HTML5新特性 -- 拖放及本地存储

1.Multer

1.1 自定义存储规则

通过Multer的diskStorage()方法可定义存储规则，其语法结构是：

```
var storage = multer.diskStorage({
    destination:function(req,file,cb){
        ...
    },
    filename:function(req,file,cb){
        ...
    }
});
```

> destination属性用于定义上传目录的相关规则，如可通过Node.js内置的fs模块对上传文件目录进行实时操作，示例代码如下：
>
> ```
> var storage = multer.diskStorage({
>  //上传时目录的相关规则
>  destination: function (req, file, cb) {
>      //构建Date()对象,作为目录名称
>      var now = new Date();
>      var fullYear = now.getFullYear();
>      var month = now.getMonth() + 1;
>      month = month < 10 ? '0' + month : month;
>      var day = now.getDate();
>
>      //构建目录名称
>      var path = 'upload/' + fullYear + '-' + month + '-' + day;
>
>      //判断目录是否存在,如果不存在则自动创建
>      if (!fs.existsSync(path)) {
>          fs.mkdirSync(path, (err) => {
>              if (err) throw err;
>          });
>      }
>
>      cb(null, path);
>  },
>  //上传时文件名称的相关规则
>  filename: function (req, file, cb) {
>      ...
>  }
> });
> ```

`filename`属性用于定义上传文件名称的相关规则，在实际使用时**"必须"**来修改上传文件的名称（1.`Multer`生成的文件名称不带有扩展名 2.实际使用时要保证文件名称的唯一性），如果要生成带有扩展名的文件名称，那么必须要知道原始的文件扩展名，故就需获取上传文件的相关信息（参见`req.file`及`req.files`）

为保证上传文件名称的唯一性，建议使用`uuid`（参见`UUID`）,示例代码如下：

```
var storage = multer.diskStorage({
 //上传时目录的相关规则
 destination: function (req, file, cb) {
   ...
 },
 //上传时文件名称的相关规则
 filename: function (req, file, cb) {
     //获取原始文件的名称
     var origin = file.originalname;

     //获取文件的扩展名
     var extension = origin.substr(origin.lastIndexOf('.') + 1);
     extension = extension.toLowerCase();

     //生成主文件名 -- 生成基于时间戳的UUID
     var main = uuid.v1();

     //生成新的文件名称
     filename = main + '.' + extension;

     cb(null, filename);
 }
});
```

1.2 `req.file`及`req.files`

`req.file`返回单文件上传时的文件信息对象

`req.files`返回多个文件上时文件信息的数组

文件信息包括：

- `originalname`，文件的原始名称
- `size`，文件字节数（以字节为单位）
- `mimetype`，文件的`MIME`类型
- `filename`，上传后的文件名称
- `destination`，文件的保存路径
- `path`，文件完整的保存路径
- `fieldname`，表单控件名称

1.3 Multer与拖放上传

1.3.1 概述

拖放上传的基本实现原理是：将上传的文件拖放到HTML页面的指定位置，然后通过AJAX上传

在通过拖放实现上传时，需要注意：

• 在HTML页面中不存在<form>元素（参见FormData对象）

• 在HTML页面中不存在<input type="file">元素

针对在HTML页面中不存在<input type="file">元素的情况可通过DataTransfer对象files属性来实现，该属性将返回FileList对象，其语法结构是：

```
FileList DataTransfer.files
```

1.3.2 FileList对象

FileList对象通常来自HTML表单内的<input type="file">元素，也可能来自用户的拖放操作。

length属性

length属性将返回FileList对象中包含的文件数量，其语法结构是：

```
variable = FileList.length
```

1.3.3 FormData对象

FormData对象提供用键/值对表示的表单数据的方式，经过它的数据可以使用AXAJ提交。

创建FormData对象

```
variable = new FormData()
```

append()方法

append()方法用于添加一个新值到已有的键名上，如果键名不存在，由自动创建，其语法结构是：

```
FormData.append(name,value)
```

1.3.4 axios控制上传进度

可以通过axios的onUploadProgress事件属性对于上传进度进行控制，其语法结构是：

```
axios.post(url,data,{
    onUploadProgress:function(progressEvent){
        //对原生进行事件进行处理
    }
});
```

`progressEvent`接口是测量HTTP请求底层流程进度的事件，该接口继承自`Event`接口

`lengthComputable`属性

`lengthComputable`属性用于获取进度是否可以被测量，其语法结构是：

```
boolean ProgressEvent.legnthComputable
```

`total`属性

`total`属性用于获取正在执行底层流程的工作总量，其语法结构是：

```
variable = ProgressEvent.total
```

`loaded`属性

`loaded`属性用于获取经执行的底层流程的工作总量，其语法结构是：

```
variable = ProgressEvent.loaded
```

2. `uuid`

2.1 概述

`UUID`（`Universally Unique Indentifier`）,通用唯一标识符，其目的是为了让分布式系统中所有的元素都能存在唯一的标识信息。

`UUID`是由一组32位数的16位进制数字组成，以连字号分为五段，形式为`8-4-4-4-12`

```
npm install --save uuid
```

2.2 方法

- **`v1()`方法**

`v1()` 方法用于生成基于时间戳的 `UUID`，其语法结构是：

```
variable = uuid.v1()
```

- **`v4()` 方法**

`v4()` 方法用于生成基于随机数的 `UUID`，其语法结构是：

```
variable = uuid.v4()
```

3.`Web Storage`

3.1 概述

`web storage` 提供了一种比 `cookie` 更加直观方式来存储数据，其结构为键/值对。

`web storage` 中提供两种机制：

1、`sessionStorage`，为每个指定的源维护一个独立的存储区域，该存储区域内的数据在页面会话期间可用（浏览器处理要开状态）

2、`localStorage`，为每个指定的源维护一个独立的存储区域，该存储区域内的数据即使在浏览器关闭后，仍然存在。

这两种机制可以通过 `window.sessionStorage` 和 `window.localStorage` 进行访问。

3.2 方法

- **`setItem()`**

`setItem()` 方法用于将键名添加到存储中，如果键名已经存在，则更新其值，语法结构是：

```
storage.setItem(key,value)
```

- **`getItem()`**

`getItem()` 方法用于返回对指定键名的值，其语法结构是：

```
variable = storage.getItem(name)
```

- **`removeItem()`**

`removeItem()` 方法用于删除指定的键名，如果键名不存在，则不执行任何操作，其语法结构是：

```
storage.removeItem(key)
```

- `clear()`

`clear()`方法用于清除所有的键名，其语法结构是：

```
storage.clear()
```

- `key()`

`key()`用于返回指定索引值的键名，但需要注意的是：键名的顺序由浏览器决定！语法结构是：

```
variable = storage.key(index)
```

3.3属性

- `length`

`length`属性用于返回数据项的数量，其语法结构是：

```
variable = storage.length
```

4.再谈`VUEX` -- 辅助函数

- `mapState`

`mapState()`函数用于为组件创建**计算属性**以返回`Vuex Store`中的状态，其语法结构是：

```
import {mapState} from 'vuex'

computed:{
    ...mapState(array | object)
}
```

- `mapGetters()`

`mapGetters()`函数用于为组件创建**计算属性**以返回`getters`的返回值，其语法结构是：

```
import {mapState} from 'vuex'

computed:{
    ...mapGetters(array|object)
}
```

- `mapMutations()`

`mapMutations()`函数用于创建**组件方法**以提交`Mutation`，其语法结构是：

```
import {mapMutations} from 'vuex';

methods:{
    ...mapMutations(array | object)
}
```

- `mapActions()`

`mapActions()`函数用于创建**组件方法**以分发`Action`，其语法结构是：

```
import {mapActions} from 'vuex';

methods:{
    ...mapActions(array | object)
}
```