在Asp[**.NET**](http://lib.csdn.net/base/dotnet)的WEBform中，上传文件与下载文件处理是很简单的事情，如果转为ASP[**.Net**](http://lib.csdn.net/base/dotnet) MVC呢？那就没有那么容易了，难少少，也不是很难，一起来看下本文吧。本文主要讲如何在Asp.net MVC中上传文件，然后如何再从服务器中把上传过的文件下载下来.

在Web Forms中,当你把一个FileUpload控件拖到设计器中，你或许没有注意到在生成的HTML中会在form标签中加入一条额外属性enctype="multipart/form-data". 而FileUpload控件本身会生成为<input type=”file” />，在MVC的view里，有许多种方法可以做到同样效果，第一种的HTML如下:

<form action="/" method="post" enctype="multipart/form-data">  
  <input type="file" name="FileUpload1" /><br />  
  <input type="submit" name="Submit" id="Submit" value="Upload" />  
</form>

注意form标签已经包括了enctype标签，而method属性则设为”post”,这样设置并不多于因为默认的提交时通过HTTP get方式进行的。下面这种方式，使用Html.BeginForm()扩展方法，会生成和上面同样的HTML：

<%  
  using (Html.BeginForm("", "home", FormMethod.Post, new {enctype="multipart/form-data"}))   
   {%>   
     <input type="file" name="FileUpload1" /><br />  
     <input type="submit" name="Submit" id="Submit" value="Upload" />  
<% }%>

注意<input type=”file”>标签的name属性，我们在后面再讨论，上面代码会如下图:

OK,现在我们可以浏览本地文件然后通过Upload提交按钮将文件提交到服务器端，下一步就是在服务器端处理上传的文件，在使用fileUpload控件时，你可以很轻松的通过FileUpload的hasFile方法来查看文件是否被上传。

但是在Asp.net MVC中貌似就不是这么方便了，你会和原始的HTTP更接近一些，然而，一个扩展方法可以处理这些:

public static bool HasFile(this HttpPostedFileBase file)  
{  
  return (file != null && file.ContentLength > 0) ? true : false;  
}

当你看到对应的Controller类的代码时，你会发现Request对象作为HttpRequestBase类型的一个属性存在。HttpReuqestBase其实是HTTP请求的一个封装，暴漏了很多属性，包括Files collection（其实是HttpFileCollectionBase的集合）,在集合中的每一个元素都是HttpPostedFileBase的集合，扩展方法是用于确保上传的文件是否存在。实际上，这和FileUpload.HasFile()方法的工作原理一致。

在Controller Action中使用起来其实很容易:

public class HomeController : Controller  
{  
  public ActionResult Index()  
  {  
    foreach (string upload in Request.Files)  
    {  
      if (!Request.Files[upload].HasFile()) continue;  
      string path = AppDomain.CurrentDomain.BaseDirectory   "uploads/";  
      string filename = Path.GetFileName(Request.Files[upload].FileName);  
      Request.Files[upload].SaveAs(Path.Combine(path, filename));  
    }  
    return View();  
  }  
}

**多文件上传**或许你已经比我更早的想到如何更好的将Request.Files作为一个集合使用。这意味着它不仅仅只能容纳一个文件，而能容纳多个，我们将上面的View改为如下：

<%  
  using (Html.BeginForm("", "home", FormMethod.Post, new {enctype="multipart/form-data"}))   
   {%>   
     <input type="file" name="FileUpload1" /><br />  
     <input type="file" name="FileUpload2" /><br />  
     <input type="file" name="FileUpload3" /><br />  
     <input type="file" name="FileUpload4" /><br />  
     <input type="file" name="FileUpload5" /><br />  
     <input type="submit" name="Submit" id="Submit" value="Upload" />  
<% }%>

效果如下:

在Controller的代码中已经检查了是否所有的文件上传框中都有文件，所以即使对于多文件上传，我们也不再需要修改Controller的代码，注意每一个<input type=”file”>都有不同的name属性,如果你需要调用其中一个，比如说，你需要引用第三个输入框只需要使用:Request.Files["FileUpload3"]. （me：使用文件名来引用）

存入**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**  
在你冲我狂吼”关注点分离”之前，我想声明下面的代码仅仅用于作为说明功能.我将ADO.Net的代码放入Controller action中，但我们都知道，这并不好。数据访问的代码应该放在Model中某个部分的数据访问层中.但是，下面这段代码仅仅可以给大家怎样将上传的文件存入数据库中一个更直观的印象，首先，我们需要创建一个数据表(FileTest)并创建一个表:FileStore

CREATE TABLE [dbo].[FileStore](  
[ID] [int] IDENTITY(1,1) NOT NULL,  
[FileContent] [image] NOT NULL,  
[MimeType] [nvarchar](50) NOT NULL,  
[FileName] [nvarchar](50) NOT NULL  
) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

FileContent域是image数据类型，用于存储以二进制数据形成的文件，而Index Action改为：

public ActionResult Index()  
{  
  foreach (string upload in Request.Files)  
  {  
    if (!Request.Files[upload].HasFile()) continue;  
  
    string mimeType = Request.Files[upload].ContentType;  
    Stream fileStream = Request.Files[upload].InputStream;  
    string fileName = Path.GetFileName(Request.Files[upload].FileName);  
    int fileLength = Request.Files[upload].ContentLength;  
    byte[] fileData = new byte[fileLength];  
    fileStream.Read(fileData, 0, fileLength);  
  
    const string connect = @"Server=./SQLExpress;Database=FileTest;Trusted\_Connection=True;";  
    using (var conn = new SqlConnection(connect))  
    {  
      var qry = "INSERT INTO FileStore (FileContent, MimeType, FileName) VALUES (@FileContent, @MimeType, @FileName)";  
      var cmd = new SqlCommand(qry, conn);  
      cmd.Parameters.AddWithValue("@FileContent", fileData);  
      cmd.Parameters.AddWithValue("@MimeType", mimeType);  
      cmd.Parameters.AddWithValue("@FileName", fileName);  
      conn.Open();  
      cmd.ExecuteNonQuery();  
    }  
  }  
  return View();  
}

修改后的代码会以循环的方式遍历Web页面中所有的上传文件，并检查<input type=”file”>中是否已经加入文件，然后，从文件中提取出3个信息：文件名，MIME类型(文件的类型)，HTTP Request中的二进制流。二进制数据被转换为byte数组，并以image数据类型存入数据库。MIME类型和文件名对于用户从数据库中提取文件来说非常重要。

将数据库中的文件返回给用户：

你如何将文件传送给用户取决于你最开始如何存储它，如果你将文件存入数据库，你会用流的方式将文件返还给用户，如果你将文件存在硬盘中，你只需要提供一个超链接即可，或者也可以以流的方式。每当你需要以流的方式将文件送到浏览器中，你都的使用到File()方法的重载（而不是使用我们先前一直使用的View()方法）,对于File()方法有3类返回类型：FilePathResult,FileContentResult和FileStreamResult,第一种类型用于直接从磁盘返回文件；第二种类型用于将byte数组返回客户端；而第三种方式将已经生成并打开的流对象的内容返回客户端。

如果你还记得的话，我们将上传的文件存入了数据库，并以byte数组的形式存入FileContent域内.而当需要提取时，它仍然会以一个byte数组进行提取，这意味着我们使用返回FileContentResult的File()重载,如果我们想让提取的文件名更有意义，我们使用接受3个参数的重载，三个参数是:byte数组，MIME类型，文件名:

public FileContentResult GetFile(int id)  
{  
  SqlDataReader rdr; byte[] fileContent = null;   
  string mimeType = "";string fileName = "";  
  const string connect = @"Server=./SQLExpress;Database=FileTest;Trusted\_Connection=True;";  
  
  using (var conn = new SqlConnection(connect))  
  {  
    var qry = "SELECT FileContent, MimeType, FileName FROM FileStore WHERE ID = @ID";  
    var cmd = new SqlCommand(qry, conn);  
    cmd.Parameters.AddWithValue("@ID", id);  
    conn.Open();  
    rdr = cmd.ExecuteReader();  
    if (rdr.HasRows)  
    {  
      rdr.Read();  
      fileContent = (byte[])rdr["FileContent"];  
      mimeType = rdr["MimeType"].ToString();  
      fileName = rdr["FileName"].ToString();  
    }  
  }  
  return File(fileContent, mimeType, fileName);  
}

在View中最简单的使用来使用这个Action只需提供一个超链接：  
<a href="/GetFile/1">Click to get file</a>  
如果在数据库中存储的图片是图片类型，和使用超链接不同的是，我们通过指向Controller action的一个带有src属性的<image>标签来获取:  
<img src="/GetFile/1" alt="My Image" />

下面再让我们来看看使用FilePathResult（用于从硬盘提取文件）是多简单的事：

public FilePathResult GetFileFromDisk()  
{  
  string path = AppDomain.CurrentDomain.BaseDirectory   "uploads/";  
  string fileName = "test.txt";  
  return File(path   fileName, "text/plain", "test.txt");  
}

而这也可以用过超链接提取:

<a href="/GetFileFromDisk">Click to get file</a>

而最后一个选择FileStreamResult也可以从磁盘中提取文件:  
public FileStreamResult StreamFileFromDisk()

{  
  string path = AppDomain.CurrentDomain.BaseDirectory   "uploads/";  
  string fileName = "test.txt";  
  return File(new FileStream(path   fileName, FileMode.Open), "text/plain", fileName);  
}

FilePathResult和FileStreamResult的区别是什么？我们又该如何取舍呢？主要的区别是FilePathResult使用HttpResponse.TransmitFile来将文件写入Http输出流。这个方法并不会在服务器内存中进行缓冲，所以这对于发送大文件是一个不错的选择。他们的区别很像DataReader和DataSet的区别。于此同时， TransmitFile还有一个bug,这可能导致文件传到客户端一半就停了，甚至无法传送。而FileStreamResult在这方面就很棒了。比如说：返回Asp.net Chart 控件在内存中生成的图表图片，而这并不需要将图片存到磁盘中.