tree/js/main.js中
```
$("#alternate_upload").change(function (event){
      notebook_list.handleFilesUpload(event,'form');
   });
```

tree/js/notebooklist.js中
可以在HandleFilesUpload()中看到在读取文件完成的回调函数onload中，保存结果至item 并调用add_upload_button()，添加一个upload按钮。
这个按钮注册了一大段点击事件
分为两种上传方式

- ipynb类型文件（略过）
- 普通文件
    - model.type = 'file';
    - model.format = format;
    - model.content = filedata;
    - content_type = 'application/octet-stream';

filedata是怎么来的呢
在HandleFilesUpload中创建了一个FileReader的实例reader 对于普通文件 reader.readAsArrayBuffer(f);（结果通过回调函数onload返回）

判断文件名是否重复等一系列问题后，最终上传语句that.contents.save(path, model).then(on_success);
可以看到on_success()中有重新载入列表的动作
contents是怎么来的呢，回到main.js 最开始有
requirejs(['contents'], function(contents_service)
又回到了熟悉的static/services/contents.js中了 有个Contents.prototype.save()函数

```
      var settings = {
         processData : false,
         type : "PUT",
         dataType: "json",
         data : JSON.stringify(model),
         contentType: 'application/json',
      };
      var url = this.api_url(path);
      return utils.promising_ajax(url, settings);
```

我们再来看对应url的handlers怎么处理put请求 在services/contents/handlers.py中
来看ContentHandlers中put()函数的说明

```
      Saves the file in the location specified by name and path.

      PUT is very similar to POST, but the requester specifies the name,
      whereas with POST, the server picks the name.

      PUT /api/contents/path/Name.ipynb
        Save notebook at ``path/Name.ipynb``. Notebook structure is specified
        in `content` key of JSON request body. If content is not specified,
        create a new empty notebook.
```

put中获取json对象 通过私有方法_upload调用contents_manager.new()方法创建出新文件
contents_manager是filemanager.py中FileContentsManager对象的实例 而new()实际上又是创建出一个新文件之后调用save
save最终由fileio.py中的FileManagerMixind的_save_file实现（编码转换后，简单地fwrite实现）