CMPS109 Make-up Examination

Max Zhao

mlzhao@ucsc.edu 1466223

1. HTTPHTMLService.cpp

```
FileCacheItem * fileCacheItem = fileCache->getFile(resource); // fetching the resource cache item
fileCacheItem = fileCacheItem->fetchContent(); // update cache item if needed and return a clone
// Instantiate an HTTPresponse object and set up its header attributes
struct tm tm;
string s = p_httpRequest->getHeaderValue("If-Modified-Since");

if(s == fileCacheItem->getLastUpdateTime()) { // sees if "if-modified-since" header is the same as cached last update time
    // creates new httpResponseHeader with new message and status code
    HTTPResponseHeader * httpResponseHeader = new HTTPResponseHeader(p_tcpSocket, "Cache copy valid", 304, "HTTP/1.1");
    httpResponseHeader->setHeader("Connection","close");
    httpResponseHeader->respond();
    delete (httpResponseHeader); // Delete the HTTP Response
    delete (fileCacheItem); // delete the cache item clone
}else{
    HTTPResponseHeader * httpResponseHeader = new HTTPResponseHeader(p_tcpSocket,"OK",200,"HTTP/1.1");
    httpResponseHeader->setHeader("Content-Type","text/html"); // Set content type
    // Fetch the date/time string of the last modified attribute and set it to the header
    httpResponseHeader->setHeader("Last-Modified",fileCacheItem->getLastUpdateTime());
    // This implies that the connection terminates after service the request; i.e. keep-alive is not supported
    httpResponseHeader->setHeader("Connection","close");
    httpResponseHeader->setHeader("Content-Length",to_string(fileCacheItem->getSize()));
    httpResponseHeader->respond(); // Write back the response to the client through the TCPSocket
    // Write back the file to the client through the TCPSocket
    p_tcpSocket->writeToSocket(fileCacheItem->getStream(),fileCacheItem->getSize());
    delete (httpResponseHeader); // Delete the HTTP Response
    delete (fileCacheItem); // delete the cache item clone
}
```

2. HTTPImageService.h

```
#ifndef HTTPIMAGESERVICE_H
#define HTTPIMAGESERVICE_H
#include "HTTPService.h"
// An HTTP Service that serves HTML files
class HTTPImageService: public HTTPService // Inherit from HTTPService Base class
{
    private:
    public:
        // Constructor, receiving file cache, resources to serve and a flag that enabled deleting the cache in the destructor
        HTTPImageService(FileCache * p_fileCache,bool p_clean_cache = true);
        // Execute the service and write back the results to the TCPSocket
        virtual bool execute(HTTPRequest * p_httpRequest,TCPSocket * p_tcpSocket);
        virtual HTTPService * clone (); // Clone and create a new object
        ~HTTPImageService(); // Destructor
};
```

3. HTTPImageService.cpp

```cpp
#include "HTTPImageService.h"
#include "HTTPResponseHeader.h"
#include "HTTPNotFoundExceptionHandler.h"

HTTPImageService::HTTPImageService(FileCache * p_fileCache,bool p_clean_cache )
        :HTTPService(p_fileCache,p_clean_cache) {} // Constructor setting data members using initialization list
bool HTTPImageService::execute(HTTPRequest * p_httpRequest,TCPSocket * p_tcpSocket)
{
    try { // Try the following block and look for exceptions
        string resource = p_httpRequest->getResource(); // Fetching the resource from the HTTPRequest object

        FileCacheItem * fileCacheItem = fileCache->getFile(resource); // fetching the resource cache item
        fileCacheItem = fileCacheItem->fetchContent(); // update cache item if needed and return a clone
        // Instantiate an HTTPresponse object and set up its header attributes
        struct tm tm;
        string s = p_httpRequest->getHeaderValue("If-Modified-Since");

        if(s == fileCacheItem->getLastUpdateTime()) { // optimization modification from problem 1
            HTTPResponseHeader * httpResponseHeader = new HTTPResponseHeader(p_tcpSocket, "Cache copy valid", 304, "HTTP/1.1");
            httpResponseHeader->respond();
            delete (httpResponseHeader); // Delete the HTTP Response
            delete (fileCacheItem); // delete the cache item clone
        }else{
            HTTPResponseHeader * httpResponseHeader = new HTTPResponseHeader(p_tcpSocket,"OK",200,"HTTP/1.1");
            httpResponseHeader->setHeader("Content-Type","img"); // Set content type
            // Fetch the date/time string of the last modified attribute and set it to the header
            httpResponseHeader->setHeader("Last-Modified",fileCacheItem->getLastUpdateTime());
            // This implies that the connection terminates after service the request; i.e. keep-alive is not supported
            httpResponseHeader->setHeader("Connection","close");
            httpResponseHeader->setHeader("Content-Length",to_string(fileCacheItem->getSize()));
            httpResponseHeader->respond(); // Write back the response to the client through the TCPSocket
            // Write back the file to the client through the TCPSocket
            p_tcpSocket->writeToSocket(fileCacheItem->getStream(),fileCacheItem->getSize());
            delete (httpResponseHeader); // Delete the HTTP Response
            delete (fileCacheItem); // delete the cache item clone
         }

        return true; // return true
    }
    catch (HTTPNotFoundExceptionHandler httpNotFoundExceptionHandler)
    { // Exception occurred and caught
        // Handle the exception and send back the corresponding HTTP status response to client
        httpNotFoundExceptionHandler.handle(p_tcpSocket);
        return false; // return false
    }
}
// Clone a new identical object and return it to the caller
HTTPService * HTTPImageService::clone ()
{
    // Instantiate an HTTPHTMLService object and set it up with the same fileCache.
    // Notice that the clean flag is set to false as the current object will be carrying this out.
    return new HTTPImageService(fileCache,false);
}
HTTPImageService::~HTTPImageService() // Destructor
{
}
```

4. HTTPServiceManager.cpp

```cpp
#include "HTTPServiceManager.h"
#include "HTTPNotAcceptableExceptionHandler.h"
#define WEB_CACHE_ROOT  "/Users/maxzhao/Desktop/CMPS109/takeHomeMidterm/www"
// Constructor:  building up the factory map
HTTPServiceManager::HTTPServiceManager()
{
    // adding the html and form service cloners
    services ["html"] = new HTTPHTMLService(new FileCache(WEB_CACHE_ROOT),true);
    services ["form"] = new HTTPXMLService(); // upon form, enter new XML service
    services ["png"] = new HTTPImageService(new FileCache(WEB_CACHE_ROOT),true); // adds support for png
    services ["jpg"] = new HTTPImageService(new FileCache(WEB_CACHE_ROOT),true); // adds support for jpg
    services ["gif"] = new HTTPImageService(new FileCache(WEB_CACHE_ROOT),true); // adds support for gif
}
// Compare the file extention to the map key first and if not found compare the whole file name
HTTPService * HTTPServiceManager::getService (string p_resource)
{
    // extract extentions
    string ext = p_resource.substr(p_resource.find_last_of(".") + 1);
    if ( services[ext]==NULL)  // if not found
    {
        // Extract file base noame
        string base_name = p_resource.substr(p_resource.find_last_of("/") + 1);
        // If not found also throw and exception
        if ( services[base_name]==NULL) throw (HTTPNotAcceptableExceptionHandler());
        else return services[base_name]->clone(); // else clone service based on base file name
    }
    else return services[ext]->clone(); // clone service based on extension
}
// Destructor
HTTPServiceManager::~HTTPServiceManager()
{
    // A for_each iterator based loop with lambda function to deallocate all the cloner objects

    for_each (services.begin(),services.end(),[](const std::pair<string,HTTPService *>& it) -> bool {
        HTTPService * httpService = std::get<1>(it);
        delete(httpService);
        return true;
    });
}
```

5. HTTPXMLService.h

```cpp
#ifndef HTTPXMLSERVICE_H
#define HTTPXMLSERVICE_H

#include "HTTPFormService.h"
// An HTTP Service that serves XML Form posts and parses
class HTTPXMLService: public HTTPFORMService // Inherit from HTTPService Base class
{
    protected:
        string compose_reply(); // overwrites HTTPFORMService parent class's compose_reply
    public:
        // Constructor, receiving file cache, resources to serve and a flag that enabled deleting the cache in the destructor
        HTTPXMLService();
        // Execute the service and write back the results to the TCPSocket
        virtual bool execute(HTTPRequest * p_httpRequest,TCPSocket * p_tcpSocket);
        virtual HTTPService * clone (); // Clone and create a new object
        ~HTTPXMLService(); // Destructor
};



#endif /* HTTPXMLSERVICE_H */
```

6. HTTPXMLService.cpp

```cpp
#include "HTTPXMLService.h"
#include "HTTPResponseHeader.h"
#include "HTTPNotFoundExceptionHandler.h"


// compose the reply body from the maps built up of the form field. The reply body is a HTML stream containing to HTML tables.
// The first table presents the HTML form fields in their raw format and the second table presents the HTML form fields after URL dec
string HTTPXMLService::compose_reply()
{
    // The reply string contains the HTML stream. We compose it step by step
    string reply = "<HTMLForm>"; // HTMLForm main tag
    // for_each iterator loop over form_data with lambda function to build XML page
    for_each (form_data.begin(),form_data.end(),[&reply](const std::pair<string,string>& it) -> bool {
            reply += "<";
            reply += std::get<0>(it); // creates form field name
            reply += ">";
            reply += std::get<1>(it); // form field value
            reply += "</";
            reply += std::get<0>(it); // form field name
            reply += ">";
            return true;
    });
    reply += "</HTMLForm>";
    return reply; // return reply


}
HTTPXMLService::HTTPXMLService( )
        :HTTPFORMService() {} // Constructor setting data members using initialization list

// Execute  the HTTPXML service
bool HTTPXMLService::execute(HTTPRequest * p_httpRequest,TCPSocket * p_tcpSocket)
{
    parse_data(p_httpRequest); // parse the request body data
    string reply = compose_reply(); // compose the HTTP reply body
    //Build and set the HTTP response Header fields.
    HTTPResponseHeader * httpResponseHeader = new HTTPResponseHeader(p_tcpSocket,"OK",200,"HTTP/1.1");
      httpResponseHeader->setHeader("Content-Type","text/xml"); // explicitly defines content type as xml
      httpResponseHeader->setHeader("Connection","close");
      httpResponseHeader->setHeader("Content-Length",to_string(reply.length()));
      httpResponseHeader->respond(); // Write back the response to the client through the TCPSocket
      // Write back the file to the client through the TCPSocket
      p_tcpSocket->writeToSocket(reply.c_str(),reply.length());
      delete (httpResponseHeader); // Delete the HTTP Response
      return true; // return true
}
 // Clone a new identical object and return it to the caller
 HTTPService * HTTPXMLService::clone ()
{
    // Instantiate an HTTPHTMLService object and set it up with the same fileCache.
    // Notice that the clean flag is set to false as the current object will be carrying this out.
    return new HTTPXMLService();
}
HTTPXMLService::~HTTPXMLService(){} // Destructor
```

For number 1, we were to optimize and modify the HTTPHTMLService class, so that the “If-Modified-Since” request header would only send back the target HTML file content if the provided date is less than the cache last modified date. I was able to check the provided date by getting the header value from p_httpRequest->getHeaderValue, and the cached date from fileCacheItem->getLastUpdateTime(). Comparing the two timestamps, if the cached time time was equal to the if-modified-since header, I passed in a new HTTPResponseHeader object with status code 304 “Cache copy valid”, and set only the connection header. Otherwise, I passed in the HTML file content normally as given in the skeleton code, with a status code of “200 OK.” For number 2, the new HTTPImageService class was very similar to the HTTPHTMLService class, created simply in sake of cleanliness, reading the file and fetching content. The biggest change was made in HTTPServiceManager, where I added new service cloners supporting the image types, “png”, “jpg”, “gif” and creating respective new HTTPImageService objects.

For number 3, I extended the HTTPFormService class to parse an HTML form post and return an XML stream representing the form fields by creating a new HTTPXMLService class. Using the protected variables of formdata and HTTPFormService’s parsing functionality, I overrode the compose_reply function to create the necessary xml document with the required tags. I also made sure to set the content-type response header as “text/xml,” and redirect the “form” service in HTTPServiceManager to create a new HTTPXMLService() in order to go straight to the new XML page upon pressing submit on the form.