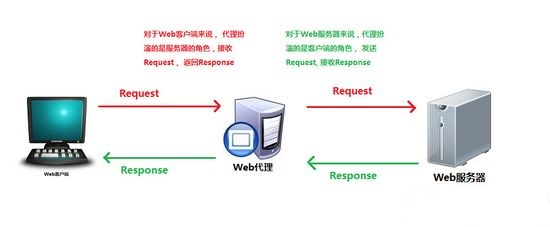
LittleProxy简介及使用指南

Web代理服务器是网络的中间实体，代理位于Web客户端和Web服务器之间，属于“中间人”的角色，Http Proxy服务器既是Web服务器又是Web客户端：



LittleProxy，是用Netty实现的一个Http proxy服务器，稳定，而且性能较优，很容易整合到项目中。使用方式有两种：

1）命令行启动

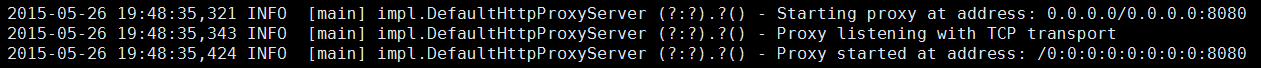
这种方式，LittleProxy作为单独的进程，运行在Linux中。运行较为简单：

$ git clone git://github.com/adamfisk/LittleProxy.git

$ cd LittleProxy

$ ./run.bash

run.bash脚本执行，编译LittleProxy后，启动代理服务器，默认端口是8080，执行结果如下所示：



2）嵌入到项目中

在pom.xml中，添加依赖

<dependency>

<groupId>org.littleshoot</groupId>

<artifactId>littleproxy</artifactId>

<version>1.0.0-beta7</version>

</dependency>

使用下面的代码，启动一个ProxyServer

HttpProxyServer server =

DefaultHttpProxyServer.bootstrap()

.withPort(8080)

.start();

在实际开发的过程中，可以向LittleProxy中添加很多的Filter，使用HttpFiltersSource(Ada

pter)添加request及response的Filters，示例如下所示：

HttpProxyServer server =

DefaultHttpProxyServer.bootstrap()

.withPort(8080)

.withFiltersSource(new HttpFiltersSourceAdapter() {

public HttpFilters filterRequest(HttpRequest originalRequest, ChannelHandlerContext ctx) {

return new HttpFiltersAdapter(originalRequest) {

@Override

public HttpResponse clientToProxyRequest(HttpObject httpObject) {

// TODO: implement your filtering here

return null;

}

@Override

public HttpResponse proxyToServerRequest(HttpObject httpObject) {

// TODO: implement your filtering here

return null;

}

@Override

public HttpObject serverToProxyResponse(HttpObject httpObject) {

// TODO: implement your filtering here

return httpObject;

}

@Override

public HttpObject proxyToClientResponse(HttpObject httpObject) {

// TODO: implement your filtering here

return httpObject;

} };}})

.start();

如何启动相同的Proxy Server，但是监听不同的端口，可以调用clone方法。Clone的Servers与原Server使用共同的Event Loops，减少资源的使用：

existingServer.clone().withPort(8081).start()

注：重要链接

github:

https://github.com/adamfisk/LittleProxy

maven库：

http://gradle.artifactoryonline.com/gradle/libs/org/littleshoot/littleproxy/

API:

http://littleproxy.org/apidocs/index.html

Netty API:

http://netty.io/4.0/api/index.html

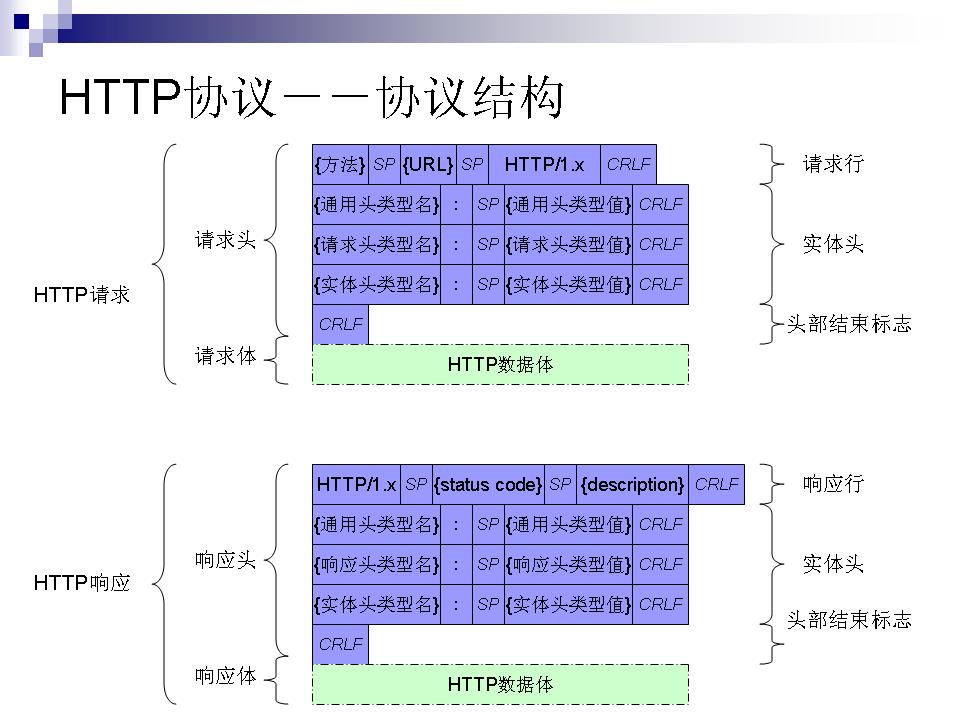
# 1. Http协议

超文件传输协议（HTTP）是应用层协议，一种请求/响应式的协议。客户机与服务器建立连接后，发送一个请求给服务器；服务器接到请求后，给予相应的响应信息。具体步骤如下：

1. 客户机与服务器建立连接，例如单击某个超级链接，HTTP的工作开始
2. 建立连接后，客户机发送一个请求给服务器，请求的格式：URL，协议版本号，后面是MIME信息（包括请求修饰符、客户机信息和可能的内容）
3. 服务器接受请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、成功或者错误的代码，后边是MIME信息（服务器信息、实体信息和可能的内容）
4. 客户端接收服务器所返回的信息，通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接

## 1.1 Request与Reponse格式

HTTP协议格式比较简单，如下图所示



1. Request请求

格式：<Http请求行，请求头，空行，可选的消息体>

GET  http://photo.test.com/inc/global.js  HTTP/1.1

Host: photo.test.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; zh-CN; rv:1.8.1) Gecko/20061010 Firefox/2.0

Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5

Accept-Language: en-us,zh-cn;q=0.7,zh;q=0.3

Accept-Encoding: gzip,deflate

Accept-Charset: gb2312,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Proxy-Connection: keep-alive

Cookie: ASP.NET\_SessionId=ey5drq45lsomio55hoydzc45

Cache-Control: max-age=0

1. Response格式

格式：<HTTP状态行，应答头，空行，可选的消息体>

HTTP/1.1 200 OK

Cache-Control: private, max-age=30

Content-Type: text/html; charset=utf-8

Content-Encoding: gzip

Expires: Mon, 25 May 2009 03:20:33 GMT

Last-Modified: Mon, 25 May 2009 03:20:03 GMT

Vary: Accept-Encoding

Server: Microsoft-IIS/7.0

X-AspNet-Version: 2.0.50727

X-Powered-By: ASP.NET

Date: Mon, 25 May 2009 03:20:02 GMT

Content-Length: 12173

<....消息内容省略>

# 2. Netty