# 一、整体思路说明

在ECCentral中，对Portal端上传文件到Service端，做了一个统一的上传处理模块，来进行文件上传管理，这个模块里包括了Portal端组件和控件，以及Service端的组件；

          这个模块的整体思路是：Portal通过一个组件工具类（或者直接用控件）来上传一个文件到Service端那边，上传成功后Portal端会得到一个该文件在Service端的唯一标识（就一个字符串），然后Portal端再发起一次对自己Domain的Restful Service的请求调用，把刚刚上传文件所得到的文件的唯一标识字符串传到自己的Restful Service里；在Restful Service里则通过Service端的一个组件工具类，根据Portal传来的文件的唯一标识，就可以取到刚刚上传的文件，然后做后续的处理（解析、转换、拷贝、更新数据库等等）；如下图所示：



在这个上传处理的体系中，对于文件上传和管理的部分则是公用和统一的（上图中绿色底色的对象模组），而Service端Restful Service则是根据具体需求来完成具体的Service的代码实现；这样对于不同的需求，但是上传文件的部分得以公用了。

# 二、具体使用说明

关于这个上传体系，其具体的使用说明如下：

1. **Portal端上传组件**

首先来看Portal端，提供了一个ECCentral.Portal.Basic.Utilities.FileUploadClient类，用来异步上传文件，通过该类可以多线程（建议设置线程不要超过2个，因为.net对http并发请求做了2个的限制，暂时还没找到Silverlight中破解这个限制的办法，设超过2个线程也不会提升上传速度了）来上传文件，支持暂停后接着续传以及出错后续传，支持上传过程中取消上传；

FileUploadClient类有5个public事件、4个public方法和4个public构造函数

大概的使用代码如下：

|  |
| --- |
| private void Button\_Upload\_Click(object sender, RoutedEventArgs e)          {              FileUploadClient client = new FileUploadClient(ConstValue.DomainName\_Customer, "D:\\Study\\test.avi", 2);              client.UploadProgressChanged += (se, args) =>              {                  long t = args.TotalUploadedDataLength;  // 已经上传的数据大小                  double percent = (double)t / args.FileInfo.Length;   // 上传数据的百分比                  // ….. 更新UI进度条等代码              };              client.UploadErrorOccured += (se, args) =>              {                  // 这里可以弹出一个提示框告诉客户上传过程中出错了，上传已自动暂停了；                  // 后续可以调用FileUploadClient.ResumeUpload()方法接着断点继续上传；                  this.Window.Alert("Error!", MessageType.Error);              };              client.UploadCompleted += (se, args) =>              {                  // 上传成功，通过args.**FileIdentity**就可以拿到这次说上传文件的唯一标识，                  // 然后在这里再用RestClient来调用Restful Service，args.**FileIdentity**把传过去                  ……              };              client.UploadCanceled += (se, args) =>              {                  this.Window.Alert("Canceled!", MessageType.Information);  // 上传取消成功后触发的事件              };              client.UploadSuspended += (se, args) =>              {                  this.Window.Alert("Suspended!", MessageType.Information); // 上传暂停成功后触发的事件              };              client.Upload();  // 上传方法          } |

上面的代码中可以看到关于UploadProgressChanged、UploadErrorOccured、UploadCompleted、UploadCanceled和UploadSuspended五个事件的具体使用；

对于FileUploadClient类除了**Upload()方法**以外，还有

* **SuspendUpload() 方法**：请求暂停本次上传，后面可以通过ResumeUpload方法来恢复接着断点续传（方法返回后并不会立即暂停，因为从请求暂停 --》 已经暂停 中间会有些时间，需要异步线程的reqeust都结束完毕请求，还涉及到可能有多个线程在并发，所以需要所有异步线程的reqeust都结束完毕请求，所以也提供了UploadSuspended的事件来表示真正暂停成功了）;
* **ResumeUpload() 方法**：暂停 或者 出错后继续接着断点上传，方法返回后上传的异步线程就已经开启执行了;
* **CancelUpload() 方法**：请求取消本次上传，取消后无法再恢复接着传了，只能重头开始再传（方法返回后并不会立即取消，道理同上面的SuspendUpload方法，所以也提供了UploadCanceled的事件来表示真正已经取消成功了）;

1. **Portal端上传控件**

Portal端还提供一个封装好了的上传控件，在大部分情况下是可以直接使用这个控件的，只是对于上传UI需要复杂定制化的，比如商品图片上传，就需要直接用第一点里提到的FileUploadClient类，然后自己灵活的处理UI显示了；

封装好了的上传控件为ECCentral.Portal.Basic.Controls.Uploader.FileUploader，放在ECCentral.Portal.Basic工程里，支持上传多个文件，采用异步上传方式，上传完毕后将会触发其AllFileUploadCompleted事件，在该事件的处理方法里，可以通过事件参数知道每一个文件的上传成功与否（args.UploadInfo[i].UploadResult），以及每一个文件在Service端的唯一标识（args.UploadInfo[i].ServerFilePath）,这样就可以继续请求相关的Restful Service，根据该文件标识进行后续的相关处理动作了；

1. **Service端的文件上传管理组件**

Service端会提供一个静态类：ECCentral.Service.Utility.FileUploadManager，该类提供如下方法：

|  |
| --- |
| public static bool FileExists(string fileIdentity)  public static string GetFileExtensionName(string fileIdentity)  public static bool MoveFile(string fileIdentity, string destinationFileName)  public static FileStream OpenFile(string fileIdentity)  public static FileStream OpenFile(string fileIdentity, FileMode mode, FileAccess access, FileShare share)  public static byte[] GetFileData(string fileIdentity)  public static void DeleteFile(string fileIdentity) |

在自己的Service端代码里就可以使用上面的静态类的静态方法（建议是在AppService层），根据Portal端传过来的FileIdentity（前面上传生成的），然后来获得文件（可以获得文件路径、文件流、文件byte[]数据、文件后缀名等，方便各种处理的需要），做自己的业务处理，最后还可以调用DeleteFile来删除已使用完毕、没有用的文件；

注意：这里没有直接提供返回文件路径的方法，如果需要文件物理地址（比如需要把文件地址保存到数据库里以供后续使用的），那么可以通过获取文件后缀名的方法GetFileExtensionName和MoveFile的方法，来构建一个新的文件地址，将上传的文件Move到新地址，这样做的目的，是不希望把文件长期留在上传目录里（比如防止有人直接将上传目录里的文件路径地址写入数据库保存，否则有人这样做了就等于把上传目录当成一个长期保存文件的地方了）；不希望文件长期留在上传目录的原因，则是因为上传目录里的文件是有过期时间，比如每天（或周或月）清理一次，把一天（或周或月）前上传的文件统统都删除掉，这样保证垃圾数据都会定期被清除；

1. **配置说明**

只需要在Service端需要配置一个HttpHandler：

其type为：ECCentral.Service.Utility.FileUploadHandler, ECCentral.Service.Utility

其path为：FileUpload.ashx（这个必须为该路径，因为Portal端上传组件中是写死的该页面路径）