示是否由jQuery决定本次发送的请求的请求头content-type字段

processData: 布尔值配置 默认也是true 表示是否由jQuery决定本次数据的序列化

# 一、AJAX

## 1.1 兼容性

AJAX的兼容系很好。最早可以兼容到IE5。

因为AJAX是一套体系。

只要浏览器能够“悄悄地发起HTTP请求，去服务器请求内容，返回到浏览器，更新部分页面或者执行其它操作”。我们就说它能够支持AJAX。

不同的浏览器支持的方式不一样。

IE8及之前： ActiveXObject("Microsoft.XMLHttp")

IE8及之后与高级浏览器： XMLHttpRequest();

能力检测:

|  |
| --- |
| 1. if (window.XMLHttpRequest) { 2. var xhr = new XMLHttpRequest(); 3. } else if (window.ActiveXObject) { 4. var xhr = new ActiveXObject("Microsoft.XMLHttp"); 5. } else { 6. throw new Error("您的浏览器不支持AJAX，请更换或升级"); 7. } |

## 1.2 XHR对象

属性和方法：

|  |
| --- |
|  |

## 1.3 初始化

初始化AJAX要有四步：

1 初始化XHR对象

2 设置onreadystatechange事件

3 调用open方法

4 调用send方法

demo:

|  |
| --- |
| 1. // 定义xhr 2. var xhr = null; 3. if (window.XMLHttpRequest) { 4. xhr = new XMLHttpRequest(); 5. } else if (window.ActiveXObject) { 6. xhr = new ActiveXObject("Microsoft.XMLHttp"); 7. } else { 8. throw new Error("您的浏览器不支持AJAX，请更换或升级"); 9. } 10. console.log(xhr); 11. // 设置onreadystatechange事件 12. xhr.onreadystatechange = function() { 13. console.log("状态发生了变化"); 14. // 什么时候状态值变为4 什么时候获取数据 15. if (xhr.readyState === 4 && xhr.status === 200) { 16. console.log(xhr.responseText); 17. var div = document.createElement("div"); 18. var obj = JSON.parse(xhr.responseText); 19. div.innerHTML = obj.data; 20. document.body.appendChild(div); 21. } 22. } 23. // 调用open方法 24. xhr.open("post", "/test", true); 25. // 调用send方法 26. xhr.send(); |

## 1.4 onreadystatechange

这是XHR对象的一个事件属性。

该事件会在每一个HTTP请求的不同阶段被触发。每一个阶段有不同的状态值。

xhr.readyState 表示不同的阶段的状态值 一共有5个值

0：未初始化

1：调用了open方法

2：响应头已经到达浏览器

3：已经接收到一部分数据

4：已经接收完所有的数据

每一次状态值的变化都会引起readystatechange事件的触发。

## 1.5 open方法

该方法用于打开TCP连接。

xhr.open(method, url, boolean);

method: HTTP请求的方式

url: HTTP请求的URL

boolean: 决定是否异步发送HTTP请求

## 1.6 send方法

该方法用于发送HTTP请求的具体内容 也可以理解为请求正文中的内容

因为GET请求没有请求正文，所以如果xhr.open("get", ……); 则send方法中不需要填写任何内容。

所以，只有在POST请求时，才会在send方法中填写内容。

xhr.send(data);

data: querystring字符串

## 1.7 接收响应数据

xhr.response、xhr.responseText 这两个属性用于获取后端返回的数据。

# 封装AJAX轮子

## 2.1 封装GET请求

|  |
| --- |
| 1. var ickt = { 2. get: function(url, data, callback) { 3. // data既可以是对象又可以是query字符串 4. var query = ""; 5. // 判定data的数据类型 6. if (typeof data === "object") { 7. // 是对象 需要循环转为query字符串 8. for (var i in data) { 9. query += i + "=" + data[i] + "&"; 10. } 11. query = query.slice(0, -1); 12. } else if (typeof data === "string") { 13. query = data; 14. } 15. // 初始化XHR 16. var xhr = null; 17. // 能力检测 18. if (window.XMLHttpRequest) { 19. xhr = new XMLHttpRequest(); 20. } else if (window.ActiveXObject) { 21. xhr = new ActiveXObject("Microsoft.XMLHttp"); 22. } else { 23. throw new Error("您的浏览器不支持AJAX，请升级"); 24. } 25. // 设置onreadystatechange事件 26. xhr.onreadystatechange = function() { 27. // 因为当前函数会执行多次 我们最好加判定 28. if (xhr.readyState === 4 && xhr.status === 200) { 29. // 接收返回的数据 30. var data = xhr.responseText; 31. // 将接收的数据从字符串转为对象 32. var obj = JSON.parse(data); 33. // 执行后续逻辑 34. callback(obj); 35. } 36. } 37. // 调用open方法 38. xhr.open("get", url + "?" + query, true); 39. // 调用send方法 40. xhr.send(null); 41. } 42. } |

## 2.2 封装POST请求

|  |
| --- |
| 1. var ickt = { 2. post: function(url, data, callback) { 3. // 定义变量 4. var str = ""; 5. // 如果是对象 需要将对象转为query字符串 6. if (typeof data === "object") { 7. for (var i in data) { 8. str += i + "=" + data[i] + "&"; 9. } 10. // 将最后的&去掉 11. str = str.slice(0, -1); 12. // str原本是 a=1&b=2&c=3&d=4& 13. // 截取时 使用的下标分别是0 和 str.length - 1; 14. // 左闭右开 15. // 截取之后 a=1&b=2&c=3&d=4 16. } else if (typeof data === "string") { 17. str = data; 18. } 20. // 如果是query字符串 直接用 21. var xhr = null; 22. // 兼容性检测 23. if (window.XMLHttpRequest) { 24. xhr = new XMLHttpRequest(); 25. } else if (window.ActiveXObject) { 26. xhr = new ActiveXObject("Microsoft.XMLHttp"); 27. } else { 28. throw new Error("抱歉，您的浏览器不支持AJAX"); 29. } 30. // 设置onreadystatechange事件 31. xhr.onreadystatechange = function() { 32. // 只在最后的时候也就是状态吗为4时 才执行代码 33. if (xhr.readyState === 4 && xhr.status === 200) { 34. // 因为后续的逻辑是传递进来的 不是固定的 所以将后续的代码放在这里执行 35. // 定义变量 36. var data = xhr.responseText; 37. // 转为对象 38. var obj = JSON.parse(data); 39. callback(obj); 40. } 41. } 42. // 调用open方法 43. xhr.open("post", url , true); 44. // 将AJAX发送的内容伪装成表单 45. xhr.setRequestHeader("content-type", "application/x-www-form-urlencoded; chartset=utf-8"); 46. // 调用send方法 47. xhr.send(str); 48. } 49. } |

# 表单序列化

表单序列化：指的是将表单中的数据，按照一定的顺序排列，得到它们的query字符串。

为什么要有表单序列化：

因为表单提交的时候，有浏览器帮助将表单中的数据整理。

但是当我们使用AJAX提交整个表单数据的时候，没有任何方式能够实现该功能。需要一个元素一个元素的去获取，这样就导致强耦合。表单序列化可以解耦。

解耦原理：

因为之前的获取方式是精确到元素，导致强耦合。

现在不精确到元素，而是只定位到表单。表单里面有什么控件，就使用什么控件。所以因为与具体的控件无关，所以解耦。

类比之前的委托模式：委托模式是将事件绑定到了父元素身上去 自身元素不再拥有事件。所以删除的时候就不需要担心内存泄露。

表单序列化：因为操作的是表单元素自身，而不是表单中的具体控件。所以现在也不需要担心结构的变化会导致代码的变化。

|  |
| --- |
| 1. var ickt = { 2. serialize: function(form) { 3. // 表单传递进来 我们首先要获取到 表单中有哪些控件 4. // 表单元素有一个属性 elements 指向当前表单中的所有控件 5. // console.log(form.elements); 6. // 我们用到的只是带有name属性的哪些 那些不带name属性的我们用不到 7. // 定义一个数组 用于存放所有的带name属性的 8. var arr = []; 9. for (var i = 0; i < form.elements.length; i++) { 10. if (form.elements[i].name) { 11. arr.push(form.elements[i]); 12. } 13. } 14. // 循环结束 数组中只剩下带name的了 15. // console.log(arr); 16. // 定义字符串 17. var query = ""; 18. // 循环arr 19. for (var i = 0; i < arr.length; i++) { 20. // text、password、email、textarea 它们是可以直接拼接的 但是radio、checkbox等它们是需要额外判定是否被选中的 21. if (arr[i].type === "text" || arr[i].type === "password" || arr[i].type === "textarea") { 22. query += arr[i].name + "=" + encodeURIComponent(arr[i].value) + "&"; 23. } else if (arr[i].type === "checkbox" || arr[i].type === "radio") { 24. // 如果是checkbox或者radio 则需要判定是否被选中 25. if (arr[i].checked) { 26. // 如果选中，就拼接 27. query += arr[i].name + "=" + encodeURIComponent(arr[i].value) + "&"; 28. } 29. // 没有选中就不做事 30. } 31. } 32. console.log("自己的" + query.slice(0, -1)); 33. return query.slice(0, -1); 34. } 35. } |

# 字符串转为对象的三种方式

## 4.1 JSON

JSON是ES5中新增的对象，负责将JSON字符串与对象进行互换。

demo:

|  |
| --- |
| 1. // 定义一个对象字符串 2. var str = '{"error":1,"data":"jintianhenre"}'; 3. // 第一种方式 JSON.parse 4. var obj = JSON.parse(str); 5. console.log(obj) |

## 4.2 eval

eval是一个函数，它用于将字符串当做代码来执行。

它可以将对象格式的字符串转为对象:

|  |
| --- |
| 1. // 定义一个对象字符串 2. var str = '{"error":1,"data":"jintianhen"}'; 3. // 第二种方式 eval 4. var obj = eval("(" + str + ")"); 5. console.log(obj) |

## 4.3 Function

Function是所有函数的构造函数。可以通过它将字符串转换成代码。等代码一执行，就会产生响应的结果。

我们这里利用的是：传递一个JSON字符串，定义成代码，当代码一旦执行，就变成了“定义一个对象”。

demo:

|  |
| --- |
| 1. // 定义一个对象字符串 2. var str = '{"error":1,"data":"jintian"}'; 3. // 通过Function将str转为对象 4. var fun = new Function("return " + str); 5. // 此时该函数的等价定义方式 function anonymous() {return {"error":1,"data":"jintian"}} 6. var result = fun(); // 函数一旦执行 返回一个新定义出来的对象 7. console.log(result); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |