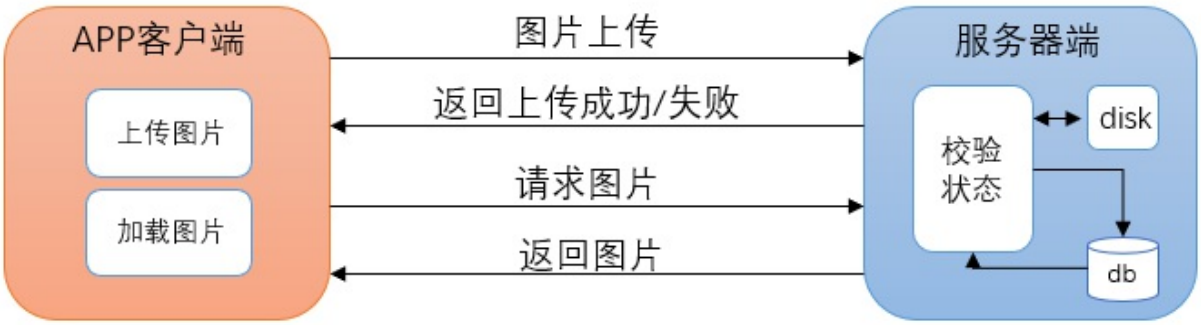
服务器接收客户端图片上传，并保存在硬盘中

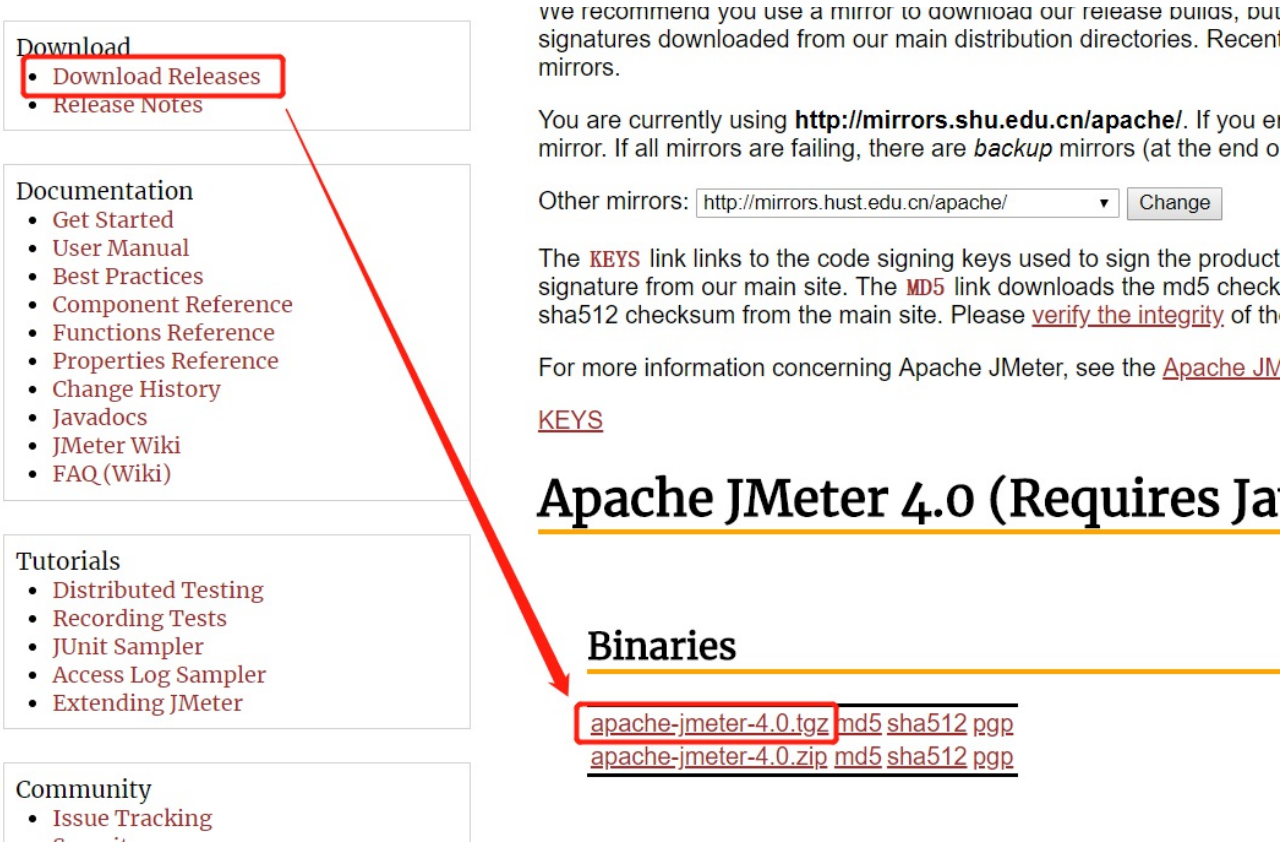
前面几小节，我们已完成了 JSON 格式的纯数据交互，在 App 服务器端的设计中，我们难免会接收客户端图片的上传，并提供端图片下载。本小节将讲解，对于客户端向服务器端上传图片，服务器端将如何处理。简单交互过程如下。



同样，在这一小节中，我们也使用工具来代替 App 客户端模拟图片的上传。我们将要用到的工具是 JMeter，它是一个强大的工具，最为熟知的是 HTTP 的测试。这里我们不去深入了解 JMeter，而只是取其一个小功能 —— HTTP POST 图片的功能来完成讲解，读者如果感兴趣，可以自行学习拓展。

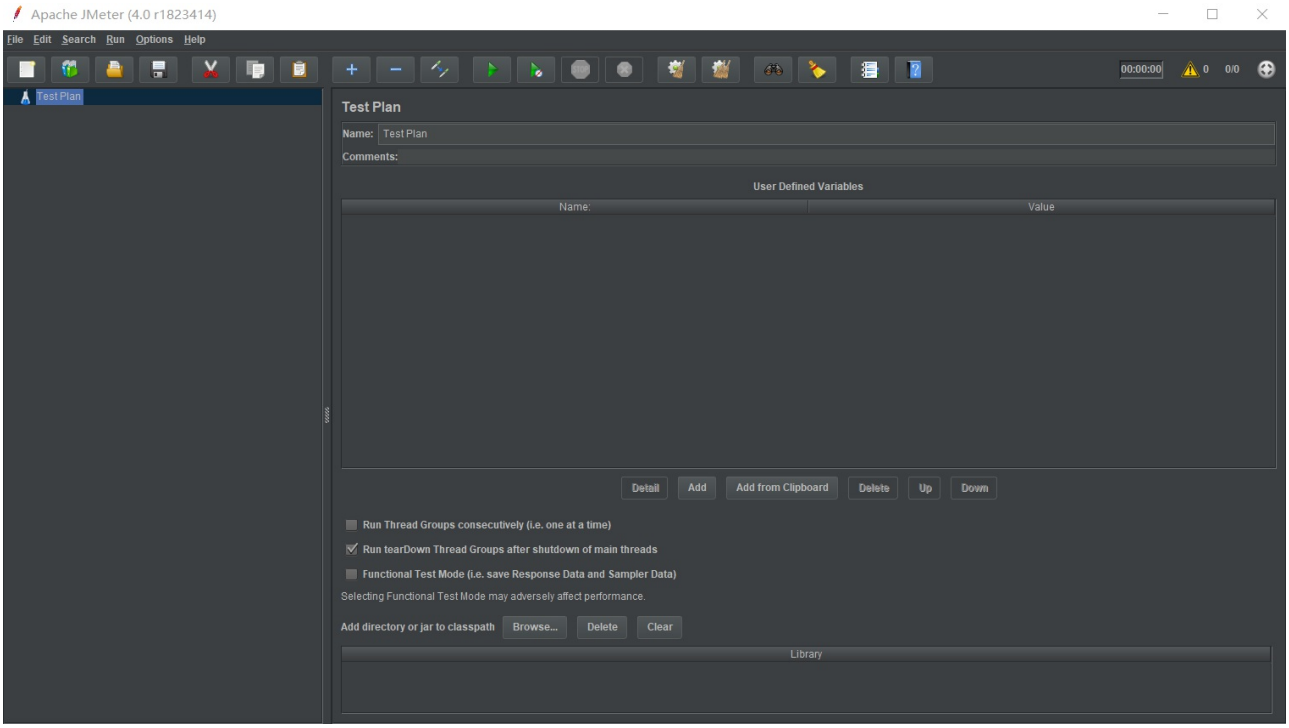
下载 JMeter

通过官网下载 JMeter：[Download Apache JMeter (http://jmeter.apache.org/download_jmeter.cgi)](http://jmeter.apache.org/download_jmeter.cgi)



安装 JMeter

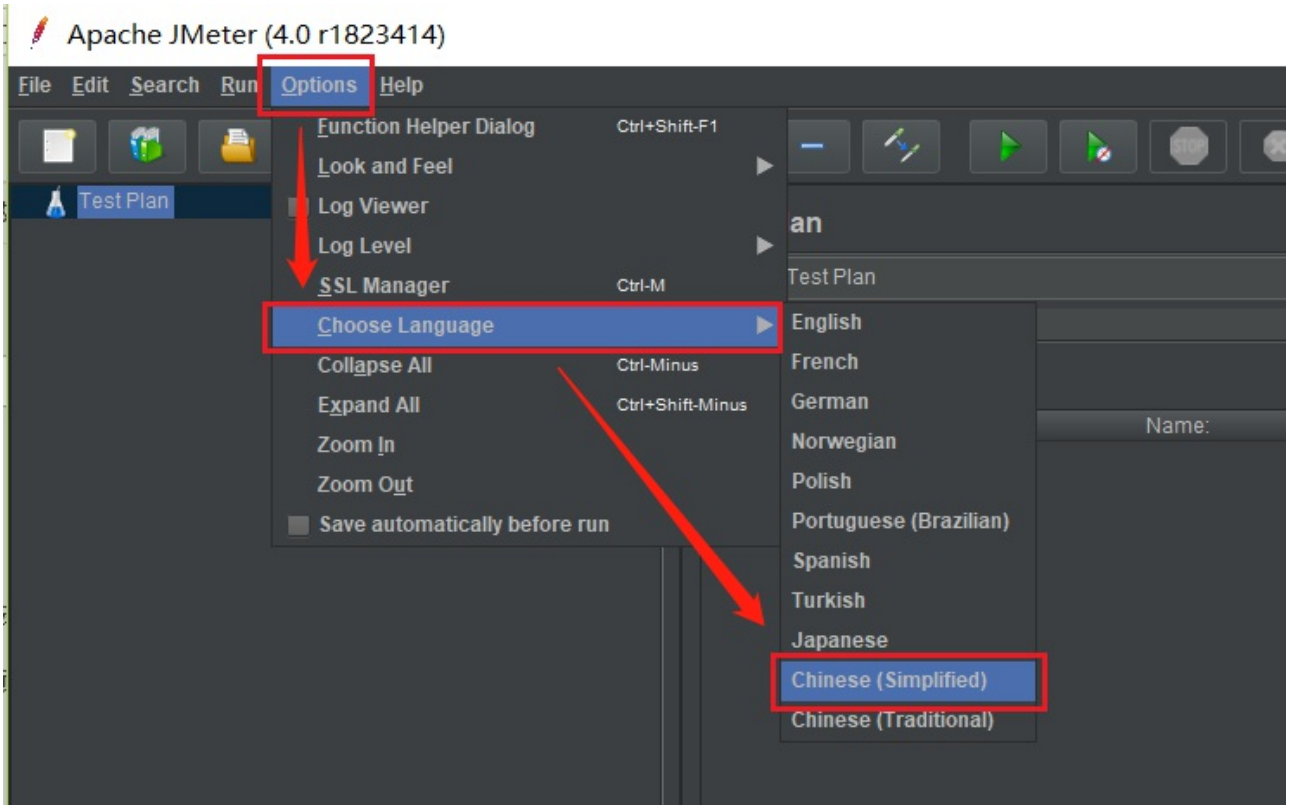
下载完成后，解压文件夹，进入 bin 目录，点击 jmeter.bat 进行 JMeter 的安装，安装成功后的界面如下。



配置测试计划

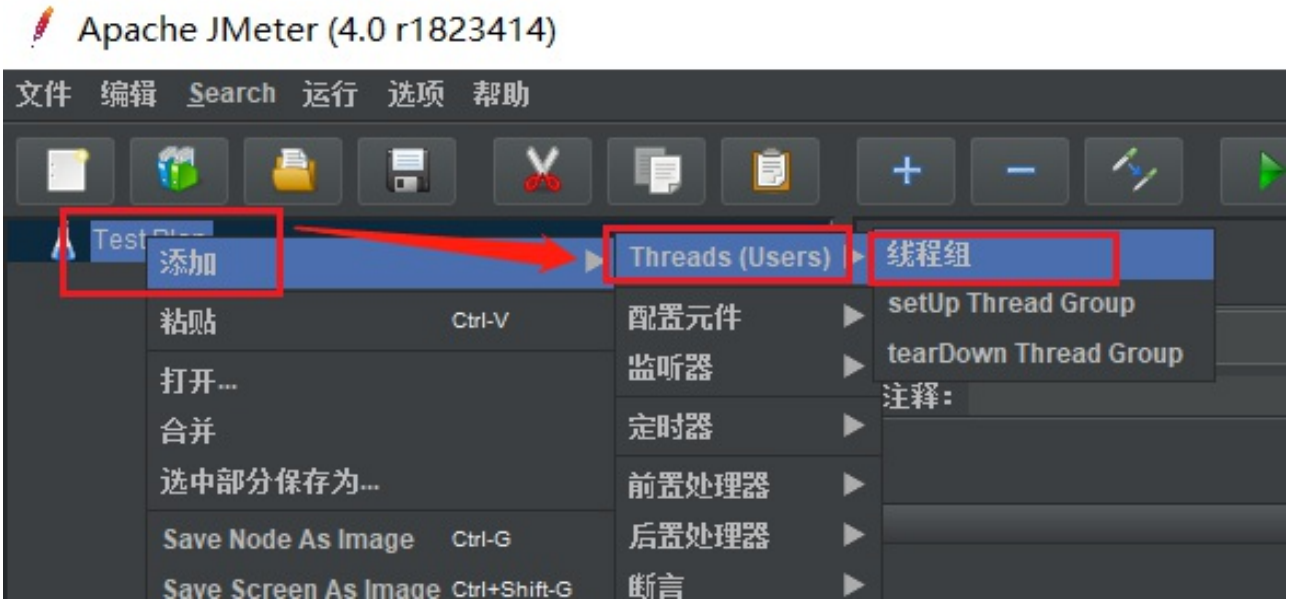
切换语言

依次选择“Options” -> “Choose Language” -> “Chinese (Simplified)”，如下图所示。

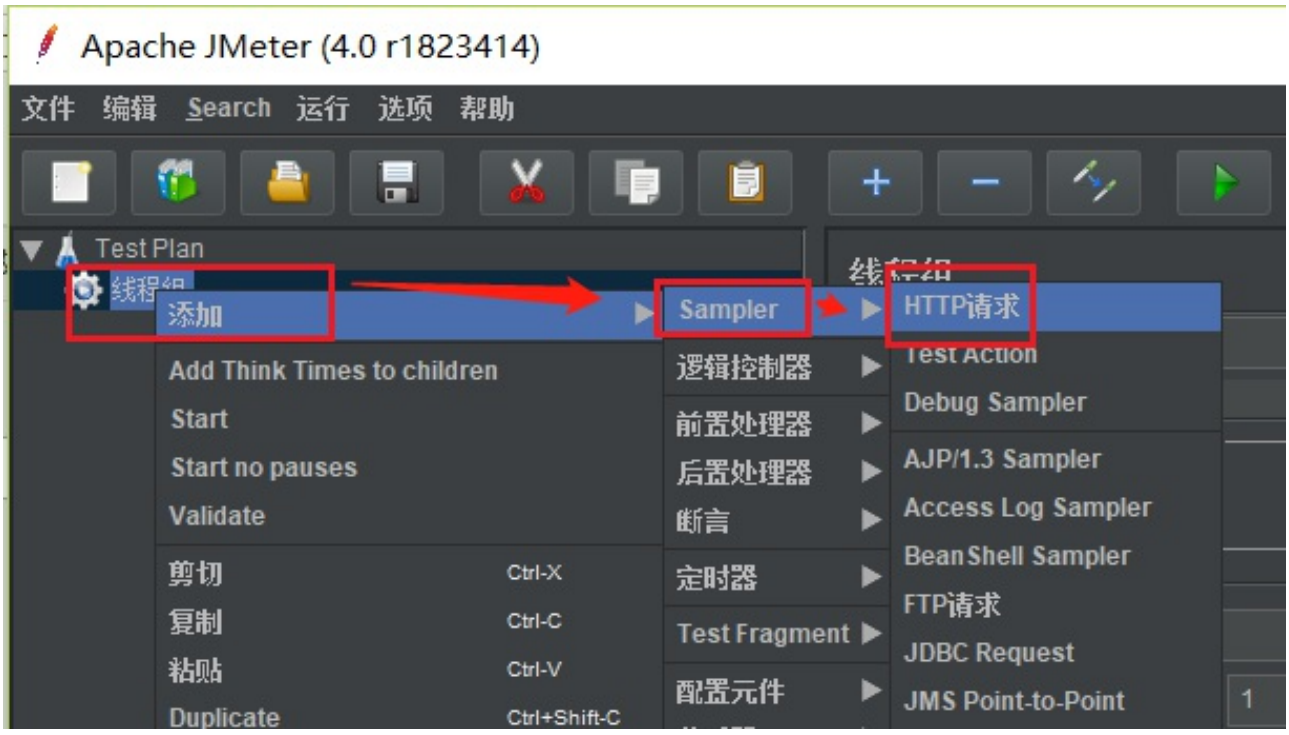


配置 HTTP 请求

右击 “Test Plan”，点击“添加” -> “Threads (Users)” -> “线程组”



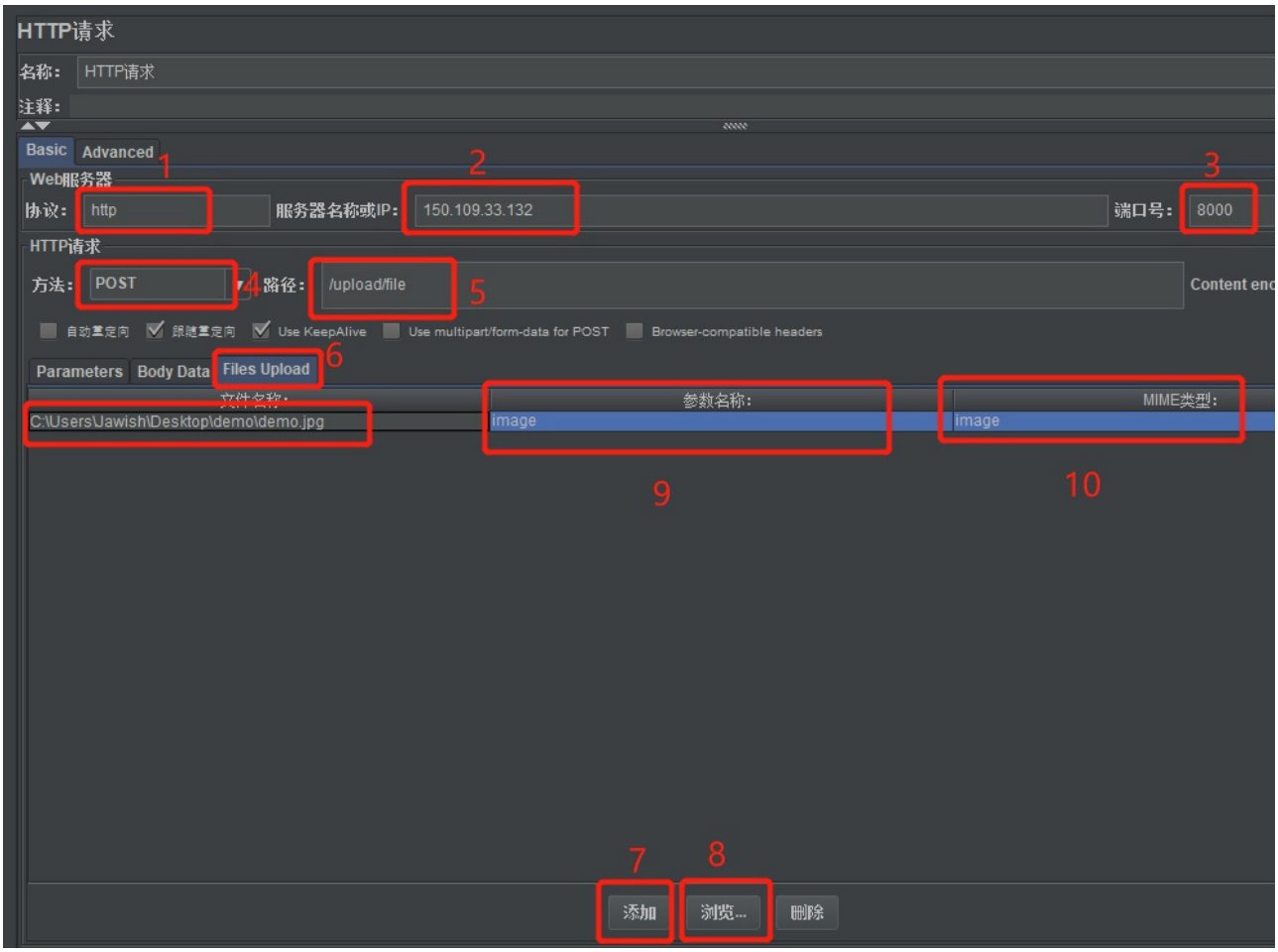
右击 “线程组”，点击 “添加” -> “Sampler” -> “HTTP 请求”



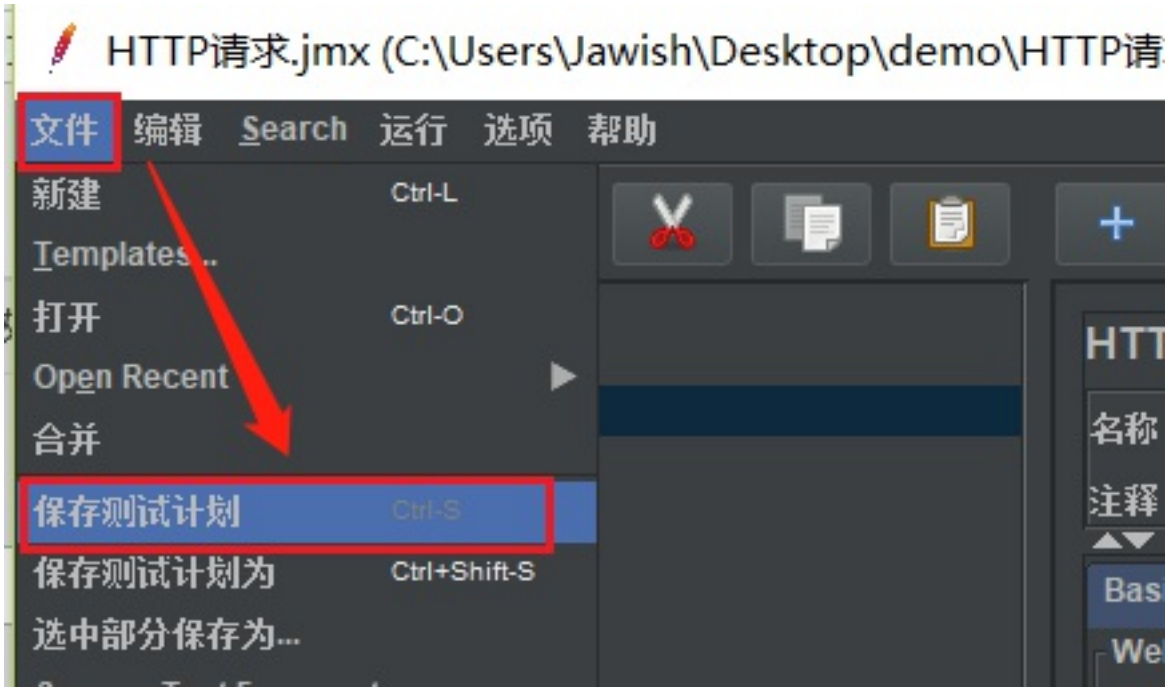
在弹出的「HTTP 请求」框中进行如下设置：

- 第 1~4 步，按照截图输入或选择；
- 第 5 步，设定我们要上传图片（文件）的 URL 路径是 `upload/file`；
- 第 6 步，选择 “Files Upload”；
- 第 7 步，点击 ”添加”；

- 第 8 步，点击 “浏览”，从本地随便选取一张图片（或本小节末尾提供的图片）；
- 第 9 步，输入该图片对象的参数名 image；
- 第 10 步，输入我们上传的文件类型 image。

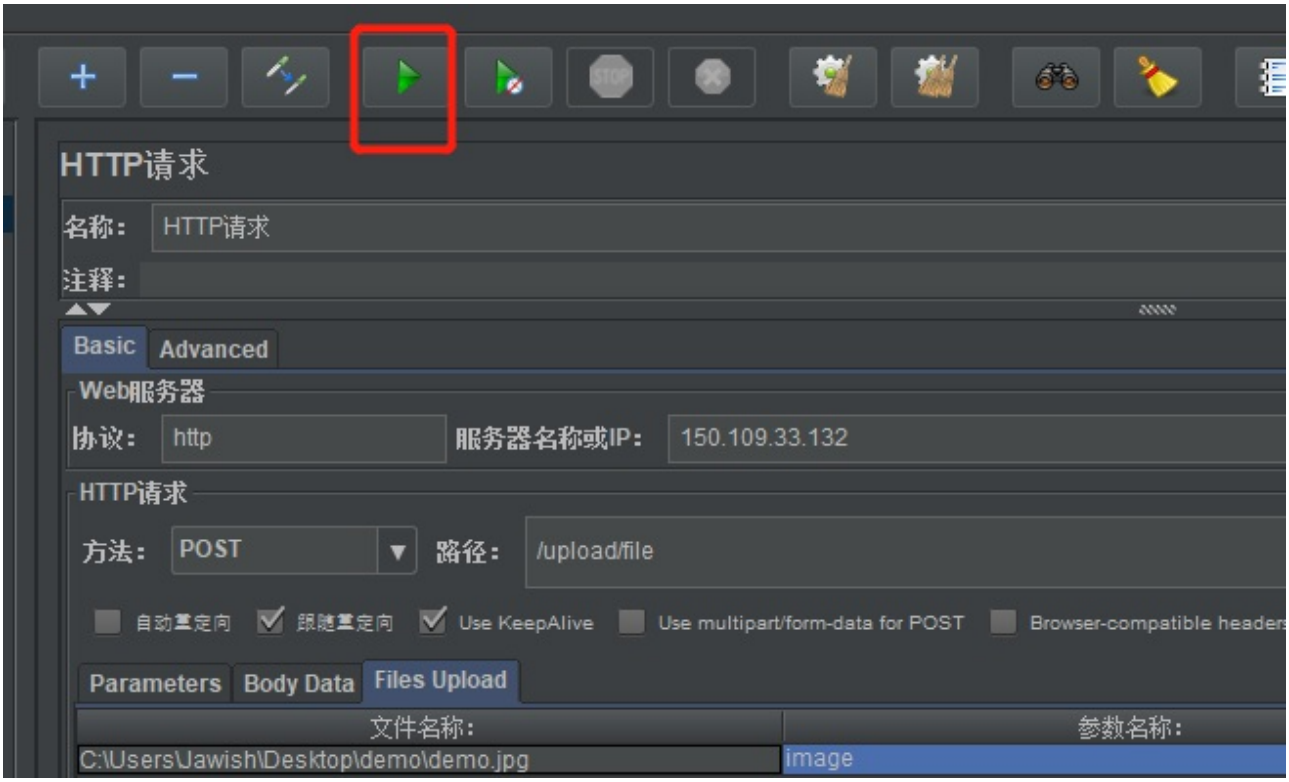


至此，请求页面已配置完毕，点击 “文件” -> “保存测试计划” 如下。



测试请求

点击如下“启动”按钮，测试是否请求成功



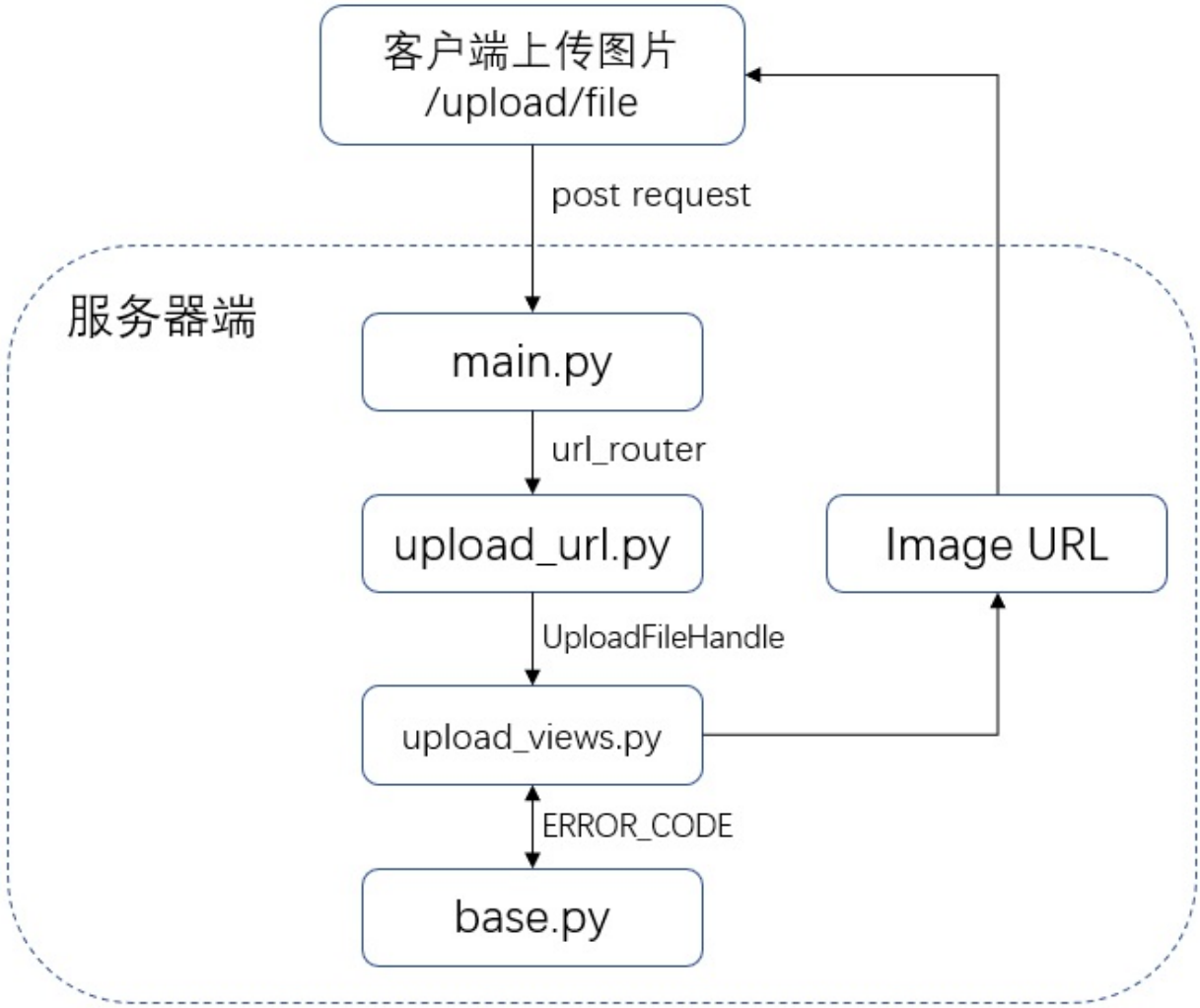
查看服务器端

```
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  models.pyc  static  templates  views
[root@VM_0_8_centos demo]# ./main.py
Tornado server is ready for service
[w 180409 07:31:21 web:2106] 404 POST /upload/file (183.53.66.21) 44.17ms
```

此打印说明服务器端接收客户端请求成功，但由于 /upload/file 路径的代码未实现，服务器端返回 404 找不到路径。接下来，将进行服务器端图片上传代码编写。

服务器端代码编写

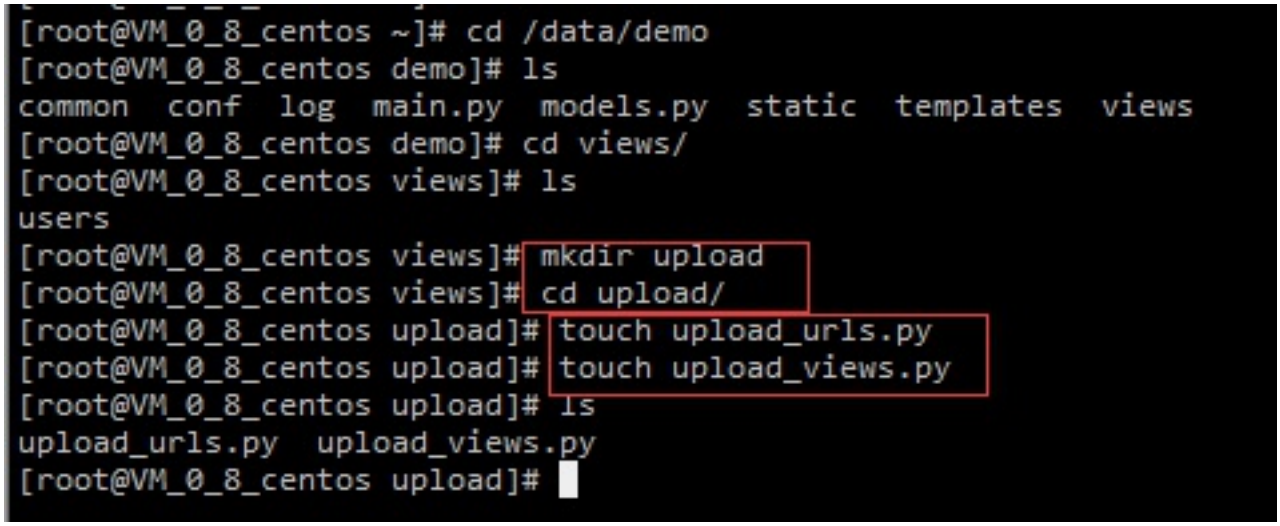
调用逻辑



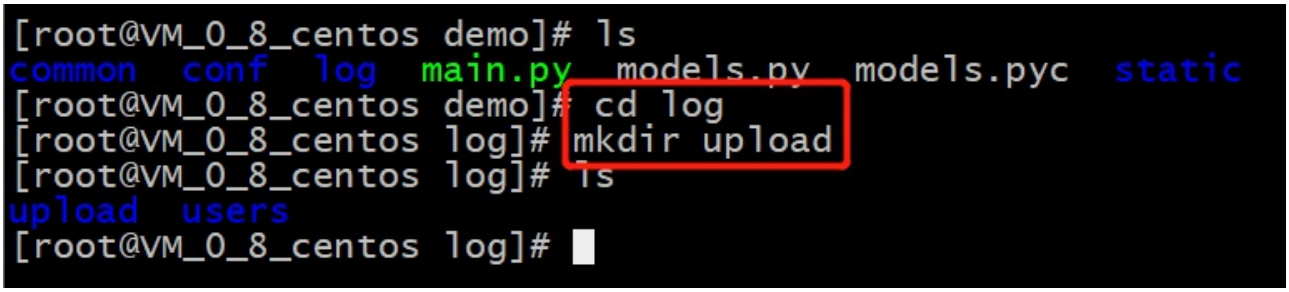
与第 6 小节用户注册请求服务器端实现类似，客户端上传图片，进入 main.py，将调用 url_router 转发到 upload_url.py 中，在 upload_urls.py 中，对应的 URL 将调用 upload_views.py 的 UploadFileHandle 类，UploadFileHandle 为真正的代码处理逻辑，在校验用户信息正确的情况下，返回图片 URL 给客户端，客户端加载该图片。

创建目录

在 views 下面创建 upload 目录，在 upload 下创建 upload_urls.py、upload_views.py等文件。



在 log 目录下创建 upload 目录，用于存放日志。



图片一般会放在 static 目录下，在实际项目中，static 下的图片目录也是分层级的，此次讲解，我们将简化，把图片直接放在 static/image 目录下。创建 image 目录如下：

```
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  models.pyc  static  templates
[root@VM_0_8_centos demo]# cd static/
[root@VM_0_8_centos static]# mkdir image
[root@VM_0_8_centos static]# ls
css  image  js
[root@VM_0_8_centos static]#
```

编写逻辑代码

修改 main.py 文件，增加 views.upload.upload_urls下的 url 路由，修改 handers 如下：

```
        handlers = url_wrapper([
        (r"/users/",
include('views.users.users_urls')),
        (r"/upload/",
include('views.upload.upload_urls'))
        ])
```

```
18
19  class Application(tornado.web.Application):
20      def __init__(self):
21          initdb()
22          handlers = url_wrapper([
23          (r"/users/", include('views.users.users_urls')),
24          (r"/upload/", include('views.upload.upload_urls'))
25          ])
26          #定义tornado服务器的配置项，如static/templates目录位置，debug级别等
27          settings = dict(
28              debug=True,
29              static_path=os.path.join(os.path.dirname(__file__),"static"),
30              template_path=os.path.join(os.path.dirname(__file__), "templates")
31          )
32          tornado.web.Application.__init__(self, handlers, **settings)
33          self.db = scoped_session(sessionmaker(bind=engine,
34                                   autocommit=False, autoflush=True,
35                                   expire_on_commit=False))
36
```

修改 upload_urls.py，输入如下代码：

```
#!/usr/bin/python3
# -*- coding:utf-8 -*-


from __future__ import unicode_literals
from .upload_views import (
    UploadFileHandle
)

urls = [
    #从/upload/file过来的请求，将调用upload_views里面
的UploadFileHandle类
    (r'file', UploadFileHandle)
]
```

修改 upload_views.py，输入如下代码：

```
#! /usr/bin/python3
# -*- coding:utf-8 -*-

import tornado.web
import os
from tornado.escape import json_decode
import logging
from logging.handlers import
TimedRotatingFileHandler
import json


#从commons中导入http_response及save_files方法
from common.commons import (
    http_response,
    save_files
```

```
)

#从配置文件中导入错误码
from conf.base import (
    ERROR_CODE,
    SERVER_HEADER
)

########## Configure logging #############
logFilePath = "log/upload/upload.log"
logger = logging.getLogger("Upload")
logger.setLevel(logging.DEBUG)
handler = TimedRotatingFileHandler(logFilePath,
                                   when="D",
                                   interval=1,

backupCount=30)
formatter = logging.Formatter('%(asctime)s \
%(filename)s[line:%(lineno)d] %(levelname)s %
(message)s',)
handler.suffix = "%Y%m%d"
handler.setFormatter(formatter)
logger.addHandler(handler)


class
UploadFileHandle(tornado.web.RequestHandler):
    """handle /upload/file request, upload image
and save it to static/image/
    :param image: upload image
    """

    def post(self):
```

```
        try:
            #获取入参
            image_metas =
self.request.files['image']
        except:
            #获取入参失败时，抛出错误码及错误信息
            logger.info("UploadFileHandle:
request argument incorrect")
            http_response(self,
ERROR_CODE['1001'], 1001)
            return

        image_url = ""
        image_path_list = []
        if image_metas:
            #获取当前的路径
            pwd = os.getcwd()
            save_image_path = os.path.join(pwd,
"static/image/")
            logger.debug("UploadFileHandle: save
image path: %s" %save_image_path)
            #调用save_file方法将图片数据流保存在硬盘中
            file_name_list =
save_files(image_metas, save_image_path)
            image_path_list = [SERVER_HEADER +
"/static/image/" + i for i in file_name_list]
            ret_data = {"imageUrl":
image_path_list}
            #返回图片下载地址给客户端
            self.write(json.dumps({"data":
{"msg": ret_data, "code": 0}}))
        else:
            #如果图片为空，返回图片为空错误信息
```

```
            logger.info("UploadFileHandle: image
stream is empty")
            http_response(self,
ERROR_CODE['2001'], 2001)
```

这里，我们从 common 导入 save_files 用于处理图片的保存，从 conf 的 base 中导入 SERVER_HEADER，定义了我们服务器的 URL 前缀。同时也看到，upload 和 users 的 Log 配置（如级别）是单独配置的，这样有助于单模块调试。下面修改 conf 目录下的 base.py 文件，增加如下：

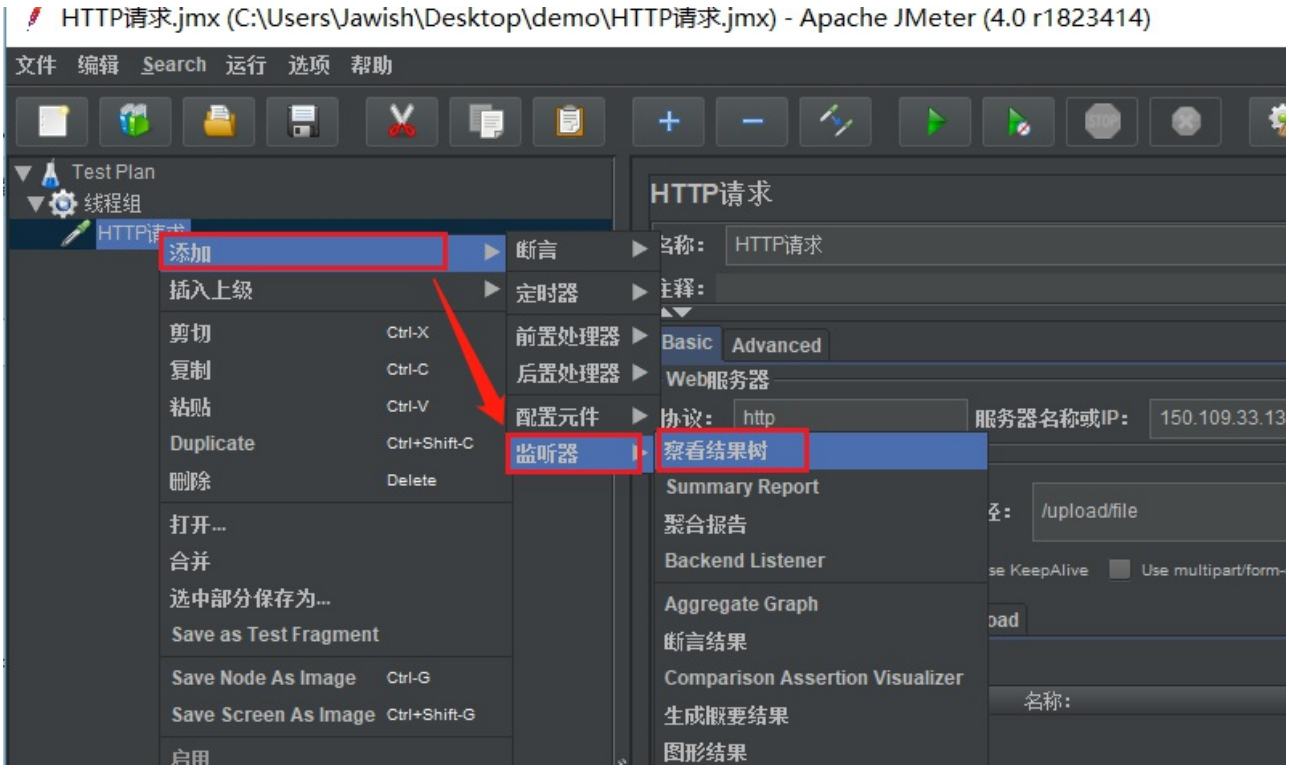
完整代码如下：

```
#! /usr/bin/python3
# -*- coding:utf-8 -*-

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import
declarative_base
engine =
create_engine('mysql://root:pwd@demo@localhost:33
06/demo?charset=utf8', encoding="utf8",
echo=False)
BaseDB = declarative_base()

#服务器端 IP+Port，请修改对应的IP
SERVER_HEADER = "http://150.109.33.132:8000"

ERROR_CODE = {
    "0": "ok",
    #Users error code
    "1001": "入参非法",
    "1002": "用户已注册，请直接登录",

    "2001": "上传图片不能为空"
}
```

commons.py 下，导入 os 模块（ import os ），并增加 save_files 方法：

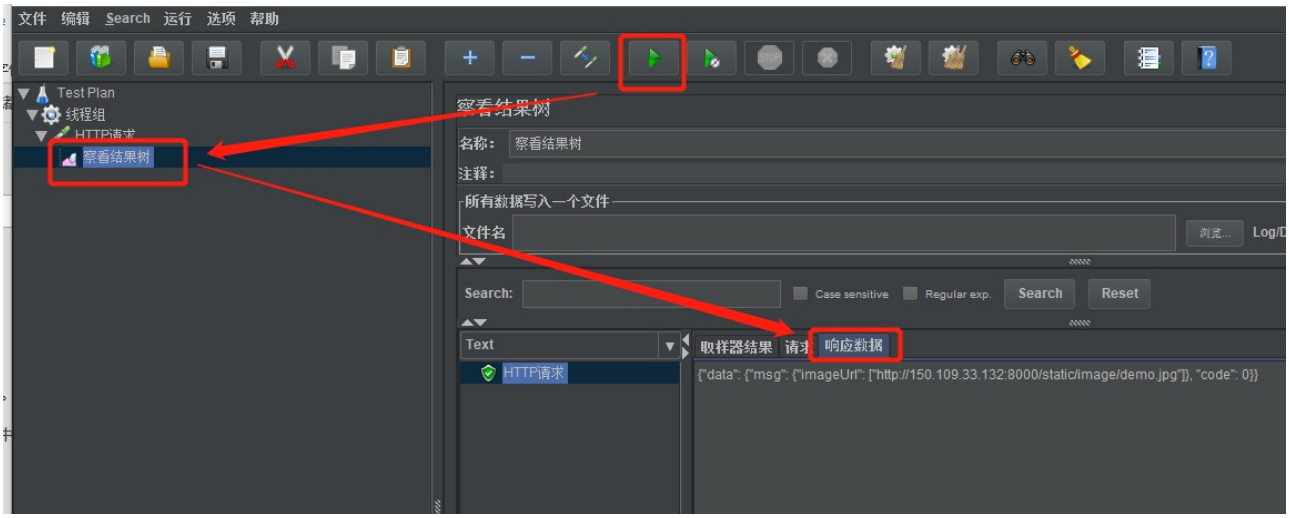
```
import os

def save_files(file_metas, in_rel_path,
type='image'):
    """
    Save file stream to server
    """
    file_path = ""
    file_name_list = []
    for meta in file_metas:
        file_name = meta['filename']
        file_path = os.path.join( in_rel_path,
file_name )
        file_name_list.append( file_name )
        #save image as binary
        with open( file_path, 'wb' ) as up:
            up.write( meta['body'] )
    return file_name_list
```

至此，服务器端的代码已完成。再次从 JMeter 触发图片上传，在触发图片上传之前，我们先创建 JMeter 的结果树。所谓结果树，就是在触发请求之后，查看服务器端返回的结构。右击 “HTTP 请求”，依次选择“添加” -> ”监听器” -> “查看结果树”，如下图所示。



触发 JMeter 图片上传，点击 “察看结果树”，切到 “响应数据” 页面，可以看到服务器端返回的数据信息：

```
{"data": {"msg": {"imageUrl": ["http://150.109.33.132:8000/static/image/demo.jpg"]}, "code": 0}}
```



查看服务器端进程打印：

```
[root@VM_0_8_centos demo]#
[root@VM_0_8_centos demo]# ./main.py
Tornado server is ready for service

[D 180410 07:20:21 upload_views:69] UploadFileHandle: save image path: /data/demo/static/image/
[I 180410 07:20:21 web:2106] 200 POST /upload/file (183.53.66.238) 40.19ms
```

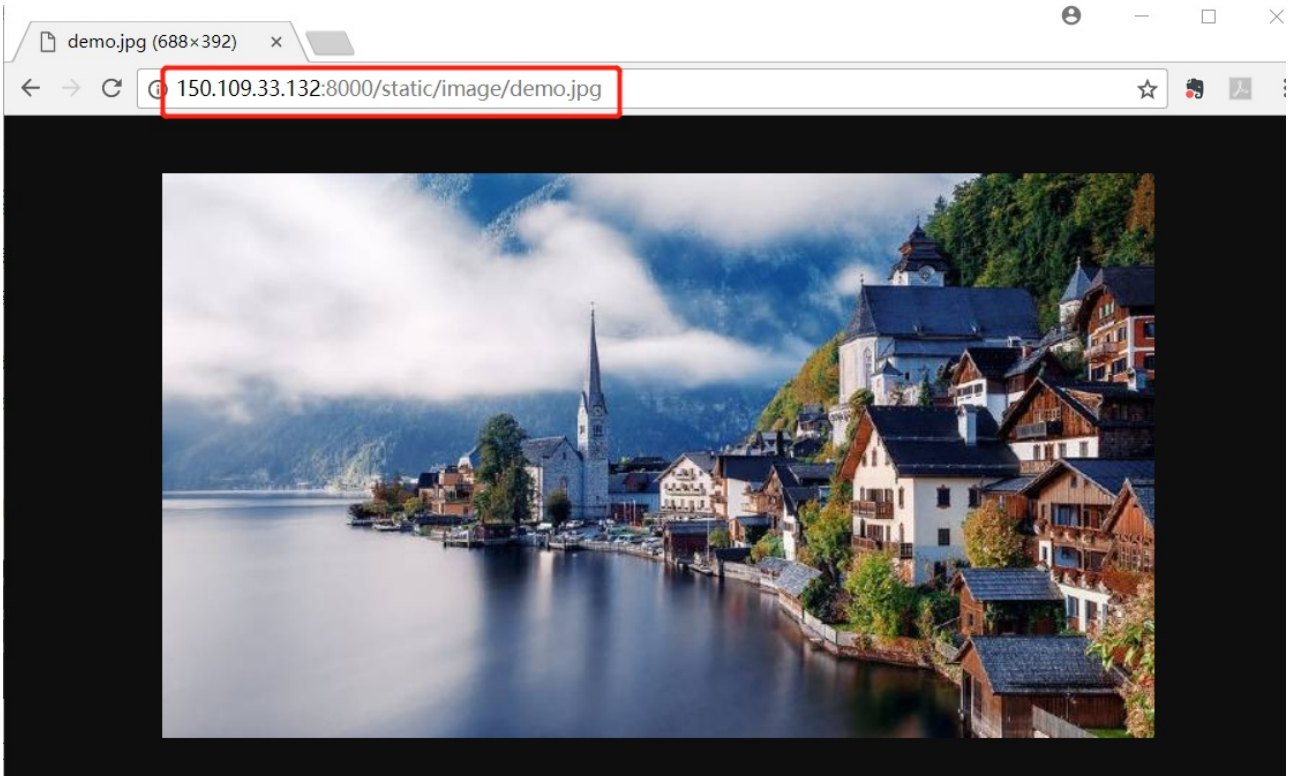
查看图片是否上传：

```
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  models.pyc  static  templates  views
[root@VM_0_8_centos demo]# cd static/image/
[root@VM_0_8_centos image]# ls
demo.jpg
[root@VM_0_8_centos image]#
```

查看 log 是否成功写入：

```
[root@VM_0_8_centos demo]# ls
common  conf  log  main.py  models.py  models.pyc  static  templates  views
[root@VM_0_8_centos demo]# cd log/upload/
[root@VM_0_8_centos upload]# ls
upload.log
[root@VM_0_8_centos upload]# more upload.log
2018-04-10 07:15:13,217 upload_views.py[line:69] DEBUG UploadFileHandle: save image path: /data/demo/static/image/
```

此时，客户端就可以通过服务器端返回的图片 URL（http://150.109.33.132:8000/static/image/demo.j 加载图片了，在浏览器中输入图片 URL，查看加载是否成功。



代码下载

到目前为止，服务器端代码及图片如下：
demo9 (https://github.com/Jawish185/demo9.git)

小结

至此，我们完成了服务器端图片上传的接收及图片 URL 返回，客户端根据服务器返回的图片 URL，即可加载该图片。这里没有写数据库的操作，读者可以尝试参考第 8 节的讲解，定义图片的 `models`，并将图片 URL 和其他信息写入数据库中。