# 实验三 HTML5拖放API项目

# 说明

- 本次实验主要包含了两个基于HTML5拖放API的应用设计实例，一是仿回收站效果的设计与实现（理论课上已布置过，选做），二是图片相框展示的设计与实现（必做）。
- 在仿回收站项目中，主要难点为元素的拖拽以及回收效果；
- 在图片相框展示项目中，主要难点为文件的拖拽、自动生成带有相框的图片以及显示图片文件信息的技术。

# 目录

# 3.1 仿回收站效果的设计与实现

背景介绍：在Windows等操作系统中均包含回收站功能，用户可以直接将不需要的文件拖拽并放置到桌面回收站图标上以实现文件删除。

# 3.1 仿回收站效果的设计与实现

功能要求：使用**HTML5**拖放**API**相关技术，在网页上实现仿回收站的类似效果。用户通过拖拽可以将页面上的元素放置到回收站中删除。效果图如图**3-1**所示。

图(a) 页面初始加载效果

图(b) 拖动文件 2 的过程

图(c) 文件 2 被删除后的效果

图 3-1 仿回收站效果示意图

# 3.1 仿回收站效果的设计与实现

## 3.1.1 界面设计核心技术概括

- 使用 标签划分区域
  - 文件展示区域
  - 回收站区域
- 使用 标签制作示例文件
  - 4个示例文件

# 3.1 仿回收站效果的设计与实现

## 3.1.2 文件拖拽与回收功能的实现

- 文件拖拽的实现

```
1.  <div class="folder" draggable="true" >文件1</div>
```

- 将回收站设置为可放置区域

```
1.  <div id="recycle" ondragover="allowDrop(event)" ></div>
```

```
1.   //ondragover事件回调函数
2.     function allowDrop(ev) {
3.         //解禁当前元素为可放置被拖拽元素的区域
4.         ev.preventDefault();
5.     }
```

# 3.1 仿回收站效果的设计与实现

## 3.1.2 文件拖拽与回收功能的实现cont.

- 回收功能的实现

```
1.  <div id="file1" class="folder" draggable="true" ondragstart="drag(event) " >文件1</div>
```

```
1.  //ondragstart事件回调函数
2.    function drag(ev) {
3.        //设置传递的内容为被拖拽元素的id名称，数据类型为纯文本类型
4.        ev.dataTransfer.setData("text/plain", ev.target.id);
5.    }
```

# 3.1 仿回收站效果的设计与实现

## 3.1.2 文件拖拽与回收功能的实现cont.

- 回收功能的实现

```
1.  <div id="recycle" ondragover="allowDrop(event)" ondrop="drop(event)" ></div>
```

```
1.      //ondrop事件回调函数
2.      function drop(ev) {
3.          ev.preventDefault(); //解禁当前元素为可放置被拖拽元素的区域
4.          var id = ev.dataTransfer.getData("text"); //获取当前被放置的元素id名称
5.          var folder = document.getElementById(id); //根据id名称获取元素对象
6.          //获取文件夹区域并删除该元素对象
7.          document.getElementById("container").removeChild(folder);
8.      }
```

# 3.1 仿回收站效果的设计与实现

## 3.1.2 文件拖拽与回收功能的实现cont.

- 以文件2为例，该文件被拖拽到回收站区域并删除的运行最终结果如下图所示。

# 3.2 图片相框展示的设计与实现

背景介绍：目前市面上一些修图工具软件带有自动为图片添加不同款式的相框功能，用户可以选择本地图片文件然后为其添加相框效果。

# 3.2 图片相框展示的设计与实现

功能要求：使用**HTML5**拖放**API**相关技术，在网页上实现为指定图片自动生成图片相框的效果。用户通过拖拽可以将本地的图片文件放置到页面上指定区域，即可在页面上自动生成带有相框效果的图片展示。效果图如图所示。

图(a) 页面初始加载效果

图(b) 本地图片文件的拖拽

图(c) 自动生成相框效果

# 3.2 图片相框展示的设计与实现

## 3.2.1 界面设计

- 使用 标签划分区域
  - 本地文件放置区域
  - 带有相框图片的展示区域
- CSS文件辅助渲染样式
  - 自定义外部样式表photoframe.css文件

# 3.2 图片相框展示的设计与实现

## 3.2.2 相框自动生成功能的实现

- 1. 可放置区域的实现

```
1.  <div id="recycle" ondragover="allowDrop(event)" >请将图片拖放至此处</div>
```

```
1.  //ondragover事件回调函数
2.      function allowDrop(ev) {
3.          //解禁当前元素为可放置被拖拽元素的区域
4.          ev.preventDefault();
5.      }
```

# 3.2 图片相框展示的设计与实现

## 3.2.2 相框自动生成功能的实现

- 2. 生成带有相框的图片效果

```
1.  <div id="recycle" ondragover="allowDrop(event)" ondrop="fileDrop(event)"  >
```

```
1.   function fileDrop(e) {
2.        …
3.        //清空上一次图片内容
4.        //获取从本地拖拽放置的文件对象数组files
5.        //使用for循环遍历同时被拖拽并放置到指定区域的所有文件
6.        }
```

# 3.2 图片相框展示的设计与实现

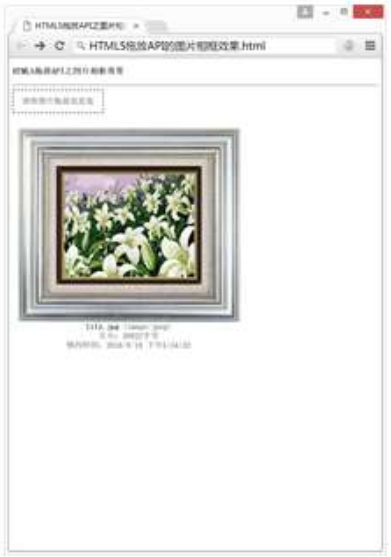
## 3.2.2 相框自动生成功能的实现

- 3. 显示图片文件信息
  - 关键代码如下：

```
1.  //获取当前文件的最新修改日期
2.          var lastModified = f.lastModifiedDate;
3.          //修改当前文件的最新修改日期的描述格式
4.          var lastModifiedStr = lastModified ? lastModified.toLocaleDateString() + ' ' +
    lastModified.toLocaleTimeString() : 'n/a';
5.          //设置图片下方状态信息栏描述内容
6.          status.innerHTML = '<strong>' + f.name + '</strong> (' + (f.type || 'n/a') + ')<br>
    大小：' + f.size + '字节<br>修改时间：' + lastModifiedStr;
```

# 3.2 图片相框展示的设计与实现

- 最终效果如下图所示。



图(a) 单个图片文件的显示效果

图(b) 多个图片文件的显示效果