웹 서버(Web Server)

시스템 소프트웨어 연구실
이건희

목차

HTTP와 웹 서버

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol)

- WWW[1] 상에서 정보를 주고받을 수 있는 프로토콜
- TCP와 UDP를 사용하며, **80번 포트**를 사용한다.
- 현재는 HTTP/2.0까지 나온 상태

[1] WWW : World-Wide-Web

웹 서버

- HTTP 요청을 받아들이고, 웹 페이지를 반환하는 프로그램
- 위에 언급한 기능을 제공하는 프로그램을 실행하는 컴퓨터

웹 서버의 기능들

- HTTP Request and Reply
- Serve static content
- Serve dynamic generated conent
- Authentication

웹 서버 소프트웨어

- **Apache**
  - 아파치 소프트웨어 재단에서 관리하는 HTTP 웹 서버
  - **Apache Version 2.4 Document**
    https://httpd.apache.org/docs/2.4/ko/
  - **Mirror of Apache HTTP Server**
    https://github.com/apache/httpd

웹 서버 라이브러리


spring
by Pivotal™


Flask
web development,
one drop at a time


django

기본 웹 서버 제작하기

Web Server Core – Multi thread

- Create & destroy Architecture
  - HTTP reqeust가 올 때마다 스레드 생성 및 파괴
  - 생성과 파괴 과정에 오버헤드가 생김

- Pool Architecture
  - 스레드를 일정 개수만큼 미리 생성
  - 실전에선 스레드 생성 개수를 예측하기 힘듬

Web Server Core – Multi process

- 요청이 올 때마다 Process를 생성하는 아키텍쳐
- Multi-Thread 아키텍쳐에 비해 속도가 느림

- 그런데 실무에선 이 아키텍쳐를 많이씀

왜? A: 웹 서버 운영에서 가장 중요한 것은 **안정성**

Server Part

```java
public void run() {
    try {
        ServerSocket serv_sock = new ServerSocket();
        InetSocketAddress ipep = new InetSocketAddress(this.port);
        serv_sock.bind(ipep);

        while(true){
            Socket sock = serv_sock.accept();
            InetAddress inetAddr = serv_sock.getInetAddress();
            ServWorker worker = new ServWorker(sock, inetAddr);
            worker.start();
            this.workers.add(worker);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Worker Thread Part

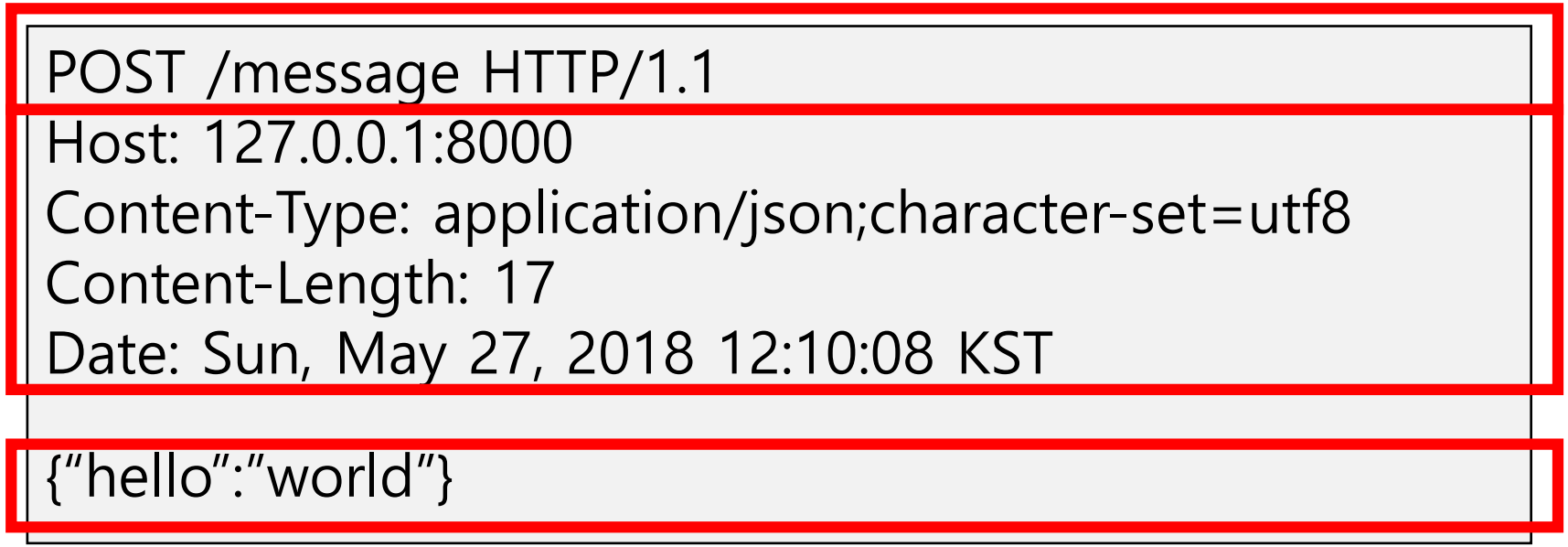
```java
@Override
public void run() {
    try {
        this.reader = sock.getInputStream();
        this.request = new HttpRequest(this.reader);
        String method = this.request.getMethod();
        String path = this.request.getPath();

        this.writer = new PrintWriter(sock.getOutputStream());

        if(!route.isRegistedRoutePath(path, method)) {
            this.response = new HttpResponse(path);
            this.response.putStatusCode( status_code: 404);
        } else {
            this.response = new HttpResponse(path);
            String data = route.getRouteTask(path,method).task(this.request, this.response);
            this.response.setBody(data);
        }
        String res = this.response.set();
        this.writer.print(res);
        this.writer.flush();

        this.close();
        this.printLog();
    } catch(Exception e){
        e.printStackTrace();
        this.close();
    }
}
```

HTTP Request



HTTP Request-Line

1. Method

   Request-URL로 식별된 리소스에 대해 **수행할 방법**을 나타냄

2. Request-URL

   요청을 적용할 **리소스를 식별** (특히, GET Method의 경우 QueryString으로 전달)

3. HTTP version

   HTTP의 버전 (현재는 HTTP/2.0까지 나옴)

HTTP header

- 버전마다 다르며, 상당히 많은 헤더들이 있다.
- 실제로 많이 쓰이는 헤더는 아래와 같다.

Header Key	내용	Header value 예시
Content-Type	컨텐츠의 유형을 나타냄	application/json
Content-Length	컨텐츠의 길이를 나타냄	35
Keep-Alive	Request에 대한 timeout 등의 정보를 담음	timeout=9; max=99
Connection	연결 상태. Keep-Alive와 주로 붙어다님	Keep-Alive

HTTP Request 파싱 방법

1. Separate HTTP header & body
   - HTTP Request 내의 "₩r₩n₩r₩n" 부분을 기준으로 자른다.

2. Parse HTTP Request-Line
   - 공백을 기준으로 자른다.

3. Parse HTTP Request Header
   - "₩r₩n" 기준으로 헤더들을 자른다.
   - 헤더의 정보는 HashMap 자료형에 저장을 한다.

```java
public class HttpRequest {
    private String method;
    private HttpRequestUrl url;
    private String version;
    private HashMap<String, String> headers;
    private StringBuffer body;

    public HttpRequest(InputStream reader){
        this.body = new StringBuffer();
        this.headers = new HashMap<>();
        parseHttpRequest(reader);
    }

    private void parseHttpRequest(InputStream reader) {
        String request_stream = this.getRequestStream(reader);
        String[] hdr_body = request_stream.split( regex: "\r\n\r\n");
        this.parseHeaders(hdr_body[0]);
        this.appendBody(hdr_body);
    }
```

```java
    private void parseHeaders(String header_stream){
        String[] header_line = header_stream.split( regex: "\r\n");

        String request_line = header_line[0];
        readRequestLine(request_line);

        for(int i=1; i<header_line.length; i++)
            readHeaderLine(header_line[i]);
    }

    private void readRequestLine(String line){
        String[] tokens = line.split( regex: "\\s");
        this.method = tokens[0];
        url = new HttpRequestUrl(tokens[1]);
        version = tokens[2];
    }

    private void readHeaderLine(String line){
        int cutline = line.indexOf(":");
        String key = line.substring(0, cutline).trim();
        String value = line.substring(cutline + 1, line.length()).trim();
        headers.put(key, value);
    }

    private void appendBody(String[] hdr_body){
        if((hdr_body.length > 1) && (headers.get("Content-Length") != null))
            for(int i=1; i<hdr_body.length; i++)
                this.body.append(hdr_body[i]);
    }
```

Java 관련 Socket IO 주의점

- BufferedReader의 readLine 메서드는 Blocking 이슈가 있다.
  - Header와 Body 사이의 “₩r₩n₩r₩n” 부분에서 문제
  - BufferedReader는 다음 입력을 대기하는 사태가 발생함.

- InputStream으로 먼저 바이트 단위로 모두 읽어온 후 파싱
  - InputStream 또한 Blocking 이슈가 있긴함
  - available()이라는 메서드를 통해 해결이 가능

Java String 관련 팁

- String을 사용하면 성능이 느려짐.
  - String의 경우는 인스턴스를 계속 생성하는 방법
  - 이미 할당된 메모리는 변하지 않기 때문에 성능상 이슈가 있음

- StringBuffer를 사용하자.
  - String에서 메모리 성능 이슈를 개선한 것. (메모리가 변함)
  - 멀티 스레드 환경에서의 동기화가 지원됨
    (같은 계열로는 StringBuilder가 있지만 이것은 동기화 지원x)
  - 실제 Spring Framework에서도 StringBuffer를 많이 사용

```java
private String getRequestStream(InputStream reader){
    try {
        StringBuffer request = new StringBuffer();
        byte[] b = new byte[4096];
        for (int n; (n = reader.read(b)) != -1;) {
            request.append(new String(b,  offset: 0, n));
            if(reader.available() <= 0) break;
        }
        return request.toString();
    }
    catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Sat, 19 May 2007 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26
(Unix)
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246

{“hello”:”world”} …
```

Status-Line

Headers

Body

HTTP Request-Line

HTTP/1.1 404 Forbidden

1. HTTP version

   HTTP의 버전 (현재는 HTTP/2.0까지 나옴)

2. Status-code

   요청 결과를 나타내는 코드 (200, 400, 404, 304, 500 등이 있음)

3. Status-code Content

   status-code의 **상세 내용**을 나타냄

HTTP Response 구성 방법

1. Set Body
   - Request의 데이터를 다룬 후의 결과 데이터를 넣는다.

2. Set Header
   - HashMap 자료형에 필요한 키와 값을 넣어준다.
   - 특히 File I/O에선 Content-Type과 Content-Length가 중요

3. Set status-line
   - 결과를 넣고 클라이언트에게 전송해주면 된다.

```
public String set(){
    StringBuffer response = new StringBuffer();
    this.putStatusCode(this.status_code);
    this.content_info = this.utility.getContentInfo(this.path);
    this.fillHeaders(response);

    response.append("₩r₩n").append(this.body);
    return response.toString();
}

private void fillHeaders(StringBuffer response){
    this.setDefaultHeaders();
    response.append(this.version).append(" ").append(this.status_code)
            .append(" ").append(this.status_info).append("₩r₩n");

    Set keys = headers.keySet();
    for(Iterator iterator = keys.iterator();iterator.hasNext();){
        String key = (String) iterator.next();
        String value = (String)this.headers.get(key);
        response.append(key).append(": ").append(value).append("₩r₩n");
    }
}
```

“Serve static content” 구현

1. **case : Text file**
   - 주로 String을 다루기 때문에, 출력은 PrintWriter로하면 쉽게 해결
   - PrintWriter의 print() 메서드 사용 후, 반드시 flush() 메서드 사용

2. **case : Binary file**
   - 입력 – InputStream , FileInputStream 사용
   - 출력 – OutputStream, PrintStream 사용

바이너리 파일 처리 메서드

```
private void sendBinaryPayload() throws IOException {
    try {
        this.writer = new PrintWriter(this.out);
        File f = new File( pathname: file_static_path + this.request_path);
        long flen = f.length();
        this.response.putHeader("Content-Length", Long.toString(flen));
        this.response.putHeader("Cache-Control","no-cache");

        String res = this.response.set();
        this.writer.append(res);
        this.writer.flush();

        byte[] b = new byte[MAXLEN];
        FileInputStream fis = new FileInputStream( name: file_static_path + this.request_path);
        int len = fis.read(b, off: 0,MAXLEN);
        while (len != -1) {
            this.out.write(b,  off: 0, len);
            len = fis.read(b);
        }

        this.out.flush();
        this.writer.close();
        fis.close();
    } catch (FileNotFoundException e) {
        this.sendFobiddenMessage();
    }
}
```

“Dynamic generated contents”

- 주로 Query String이나 Request Body로 온 데이터 처리
- DB 쿼리 처리, cgi파일 처리 등등 종류가 다양함

```
private void sendPayload(String data){
    this.writer = new PrintWriter(this.out);
    this.response.setBody(data);

    String res = this.response.set();
    this.writer.print(res);

    this.writer.flush();
    this.writer.close();
}
```

```
Server server = new Server( port: 8000);
server.setStaticPath("C:\\KeonHee\\framework\\test");

server.route( path: "/image1.jpg", method: "GET", new RouteTask() {
    @Override
    public String task(HttpRequest request, HttpResponse response) {
        return null;
    }
});

server.route( path: "/message", method: "POST", new RouteTask() {
    @Override
    public String task(HttpRequest request, HttpResponse response) {
        JSONObject req = request.json();
        String message = (String)req.get("content");

        String res_msg ;
        if(message.equals("hello"))
            res_msg = "hello:)";
        else res_msg = "hungry :(";

        JSONObject res = new JSONObject();
        JSONObject temp = new JSONObject();
        temp.put("text", res_msg);
        res.put("message", temp);

        return response.jsonify(res);
    }
});

server.run();
```

GET /image1.jpg Routing Part

- Serve static contents

POST /message Routing Part

- Serve dynamic generated contents

챗봇 테스트 코드로 **테스트**

POST /message Routing Part testcase

Client

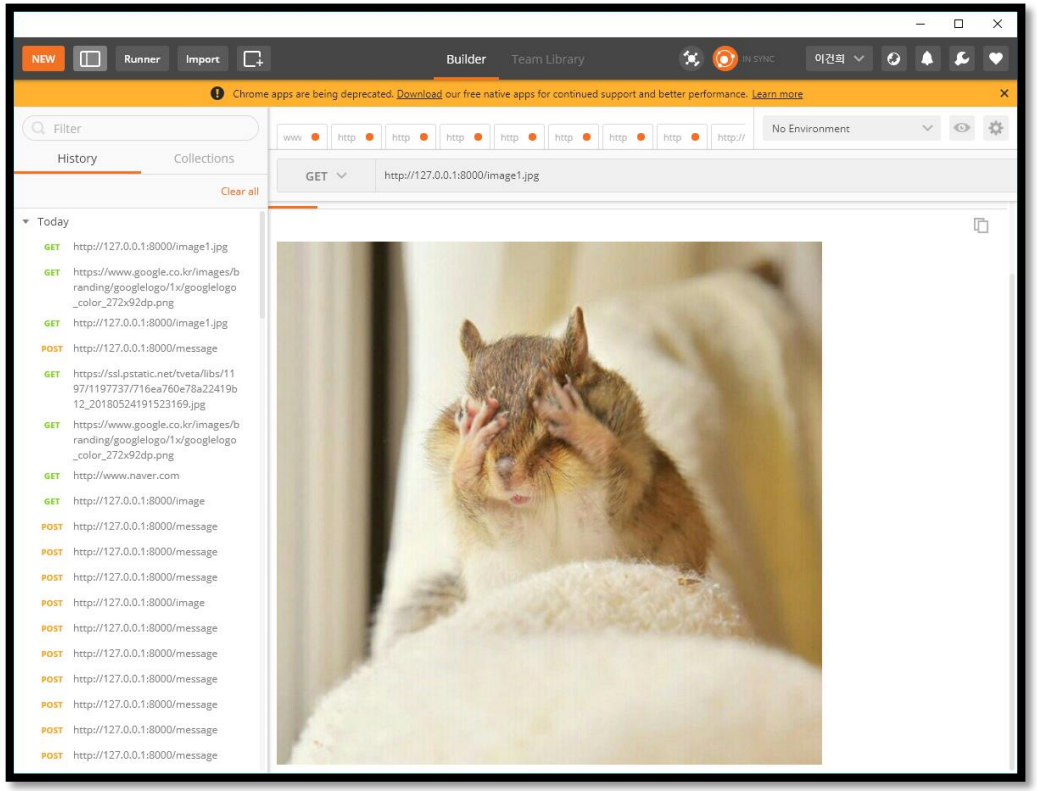
```
C:₩Python35₩python.exe C:/Users/user/IoT-Pet-Home-System/test/client_kakao.py
User >> 안녕하세요
Server << 반갑습니다:)
User >> 반가워요
Server << 배고파요 :(
User >>
```

Server

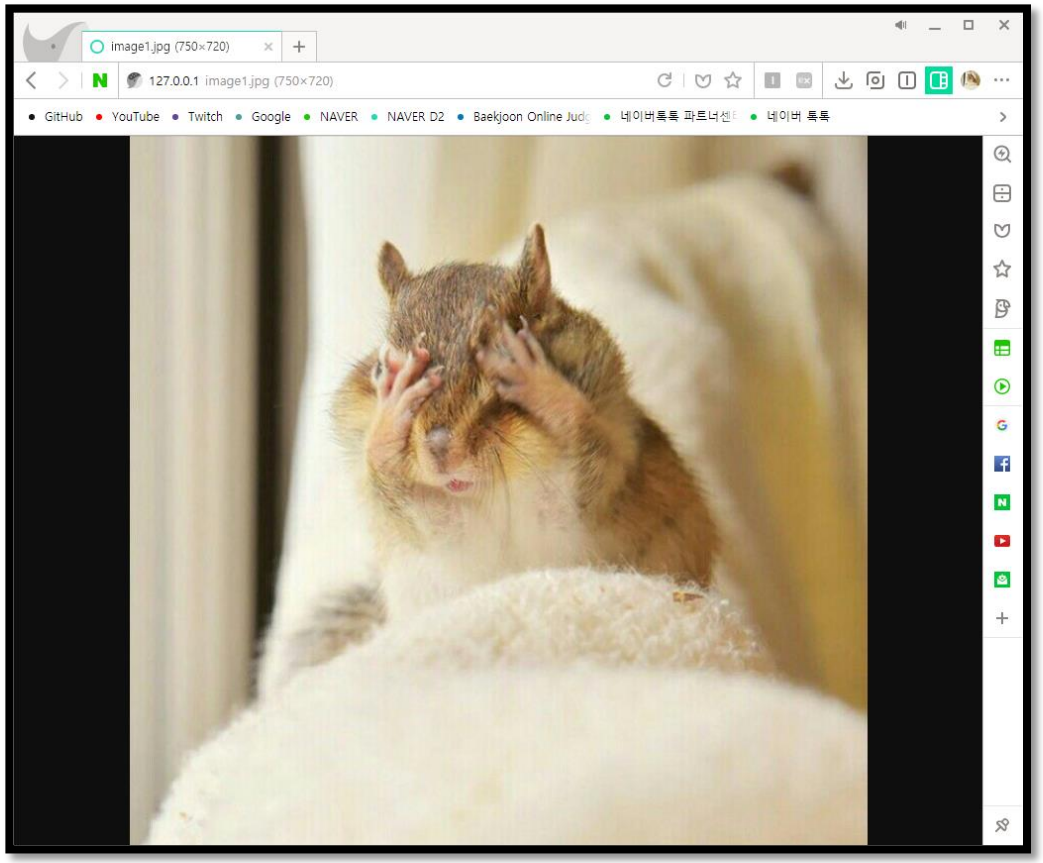
```
"C:₩Program Files₩Java₩jdk1.8.0_101₩bin₩java.exe" ...
+0.0.0.0            [Tue, 29 May 2018 02:05:24 KST] - POST      /message        200 - OK
+0.0.0.0            [Tue, 29 May 2018 02:05:32 KST] - POST      /message        200 - OK
```

이미지 요청 테스트

GET /image1.jpg Routing Part testcase



Postman

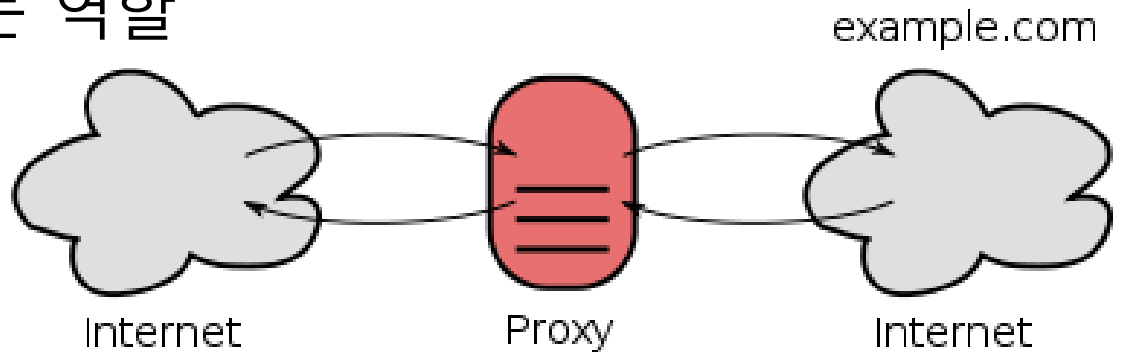


Whale Browser

추가적으로 고려해야될 사항

Proxy & Cache

- 프록시 서버 (Proxy server)
  - 클라이언트가 서비스에 간접적으로 접근하게 하는 서버
  - Load-Balancer, Redirection의 역할을 함
- 캐시 (Cache)
  - 프록시 서버는 이를 통해 요청한 데이터를 저장
  - 전송 시간 감소 및 트래픽을 줄이는 역할



TLS – Transport Layer Security

- HTTP 자체로서는 보안이 매우 취약
- 흔히 알고있는 HTTPS 프로토콜을 사용
- SSL 인증서를 통해 이용가능

TLS – Transport Layer Security

Authentication

- 웹 서버 내에서 자신이 누군가인지를 확인하는 절차
- 권한 부여에 관련되어 있음.
- BASIC, DIGEST, Form base, SSL Client 등

참고자료

- Apache 웹 서버 문서 - https://httpd.apache.org/docs/2.2/ko/programs/httpd.html
- 생활코딩 - https://opentutorials.org/course/228/4894

구현 시 도움이 되었던 좋은 자료들

- apache / httpd : https://github.com/apache/httpd
- pallets / flask : https://github.com/pallets/flask
- spring-project / spring-framework : https://github.com/spring-projects/spring-framework

QnA