15장 엔터티(표현:렙러젠테이션)와 인코딩

15.1 메시지는 컨테이너 ,엔터티는 화물

- HTTP 메시지를 인터넷 운송 시스템의 컨테이너라고 생각하면 HTTP 엔터티는 메시지의 실질적인 화물이다.
- 엔터티 헤더는 겨우 18자에 불과함(Content-Length : 18) 플레인텍스트 문서(Content-Type:text/plain)를 의미한다.항상 그렇듯이, 빈줄(CRLF)헤더 필드와 본문의 시작을 나눈다.

헤더	설명
Content-Type	객체의 종류
Content-Length	메시지 길이
Content-Language	대응되는 자연어
Content-Encoding	압축 등 변형
Content-Location	객체의 또다른 위치
Content-Range	엔터티가 전체에서 어느 부분에 해당하는지
Content-MD5	본문에 대한 체크섬
Last-Modified	생성 혹은 수정된 날
Expires	엔터티가 유효하지 않기 시작하는 시
Allow	어떤 요청 메서드가 허용되는지 (ex. GET, HEAD)
Etag	인스턴스에 대한 고유 식별
Cache-Control	캐시되는 방법

- Etag와 Cache-Control는 엔터티 헤더로는 정의되어 있지 않으나, 중요한 것들이다.

15.1.1 엔터티 본문

- 엔터티 본문은 가공되지 않은 데이터만을 담고 있다. 나머지 정보는 헤더에 담긴다.
- 엔터티 본문은 가공되지 않은 날 데이터에 불과하기 때문에 엔터티 헤더는 그 데이터의 의미에 대해 설명할 필요가 있다.
- 엔터티 본문은 헤더 필드의 끝을 의미하는 빈 CRLF 줄 바로 다음부터 시작된다.
- 콘텐츠 내용이 어떤 내용이든 상관없이 항상 CRLF 바로 다음에 위치한다.

15.2 Content-length : 엔터티의 길이

- Content-Length 헤더는 메시지의 엔터티 본문의 크기를 바이트 단위로 나타낸다.
- 어떻게 인코딩 되었든 상관없이 크기를 표현할 수 잇다.
- 메시지를 청크 인코딩으로 전송하지 않는 이상, 엔터티 본문을 포함한 메시지에서는 필수적으로 있어야한다.
- Content-Length는 서버 충돌로 인해 메시지가 잘렸는지 감지하고자 할 때와 지속 커넥션을 공유하는 메시지를 올바르게 분할하고자 할 때 필요하다.

15.2.1 잘림 검출

- 이전에는 커넥션이 정상적으로 닫힌 것인지 메시지 전송중에 서버에 충돌이 발생한 것인지 구분을 못함
- 클라이언트는 메시지 잘림을 검출하기 위해 Content-Length를 필요로 한다.
- 캐싱 프락시 서버는 명시적으로 Content-Length 헤더를 갖고 있지 않은 HTTP 본문을 보통 캐시하지 않는다.

15.22 잘못된 Content-Length

- Coontent-Length가 잘못된 값을 담고 있을 경우 큰 피해를 유발 시킬 수 있다.
- 공식적으로 HTTP/1.1 사용자 에이전트는 잘못 된 길이를 받고 그 사실을 인지 했을때 사용자에게 알려주게 되어있다.

15.2.3 Content-Length와 지속 커넥션(Persistent Connection)

- Content-Length는 지속 커넥션을 위한 필수다. 응답이 지속 커넥션을 통해서 온 것이라면 , 또 다른 HTTP 응답이 즉시 그 뒤를 이을 것이다.
- Content-Length헤더는 클라이언트에게 메시지 하나가 어디서 끝나고 다음 시작은 어디인지 알려준다. 커넥션이 지속적이기 때문에 , 클라언트가 커넥션이 닫힌 위치를 근거로 메시지 끝을 인식하는 것을 불가능하다.
- HTTP 애플리케이션은 Content-Length 헤더 없이는 어디까지가 엔터티 본문이고 어디부터가 다음 메신지 인지 알 수 없을 것이다.

15.2.4 콘텐츠 인코딩

- HTTP는 보안을 강화하거나 압축을 통해 공간을 절약할 수 있도록, 엔터티 본문을 인코딩 할 수 있게 해준다. 만약 본문의 콘텐츠가 인코딩되어 있다면, Content-Lengh헤더는 인코딩 되지 않은 원본의 길이가 아닌 인코딩된 본문의 길이를 바이트 단위로 정의한다.
- Content-Lengh헤더에 인코딩 전의 크기를 보내는 경우 심각한 오류를 유발한다.
- HTTP/1.1 명세로는 원 본문의 길이를 알 수 없기 때문에, 클라이언트는 자신이 올바르게 디코딩했는지 검증할 수 없다.

15.2.5 엔터티 본문 길이 판별을 위한 규칙

- 본문이 없는 HTTP메시지에서는 , 본문 계산을 위한 Content-Length헤더가 무시되며 Content-Length 헤더는 부가 정보에 불과하다.(실제 본문 길이를 서술하지 않는다)
- 메시지가 Transfer-Encoding 헤더를 포함하고 있다면 메시지가 커넥션이 닫혀서 먼저 끝나지 않는 이상 엔터티는 “0바이크 청크”라 불리는 특별한 패턴으로 끝나야 한다.
- 메시지가 Content-Length 헤더를 갖는다면,Transfer-Encoding 헤더가 존재하지 않는 이상 Content-Length값은 본문의 길이를 담게 된다. Transfer-Enconding 헤더 필드를 갖고 있는 메시지를 받았다면 반드시 Content-Length 헤더를 무시해야 한다. 이유는 전송 인코딩은 엔터티 본문을 표현하고 전송하는 방식을 바꿀 것이기 때문이다.
- 메시지가 “miultipart/byteranges” 미디어 타입을 사용하고 엔터티 길이가 별도로 정의되지 않았다면(Content-Length헤더로), 멀티 파트 메시지의 각 부분은 각자가 스스로의 크기로 정의할 것이다. 미디어 타입은, 수신자가 이것을 해석할 수 있다는 사실을 수신자가 알기 전까지 절대로 보내지 말아야 한다.이 규칙에도 해당 되지 않는다면 엔터티는 커넥션이 닫힐 때 끝난다.
- 클라이언트는 클라이언트 메시지가 끝났다라는 신호를 위해 커넥션을 닫을 수 없다.
- HTTP/1.0 애플리케이션과의 호환을 위해, 엔터티 본문을 갖고 있는 HTTP1.1 요청은 반드시 유효한 Content-Length 헤더도 갖고 있어야 한다.

15.3 엔터티 요약

- 엔터티본문 데이터에 대한 의도하지 안는 변경을 감지하기 위해, 최초 엔터티가 생성될 때 송신자는 데이터에 ㅇ대한 체크섬을 생성할 수 있으며, 수신자는 모든 의도하지 않은 엔터티의 변경을 잡아내기 위해 그 체크섬으로 기본적인 검사를 할 수 있다.

- Content-MD5 헤더는 서버가 엔터티 본문에 MD5 알고리즘을 적용한 결과를 보내기 위해 사용 된다.
- Content-MD5 헤더는 전송 인코딩이 아직 적용 되지 않은 엔터티 본문에 대한 MD5를 담고 있다.
- 메시지의 무결성을 검증을 하기 위해 전송 인코딩을 디코딩 한후 MD5를 계산한다.
- MD5는 문서 위치를 빠르게 알아내고 콘텐츠의 중복 정장을 방지하기 위해 해시 테이블의 키로 이용될 수 있으며 이런 활용에도 불구하고 Content-MD5 헤더는 그다지 자주 전송되지 않는다.

15.4 미디어 타입 과 차셋(Cahrset)

- Content-Type 헤더 필드는 엔터티 본문의 MIME 타입을 기술한다.
- 클라이언트 애플리케이션은 콘텐츠를 적절히 해독하고 처리하기 위해 MIME 타입을 이용한다.
- Content-Type의 값은 인터넷 할당 번호 기관에 등록된 표준화된 MIME 타입이다. MIME 타입은 주 미디어 타입(텍스트,이미지,오디오 등)으로 시작해서 뒤이어 빗금(/) 그리고 미디어 타입더 구체적으로 서술하는 부 타입(subtype)으로 구성된다.
- Content0Type헤더가 원본 엔터티 본문의 미디어 타입을 명시한다는 것은 중요하다.

미디어 타입	설명
text/html	엔터티 본문은 HTML 문서
text/plain	엔터티 본문은 플레인 텍스트 문서
image/gif	엔터티 본문은 GIF이미지
image/jpeg	엔터티 본문은 JPEG 이미지
audio/x-way	엔터티 본문은 WAV 음향 데이터를 포함
modle/vrml	엔터티 본문은 삼차원 VRML 모델
application/vnd.ms-powerpint	엔터티 본문은 마이크로소프트 파워 포인트 프레젠테이션
multipart/byteranges	엔터티 본문은 여러부분으로 나뉘는데, 각 부분은 전체 문서의 특정 범위(Bite 단위)를 담고 있다.
message/http	엔터티 본문은 완전한 HTTP 메시지를 담고 있다.

15.4.1 텍스트 매체를 위한 문자 인코딩

- Content-Type 헤더는 내용 유형을 더 자세히 지정하기 위한 선택적인 매개변수도 지원한다.

15.4.2 멀티파트 미디어 타입

- MIME”멀티파트”이메일 메시지는 서로 붙어있는 여러 개의 메시지를 포함하며, 하나의 복합 메시지로 보내진다. 각 구성요소는 자족적으로 자신에 대한 서술하는 헤더를 포함한다. 여러 구성요소들이 이어져 있고, 문자열 하나로 서로의 경계가 식별 된다.
- HTTP는 멀티파트 본문도 지원한다. 그러나 일반적으로는 폼을 채워서 제출 할때와 문서의 일부분을 실어 나르는 범위 응답을 할때 두가지 경우에만 사용된다.

15.4.3 멀티 파트 폼 제출

- HTTP 폼을 채워서 제출하면 , 가변 길이 텍스트 필드와 업로드 될 객체는 각각 멀티 파트 본문을 구성하는 하나의 파트가 되어 보낸진다.
- 멀티 파트 본문은 여러 다른 종류와 길이의 값을 채워진 폼을 허용한다.

15.4.4 멀티 파트 범위 응답

- 범위 요청에 대한 HTTP 응답 또한 멀티 파트가 될 수도 있다. 그러한 응답 Content-Typ: multiparbteranges 헤더 및 각각 다른 범위를 담고 있는 멀티파트 본문이 함께 온다.

15.5 콘텐츠 인코딩

- HTTP 애플리케이션은 때때로 콘텐츠를 보내기 전에 인코딩을 하려고 한다
- 서버는 전송시간을 줄이기 위해 압축을 할 수 있으며 암호화 하거나 뒤썼어 보낼 수도 있다.

15.5.1 콘텐츠 인코딩 과정

1. 서버가 Content-Type과 Content-Length 헤더를 수반한 원본 응답 메시지를 생성한다.

2. 인코딩 서버가 인코딩 메시지를 생성한다. Content-Type은 같지만 Content-Length는 다르다.
3. 서버는 Encoding 헤더를 인코딩된 메시지에 추가하여, 수신측 애플리케이션이 내용을 디코딩할 수 있도록 한다.
4. 수신 측 프로그램은 인코딩된 메시지를 받아서 디코딩하고 원본을 얻는다.

15.5.2 콘텐츠 인코딩 유형

- HTTP는 몇 가지 표준 콘텐츠 인코딩 유형을 정의하고 확장 인코딩으로 인코딩을 추가하는 것도 허용한다.

콘텐츠 인코딩 값	설명
gzip	엔터티에 GNU zip인코딩이 적용되었음을 의미한다.
compress	엔터티에 대해 유닉스 파일 압축 프로그램인 “compress”가 실행되었음을 의미한다.
deflate	엔터티가 zlib포맷으로 압축되었음을 의미한다.
identity	엔터티에 어떤 인코딩도 수해되지 않았음을 의미한다. content-Encoding 헤더가 존재하지 않는다면 이 값 인 것을 간주한다.

- gzi,compress,deflat 인코딩은 전송되는 메시지의 크기를 정보의 손실 없이 줄이기 위한 무 손실 압축 알고리즘이다.
- gzip 일반적으로 가장 효율적이고 가장 널리 쓰이는 압축 알고리즘이다.

15.5.4 Accep-Encoding 헤더

- 우리는 클라이언트가 해독할 수 없는 방법으로 서버가 콘텐츠를 인코딩하는 것을 원하지 않는다.
- 서버에서 클라이언트가 지원하지 않는 인코딩을 사용하는 것을 막기 위해, 클라이언트는 자신이 지원하는 인코딩의 목록은 Accept-Encoding 요청 헤더를 통해 전달한다.
- 만약 HTTp 요청 Accept-Encoding 헤더를 포함하지 않는다면, 서버는 클라이언트가 어떤 인코딩이든 받아들일 수 있는 것으로 간주한다.(Accept-Encoding:*을 전달한 경우도 같다)
- 클라이언트는 각 인코딩에 Q(quality) 값을 매개변수로 더해 선호도를 나타낼 수 있다. (Q값의 범위는 가장 원치 않음을 의미하는 0.0에서 가장 선호함을 의미하는 1.0까지다. 토큰”*”은 그 외 모두를 의미한다.)

15.6 전송 인코딩과 청크 인코딩

- 콘텐츠 인코딩은 콘텐츠 포맷과 긴밀하게 연관되어있다. 전송 인코딩 또한 엔터티 본문에 적용 되는 가역적 변환이지만 그들은, 구조적인 이유 대문에 때문에 적용되는 것이며 콘텐츠의 포맷과 독릭접이다.

15.6.1 안전한 전송

- 전송 인코딩은 다른 프로토콜에서도 네트워크를 통한 “안전한 전송”을 위해 존재했다.

알 수 없는 크기

- 콘텐츠를 먼저 생성하지 않고서는 메시지 본문 최종 크기를 판단할수 없다. 이서버들은 그 사이즈를 알기 전에 데이터의 전소을 시작하려고 한다. HTTP는 데이터 앞서 Content-Length 헤더 요구하기 때문에, 몇몇서버는 데이터의 끝을 알리는 특별한 종결 꼬리말을 포함시켜 전송인코딩으로 데이터를 보내려 시도한다.

보안

- 공용 전송 네트워크로 메시지 콘텐츠를 보내기 전에 전송 인코딩을 사용해 알아보기 어렵게 뒤섞어 버리는 방법도 있다
- SSL과 같은 유명한 전송 계층 보안 방식이 있기 때문에 인코딩 보아는 흔하지 않다

15.6.2 Transfer-Encoding 헤더

- 전송 인코딩을 제어하고 서술하기 위해 정의된 헤더는 단 두개 뿐이다.
- Transfer-Encoding
    - 안전한 전송을 위해 어떤 인코딩이 메시지에 적용되었는지 수신자에게 알려준다.
- TE : 어떤 확장된 전송 인코딩을 사용할 수 있는지 서버에게 알려주기 위해 요청 헤더에 사용한다.

15.6.3 청크 인코딩

- 청크 인코딩은 메시지를 일정 크기의 청크 여럿으로 쪼갠다. 서버는 각 청크를 순차적으로 보낸다.
    - 청크 인코딩을 이용하면 메시지를 보내기 전에 전테 크기를 알 수 없다.

    - 본문이 동적으로 ㅅㅇ성되고 서버는 그중 일부를 버퍼에 담은 뒤 한 청크를 컹크의 크기와 함께 보낼 수 있다.
    - 본문 전체를 보내때까지 이 단계를 반복한다.
- **청크와 지속 커넥션**
    - 서버는 크기가 0인 청크로 본문이 끝났음을 알리고 다음 응답을 위해 커넥션을 열린 채로 유지할 수 있다.
    - 청크의 길이가 0인 것은 본문의 끝을 의미한다.
    - 청크 요청이 411Length required 응답으로 거절당하는 것에 대비해야 한다.

청크 인코딩된 메시지의 트레일러

- 아래 조건중 만족하면 청크 에시지에 트레일러를 추가할 수 있다.
    - 클라이언트의 TE 헤더가 트레일러를 받아 들일 수 있음을 나타내고 있는 경우
    - 트레일러가 응답을 만든 서버에 의해 추가되었으며, 그 트레일러의 콘텐츠는 클라이언트가 이해하고 사용할 필요가 없는 선택적인 메타데이터 이므로 클라이언트가 무시될 수 있다.
- 트레일러 본문의 콘텐츠가 먼저생성되어야 한다거 하는 등의 이유로 메시지 시작 시점에서 그 값을 알수 없는 추가적인 헤더필드를 담을 수 있다.
- 메시지 다음에 오게될 헤더들을 나열하는 Trailer헤더를 포함하고 있다. 마지막 청크 다음에 Trailer헤더에 나열했던 헤더들이 온다.

15.6.4 콘텐츠와 전송 인코딩의 조합

- 콘텐츠 인코딩과 전소 인코딩이 동시에 사용할 수 있다.

15.6.5 전송 인코딩 규칙

- 전송 인코딩은 HTTP/1.1에서의 새로운 기능이므로, 비 HTTP/1.1에 전송 인코딩 메시지를 보내지 않도록 주의해야 한다.
    - 전송 인코딩의 집합은 반드시 chunked를 포함해야한다. 유일한 예외는 메시지가 커넥션의 종료로 끝나는 경우이다.
    - 청크 전송 인코딩이 사용 되었다면, 메시지 본문에 적용된 마지막 전송 인코딩이 존재해야한다.
    - 청크 전송 인코딩은 반드시 메시지 본문에 한번 이상 적용 되어야한다.

15.7 시간에 따라 바뀌는 인스턴스

- 웹 객체는 정적이지 않다. 같은 URL은 시간에 따라 다른 버전의 객체를 가리킬 수 있다.
- HTTP 프로토콜은 어떤 특정한 종류의 요청이나 응답을 다루는 방법들을 정의하는데, 이것은 인스턴스 조작이라고 불리며 객체의 인스턴스에 작용한다.
    - 인스턴스 조작의 종류 : 범위 요청과 델타 인코딩이 있다.
- 클라이언트가 자신이 갖고 있는 리소스의 사본이 서버가 갖고 있는 것과 정확히 같은지 판단하고 상황에 따라서 새 인스턴스를 요청할 수 있는 능력을 가질 것을 요구한다

15.8 검사기와 신선도

- 클라이언트는 처음에 리소스의 사본을 갖고 있지 않으므로, 서버에게 달라고 요청을 보낸다.
- 클라이언트는 반드시 서버에게 최신 사본을 요구 해야 한다. 서버의 문서가 변경되지 않았다면 클라이언트는 다시 받을 필요 없다.
- 조건부 요청 클라이언트가 서버에게 자신이 갖고 있는 버전을 말해주고, 검사기로 자신의 버전이 더 이상 유효하지 않을 때만 새로운 사본을 보내 달라고 요청하는 것

15.8.1 신성도

- 신선도는 서버는 클라이언트에게 얼마나 오랫동안 콘텐츠를 캐시하고 그 내용이 가정할 수 있는지에 대한 정보를 줄 것이다.
- Expires : 헤더는 문서가 만료되어 더 이상 신선하다고 간주 할 수 없게 되는 정확한 날짜를 명시한다. Expires헤더의 문법은 다음과 같다

```
Expires : Sun Mar 18 23:5959 GMT 2001
```

- Cache-Control : 서버와 클라이언트 양족에서, 더 많은 지시자들과 함께, 단지 수명이나 유효기간뿐 아니라 선도(신선도)를 서술하기 위해 사용된다.
    - **no-cache**: 캐시는 응답을 캐시에 저장할 수 있지만, 이전에 검증되지 않은 상태로 다시 확인하도록 요청해야 합니다.
    - **no-store**: 캐시는 응답을 저장하면 안 됩니다. 모든 요청은 원 서버로 전달되어야 합니다.

- **max-age**: 캐시된 응답을 유효하게 유지할 시간(초 단위)을 나타냅니다.
- **s-maxage**: 공유 캐시에 대한 max-age를 재정의합니다.

15.8.2 조건부 요청과 검사기

- 캐시는 자신이 갖고 있는 사본을 신선한 것으로 만들 필요가 있다.
- 신선도를 검사하지 않고 사본을 요청할 경우 캐시와 서버에 불필요한 부하를 주고 느려진다.
- 조건부 요청은 “if-”로 시작하는 조건부 헤더에 의해 구현된다.
  - 조건부 헤더는 If-Modified-Since이다. 조건부 헤더는 조건이 참일 때만 수행되도록 한다.
- HTTP는 검사기를 약한 검사기와 강한 검사기의 두 가지로 분류한다.
- If-None-Match 조건부 헤더는 문서의 Etag값을 평가한다. ETag는 특별한 키워드이거나 엔터티와 관련된 버전 식별 태그이다.
  - 약한 검사기 : 리소스의 인스턴스를 고유하게 식별하지 못하는경우도 있다.(, 최,서버는 태그 앞에 “W/”를 붙임으로써 “약한” 엔터티 태그임을 알린다)
    - 바이트 단위의 크기
    - 종 변경 시각
  - 강한 검사기 : 언제나 고유하게 식별한다. 리소스의 콘텐츠에 대한 암호 체크섬(MD5)는 강한 검사기다.
    - ETag
    - MD5 체크섬

15.9 범위 요청

- HTTP는 클라이언트가 문서의 일부분이나 특정 범위만 요청할 수 있도록 해준다.
- 범위요청을 이용하면 HTTP 클라언트는 받다가 실패한 엔터티를 일부 혹은 범위로 요청함으로써 다운로드를 중단된 시점에서 재개할 수 있다.

```
GET /bigfile.html HTTP/1.1
Host : WWW.joes-hardware.com
```

```
range : bytes = 4000-
User-Agent : Mozilla/4.61 [en](WinNt;I)
```

- range 헤더를 이용해 문서의 일부분이나 특정 범위만 요청한다.
- range 헤더는 P2P파일 공유 클라이언트가 멀티 미디어 파일의 다른 부분을 여러 다른 피어로부터 동시에 다운로드 받을때도 사용한다.

15.10 델타 인코딩

- 델타 인코딩은 객체 전체가 아닌 변경된 부분에 대에서만 통신하여 전송량을 최적화하는 HTTP 프로토콜의 확장이다.
- 델타 인코딩은 일종의 인스턴스 조작인데, 왜냐하면 어떤 객체의 특정 인스턴스들에 대한 클라이언트와 서버 사이의 정보교환에 의존하기 때문이다.
- RFC 3329는 델타 인코딩에 대해 기술하고 있다
  - 클라이언트는 페이지의 어떤 버전을 갖고 있는지 서버에게 말해주어야한다.
  - 클라이언트는 자신이 갖고 있는 현재 버전에 델타를 적용하기 위해 어떤 알고리즘을 알고 있는지 서버에 말해 주어야한다.
  - 서버는 자신과 클라이언 버전이 맞는지 확인한다.
  - 서버와 클라이언트는 버전 사이의 델타를 계산할 것인지 체크한다.
  - 델타를 계산해서 클라이언트에게 보내주고, 서버가 델타를 보내주고 있음을 클라이언에게 알려주고, 페이지의 최신버전에 대한 새 식별자를 명시해야한다.
  - IfNone-Match 헤더는 최신 버전의 페이지를 보내달라는 클라이언트의 방식이다. 그러면 서버는 클라이언트에게 최신 버전 전체를 보내게 될 것이다.그러나 A-IM은 자신이 페이지에 대한 델타를 받아들일 수 있음을 알려 줄 수 도 있다.

15.10.1 인스턴스 조작 , 델타 생성기 그리고 델타 적용기

- 클라이언트는 A-Im 헤더를 사용해서 자신이 받아들일 수 있는 인스턴스 조작의 종류를 명시할 수 있다.
- 서버는 IM 헤더에 사용한 인스턴스 조작의 종류를 명시할 수 있다.
- 서버의 “델타 생성기”는 기저 문서와 그 문서의 최신 인스턴스를 취하여 클라이언트의 A-IM 헤더에 지정된 알고리즘을 이용해 둘 사이의 델타를 계산한다.

- 델타 인코딩은 전송 시간을 줄일 수 있지만 구현하기가 까다로울 수 있다.