FormData

FormData 简单介绍

FormData是 `Ajax 2.0` -XMLHttpRequest Level 2 提供的一个接口对象，可以使用该对象来模拟和处理表单并方便的进行**文件上传**操作。

```
console.dir(FormData)
ƒ FormData()
    arguments: null
    caller: null
    length: 0
    name: "FormData"
    prototype: FormData
        append: ƒ append()
        delete: ƒ delete()
        entries: ƒ entries()
        forEach: ƒ forEach()
        get: ƒ ()
        getAll: ƒ getAll()
        has: ƒ has()
        keys: ƒ keys()
        set: ƒ ()
        values: ƒ values()
        constructor: ƒ FormData()
        Symbol(Symbol.iterator): ƒ entries()
        Symbol(Symbol.toStringTag): "FormData"
        __proto__: Object
    __proto__: ƒ ()
    [[Scopes]]: Scopes[0]
```

通过打印并查看formData的结构，可以发现该接口对象本身非常简单。在FormData构造函数原型对象上只有 `append` 、 `forEach` 、 `keys` 等少数方法。

FormData的主要用处

> ❒ 网络请求中模拟和处理表单数据
> ❒ 网络请求中用来异步的上传文件

FormData实例的创建

◆ `new FormData (form? : HTMLFormElement)`

在使用FormData构造函数创建实例对象的时候，可以传递一个HTML表单元素，该表单元素允许是任何形式的表单控件，包括文件输入框、复选框等。

HTML

```
<form name="formTest">
```

```
<form name="formTest">
    <input type="text" placeholder="请输入用户名" name="user" value="wendingding">
    <input type="password" placeholder="请输入密码" name="pass" value="123456789">
</form>
```

```
//列出创建formData实例对象的几种方式
//001 通过构造函数创建不传递任务参数
var formData1 = new FormData();   //空的实例对象

//通过调用对象的方法来设置数据(模拟表单)
//设置数据
formData1.set("name","文顶顶");
formData1.set("email","wendingding_ios@126.com");
formData1.set("friends","熊大");

//设置数据(追加)
formData1.append("friends","光头强");
formData1.append("friends","萝卜头");

//查看实例数据
formData1.forEach(function(value,key){
    console.log(key,value);
})
console.log("----------------------------------");

//002 获取表单标签传递给FormData构造函数
var formData2 = new FormData(document.forms.namedItem("formTest"))
formData2.forEach(function(value,key){
    console.log(key,value);
})
```

> **注意** 表单标签必须要设置name属性节点才能获取其数据

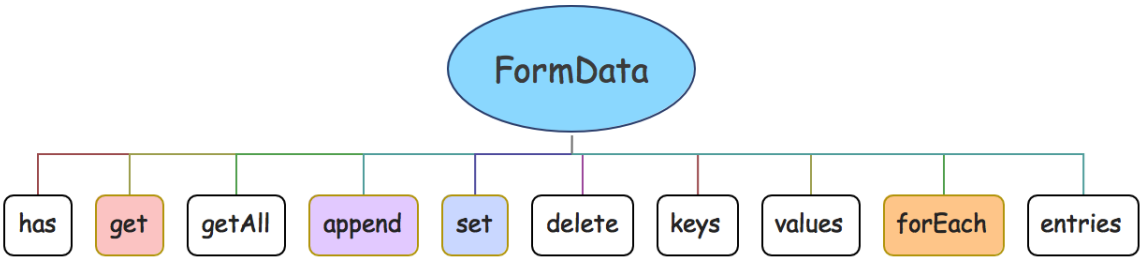
说明 在上面的示例代码中介绍了两种创建(获取)formData实例对象的方式，可以先创建一个空的实例对象也可以直接通过页面中的表单标签来进行初始化处理。

当formData数据装填好之后，可以直接通过ajax方法提交到服务器端，下面给出上面代码的执行结果。

BASH

```
name 文顶顶
email wendingding_ios@126.com
friends 熊大
friends 光头强
friends 萝卜头
----------------------------------
user wendingding
pass 123456789
```

FormData的主要方法



如上图所示，FormData构造函数的原型对象上面定义了一堆方法。这些方法使用方式都很简单，接下来我们通过代码的方式简单介绍他们。

```
//01 创建空的formData实例对象
   var data = new FormData();

   //02 设置数据(添加)
   data.set("age",18);
   data.set("name","LiuY");
   data.set("type","法师");
   data.set("address","泉水中心");
   //03 设置数据(修改和删除)
   data.set("name","MiTaoer");
   data.delete("address");
   //04 设置数据(追加)
   data.append("type","战士");
   data.append("type","辅助");

   //05 读取数据(指定key-one)
   console.log(data.get("name"));          //MiTaoer
   console.log(data.get("type"));          //法师

   //06 读取数据(指定key-All)
   console.log(data.getAll("type"));       //["法师", "战士", "辅助"]

   //07 检查是否拥有指定的key
   console.log(data.has("age"));           //true
   console.log(data.has("email"));         //false

   //08 迭代器的基本使用(keys)
   var keyIterator = data.keys()      //获取迭代器对象
   console.log(keyIterator.next());   //{done: false, value: "age"}
   console.log(keyIterator.next());   //{done: false, value: "name"}
   console.log(keyIterator.next());   //{done: false, value: "type"}
   console.log(keyIterator.next());   //{done: false, value: "type"}
   console.log(keyIterator.next());   //{done: false, value: "type"}
   console.log(keyIterator.next());   //{done: true, value: undefined}

   console.log("____________________");

   //09 迭代器的基本使用(values)
   var valueIterator = data.values();  //获取迭代器对象
   console.log(valueIterator.next());  //{done: false, value: "18"}
   console.log(valueIterator.next());  //{done: false, value: "MiTaoer"}
```

```
console.log(valueIterator.next());  //{done: false, value: "法师"}
console.log(valueIterator.next());  //{done: false, value: "战士"}
console.log(valueIterator.next());  //{done: false, value: "辅助"}
console.log(valueIterator.next());  //{done: true, value: undefined}

//10 迭代器的基本使用(entries)
console.log(data.entries().next());     //{done: false, value: ["age", "18"]}

//11 formData对象的遍历
data.forEach(function(value,key){
    //输出结果
    // age 18
    // name MiTaoer
    // type 法师
    // type 战士
    // type 辅助
    console.log(key,value);
})
```

代码说明

formData对象的这些方法其实不用进行过多的赘述，上面的代码和说明简单易懂。总体上来说，它提供了一整套的操作数据的方法囊括了添加(set)、修改、查询和删除等操作， `append` 方法和set方法的不同之处在于它不会覆盖而是以数组push的方式来处理同名的数据。

formData对象的 `keys()` 、 `values()` 和 `entries()` 方法使用类似，调用后将得到一个 `Iterator类型` 的迭代器对象，该对象能够能够调用 `next()` 方法来进行迭代操作，打印结果中的 `done` 使用布尔类型的值来进行标记，如果迭代结束那么值为**true**。

formData对象的 `forEach()` 接收一个回调函数参数，其中第一个参数为当前遍历数据的 `value` 值，第二个参数为 `key` (同数组的forEach方法一致)。如果是Ajax发送GET请求，需要通过formData对象的方式来提交表单数据，那么可以借助该方法来拼接查询字符串。

FormData的典型用法

这里给定如下的表单数据，然后介绍如何使用FormData来处理表单数据发送GET和POST请求。

HTML

```
<form name="formTest">
    <input type="text" name="user" placeholder="请输入用户名"><br>
    <input type="text" name="email" placeholder="请输入邮箱"><br>
    <input type="password" name="pass" placeholder="请输入密码"><br>
    <input type="checkbox" name="check"> 是否勾选<br>
</form>
<button>提交表单数据</button>
```

GET请求

```
//01 获取页面中的btn标签
var oBtn = document.getElementsByTagName("button")[0];

//02 给按钮标签添加点击事件
oBtn.onclick = function(){
```

```
    //03 使用Ajax发送GET请求
    var xhr = new XMLHttpRequest();
    xhr.open("GET","http://127.0.0.1:3000?"+getData(),true);
    xhr.send();
    xhr.onreadystatechange = function(){
        if(xhr.status >= 200 && xhr.status <=300 || xhr.status == 304)
        {
            console.log("请求成功"+xhr.responseText);
        }else{
            console.log("请求失败"+xhr.statusText);
        }

    }
}

//获取页面中的表单数据并处理为查询字符串
function getData(){
    var arr = [];
    var data = new FormData(document.forms.namedItem("formTest"));
    data.append("age",18);
    data.forEach(function(value,key){
        arr.push(key+"="+value);
    })
    return arr.join("&");
}
```

通过上面的代码示例可以发现，使用formData来处理表单数据发送GET请求并没有什么优势，也需要通过循环来处理然后把键值对转换为查询字符串的形式拼接在 URL字符串 的后面。

POST请求

```
//01 获取页面中的btn标签
var oBtn = document.getElementsByTagName("button")[0];

//02 给按钮标签添加点击事件
oBtn.onclick = function(){

    //03 处理参数
    //方式(1) 模拟表单数据
    var data = new FormData();
    data.set("name","文顶顶");
    data.set("color","red");
    data.set("email","yangyong@520it.com");
    data.append("email","wendingding_ios@126.com");

    //方式(2) 获取表单数据
    //var data = new FormData(document.forms.namedItem("formTest"));

    //04 使用Ajax发送GET请求
    var xhr = new XMLHttpRequest();
    xhr.open("POST","http://127.0.0.1:3000",true);
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    xhr.send(data);
    xhr.onreadystatechange = function(){
        if(xhr.status >= 200 && xhr.status <=300 || xhr.status == 304)
        {
```

```
            console.log("请求成功"+xhr.responseText);
        }else{
            console.log("请求失败"+xhr.statusText);
        }
    }
}
```

如果发送的是POST请求，那么提交表单数据需要通过 `setRequestHeader` 方法来设置 `'Content-Type', 'application/x-www-form-urlencoded'` ，而formData数据直接作为 `send` 方法的参数来进行提交即可。 `POST` 请求通过formData提交给服务器端的数据，如果是Node服务器端则很难处理(同文件一样)。formData最主要的用途其实是用来异步的进行文件上传。

POST请求进行文件上传

HTML

```
<form>
    <input type="text" name="user" id="userID">
    <input type="file" name="file-name" id="fileID" multiple>
</form>
<button>上传文件</button>
```

```
//01 获取页面中的btn标签
var oBtn    = document.getElementsByTagName("button")[0];
var oUser   = document.getElementById("userID");
var oFileID = document.getElementById("fileID");

//02 给按钮标签添加点击事件
oBtn.onclick = function(){

    //03 获取表单中的文件内容
    var data = new FormData();
    data.set("user",oUser.value);
    Array.from(oFileID.files).forEach(function(file){
        data.append("fileName",file);

    })

    //04 使用Ajax发送GET请求
    var xhr = new XMLHttpRequest();
    xhr.open("POST","http://127.0.0.1:5000/api",true);
    xhr.send(data);
    xhr.onreadystatechange = function(){
        if(xhr.status >= 200 && xhr.status <=300 || xhr.status == 304)
        {
            console.log("请求成功"+xhr.responseText);
        }else{
            console.log("请求失败"+xhr.statusText);
        }
    }
}
```

这里顺便贴出测试文件上传写的Node代码以及文件上传后的监听结果。

```
//备注：node文件名称为uploadServer.js
//01 导入模块(需先通过npm来进行安装)
var express   =   require('express');
var multer    =   require('multer');
var body      =   require('body-parser');

var app = express();
app.listen(5000);
app.use(body.urlencoded( { extended: false } ));
app.use(multer( { dest: './upload/' } ).any());

//02 监听网络请求并设置打印接收到的参数信息
app.post('/api', function (req,res){

    res.setHeader('Access-Control-Allow-Origin', '*');
    res.send("Nice ! 上传成功 ~ ");

    console.log(req.body);      //普通POST数据
    console.log(req.files);     //文件POST数据

});
app.use(express.static('./html/'));
```

代码说明 需要先通过 `npm install express multer body-parser` 命令在当前路径中安装对应的模块。

BASH

```
wendingding$ node uploadServer.js
{ user: 'wen' }
[ { fieldname: 'fileName',
    originalname: 'formData.png',
    encoding: '7bit',
    mimetype: 'image/png',
    destination: './upload/',
    filename: 'f416da3b522ece9e4cc2eccd5b7a62e8',
    path: 'upload/f416da3b522ece9e4cc2eccd5b7a62e8',
    size: 50002 },
  { fieldname: 'fileName',
    originalname: 'Snip20190107_1.png',
    encoding: '7bit',
    mimetype: 'image/png',
    destination: './upload/',
    filename: '2a2dd60e217b9cc08f2cc0048a1d27ab',
    path: 'upload/2a2dd60e217b9cc08f2cc0048a1d27ab',
    size: 1309894 } ]
```