RFC 2616

Hypertext Transfer Protocol -- HTTP 1.1

출처: 한국전자통신연구소
편집: 서동규 / NHN NEXT

RFC2616 원문
http://www.rfc-base.org/rfc-2616.html

한국전자통신연구소 변역문서
https://docs.google.com/file/d/0B1S1kwnuRJ6jMmNaVDZZdWNKbHM

Hypertext Transfer Protocol -- HTTP 1.1

요약

하이퍼텍스트 전송 규약(HTTP)은 분산 정보 시스템, 종합 정보시스템 및 하이퍼미디어 정보시스템에서 사용하는 응용 계층 규약으로서 요구 방법의 확장을 통해서 네임서버와 분산 객체 관리 시스템과 같은 수많은 작업에 사용될 수 있는 보편적인 객체지향형 규약이다. HTTP 는 어떤 문서의 데이터 표현 형식을 규정하고 협상하여 전송 중인 데이터와 무관하게 시스템을 구축할 수 있게 한다.
HTTP 는 1990 년 이후 World-Wide Web 범 세계 정보 이니셔티브에 의하여 사용되고 있다. 이 규격은 "HTTP/1.1"로 언급되는 규약을 정의하고 있다.

목차

[Page 6]

1 서론

1.1 목적

하이퍼텍스트 전송 규약(HTTP)은 분산 정보시스템, 종합 정보시스템 및 하이퍼미디어 정보시스템에서 사용하는 응용계층의 규약이다. HTTP 는 1990 년 이후 World-Wide Web 범 세계 정보 이니셔티브에 의하여 사용되고 있다. "HTTP/0.9"로 언급되는 HTTP 의 첫 버전은 인터넷 상에서 저장되어 있는 원래 데이터(raw data)를 전송하기 위한 단순한 규약이었다. RFC 1945 [6]이 규정한 HTTP/1.0 은 메시지를 전송되는 문서 데이터에 대한 메타 정보 및 요구/응답 용어의 변경자를 포함하는 MIME 과 유사한 메시지의 형식으로 사용할 수 있도록 함으로써 규약을 향상시켰다. 그러나 HTTP/1.0 은 계층적 프락시(hierarchical proxies), 캐시, 지속적인 연결의 필요성 및 가상 호스트(virtual host) 등의 영향을 충분히 고려하지 않았다. 또한 "HTTP/1.0"을 지원한다고 하면서도 규약의 불완전 구현 및 오해에 의한 잘못된 구현 등에 의해 응용 프로그램 사이에 종종 문제가 발생하였기에 상호 협상할 수 있는 응용 프로그램이 상대방의 진정한 성능을 파악할 수 있도록 규약 버전을 갱신할 필요가 생겼다.

이 규격은 "HTTP/1.1"로 불리우는 하이퍼텍스트 전송 규약을 정의한다. 이 규약은 기능을 신뢰할 수 있도록 구현하기 위해 HTTP/1.0 보다 더 엄격한 필요 조건을 포함하고 있다.

실제적인 정보 시스템은 단순한 조회보다 검색, 프런트-엔드(front-end) 갱신 및 주석 달기 등 많은 기능을 필요로 한다. HTTP 는 요구의 목적을 표시하는 일련의 개방된 method 를 (open-ended set of methods) 허용한다. 이 규약은 보편적 자원 식별자(URI) [3][20], 자원 위치 (URL) [4] 또는 자원 이름(URN)이 제공하는 참고 방법에 따라 method 를 적용할 자원을 지칭하는 데 사용한다. 메시지는 다용도 인터넷 메일 확장(MIME)에서 정의된 것처럼 인터넷 메일에서 사용되는 것과 유사한 형식으로 전송된다.

HTTP 는 사용자 에이전트, 프락시/게이트웨이와 SMTP [16], NNTP [13], FTP [18], Gopher [2], 및 WAIS [10] 등을 지원하는 다른 인터넷 시스템 사이의 통신을 위한 범용 규약으로서 사용된다. 이러한 방식으로 HTTP 는 기본적인 하이퍼미디어가 다양한 애플리케이션의 자원에 접근할 수 있도록 한다.

1.2 필요 조건

이 규격은 각각의 특별한 필요 조건의 중요도를 정의할 때 RFC 1123 [8]와 동일한 용어를 사용한다. 이러한 용어는 다음과 같다.

MUST
  이 단어 또는 "요구된"이라는 형용사는 해당 항목이 규격의 절대적인 필요 조건임을 의미한다.

[Page 7]

SHOULD
  이 단어 또는 "추천된"이라는 형용사는 특정 상황에서 해당 항목을 무시할 합당한 이유가 있을 수 있다는 것을 의미한다. 그러나 충분히 함축적 의미를 이해해야 하고 다른 방법을 선택하기 전에 사례를 충분히 검토해야 한다.

MAY
  이 단어 또는 "선택적"이라는 형용사는 해당 항목이 진정으로 선택적이라는 것을 의미한다. 한 판매회사는 특정 항목을 특정 시장이 요구하기 때문에 또는 예를 들어 제품의 기능을 향상시켜 주기 때문에 다른 판매 회사와 달리 동일한 항목을 포함할 수 있다.

구현 방법이 하나 또는 그 이상의 MUST 규약 필요 조건을 충족시켜 주지 못하면 규약에 따르지 않는 것이다. 구현 방식이 모든 MUST 및 SHOULD 필요 조건을 충족한다면 "무조건적으로 충족한다"고 할 수 있고, 모든 MUST 필요 조건을 충족하지만 모든 SHOULD 필요 조건을 충족하지 못한다면 "조건적으로 충족한다"고 할 수 있다.

1.3 용어

이 규격은 HTTP 통신의 참여자 및 객체가 수행하는 역할을 지칭하는 몇몇 용어를 사용하고 있다.

connection(연결)
  통신을 목적으로 두 프로그램 간에 설정된 전송 계층의 가상적 회로

message(메시지)
  HTTP 통신의 기본 전송 단위. 4 장에 규정된 의미론을 따르는 구조적인 데이터 표현 형태이며, 일련의 8 비트(octets)로 구성되어 있고 연결을 통하여 전송된다.

request(요구)
  5 장에 규정된 HTTP 요구 메시지.

response(응답)
  5 장에 규정된 HTTP 응답 메시지.

resource(자원)
  3.2절에 규정되어 있는 URI에 의하여 식별되는 네트워크 데이터 객체 또는 서비스. 자원은 다양한 표현 형태 (예를 들어 언어, 데이터 형식, 크기 및 해상도)를 지닐 수 있으며 다양한 방법으로 변형될 수 있다.

[Page 8]

entity(엔터티)
  요구나 응답 메시지의 페이로드(payload)로서 전송되는 정보. 엔터티는 7 장에서 설명된 대로 Entity-Header 필드 형태의 메타 정보 및 Entity-Body 형태의 내용으로 구성되어 있다.

representation(표현)
  12 장에서 기술한 내용 협상의 통제를 따르는 응답에 포함된 엔터티. 특정한 응답 상태와 연관된 다수의 표현 방법이 있을 수 있다.

content negotiation(내용 협상)
  12 장에서 기술한 대로 요구를 처리할 때 적절한 표현 방법을 선택하는 메커니즘. 어떠한 응답에서는 엔터티의 표현은 협상할 수 있다.(에러 응답 포함)

variant(변형자)
  자원은 특정한 경우에 자원과 관련된 하나 이상의 표현 방식을 가질 수 있다. 이러한 각각의 표현 방식을 "변형자"라고 부른다. "변형자"라는 용어를 사용한다고 해서 자원이 반드시 내용 협상의 대상인 것은 아니다.

client(클라이언트)
  요구 메시지를 전송할 목적으로 연결을 설정하는 프로그램.

user agent(사용자 에이전트)
  요구 메시지를 시작하는 클라이언트. 이것은 종종 브라우저, 편집기, 스파이더(웹을 탐색하는 로봇) 또는 다른 사용자 툴(tool)일 수 있다.

server(서버)

요구 메시지를 처리하기 위해 접속을 수신하는 애플리케이션으로서 응답 메시지를 전송한다. 어떤 프로그램이든 동시에 클라이언트와 서버가 될 수 있다. 이 규격에서 이 용어를 사용하는 것은 프로그램의 일반적인 능력을 참조하기보다는 특정한 연결을 위해 프로그램이 수행하는 역할만을 참조하는 것이다. 마찬가지로 어떠한 서버도 원서버, 프락시, 게이트웨이, 터널 등 각 요구의 성격에 따라 동작을 전환하는 역할을 할 수 있다.

origin server(원서버)
해당 자원이 보관되어 있거나 자원을 생성할 수 있는 서버.

[Page 9]

proxy(프락시)
다른 클라이언트를 대신하여 요구를 작성할 목적으로 서버와 클라이언트의 역할을 모두 수행하는 중간 프로그램. 요구는 내부적으로 처리되어 가능하면 해석되어 다른 서버로 전달된다. 프락시는 이 규격의 클라이언트와 서버의 필요 조건을 모두 구현해야만 한다.

gateway(게이트웨이)
다른 서버를 위해 중간 역할을 하는 서버. 프락시와는 달리 게이트웨이는 요구 메시지를, 요청받은 자원을 서비스하는 최종적인 원서버처럼 수신한다. 요구한 클라이언트는 자신이 게이트웨이와 통신하고 있다는 것을 알지 못할 수 있다.

tunnel(터널)
두 연결 사이를 무조건 중계하는 역할을 하는 중간 프로그램. 활성화되면 비록 HTTP 요구에 의하여 시작되지만 터널은 HTTP 통신의 참여자로 간주되지 않는다. 터널은 중계하고 있는 양 쪽의 연결이 종결되면 사라진다.

cache(캐시)
프로그램이 응답 메시지를 저장하는 로컬 저장소. 메시지 보관, 조회 및 삭제를 제어하는 하부 시스템이기도 하다. 캐시는 응답 시간, 향후 네트워크 대역폭 소모 및 동일한 요구를 감소시킬 목적으로 캐시할 수 있는 응답을 저장한다. 어떤 클라이언트나 서버도 캐시를 포함할 수 있다. 단지 터널 역할을 하는 서버는 캐시를 사용할 수 없다.

cachable(캐시할 수 있는)
응답 메시지의 사본을 저장하여 계속적인 요구 응답에 사용할 수 있으면 응답을 캐시할 수 있다고 한다. HTTP 응답의 캐시 가능 여부를 결정하는 원칙은 13 장에 정의되어 있다. 자원을 캐시할 수 있다 하더라도 캐시가 특정 요구에 대하여 캐시 된 사본을 사용할 수 있는지 여부에 대한 추가적인 제한 사항이 있을 수 있다.

first-hand(직접)
응답이 직접적으로 오며 원서버로부터 하나 또는 그 이상의 프락시를 거쳐옴으로써 발생하는 불필요한 지연이 없을 경우 응답이 직접 온다고 할 수 있다. 또한 검증이 원서버에서 직접 이루어진다면 응답이 직접 온다고 할 수 있다.

explicit expiration time(명백한 유효 시간)
원서버가 추가적인 검증 없이는 캐시에 의해 엔터티를 더 이상 되돌려 주지 않기로 한 시간. 즉, 원서버가 캐시된 데이터의 유효성을 보장할 수 있는 시간.

[Page 10]

heuristic expiration time(자동으로 설정되는 유효 시간)
분명한 유효 시간이 설정되어 있지 않을 때 캐시가 할당하는 유효 시간

age(경과 시간)
응답 메시지의 경과 시간은 원서버로부터 전송된 후, 또는 성공적으로 검증된 후의 시간.

freshness lifetime(신선한 기간)
응답의 생성 시점과 유효시간 만기 시점 사이의 시간 길이

fresh(신선한)
응답의 경과 시간이 신선한 기간을 넘어서지 않았을 때 응답이 신선하다고 할 수 있다.

stale(낡은)
응답의 경과 시간이 신선한 기간을 넘어섰다면 응답이 낡았다고 할 수 있다.

semantically transparent(의미상으로 분명한)
성능을 향상시키고자 하는 목적을 제외하고 캐시의 사용이 요구하는 클라이언트나 원서버에 영향을 미치지 않을 때 특정한 요구에 대하여 캐시가 "의미상으로 분명하게" 작동한다고 할 수 있다. 캐시가 의미상으로 분명할 때 클라이언트는 원서버가 직접 처리했을 때와 완전히 동일할 응답을 수신하게 된다.(hop-by-hop 헤더는 제외).

validator(검증자)
캐시 엔트리가 엔터티의 복사본과 동일한지 알아내는 데 사용하는 규약 요소(예를 들면 엔터티 태그나 Last-Modified 시간)

1.4 Overall Operation

HTTP 규약은 요구/응답 규약이다. 클라이언트는 요구 method, URI, 규약 버전의 형태로 서버에 요구 메시지를 전송한다. 요구 변경자, 클라이언트 정보, 서버와의 접속에 사용되는 본문 내용을 포함하는 MIME 유형의 메시지가 뒤따른다. 서버는 메시지의 규약 버전 및 성공 또는 실패 코드를 포함하는 상태 정보로서 응답한다. 서버 정보, 엔터티 메타 정보, Entity-Body 내용을 포함하는 MIME 유형의 메시지도 뒤따른다.

[Page 11]

대부분의 통신은 사용자 에이전트가 구동하며 특정 원서버에 적용할 요구로 구성되어 있다. 가장 단순한 경우 이것은 사용자 에이전트(UA)와 원서버(O) 사이의 단일 접속(v)에 의해 성취할 수 있을 것이다.

```
request chain ---------------------->
UA ---------------- v ------------------- O
   <---------------------- response chain
```

좀 더 복잡한 상황은 Request/Response chain 에 하나 또는 그 이상의 중간 매개자가 있는 경우이다. 프락시, 게이트웨이 및 터널의 세 가지 일반적인 중간 매개 형태가 있다. 프락시는 전송 에이전트로 절대 표현 형태의 URI 요구를 수신하여 메시지의 전체 혹은 부분을 재작성한 후 URI 가 표시하는 서버로 재구성된 요구 메시지를 전달한다. 게이트웨이는 수신 에이전트로 다른 서버 위의 계층 역할을 수행하며 필요하다면 원서버의 규약에 맞도록 요구를 해석하기도 한다. 터널은 메시지를 변경하지 않고 두 연결 지점을 연결하는 중계역할을 수행한다. 터널은 통신(communication)이 중간 매개자가 메시지의 내용을 이해할 수 없을 때라도 방화벽과 같은 중간 매개자를 통과할 필요가 있을 때 사용한다.

```
request chain ------------------------------------>
UA -----v----- A -----v----- B -----v----- C -----v----- O
   <------------------------------------ response chain
```

위의 도표는 사용자 에이전트와 원서버 사이의 세 중간 매개자(A, B 및 C)를 보여 준다. 전체 고리를 통과하는 요구 또는 응답 메시지는 네 개의 별도 연결을 통과하게 된다. 몇몇 HTTP 통신 선택 사항은 최고 근접 거리의 비터널 이웃과의 통신, 연쇄적 연결 고리의 마지막 부분에만 또는 연결 고리에 따르는 모든 연결에 적용되기 때문에 이러한 구분은 중요하다. 그림이 선형이지만 각 참여자는 복수의 동시 통신에 참여할 수 있다. 예를 들어 B 는 A 의 요구를 처리함과 동시에 A 를 제외한 복수의 클라이언트 요구를 수신하고/수신하거나 C 이외의 서버에게 요구를 전송할 수 있다.

터널 역할을 수행하는 것이 아닌 통신에 참여하는 어떤 것이라도 요구를 처리할 때 내부 캐시를 사용할 수 있다. 캐시의 효과는 연결 고리를 따라 참가자 중 하나가 해당되는 요구에 적용할 수 있는 캐시된 응답을 갖고 있다면 Request/Response chain 이 짧아진다. 다음은 UA 또는 A 가 캐시하지 않은 요구에 대한 O (C 를 경유) 초기 응답의 사본을 B 가 가지고 있을 때의 결과 고리를 설명하고 있다.

[Page 12]

```
                request chain --------->
UA -----v----- A -----v----- B - - - - - - C - - - - - - O
                <-------- response chain
```

보통 모든 응답을 캐시할 수 있는 것은 아니며 어떤 요구는 캐시 방식에 특별 요구를 하는 변경자를 포함할 수 있다. 13 장에 캐시 방식과 캐시할 수 있는 응답에 대한 필요 조건이 기록되어 있다.

사실상 World Wide Web 에는 현재 실험되고 있거나 배포되고 있는 캐시와 프락시의 다양한 아키텍쳐와 환경설정 방법이 있다. 이러한 것 중에는 대륙간 대역폭을 절약하기 위한 프락시 캐시의 국가적 계층, 캐시 엔트리를 배포하거나 복수로 배포하는 시스템, CD-ROM 등을 통하여 캐시 된 데이터의 하부 세트를 배포하는 조직 등이 있다. HTTP 시스템은 광대역 연결을 통한 기업 인트라넷, 저동력 무선 연결의 PDA 를 통한 연결 및 간헐적인 연결에 사용된다. HTTP/1.1 의 목적은 고도의 신뢰성과 신뢰성을 확보할 수 없다면 신뢰할 수 있는 실패의 표시 기능을 지닌 웹 응용프로그램을 개발하는 개발자의 요구를 충족하는 규약 구조물을 새로 소개하면서도 이미 배포된 다양한 환경을 지원하는 것이다.

HTTP 통신은 대개 TCP/IP 연결을 통하여 이루어진다. 기본 포트는 TCP 80 이지만 다른 포트를 사용할 수도 있다. 그러나 이것은 HTTP 가 인터넷 상의 다른 규약이나 다른 네트워크 위에서 구현될 수 없게 하는 것은 아니다. HTTP 는 단순히 신뢰할 수 있는 전송 수단을 가정할 뿐이며 이러한 보장을 해 줄 수 있는 어떠한 규약을 사용해도 된다. HTTP/1.1 의 요구 응답 구조를 적용하고자 하는 규약의 전송 데이터 단위로 배치(mapping)하는 것은 이 규격의 범위 밖의 것이다.

HTTP/1.0 에서 대부분의 구현 방식은 각각의 요구/응답 교환에 새로운 접속을 사용하는 것이다. 또한 HTTP/1.1 에서는 하나의 접속을 하나 또는 그 이상의 요구/응답 교환에 사용할 수 있으나 연결이 여러 가지 이유로 단절될 수 있다.(8.1 절 참조)

2 기호 관례 및 일반적인 문법

2.1 추가된 BNF

이 문서에서 명시된 모든 메커니즘은 설명형 문구로서 RFC 822 [9]에서 사용한 것과 유사한 추가된 Backus-Naur Form (BNF)으로 설명되어 있다. 구현자는 이 규격을 이해하기 위하여 이러한 기호에 익숙할 필요가 있다. 추가된 BNF 는 다음의 구성 요소를 포함한다.

[Page 13]

name = definition

규칙의 이름이 이름 그 자체(둘러싸는 "<" 및 ">"이 없는)이며 정의 부분과는 등호 문자("=")로 구별된다. 계속되는 공백 문자는 규칙에 대한 규정이 한 줄 이상에 걸쳐 있음을 표시하는 들여쓰기의 경우에만 의미가 있다. SP, LWS, HT, CRLF, DIGIT, ALPHA 등과 같은 몇몇 기본 규칙은 대문자로만 사용한다. 정의문 내에서 소괄호는 규칙

이름의 사용 구별을 용이하게 해줄 경우에는 언제든지 사용한다.

"literal"
인용 부호로 문자 텍스트 주위를 감싼다. 별도의 언급이 없으면 문자는 대소문자를 구별한다.

rule1 | rule2
막대 ("|")로 구분된 요소는 선택 사항이다. 예를 들어 "yes |no" 는 yes 나 no어느 것이든 가능하다.

(rule1 rule2)
괄호로 둘러싼 요소는 단일 요소로 취급한다. 따라서 "(elem (foo | bar) elem)"는 "elem foo elem" 및 "elem bar elem"의 토큰 순서를 허용한다.

*rule
이것은 반복을 의미하는 것으로서 뒤이어서 나올 #rule과 혼동을 일으키는 표현 방식이므로 유의해야 한다. 반복을 통해 이루어지는 결과는 하나의 단어나 수와 같이 한 개 요소의 표현 형태로 되는 것이며, #rule에서는 똑같은 반복이지만 여러 개 단어나 수의 열 형태와 같이 여러 개 요소의 나열 형태로 표현되는 것이다. <n>*<m>element와 같은 표기 방법으로 쓰인다. 이것은 적어도 n개와 최대 m개의 요소로 구성되는 한 가지 결과를 의미한다. 즉, 1*2DIGIT 라는 표현은 숫자가 적어도 한 개 최대 두 개로 구성되어 한 개의 수를 나타낸다는 뜻이다. 4는 한 가지 예이며, 45도 한 가지 예가 된다. 그러나 345의 경우에는 숫자 세 개로 구성된 한 개 요소이므로 최대 갯수에 위배되어 적합하지 않다. n과 m은 생략될 수 있으며, 이 경우에 n의 기본값은 0이고 m의 기본값은 무한대이다. 그러므로 "*(element)"는 0개를 포함해서 어떤 개수라도 가능하고, "1*element"의 경우는 한 요소의 표현에 있어 적어도 한 개는 있어야 하며 최대 갯수에는 제한이 없다.

[rule]
대괄호는 선택 요소를 둘러 싼다. "[foo bar]" 는 "*1(foo bar)"와 동일하다.

N rule
특정 횟수의 반복을 나타낸다. "<n>(element)" 은 "<n>*<n>(element)"와 동일하다. 즉 요소(element)가 정확하게 <n> 번 표시된다. 따라서 2 DIGIT 는 2 자리 숫자, 3 ALPHA 는 세 개의 알파벳 문자로 구성된 문자열이다.

#rule
앞서 설명한 것처럼 반복을 나타내긴 하지만 요소들의 나열로서 표현되는 것이다. 즉, 1#DIGIT 라고 하면 여러 개의 수로 구성된 수열로서 표현되는데, 최소 한 개의 수는 있어야 하고 최대 갯수는 제한이 없는 수열이 된다. 각 요소들 사이의 구분은 ","와 LWS를 이용하는데, 여러 개의 나열 형태를 쉽게 표현할 수 있게 해준다.
예를 들어,
(*LWS element *(*LWS "," *LWS element))
이것을 간단하게 1#element 이와 같이 표현할 수 있다.

또 다른 예를 들자면,
1#2(2DIGIT)
이것은 숫자 두 개로 구성된 수가 적어도 한 개가 있어야 하며 최대 두 개까지 가능하다는 것이다.
즉, 23 이렇게 표현될 수도 있고, 23, 56 이렇게 두 개로 표현될 수도 있다.

이것이 *rule과의 차이점이고, #rule 에서도 "#element" 의 구성이 그대로 성립한다. 이에 대한 설명은 *rule 의 경우와 같다. ","를 이용하여 나열함에 있어, null element가 허용된다. 예를 들어, 1#3(2DIGIT)과 같은 표현식에 대해23, , 56, 34 이렇게 null element 표시가 가능하지만, 실제 개수는 세 개로서 간주된다. 따라서 최소 한 개 최대 세 개의 제한에 위배되지 않는다.

[Page 14]

; comment
규칙 문장에서 오른쪽으로 약간 떨어져 있는 세미콜론은 해당 라인의 끝에까지 계속되는 주석의 시작을 의미한다. 이것은 규격과 병행하여 적절한 설명을 포함시키기 위한 방법이다.

implied *LWS

두 개의 인접한 단어 (token or quoted-string) 또는 인접한 토큰(tokens)과 식별자 (tspecials) 사이에 LWS (linear whitespace)가 포함될 수 있다. 여기서 두 개의 토큰 사이에는 반드시 적어도 하나의 식별자가 존재하여 각기 하나의 토큰으로 간주되지 않게끔 구별되어야 한다.

2.2 기본 규칙

다음의 규칙은 기본적인 분석 구조를 설명하기 위해 이 규격 전반에 걸쳐 사용되고 있다. US-ASCII 로 코드화 된 문자 집합은 ANSI X3.4-1986 [21]에 의하여 규정되었다.

```
OCTET          = <모든 8-bit 연속 데이터>
CHAR           = <모든 US-ASCII 문자 (octets 0 - 127)>
UPALPHA        = <모든US-ASCII 대문자 "A".."Z">
LOALPHA        = <모든 US-ASCII 소문자 "a".."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <모든 US-ASCII 숫자 "0".."9">
CTL            = <모든 US-ASCII 제어 문자 (octets 0 - 31) 및 DEL (127)>
CR             = <US-ASCII CR, 캐리지 리턴(13)>
LF             = <US-ASCII LF, 라인피드 (10)>
SP             = <US-ASCII SP, 스페이스 (32)>
HT             = <US-ASCII HT, 수평 탭 (9)>
<">            = <US-ASCII 이중 인용 부호(34)>
```

[Page 15]

HTTP/1.1 은 연속적인 CR LF 를 Entity-Body 를 (부록 19.3 참조) 제외한 모든 규약 요소의 라인 마감 부호로 정의한다. Entity-Body 내에서의 라인 마감 부호는 3.7 절에서 설명된 것처럼 연관된 media type 에 의하여 정의한다.

```
CRLF           = CR LF
```

HTTP/1.1 헤더는 계속되는 라인이 스페이스나 수평 탭으로 시작한다면 복수의 라인에 걸쳐 계속 작성할 수 있다. 폴딩(folding)을 포함한 모든 선형 공백 스페이스는 SP 와 동일한 의미를 가진다.

```
LWS            = [CRLF] 1*( SP | HT )
```

TEXT 규칙은 메시지 분석기가 해석하지 않도록 정의한 설명 필드 내용이나 값에 사용한다. *TEXT 의 단어는 RFC 1522 [14]의 규칙에 따라 인코딩되었을 경우에만 ISO 8859-1 [22] 이외 문자세트의 문자를 포함할 수 있다.

```
TEXT           = < CTLs을 제외한 (그러나 LWS는 포함) 모든 OCTET>
```

16 진수 숫자는 몇몇 규약 요소에서 사용할 수 있다.

```
HEX            = "A" | "B" | "C" | "D" | "E" | "F"
                 | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
```

많은 HTTP/1.1 헤더 필드 값은 LWS 나 특수 문자로 구별되는 단어로 구성되어 있다. 파라미터 값 내에서 사용할 이러한 특별 문자는 반드시 인용 문자열 내에 있어야 한다.

```
token          = 1*<CTL 또는 tspecial를 제외한 모든 CHAR>
tspecials      = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | ""
                 | <"> | "/" | "[" | "]" | "?" | "=" | "{" | "}" | SP | HT
```

주석은 주석문을 괄호로 둘러싸서 몇몇 HTTP 헤더 필드에 포함할 수 있다. 주석은 "comment"를 필드 값 정의의 한 부분으로 포함하는 필드에서만 사용할 수 있다. 다른 필드에서 괄호는 필드 값의 일부로 간주된다.

```
comment        = "(" *( ctext | comment ) ")"
ctext          = < "(" and ")"을 제외한 모든 TEXT >
```

[Page 16]

텍스트 문자열이 이중 인용 부호를 사용하여 인용되었으면 단일 단어로 간주한다.

```
quoted-string  = ( <"> *(qdtext) <"> )
qdtext         = <<">을 제외한 모든 TEXT>
```

백슬래시 문자("")는 인용된 문자열이나 주석 내에서만 단일문자 인용 메커니즘으로서 사용할 수 있다.

```
quoted-pair    = "" CHAR
```

3 규약 파라미터

3.1 HTTP 버전

HTTP 는 "<주요한 변경>.<사소한 변경>" 번호 체계를 규약의 버전을 표시할 때 사용한다. 규약 버전 부여 정책은 발송자가 통신을 통하여 획득한 기능보다는 메시지의 형식 및 계속적인 HTTP 통신을 이해할 능력이 있음을 표시할 수 있도록 하기 위해 정의되었다. 단순히 확장할 수 있는 필드 값을 추가하거나 통신 방식에 영향을 미치지 않는 메시지 구성 요소를 추가했을 경우에는 버전 숫자에 변화가 없다. <사소한 변경> 숫자는 일반적인 메시지 분석 알고리즘에 대한 변화는 없지만 메시지 의미에 대한 추가 사항이나 발송자의 추가적인 능력을 의미하는 규약 추가 기능에 대한 변경이 있을 경우 증가된다. <주요한 변경> 숫자는 규약 내부의 메시지 형식이 변경되었을 때 증가한다.

HTTP 메시지의 버전은 메시지 첫 라인의 HTTP-Version 필드에 표시된다.

```
HTTP-Version  = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

주요 및 사소한 부분을 표시하는 숫자는 반드시 별도의 정수 값으로 구분되어야 하며 10 단위 이상으로 증가할 수 있음을 주의해야 한다. 따라서 HTTP/2.4 은 HTTP/2.13 보다 이전 버전이며 또한 HTTP/12.3 보다 이전 버전이다. 수신측에서는 앞 부분에 나오는 0 을 반드시 무시해야 하며 전송해서는 안 된다.

이 규격이 규정하는 대로 요구나 응답 메시지를 전송하는 애플리케이션은 반드시 HTTP-Version 을 "HTTP/1.1"로 설정해야 한다. 이 버전 번호를 사용하는 것은 발송하는 애플리케이션이 최소한 부분적으로는 이 규격을 따르고 있음을 표시한다.

애플리케이션의 HTTP 버전은 해당 프로그램이 최소한의 조건으로 상호 동작을 지원할 수 있는 최고 HTTP 버전 값이다.

[Page 17]

프락시 및 게이트웨이 프로그램의 규약 버전이 애플리케이션과 상이할 경우 메시지를 전달할 때 주의해야 한다. 규약 버전은 발송자의 규약 능력을 표시하기 때문에 프락시/게이트웨이는 실제 자신의 버전보다 높은 버전 표시를 사용하여 메시지를 발송해서는 절대로 안 된다. 상위 버전의 요구가 수신되었으면 프락시/게이트웨이는 반드시 요구 버전을 내리거나, 에러를 발송하거나 터널로 전환해야만 한다. 프락시/게이트웨이 버전보다 낮은 요구는 상위 버전으로 업그레이드 할 수는 있으나 요구 받은 버전의 주요 버전은 반드시 동일해야 한다.

주의: HTTP 버전 간의 변환은 관련된 버전이 요구하거나 금지한 헤더 필드의 변경을 수반할 수도 있다.

3.2 보편적 자원 식별자(Uniform Resource Identifier - URI)

URI는 WWW 주소, 보편적인 문서 식별자, 보편적 자원 식별자 또는 보편적 자원 위치 지정자(URL)와 이름(URN)의 결합에 이르기까지 많은 이름으로 불리우고 있다. HTTP로서는 보편적 자원 식별자란 이름, 위치 또는 다른 어떤 특징을 이용하여 자원을 식별해 주는 정형화 된 문자열일 뿐이다.

3.2.1 일반적 형식

HTTP 규약에서 URI는 사용되는 상황에 따라 절대적인 형태로 표현할 수도 있고 알려진 기본 URI의 상대적인 형태로 표현할 수도 있다. 이 두 형태는 절대적 URI는 항상 콜론이 뒤 따르는 scheme으로 시작한다는 사실로 구분할 수 있다.

```
URI           = ( absoluteURI | relativeURI ) [ "#" fragment ]
AbsoluteURI   = scheme ":" *( uchar | reserved )
RelativeURI   = net_path | abs_path | rel_path
net_path      = "//" net_loc [ abs_path ]
abs_path      = "/" rel_path
rel_path      = [ path ] [ ";" params ] [ "?" query ]
path          = fsegment *( "/" segment )
fsegment      = 1*pchar
segment       = *pchar
params        = param *( ";" param )
param         = *( pchar | "/" )
```

[Page 18]

```
scheme        = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc       = *( pchar | ";" | "?" )
query         = *( uchar | reserved )
fragment      = *( uchar | reserved )
pchar         = uchar | ":" | "@" | "&" | "=" | "+"
uchar         = unreserved | escape
unreserved    = ALPHA | DIGIT | safe | extra | national
escape        = "%" HEX HEX
reserved      = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra         = "!" | "*" | "'" | "(" | ")" | ","
```

```
safe            = "$" | "-" | "_" | "."
unsafe          = CTL | SP | <"> | "#" | "%" | "<" | ">"
national        = < ALPHA, DIGIT, reserved, extra, safe및unsafe을 제외한 모든 OCTET>
```

URL 형식과 의미 규정에 관한 정보는 RFC 1738 [4] 및 RFC 1808 [11]을 따르고 있다. 상기 BNF 는 RFC 1738 에 명시되어 있는 유효한 URL 의 형태에서 허용하지 않고 있는 국가 문자를 포함하고 있다. 이는 HTTP 서버가 주소에서 rel_path 부분을 표시하는 데 사용할 수 있는 예약되어 있지 않는 문자 집합에 제한을 받지 않고, HTTP 프락시가 RFC 1738 에 규정되지 않은 URI 요구를 수신할 수도 있기 때문이다. HTTP 규약은 URI 의 길이에 대한 어떠한 사전 제한도 두지 않는다. 서버는 반드시 자신이 제공하는 어떠한 자원의 URI 도 처리할 수 있어야 하며 이러한 URI 를 생성할 수 있는 GET 에 기초한 폼을 (GET-based forms) 제공한다면 무제한 길이의 URI 를 처리할 수 있어야만 한다. 서버는 URI 의 길이가 자신의 처리할 수 있는 (10.4.15 절 참조) 것보다 긴 경우 414 (Request-URI Too Long)를 응답으로서 돌려주어야 한다.

주의: 서버는 255 바이트 이상의 URI 길이를 사용할 때 몇몇 이전 클라이언트나 프락시 구현 방식이 이러한 길이를 적절히 지원할 수 없는 경우가 있기 때문에 주의해야 한다.

3.2.2 http URL

"http" scheme 은 HTTP 규약을 통하여 네트워크 자원의 위치를 파악하는 데 사용한다. 이 절은 http URL 에 사용되는 scheme 특유의 형식과 의미를 규정한다.

[Page 19]

```
http_URL        = "http:" "//" host [ ":" port ] [ abs_path ]
host            = <합법적인 인터넷 호스트 도매인 이름 또는 RFC 1123의 2.1 절에서 정의한 방식의
                   IP 주소(점으로 구분된 형식)>
port            = *DIGIT
```

포트 항목이 비어 있거나 명시되지 않았으면 포트는 80 으로 간주한다. TCP 연결 요구를 기다리고 있는 해당 호스트 서버의 해당 포트에 식별된 자원이 위치하고 있으며 자원의 Request-URI 는 abs_path 라는 것이 의미한다는 내용이다. URL 의 IP 주소의 사용은 가능한 한 피해야만 한다 (RFC 1900 [24] 참조). URL 에 abs_path 가 명시되어 있지 않으면 자원(5.1.2 절)을 위한 Request-URI 로서 사용할 때 반드시 "/"가 주어져야 한다.

3.2.3 URI 비교

URI 가 서로 일치하는지 여부를 결정하기 위해 URI 를 비교할 때 클라이언트는 전체 URI 에 대하여 대소문자를 구별하는 8 진수 대 8 진수 비교 방법(octet-by-octet comparison)을 사용해야만 하며 다음의 예외 사항이 있다.

- 비어 있거나 명시되지 않은 포트는 기본 포트 80번으로 정의한다;
- 호스트 이름의 비교에는 반드시 대소문자를 구별하지 않는다;
- scheme 이름의 비교는 반드시 대소문자를 구별하지 않는다;
- 비어 있는 abs_path는 "/"인 abs_path와 동일하다.

"예약되거나(reserved)" "안전하지 않는(unsafe)" 문자 집합 (3.2 절 참조) 이외의 문자는 ""%" HEX HEX" 인코딩과 동일하다.

예를 들어 다음의 세 URI는 동일하다.

http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html

[Page 20]

3.3 날짜/시간 형식

3.3.1 완전한 날짜

HTTP 프로그램은 역사적으로 세 가지 방법으로 시간/날짜를 표시해 왔다.

```
Sun, 06 Nov 1994 08:49:37 GMT          ; RFC 822, RFC 1123에서 갱신
Sunday, 06-Nov-94 08:49:37 GMT         ; RFC 850, RFC 1036에서 폐기
Sun Nov  6 08:49:37 1994               ; ANSI C의 asctime() 형식
```

첫번째의 형식이 인터넷 표준으로 우선권을 가지고 있으며 RFC 1123 (RFC 822의 개정판)에서 규정한 고정 길이의 하부 세트를 표시한다. 두 번째 형식은 일반적으로 사용되기는 하지만 폐기된 RFC 850 [12] 날짜 형식에 기초하고 있으며 4 단위 년도 표시가 결여되어 있다. 날짜를 분석하는 HTTP/1.1 클라이언트 및 서버는 반드시 상기 세 형식을 모두 수용해야 한다. 그러나 헤더 필드의 HTTP-날짜 값을 표시할 때는 반드시 RFC 1123 형식만을 생산해야 한다.

주의 : 날짜 값 수신처는 메시지를 프락시/게이트웨이를 통하여 SMTP나 NNTP로 조회 또는 발송하는 경우처럼 비 HTTP 애플리케이션이 발송한 날짜 값을 수신하는 데 적극적인 조치를 취할 것을 장려한다.

모든 HTTP 날짜/시간 표시는 예외 없이 반드시 그린이치 표준 시간(GMT))을 따라야 한다. 이는 처음 두 형식에서 시간대를 표시하는 3 문자의 축약어인 "GMT"를 포함함으로써 표시되어 있다. 또한 asctime 형식의 날짜를 읽을 때도 "GMT"라고 반드시 가정해야 한다.

```
HTTP-date     = rfc1123-date | rfc850-date | asctime-date
rfc1123-date  = wkday "," SP date1 SP time SP "GMT"
rfc850-date   = weekday "," SP date2 SP time SP "GMT"
asctime-date  = wkday SP date3 SP time SP 4DIGIT
date1         = 2DIGIT SP month SP 4DIGIT
                  ; day month year (e.g., 02 Jun 1982)
date2         = 2DIGIT "-" month "-" 2DIGIT
                  ; day-month-year (e.g., 02-Jun-82)
date3         = month SP ( 2DIGIT | ( SP 1DIGIT ))
                  ; month day (e.g., Jun  2)
time          = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                  ; 00:00:00 - 23:59:59
wkday         = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
```

[Page 21]

```
weekday       = "Monday" | "Tuesday" | "Wednesday" | "Thursday"
                | "Friday" | "Saturday" | "Sunday"
month         = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun"
                | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"
```

주의: 날짜/시간 표현에 대한 HTTP 필요 조건은 규약 스트림 내부에서 사용할 때만 적용된다. 클라이언트와 서버는 사용자의 표시 방법, 요구 로깅 등에서는 이러한 형식을 반드시 사용해야 할 필요는 없다.

3.3.2 Delta Seconds

몇몇 HTTP 헤더는 메시지가 수신된 이후의 시간을 10 진법의 정수로 초를 명시할 수 있도록 한다.

```
delta-seconds  = 1*DIGIT
```

3.4 문자 집합

HTTP 는 MIME 에서 설명된 "문자 집합"이라는 용어를 동일하게 사용한다. 일련의 8bit 데이터를 적절한 대응 관계에 있는 일련의 글자로 변환시킬 수 있도록 한 개 또는 그 이상의 표로서 만들어서 참조하게 하는 수단이다. 그러므로 무조건 변환시켜서는 안 되고, 모든 글자가 문자 집합에 정의되어 있지 않을 수 있고, 특정한 글자를 표현하기 위해 하나 이상의 8bit 데이터열이 존재할 수도 있다. 이 정의에 따르면, US-ASCII 와 같은 단순한 변환표로부터 ISO 2022 의 경우에서와 같이 복잡한 변환표에 이르기까지 다양한 종류의 문자 인코딩들을 허용한다. 하지만 MIME 문자 집합 이름과 관련된 정의는 8bit 데이터로부터 글자로의 변환에 관한 사항을 완전하게 명시하여야 한다. 완전한 변환 관계를 정의하기 위해 다른 수단을 통한 외부 정보를 활용해서는 안 된다.

주의 : 이러한 "문자 집합"이라는 용어의 사용은 보통 "문자 인코딩"으로 지칭된다. 그러나 HTTP와 MIME은 동일한 등록표를 사용하기 때문에 용어를 공유하는 것 또한 중요하다.

[Page 22]

HTTP 문자 집합은 토큰에 의해 식별되며 대소문자를 구별하지 않는다. 완전한 토큰 세트는 IATA 문자 집합 등록표(IANA Character Set registry [19])에 규정되어 있다.

```
charset        = token
```

HTTP 가 charset 값으로 임의의 토큰을 사용하도록 허용하지만 IATA 문자 집합 등록에 사전 정의된 모든 토큰은 반드시 이 등록표에 등록된 문자 집합을 표시해야 한다. 애플리케이션은 사용하는 문자 집합을 IATA 등록 표에서 규정된 것으로 제한해야만 한다.

3.5 내용 코딩(Content Codings)

내용 코딩 값은 엔터티에 적용하였거나 적용할 수 있는 인코딩 변환을 표시한다. 내용 코딩은 문서를 압축하거나, 그렇지 않다면 내용의 media type 의 정체를 상실하거나 정보를 손실하지 않고 유용하게 변형하는 데 사용한다. 종종 엔터티는 코드화 된 폼에 저장되고 직접 전송되어 수신측만이 이를 해독한다.

```
content-coding   = token
```

모든 내용 코딩의 값은 대소문자를 구별하지 않는다. HTTP/1.1 은 Accept-Encoding (14.3 절) 및 Content-Encoding (14.12 절) 헤더 파일에 내용 코딩 값을 사용한다. 그 값이 Content-Coding 을 설명하는 것이지만 더욱 중요한 것은 인코딩을 제거하기 위해 필요한 해독 메커니즘을 표시한다는 것이다.

인터넷에서 할당된 숫자 체계(Internet Assigned Numbers Authority (IANA))는 Content-Coding 값 토큰의 등록표 역할을 수행한다. 처음에 이 등록표에는 다음의 토큰이 포함되어 있다.

gzip

RFC 1952 [25]에 설명된 대로 파일 압축 프로그램인 "gzip"에 의하여 생성된 인코딩 포맷. 이 포맷은 32 bit CRC를 가진 Lempel-Ziv coding (LZ77)이다.

compress
일반적인 UNIX 파일 압축 프로그램인 "compress"에 의하여 생성된 인코딩 포맷. 이 포맷은 Lempel-Ziv-Welch 코딩(LZW)을 수정한 것이다.

[Page 23]

주의: 인코딩 포맷을 식별하는 프로그램 이름의 사용은 바람직하지 않으며 향후 인코딩을 위해서 사용하지 말도록 권고한다. 프로그램 이름을 여기에서 사용한 것은 역사적인 관례이며 훌륭한 디자인은 아니다. HTTP의 이전 구현법과 호환성을 유지하기 위해 애플리케이션은 "x-gzip" 및 "x-compress" 을 "gzip" 과 "compress" 각각 동일한 것으로 간주해야 한다.

deflate
RFC 1951[29]에 설명된 "deflate" 압축 메커니즘과 결합하여 RFC 1950[31]에 정의된 "zlib" 포맷.

새로운 Content-Coding 값 토큰은 등록해야 한다. 클라이언트와 서버가 상호 운용성을 가지도록 하기 위해 새로운 값을 구현하는 데 필요한 내용 코딩 알고리즘에 대한 규격은 일반인이 사용할 수 있어야 하고 독립적으로 구현하기에 적합해야 하며 이 절에 규정된 내용 코딩의 목적에 따라야 한다.

3.6 전송 코딩 (Transfer Codings)

전송 코딩 값은 네트워크를 통한 "안전 전송"을 확보하기 위해 Entity-Body 에 적용하였거나, 적용할 수 있거나 또는 적용할 필요가 있는 인코딩 변환을 표시하는 데 사용한다. 전송 코딩은 메시지의 특성 중의 하나이며 원래 엔터티의 특성이 아니라는 점이 내용 코딩과 다른 점이다.

```
transfer-coding         = "chunked" | transfer-extension
transfer-extension      = token
```

모든 transfer-coding 값은 대소문자를 구별하지 않는다. HTTP/1.1 은 Transfer-Encoding 헤더 필드(14.40 절)의 전송 코딩 값을 사용한다.

전송 코딩은 7 비트 전송 서비스로 바이너리 데이터를 안전하게 전송할 수 있도록 디자인 된 MIME 의 Content-Transfer-Encoding 값과 유사하다. 그러나 8 비트 전송 규약에서 안전 전송의 중점은 다른 곳에 있다. HTTP 에서 유일한 Message-Body 의 불안전한 특징은 정확한 본문 길이(7.2.2 절)를 결정하기 어렵다는 것과 공유하는 전송체계에서 데이터를 암호화하기 어렵다는 것이다.

덩어리 인코딩(chunked encoding)은 메시지를 일련의 덩어리로 전송하기 위하여 메시지 본문을 변경한다. 이 덩어리는 각각 자신의 크기 표시자를 가지고 있으며 Entity-Header 필드를 포함하고 있는 선택적인 각주(footer)가 뒤따른다. 이를 통하여 역동적으로 생산된 내용물이 수신인이 메시지 전체를 수신하였다는 것은 증명하는 데 필요한 정보와 함께 전송될 수 있도록 한다.

[Page 24]

```
Chunked-Body            = *chunk
                          "0" CRLF
                          footer
                          CRLF
chunk                   = chunk-size [ chunk-ext ] CRLF
                          chunk-data CRLF
```

```
hex-no-zero             = < "0"을 제외한 HEX >
chunk-size              = hex-no-zero *HEX
chunk-ext               = *( ";" chunk-ext-name [ "=" chunk-ext-value ] )
chunk-ext-name          = token
chunk-ext-val           = token | quoted-string
chunk-data              = chunk-size(OCTET)
footer                  = *Entity-Header
```

덩어리 인코딩은 크기 0 의 덩어리로 종결되며 빈 라인으로 종료되는 각주가 뒤따른다. 각주의 목적은 역동적으로 생성된 엔터티에 대한 정보를 효과적으로 제공하도록 하는 것이다. 애플리케이션은 Content-MD5, 디지털 서명이나 다른 기능을 위한 HTTP 향후 확장으로 명백하게 규정되지 않은 헤더 필드를 결코 각주에 넣어서 전송해서는 안 된다.
Chunked-Body 를 해독하는 예가 부록 19.4.6 에 제시되어 있다.

모든 HTTP/1.1 애플리케이션은 반드시 덩어리 전송 코딩을 해독하고 수신할 수 있어야 하며 해독할 수 없는 전송 코딩 확장은 반드시 무시해야 한다. 해독할 수 없는 Transfer-Coding 과 함께 Entity-Body 를 수신하는 서버는 501 (Unimplemented)을 응답으로 돌려주고 연결을 종료해야 한다. 서버는 결코 Transfer-Coding 을 HTTP/1.0 클라이언트에 보내서는 안 된다.

3.7 미디어 형식(Media type)

HTTP 는 공개적이고 확장 가능한 데이터 유형 설정 및 유형 협상 기능을 제공하기 위해 Content-Type (14.18 절) 및 Accept (14.1 절) 헤더 필드의 인터넷 미디어 형식을 사용한다.

```
media-type      = type "/" subtype *( ";" parameter )
type            = token
subtype         = token
```

attribute/value(속성/값) 쌍 형태의 파라미터가 type/subtype 유형을 뒤따른다.


```
parameter       = attribute "=" value
attribute       = token
value           = token | quoted-string
```

유형, 하부 유형 및 파라미터 속성 이름은 모두 대소문자를 구분하지 않는다. 파라미터 값은 파라미터 이름의 의미에 따라 대소문자를 구별할 수도 있고 않을 수도 있다. 선형 공백 스페이스(LWS)는 유형과 하부 유형, 속성과 속성 값 사이에 절대 사용해서는 안 된다. 미디어 형식을 인지하는 사용자 에이전트는 반드시 해당 MIME 유형의 파라미터를 해당 type/subtype 정의가 설정한 방식으로 처리해야 한다. (또는 사용자 에이전트가 해당 type/subtype 을 처리하는 데 사용하는 외부 애플리케이션이 처리하도록 주선해야 한다.) 또한 발견된 모든 문제를 사용자에게 알려 주어야 한다.

주의 : 이전 HTTP 애플리케이션은 미디어 형식 파라미터를 인지하지 못한다. 이전 HTTP 애플리케이션으로 데이터를 전송할 때 구현 방식은 해당 type/subtype 정의가 요구할 때만 미디어 형식 파라미터를 사용해야 한다.

미디어 유형 값은 인터넷 할당 숫자 체계(IANA)에 등록된다. 미디어 유형을 등록하는 절차는 RFC 2048 [17]에 윤곽이 설명되어 있다. 등록되지 않은 미디어 유형을 사용하는 것은 권하지 않는다.

3.7.1 정형화(Canonicalization) 및 텍스트 기본값

인터넷에서의 미디어 형식은 정형화된 형식으로서 등록되어 있다. 보통 HTTP 메시지를 통하여 전송되는 Entity-Body 는 반드시 전송되기 이전에 적절한 정형화된 형식으로 표시되어야 한다. 이의 예외는 다음 문구에서 정의된 "text" 유형이다.

정형화된 형식으로서 text 형식의 미디어 subtype 은 CRLF 를 텍스트 라인 줄 바꿈으로 사용한다. HTTP 는 이러한 필요 조건을 완화하여 Entity-Body 전반에 걸쳐 일관성 있게 동일한 방법을 사용하였을 경우 단순히 CR 또는 LF 하나를 줄 바꿈으로 표현하는 텍스트 미디어 전송을 허용한다. HTTP 애플리케이션은 CRLF, 단편적인 CR 또는 LF 를 HTTP 를 통하여 수신한 텍스트 미디어에서 줄 바꿈을 표시하는 것으로 인정해야만 한다. 또한 텍스트가 몇몇 멀티바이트 문자 집합의 경우처럼 8 진수 13 과 10 을 CR 과 LF 로 사용하지 않는 문자 집합을 사용하고 있을 경우 HTTP 는 어떠한 일련의 octets 가 해당 문자 집합에서 줄바꿈을 위한 CR 및 LF 를 대표하는 것으로 규정하든 이를 허용한다. 이러한 줄바꿈에 대한 유연성은 Entity-Body 의 텍스트 미디어에만 적용되며 단편적인 CR 또는 LF 는 어떠한 HTTP 제어 구조(헤더 필드 및 multipart 경계와 같은)에서도 CRLF 를 대체해서는 안 된다.

Entity-Body 가 Content-Encoding 으로 인코딩 되었다면 내부 데이터는 인코딩 되기 이전에 위에서 규정한 형식으로 표현되어 있어야 한다.

[Page 26]

"charset" 파라미터는 몇몇 미디어 형식에서 데이터의 문자 집합(3.4 절)을 규정하는 데 사용한다. 송신자가 명백한 charset 파라미터를 제공하지 않았다면 "text" 유형의 미디어 subtype 형식은 HTTP 를 통하여 수신했을 때 "ISO-8859-1"의 charset 기본값을 갖도록 규정되어 있다. "ISO-8859-1" 이외 문자 집합의 데이터나 그 하부 세트는 적절한 charset 값으로 명명되어야 한다.

몇몇 HTTP/1.0 소프트웨어는 charset 파라미터 없는 Content-Type 헤더를 "수신측이 짐작해야 한다"라고 잘못 해석하였다. 이러한 방식을 방지하고자 하는 송신자는 charset 가 ISO-8859-1 일 때도 charset 파라미터를 포함할 수 있다. 또한 수신측에게 혼선을 주지 않는다는 것을 알 수 있을 때도 그렇게 해야 한다.

불행하게도 몇몇 이전 HTTP/1.0 클라이언트는 명확한 charset 파라미터를 적절히 처리하지 못했다. HTTP/1.1 수신측은 송신측이 제공하는 charset 라벨을 반드시 감안해야 한다. 또한 charset 을 추측하는 조항을 가진 사용자 에이전트는 Content-Type 필드의 charset 를 지원한다면 처음 문서의 내용을 표시할 때 수신측의 선호도에 따르기보다는 반드시 이 charset 를 사용해야 한다.

3.7.2 Multipart Type

MIME 은 많은 "multipart" 형식을 제공하고 있는데 하나 또는 그 이상의 엔터티를 단일 메시지 본문 내에 포함시킬 수 있도록 하는 것이다. 모든 multipart 형식은 MIME [7]에 규정되어 있는 공통적 표기법에 따르며 미디어 형식 표시값의 일부로서 경계 파라미터를 포함해야 한다. 메시지 본문 자체는 규약의 한 요소이며, Body-Part 간의 줄 바꿈을 표시할 때 반드시 CRLF 만을 사용해야 한다. MIME 과는 달리 모든 multipart 메시지의 맺음말은 반드시 비어 있어야 한다. HTTP 애플리케이션은 맺음말을 절대로 전송해서는 안 된다 (비록 원래의 multipart 가 맺음말을 포함하고 있다 하여도).

HTTP 에서 multipart 의 Body-Part 는 해당 부분의 의미에 중대한 영향을 끼치는 헤더필드를 포함할 수 있다. Content-Location 헤더 필드(14.15 절)는 URL 로 확인할 수 있는 엔터티의 Body-Part 에 포함되어야 한다. 일반적으로 HTTP 사용자 에이전트는 MIME 사용자 에이전트가 multipart 유형을 수신했을 때 처리하는 방식과 동일하거나 유사한 방식에 따라야 한다.

주의: RFC 1867 [15]에서 설명된 것처럼 "multipart/form-data" 유형이 POST 요구 method으로 처리하기에 적합한 폼 데이터를 전송하기 위해 특별히 규정되었다.

[Page 27]

3.8 제품 토큰

제품 토큰은 통신 애플리케이션이 자신의 소프트웨어 이름 및 버전을 확인하는 데 사용한다. 제품 토큰을 사용하는 대부분의 필드는 공백 문자로 구분되는 목록에 열거할 애플리케이션의 중요한 부분을 형성하는 부수적 제품명의 표시를 허용한다. 관례상 제품은 애플리케이션을 식별해 주는 제품의 중요도에 따라 열거된다.

```
product             = token ["/" product-version]
product-version     = token
```

예:

```
User-Agent   : CERN-LineMode/2.15 libwww/2.17b3
Server       : Apache/0.8.4
```

제품 토큰은 간략해야 하며 요점이 있어야 한다. 광고나 필수적이지 않은 다른 정보를 위해 사용하는 것은 분명하게 금지되어 있다. 어떠한 토큰 문자라도 제품 정보에 표시할 수 있지만 이 토큰은 버전 식별자에만 사용해야 한다.(동일한 제품의 계속적인 버전은 제품 값(value)의 product-version 부분만 달라야 한다.)

3.9 품질 등급 값

HTTP 내용 협상(12 장)은 짤막한 "부동소수점" 숫자를 사용하여 다양한 협상 가능 파라미터의 상대적 중요성("가중치")을 표시한다. 가중치는 최소값 0 부터 최대값 1 까지 범위의 실수로 정형화할 수 있다. HTTP/1.1 애플리케이션은 부동소수점 이후의 자리에 세 자리 이상의 숫자를 절대 사용해서는 안 된다. 이러한 값들에 대한 사용자의 값 설정은 또한 다음의 방식으로 한정되어야 한다.

```
qvalue         = ( "0" [ "." 0*3DIGIT ] )
                 | ( "1" [ "." 0*3("0") ] )
```

"품질등급 값"은 이 값이 단순히 원하는 품질에 대한 상대적인 질 저하를 표현하는 것이기 때문에 잘못된 명칭이다.

3.10 언어 태그

언어 태그는 인간이 다른 인간과 정보를 교환하기 위하여 말하거나, 쓰거나 혹은 전달하는 자연적인 언어를 표시한다. 컴퓨터 언어는 분명히 제외된다. HTTP 는 Accept-Language 및 Content-Language 필드를 이용하여 언어 태그를 표시한다.

[Page 28]

HTTP 언어 태그의 의미 및 등록표는 RFC 1766 [1]에서 규정한 것과 동일한 것을 사용한다. 요약하면 언어 태그는 하나 또는 그 이상의 부분으로 구성되어 있다. 주요 언어 태그 및 비어 있을 수도 있는 일련의 하부 태그는 다음과 같다.

```
language-tag   = primary-tag *( "-" subtag )
primary-tag    = 1*8ALPHA
subtag         = 1*8ALPHA
```

태그 내에서는 공백 문자를 사용할 수 없고 모든 태그는 대소문자를 구별하지 않는다. 언어 태그의 이름 영역은 IANA 에서 관리한다.

```
태그 예제는:
en, en-US, en-cockney, i-cherokee, x-pig-latin
```

여기서 두 자리 문자로 되어 있는 제일 앞 태그는 ISO 639 언어 축약어 형태이며, 두 자리 문자로 되어 있는 첫 하부 태그는 ISO 3166 국가 코드이다.(위에서 마지막 세 가지 태그는 등록되지 않은 태그이다. 마지막 태그만 향후 등록될 수 있다.)

3.11 엔터티 태그

엔터티 태그는 동일하게 요구된 자원에서 둘 또는 그 이상의 엔터티를 비교하는 데 사용한다. HTTP/1.1 은 엔터티 태그를 ETag (14.20 절), If-Match (14.25 절), If-None-Match (14.26 절) 및 If-Range (14.27 절) 헤더 필드에서 사용한다. 사용 방법 및 캐시 검증자와의 비교에 관한 정의는 13.3.3 절에 있다. 엔터티 태그는 명쾌하지 않은 인용 문자열로 (an opaque quoted string) 구성되어 있으며 약함(weakness) 표시자가 접두사로 붙을 수 있다.

```
entity-tag     = [ weak ] opaque-tag
weak           = "W/"
opaque-tag     = quoted-string
```

"강한 엔터티 태그(strong entity tag)"는 8 진수의 질이 (octet equality) 동일할 경우에만 두 엔터티가 자원을 공유할 수 있다.

"W/" 접두사로 표시되는 "약한 엔터티 태그(weak entity tag)"는 엔터티가 동일하고 의미상 심각한 변화 없이도 서로 대체할 수 있을 경우에만 두 엔터티가 자원을 공유할 수 있다. 약한 엔터티 태그는 약한 비교에만 사용할 수 있다.

엔터티 태그는 반드시 특정 자원과 연관된 모든 엔터티의 모든 버전을 통틀어 유일해야 한다. 특정 엔터티 태그값을 상이한 URI 에 대한 요구로부터 획득한 엔터티에 동일하다는 아무런 표시 없이 사용할 수 있다.

[Page 29]

3.12 영역 단위

HTTP/1.1 는 클라이언트가 엔터티의 일부분(특정 영역)만 응답으로 전송해 달라고 요구할 수 있다. HTTP/1.1 은 Range (14.36 절) 및 Content-Range (14.17 절) 헤더 필드의 영역 단위를 사용한다. 엔터티는 여러 가지 구조적 단위 크기에 따라 여러 하부 영역으로 분리할 수 있다.

```
range-unit              = bytes-unit | other-range-unit
bytes-unit              = "bytes"
other-range-unit        = token
```

HTTP/1.1 에서 정의한 유일한 영역 단위는 "바이트(bytes)"이다. HTTP/1.1 의 구현 방식은 다른 단위를 사용하여 명시한 영역을 무시할 수 있다. HTTP/1.1 은 영역에 대한 인식 여부에 상관없이 애플리케이션의 구현을 허용하고 있다.

4 HTTP 메시지

4.1 메시지 유형

HTTP 메시지는 클라이언트로부터 서버로의 요구 및 서버로부터 클라이언트로의 응답으로 구성되어 있다.

```
HTTP-message            = Request | Response      ; HTTP/1.1 messages
```

요구(5 장) 및 응답(6 장) 메시지는 엔터티를 전송(message payload)하기 위해 RFC 822 [9]의 일반적 메시지 형식을 사용한다. 두 메시지 형식 모두는 시작 라인, 하나 또는 그 이상의 헤더 필드("헤더"라고도 알려졌다.), 헤더 필드의 끝을 표시하는 빈 라인(예를 들어 CRLF 이전에 아무 것도 없는 라인) 및 선택 사항인 Message-Body 으로 구성되어 있다.

```
generic-message         = start-line
                          *message-header
                          CRLF
                          [ message-body ]
start-line              = Request-Line | Status-Line
```

안정적인 동작(robustness)을 위해 서버는 Request-Line 이 있어야 할 곳에 빈 라인을 수신하면 이를 무시해야 한다. 다른 말로 표현 한다면, 만약 서버가 규약 스트림을 읽는 도중 메시지 처음에 CRLF 를 수신하게 되면 이를 무시해야 한다.

[Page 30]

주의: HTTP/1.0 클라이언트로서 잘못 구현한 방법은 POST 요구 후 추가적인 CRLF를 생성한다는 것이다. BNF가 분명하게 금지하고 있는 것을 다시 언급한다면 HTTP/1.1 클라이언트는 여분의 CRLF로 요구를 시작하거나 따라서는 안 된다.

4.2 메시지 헤더

Request-Header(5.3 절), Response-Header(6.2 절) 및 Entity-Header(7.1 절) 필드를 포함하는 HTTP 헤더 필드는 RFC 822 [9] 3.1 절에서 규정한 일반적 형식을 동일하게 따르고 있다. 각각의 헤더 필드는 콜론 (":") 및 필드값이 뒤 따르는 이름으로 구성되어 있다. 필드 이름은 대소문자를 구분하지 않는다. 필드 값은 단일 SP 가 우선적이지만 무한정한 LWS 가 선행될 수 있다. 헤더 필드는 최소한 하나의 SP 및 HT 가 선행되는 여분의 라인이 선행되는 복수의 행에 걸쳐 확장될 수 있다. 애플리케이션은 공통적인 구성 형태를 벗어난 것을 처리하지 못 하는 구현 결과도 있을 수 있기 때문에 HTTP 구조를 생성할 때 "일반적인 형식"을 따라야 한다.

```
message-header      = field-name ":" [ field-value ] CRLF
field-name          = token
field-value         = *( field-content | LWS )
field-content       = <필드 값을 구성하는 OCTET 이며 *TEXT 또는 토큰,
                       tspecials,인용 스트링의 결합으로 구성된다.>
```

다른 필드 이름으로 수신된 헤더 필드의 정렬 순서는 중요하지 않다. 그러나 General-Header 필드를 맨 처음 나오고 Request-Header 이나 Response-Header 필드가 뒤를 따르고 마지막에 Entity-Header 필드가 나오는 것이 "바람직한 관행" 이다. 동일한 필드 이름을 가진 복수의 Message-Header 필드가 해당되는 헤더 필드의 전체 field-value 가 콤마로 구분된 목록[예를 들면 #(values)]으로 규정되어 있을 경우에만 메시지에 존재할 수 있다. 복수의 헤더 필드를 뒤 따르는 field-value 를 처음에 추가하여(각각의 값은 콤마로 구분된다) 메시지의 의미를 변화 시키지 않고 단일 "field-name: field-value" 쌍으로 결합할 수 있어서는 안 된다. 따라서 동일한 field-name 의 헤더 필드를 수신하는 순서는 결합된 필드 값을 해석하는 데 중요하게 된다. 그러므로 프락시는 메시지를 전송할 때 이러한 필드 값의 순서를 절대 변경해서는 안 된다.

[Page 31]

4.3 메시지 본문

HTTP 메시지의 Message-Body 는 (만약 있다면) 요구 또는 응답과 관련된 Entity-Body 를 전송하는 데 사용한다. Message-Body 는 Transfer-Encoding 헤더 필드(14.40 절)에 설명된 전송 코딩이 적용되었을 때에만 Entity-Body 와 다르다.

```
message-body        = Entity-Body
                    | <Transfer-Encoding에 따라 인코딩된 Entity-Body>
```

안전하고도 적절한 메시지 전송을 가능하게 하기 위해 애플리케이션이 적용한 전송 코딩 방식을 표시하기 위해 반드시 Transfer-Encoding 을 사용해야 한다. Transfer-Encoding 은 메시지의 특성이지 엔터티의 특성이 아니기 때문에 요구/응답에 따라서 애플리케이션이 추가 또는 삭제할 수 있다.

Message-Body 를 언제 메시지에서 사용할 수 있는가에 대한 규칙은 요구와 응답에 대해 각각 다르다. 요구 메시지에 있어 Message-Body 의 존재는 요구 Message-Header 에 Content-Length 또는 Transfer-Encoding 헤더 필드를 포함함으로써 표시할 수 있다. Message-Body 는 요구 method(5.1.1 절)가 Entity-Body 를 허용할 때만 요구 메시지에 포함할 수도 있다.

응답 메시지의 경우 Message-Body 가 메시지에 포함되어 있는가의 여부는 요구 method 및 응답상태 코드(6.1.1 절) 모두에 달려 있다. HEAD 요구 method 에 대한 모든 응답은 Entity-Header 필드가 포함한 것처럼 믿게 하여도 Message-Body 를 포함해서는 절대 안 된다. 모든 1xx (Informational), 204 (No Content) 및 304 (Not Modified) 응답은 Message-Body 를 절대 포함해서는 안 된다. 다른 모든 응답은 비록 길이가 제로라 할지라도 Message-Body 를 포함한다.

4.4 메시지 길이

Message-Body 가 메시지에 포함되어 있을 때 그 본문의 길이는 다음 중의 하나에 의해 결정된다.(우선 순위에 따라)

1. Message-Body를 절대 포함해서는 안 되는 모든 응답 메시지는 (1xx, 204 및 304 응답 메시지와 HEAD

요구에 대한 모든 응답) Entity-Header 필드의 존재 유무에 관계없이 헤더 필드 다음의 첫 빈 라인으로 항상 종료된다.

2. Transfer-Encoding 헤더 필드가 (14.40 절) 존재하고 "chunked" 전송 코딩이 적용되었음을 표시하고 있으면 길이는 chunked 인코딩에 (3.6 절) 의해 규정된다.

[Page 32]

3. Content-Length 헤더 필드가 (14.14 절) 존재하고 그 바이트 단위의 값이 Message-Body의 길이를 표시한다.

4. 메시지가 자신 스스로의 경계 설정 요소로서 미디어 형식 "multipart/byteranges"를 사용하고 있다면 바로 이것이 길이를 규정한다. 이 미디어 형식은 송신자가 수신측이 그것을 분석할 수 있다는 것을 알 수 없을 때에는 절대 사용해서는 안 된다. 어떤 요구 메시지가 복수의 Byte-Range 명시자를 가진 Range 헤더를 갖고 있으면 클라이언트가 multipart/byteranges 응답을 분석할 수 있음을 의미한다.

5. 연결을 단절하는 서버에 의하여. (연결 단절은 서버가 응답을 되돌려 줄 가능성을 전혀 남겨 두지 않기 때문에 응답 본문의 종료를 표시하는 데 사용해서는 안 된다.)HTTP/1.0 애플리케이션과의 호환성 유지를 위해 Message-Body를 가지고 있는 HTTP/1.1 요구는 서버가 HTTP/1.1을 따른다는 것을 알기 전에는 반드시 유효한 Content-Length 헤더 필드를 포함해야 한다. 요구가 Message-Body를 포함하고 있고 Content-Length가 주어지지 않았으면 서버는 메시지의 길이를 결정할 수 없을 때는 400(Bad Request)을 응답으로 보내고, 계속하여 유효한 Content-Length 수신을 기다리고자 할 때는 411(Length Required) 메시지를 반송하여야 한다.

엔터티를 수신하는 모든 HTTP/1.1 애플리케이션은 반드시 "chunked" 전송 코딩(3.6 절)을 허용해야만 한다. 이렇게 함으로서 메시지의 길이를 미리 결정할 수 없을 때 이 메커니즘이 사용될 수 있도록 한다.

메시지는 Content-Length 헤더 필드 및 "chunked" 전송 코딩을 모두 포함해서는 안 된다. 만약 둘 다를 수신하였으면 Content-Length 는 반드시 무시해야 한다.

Message-Body 가 허용된 메시지에 Content-Length 가 주어졌을 때 그 필드 값은 반드시 Message-Body 의 OCTET 숫자와 정확하게 일치해야 한다. HTTP/1.1 사용자 에이전트는 유효하지 않은 길이를 수신했거나 탐지했을 때 반드시 사용자에게 이를 알려야 한다.

[Page 33]

4.5 일반 헤더 필드

요구와 응답 메시지 모두에 일반적으로 적용할 수 있지만 전송되는 엔터티에는 적용되지 않는 헤더 필드가 몇 가지 있다. 이러한 헤더 필드는 전송되는 메시지에만 적용된다.

```
general-header = Cache-Control          ; 14.9 절
               | Connection             ; 14.10 절
               | Date                   ; 14.19 절
               | Pragma                 ; 14.32 절
               | Transfer-Encoding      ; 14.40 절
               | Upgrade                ; 14.41 절
               | Via                    ; 14.44 절
```

General-Header 필드 이름을 추가하고자 한다면 HTTP 규약 버전이 변경되어야 한다. 그러나 통신에 참여하는 모든 대상이 새로운 또는 실험적인 헤더 필드를 General-Header 필드로 인지한다면 이들 헤더 필드를 일반 헤더의 의미로 적용할 수 있다. 인식되지 않은 헤더 필드는 Entity-Header 필드로 처리된다.

5 요구(Request)

클라이언트로부터 서버로의 요구 메시지는 해당 메시지의 첫 라인 내에 자원, 자원의 식별자 및 사용 중인 규약 버전에 적용할 method 를 포함한다.

```
Request       = Request-Line            ; 5.1 절
                *( general-header       ; 4.5 절
                | request-header        ; 5.3 절
                | Entity-Header )       ; 7.1 절
                CRLF
                [ message-body ]        ; 7.2 절
```

5.1 Request-Line

Request-Line 은 method 토큰으로 시작하며 Request-URI 및 규약 버전이 뒤 따르며 CRLF 로 종결된다. 각 요소는 SP 문자로 구분된다. CR 또는 LF 는 마지막 CRLF 순서 이외에는 허용되지 않는다.

```
Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

[Page 34]

5.1.1 Method

Method 토큰은 Request-URI 로 식별되는 자원에서 수행할 method 를 표시한다. method 는 대소문자를 구별한다.

```
Method                  = "OPTIONS"               ; 9.2 절
                        | "GET"                   ; 9.3 절
                        | "HEAD"                  ; 9.4 절
                        | "POST"                  ; 9.5 절
                        | "PUT"                   ; 9.6 절
                        | "DELETE"                ; 9.7 절
                        | "TRACE"                 ; 9.8 절
                        | extension-method
extension-method        = token
```

자원이 허용하는 method 의 목록은 Allow 헤더 필드(14.7 절)에 명시할 수 있다. 응답의 리턴 코드는 허용된 method 세트가 역동적으로 변할 수 있기 때문에 항상 method 가 현재 자원에서 허용되는지 여부를 클라이언트에게 알려 준다. 서버는 서버가 method 를 알고는 있으나 요구된 자원에서는 사용할 수 없을 때 상태 코드 405(Method Not Allowed)를, 서버가 method 를 인지하지 못하거나 구현하지 않았을 때는 상태 코드 501(Not Implemented)을 리턴해야만 한다. 서버가 알고 있는 method 의 목록은 Public Response-Header 필드(14.35 절)에 나열할 수 있다.

GET 및 HEAD method 는 모든 일반적인 목적의 서버가 반드시 지원해야 한다. 다른 모든 method 는 선택적이다. 하지만 GET 및 HEAD method 가 구현되었으면 반드시 9 장에 명시된 의미와 동일하게 구현되어야 한다.

5.1.2 Request-URI(Request-URI)

Request-URI 는 보편적인 자원 식별자(3.2 절)이며 요구를 적용할 자원을 식별한다.

```
Request-URI    = "*" | absoluteURI | abs_path
```

Request-URI 의 세 가지 선택 사항은 요구의 성격에 달려 있다. 별표 "*"는 요구를 특별한 자원에 적용하지 않고 서버 자체에 적용한다는 것을 의미하며 사용된 method 가 반드시 자원에 적용되는 것은 아닐 때 사용할 수 있다.

한 예를 보면;

```
OPTIONS * HTTP/1.1
```

[Page 35]

프락시에게 요구를 만들 때는 absoluteURI 형식이 필요하다. 프락시는 유효한 캐시로부터 요구를 전송하거나 처리하여 응답을 되돌려 주어야 한다. 프락시는 absoluteURI 에 명시된 대로 요구를 다른 프락시로 전송하거나 서버로 직접 전송할 수 있다는 점을 주목해야 한다. 요구가 무한 루프를 도는 것을 방지하기 위하여 프락시는 반드시 모든 별명(aliases), 지역적 변이 및 IP 주소 숫자를 포함한 모든 서버 이름을 인지할 수 있어야 한다. Request-Line 의 예는 다음과 같다.

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
```

향후 버전 HTTP 에서 모든 요구가 absoluteURI 로 이전될 수 있도록 하기 위해 모든 HTTP/1.1 서버는 비록 HTTP/1.1 클라이언트가 단지 프락시에 대한 요구에서만 그것을 생산한다 할지라도 반드시 요구의 absoluteURI 형식을 수용해야 한다.

가장 일반적인 형태의 Request-URI 는 원서버나 게이트웨이의 자원을 식별하는 데 사용한다. 이 경우 URI 의 절대적 경로는 반드시 Request-URI 처럼 전송(3.2.1 절의 abs_path 참조)되어야 하며 URI 의 네트워크 위치는 반드시 Host 헤더 필드를 이용하여 전송되어야 한다. 예를 들어 원서버에서 직접 자원을 조회하고자 하는 클라이언트는 "www.w3.org" 호스트의 포트 80 으로 TCP 접속을 한 다음 아래의 라인을 전송할 것이다.

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
```

위 내용 다음에 요구 메시지의 나머지 부분이 뒤따른다. 절대 경로는 절대 비어서는 안 된다. 원래 URI 의 절대 경로가 비어있을 때에는 반드시 "/" (서버의 루트 디렉토리)를 추가한다.

프락시가 Request-URI 에 아무런 경로가 없는 요구를 수신하고 명시된 method 가 별표 모양의 요구를 지원할 수 있으면 응답 메시지의 전달 경로 상의 (Request chain) 마지막 프락시는 반드시 요구 메시지에 마지막 Request-URI 로서 "*"를 첨부하여 전송해야 한다.

예를 들어 다음과 같은 요구 메시지를

```
OPTIONS  HTTP/1.1
```

프락시는 "".호스트의 포트 8001 과 연결한 다음 아래와 같이 전송할 것이다.

```
OPTIONS * HTTP/1.1
Host:
```

Request-URI 는 3.2.1 절에서 명시한 형식으로 전송된다. 원서버는 요구를 적절히 해석하기 위하여 반드시 Request-URI 를 해독해야 한다. 서버는 유효하지 않는 Request-URI 를 수신하면 적합한 상태 코드로 응답해야 한다.

[Page 36]

요구 메시지를 전송할 때 프락시는 어떤 방식으로든 위에서 설명한 것처럼 비어 있는 절대 경로를 "*"로 대체하는 것 외에는 프락시 내부의 구현 방법에 관계없이 절대로 Request-URI 의 "abs_path" 부분을 재작성해서는 안 된다.

주의 : "재작성 금지" 규칙은 원서버가 특정 목적을 위해서 예약되지 않은 URL 문자를 적절하게 사용하지 못하고 있을 때 요구의 의미를 변경하지 못하도록 한다. 구현자는 몇몇 HTTP/1.1 이전의 프락시는 Request-URI를 재작성하는 것으로 알려져 있음을 인식하고 있어야만 한다.

5.2 요구에 의해 식별되는 자원

HTTP/1.1 원서버는 인터넷 요구가 식별하는 정확한 자원은 Request-URI 및 Host 헤더 필드 모두를 조사하여 결정된다는 것을 인지하고 있어야 한다.

요구받은 호스트와 다른 자원을 허용하지 않는 원서버는 Host 헤더 필드 값을 무시할 수 있다. (그러나 HTTP/1.1 에서의 Host 지원에 관한 다른 필요 조건에 관하여는 19.5.1 절을 참조한다.)

요구받은 호스트에 기초하여 자원을 구별하는 원서버(때로 가상 호스트 또는 허영 호스트 이름이라고 불린다)는 HTTP/1.1 요구에 대한 자원을 결정할 때 다음의 규칙을 반드시 따라야 한다.

1. Request-URI 가 absoluteURI이면 호스트는 Request-URI의 일부분이다. 요구의 어떠한 Host 헤더 필드 값도 반드시 무시해야 한다.

2. Request-URI 가 absoluteURI가 아니면 요구는 Host 헤더 필드를 포함한다. 호스트는 Host 헤더 필드 값으로 결정된다.

3. 규칙 1, 2에 의하여 지정된 호스트가 서버의 유효한 호스트가 아니면 응답은 반드시 400(Bad Request) 에러 메시지이어야 한다.

Host 헤더 필드가 없는 HTTP/1.0 의 수신측은 정확하게 무슨 자원을 요구했는지 결정하기 위해 발견법을 (heuristics - 예를 들어 유일한 무엇인가의 특별한 호스트로의 URI 경로를 검사) 사용할 수도 있다.

5.3 요구 헤더 필드

요구 헤더 필드는 요구 및 클라이언트 자신에 관한 추가 정보를 클라이언트가 서버에게 전달할 수 있도록 한다. 이 필드는 프로그래밍 언어 method 호출시 사용하는 파라미터와 동일한 의미로 요구 변경자(request modifiers)의 역할을 수행한다.

[Page 37]

```
request-header          = Accept                ; 14.1 절
                        | Accept-Charset        ; 14.2 절
```

```
                | Accept-Encoding       ; 14.3 절
                | Accept-Language       ; 14.4 절
                | Authorization         ; 14.8 절
                | From                  ; 14.22 절
                | Host                  ; 14.23 절
                | If-Modified-Since     ; 14.24 절
                | If-Match              ; 14.25 절
                | If-None-Match         ; 14.26 절
                | If-Range              ; 14.27 절
                | If-Unmodified-Since   ; 14.28 절
                | Max-Forwards          ; 14.31 절
                | Proxy-Authorization   ; 14.34 절
                | Range                 ; 14.36 절
                | Referer               ; 14.37 절
                | User-Agent            ; 14.42 절
```

Request-Header 필드 이름은 규약 버전의 변경과 함께 확장했을 때만 신뢰성 있게 확장될 수 있다. 그러나 통신에 참여하는 모든 대상이 그것을 Request-Header 필드로 인지한다면 새롭거나 실험적인 헤더 필드에 요구 헤더의 의미를 적용할 수 있다. 인정되지 않은 헤더 필드는 Entity-Header 필드로 처리된다.

6 응답

요구 메시지를 수신하고 해석한 후 서버는 HTTP 응답 메시지로 응답한다.

```
Response      = Status-Line             ; 6.1 절
                *( general-header       ; 4.5 절
                 | response-header      ; 6.2 절
                 | Entity-Header )      ; 7.1 절
                CRLF
                [ message-body ]        ; 7.2 절
```

6.1 상태 라인(Status-Line)

응답 메시지의 첫 라인은 상태 라인이다. 상태 라인은 규약 버전과 이에 뒤따르는 숫자 상태 코드 및 연관된 텍스트 문구로 구성되어 있으며 각 요소는 SP 문자로 구분된다. CR 또는 LF 는 마지막 CRLF 에만 허용된다.

[Page 38]

```
Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

6.1.1 상태 코드 및 이유 문구

Status-Code(상태 코드) 요소는 요구 메시지를 이해하고 이에 따라 서비스를 제공하려는 데에 대한 결과로서 세 자리의 정수 코드이다. 이 코드는 10 장에 모두 정의되어 있다. Reason-Phrase(이유 문구)는

상태 코드에 대해 짧은 텍스트 형태의 설명을 제공하기 위해 사용한다. 상태 코드는 오토마타(automata)가 사용하고 이유 문구는 인간 사용자가 사용하기 위해서이다.

상태 코드의 첫 자리 숫자는 응답의 클래스를 규정하며 마지막 두 자리 숫자는 아무런 구분 역할을 가지고 있지 않다. 첫 자리에는 다섯 가지의 숫자가 올 수 있다.

- 1xx: 알림 정보 (Informational)
  - 요구가 수신되어 계속 처리

- 2xx: 성공 (Success)
  - 요구 메시지를 성공적으로 수신 및 해석을 하고 이를 허용

- 3xx: 방향 재설정 (Redirection)
  - 요구에 대한 처리를 완료하기 위하여 추가 조치가 필요

- 4xx: 클라이언트 오류 (Client Error)
  - 요구 메시지가 잘못된 형식으로 구성되어 있기나 제대로 처리할 수 없는 경우

- 5xx: 서버 오류 (Server Error)
  - 명백히 유효한 요구 메시지를 서버가 처리할 수 없을 때

HTTP/1.1 이 규정한 숫자 상태 코드의 개별적인 값 및 이에 상응하는 Reason-Phrase 의 예가 아래에 제시되어 있다. 여기에 열거된 이유 구문은 단지 권고 사항일 뿐이다. - 이유 구문은 규약에 영향을 미치지 않고도 지역적인 등가물(equivalents)로 대체할 수 있다.

```
Status-Code         = "100"  ; Continue(계속)
                    | "101"  ; Switching Protocols(규약 전환)
                    | "200"  ; OK
                    | "201"  ; Created(생성 되었음)
                    | "202"  ; Accepted(접수 되었음)
                    | "203"  ; Non-Authoritative Information(비 인증 정보)
                    | "204"  ; No Content (내용이 없음)
                    | "205"  ; Reset Content(내용을 지움)
                    | "206"  ; Partial Content(부분 내용)
                    | "300"  ; Multiple Choices(복수 선택)
                    | "301"  ; Moved Permanently(영구 이동)
                    | "302"  ; Moved Temporarily(임시 이동)
```

[Page 39]

```
                    | "303"  ; See Other(다른 것을 참조)
                    | "304"  ; Not Modified(변경되지 않았음)
                    | "305"  ; Use Proxy(프락시를 사용할 것)
                    | "400"  ; Bad Request(잘못된 요구)
                    | "401"  ; Unauthorized(인증되지 않았음)
                    | "402"  ; Payment Required(요금 지불 요청)
                    | "403"  ; Forbidden(금지되었음)
                    | "404"  ; Not Found(찾을 수 없음)
                    | "405"  ; Method Not Allowed(method를 사용할 수 없음)
                    | "406"  ; Not Acceptable (접수할 수 없음)
                    | "407"  ; Proxy Authentication Required(프락시 인증 필요)
```

```
                    | "408"   ; Request Time-out(요구 시간 초과)
                    | "409"   ; Conflict(충돌)
                    | "410"   ; Gone(내용물이 사라졌음)
                    | "411"   ; Length Required(길이가 필요함)
                    | "412"   ; Precondition Failed(사전 조건 충족 실패)
                    | "413"   ; Request Entity Too Large (요구 엔터티가 너무 큼)
                    | "414"   ; Request-URI Too Large(Request-URI가 너무 김)
                    | "415"   ; Unsupported Media Type(지원되지 않는 미디어 유형)
                    | "500"   ; Internal Server Error(서버 내부 에러)
                    | "501"   ; Not Implemented(구현되지 않았음)
                    | "502"   ; Bad Gateway(불량 게이트웨이)
                    | "503"   ; Service Unavailable(서비스를 사용할 수 없음)
                    | "504"   ; Gateway Time-out(게이트웨이 시간 초과).
                    | "505"   ; HTTP Version not supported (지원되지 않는 HTTP 버전)
                    | extension-code
  extension-code    = 3DIGIT
  Reason-Phrase     = *<TEXT, excluding CR, LF>
```

HTTP 상태 코드는 확장할 수 있다. HTTP 애플리케이션이 반드시 모든 등록된 상태 코드의 의미를 이해할 필요는 없다.(이것이 분명 바람직하기는 하다.) 그러나 애플리케이션은 반드시 첫 단위가 표시하는 상태 코드의 클래스를 이해해야 하며 인식할 수 없는 응답은 해당 클래스의 x00 상태 코드와 동일한 것으로 처리해야 한다. 인식되지 않은 응답은 절대 캐시해서는 안 된다. 예를 들어 클라이언트가 인식되지 않은 상태 코드 431 을 수신하였으면 클라이언트는 요구에 무엇인가 잘못이 있었으며 응답을 400 상태 코드로 수신한 것으로 안전하게 가정할 수 있다. 이러한 경우 사용자 에이전트는 응답과 함께 리턴 된 엔터티를 사용자에게 제시해야 한다. 엔터티는 대개 사람이 읽을 수 있는 정보(비 정상적인 상태를 설명하는 정보)를 포함하고 있기 때문이다.

[Page 40]

6.2 응답 헤더 필드

응답 헤더 필드는 서버가 Status-Line 에 표시할 수 없는 응답에 대한 추가 정보를 전달할 수 있도록 한다. 이러한 헤더 필드는 서버와 Request-URI 가 식별하는 자원에 추가적으로 접근하는 방법에 대한 정보를 제공한다.

```
  response-header   = Age                 ; 14.6 절
                    | Location            ; 14.30 절
                    | Proxy-Authenticate  ; 14.33 절
                    | Public              ; 14.35 절
                    | Retry-After         ; 14.38 절
                    | Server              ; 14.39 절
                    | Vary                ; 14.43 절
                    | Warning             ; 14.45 절
                    | WWW-Authenticate    ; 14.46 절
```

Response-header 필드 이름은 규약 버전의 변경과 함께 확장했을 때만 신뢰성 있게 확장될 수 있다. 그러나 새로운 또는 실험적으로 사용하는 헤더 필드에 응답 헤더의 의미를 적용할 수 있는데, 이 경우는 통신에 참여하는 모든 대상이 그것을 response-header 필드로 인식할 수 있을 때만 가능하다. 인식할 수 없는 헤더 필드는 Entity-Header 필드로 처리된다.

7 엔터티(Entity)

요구와 응답 메시지는 별도로 요구 method 나 응답 상태 코드에 의하여 제한을 받지 않는 한 엔터티를 전송할 수도 있다. 어떤 응답은 엔터티 헤더만을 포함할 수도 있지만 엔터티는 Entity-Header 필드와 Entity-Body 로 구성되어 있다. 이 절에서 송신자와 수신자는 누가 엔터티를 발송하고 누가 엔터티를 수신하는가에 따라 클라이언트와 서버 어느 쪽이든지 지칭할 수 있다.

7.1 엔터티 헤더 필드

엔터티 헤더 필드는 Entity-Body 에 대한 또는 본문이 없다면 요구 메시지를 통해 확인할 수 있는 자원에 대한 선택적인 메타 정보를 정의하고 있다.


```
Entity-Header         = Allow                ; 14.7 절
                      | Content-Base         ; 14.11 절
                      | Content-Encoding     ; 14.12 절
                      | Content-Language     ; 14.13 절
                      | Content-Length       ; 14.14 절
                      | Content-Location     ; 14.15 절
                      | Content-MD5          ; 14.16 절
                      | Content-Range        ; 14.17 절
                      | Content-Type         ; 14.18 절
                      | ETag                 ; 14.20 절
                      | Expires              ; 14.21 절
                      | Last-Modified        ; 14.29 절
                      | extension-header
      extension-header = message-header
```

Extension-Header 메커니즘은 규약을 변경하지 않고도 추가적인 Entity-Header 필드를 정의할 수 있게 한다. 그러나 이러한 필드를 수신자가 인식할 수 있다고 가정할 수는 없다. 인식할 수 없는 필드는 수신측이 무시해야 하며 프락시를 통하여 전송한다.

7.2 엔터티 본문 (Entity Body)

HTTP 요구나 응답과 함께 발송된 (만약 있다면) Entity-Body 는 Entity-Header 필드에서 규정한 포맷 및 인코딩에 따른다.

```
Entity-Body    = *OCTET
```

Entity-Body 는 4.3 절에서 설명한 대로 Message-Body 가 있을 때만 메시지 내에 존재한다. 안전하고 적절한 메시지 전송을 위해 적용되었을 수도 있는 Transfer-Encoding 을 해독하여 Message-Body 에서 Entity-Body 를 얻을 수 있다.

7.2.1 유형 (Type)

메시지에 Entity-Body 가 포함되어 있으면 해당 본문의 데이터 타입은 Content-Type 및 Content-Encoding 의 헤더 필드를 통하여 결정된다. 이는 2 계층의 순서가 정해진 인코딩 모델을 규정한다.

```
Entity-Body     := Content-Encoding( Content-Type( data ) )
```

Content-Type 은 메시지 본문 내용의 미디어 형식을 명시한다. Content-Encoding 은 대개 데이터를 압축할 목적으로 데이터에 적용된 추가적인 내용 코딩(요구된 자원의 속성이다)을 표시하는 데 사용할 수 있다.

[Page 42]

Entity-Body 를 포함하고 있는 모든 HTTP/1.1 메시지는 해당 본문의 미디어 형식을 규정하는 Content-Type 헤더 필드를 포함하여야 한다. Content-Type 필드가 미디어 형식 정보를 제공하지 않는 경우에만 수신측은 자원을 확인하는 데 사용되는 URL 이름 확장자 및/또는 내용 검사를 통하여 미디어 형식을 짐작하려 시도할 수도 있다. 계속 미디어 형식을 알 수 없다면 수신측은 그것을 "application/octet-stream" 유형으로 처리해야만 한다.

7.2.2 길이

Entity-Body 의 길이는 모든 전송 인코딩(Transfer-Coding)이 디코딩된 후의 Message-Body 의 길이이다. 4.4 절은 Message-Body 의 길이를 결정하는 방법을 규정하고 있다.

8 접속(Connections)

8.1 지속형 연결(Persistent Connections)

8.1.1 목적

지속형 연결 개념이 등장하기 이전에는 각 URL 의 정보를 가져오기 위해 매번 별도의 TCP 연결을 설정하여 HTTP 서버의 부하를 증가시키고 인터넷의 트래픽 혼잡을 유발했었다. 하이퍼링크되어 있는 이미지와 및 기타 관련 데이터의 사용은 종종 클라이언트가 아주 짧은 시간에 여러 개의 요구를 동일한 서버에 하도록 만들 수 있다. 이러한 성능 문제에 대한 분석을 [30][27]에서 참고할 수 있으며, 프로토타입 구현의 분석 및 결과를 [26]에서 참고할 수 있다.

지속형 HTTP 접속은 여러 가지 이점이 있다.

- TCP 연결을 시작하고 종료하는 회수를 줄임으로써 CPU 시간을 줄이고 TCP 규약 제어 블록에 사용되는 메모리를 절약할 수 있다.

- HTTP 요구와 응답이 연결선 상에서 파이프라인 될 수 있게 한다. 파이프라인을 사용하면 클라이언트는 각 응답을 기다리지 않고도 복수의 요구를 할 수 있어 하나의 TCP 연결을 효과적으로 빠른 시간 내에 이룩할 수 있다.

- TCP를 시작할 때 발생하는 패킷의 숫자를 감소시키고 네트워크의 혼잡 상태를 결정할 충분한 시간을 TCP에 주어 네트워크 혼잡을 줄일 수 있다.

- 오류가 발생해도 TCP 접속을 단절하지 않고도 이를 보고할 수 있기 때문에 HTTP가 좀 더 잘 작동될 수 있도록 한다. HTTP 향후 버전을 사용하는 클라이언트는 새로운 기능을 시도할 수 있으나 이전 서버와 통신을 할 때

에러가 발생하면 이전 버전으로 재시도 한다.

HTTP 구현은 지속형 연결을 구현해야 한다.

[Page 43]

8.1.2 전반적인 운영

HTTP/1.1 과 이전 버전의 HTTP 의 큰 차이점은 지속적인 접속이 모든 HTTP 접속의 기본 방식이라는 것이다. 별도의 표시가 없으면 클라이언트는 지속적인 접속을 유지한다고 가정한다.

지속적인 접속은 클라이언트와 서버가 TCP 연결 종결을 알릴 수 있는 메커니즘을 제공한다. 이러한 알림은 Connection 헤더 필드를 이용한다. 종료 신호가 통보되면 클라이언트는 더 이상 해당 연결에 요구를 보내서는 안 된다.

8.1.2.1 협상(Negotiation)

HTTP/1.1 서버는 요구 메시지에 "종료" Connection-Token 이 포함된 Connection 헤더가 발송되지 않는 한 HTTP/1.1 클라이언트는 지속적인 접속을 유지하고자 한다고 가정한다. 서버가 응답을 발송한 후 곧 바로 연결을 종료하고자 한다면 종료 Connection-Token 이 포함된 Connection 헤더를 발송해야만 한다.

HTTP/1.1 클라이언트는 접속이 계속 유지되기를 기대하지만 서버로부터의 응답이 종료 Connection-Token 의 Connection 헤더를 포함하고 있는가 여부에 따라 접속 유지 여부를 결정할 수도 있다. 클라이언트가 해당 요구 이상의 접속을 유지하기 원치 않는다면 클라이언트는 종료 Connection-Token 이 포함된 Connection 헤더를 발송해야 한다.

클라이언트 또는 서버가 Connection 헤더에 종료 토큰을 발송하면 해당 요구는 접속에 대한 마지막 요구가 된다.

클라이언트나 서버는 명확하게 표시되지 않는 한 1.1 이전의 HTTP 버전에서 지속적인 접속이 유지된다고 가정해서는 안 된다. HTTP/1.0 클라이언트와의 호환성 유지에 관한 정보는 19.7.1 절을 참조한다.

지속적으로 유지되기 위해서 연결선 상의 모든 메시지는 4.4 절에서 설명된 스스로 정의된 메시지 길이(예를 들면 접속 종료에 의해 규정되지 않는)를 포함하고 있어야 한다.

8.1.2.2 파이프라인 사용

지속적인 접속을 지원하는 클라이언트는 요구를 파이프라인(복수의 요구를 각각의 응답을 기다리지 않고 발송)할 수 있다. 서버는 반드시 이러한 요구에 대한 응답을 요구가 수신된 동일한 순서로 발송해야 한다.

[Page 44]

접속된 후 곧 바로 지속적인 접속이나 파이프라인(pipeline)을 예상하는 클라이언트는 첫 파이프라인 시도가 실패할 경우 재시도할 준비가 되어 있어야 한다. 클라이언트가 재시도를 했을 때 접속이 지속적인지 알기 전에는 파이프라인 기능을 절대로 사용해서는 안 된다. 클라이언트는 서버가 모든 상응하는 응답을 발송하기 전에 접속이 종료되었으며 재발송할 준비가 되어 있어야만 한다.

8.1.3 프락시 서버

프락시가 14.2.1 절에 명시된 Connection 헤더 필드의 특성을 정확하게 구현하는 것이 특히 중요하다.

프락시 서버는 연결하고 있는 클라이언트와 원서버(또는 다른 프락시 서버)의 지속적인 접속을 반드시 구분하여 알려야 한다. 각각의 지속적 접속은 단지 하나의 전송 링크에만 적용한다.

프락시 서버는 절대 HTTP/1.0 과 지속적인 접속을 설정해서는 안 된다.

8.1.4 실제적인 고려 사항

서버는 대개 비활성 접속을 더 이상 유지하지 않을 시간 초과 값을 가지고 있다. 프락시 서버는 클라이언트가 동일한 서버를 통하여 더 많은 접속을 설정하기 쉽기 때문에 이 값을 더 높게 할 수 있다. 지속적인 접속의 사용은 클라이언트나 서버의 시간 초과 길이에 어떠한 필요 조건을 두지 않는다.

클라이언트 또는 서버가 시간 초과 기능을 사용하고자 할 때 전송 접속 상에 종료를 알려야 한다. 클라이언트와 서버는 전송 선상의 다른 쪽의 접속 종료를 항상 주시하고 있다가 적절하게 반응하여야 한다. 클라이언트 또는 서버가 다른 쪽의 종료를 신속하게 감지하지 못하면 네트워크 상의 불필요한 자원 낭비를 초래하게 된다.

클라이언트, 서버 또는 프락시는 언제든지 전송 접속을 종료할 수 있다. 예를 들어 클라이언트는 서버가 "사용하지 않는" 접속을 종료하기로 결정한 바로 그 순간에 새로운 요구 발송을 시작했을 수 있다. 서버 관점에서 보면 접속은 사용하지 않고 있는 동안은 종료되고 있는 것이며 클라이언트의 관점에서 보면 요구가 처리되고 있는 것이다.

이는 클라이언트, 서버 및 프락시가 반드시 동시 종료 이벤트에서 회복할 수 있어야 한다는 것을 의미한다. 클라이언트 소프트웨어는 전송 접속을 재개할 수 있어야 하며 요구 method 가 멱등원(冪等元) method(9.1.2 절 참조)일 경우 사용자와의 상호 작용 없이도 중단된 요구를 재전송할 수 있어야 한다. 다른 method 는 사용자 에이전트가 인간 운영자에게 요구를 재시도할 수 있는 선택권을 줄 수도 있지만 자동적으로 재시도해서는 안 된다.

[Page 45]

그러나 두 번째 요구가 실패할 경우 자동적인 재시도를 반복해서는 안 된다.

서버는 가능하다면 항상 한 접속 건 당 최소한 하나의 요구에 응답해야 한다. 서버는 네트워크나 클라이언트 실패가 발생할지도 모르는 상황이 아니면 응답을 전송하는 도중에 접속을 종료해서는 안 된다.

지속적인 접속을 사용하는 클라이언트는 특정 서버로의 동시 접속 숫자에 제한을 두어야 한다. 단독 사용자 클라이언트는 최대 2 개의 서버나 프락시 접속을 유지해야 한다. 프락시는 최대 2*N 개의 서버나 프락시 접속을 할 수 있으며 여기서 N 은 사용하고 있는 동시 사용자의 숫자이다. 이러한 지침은 HTTP 응답 시간을 향상하고 인터넷이나 다른 네트워크의 혼잡을 피하기 위함이다.

8.2 메시지 전송 필요 조건

일반적인 필요 조건:

- HTTP/1.1 서버는 지속적인 접속을 유지하고 있어야 하며 일시적인 오버로드(overloads)를 완화 시키기 위하여

클라이언트가 재시도할 것이라는 예상으로 접속을 종결하기보다는 흐름 제어 메커니즘을 사용해야 한다. 후자의 방법을 사용하면 네트워크 혼잡이 가중될 수 있다.

- Message-Body를 발송하는 HTTP/1.1 (또는 이후) 클라이언트는 요구를 전송하는 동안 네트워크 접속에 에러가 발생하는지 점검해야 한다. 클라이언트가 에러를 감지하였으면 본체 전송을 즉시 중단해야 한다. 본체가 "덩어리" 인코딩(3.6 절) 기법을 사용하여 전송되고 있을 경우에는 제로 길이의 덩어리와 비어 있는 주석을 사용하여 메시지의 종료를 조기에 표시할 수도 있다. 본문에 앞서 Content-Length 헤더가 있으면 클라이언트는 접속을 종료해야 한다.

- HTTP/1.1 (또는 이후) 클라이언트는 정상적인 응답이 뒤 따르는 100(Continue) 상태 메시지를 접수할 준비가 반드시 되어 있어야 한다.

- HTTP/1.0 (또는 이전) 클라이언트로부터 요구를 수신한 HTTP/1.1 (또는 이후) 서버는 100 (Continue) 응답 메시지를 전송해서는 안 된다. 서버는 요구가 정상적으로 완료될 때까지(이렇게 하여 중단된 요구가 발생하지 않도록 함) 기다리던가 조기에 접속을 종료해야 한다.

[Page 46]

HTTP/1.1 (또는 이후) 클라이언트로부터 이러한 필요 조건을 만족해야 하는 method 를 수신하면 HTTP/1.1 (또는 이후) 서버는 반드시 100 (Continue) 상태로 응답하고 입력 스트림에서 계속적으로 요구를 읽어 들이든지 에러 상태 코드로 응답해야 한다. 에러 상태 코드로 응답하였으면 전송(TCP) 접속을 종료하던지 또는 계속적으로 요구를 읽어 들이고 요구의 나머지 부분을 폐기할 수 있다. 에러 상태 코드를 리턴했을 때는 요구 받은 method 를 절대로 처리해서는 안 된다.

클라이언트는 최소한 가장 최근에 사용된 서버의 버전 숫자를 기억하고 있어야 한다. HTTP/1.1 클라이언트가 HTTP/1.1 또는 이후의 응답을 서버로부터 얻고 서버로부터 상태 메시지를 수신하기 전에 연결이 종료되었으면 클라이언트는 요구 method 가 멱등원(冪等元) method(9.1.2 절 참조)이 아닌 이상 사용자와의 상호 작용 없이 요구를 재시도해야 한다. 사용자 에이전트가 인간 운영자에게 요구를 재시도할 선택권을 줄 수는 있지만 다른 method 를 자동적으로 재시도해서는 안 된다.

클라이언트가 요구를 재시도 했으면 클라이언트는
- 반드시 먼저 요구 헤더 필드를 발송해야 하며 그런 다음
- 서버가 100 (Continue) 응답(이 경우 클라이언트는 계속 진행해야 한다)이나 에러 코드를 응답할 때 까지 반드시 기다려야 한다.

HTTP/1.1 클라이언트가 HTTP/1.1 이나 이후 응답을 서버로부터 얻지 못하면 서버가 HTTP/1.0 이나 이전 버전을 구현하고 있으면 100 (Continue) 응답을 사용하지 않을 것으로 가정해야 한다. 이 경우 클라이언트가 서버로부터 상태 메시지를 접수하기 전에 접속이 종료되면 클라이언트는 요구를 재시도해야 한다. 클라이언트가 이 HTTP/1.0 서버에 요구를 재시도할 때 아래의 "바이너리 지수 백오프(binary exponential backoff)" 알고리즘을 사용하여 신뢰할 수 있는 응답을 얻었음을 확인해야 한다.

1. 서버로 새로운 접속을 시작한다.
2. Request-Header을 전송한다.
3. 변수 R을 서버로의 예상 왕복 여행 시간으로 초기화한다.(예를 들면 접속을 설정하기 위해 소요되는 시간에 기초한다)
4. T = R * (2**N)을 계산한다. 이때 N은 이 요구를 이전에 재시도한 숫자이다.
5. 서버로부터 에러 응답을 기다리던가 T 초 동안 기다린다. (어떤 것이든 먼저 오는 것)

[Page 47]

6. 아무런 에러 응답도 수신되지 않았으면 T 초가 경과한 후 요구의 본문을 전송한다.
7. 클라이언트가 접속이 조기에 종료되는 것을 알았으면 요구가 접수되었거나, 에러 응답을 수신하였거나 또는

사용자가 더 이상 기다릴 수 없어 재시도 절차를 종료할 때 까지 첫 스텝부터 계속한다.

서버 버전에 관계없이 에러 상태가 접수되었으면 클라이언트는
- 절대로 계속해서는 안 되며
- 메시지 전송을 완료하지 않았으면 접속을 반드시 종료해야 한다.

다른 어떤 상태 메시지를 접수하기 이전에 100 (Continue)을 수신한 후 접속을 종료한 것을 인지한 HTTP/1.1 (또는 이후) 클라이언트는 요구를 재시도해야 하며 100 (Continue) 응답을 기다릴 필요가 없다.(그러나 이것이 구현 방법을 단순하게 한다면 기다려도 된다.)

9 Method 정의

HTTP/1.1 에서 사용되는 일반적인 method 세트를 아래에 규정하였다. 이 세트를 확장할 수 있지만 추가된 method 를 별도로 확장된 클라이언트와 서버가 동일한 의미를 공유하고 있다고 가정할 수 없다.호스트 Request-Header 필드(14.23 절)는 반드시 모든 HTTP/1.1 요구를 따라야 한다.

9.1 안전 및 멱등원(冪等元) method

9.1.1 안전 method

구현자는 소프트웨어가 사용자와의 상호작용이 인터넷을 통하여 표시된다는 점을 인지해야 하며 사용자에게 자신이 취하는 행동이 자신과 다른 사용자에게 예상하지 못한 중요성을 가질 수 있다는 점을 인지시키도록 주의해야 한다.

특히 GET 및 HEAD method 가 조회 이외의 작업을 수행하는 중요성을 가져서는 안 된다는 관례가 확립되었다. 이러한 method 는 안전한 것으로 간주해야 한다. 이렇게 하여 사용자 에이전트가 POST, PUT 및 DELETE 와 같은 다른 method 를 특별한 방식으로 표현할 수 있게 하며 사용자가 안전하지 못한 처리를 요구하고 있다는 사실을 인식할 수 있게 한다.

당연히 서버가 GET 요구를 수행한 결과로서 부작용을 발생하지 않고 있음을 보장할 수 없다.

[Page 48]

사실상 몇몇 역동적인 자원은 이것을 하나의 기능으로 보고 있다. 중요한 구별 점은 사용자가 부작용을 요청하지 않았기 때문에 부작용에 대한 책임을 부과할 수는 없다는 것이다.

9.1.2 멱등원(冪等元) method

Method 는 또한 N > 0 과 동일한 요구의 부작용이 단일 요구의 부작용과 동일하다는 점에서 "멱등원" 특성을 가질 수 있다.(에러 또는 만기일의 문제는 별도로 하고) GET, HEAD, PUT 및 DELETE method 는 이 특성을 공유하고 있다.

9.2 OPTIONS

OPTIONS method 는 Request-URI 에 의하여 식별되는 Request/Response chain 에서 사용할 수 있는 통신 선택 사항에 관한 정보 요구를 표시한다. 이 method 는 클라이언트가 자원 처리를 시도하거나 자원

조회를 시작하지 않고도 선택 사항 및/또는 자원과 관련된 필요 조건, 서버의 처리 능력을 결정할 수 있게 한다.

서버의 응답이 에러가 아닌 이상 응답은 통신 선택 사항이라고 간주할 수 있는 것 이외의 엔터티 정보를 포함해서는 안 된다.(예를 들어 Allow 는 적합하지만 Content-Type 은 적합하지 않다).

Request-URI 가 별표("*")이면 OPTIONS 요구는 전체를 서버에 적용하려는 것이다. 200 응답은 모든 적용 가능한 일반 필드 또는 Response-Header 필드 이외에 이 규격에서 규정하지 않는 모든 확장을 포함하여 서버가 구현한(예 Public) 선택 기능을 표시하는 모든 헤더 필드를 포함해야 한다. 5.1.2 절에서 설명된 것처럼 "OPTIONS *" 요구는 경로 정보 없이 Request-URI 에 목적지 서버를 명시함으로써 프락시를 통하여 적용할 수 있다.

Request-URI 가 별표가 아니면 OPTIONS 요구는 해당 자원과 통신할 때 사용할 수 있는 선택 사항에만 적용된다. 200 응답은 모든 적용 가능한 일반 필드 또는 Response-Header 필드 이외에 이 규격에서 규정하지 않는 모든 확장을 포함하여 서버가 구현한(예 Allow) 선택 기능을 표시하는 모든 헤더 필드를 포함해야 한다. OPTIONS 요구가 프락시를 통한다면 프락시는 프락시의 성능에 관계되는 선택 사항 및 프락시를 통하여 사용할 수 없는 것으로 알려진 선택 사항을 제외할 수 있도록 응답을 반드시 편집해야 한다.

[Page 49]

9.3 GET

GET method 는 Request-URI 가 식별하는 모든 정보(엔터티의 형태로)를 조회한다는 것을 의미한다. Request-URI 가 데이터를 생성하는 프로세스를 참조한다면 텍스트가 우연히 프로세스의 산출물이 아닌 이상 엔터티로서 리턴 되는 것은 프로세스의 소스 텍스트가 아니라 생성된 데이터이다.

GET method 의 의미는 요구 메시지가 If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match 또는 If-Range 헤더 필드를 포함하고 있으면 "조건적인 GET"으로 변화한다. 조건적인 GET method 는 엔터티가 조건 헤더 필드가 명시한 조건 하에서만 전송되도록 요청하는 것이다. 조건적 GET method 는 복수의 요구나 클라이언트가 이미 보유하고 있는 데이터를 전송하지 않고도 캐시 된 엔터티를 갱신할 수 있도록 함으로서 불필요한 네트워크 사용을 감소시키기 위해 사용한다.

GET method 의 의미는 요구 메시지가 Range 헤더 필드를 포함하고 있으면 "부분적인 GET"으로 변화한다. 부분적인 GET method 는 14.36 절에 설명된 것처럼 엔터티의 일 부분만 전송하도록 요청하는 것이다. 부분적 GET method 는 클라이언트가 이미 보유하고 있는 데이터를 전송하지 않고도 부분적으로 조회한 엔터티가 완성될 수 있도록 함으로써 불필요한 네트워크 사용을 감소시키기 위해 사용한다.

GET 요구에 대한 응답은 13 장에 설명된 대로 HTTP 캐시 요구 필요 조건을 만족할 경우에만 캐시할 수 있다.

9.4 HEAD

HEAD method 는 서버가 응답 메시지에 Message-Body 를 반드시 리턴해야 한다는 것 이외에는 GET 과 동일하다. HEAD 요구에 대한 응답으로 HTTP 헤더에 포함된 메타 정보는 GET 요구에 대한 응답으로 발송된 정보와 반드시 동일해야 한다. 이 method 는 Entity-Body 자체를 전송하지 않고도 요구가

내포하는 엔터티에 대한 메타 정보를 얻는 데 사용할 수 있다. 이 method 는 종종 하이퍼텍스트 링크의 유효성, 접근성 및 최근의 변경 사항을 테스트하기 위해 사용된다.

HEAD 요구에 대한 응답은 응답에 포함된 정보를 해당 자원의 이전 캐시 엔터티를 갱신하는 데 사용할 수 있다는 의미에서 캐시할 수 있다. 새로운 필드 값이 캐시 된 엔터티가 현재 의 엔터티 (Content-Length, Content-MD5, ETag 또는 Last-Modified 의 변화에 의해 표시되는 것과 같은)와 상이함을 표시할 때는 캐시는 반드시 이 캐시 엔트리를 낡을 것으로 취급해야 한다.

[Page 50]

9.5 POST

POST method 는 서버에게 Request-Line 의 Request-URI 가 식별하는 자원의 새로운 부속물로서 요구에 포함된 엔터티를 접수할 것을 요구하는 데 사용한다. POST 는 다음의 기능을 수행하는 일관된 method 를 사용할 수 있도록 디자인 되었다.

- 기존 자원의 주해;
- 게시판, 뉴스그룹, 편지 발송 목록 또는 유사한 기사 그룹으로 메시지를 송부;
- 폼을 제출한 결과로 발생한 데이터 블록을 데이터 처리 프로세스에 제공;
- 추가 작업을 통한 데이터 베이스의 확장.

POST method 가 실제적으로 수행하는 기능은 서버가 결정하며 보통 Request-URI 에 달려 있다. 발송된 엔터티는 파일이 파일을 포함하고 있는 디렉토리에 종속되고, 뉴스 기사가 발송한 뉴스그룹에 종속되며 레코드가 데이터베이스에 종속되듯이 해당 URI 에 종속된다.

POST method 가 수행하는 작업이 URI 로 식별할 수 있는 자원을 생성하지 않을 수도 있다. 이러한 경우 응답이 결과를 설명하는 엔터티를 포함하고 있는가 여부에 따라 200(OK)이나 204(No Content)가 적절한 응답 상태이다. 새로운 자원이 원서버에서 생성되었다면 응답은 201(Created)이어야 하고 요구의 상태를 설명하며 새로운 자원 및 Location 헤더 (14.30 절 참조)를 지칭하는 엔터티를 포함해야 한다.

이 method 에 대한 응답은 응답이 적절한 Cache-Control 또는 Expires 헤더 필드를 포함하지 않는 한 캐시할 수 없다. 그러나 303(See Other) 응답을 사용하여 사용자 에이전트에게 캐시할 수 있는 자원을 조회하도록 지시할 수 있다.

POST 요구는 8.2 절에 설정된 메시지 전송 필요 조건을 반드시 따라야 한다.

[Page 51]

9.6 PUT

PUT method 는 동봉된 엔터티를 제공된 Request-URI 에 저장하도록 요구한다. Request-URI 가 이미 존재하는 자원을 지칭할 경우 동봉된 엔터티는 원서버에 있는 엔터티의 변경된 버전으로 간주해야 한다. Request-URI 가 기존 자원을 지칭하지 않고 URI 가 요구하는 사용자 에이전트가 새로운 자원으로 규정할 수 있다면 원서버는 해당 URI 로 자원을 생성할 수 있다. 만약 새로운 자원이 생성되었으면 원서버는 사용자 에이전트에게 201(Created) 응답을 알려야 한다. 기존 자원이 변경되었다면 200(OK)이나 204(No Content) 응답 코드를 발송하여 요구를 성공적으로 완료하였음을 표시하여야 한다. Request-URI 로 자원을 생성하거나 변경할 수 없는 경우에는 문제의 기본 성격을 반영하는 적절한 에러 응답을 발송해야 한다. 엔터티의 수신측은 이해 또는 구현할 수 없는 Content-* (예: Content-Range) 헤더를 반드시 무시해야 하고 이러한 경우 501(Not Implemented) 응답을 리턴해야 한다.

요구가 캐시를 통과할 경우 Request-URI는 하나 또는 그 이상의 현재 캐시 된 엔터티를 식별한다. 이러한 엔터티는 낡은 것으로 취급해야 하며 이러한 method에 대한 응답은 캐시할 수 없다.

POST와 PUT 요구의 근본적인 차이점은 Request-URI의 다른 의미에 반영된다. POST의 URI는 동봉된 엔터티를 처리할 자원을 식별한다. 그 자원은 데이터를 접수하는 프로세스, 다른 규약으로의 게이트웨이 또는 주석을 접수하는 별도의 엔터티일 수 있다. 이에 비하여 PUT 요구의 URI는 요구에 포함된 엔터티를 식별한다. - 사용자 에이전트는 어떤 URI를 의도하고 있으며 서버는 요구를 다른 자원에 적용해서는 절대로 안 된다는 것을 알고 있다. 만약 서버가 해당 요구를 다른 URI에 적용하고자 한다면 서버는 301(Moved Permanently) 응답을 반드시 발송해야 한다. 그러면 사용자 에이전트는 해당 요구의 방향을 재설정 할 것인지에 관한 자신의 결정을 한다.

단일 자원이 복수의 상이한 URI에 의하여 식별될 수 있다. 예를 들어 기사(article)는 각각의 특별한 버전을 식별하는 URI와 구별되는 "현재 버전"을 확인하기 위한 URI를 가질 수 있다. 이 경우 일반 URI의 PUT 요구는 원서버에 의하여 규정되는 복수의 URI를 생성할 수도 있다.

HTTP/1.1은 PUT method가 어떻게 원서버의 상태에 영향을 미치는가에 대해서는 규정하지 않는다.

PUT 요구는 8.2 절에 설정된 메시지 전송 필요 조건을 반드시 따라야 한다.

[Page 52]

9.7 DELETE

DELETE method는 Request-URI가 식별하는 자원을 삭제하도록 원서버에 요구한다. 이 method는 원서버에서 사용자의 개입(또는 다른 방법)에 의하여 무시될 수 있다. 클라이언트는 비록 원서버에서 발송한 상태 코드가 해당 작업이 성공적으로 완수되었다는 표시를 하여도 실제로 작업이 완료되었다는 보장을 받을 수 없다. 그러나 서버는 요구를 접수한 시점에서 자원을 삭제하거나 접근할 수 없는 위치로 이동할 의사가 없는 한 성공을 표시해서는 안 된다. 성공적인 응답은 응답이 상태를 설명하는 엔터티를 포함한다면 200 (OK), 처리가 시작되지 않았으면 202 (Accepted), 응답은 OK이나 엔터티를 포함하지 않고 있으면 204 (No Content)이다.

요구가 캐시를 통과할 경우 Request-URI는 하나 또는 그 이상의 현재 캐시 된 엔터티를 식별한다. 이러한 엔터티는 낡은 것으로 취급해야 하며 이러한 method에 대한 응답은 캐시할 수 없다.

9.8 TRACE

TRACE method는 요구 메시지의 원격지, 애플리케이션-계층 루프백(loop back)을 호출하는 데 사용한다. 응답의 최종 수신측은 클라이언트에게 되돌려 진 메시지를 200(OK) 응답의 Entity-Body로 수신해야 한다. 마지막 수신측은 메시지의 Max-Forwards 제로 값(14.31 절)을 수신하는 원서버, 첫 프락시 또는 게이트웨이이다. TRACE 요구는 절대 엔터티를 포함해서는 안 된다.

TRACE는 클라이언트가 Request chain의 다른 끝 쪽에 무엇이 수신되는가를 알 수 있게 하며 그 데이터를 시험 또는 진단 정보로 사용한다. Via 헤더 필드(14.44 절)의 값은 Request chain의 추적 역할을 수행하기 때문에 특히 주목할 만하다. Max-Forwards 헤더 필드를 사용하면 클라이언트가 Request chain의 길이를 제한할 수 있으며 이는 무한 루프에서 메시지를 전달하는 프락시 고리를 테스트하는 데 유용하다.

성공적이면 응답은 "message/http" 의 Content-Type 을 가진 Entity-Body 의 전체 요구 메시지를 포함할 수 있어야 한다. 이러한 method 에 대한 응답을 절대 캐시해서는 안 된다.

10 Status Code Definitions

각 Status-Code 가 어떤 method 를 따를 수 있는가에 대한 설명과 응답에서 필요로 하는 헤더 정보를 포함하여 아래에 설명되어 있다.

[Page 53]

10.1 정보를 알려 주는 1xx

이 상태 코드 클래스는 잠적적인 응답을 표시하며 Status-Line 과 선택적인 헤더로 구성되어 있다. 이 클래스는 빈 라인으로 종료된다. HTTP/1.0 은 어떠한 1xx 상태 코드로 정의하지 않기 때문에 실험적인 상황 이외에 서버는 1xx 응답을 HTTP/1.0 클라이언트에 발송해서는 안 된다.

10.1.1 100 계속

클라이언트는 요구를 계속 진행할 수 있다. 이 잠정적인 응답은 클라이언트에게 응답의 시초 부분이 수신되었으며 서버가 아직 거부하지 않았음을 알리는 데 사용한다. 클라이언트는 요구의 나머지 부분을 발송하여야 하며 요구가 완료 되었으면 이 응답을 무시해야 한다. 서버는 요구가 완료된 다음 마지막 응답을 발송한다.

10.1.2 101 규약 전환

서버가 이해하였으며 기꺼이 Upgrade 메시지 헤더 필드(14.41 절)를 통하여 접속에 사용되고 있는 애플리케이션 규약 변경에 관한 클라이언트의 요구에 따른다. 서버는 101 응답을 종료하는 빈 라인 바로 다음 응답 메시지의 Upgrade 헤더 필드가 정의한 규약으로 전환할 것이다.

규약은 전환하는 것이 유리한 경우에만 전환된다. 예를 들어 새로운 버전의 HTTP 로 전환하는 것이 이전 버전을 사용하는 것보다 유리하며 해당 기능을 사용하는 자원을 배달할 때 실시간, 동시 규약으로 전환하는 것이 유리하다.

10.2 성공을 알리는 2xx

이 상태 코드 클래스는 클라이언트의 요구가 성공적으로 수신, 해석 및 접수되었음을 표시한다.

10.2.1 200 OK

요구를 성공적으로 전달하였다. 응답과 함께 리턴 되는 정보는 요구에 사용된 method 에 달려 있다.

예를 들면:

GET 요구한 자원에 상응하는 엔터티는 응답에 포함되어 발송된다.

HEAD 요구한 자원에 상응하는 Entity-Header 필드는 Message-Body 없이 응답에 포함되어 발송된다.

[Page 54]

POST 처리 결과를 설명 또는 포함하는 엔터티.
TRACE 수신 서버가 수신한 요구 메시지를 포함하고 있는 엔터티

10.2.2 201 Created (생성 되었음)

요구가 충족되었으며 새로운 자원이 생성되고 있다. 새로 생성된 자원은 응답 엔터티의 리턴 된 URI 를 통하여 참조할 수 있으며 자원의 가장 상세한 URL 은 Location 헤더 필드로 알 수 있다. 원서버는 201 상태 코드를 리턴하기 전에 반드시 자원을 생성해야 한다. 처리가 즉각적으로 수행될 수 없을 때에 서버는 202(Accepted) 응답으로 대신 응해야 한다.

10.2.3 202 Accepted (접수 되었음)

처리를 위해 응답을 접수하였으나 처리는 완료되지 않았다. 요구는 엔터티의 처리 과정에서 허용되지 않을 수도 있기 때문에 궁극적으로 처리될 수도 있고 처리되지 않을 수도 있다. 이와 같은 동시 작업에서 상태 코드를 재발송하는 설비는 없다.

202 응답은 의도적으로 작업을 수행하지 않는다. 이 응답의 목적은 서버가 사용자 에이전트가 프로세스가 완료될 때까지 서버에 지속적으로 연결되지 않고도 다른 프로세스에 대한 요구(하루에 한 번만 실행되는 배치 지향적인 프로세스일 수도 있다.)를 접수할 수 있도록 하는 데 있다. 이 응답을 리턴하는 엔터티는 상태 점검자(monitor)에 대한 지시자 또는 사용자가 언제 요구가 완료될 수 있는지에 대한 예상 및 요구의 현재 상태에 대한 표시를 포함해야 한다.

10.2.4 203 Non-Authoritative Information(비 인증 정보)

Entity-Header 의 리턴 된 메타 정보는 서버에서 사용할 수 있는 정의 세트가 아니고 지역 또는 제 3 자의 복사본에서 수집한 것이다. 제시된 세트는 원래 버전의 하부 세트 또는 상위 세트일 수 있다. 예를 들어 자원에 대한 지역적 주해 정보를 포함하면 원서버가 알고 있는 메타 정보에 대한 상위 세트를 만들어 낼 수도 있다. 이 응답 코드를 사용하는 것은 의무사항이 아니며 응답이 203 이 아니면 200 (OK)일 때만 적합하다.

10.2.5 204 No Content(내용이 없음)

서버가 요구를 완전히 처리 했으나 반송할 새로운 정보가 없다. 클라이언트가 사용자 에이전트이면 요구를 발송하도록 한 문서 내용을 변경해서는 안 된다.

[Page 55]

이 응답은 주로 사용자 에이전트의 문서 내용에 대한 변화를 초래하지 않고 처리를 위한 입력을 실행하도록 하기 위해 사용한다. 응답은 Entity-Header 형태의 새로운 메타 정보를 포함한다. 이 정보는 사용자 에이전트의 현재 문서에 적용해야 한다.

204 응답은 Message-Body 를 포함해서는 안되며 항상 헤더 필드 다음의 첫 빈 라인으로 종료된다.

10.2.6 205 Reset Content(내용을 지움).

서버가 요구를 완전히 처리하였으며 사용자 에이전트는 요구를 발송하도록 한 문서의 내용을 지워야 한다. 이 응답은 주로 사용자 입력을 통하여 처리를 위한 입력이 발생하도록 하기 위해 사용한다. 이 응답 뒤에 입력을 수행한 폼을 지워 사용자가 다른 입력 요구를 쉽게 시작할 수 있게 한다. 이 응답은 엔터티를 포함해서는 안 된다.

10.2.7 206 Partial Content(부분적 내용).

서버가 자원에 대한 부분적 GET 요구를 완료하였다. 이 요구는 반드시 원하는 영역을 표시하는 Range 헤더 필드(14.36 절)를 포함해야 한다. 응답은 이 응답에 포함된 영역을 표시하는 Content-Range 헤더 필드(14.17 절)나 각 파트의 Content-Range 필드를 포함하는 multipart/byteranges Content-Type 을 포함해야 한다. multipart/byteranges 를 사용하지 않았으면 응답의 Content-Length 헤더 필드는 Message-Body 로 전송된 OCTET 의 실제 숫자와 정확하게 일치해야 한다.

Range 및 Content-Range 헤더를 지원하지 않는 캐시는 206(Partial Content) 응답을 캐시해서는 안 된다.

10.3 (방향을 재설정하는 3xx)

이 상태 코드 클래스는 사용자 에이전트가 요구를 완전히 처리하기 위해서는 추가적인 처리가 필요하다는 것을 표시한다. 요구되는 처리는 두 번째 요구에 사용된 method 가 GET 또는 HEAD 일 경우에만 사용자와의 상호작용 없이도 수행될 수 있다. 사용자 에이전트는 이러한 방향 재설정이 무한 루프를 표시하는 것이기 때문에 다섯 번 이상 자동적으로 요구 방향 재설정을 해서는 안 된다.

[Page 56]

10.3.1 300 Multiple Choices (복수 선택)

요구된 자원이 각자 자신 특유의 위치를 가지고 있는 표현 세트 중의 하나와 대응되며 사용자(또는 사용자 에이전트)가 선호하는 표현 방식을 선택하고 요구를 해당 위치로 재설정할 수 있도록 에이전트가 주도하는(agent-driven) 협상 정보가 제공된다.

HEAD 요구가 아닌 이상 응답은 사용자 또는 사용자 에이전트가 가장 적합한 것을 선택할 수 있는 자원 특징 및 위의 목록을 포함한 엔터티를 포함한다. 엔터티 포맷은 Content-Type 헤더 필드가 설정한 media type 에 의해 명시된다. 사용자 에이전트의 포맷 및 성능에 따라 가장 적합한 선택을 결정하는 것은 자동으로 수행될 수 있다. 그러나 이 규격은 이러한 자동 선택의 표준에 대하여 아무런 규정도 하지 않는다.

서버가 선호하는 표시 방법을 가지고 있으면 Location 필드에 해당 표시 방법에 대한 상세한 URL 을 포함해야 한다. 사용자 에이전트는 Location 필드 값을 이용하여 자동으로 방향을 재설정할 수 있다. 이 응답은 별도의 표시가 없는 한 캐시할 수 있다.

10.3.2 301 301 Moved Permanently (영구 이동)

요구된 자원에 새로운 영구 URI 가 할당되었으며 향후 이 자원에 대한 참조는 리턴 된 URI 중 하나를 이용하여 이루어질 수 있다. 링크를 편집할 수 있는 능력이 있는 클라이언트는 가능하다면 Request-URI

에 대한 참조를 서버가 리턴한 하나 또는 그 이상의 새로운 참고처로 자동적으로 재링크시켜야 한다. 다르게 표시되어 있지 않으면 이 응답은 캐시할 수 있다.

새로운 URI가 위치이면 해당 URL은 응답의 Location 필드가 부여해야 한다. 요구 method가 HEAD가 아니면 응답의 엔터티는 새로운 URI로의 하이퍼링크가 표시된 짧은 하이퍼텍스트 주석을 포함하고 있어야 한다.

GET 또는 HEAD 이외의 요구에 대한 응답에 301 상태 코드가 접수되면 사용자 에이전트는 사용자가 확인하지 않는 한 요구를 발행한 조건을 변경할 수도 있기 때문에 자동적으로 요구의 방향을 재설정해서는 안 된다.

주의 : 301 상태 코드를 수신한 후 자동적으로 POST 요구의 방향을 재설정할 때 기존의 몇몇 HTTP/1.0 사용자 에이전트는 실수로 POST 요구를 GET 요구로 변경한다.

[Page 57]

10.3.3 302 Moved Temporarily(임시 이동)

요구된 자원이 별도의 URI에 임시로 보관되어 있다. 방향 재설정은 종종 변경될 수 있기 때문에 클라이언트는 향후 요구를 위해서 계속해서 Request-URI를 사용해야 한다. 이 응답은 Cache-Control 또는 Expires 헤더 필드가 표시할 경우에만 캐시할 수 있다.

새로운 URI가 위치이면 해당 URL은 응답의 Location 필드가 부여해야 한다. 요구 method가 HEAD가 아니면 응답의 엔터티는 새로운 URI로의 하이퍼링크가 표시된 짧은 하이퍼텍스트 주석을 포함하고 있어야 한다.

GET 또는 HEAD 이외의 요구에 대한 응답에 301 상태 코드가 접수되면 사용자 에이전트는 사용자가 확인하지 않는 한 요구를 발행한 조건을 변경할 수도 있기 때문에 자동적으로 요구의 방향을 재설정해서는 안 된다.

주의 : 301 상태 코드를 수신한 후 자동적으로 POST 요구의 방향을 재설정할 때 기존의 몇몇 HTTP/1.0 사용자 에이전트는 실수로 POST 요구를 GET 요구로 변경한다.

10.3.4 303 See Other(다른 것을 참조)

요구된 자원이 별도의 URI에 임시로 보관되어 있으며 해당 자원에서 GET method를 사용하여 조회해야 한다. 이 method는 주로 POST가 활성화한 스크립트의 산출물을 사용자 에이전트가 선택된 자원으로 방향을 재설정할 수 있도록 하기 위해 사용된다. 새로운 URI는 처음 요구된 자원에 대한 대체 참고처가 아니다. 303 응답은 캐시할 수 없으나 두 번째(재설정된) 요구에 대한 응답은 캐시할 수 있다.

GET 또는 HEAD 이외의 요구에 대한 응답에 301 상태 코드가 접수되면 사용자 에이전트는 사용자가 확인하지 않는 한 요구를 발행한 조건을 변경할 수도 있기 때문에 자동적으로 요구의 방향을 재설정해서는 안 된다.

10.3.5 304 Not Modified(변경되지 않았음)

클라이언트가 조건적 GET 요구를 실행했고 접근할 수 있으나 문서가 변경되지 않았으면 서버는 이 상태 코드로 응답해야 한다. 이 응답은 Message-Body를 포함해서는 안 된다.

[Page 58]

응답은 다음의 헤더 필드를 포함하고 있어야 한다.

- 날짜
- ETag 및/또는 Content-Location, 동일한 요구에 대한 200 응답 속에 헤더가 발송되었을 경우
- Expires, Cache-Control, 및/또는 Vary, 동일한 변이에 대한 이전 응답 속에 발송된 field-value가 상이할 경우

조건적 GET 이 강한 캐시 검증자(13.3.3 절 참조)를 사용했다면 응답은 다른 Entity-Header 를 포함해서는 안 된다. 그렇지 않으면(조건적 GET 이 약한 캐시 검증자를 사용할 때) 응답은 Entity-Header 을 포함해서는 안 된다. 이렇게 하여 캐시 된 Entity-Body 과 갱신된 헤더 사이의 불일치를 방지할 수 있다.

304 응답이 현재 캐시 되지 않은 엔터티를 표시할 때 캐시는 이 응답을 무시하고 조건 없이 요구를 반복해야 한다.

캐시가 수신한 304 응답을 캐시 엔트리의 갱신에 사용한다면 캐시는 응답이 가지고 있는 새로운 필드 값을 반영하기 위해 엔트리를 반드시 갱신해야 한다.

304 응답은 Message-Body 를 포함해서는 안되므로 항상 헤더 필드 다음의 첫 공백 라인으로 종료되어야 한다.

10.3.6 305 Use Proxy(프락시를 사용할 것)

요구된 자원을 Location 필드에 명시된 프락시를 통하여 접근해야만 한다. Location 필드가 프락시의 URL 을 제공한다. 수신측은 프락시를 통한 요구를 반복할 것으로 기대된다.

10.4 Client Error 4xx (클라이언트 에러 4xx)

상태 코드의 4xx 클래스는 클라이언트가 에러를 발생한 것처럼 보일 경우에 사용된다. HEAD 요구에 응답하는 경우를 제외하고는 서버는 임시적이건 영구적이건 에러 상황에 대한 설명을 포함한 엔터티를 포함해야 한다. 이러한 상태 코드는 모든 요구 method 에 적용할 수 있다. 사용자 에이전트는 사용자에게 포함된 엔터티를 표시해야 한다.

주의 : 클라이언트가 데이터를 발송한다면 TCP를 사용하는 서버 구현 방식은 서버가 입력 접속을 종료하기 전에 응답을 포함하고 있는 패킷 접수를 확인할 수 있도록 주의해야 한다. 클라이언트가 접속이 종료된 후에도 계속해서 데이터를 전송한다면 서버의 TCP 스택은 리셋 패킷을 클라이언트에게 발송할 것이다.

[Page 59]

이 리셋 패킷은 HTTP 애플리케이션이 읽거나 해석하기 전에 클라이언트가 확인한 입력 버퍼를 지운다.

10.4.1 400 Bad Request(잘못된 요구)

잘못된 형식 때문에 서버가 요구를 이해할 수 없다. 클라이언트는 변경 없이 요구를 반복해서는 안 된다.

10.4.2 401 Unauthorized (인증되지 않았음)

응답이 사용자 인증을 요구한다. 이 응답은 요구된 자원에 적용할 수 있는 설명 요구(challenge)를 포함하고 있는 WWW-Authenticate 헤더 필드(14.46 절)를 포함하고 있어야 한다. 클라이언트는 적절한 Authorization 헤더 필드(14.8 절)를 가지고 요구를 반복할 수 있다. 요구가 벌써 Authorization 증명서를 포함하고 있다면 401 응답은 해당 증명서에 대한 인증이 거절되었음을 표시한다. 401 응답이 이전 응답과 동일한 설명 요구를 포함하고 있고 사용자 에이전트가 한 번 이상 인증 획득을 시도했다면 해당 엔터티가 관련된 진단 정보를 포함하고 있기 때문에 사용자에게 응답에 표시된 엔터티를 표시해주야 한다. HTTP 접속 인증은 11 장에 설명되어 있다.

이 코드는 향후 사용을 위해 예약되었다.

10.4.4 403 Forbidden(금지되었음)

서버가 요구를 이해했으나 완료하는 것을 거절하고 있다. 인증은 적용되지 않으며 요구를 반복될 수 없다. 요구 method 가 HEAD 가 아니고 서버가 왜 요구가 완료되었는지 알리고 싶다면 엔터티 안에 거절한 이유를 기록해야 한다. 이 상태 코드는 서버가 요구가 거부 사유를 밝히기 원하지 않을 때나 다른 응답을 적용할 수 없을 때 일반적으로 사용된다.

10.4.5 404 Not Found(찾을 수 없음)

서버가 Request-URI 와 일치하는 것을 아무것도 발견하지 못했다. 이러한 상태가 잠정적인지 영구적인지 관한 아무런 표시도 주어지지 않는다.

[Page 60]

서버가 이 정보를 클라이언트에게 알리고 싶지 않을 경우 상태 코드 403(Forbidden)을 대신 사용할 수 있다. 내부적으로 환경을 설정할 수 있는 메커니즘을 통하여 이전의 자원을 영구적으로 사용할 수 없으며 전송 주소가 없다는 것을 알 수 있으면 410(Gone) 상태 코드를 사용한다.

10.4.6 405 Method Not Allowed(Method 를 사용할 수 없음)

Request-Line 에 명시된 method 를 Request-URI 로 확인할 수 있는 자원에서 사용할 수 없다. 응답은 요구된 자원에 사용할 수 있는 method 의 목록을 포함한 Allow 헤더를 포함해야 한다.

10.4.7 406 Not Acceptable(접수할 수 없음)

요구가 확인한 자원이 요구 메시지와 함께 발송된 Accept 헤더에 의해서 접수할 수 없는 내용 특징을 가지고 있는 응답 엔터티만을 생성할 수 있다.

HEAD 요구가 아닌 이상 응답은 사용자 또는 사용자 에이전트가 가장 적합한 것을 선택할 수 있는 자원 특징 및 위의 목록을 포함한 엔터티를 포함한다. 엔터티 포맷은 Content-Type 헤더 필드가 설정한 media type 에 의해 명시된다. 사용자 에이전트의 포맷 및 성능에 따라 가장 적합한 선택을 결정하는 것은 자동으로 수행될 수 있다. 그러나 이 규격은 그러한 자동 선택의 표준에 대하여 아무런 규정도 하지 않는다.

주의 : HTTP/1.1 서버는 요구 메시지와 함께 발송된 Accept 헤더에 의해서 접수할 수 없는 응답을 리턴할 수 있게 한다. 어떤 경우엔 이것이 406 응답을 발송하는 것보다 좋을 수도 있다. 사용자 에이전트는 도착하는 응답의 헤더를 검사하여 그것의 접수 여부를 결정하도록 추천한다. 응답을 접수할 수 없을 때 사용자

에이전트는 잠정적으로 더 이상의 데이터를 수신하지 말아야 하며 추가 행동을 취할 것인지 사용자에게 질의한다.

10.4.8 407 Proxy Authentication Required(프락시 인증 필요)

이 코드는 401(Unauthorized)과 유사하지만 클라이언트는 먼저 프락시에서 자기 자신을 인증해야 한다는 것을 표시한다. 프락시는 요구된 자원의 프락시에 적용할 수 있는 설명 요구를 포함하는 Proxy-Authenticate 헤더 필드(14.33 절)를 리턴해야 한다. 클라이언트는 적절한 Proxy-Authorization 헤더 필드(14.34 절)와 함께 요구를 반복해야 한다. HTTP 접속 인증 획득에 대해서는 11 장에 설명되어 있다.

[Page 61]

10.4.9 408 Request Timeout(요구 시간 초과)

클라이언트가 서버가 기다리도록 준비한 시간 내에 요구를 만들어 낼 수 없다. 클라이언트는 나중에 변경 없이 요구를 반복할 수 있다.

10.4.10 409 Conflict(충돌)

자원의 현재 상태와의 충돌 때문에 요구를 완료할 수 없다. 이 코드는 사용자가 충돌을 해결하고 요구를 재전송할 수 있을 것으로 기대할 수 있는 상황에서만 사용할 수 있다. 응답 본문은 사용자가 충돌의 원인을 인지할 수 있도록 충분한 정보를 포함해야 한다. 이상적으로는 응답 엔터티가 사용자 또는 사용자 에이전트가 문제를 해결할 수 있을 정도의 충분한 정보를 포함할 수 있을 것이다. 그러나 가능하지 않을 수도 있으며 필수 사항은 아니다.

충동은 PUT 요구에 대한 응답으로 발생할 가능성이 높다. 버전 관리를 사용하고 있고 PUT 요구를 하는 엔터티가 이전 요구(제 3 자)가 작성한 요구와 충돌되는 자원에 대한 변경 사항을 포함하고 있다면 서버는 409 응답을 사용하여 요구를 완료할 수 없음을 표시해야 한다. 이 경우 응답 엔터티는 응답 Content-Type 이 규정한 형식으로 두 버전 사이의 차이점 목록을 포함해야 한다.

10.4.11 410 Gone (내용물이 사라졌음)

요구된 자원이 서버에 더 이상 존재하지 않으며 전송 주소를 알 수 없다. 이 조건은 영구적인 것으로 간주해야 한다. 링크를 편집할 기능이 있는 클라이언트는 사용자 인증 후의 Request-URI 에 대한 참고는 삭제해야 한다. 서버가 그 조건이 영구적인지 여부를 알 수 없거나 결정할 시설이 없으면 상태 코드 401(Unauthorized)을 대신 사용해야 한다. 다르게 표시되지 않는 한 이 응답은 캐시할 수 있다.

410 응답은 주로 수신측에게 자원을 의도적으로 사용할 수 없게 하였고 서버의 소유주가 해당 자원에 대한 원격 링크를 제거하고자 한다는 것을 알림으로써 웹 유지 작업을 지원하기 위해 사용된다. 이러한 일은 제한된 시간, 선전용 서비스 및 서버의 사이트에서 더 이상 일하지 않는 개인에게 소속된 자원에서 공통적으로 발생할 수 있다. 영구적으로 사용할 수 없는 모든 자원을 "사라진" 것으로 표시하거나 특정 시간 동안 표시를 유지할 필요는 없다. - 이것은 서버 소유자의 판단에 달려 있다.

[Page 62]

10.4.12 411 Length Required(길이가 필요함)

서버가 규정된 Content-Length 없는 요구 접수를 거부하였다. 요구 메시지 내의 Message-Body 의 길이를 포함하는 유효한 Content-Length 헤더 필드를 추가한다면 클라이언트는 요구를 반복할 수 있다.

10.4.13 412 Precondition Failed(사전 조건 충족 실패)

하나 또는 그 이상의 Request-Header 필드에 기입된 사전 조건이 서버에서 테스트 했을 때 거짓으로 평가되었다. 이 응답 코드는 클라이언트가 현재 자원의 메타 정보에 사전 조건을 부여할 수 있게 하여 의도하지 않는 자원에 요구 method 를 적용하는 것을 방지한다.

10.4.14 413 Request Entity Too Large(요구 엔터티가 너무 큼)

요구 엔터티가 서버가 처리할 수 있거나 처리하려는 것보다 크기 때문에 서버가 요구 처리를 거부하였다. 서버는 클라이언트가 계속적으로 요구하는 것을 방지하기 위하여 연결을 종료한다. 조건이 잠정적이면 서버는 Retry-After 헤더 필드를 포함하여 조건이 잠정적이며 얼마 후에 클라이언트가 재시도할 것인지를 표시한다.

10.4.15 414 Request-URI Too Long(Request -URI 가 너무 김)

Request-URI 가 서버가 해석할 수 있는 것보다 크기 때문에 서버가 요구 처리를 거부하였다. 이처럼 드문 조건은 클라이언트가 부적절하게 질의 정보가 긴 POST 요구를 GET 요구로 변환했을 때, 클라이언트가 방향 재설정의 URL "블랙 홀"로 빠졌을 때(방향이 재설정된 URL 접두사가 자신의 접미사를 지칭할 때), Request-URI 를 읽거나 조작하는 고정-길이 버퍼를 사용하는 몇몇 서버에 존재하는 보안의 허점을 이용하려는 클라이언트로부터 서버가 공격을 받을 때만 발생하는 것 같다.

10.4.16 415 Unsupported Media Type(지원되지 않는 media type)

요구의 엔터티가 요구 받은 method 의 자원이 지원하지 않는 포맷으로 구성되어 있기 때문에 요구 처리를 거부하였다.

[Page 63]

10.5 Server Error 5xx(서버 에러 5xx)

숫자 "5"로 시작하는 응답 상태 코드는 서버가 에러를 발생시켰으며 요구를 처리할 능력이 없음을 인지한 경우를 표시한다. HEAD 요구에 응답하는 때를 제외하고는 서버는 에러 상황에 대한 설명 및 에러가 잠정적인지 영구적인지에 관한 상황 설명을 포함하는 엔터티를 포함해야 한다. 사용자 에이전트는 포함된 모든 엔터티를 사용자에게 표시하여야 한다. 이러한 응답 코드는 모든 요구 method 에 적용할 수 있다.

10.5.1 500 Internal Server Error(서버 내부 에러)

서버가 요구를 처리하지 못하도록 하는 예상치 못한 상황에 접했다.

10.5.2 501 Not Implemented(구현되지 않았음)

서버가 요구를 완료하는 데 필요한 기능을 지원하지 않는다. 이것은 서버가 요구 method 를 인지할 수 없고 어떠한 자원을 사용해도 지원할 수 없을 때 적절한 응답이다.

10.5.3 502 Bad Gateway(불량 게이트웨이)

게이트웨이나 프락시 역할을 수행하는 서버가 요구를 완료하려는 시도에서 접근한 업스트림(upstream) 서버로부터 유효하지 않은 응답을 수신했을 경우이다.

10.5.4 503 Service Unavailable(서비스를 사용할 수 없음)

서버가 현재 잠정적인 오버로딩(overloading)이나 서버의 유지 작업 때문에 요구를 처리할 수 없다. 이것의 의미는 이것이 잠정적인 상황이며 얼마 후에는 완화될 수 있다는 것이다. 알 수 있다면 지연 시간 길이를 Retry-After 헤더에 표시할 수 있다. 아무런 Retry-After 정보가 없으면 클라이언트는 500 응답을 처리하는 것처럼 응답을 처리해야 한다.

주의 : 503 상태 코드가 있다는 것이 서버가 오버로드 되었을 때 이것을 반드시 사용해야 된다는 것을 의미하지 않는다. 어떤 서버는 단순히 접속을 거부하고자 한다.

10.5.5 504 Gateway Timeout(게이트웨이 시간 초과)

게이트웨이나 프락시 역할을 수행하는 서버가 시간 내에 요구를 완료하려는 시도에서 접근한 업스트림(upstream) 서버로부터 응답을 수신하지 못했을 경우이다.

[Page 64]

10.5.6 505 HTTP Version Not Supported(지원되지 않는 HTTP 버전)

서버가 요구 메시지에서 사용된 HTTP 규약 버전을 지원하지 않거나 지원하기를 거부했다. 서버는 이 에러 메시지 이외에는 3.1 절에서 설명한 대로 클라이언트가 사용하는 동일한 주요 버전을 사용하여 요구를 완료할 의사나 능력이 없음을 표시한다. 응답은 왜 해당 버전이 지원되지 않으며 서버가 어떤 규약을 지원하는가를 설명하는 엔터티를 포함해야 한다.

11 접속 인증

HTTP 는 서버는 클라이언트의 요구를 시도하고 클라이언트는 인증 정보를 제공하는 단순한 Try-Response 인증 획득 메커니즘을 제공한다. 이것은 확장 가능하고 대소문자를 구별하지 않는 토큰을 사용하여 인증 scheme 을 확인한다. 이 scheme 뒤에는 이 scheme 을 통하여 인증을 획득하는 데 필요한 파라미터를 가지고 있는 콤마로 구분된 attribute-value 쌍의 목록이 뒤따른다.

```
auth-scheme          = token
auth-param           = token "=" quoted-string
```

원서버는 401(Unauthorized) 응답 메시지를 사용하여 사용자 에이전트의 인증을 시도한다. 이 응답은 요구된 자원에 적용할 수 있는 최소한 하나의 시도를 포함한 WWW-Authenticate 헤더 필드를 포함해야 한다.

```
challenge       = auth-scheme 1*SP realm *( "," auth-param )
```

```
realm           = "realm" "=" realm-value
realm-value     = quoted-string
```

인증을 시도하는 모든 인증 scheme 은 영역 속성(대소문자를 구별하지 않음)을 가지고 있어야 한다. 영역 속성(대소문자를 구별함)은 접속하려는 서버의 정형적 루트 URL (5.1.2 절 참조)과 결합하여 보호 구역(protection space)을 정의한다. 이 영역은 서버의 보호된 자원이 각각 자신의 인증 획득 scheme 및/또는 인증 데이터 베이스를 가지고 보호 구역 세트로 분할될 수 있도록 한다. 영역 값은 문자열이며 보통 원서버가 지정한다. 원서버는 인증 획득 scheme 에 한정된 추가적인 의미를 가질 수 있다.

서버에 자신의 인증을 얻고자 하는 사용자 에이전트는 - 대개의 경우 필수적인 사항은 아니지만 401 또는 411 응답을 수신한 후 -요구에 Authorization 헤더 필드를 포함하여 인증을 얻을 수 있다. Authorization 필드 값은 요구하고 있는 자원의 영역에 대한 사용자 에이전트의 인증 획득 정보를 포함하고 있는 인증 증명서로 구성되어 있다.

[Page 65]

```
credentials     = basic-credentials
                | auth-scheme #auth-param
```

사용자 에이전트에 의해 증명서가 자동적으로 적용될 수 있는 도메인(domain)은 보호 구역에 의하여 결정된다. 이전의 요구가 인가되었으면 인증 획득 scheme, 파라미터 및/또는 사용자의 선호에 따라 결정되는 기간 동안 해당 보호 구역 내에서는 동일한 증명서를 재사용할 수 있다. 인증 획득 scheme 에 의해 다르게 규정되지 않는 한 단일 보호 구역은 서버의 범위를 넘어서 확장될 수 없다.

요구서버가 요구와 함께 수신한 증명서를 접수하고 싶지 않으면 서버는 401(Unauthorized) 응답을 리턴해야 한다. 응답은 요구된 자원 및 거절 이유를 설명하는 엔터티에 적용할 수 있는 시도(재시도일 수도 있다)를 포함한 WWW-Authenticate 헤더 필드를 반드시 포함해야 한다.

HTTP 규약은 접속 인증 획득을 위해 이 단순한 Try-Response 메커니즘만을 애플리케이션이 사용하도록 제한하지는 않는다. 전송 수준 또는 메시지 내포화(encapsulation)를 통한 암호화 등과 같은 추가적인 메커니즘을 인증 획득 정보를 명시하는 추가적인 헤더 필드와 더불어 사용할 수 있다.

프락시는 반드시 사용자 에이전트 인증 획득에 관하여 완전히 투명해야 한다. 말하자면 프락시는 반드시 WWW-Authenticate 및 Authorization 헤더를 변경하지 않고 전송해야 하며 14.8 절에 있는 규칙에 따라야 한다.

HTTP/1.1 은 클라이언트가 인증 획득 정보를 Proxy-Authenticate 및 Proxy-Authorization 헤더를 통하여 프락시와 주고 받을 수 있도록 허용해야 한다.

11.1 기본 인증 scheme

"기본적(basic)" 인증 scheme 은 사용자 에이전트가 각 영역에서 사용자 ID 및 암호로서 자신의 인증을 획득해야 한다는 모델에 기초하고 있다. 영역 값은 동일 서버의 다른 영역과의 동일성이 비교될 수 있는 불투명한 문자열로 간주해야 한다. 서버는 Request-URI 의 보호 구역에서 사용자 ID 와 암호를 검증할 수 있을 때만 요구를 청할 것이다. 선택적인 인증 획득 파라미터는 없다.

[Page 66]

보호 구역 내의 URI 에 대한 허가되지 않는 요구를 수신하면 서버는 다음과 같은 시도로 응답할 수 있다.

```
WWW-Authenticate: Basic realm="WallyWorld"
```

여기서 "WallyWorld"는 서버가 지정한 문자열로 Request-URI 의 보호 구역을 확인해 준다.

인증 획득을 수신하기 위해서 클라이언트는 증명서 내의 base64 로 인코딩된 문자열 내에서 단일 콜론(":") 문자로 구분된 사용자 ID 와 암호를 발송한다.

```
basic-credentials       = "Basic" SP basic-cookie
basic-cookie            = <user-pass의 base64 [7] 인코딩, 라인 당 76 문자에 제한을 받지 않을 경우>
user-pass               = userid ":" password
userid                  = *<TEXT excluding ":">
password                = *TEXT
```

Userids 는 대소문자를 구별할 수도 있다.

사용자 에이전트가 userid "Aladdin" 과 암호 "open sesame"를 송신하고 싶다면 다음의 헤더 필드를 사용해야 할 것이다:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

기본 인증 획득과 관련된 보안에 대한 고려 사항은 15 장을 참고한다.

11.2 요약 인증 scheme

HTTP 의 요약 인증 scheme 은 RFC 2069 [32]에 명시되어 있다.

12 내용 협상(Content Negotiation)

대부분의 응답은 인간 사용자가 해석하는 정보를 포함한 엔터티를 포함하고 있다. 당연히 사용자에게 요구에 상응하는 "최상의 사용 가능" 엔터티를 제공하는 것이 바람직하다. 서버와 캐시에게는 불행하게도 모든 사용자가 무엇이 최상인가에 대한 동일한 선호 사항을 가지고 있는 것이 아니며 모든 사용자 에이전트가 평등하게 모든 엔터티 유형을 표시할 능력이 있는 것이 아니다. 이러한 이유로 HTTP 는 "내용 협상"을 위한 몇몇 메커니즘을 제공하고 있다.

[Page 67]

내용 협상이란 복수의 표현 방법을 사용할 수 있을 때 특정 응답에 대한 최상의 표현 방법을 선택하는 과정이다.

주의 : 대체하는 표시 방법이 동일한 media type이고 동일 유형의 다른 능력을 사용하거나 다른 언어로 되어 있을 수 있기 때문에 "포맷 협상" 이라고 부르지 않는다.

에러 응답을 포함하여 Entity-Body 를 가지고 있는 모든 응답은 협상의 대상이 될 수 있다.

HTTP 에서 사용할 수 있는 두 가지 종류의 내용 협상이 있다 - 서버가 주도하는 협상과 에이전트가 주도하는 협상. 이 두 종류의 협상은 직교하기 때문에 분리하여 사용할 수도 있고 결합하여 사용할 수도 있다. 투명한 협상으로 지칭되는 결합의 한 방법은 직후의 요구에 대하여 서버가 주도하는 협상을 제공하기 위해서 캐시가 원서버가 제공하는 에이전트 주도의 협상 정보를 사용할 때 발생한다.

12.1 서버가 주도하는 협상

응답에 대한 최상의 표현 방식 선택이 서버에 위치한 알고리즘에 의하여 이루어 질 때 이를 서버가 주도하는 협상이라 부른다. 선택은 사용 가능한 응답 표시 방법(변형할 수 있는 차원. 예를 들어 언어 Content-Codings 등), 요구 메시지의 특정 헤더 필드의 내용 또는 요구와 관련된 기타 정보(클라이언트의 네트워크 주소 같은 것)에 기초한다.

서버가 주도하는 협상은 사용 가능한 표현 방법 중에서 선택하는 알고리즘이 사용자 에이전트에게 설명하기가 어려울 때 또는 서버가 자신의 "최상의 예측"을 첫 응답에 뒤 따라서("최상의 예측"이 사용자에게 충분할 정도로 좋다면 계속되는 요구의 왕복 여행으로 인한 지연을 피하려는 희망에서) 클라이언트에게 발송할 때 유리하다. 서버의 예측을 향상시키기 위해 사용자 에이전트는 그러한 응답에 대한 자신의 선호를 표시하는 요구 헤더 필드(Accept, Accept-Language, Accept-Encoding 등)를 포함할 수도 있다.

서버가 주도하는 협상은 다음의 단점이 있다.

1. 서버가 정확하게 특정 사용자에게 "최상"이 무엇인지 결정한다는 것은 사용자 에이전트의 능력 및 응답의 의도된 사용 용도에 대한 완벽한 이해를 필요로 하기 때문에 불가능하다. (예를 들면 사용자가 그것을 화면에서 보기를 원하는가 아니면 종이에 인쇄하기를 원하는가?)

2. 사용자 에이전트에게 요청할 때마다 자신의 능력을 설명하도록 하는 것은 매우 비효율적(적의 퍼센트의 응답만이 복수의 표현 방법을 가지고 있다고 가정하면)이면서도 사용자의 프라이버시를 침해할 가능성이 있다.

[Page 68]

3. 원서버의 구현 방법 및 요구에 대한 응답을 생성하는 알고리즘을 복잡하게 만든다.

4. 복수의 사용자 요구에 대해 동일한 응답을 사용할 수 있는 보편적인 캐시의 능력을 제한할 수 있다.

HTTP/1.1 은 사용자 에이전트의 능력 및 사용자의 선호를 기술하여 서버가 주도하는 협상을 가능하게 하는 다음의 Request-Header 필드를 포함하고 있다. - Accept (14.1 절), Accept-Charset (14.2 절), Accept-Encoding (14.3 절), Accept-Language (14.4 절) 및 User-Agent (14.42 절). 그러나 원서버는 이러한 차원에 제한 받지 않고 Request-Header 필드 이외의 정보 또는 이 규격이 규정하지 않은 확장 헤더 필드를 포함하는 요구의 어떠한 측면에 따라 변형될 수 있다. HTTP/1.1 원서버는 서버가 주도하는 협상에 기초한 캐시할 수 있는 모든 응답에서 반드시 적절한 Vary 헤더 필드(14.43 절)를 포함해야 한다. Vary 헤더 필드는 응답이 변형될 수 있는 차원을 설명한다. (예를 들어 원서버가 복수의 표현 방식으로부터 "최상의 예측"을 끄집어 낼 수 있는 차원)

HTTP/1.1 공공 캐시는 응답에 포함되어 있는 Vary 헤더 필드를 인지해야 하고 캐시와 내용 협상의 상호 작용을 기술하고 있는 13.6 절에 설명된 필요 조건을 충족해야 한다.

12.2 에이전트가 주도하는 협상

에이전트가 주도하는 협상에서 응답에 대한 최상의 표현 방식의 선택은 원서버로부터 첫 응답을 수신한 다음 사용자 에이전트가 수행한다. 선택은 각각의 표현 방식은 자신의 URI 에 의하여 식별하면서 헤더 필드(이 규격은 부록 19.6.2.1 에 기술한 대로 필드 이름 Alternates 를 예약했다.) 내에 포함되어 있는 사용 가능한 표현 방식의 목록이나 첫 응답의 Entity-Body 에 기초한다. 표현 방식 선택은 자동적으로 수행(사용자 에이전트가 그렇게 할 능력이 있다면)될 수도 있고 사용자가 생성된(하이퍼텍스트일 수 있다) 메뉴에서 선택할 수도 있다.

에이전트가 주도하는 협상은 응답이 일반적으로 사용하는 차원(유형, 언어 또는 인코딩)에 따라 변할 때, 원서버가 응답을 관찰하여 사용자 에이전트의 능력을 결정할 수 없을 때 또는 서버의 부하를 분산하고 네트워크 사용을 감소시키기 위해 공공 캐시가 사용되었을 때 유리하다.

[Page 69]

에이전트가 주도하는 협상은 최적의 대체 표시 방법을 얻기 위해서 두 번째 요구가 필요하다는 단점이 있다. 이 두 번째 요구는 캐시가 사용될 때만 효과적이다. 또한 이 규격은 자동적 선택을 지원하는 어떠한 메커니즘도 규정하지 않는다. 그러나 이 규격은 또한 이러한 메커니즘이 확장으로서 개발되고 HTTP/1.1 내에서 사용되는 것을 금지하지는 않는다.

HTTP/1.1 은 서버가 주도하는 협상을 이용하는 변화된 응답을 제공할 수 없거나 제공하려 하지 않을 때 에이전트가 주도하는 협상이 가능하도록 300 (Multiple Choices) 및 406 (Not Acceptable) 상태 코드를 규정한다.

12.3 투명한 협상(Transparent Negotiation)

투명한 협상은 서버가 주도하는 협상과 에이전트가 주도하는 협상의 복합체이다. 캐시가 응답의 사용 가능한 표현 방식 목록 형태로 제공되고(에이전트가 주도하는 협상처럼) 캐시가 변이의 차원을 완전히 이해했으면 해당 자원에 대한 계속적인 요구에 대하여 원서버를 대신하여 캐시는 서버가 주도하는 협상을 수행할 수 있게 된다.

투명한 협상은 그렇지 않다면 원서버가 수행해야 하는 협상 작업을 분산할 수 있고 캐시가 정확하게 올바른 응답을 예측할 수 있을 때 에이전트가 주도하는 협상의 두 번째 요구 지연을 제거할 수 있다는 장점을 가지고 있다.

이 규격은 투명한 협상에 대한 어떠한 메커니즘도 규정하지 않는다. 그러나 이 규격은 또한 이러한 메커니즘이 확장으로서 개발되고 HTTP/1.1 내에서 사용되는 것을 금지하지는 않는다. 투명한 협상을 수행하는 HTTP/1.1 캐시는 HTTP/1.1 과의 올바른 상호 작용을 확보하기 위하여 캐시할 수 있다면 반드시 응답(변이의 차원을 정의)에 Vary 헤더 필드를 포함해야 한다. 원서버가 제공하는 에이전트가 주도하는 협상에 대한 정보는 투명하게 협상된 응답에 포함되어야 한다.

13 HTTP 에서의 캐시

HTTP 는 전형적으로 응답 캐시를 사용하여 성능을 향상시킬 수 있는 분산 정보 시스템에서 사용된다. HTTP/1.1 규약은 캐시 작업을 가능한 한 잘 수행하기 위한 몇몇 요소를 포함한다. 이러한 요소는 규약의 다른 측면에서 제외할 수 없는 것이기 때문에 또한 서로 상호 작용을 하기 때문에 method, 헤더, 응답 코드 등에 대한 자세한 설명과는 별도로 HTTP 의 기본 캐시 디자인을 설명하는 것이 유용하다.

[Page 70]

성능을 상당히 개선하지 못한다면 캐시는 쓸모없는 것이 될 것이다. HTTP/1.1 캐시의 목적은 많은 경우에 요구를 발송할 필요를 제거하고 또 다른 많은 경우에 완전한 응답을 발송할 필요를 제거하는 것이다. 전자는 많은 운영에서 네트워크의 왕복 여행 숫자를 줄여 준다. 우리는 이 목적을 위해서 "만기일" 메커니즘을 사용한다. (13.2 절 참조). 후자는 네트워크 대역폭 요구를 감소시켜 준다. 우리는 이 목적을 위해서는 "검증" 메커니즘을 사용한다.(13.3 절 참조)

성능, 가용성 및 단절된 운영에 대한 필요 조건이 우리에게 의미 투명성(semantic transparency)의 목적을 완화할 수 있도록 요구한다. HTTP/1.1 규약은 원서버, 캐시, 클라이언트가 필요하다면 분명하게 투명성을 감소할 수 있도록 한다. 그러나 비-투명적 작업은 비 전문가 사용자에게 혼선을 줄 수 있고, 특정 서버 애플리케이션과 호환되지 않을 수 있기 때문에(제품 주문을 위한 애플리케이션처럼) 규약은 투명성을 완화시킬 것을 요구한다.

- 클라이언트나 원서버에 의해 완화되었을 때는 분명하게 규약 수준의 요구에 의해서만
- 캐시나 클라이언트에 의해 완화되었을 때는 사용자에게 분명한 경고를 줌으로써

[Page 71]

따라서 HTTP/1.1 규약은 다음의 중요한 요소를 제공한다.

1. 모든 참가자가 요구할 때 완전한 의미 투명성을 제공하는 규약 기능
2. 원서버나 사용자 에이전트가 비 투명적 작업을 분명하게 요청하고 제어할 수 있도록 하는 규약 기능
3. 캐시가 요구한 의미 투명성에 대한 근접 요구를 유지할 수 없는 응답에 경고를 첨부하는 규약 기능

기본 규칙은 클라이언트가 잠재적인 의미 투명성의 완화를 감지할 수 있어야 한다는 것이다.

주의 : 서버, 캐시 또는 클라이언트 구현자는 이 규격에서 분명하게 토의되지 않은 디자인을 결정해야 하는 문제에 직면하게 된다. 결정 사항이 의미 투명성에 영향을 미치게 되면 구현자는 주의 깊고 완전한 분석이 투명성을 어김으로써 상당한 혜택을 주는 것으로 나타나지 않는 한 투명성을 유지하는 데 치우쳐야 한다.

13.1.1 캐시의 정확성

정확한 캐시는 반드시 아래의 조건 중 하나를 만족하며 요구에 적합한(13.2.5, 13.2.6 및 13.12 절 참조) 보유하고 있는 캐시 중 가장 최근의 응답으로 요구에 답해야 한다.

1. 원서버가 원서버를 사용하여 응답을 재검증한 후 되돌려 주었을 것과 같은 것인지 점검하였다.(13.3 절)

2. 충분히 신선하다.(13.2 절 참조). 기본적인 경우 이것은 클라이언트, 서버 및 캐시의 최소한도의 신선도 필요 조건을 만족한다는 것을 의미한다.(14.9 절 참조). 원서버가 그렇게 명시하였으면 그것은 원서버의 신선도 필요 조건일 뿐이다.

3. 클라이언트 또는 서버의 신선도 요구가 위반되었을 경우(13.1.5 또는 14.45 절 참조) 경고를 포함하고 있다.

4. 적절한 304 (Not Modified), 305 (Use Proxy), 또는 error (4xx or 5xx) 응답 메시지이다.

캐시가 원서버와 통신할 수 없다면 정확한 캐시는 위처럼 응답해야 한다.(캐시가 정확한 응답을 할 수 있다면). 그렇지 못하면 캐시는 통신 실패가 있었음을 알리는 에러 또는 경고를 리턴해야 한다.

[Page 72]
캐시가 보통 클라이언트로 전달하게 되는 응답을 수신하고(전체 응답 혹은 304(Not Modified)응답) 수신한 응답이 더 이상 신선하지 않으면 캐시는 응답을 새로운 Warning(그러나 기존의 Warning 헤더는 제거하지 않고)을 추가하지 않고 요구한 클라이언트로 전달한다. 캐시는 응답이 그 동안 낡은 것이 되기 때문에 검증하려 시도해서는 안 된다. 시도하면 무한 루프로 빠지게 될 것이다. Warning 없는 낡은 응답을 수신한 사용자 에이전트는 사용자에게 경고 표시를 할 수 있다.

13.1.2 경고

캐시가 처음이 아니거나 "충분히 신선하지"(13.1.1 절의 조건 2 의 의미) 않은 응답을 리턴할 때는 Warning Response-Header 을 이용하여 이러한 취지로 경고를 부착하여야 한다. 이 경고는 클라이언트가 적절한 조치를 취할 수 있도록 한다.

경고는 캐시와 관련되거나 별도의 다른 목적을 위해 사용할 수 있다. 에러 상태 코드 대신 경고를 사용하여 이 응답을 진짜 실패와 구별할 수 있게 한다.

경고는 응답의 투명성을 결코 약하게 하지 않기 때문에 언제나 캐시할 수 있다. 이는 경고를 HTTP/1.0 에게 위험 없이 전달할 수 있음을 의미한다. 이러한 캐시는 단순히 경고를 응답의 Entity-Head 헤드와 함께 전달한다.

경고는 0 부터 99 까지의 숫자로 지정한다. 이 규격은 코드 숫자와 현재 각각에 지정된 경고의 의미를 규정하여 클라이언트 또는 캐시가 몇몇 경우(모든 경우는 아니다.) 자동화된 조치를 취할 수 있게 한다.

경고는 또한 경고문을 수반한다. 경고문은 적당한 자연 언어(아마도 클라이언트의 Accept 헤더에 기초하여)로 작성될 수 있으며 선택적으로 사용된 문자 집합 표시를 포함할 수 있다.

동일한 코드 번호의 복수의 경고를 포함하는 복수의 경고가 응답에 부착될 수 있다.(원서버나 캐시에 의해서). 예를 들어 서버는 동일한 경고를 영어와 바스크어로 된 경고문으로 제공할 수 있다.

복수의 경고가 응답에 부착되었을 때 이 모두를 사용자에게 보여 주는 것은 실질적이지 않거나 비합리적일 수 있다. 이 HTTP 버전은 어떤 경고를 어떤 순서에 입각하여 표시할 것인지 결정하는 엄격한 우선권 규칙은 명시하지 않지만 약간의 발견법(heuristics)을 제안하기는 한다.

[Page 73]
Warning 헤더와 현재 정의된 경고는 14.45 절에 기술되어 있다.

13.1.3 Cache-Control 메커니즘

HTTP/1.1 의 기본적인 캐시 메커니즘(서버가 명시한 유효 시간 및 검증자)은 캐시에 내재된 지시를 하는 것이다. 어떤 경우에 서버나 클라이언트는 내재된 지시자를 HTTP 캐시에게 제공할 필요가 있다. 우리는 Cache-Control 헤더를 이 목적으로 사용한다.

Cache-Control 헤더는 클라이언트나 서버가 요구나 응답의 다양한 지시자를 전달할 수 있도록 한다. 이 지시자는 대개의 경우 기본 캐시 알고리즘을 무시한다. 보편적인 원칙으로 만약 헤더 값 사이에 분명한 충돌이 있으면 가장 엄격한 해석을 사용해야 한다.(말하자면 의미 투명성을 가장 잘 보존할 수 있는 것). 그러나 어떤 경우에는 Cache-Control 지시자가 분명하게 의미 투명성의 근사치를 약화시키는 것으로

명시할 수 있다.(예를 들어 "max-stale" 또는 "public" Cache-Control 지시자는 14.9 절에 자세하게 기술되어 있다.

13.1.4 명백한 사용자 에이전트 경고

많은 사용자 에이전트는 사용자가 기본적인 캐시 메커니즘을 무시할 수 있도록 한다. 예를 들어 사용자 에이전트는 사용자가 캐시 된 엔터티(분명하게 낡은 캐시까지도)를 결코 검증하지 말도록 명시하는 것을 허락한다. 또는 사용자 에이전트가 습관적으로 "Cache-Control: max-stale=3600" 을 모든 요구 내에 첨가할 수도 있다. 사용자는 투명하지 않는 방식이나 비정상적으로 비효과적인 캐시를 초래하는 방식을 확실하게 요구해야 한다.

사용자가 기본적인 캐시 메커니즘을 무시했다면 이것이 서버의 투명성 필요 조건을 만족시켜 주지 못할 정보 표시를 초래하게 될 때 사용자 에이전트는 항상 사용자에게 분명하게 표시해 주어야 한다. 보통 규약은 사용자 에이전트가 응답이 낡은 것인지 아닌지 여부를 결정할 수 있도록 하기 때문에 실제로 발생했을 경우에는 이것을 화면에 표시할 필요가 있다. 이 표시는 반드시 대화 박스일 필요는 없고 아이콘(예를 들어 부패된 생선 그림)이나 다른 시각적 표시자일 수도 있다.

사용자가 캐시의 효과성을 비정상적으로 감소시키는 방식으로 캐시 메커니즘을 무시했다면 사용자 에이전트는 계속해서 화면에 이를 표시(예를 들어 불타는 지폐 그림)하여 사용자가 부주의하게 과도한 자원을 낭비하거나 지나치게 기다리지 않도록 해야 한다.

[Page 74]

13.1.5 규칙 및 경고의 예외 사항

어떤 경우에는 캐시 운영자는 클라이언트가 요구하지 않았어도 낡은 응답을 리턴하도록 환경을 설정할 수 있다. 이러한 결정은 가볍게 해서는 안되지만 가용성이나 성능 특히 캐시가 원서버와 약하게 연결되어 있을 때는 필요할 수도 있다. 캐시가 낡은 응답을 리턴할 때마다 이러한 상태를 표시해야 한다.(Warning 헤더를 이용하여). 이렇게 하여 클라이언트 소프트웨어가 사용자에게 문제가 발생할 소지가 있음을 경고할 수 있도록 한다.

또한 사용자 에이전트가 처음 또는 새로운 응답을 얻는 조치를 취할 수 있도록 한다. 이러한 이유로 캐시는 클라이언트가 분명하게 처음 또는 새로운 캐시를 요구하면 기술적인 이유나 정책적인 이유에 따르는 것이 불가능하지 않는 한 낡은 응답을 리턴해서는 안 되는 것이다.

13.1.6 클라이언트가 제어하는 행태

원서버(최소한 응답의 시간 경과에 공헌한 것을 감안하여 중간 캐시)가 만기일 정보의 주요 소스일 때 어떤 경우에는 클라이언트가 검증 없이 캐시 된 응답을 리턴할 것인지 여부에 대한 결정을 제어할 필요가 있다. 클라이언트는 몇몇 Cache-Control 헤더 지시자를 사용하여 이를 수행한다.

클라이언트의 요구는 검증되지 않은 응답을 접수할 수 있는 최대한의 경과시간을 명시한다. 이 값을 제로로 설정하면 캐시가 모든 응답을 재검증하도록 한다. 클라이언트는 응답이 만료되기 전에 남아 있는 최소한의 시간을 명시한다. 이 두 선택 사항 모두 캐시의 행태에 대한 제한을 증대시키기 때문에 캐시의 의미 투명성 근사치를 추가로 완화시켜 주지는 못한다. 클라이언트는 또한 최대한 어떠한 수준까지의 낡은 응답을 접수할 것임을 명시할 수 있다. 이것은 캐시에 대한 제한을 완화시켜 주기 때문에 원서버가 명시한

의미 투명성에 대한 제한 사항을 위반할 수도 있지만 접속이 단절된 상태에서의 운용, 불량한 접속에 직면하여 높은 가용성을 지원하기 위해 필요할 수도 있다.

13.2 만기일 모델

13.2.1 서버가 명시한 만기일

HTTP 캐시는 원서버로 요구를 발송하는 것을 완전히 피할 수 있을 때 최상으로 작동한다. 요구를 피하는 주요 메커니즘은 원서버가 분명하게 해당 응답이 계속되는 요구를 만족시킬 수 있다는 것을 표시하는 미래의 만료 시간을 제공하는 것이다. 다른 말로 표현하면 캐시가 먼저 서버와 접촉하지 않고도 새로운 응답을 리턴할 수 있다는 것이다.

[Page 75]

우리가 기대하는 것은 서버가 만기일이 도착 전에 엔터티가 의미상으로 중대하게 변화하지 않을 것이라는 믿음으로 미래의 분명한 만료 시간을 부여하는 것이다. 이렇게 하면 서버의 유효 시간이 신중하게 선택된 한 대개의 경우 의미 투명성을 보존한다.

유효일 메커니즘은 캐시에서 얻은 응답에만 적용되며 요구한 클라이언트에게 직접적으로 전달되는 첫 응답에는 적용되지 않는다.

원서버가 모든 요구를 검증하기 위해 의미상으로 투명한 캐시를 요구한다면 과거 시점의 유효 시간을 부여할 수도 있다. 이는 응답이 항상 낡은 것이기 때문에 계속되는 요구에 이것을 사용하기 위해서는 반드시 먼저 검증을 해야 한다는 것을 의미한다. 검증을 강제로 요구하는 제한적인 방법에 관한 추가 정보는 14.9.4 절을 참조한다.

원서버가 HTTP/1.1 캐시가 모든 요구를 검증하도록 하려면 어떤 방식으로 환경이 설정되었든 "must-revalidate" Cache-Control 지시자(14.9 절 참조)를 사용해야 한다.

서버는 Expires 헤더 또는 Cache-Control 헤더의 max-age 지시자를 사용하여 분명하게 유효 시간을 명시한다.

유효 시간은 사용자 에이전트가 자원을 화면에 표시하거나 갱신하도록 만드는 데 사용할 수 없다. 이 의미는 캐시 메커니즘에만 적용되며 이러한 메커니즘은 해당 자원에 대한 새로운 요구가 시작되었을 때 자원의 유효일 상태만을 점검할 필요가 있다. 캐시와 history 메커니즘의 차이점에 대한 설명은 13.13 절을 참고한다.

13.2.2 스스로 유효일을 찾음(Heuristic Expiration)

원서버가 언제나 명백한 유효 시간을 제공하는 것이 아니므로 HTTP 캐시는 전형적으로 그럴듯한 유효 시간을 짐작하기 위해 다른 헤더 값(Last-Modified 시간과 같은)을 사용하는 알고리즘을 활용하는 발견법(heuristic) 유효 시간을 할당한다. HTTP/1.1 규격은 상세한 알고리즘을 제공하지는 않지만 결과에 대한 최악의 경우 제한 사항을 부과하고 있다. 발견법에 의한 유효 시간은 의미 투명성 때문에 정확하지 않을 수도 있기 때문에 조심해서 사용해야 하며 우리는 원서버가 가능한 한 분명한 유효 시간을 제공하도록 권고한다.

[Page 76]

13.2.3 경과 시간 계산(Age Calculations)

캐시 된 엔트리가 새로운 것인지 확인하기 위해서 캐시는 캐시의 경과 시간이 신선한 기간(freshness lifetime)을 초과했는지 알 필요가 있다. 신선한 기간을 계산하는 방법에 대해서는 13.2.4 절에서 토의하고 이 절에서는 응답이나 캐시 엔트리의 경과 시간을 계산하는 방법을 설명한다.

이 설명에서 우리는 "지금" 이라는 용어를 "계산을 수행하는 호스트 시계의 현재 값"을 의미하는 것으로 사용한다. HTTP 를 사용하는 호스트, 특히 원서버와 캐시를 운영하는 서버는 NTP [28] 나 유사 규약을 사용하여 자신의 시계를 국제적으로 정확한 시간 기준과 동시화해야 한다.

HTTP/1.1 은 원서버가 모든 응답에 응답이 생성된 시간을 알려 주는 Date 헤더를 포함하여 발송할 것을 요구한다는 점에 주의한다. 우리는 "date_value"라는 용어를 Date 헤더의 값을 사칙연산에 적합한 형식으로 표시하는 것으로 사용한다.

HTTP/1.1 은 Age Response-Header 을 사용하여 캐시 사이의 경과 시간 정보를 전달한다. Age 헤더 값은 응답이 원서버에서 생성된 이후의 발송자의 예측이다. 원서버가 검증한 캐시 된 응답의 경우 Age 값은 원래의 응답이 아닌 재검증 시간에 기초한다.

핵심적으로 Age 값은 원서버로부터의 경로를 따라서 응답이 각 캐시에 보관되어 있던 시간의 총합 및 네트워크 경로를 따라서 이동되었던 시간의 양이다.

우리는 "date_value"라는 용어를 Date 헤더의 값을 사칙연산에 적합한 형식으로 표시하는 것으로 사용한다.

응답의 경과 시간은 완전히 독립적인 두 가지 방법으로 계산할 수 있다.

1. 현재 마이너스 date_value, 지역 시계가 원서버 시계와 비교적 잘 동시화 되어 있을 경우. 결과 값이 마이너스이면 결과를 제로로 대체한다.
2. age_value, 응답 경로에 따른 모든 캐시가 HTTP/1.1을 구현할 경우.

응답을 수신하였을 때 응답의 경과 시간을 계산하기 위한 두 가지의 독립적인 방법을 가지고 있다고 가정하면 우리는 그것들을 다음처럼 결합할 수 있다.

```
corrected_received_age = max(now - date_value, age_value)
```

우리가 거의 동시화 된 시계와 모든 HTTP/1.1 경로를 가지고 있다면 신뢰할 만한(조심스러운) 결과를 얻을 수 있다.

[Page 77]

이러한 수정은 경로를 따라 각각의 HTTP/1.1 캐시에 적용되기 때문에 경로에 HTTP/1.0 캐시가 있으면 수정된 수신 경과 시간을 수신하는 캐시의 시계가 거의 동시화 되어 있는 한 계산할 수 있다. 양편 모두의 시계가 동시화 될 필요는 없으며(바람직하기는 하지만) 시계를 명백하게 동시화(synchronization) 하는 절차는 없다.

네트워크가 부과한 지연때문에 서버가 응답을 생성한 시간 또는 다음의 외부 방향 캐시나 클라이언트가 수신한 시간 이후로 중요한 중간 시간이 경과했을 수도 있다. 수정하지 않으면 이러한 지연은 부적절하게 짧은 경과시간을 초래할 수도 있다.

리턴 된 Age 값을 생산하는 요구는 반드시 해당 Age 값이 생산되기 이전에 시작되어야 하기 때문에 요구가 시작된 시간을 기록함으로써 네트워크가 부과한 지연 시간을 결정할 수 있다. 따라서 Age 값이 수신되면 반드시 응답이 수신된 시간이 아닌 요구가 시작된 시간과 상대적으로 해석해야 한다. 이 알고리즘은 얼마나 많은 지연 시간이 발생했는가에 관계없이 조심스러운 행태를 낳게 된다. 따라서 우리는 다음과 같이 계산한다.

```
corrected_initial_age = corrected_received_age + (now - request_time)
```

여기서 "request_time"은 이 응답을 이끌어 낸 응답이 발송된 시간(지역 시계의 시간에 따라)이다.
캐시가 응답을 수신했을 때 경과 시간 계산 알고리즘의 요약은 다음과 같다.

```
        /*
        * age_value
        *       는 Age의 값이다: 캐시가 이 응답과 더불어 수신한 헤더
        * date_value
        *       는 원서버의 Date 값이다: 헤더
        * request_time
        *       는 이 캐시 된 응답을 만들어 낸 요구를 캐시가 요구한 (지역)시간이다.
        * response_time
        *       는 캐시가 응답을 수신한 (지역)시간이다.
        * now
        *       는 현재 (지역) 시간이다.
        */

apparent_age            = max(0, response_time - date_value);

                                                [Page 78]
corrected_received_age  = max(apparent_age, age_value);
response_delay          = response_time - request_time;
corrected_initial_age   = corrected_received_age + response_delay;
resident_time           = now - response_time;
current_age             = corrected_initial_age + resident_time;
```

캐시가 응답을 송신하였을 때 캐시는 응답이 지역적으로 보관되었던 시간의 양을 corrected_initial_age 에 추가하여야 한다. 그런 다음 캐시는 이 합산된 경과 시간을 Age 헤더를 이용하여 다음 수신측 캐시로 전달해야 한다.

클라이언트는 응답이 처음이라는 것을 신뢰성 있게 말할 수 없음을 주의해야 한다. 그러나 Age 헤더가 있다는 것은 응답이 분명 처음은 아니라는 것을 표시한다. 또한 응답의 Date 가 클라이언트의 지역 요구 시간보다 앞설 경우 해당 응답은 아마도 첫 응답이 아닐 것이다.(심각할 정도로 시계의 시간이 빗나가지 않았을 때)

13.2.4 유효일 계산

응답이 신선한지 낡은 것인지 결정하기 위해 우리는 경과된 시간과 신선한 기간(freshness lifetime)을 비교할 필요가 있다. 경과된 시간은 13.2.3 절에서 설명한 대로 계산한다. 이 절은 신선한 기간을 계산하는 방법을 기술하고 응답이 만료되었는지 결정한다. 아래의 설명에서 값은 사칙연산 수행에 적합한 어떠한 형식으로도 표현될 수 있다.우리는 "expires_value" 라는 용어를 Expires 헤더 값을 표시하는 것으로 사용한다. 또한 우리는 "max_age_value" 라는 용어를 응답(14.10 절 참조)에 있는

Cache-Control 헤더의 max-age 지시자가 가지고 있는 적절한 초 값을 의미하는 것으로 사용한다. max-age 지시자는 Expires 보다 우선권을 갖는다.

따라서 응답에 max-age 가 있으면 계산은 간단히 다음과 같다.

```
freshness_lifetime      = max_age_value
```

그렇지 않고 Expires 가 응답에 있으면 계산은:

```
freshness_lifetime      = expires_value - date_value
```

모둔 정보가 원서버에서 오기 때문에 두 계산 방법 모두 정확하지 않는 시계에 취약하지 않다는 점에 유의한다.

Expires 및 Cache-Control: max-age 모두가 응답에 없으면 해당 응답은 캐시에 대한 제약 사항을 포함하지 않는다.

[Page 79]

이 때 캐시는 발견법 사용하여 신선한 기간을 산출한다. 값이 24 시간보다 클 때는 캐시는 Warning 13 을, 이 경고가 추가되지 않았다면 경과 시간이 24 시간 이상인, 응답에 첨가해야 한다. 또한 응답이 Last-Modified 시간을 포함하고 있다면 발견법에 의한 유효일 값은 그 시간 이후의 중간 시간의 한 부분보다 커서는 안 된다. 이 부분의 전형적인 설정값은 10%가 될 것이다.

응답이 만료되었는지 결정하기 위한 계산은 굉장히 단순하다:

```
response_is_fresh       = (freshness_lifetime > current_age)
```

13.2.5 유효일 값을 명확하게 하기

유효일 값이 낙천적으로 부여되기 때문에 두 캐시가 내용이 다른 동일한 자원에 대한 신선도 값을 포함할 수 있다.

조회 작업을 수행하는 클라이언트가 요구에 대해 이미 자체 캐시에서 신선했던 처음이 아닌 응답을 수신하면 기존 캐시 엔트리의 Date 헤더는 새로운 응답의 Date 보다 새로운 것이다. 이 때 클라이언트는 응답을 무시할 수 있다. 만약 그렇다면 클라이언트는 원서버가 점검하도록 강요하기 위해 "Cache-Control: max-age=0" 지시자(14.9 절 참조)를 포함한 요구를 다시 시도할 수 있다.

캐시가 상이한 검증자의 동일한 표현을 위한 두개의 새로운 응답을 가지고 있으면 가장 최근의 Date 헤더를 가지고 있는 것을 사용해야 한다. 이 상황은 캐시가 다른 캐시로부터 응답을 모을 수(pooling) 있고 클라이언트가 분명히 새로운 캐시 엔트리를 갱신 또는 재검증하도록 요구할 수 있기 때문에 발생할 수 있다.

13.2.6 복수의 응답을 명확하게 하기

클라이언트가 복수의 경로를 통해 응답을 수신할 수 있기 때문에(어떤 응답은 한 캐시 세트를 통해서 오고 다른 응답은 다른 캐시 세트를 통해 올 수 있기 때문에) 클라이언트는 응답을 원서버가 발송한 순서와 다르게 수신할 수도 있다. 이전의 응답이 아직도 분명 새롭다 할지라도 우리는 클라이언트가 가장 최근에 생성된 응답을 사용하기 바란다.

나중의 응답이 의도적으로 더 빠른 유효 시간을 가지고 있을 수 있기 때문에 엔트리 태그나 유효일 값 모두 응답의 순서에 대하여 영향을 미칠 수 없다. 그러나 HTTP/1.1 규격은 모든 응답에 Date 헤더를 전송해야 하며 Date 값은 1 초 단위로 순서가 매겨져야 한다.

[Page 80]

클라이언트가 캐시 엔트리의 재검증을 시도할 때, 수신하는 응답이 기존 엔트리의 Date 헤더보다 더 오래된 것처럼 보이는 Date 헤더를 포함할 때 클라이언트는 요구를 무조건적으로 반복해야 하며 다음을 포함해야 한다.

Cache-Control: max-age=0

중간 캐시에게 자신의 사본을 원서버와 직접적으로 검증하도록 강요한다.

Cache-Control: no-cache

중간 캐시에게 원서버에서 새로운 사본을 얻도록 강요한다.

Date 값이 동등하면 클라이언트는 양쪽 응답을 모두 사용할 수 있다. (또는 충분히 신중하다면 새로운 응답을 요구할 수 있다.) 서버는 두 응답의 유효일이 중첩된다면 클라이언트가 동일한 시간에 생성된 응답 중에서 하나를 과감하게 선택할 수 있다고 믿어서는 절대 안 된다.

13.3 검증 모델

캐시가 클라이언트 요구에 대한 응답으로 사용하고자 하는 낡은 엔트리를 가지고 있을 때 캐시 된 엔트리를 아직도 사용할 수 있는지 알아보기 위해서 클라이언트는 먼저 원서버(새로운 응답을 가진 중간 캐시일 수도 있다.)를 점검해야 한다. 우리는 이것을 캐시 엔트리를 "검증한다"고 한다. 우리는 캐시 된 엔트리의 상태가 좋을 때 전체 응답을 재전송해야 하는 오버헤드를 갖길 원하지 않기 때문에, 또한 캐시 된 엔트리가 유효하지 않을 때 왕복 여행을 해야 하는 오버헤드를 갖기 원하지 않기 때문에 HTTP/1.1 규약은 조건적인 method 사용을 지원한다.

조건적 method 를 지원하는 핵심 규약 기능은 "캐시 검증자"에 관련된 것들이다. 원서버가 완전한 응답을 생성할 때 서버는 일종의 검증자를 부착하며 이것은 캐시 엔트리와 함께 보관된다. 클라이언트(사용자 에이전트 또는 프락시 캐시)가 캐시 엔트리를 가지고 있는 자원에 대한 조건적인 요구를 할 때 클라이언트는 요구에 관련된 검증자를 포함한다.

그런 다음 서버는 해당 검증자를 엔터티의 현재 검증자에 비추어 점검한다. 서로 일치하면 서버는 특수 상태 코드(대개의 경우 304 (Not Modified))와 Entity-Body 가 없는 것으로 응답한다. 따라서 우리는 검증자가 일치하면 전체 응답 전송을 피할 수 있고 일치하지 않으면 추가적인 왕복 여행을 피할 수 있다.

[Page 81]

주의 : 검증자가 일치하는지 여부를 결정하는 데 사용되는 비교 기능은 13.3.3 절에 규정되어 있다.

HTTP/1.1 에서 조건적인 요구는 요구가 method(대개 GET)를 조건적으로 변경시키는 특수 헤더(검증자를 포함하는)를 가지고 있는 경우를 제외하고는 동일한 자원에 대한 정상적인 요구와 정확하게 동일한 것처럼 보인다. 규약은 긍정적 및 부정적 의미의 Cache-Validation 조건을 포함한다. 이는 검증자가 일치하는 경우 및 일치하지 않는 경우 모두 method 를 수행하도록 요구할 수 있다는 것이다.

주의 : 검증자가 없는 응답도 Cache-Control 지시자가 분명하게 금지하지 않는 한 캐시할 수 있으며 만료될

때까지 사용할 수 있다. 그러나 캐시는 엔터티에 대한 검증자가 없으면 조건적인 조회를 할 수 없다. 이는 캐시가 만료된 다음에는 갱신할 수 없다는 것을 의미한다.

13.3.1 최종 갱신 날짜(Last-modified Dates)

Last-Modified Entity-Header 필드 값은 종종 캐시 검증자로 사용된다. 간단히 말하면 캐시 엔트리는 엔터티가 Last-Modified 값 이후에 변경되지 않았으면 유효한 것으로 간주된다는 것이다.

13.3.2 엔터티 태그 캐시 검증자(Validators)

ETag Entity-Header 필드 값, 엔터티 태그는 "불투명한" 캐시 검증자를 제공한다. 이 검증자는 변경된 날짜를 저장하는 것이 불편한 상황에서, HTTP 날짜 값을 1 초 동안 분석하는 것이 충분하지 않은 상황에서 또는 원서버가 변경된 날짜를 사용하여 발생하는 특정 역설을 피하고자 하는 상황에서 좀더 신뢰성 있는 검증을 가능하게 한다.

엔터티 태그는 3.11 절에 기술되어 있고 엔터티 태그에 사용되는 헤더는 14.20, 14.25, 14.26 및 14.43 절에 기술되어 있다.

13.3.3 약한/강한 검증자

원서버 및 캐시 모두는 검증자가 동일한 엔터티를 표현하는지 상이한 엔터티를 표시하는지 결정하기 위해 두 검증자를 비교하기 때문에 우리는 엔터티(Entity-Body 또는 모든 Entity-Header)가 어떤 식으로든 변경되면 연관된 검증자도 또한 변경되리라 예상할 수 있다. 이것이 사실이라면 우리는 이 검증자를 "강한 검증자"라고 부른다.

그러나 서버가 엔터티의 미미한 측면이 변화할 때보다는 의미상으로 중대한 변화에 대해서만 검증자를 변경하려 하는 경우가 있을 수 있다.

[Page 82]

자원이 변화할 때마다 변화하지 않는 검증자가 "약한 검증자" 이다.

엔터티 태그는 대개 "강한 검증자"이지만 규약은 엔터티 태그를 "약한" 것으로 태그를 붙일 수 있는 메커니즘을 제공한다. 우리는 강한 검증자를 엔터티의 일 부분이라도 변하면 따라서 변하는 것이고 약한 값은 엔터티의 의미가 변화할 때마다 변화한다고 생각할 수 있다. 다르게 표현하면 강한 검증자는 특정 엔터티에 사용되는 식별자의 일부분으로 약한 검증자는 의미상으로 동일한 엔터티 세트를 위한 식별자의 일부라고 생각할 수 있다.

주의 : 강한 검증자의 한 예는 엔터티가 변할 때마다 불변 기억장치 내에서 증가되는 정수이다.

엔터티의 변경 시간은 일 초 동안의 분석으로 표시된다면 약한 검증자일 수 있다. 자원이 일 초 동안 두 번 변경될 수 있기 때문이다.

약한 검증자를 지원하는 것은 선택 사항이다. 그러나 약한 검증자는 동일한 객체에 대한 좀 더 효과적인 캐시를 가능하게 한다. 예를 들어 사이트의 방문 카운터는 2 - 3 일 또는 주마다 갱신해도 충분하다. 또한 이 기간 동안의 값은 충분히 동일한 것으로 간주할 수 있다.

검증자를 사용하는 것은 클라이언트가 요구를 생성하여 검증 헤더 필드에 검증자를 포함할 때 또는 서버가 두 검증자를 비교하는 때이다.

강한 검증자는 어떤 상황에서든지 유용하다. 약한 검증자는 엔터티가 완벽하게 동일하지 않아도 되는 상황에서 유용하다. 예를 들어 강한 검증자만이 하부-영역 검색에 유용하다. 그렇지 않다면 클라이언트는 내부적으로 일치하지 않은 엔터티로 종결될 것이다.

HTTP/1.1 규약이 검증자에 규정한 유일한 기능은 비교이다. 비교 상황이 약한 검증자의 사용을 허용하는가 여부에 따라 두 개의 검증자 비교 기능이 있다.

- 강한 비교 기능
  - 동등한 것으로 간주되기 위해서 두 검증자는 모든 면에서 동일해야 하며 모두 다 약한 검증자이면 안 된다.

- 약한 비교 기능
  - 동등한 것으로 간주되기 위해서 두 검증자는 모든 면에서 동일해야 하나 둘 중의 하나 또는 모두가 결과에 영향을 미치지 않고 "약한" 것으로 태그를 붙일 수 있다.

약한 비교 기능은 간단한(하부 영역이 아닌) GET 요구에 사용할 수 있다. 강한 비교 요구는 모든 다른 경우에 반드시 사용해야 한다.

[Page 83]

엔터티 태그는 분명하게 약한 것으로 태그를 붙이지 않는 한 강하다. 3.11 절에 엔터티 태그의 의미론이 있다.

요구의 검증자로 사용되었을 때 Last-Modified 시간은 다음의 규칙을 사용하여 강한 것으로 연역할 수 없는 한 함축적인 의미에서 약하다.

- 원서버가 엔터티의 실제적인 현재 검증자와 검증자를 비교하고 있거나

- 원서버가 신뢰할 수 있을 정도로 관련된 엔터티가 제시된 검증자가 다루고 있는 동안 두 번 변경되지 않았다는 사실을 알 수 있다.

- 클라이언트가 연관된 엔터티의 캐시 엔트리를 가지고 있기 때문에 If-Modified-Since 또는 If-Unmodified-Since 헤더에서 검증자를 사용하려고 한다.

- 해당 캐시 엔트리가 원서버가 원래의 응답을 발송한 시간을 알려 주는 Date 값을 포함하고 있다.

- 제시된 Last-Modified 시간이 최소한 Date 값보다 60초 이전이다.

또는

- 중간 캐시가 검증자와 엔터티에 사용되는 캐시 엔트리에 저장된 검증자를 비교하고 있는 중이다.

- 해당 캐시 엔트리가 원서버가 원래의 응답을 발송한 시간을 알려 주는 Date 값을 포함하고 있다.

- 제시된 Last-Modified 시간이 최소한 Date 값보다 60초 이전이다.

이 method 는 원서버가 두 개의 다른 응답을 같은 시간에 발송했으나 둘 모두 동일한 Last-Modified 시간을 가지고 있으면 이 응답 중 최소한 하나는 Date 값이 Last-Modified 시간과 동등하다는 사실에 기초하고 있다. 임의적인 60 초 제한은 Date 와 Last-Modified 값이 별도의 시계에서 생성되었거나

응답을 준비하는 동안 다른 시간대에 생성되었을 가능성에 대비시켜 준다. 구현 방법에 60 초가 너무 짧다고 생각되면 60 초 이상의 값을 사용할 수 있다.

클라이언트가 Last-Modified 시간과 불투명하지 않은 검증자를 가지고 있는 값에 대하여 하부 영역 조회를 수행하고 싶으면 Last-Modified 시간이 여기에서 기술한 의미에서 강할 경우에만 수행할 수 있다.

[Page 84]

전체-본문 GET 요구 이외의 Cache-Conditional 요구를 수신하고 있는 캐시 또는 원서버는 반드시 조건을 평가하기 위해 강한 비교 기능을 사용해야 한다.

이러한 규칙은 HTTP/1.1 캐시와 클라이언트가 HTTP/1.0 서버에서 획득된 값에 대하여 하부 영역 조회를 안전하게 수행할 수 있게 한다.

13.3.4 엔터티 태그와 최종 갱신 날짜를 사용할 때를 결정하는 규칙

어떠한 목적이건 다양한 검증자 유형을 사용해야 할 때의 원서버, 클라이언트 및 캐시를 위한 일련의 규칙과 권고안을 채택하였다.

HTTP/1.1 원서버:

- 새로운 것을 생성하는 것이 불가능하지 않는 한 엔터티 태그 검증자를 발송해야 한다.
- 성능에 대해 고려했을 때 약한 엔터티 태그를 사용해도 될 때 또는 강한 엔터티 태그를 발송하는 것이 실현성이 없을 때 강한 엔터티 태그 대신 약한 엔터티 태그를 발송할 수도 있다.
- If-Modified-Since 헤더의 날짜를 사용하면 의미 투명성이 파괴될 위험성이 심각한 문제를 초래하지 않는 한, Last-Modified값을 발송하는 것이 가능하면 값을 보내야 한다.

달리 표현하면 HTTP/1.1 원서버의 바람직한 행태는 강한 엔터티 태그와 Last-Modified 값 모두를 발송하는 것이다.

합법적이면 강한 엔터티 태그는 관련된 엔터티 값이 어떤 식으로든 변경될 때마다 변경되어야 한다. 약한 엔터티 태그는 관련된 엔터티가 의미상 상당히 변경되었으면 변경되어야 한다.

주의 : 의미상 투명한 캐시를 제공하기 위해서 원서버는 두 개의 별도 엔터티에 특정한 강한 엔터티 값을 재사용하지 말아야 한다. 캐시 엔트리는 유효 시간에 관계없이 임의적으로 긴 기간동안 지속될 수 있다. 따라서 과거의 특정 시점에 획득한 검증자를 사용하는 캐시가 결코 엔터티를 검증하려 시도하지 않는다고 예상하는 것은 적절하지 않다.

HTTP/1.1 클라이언트:

- 서버가 엔터티 태그를 제공하였으면 클라이언트는 어떠한 Cache-Conditional 요구에서든지 그 엔터티 태그를 반드시 사용해야 한다.(If-Match 또는 If-None-Match를 사용)

[Page 85]

- 원서버가 단지 Last-Modified 값만을 제공했을 때 클라이언트는 하부 영역이 아닌 Cache-Conditional 요구에 그 값을 사용할 수 있다. (If-Modified-Since를 사용)
- HTTP/1.0 원서버가 Last-Modified 값만을 제공했을 때 클라이언트는 그 값을 하부 영역의 Cache-Conditional 요구에 사용할 수 있다.(If-Unmodified-Since:를 사용). 사용자 에이전트는 장애가 있을 경우 이 기능을 중지시킬 방법을 제공해야 한다.

• 원서버가 엔터티 태그 및 Last-Modified 값 모두를 제공했을 때 클라이언트는 양쪽 검증자를 Cache-Conditional 요구에 사용해야 한다. 이것은 HTTP/1.1 및 HTTP/1.0 모두가 적절히 응답할 수 있도록 한다.

요구를 접수하자마자 HTTP/1.1 캐시는 클라이언트의 캐시 엔트리가 캐시 자신의 캐시 엔트리와 일치하는지 여부를 결정할 때 가장 제한적인 검증자를 반드시 사용해야 한다. 이것은 요구가 엔터티 태그와 last-modified-date 검증자(If-Modified-Since 또는 If-Unmodified-Since) 모두를 포함하고 있을 때만 문제시 된다.

논리에 대한 주석

- 이 규칙 뒤의 일반적인 원칙은 HTTP/1.1 서버와 클라이언트는 자신의 응답 및 요구에서 최대한 중첩되지 않는 정보를 전달해야 한다는 것이다. 이 정보를 수신하는 HTTP/1.1 시스템은 자신이 수신하는 검증자에 관한 가장 조심스러운 가정을 할 것이다.

HTTP/1.0 클라이언트와 캐시는 엔터티 태그를 무시할 것이다. 일반적으로 이 시스템들이 수신하거나 사용하는 Last-Modified 값은 투명하거나 효과적인 캐시를 지원하기 때문에 HTTP/1.1 원서버는 Last-Modified 값을 제공해야 한다. HTTP/1.0 시스템이 Last-Modified 값을 검증자로 사용하여 심각한 문제를 초래하는 극히 드문 경우에는 HTTP/1.1 원서버는 값을 제공하지 말아야 한다.

13.3.5 검증을 하지 않는 조건법

엔터티 태그 뒤의 원칙은 서비스 저작자만이 자원의 의미를 충분히 알아서 적합한 캐시 검증 메커니즘을 선택할 수 있다는 것이다. 바이트-동등(byte-equality) 비교보다 더 복잡한 검증자 비교 기능을 열거하면 아주 복잡하게 될 것이다. 따라서 캐시 엔트리를 검증할 목적으로 다른 어떤 헤더의 비교(HTTP/1.0 과의 호환성을 위해 Last-Modified 는 제외) 도 사용해서는 안 된다.

13.4 응답을 캐시할 수 있는 정도(Cachability)

Cache-Control (14.9 절) 지시자가 특별히 통제하지 않는 한 캐시 시스템은 언제나 성공적인 응답을 캐시 엔트리로서 저장할 수 있고, 새로운 것이라면 검증 없이 리턴할 수 있으며 성공적인 검증 후에 리턴할 수도 있다.

[Page 86]

캐시 검증자도 없고 응답과 관련된 명확한 유효 시간도 없으면 우리는 그것을 캐시할 수 있다고 기대하지 않는다. 그러나 특정한 캐시는 이러한 기대를 위반할 수도 있다.(예를 들어 네트워크가 약하게 연결되었거나 연결되지 않았을 때). 클라이언트는 보통 이러한 응답이 캐시에서 나왔다는 것을 Date 헤더와 현재 시간을 비교하여 탐지할 수 있다.

몇몇 HTTP/1.0 캐시는 아무런 Warning 도 제공하지 않고 이러한 기대를 위반하는 것으로 알려졌다는 점에 주의한다.

그러나 어떤 경우에는 캐시가 엔터티를 보유하고 있는 것이, 또는 캐시를 계속되는 요구에 대한 응답으로 리턴하는 것이 적절하지 않을 수 있다. 이는 서비스 저작자는 절대적인 의미 투명성을 당연한 것으로 여기고 있기 때문이거나 또는 보안이나 사생활 보호에 관한 고려 사항 때문이다. 따라서 다른 고려 사항에 관계없이 특정 엔터티 또는 엔터티의 일부를 캐시할 수 없다는 것을 표시하기 위해 특정 Cache-Control 지시자가 제공되었다.

14.8 절은 대개 요구가 Authorization 헤더를 포함하고 있으면 공유된 캐시가 이전 요구에 대한 응답을 저장하거나 리턴할 수 없게 한다는 점에 주의한다.

Cache-Control 지시자가 캐시를 금지하지 않는 한 상태 코드 200, 203, 206, 300, 301 또는 410 와 더불어 수신한 응답은 유효일 메커니즘을 조건으로 하여 캐시로 저장할 수 있고 계속되는 요구에 대한 응답에 사용할 수 있다. 그러나 Range 및 Content-Range 헤더를 지원하지 않는 캐시는 206(Partial Content) 응답을 절대로 캐시해서는 안 된다. 다른 상태 코드와 함께 수신된 응답은 Cache-Control 지시자나 이를 명백하게 허용하는 헤더가 없으면 계속되는 요구에 대한 응답으로 절대 리턴해서는 안 된다. 예를 들어 이러한 것들은 다음과 같다. - Expires 헤더 (14.21 절); "max-age", "must-revalidate", "proxy-revalidate", "public" 또는 "private" Cache-Control 지시자 (14.9 절).

13.5 캐시에서 응답을 구축하기

HTTP 캐시의 목적은 요구에 대한 응답으로 수신된 정보를 저장하여 미래의 요구에 대한 응답을 사용하기 위함이다. 많은 경우에 캐시는 단순히 요구자에게 응답의 적합한 부분을 리턴한다. 그러나 캐시가 이전 응답에 기초한 캐시 엔트리를 보유하고 있으면 새로운 응답의 일 부분과 기존 캐시 엔트리에 보관하고 있던 것을 결합해야만 한다.

[Page 87]

13.5.1 End-to-end 및 Hop-by-hop 헤더

캐시와 캐시를 지원하지 않는 프락시의 행태를 규정할 목적으로 우리는 HTTP 헤더를 두 종류도 구분하였다.

- End-to-end 헤더
  - 요구나 응답의 궁극적인 수신측에게 전달되어야 한다. 응답의 End-to-end 헤더는 캐시 엔트리의 일 부분으로 저장되어야 하고 캐시 엔트리에서 구성된 어떠한 응답에도 전송되어야 한다.

- Hop-by-hop 헤더
  - 단일-전송-수준 접속에만 의미가 있으며 캐시가 저장하지도 않고 프락시가 전송하지도 않는다.

다음의 HTTP/1.1 헤더들이 hop-by-hop 헤더이다:

- Connection
- Keep-Alive
- Public
- Proxy-Authenticate
- Transfer-Encoding
- Upgrade

HTTP/1.1 이 규정한 다른 모든 헤더는 end-to-end 헤더이다.

HTTP 향후 버전에 소개될 Hop-by-hop 헤더는 14.10 절에 기술한 것처럼 반드시 Connection 헤더에 열거되어야 한다.

13.5.2 변경할 수 없는 헤더

Digest Authentication 과 같은 HTTP/1.1 규약의 몇몇 기능은 특정 end-to-end 헤더 값에 의존한다. 캐시나 캐시를 지원하지 않는 프락시는 해당 헤더의 규정이 이를 요구하거나 특별히 허용하지 않는 한 end-to-end 헤더를 변경해서는 안 된다. 캐시나 캐시를 지원하지 않는 프락시는 요구나 응답에서 다음의 필드 중 어떤 것도 변경해서는 안 되며 기존에 존재하지 않으면 추가해서도 안 된다.

- Content-Location
- ETag
- Expires
- Last-Modified

[Page 88]

캐시나 캐시를 지원하지 않는 프락시는 변환 금지 Cache-Control 지시어를 포함한 응답 또는 어떠한 요구에서도 다음의 필드 중 어떤 것도 변경하거나 추가해서는 안 된다.

- Content-Encoding
- Content-Length
- Content-Range
- Content-Type

캐시나 캐시를 지원하지 않는 프락시는 변환 금지(no-transform)를 포함하지 않은 응답에서 이러한 필드를 변경하거나 추가해서는 안 된다. 그렇게 했다면 응답에 Warning 14(Modification Applied)가 없으면 이를 추가해야 한다.

경고 : 불필요한 end-to-end 헤더의 변경은 HTTP의 이후 버전에서 더 강력한 인증 획득 메커니즘이 도입된다면 인증 획득 실패를 초래할 수 있다. 이러한 인증 획득 메커니즘은 여기에 열거되지 않은 헤더 필드 값에 의존할 수 있다.

13.5.3 헤더의 결합

캐시가 서버에게 검증 요구를 하고 서버가 304(Not Modified) 응답을 주었을 때 캐시는 요구한 클라이언트에게 발송하기 위해 응답을 구축해야 한다. 캐시는 캐시 엔트리에 보관된 Entity-Body 를 이 발송 응답의 Entity-Body 로 사용한다. 캐시 엔트리에 저장된 end-to-end 헤더는 304 응답이 제공한 end-to-end 헤더가 캐시 엔트리에서 상응하는 헤더를 반드시 대체해야만 하는 경우를 제외하고 구축된 응답을 위해 사용된다. 캐시가 캐시 엔트리를 제거하기로 결정하지 않는 한 캐시는 반드시 캐시 엔트리에 저장된 end-to-end 헤더를 들어오는 응답에서 수신한 헤더로 대체해야 한다.

달리 표현하면 들어오는 메시지에서 수신한 end-to-end 헤더 세트가 캐시 엔트리에 저장된 모든 end-to-end 헤더를 무시한다는 것이다. 이 캐시는 이 세트에 Warning 헤더를 추가할 수 있다.(14.45 절 참조)

들어오는 응답의 헤더 필드 이름이 캐시 엔트리의 헤더와 하나 이상 일치하면 이러한 모든 오래된 헤더는 대체된다.

주의 : 이 규칙은 원서버가 304(Not Modified) 응답을 사용하여 동일 엔터티에 대한 이전 응답에 관련된 헤더를 갱신할 수 있도록 해 준다. 이렇게 하는 것이 항상 의미가 있거나 올바를 것은 아닐 수 있다. 이 규칙은 원서버가 304(Not Modified) 응답을 사용하여 이전 응답에 제공했던 헤더를 완전히 삭제할 수 있도록 허용하는 것은 아니다.

[Page 89]

13.5.4 바이트 영역(Byte Ranges)의 결합

응답은 요구가 하나 또는 그 이상의 Range 규격을 포함하고 있거나 접속이 조기에 단절되었기 때문에 Entity-Body 바이트의 하부 영역만을 전송할 수 있다. 이러한 방식으로 몇 번 전송하면 캐시는 동일한 Entity-Body 의 몇몇 영역을 수신할 수 있을 것이다.

캐시가 엔터티의 저장되어 있고 공백이 아닌 하부 영역 세트를 가지고 있고 또한 들어오는 응답이 다른 하부 영역을 전송한다면 캐시는 새로운 하부 영역을 기존의 세트와 결합할 수 있다. (아래의 두 조건을 모두 만족해야 한다.)

- 들어오는 응답과 캐시 엔트리 모두 캐시 검증자를 가지고 있어야 한다.
- 두 캐시 검증자는 강한 비교 기능(13.3.3 절 참조)을 사용하여 서로 일치해야 한다.

두 조건 모두가 만족되지 않으면 캐시는 가장 최근의 부분 응답(모든 응답과 함께 전송되는 Date 값에 기초하거나 또는 이 값이 동등하거나 빠져 있으면 들어오는 응답을 사용하여)만을 사용해야 하고 다른 부분 정보는 폐기해야 한다.

13.6 협상을 통한 응답을 캐시하기

응답의 Vary 헤더 필드의 존재로 표시되는 서버가 주도하는 내용 협상(12 장)의 사용은 캐시가 계속적인 요구에 대한 응답에 사용할 수 있는 조건 및 절차를 변경한다.

서버는 Vary 헤더 필드(14.43 절)를 사용하여 캐시에게 어떤 헤더 필드 차원을 이용하여 캐시할 수 있는 응답의 다양한 표시 방법 중에서 하나를 선택하였는지 알려 준다. 캐시는 해당 자원에 대한 계속적인 요구에 응답하기 위하여 계속되는 요구가 Vary Response-Header 안에 명시된 모든 헤더 필드의 동일 또는 동등한 값을 가지고 있을 때만 선택된 표시 방법(특정 응답에 포함된 엔터티)을 이용할 수 있다. 하나 또는 그 이상의 이러한 헤더 필드를 가진 요구는 원서버로 전송될 것이다.

엔터티 태그가 표시 방법에 부여되었을 때 전송된 요구는 조건적이어야 하며 Request-URI 의 모든 캐시 엔트리로부터의 If-None-Match 헤더 필드에 있는 엔터티 태그를 포함해야 한다. 이것은 현재 캐시가 가지고 있는 엔터티 세트를 서버에게 전달한다. 이렇게 함으로써 만약 이 엔터티 중 하나라도 요구된 엔터티와 일치한다면 서버는 304 (Not Modified) 응답의 ETag 헤더를 이용하여 어떤 엔트리가 적합한지 캐시에게 알려 준다. 새로운 응답의 Entity-Tag 가 기존 엔트리의 Entity-Tag 와 일치한다면 새로운 응답을 이용하여 기존 엔트리의 헤더 필드를 갱신해야 하고 결과는 클라이언트에게 돌려주어야 한다.

[Page 90]

또한 Vary 헤더 필드는 캐시에게 Request-Header 에 한정되지 않은 범주를 이용하여 표시 방법이 선택되었음을 알릴 수 있다. 이 경우 캐시는 캐시가 새로운 요구를 조건적인 요구로 원서버에게 중계하거나 서버가 엔터티 태그 또는 어떤 엔터티를 사용하여야 하는지 표시하는 Content-Location 을 포함하여 304(Not Modified)로 응답하지 않는 한 응답을 계속되는 요구에 대한 대답으로 사용해서는 절대 안 된다. 기존의 캐시 엔트리 중 어떤 것이라도 관련된 엔터티의 부분적인 내용을 포함하고 있다면 요구가 해당 엔트리가 충분히 충족시킬 수 있는 영역에 관한 것이 아닌 이상 그것의 Entity-Tag 를 If-None-Match 헤더에 포함해서는 안 된다.

캐시가 Content-Location 필드가 동일한 Request-URI를 위한 기존 캐시 엔트리의 그것과 일치하며 Date가 기존 엔트리의 그것보다 최신인 성공적인 응답을 수신하였으면 기존의 엔트리를 향후 요구에 대한 응답으로 리턴해서는 안 되며 캐시에서 삭제해야 한다.

13.7 공유/비공유 캐시

보안 및 사생활 보호의 이유 때문에 "공유" 및 " 비공유" 캐시를 구별할 필요가 있다. 비공유 캐시는 단일 사용자만 접근할 수 있는 캐시이다. 이 경우의 접근성은 적절한 보안 메커니즘으로 집행된다. 다른 모든 캐시는 "공유"된 캐시로 간주된다. 이 규격의 다른 섹션에서는 공유된 캐시의 운영에 대해 사생활 보호와 접근 통제 실패를 방지하기 위하여 어떠한 제한을 두고 있다.

13.8 에러 또는 불완전한 응답 캐시 행태

불완전한 응답을 수신하는 캐시(예를 들어 Content-Length 헤더에 명시된 것보다 적은 데이터)는 응답을 저장할 수 있다. 그러나 캐시는 이것을 부분적 응답으로 처리해서는 안 된다. 부분적 응답은 13.5.4 절에서 기술한 것처럼 결합할 수 있다. 그 결과는 완전한 응답이 될 수도 있고 여전히 부분적일 수도 있다. 캐시는 클라이언트에게 분명하게 206(Partial Content) 상태 코드를 사용하여 그렇다고 표시하지 않고는 부분적 응답을 리턴해서는 안 된다. 캐시는 상태 코드 200(OK)을 이용하여 부분적 응답을 리턴해서는 절대로 안 된다.

캐시가 엔트리를 재검증하려 시도하는 동안 5xx 응답을 수신하면 캐시는 이 응답을 요구한 클라이언트에게 전송하든지 서버가 응답에 실패한 것처럼 처리해야 한다. 후자의 경우 캐시 된 엔트리가 "must-revalidate" Cache-Control 지시자(14.9 절 참조)를 포함하고 있지 않으면 캐시는 이전에 수신된 응답을 리턴할 수도 있다.

[Page 91]

13.9 GET 및 HEAD의 부작용

원서버가 분명하게 자신의 응답을 캐시하는 것을 금지하지 않는 한 GET 및 HEAD method를 어떤 자원에 적용하는 것은 이러한 응답이 캐시에서 왔다면 잘못된 행태로 이끄는 부작용을 갖지 않아야 한다. 그래도 부작용은 있을 수 있으나 캐시는 캐시 실행 여부에 대한 결정을 할 때 이러한 부작용을 고려하도록 요구 받지는 않는다. 캐시는 항상 원서버의 캐시에 대한 명백한 제한 사항을 준수할 것으로 예상된다. 우리는 이 규칙에 대한 예외 하나를 기록하고자 한다: 몇몇 애플리케이션은 전통적으로 GET 과 HEAD를 질의 URL(rel_path part에 "?"를 포함한 URL)과 함께 사용하여 상당한 부작용을 가진 작업을 수행했다. 캐시는 서버가 분명한 유효 시간을 제공하지 않는 한 이러한 URL에 대한 응답을 새로운 것으로 취급해서는 안 된다. 이것은 특히 HTTP/1.0 서버로부터의 응답을 캐시에서 가져와서는 안 된다는 것을 의미한다. 관련된 정보를 9.1.1 절을 참고한다.

13.10 갱신 또는 삭제 후의 무효화

원서버에서 특정 method의 결과는 하나 또는 그 이상의 기존 캐시 엔트리를 투명하지 않게 무효한 것으로 만들 수 있다. 이는 비록 캐시가 계속해서 "새롭지만" 이것이 원서버가 새로운 요구를 받았을 때 리턴할 것을 정확하게 반영하지는 않는다.

HTTP 규약이 모든 이러한 캐시를 무효한 것으로 표시한다는 보장을 할 방법이 없다. 예를 들어 원서버의 변화를 초래한 요구가 캐시 엔트리가 저장되어 있는 프락시를 거치지 않았을 수도 있다. 그러나 몇몇

규칙이 잘못된 행태가 발생할 수 있는 경우를 감소시켜 준다. 이 섹션에서 "엔트리를 무효화 한다"는 것은 캐시가 해당 엔터티의 모든 경우(instance)를 저장 장소에서 삭제하고, "무효"로 표시하며 계속되는 요구에 대한 응답으로 리턴하기 전에 의무적으로 재검증을 할 필요가 있다는 것을 의미한다.

[Page 92]

몇몇 HTTP method 는 엔터티를 무효화할 수 있다. 그것은 Request-URI 로 참조하거나 Location 또는 Content-Location Response-Header(존재한다면)로 참조하는 엔터티이다.

그러한 method 는:

- PUT
- DELETE
- POST

서비스 시도의 거절을 방지하기 위해 Location 또는 Content-Location 헤더의 URI 에 기초한 무효화는 오직 호스트 부분이 Request-URI 의 호스트 부분과 동일할 경우에만 실행해야 한다.

13.11 의무적으로 서버를 통하여 기입(Write-Through Mandatory)

원서버 자원에 대한 변경을 초래할 것으로 예상되는 모든 method 는 원서버를 통하여 기입해야 한다. 현재 이것은 GET 및 HEAD 를 제외한 모든 method 를 포함한다. 캐시는 수신하는 쪽 서버로 요구를 전달하기 전 및 수신하는 쪽 서버로부터 상응하는 응답을 수신하기 전에 절대로 클라이언트에서 이러한 요구에 응답해서는 안 된다. 이것이 캐시가 수신하는 쪽 서버가 응답하기 전에 100 (Continue) 응답을 발송하는 것을 방지하지는 못한다. 대안("역으로 쓰기(write-back)" 또는 "역으로 복사(copy-back)" 캐시로 알려진)은 역으로 쓰기(write-back) 이전의 서버, 캐시 또는 네트워크 실패 때문에 발생하는 문제와 지속적인 갱신을 제공하기 어렵기 때문에 HTTP/1.1 에서 허용되지 않는다.

13.12 캐시 대체

모든 동일한 자원에 대한 기존 응답이 캐시되었을 때 새로운 캐시할 수 있는(14.9.2, 13.2.5, 13.2.6 및 13.8 절 참조) 응답을 자원으로부터 수신하였으면 캐시는 현재 의 요구에 대한 대답을 할 때 새로운 응답을 사용해야 한다. 캐시는 그것을 캐시 저장소에 삽입하고 모든 다른 조건이 충족되면 이전 응답이 리턴 되었을 수도 있는 미래의 요구에 대한 응답으로 사용한다. 새로운 응답을 캐시 저장소에 삽입한다면 13.5.3 절의 규칙을 따라야 한다.

주의 : 기존에 캐시된 응답보다 오래된 Date 헤더 값을 가진 새로운 응답을 캐시할 수 없다.

13.13 히스토리 목록(History list)

사용자 에이전트는 종종 "이전" 버튼과 history 목록과 같은 history 메커니즘을 사용한다. 이것은 세션에서 이전에 조회한 엔터티를 다시 화면에 표시하기 위해 사용한다.

[Page 93]

history 메커니즘과 캐시는 다르다. 특히 history 메커니즘은 자원의 현재 상태를 의미상 투명한 모양으로 보여 주려 시도해서는 안 된다. history 메커니즘은 자원을 조회했을 때 사용자가 본 것과 동일한 것을 보여 주기 위한 것이다.

기본적으로 유효 시간은 history 메커니즘에 적용되지 않는다. 엔터티가 여전히 저장소에 있으면 history 메커니즘은 엔터티가 만료되었다 할지라도 사용자가 만료된 history 문서를 갱신하도록 상세하게 에이전트의 환경을 설정하지 않은 한 이것을 보여 주어야 한다.

이것은 history 메커니즘이 사용자에게 현재 의 화면이 낡은 것일 수도 있음을 알리는 것을 금지하는 것으로 해석해서는 안 된다.

주의 : History 목록 메커니즘이 불필요하게 사용자가 낡은 자원을 볼 수 없도록 한다면 이는 서비스 저작자들에게 그렇지 않았더라면 사용하고 싶어하는 HTTP 유효일 제어 및 캐시 제어의 사용을 피하도록 강요할 수 있다. 서비스 저작자들은 그들이 운행 제어("이전" 버튼과 같은 navigation control)를 사용하여 이전에 가져온 자원을 보고자 할 때 사용자에게 에러 메시지나 경고 메시지를 표시하지 않는 것이 중요하다고 생각할 수 있다. 때때로 이러한 자원을 캐시하지 말거나 빨리 만료해야 할 수도 있지만 서비스 저작자들은 사용자 인터페이스를 고려하여 사용자가 history 메커니즘이 부적합하게 작동하여 고통을 받지 않도록 캐시를 방지할 수 있는(예를 들어 "once-only" URL) 다른 방법에 호소하도록 만든다.

14 헤더필드 정의

이 섹션은 HTTP/1.1 의 모든 표준 헤더 필드에 형식과 의미를 정의한다. Entity-Header 필드에서 발송자와 수신측은 누가 엔터티를 발송하고 누가 엔터티를 수신하는가에 따라 클라이언트 또는 서버 모두를 지칭할 수 있다.

[Page 94]

14.1 Accept

Accept request-header 필드는 응답에 사용할 수 있는 특정 media type 을 명시하는 데 사용할 수 있다. Accept 헤더는 라인에 포함된 이미지(in-line image)에 대한 요구처럼 요구가 상세하게 원하는 유형의 작은 세트에 한정되어 있음을 표시하는 데 사용한다.

```
Accept              = "Accept" ":"
                      #( media-range [ accept-params ] )
media-range         = ( "*/*"
                      | ( type "/" "*" )
                      | ( type "/" subtype )
                      ) *( ";" parameter )
accept-params       = ";" "q" "=" qvalue *( accept-extension )
accept-extension    = ";" token [ "=" ( token | quoted-string ) ]
```

별표("*") 문자는 media type 을 영역으로 그룹핑하는 데 사용한다. "*/*"는 모든 media type 을, "type/*은 해당 type 의 모든 subtype 을 표시한다. Media-range 는 해당 영역에 적용할 수 있는 media type 파라미터를 포함할 수 있다.

모든 media-range 에는 하나 또는 그 이상의 상대적 품질 요소를 표시하는 "q" 파라미터로 시작하는 accept-params 가 뒤따른다. 처음의 "q" 파라미터(있다면)는 media-range 파라미터를 accept-params 로부터 분리시킨다. 품질 요소(quality factor)는 사용자 또는 사용자 에이전트가 qvalue(섹션 3.9) 척도를 0 부터 1 까지 사용하여 해당 media-range 에 대한 상대적 선호도를 표시할 수 있도록 한다. 기본값은 q=1 이다.

주의 : "q" 파라미터 이름을 media type 파라미터를 Accept 확장 파라미터로부터 분리하는 데 사용하는 것은 계속적인 관행 때문이다. 이 관행이 어떠한 media type 파라미터에도 "q"라는 이름을 media range에 사용할 수 없게 하지만 IATA media type 등록표에서 "q" 파라미터가 많이 사용되지 않고 있고 Accept에서 media type 파라미터를 잘 사용하지 않는다는 점을 감안할 때 이러한 경우는 발생하기 어렵다. 향후 media type은 "q"라는 이름의 파라미터를 등록하지 말아야 한다.

예:

Accept: audio/*; q=0.2, audio/basic

위의 문장은 "나는 audio/basic 을 선호하지만 80% 이하로 질이 떨어지면 사용할 수 있는 가장 좋은 다른 audio type 을 발송해 주시오."로 해석해야 한다.

[Page 95]

Accept 헤더 필드가 없다면 클라이언트가 모든 media type 을 수용한다고 가정한다. Accept 헤더 필드가 있고 서버가 결합된 Accpet 필드 값에 적합한 응답을 발송할 수 없을 때 서버는 406 (not acceptable) 응답을 발송해야 한다.

좀 더 자세한 예는

```
Accept:         text/plain; q=0.5, text/html,
                text/x-dvi; q=0.8, text/x-c
```

말로 표현한다면 이것을 "text/html 및 text/x-c 가 선호하는 media type 이지만 이것이 존재하지 않으면 text/x-dvi 엔터티를 발송하고 존재하면 text/plain 엔터티를 발송하시오."라고 해석할 수 있다.

Media ranges 는 좀더 상세한 media range 및 media type 에 의하여 무시될 수 있다. 특정 type 에 하나 이상의 media range 를 적용했을 때 가장 상세한 것이 우선권을 갖는다.

예를 들면,

Accept: text/*, text/html, text/html;level=1, */*

은 다음의 우선순위를 가진다.

1) text/html;level=1
2) text/html
3) text/*
4) */*

특정 type 과 관련된 media type 품질 요소는 해당 type 에 일치하는 media range 중 최고의 우선권을 갖는 것을 발견하여 결정한다.

예를 들어,

```
Accept:         text/*; q=0.3, text/html; q=0.7, text/html; level=1,
                text/html; level=2; q=0.4, */*; q=0.5
```

는 아래와 같은 값이 연관되도록 한다.

```
text/html;level=1       = 1
text/html               = 0.7
text/plain              = 0.3
image/jpeg              = 0.5
```

text/html;level=2 = 0.4
text/html;level=3 = 0.7

주의 : 사용자 에이전트는 특정 media range에 대한 품질의 기본 값 세트를 가지고 있을 수 있다.

[Page 96]

그러나 사용자 에이전트가 다른 표시 에이전트와 상호 작용할 수 없는 폐쇄된 시스템이 아니라면 기본 값 세트는 사용자가 설정할 수 있어야 한다.

14.2 Accept-Charset

Accept-Charset request-header 필드는 응답에 사용할 수 있는 문자 조합을 표시하는 데 사용한다. 이 필드는 광범위하고 전문적인 목적의 문자 조합을 이해할 수 있는 클라이언트가 이러한 문자 조합으로 문서를 표시할 수 있는 서버에게 자신의 능력을 알려 줄 수 있도록 한다. 모든 클라이언트는 ISO-8859-1 문자 조합을 사용할 수 있는 것으로 가정한다.

```
Accept-Charset = "Accept-Charset" ":"
                 1#( charset [ ";" "q" "=" qvalue ] )
```

문자 조합 값은 3.4 절에 설명되어 있다. 각각의 charset 는 해당 charset 에 대한 사용자의 선호 사항을 표시하는 관련된 품질 값을 가질 수 있다. 기본값은 q=1 이며 예는 다음과 같다.

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

Accept-Charset 헤더가 있으면 기본값은 모든 문자 조합을 사용할 수 있다. Accept-Charset 헤더가 있고 서버가 Accept-Charset 헤더를 사용할 수 있는 응답을 발송할 수 없을 때 비록 응답을 발송할 수는 있지만 서버는 406 (not acceptable) 상태 코드의 에러 메시지를 발송해야 한다.

14.3 Accept-Encoding

Accept-Encoding request-header 필드는 Accept 와 유사하지만 응답에서 사용할 수 있는 content-coding 값(섹션 14.12)에 제한이 있다.

```
Accept-Encoding = "Accept-Encoding" ":"
                  #( content-coding )
```

이의 사용 예는,

```
Accept-Encoding: compress, gzip
```

요청에 Accept-Encoding 헤더가 없으면 서버는 클라이언트가 어떠한 content coding 도 접수할 수 있다고 가정할 수 있다. Accept-Encoding 헤더가 있고 서버가 Accept- Encoding 헤더를 사용할 수 있는 응답을 발송할 수 없을 때 비록 응답을 발송할 수는 있지만 서버는 406 (not acceptable) 상태 코드의 에러 메시지를 발송해야 한다.

[Page 97]

값이 비어 있으면 아무 것도 접수할 수 없음을 표시한다.

14.4 Accept-Language

Accept-Language request-header 필드는 Accept 와 유사하지만 요구에 대한 응답으로 사용할 수 있는 자연스러운 언어(natural languages) 세트에 제한이 있다.

```
Accept-Language        = "Accept-Language" ":"
                         1#( language-range [ ";" "q" "=" qvalue ] )
language-range         = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

각각의 language-range 는 해당 range 가 명시하는 언어에 대한 사용자의 예상 선호 상태를 표시하는 관련 품질 값을 가지고 있다. 품질 기본 값은 "q=1" 이며 예는 다음과 같다.

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

이것은 "나는 덴마크어를 선호하지만 영국 영어 및 다른 유형의 영어도 접수할 것이다."를 의미한다. 태그가 완전히 동일하거나 접두사 뒤의 첫 태그 문자가 "-" 이면 language-range 는 language-tag 와 일치한다. 특수 영역(special range) "*"가 Accept-Language 필드에 있으면 Accept-Language 필드에 있는 다른 range 에 일치하지 않는 모든 태그를 일치시켜 준다.

주의 : 접두사 일치 원칙의 사용은 언어 태그가 "사용자가 특정 태그의 언어를 이해한다면 이 사용자는 이 태그가 접두사로 쓰인 모든 언어 태그를 이해할 것이다."라는 것이 항상 사실이라는 방식으로 부여 되었다는 것을 의미하지는 않는다. 접두사 원칙은 단순히 이것이 사실일 경우에만 접두사 사용을 허락하는 것이다.

Accept-Language 필드의 language-tag 가 부여한 언어 품질 요소는 language-tag 와 일치하는 필드의 가장 긴 language-range 품질 값이다. 필드의 language-range 와 일치하는 태그가 없으면 언어 품질 요소는 0 으로 부여된다. 요구에 Accept-Language 헤더가 없으면 서버는 모든 언어를 동등하게 수용할 수 있다고 가정해야 한다. Accept-Language 헤더가 있으면 0 이상의 품질 요소를 부여 받은 모든 언어를 수용할 수 있다. 모든 요구에 사용자의 전체적인 언어 선호 상태를 포함한 Accept-Language 헤더를 발송하는 것이 사용자의 사생활 보호에 대한 기대에 역행할 수도 있다. 이 문제에 관한 토의는 15.7 절을 참조한다.

[Page 98]

주의 : 이해정도(intelligibility)는 개별적인 사용자에 따라 다르므로 클라이언트 애플리케이션은 사용자가 원하는 언어를 선택할 수 있도록 할 것을 추천한다. 선택을 할 수 없으면 요구에 Accept-Language 헤더 필드를 제공해서는 안 된다.

14.5 Accept-Ranges

Accept-Ranges response-header 필드는 서버가 자원에 대한 영역 요구(range requests)를 접수하였다는 것을 표시한다.

```
Accept-Ranges          = "Accept-Ranges" ":" acceptable-ranges
acceptable-ranges      = 1#range-unit | "none"
```

Byte-range 요구를 접수한 원서버는 다음과 같이 발송할 수 있다.

```
Accept-Ranges: bytes
```

그러나 반드시 이렇게 해야 하는 것은 아니다. 클라이언트는 관련된 자원에 대해 이 헤더를 수신하지 않고도 byte-range 요구를 생산할 수 있다. 어떠한 형태의 byte-range 요구도 접수하지 않는 원서버는 다음을 발송하여 클라이언트가 영역 요구를 시도하지 말도록 충고할 수 있다.

```
Accept-Ranges: none
```

14.6 Age

Age response-header 필드는 원서버가 응답(또는 이의 검증)을 생성한 이후 시간에 대한 발송자의 예상 값을 전달한다. 캐시된 응답은 경과 시간이 신선한 시간(freshness lifetime)을 초과하지 않았으면 "신선하다." 경과 시간 값은 13.2.3 절에 기술한 바와 같이 산출한다.

```
Age             = "Age" ":" age-value
age-value       = delta-seconds
```

경과 시간 값은 음수가 아닌 십진수 정수이며 시간을 초로 표시한다.

[Page 99]

캐시가 가장 크게 표시할 수 있는 정수 값보다 큰 값을 수신하거나 경과 시간 계산이 오버플로우(overflow)되면 Age 헤더의 값을 반드시 2147483648 (2^31)로 전송해야 한다. HTTP/1.1 캐시는 모든 응답에 반드시 Age 헤더를 발송해야 한다. 캐시는 최소 31 비트 범위의 사칙연산 유형 값을 사용해야 한다.

14.7 Allow

Allow entity-header 필드는 Request-URI 가 식별한 자원이 지원하는 method 세트 목록을 표시한다. 이 필드의 용도는 엄격하게 자원과 관련된 유효한 method 의 수신을 알리기 위함이다. Allow 헤더 필드는 반드시 405 (Method Not Allowed) 응답 내에 표시되어야 한다.

```
Allow           = "Allow" ":" 1#method
```

이의 사용 예는,

```
Allow: GET, HEAD, PUT
```

이 필드는 클라이언트가 다른 methods 를 사용하고자 시도하는 것을 방지할 수는 없다. 그러나 Allow 헤더 필드가 표시하는 내용은 준수해야만 한다. 사용할 수 있는 실제 세트는 각 요구가 발송되는 시점에서 원서버가 결정한다.

Allow 헤더 필드에 PUT 요청을 새롭거나 변경된 자원이 지원하는 method 를 추천하기 위해 포함할 수 있다. 서버가 반드시 이러한 method 를 지원할 필요는 없으나 실제적으로 사용할 수 있는 method 를 제공하는 Allow 헤더 필드를 응답에 포함해야만 한다.

프락시는 명시된 모든 method 를 이해하지 못하더라도 사용자 에이전트가 다른 수단을 통하여 원서버와 통신할 수도 있기 때문에 Allow 헤더 필드를 변경해서는 절대로 안 된다. Allow 헤더 필드는 서버 수준에서 어떠한 method 가 구현되었는가 표시하지 않는다. 서버는 전체적으로 서버 상에서 어떠한 method 가 구현되었는가 표시하기 위해 Public response-header 필드(섹션 14.35)를 사용할 수 있다.

14.8 Authorization

서버에서 자신을 인증하고자 하는 사용자 에이전트는(꼭 그런 것은 아니지만 대개의 경우 401 응답을 수신한 후) 요구에 Authorization request-header 필드를 포함하여 자신의 인증 획득을 시도할 수 있다. Authorization 필드 값은 요구하고 있는 자원의 영역에 대한 사용자 에이전트의 인증 획득 정보를 포함하고 있는 보증서(credentials)로 구성되어 있다.

[Page 100]

```
Authorization             = "Authorization" ":" credentials
```

HTTP 접속 인증 획득은 11 장에 기술되어 있다. 요구에 대한 인증을 획득하고 영역이 명시되면 동일한 보증서는 해당 영역 내의 다른 요구에 대해서도 유효해야 한다.

공유된 캐시(13.절 참조)가 필드가 포함된 요구를 수신하면 캐시는 다음에 명시된 예외 사항 이외에는 다른 요구에 대한 대답으로서 해당 응답을 리턴해서는 절대 안 된다.

1. 응답이 "proxy-revalidate" Cache-Control 지시자를 포함하고 있지 않으면 캐시는 해당 응답을 계속되는 요구에 대한 대답으로 사용할 수 있다. 그러나 프락시 캐시는 원서버가 새로운 요구를 인증할 수 있도록 새로운 요구의 request-header를 이용하여 반드시 먼저 새로운 요구를 재검증해야 한다.

2. 응답이 "proxy-revalidate" Cache-Control 지시자를 포함하고 있으면 캐시는 해당 응답을 계속되는 요구에 대한 대답으로 사용할 수 있다. 그러나 모든 캐시는 원서버가 모든 요구를 인증할 수 있도록 새로운 요구의 request-header를 이용하여 반드시 먼저 새로운 요구를 재검증해야 한다.

3. 응답이 " public" Cache-Control 지시자를 포함하고 있으면 이를 계속되는 요구에 대한 대답으로 리턴할 수 있다.

14.9 Cache-Control

Cache-Control general-header 필드는 Request/Response chain 에 따라 모든 캐시 메커니즘이 반드시 따라야 하는 지시자를 표시하는 데 사용한다. 지시자는 캐시가 요구나 응답을 바람직하지 못하게 방해하지 못하도록 하는 행태(behavior)를 명시한다. 이러한 지시자들은 대개 기본적인 캐시 알고리즘을 무시한다. 캐시 지시자 요구에 지시자가 존재한다는 것이 응답에도 동일한 지시자를 부여해야 한다는 것의 의미하지 않다는 의미에서 단 방향(unidirectional)이다. HTTP/1.0 캐시는 Cache-Control 을 구현하고 있지 않으면 Pragma: no-cache (14.32 절 참조)만을 구현하고 있다는 것에 주의한다.

캐시 지시자는 Request/Response chain 를 따라서 모든 수신측에게 적용할 수 있기 때문에 해당 애플리케이션에서 차지하는 중요도에 관계 없이 프락시나 게이트웨이 애플리케이션은 이 지시자를 반드시 통과시켜야 한다. 특정한 캐시를 위한 cache-directive 를 명시하는 것이 불가능하지는 않다.

```
Cache-Control             = "Cache-Control" ":" 1#cache-directive
cache-directive           = cache-request-directive
                          | cache-response-directive
```

[Page 101]

```
cache-request-directive   = "no-cache" [ "=" <"> 1#field-name <"> ]
                          | "no-store"
                          | "max-age" "=" delta-seconds
```

```
                        | "max-stale" [ "=" delta-seconds ]
                        | "min-fresh" "=" delta-seconds
                        | "only-if-cached"
                        | cache-extension
cache-response-directive = "public"
                        | "private" [ "=" <"> 1#field-name <"> ]
                        | "no-cache" [ "=" <"> 1#field-name <"> ]
                        | "no-store"
                        | "no-transform"
                        | "must-revalidate"
                        | "proxy-revalidate"
                        | "max-age" "=" delta-seconds
                        | cache-extension
cache-extension         = token [ "=" ( token | quoted-string ) ]
```

지시자에 어떠한 1#field-name 파라미터도 없으면 그 지시자는 전체 요구 또는 응답에 적용된다. 지시자에 1#field-name 파라미터가 있으면 그 지시자는 해당 필드 또는 필드들에게만 적용되고 나머지 요구나 응답에는 적용되지 않는다. 이 메커니즘이 확장성을 지원한다. 향후 HTTP 규약의 구현은 HTTP/1.1 에 정의되지 않은 헤더 필드에 이 지시자를 적용할 수도 있다.

Cache-control 지시자는 다음과 같은 일반적인 범주로 분류할 수 있다.

- 캐시할 수 있는 것에 대한 제한: 오직 원서버만이 제한을 둘 수 있다.
- 기본적인 유효일 메커니즘의 변경: 원서버 및 사용자 에이전트 모두가 부과할 수 있다.
- 캐시 검증이나 갱신에 대한 통제: 오직 원서버만이 통제할 수 있다.
- 엔터티의 변형에 대한 통제.
- 캐시 시스템에 대한 확장(extensions)

[Page 102]

14.9.1 무엇을 캐시할 수 있는가

기본적으로 요구 method 의 요구사항, 요구 헤더 필드, 응답 상태가 캐시할 수 있다고 표시하는 응답은 캐시할 수 있다. 13.4 절은 캐시할 수 있는 기본값들에 대하여 요약해 놓았다. 다음의 Cache-Control 응답 지시자는 원서버가 응답의 캐시 가능성을 무시할 수 있도록 한다.

public

보통 비 공유 캐시 내에서만 캐시할 수 있거나 캐시할 수 없지만 어떤 캐시이든 응답을 캐시할 수 있음(cachable)을 표시한다.(추가적인 정보는 14.8절의 Authorization 참조)

private

응답 메시지의 전체 혹은 일부분을 단일 사용자만이 사용하며 절대 공유 캐시(shared cache)에 의해 캐시해서는 안됨을 표시한다. 원서버가 응답의 특정 부분이 단일 사용자만을 위한 것이며 다른 사용자의 요구에 대한 유효한 응답은 아니라는 것을 명시할 수 있도록 한다. 사적인(비 공유) 캐시는 응답을 캐시할 수도 있다.

주의 : 사적이라는 단어 사용의 의미는 응답을 캐시할 수 있는 부분만을 통제하는 것이며 메시지 내용에 대한 보호를 확보할 수는 없다.

no-cache

응답의 전체 혹은 부분을 반드시 캐시해야 함을 표시해야 한다. 원서버가 클라이언트 요구에 낡은 응답(stale response)을 리턴하도록 설정된 캐시에 의해서도 캐시를 하지 못하도록 한다.

주의 : 대부분의 HTTP/1.0 캐시는 이 지침을 인지하지 못하거나 따르지 않을 것이다.

14.9.2 캐시에 의해 무엇을 저장할 수 있는가

저장 금지(no-store) 지시자의 목적은 부주의하게 민감한 정보를 보유(예를 들어 백업 테이프 위에)하거나 배포하는 것을 방지하는 것이다. No-store 지시자는 전체 메시지에 적용되면 응답 및 요구 모두에 발송할 수 있다. 요구 내에 포함하여 발송했으면 캐시는 요구의 어떤 부분 또는 이 요구에 대한 어떠한 응답도 캐시해서는 안 된다. 응답에 발송했으면 캐시는 이 응답의 어떤 부분 또는 응답을 이끌어 낸 요구를 저장해서는 안 된다. 이 지시자는 비 공유 및 공유 캐시에 모두 적용된다. 이 문장의 "저장해서는 안 된다"의 의미는 캐시가 의도적으로 고정 저장 매체(non-volatile storage)에 정보를 저장해서는 안 되며 비 고정 저장 매체에서는 정보를 전송한 후 최대한 빨리 정보를 삭제하도록 최선의 노력을 다해야 한다는 것이다.

[Page 103]

이 지시자가 응답과 관련 되었을 때도 사용자는 명백하게 이 응답을 캐시 시스템 외부에 저장할 수 있다.(예를 들어 "Save As" 대화상자) 기록 버퍼는 이러한 응답을 일반적 작업의 일 부분으로 저장할 수도 있다.

이 지시자의 목적은 특정 사용자의 명시된 필요 조건 및 캐시 데이터 구조체에 예상하지 못한 접속을 통한 우발적인 정보의 유출을 걱정하는 서비스 저작자의 필요 조건을 충족시키는 것이다. 어떤 의미에서 이 지시자의 사용이 사생활 보호를 향상 시켜 줄 수 있지만 이것이 사생활을 보호하는 신뢰하거나 충분한 메커니즘은 아니라는 점에 유의해야 한다. 특히 나쁜 의도를 가지거나 타협적인 캐시는 이 지시자를 인지하지 못하거나 복종하지 않을 수 있다. 또한 통신 네트워크는 정보 유출에 취약한 편이다.

14.9.3 기본적인 만기일 메커니즘의 변경

원서버는 Expires 헤더(14.21 절 참조)를 이용하여 엔터티의 유효 시간을 명시한다. 대안으로 응답에 max-age 지시자를 사용하여 표시할 수도 있다.

응답에 Expires 및 max-age 지시자가 모두 포함되어 있으면 max-age 지시자가 Expires 헤더가 더 제한적이라 할지라도 이를 무시한다. 이 원칙은 원서버가 HTTP/1.0 캐시에 HTTP/1.1 캐시(또는 이후 버전)보다 긴 유효시간을 응답에 부여할 수 있도록 한다. 어떠한 HTTP/1.0 캐시가 동시화되지 않은(desynchronized) 시계 때문에 부적절하게 경과 시간이나 유효 시간을 계산했을 때 유용하다.

주의 : 이 규격을 따르지 않는 대부분의 이전 캐시는 Cache-Control 지시자를 구현하지 않는다. Cache-Control 지시자를 사용하기 원하지만 HTTP/1.1을 따른 캐시를 금지하는 않지만 제한하는 원서버는 max-age 지시자가 Expires 헤더를 무시하며 HTTP/1.1을 따르지 않는 캐시는 max-age 지시자를 준수하지 않는다는 사실을 이용할 수 있다.

다른 지시자는 사용자 에이전트가 기본적인 유효일 메커니즘을 변경할 수 있도록 한다. 이러한 지시자는 요구에 명시할 수 있다.

max-age
클라이언트가 초로 표시된 시간보다 크지 않은 경과 시간을 가진 응답을 기꺼이 접수한다는 것을 표시한다. Max-stale 지시자도 포함되어 있지 않으면 클라이언트는 낡은 응답을 접수할 의사가 없는 것이다.

min-fresh
클라이언트가 신선한 기간이 초로 표시된 현재 의 경과 시간보다 크지 않은 응답을 기꺼이 접수한다는 것을 표시한다. 이는 클라이언트가 최소한 초로 표시된 기간 동안만은 신선한 응답을 원하는 것이다.

[Page 104]

max-stale
클라이언트가 유효시간을 초과한 응답을 기꺼이 접수한다는 것을 표시한다. Max-stale에 값이 부여되었으면 클라이언트는 명시된 초를 초과하지 않는 응답을 기꺼이 접수한다. Max-stale에 값이 부여되지 않았으면 클라이언트는 모든 응답을 기꺼이 접수한다.

캐시가 요구의 max-stale 지시자 응답의 유효 시간을 무시하도록 설정되어 낡은 응답을 리턴하면 캐시는 반드시 Warning 10 (Response is stale)을 이용하여 Warning 헤더를 낡은 응답에 부착하여야 한다.

14.9.4 캐시의 재검증 및 Reload 제어

때때로 사용자 에이전트는 캐시가 원서버에서 캐시를 재검증하거나 원서버에서 캐시 엔트리를 갱신할 것을(원서버로 향한 경로의 다음 캐시만이 아닌) 원하거나 고집할 수 있다. End-to-end 재검증은 캐시나 원서버가 캐시된 응답의 유효 시간을 과대 평가했을 때 필요할 수 있다. End-to-end 갱신은 어떠한 이유 때문에 캐시 엔트리가 오염되었을 때 필요하다.

클라이언트가 자신의 지역 캐시 사본을 가지고 있지 않거나("명시되지 않은 end-to-end 재검증"이라 부른다), 가지고 있을 때("명시된 end-to-end 재검증이라 부른다.)End-to-end 재검증을 요구할 수 있다. 클라이언트는 Cache-Control 지시자를 사용하여 다음의 세 가지 처리를 명시할 수 있다.

End-to-end reload
요구에 "no-cache" Cache-Control 지시자가 포함되어 있거나 HTTP/1.0클라이언트와의 호환성 유지를 위해 "Pragma: no-cache"를 포함하고 있다. 요청의 no-cache 지시자에는 아무런 필드 이름도 포함되지 않는다. 서버는 이러한 요구에 응답할 때 캐시된 사본을 사용해서는 안 된다.

Specific end-to-end revalidation
요구가 원서버로 향한 경로를 따라 각각의 캐시가 자신의 엔트리를 다음 캐시나 원 서버와 재검증하도록 강요하는 "max-age=0" Cache-Control 지시자를 포함하고 있다. 첫 요구는 클라이언트의 현재 검증자와 더불어 캐시-검증 조건을 포함하고 있다.

[Page 105]

Unspecified end-to-end revalidation
요구가 원서버로 향한 경로를 따라 각각의 캐시가 자신의 엔트리를 다음 캐시나 원 서버와 재검증하도록 강요하는 "max-age=0" Cache-Control 지시자를 포함하고 있다. 첫 요구는 클라이언트의 현재 검증자와 더불어 캐시-검증 조건을 포함하고 있지 않다. 해당 자원의 캐시 엔트리를 가지고 있는 경로의 첫 캐시가 현재 검증자와 더불어 캐시-검증 조건을 포함하고 있다

Max-age=0 지시자 때문에 가장 가까운 캐시가 자신의 캐시 엔트리를 재검증하도록 강요 받았을 때 클라이언트는 요구에 자신의 검증자를 제공하며 제공된 검증자는 캐시 엔트리에 현재 저장된 검증자와

상이할 수 있다. 이 경우 캐시는 의미 투명성에 영향을 미치지 않고 자신의 요구를 만드는 데 두 검증자 모두를 사용할 수 있다.

그러나 검증자의 선택이 성능에 영향을 미칠 수 있다. 최상의 접근법은 가장 가까운 캐시가 요구를 만들 때 자기 자신의 검증자를 사용하는 것이다. 서버는 304 (Not Modified)로 응답하고 캐시는 새롭게 검증된 사본을 클라이언트에게 200 (OK) 응답과 함께 되돌려 주어야 한다. 서버가 새로운 엔터티나 캐시 검증자로 응답해도 가장 가까운 캐시는 강한 비교 기능(strong comparison function)을 이용하여 클라이언트의 요구가 제공하는 검증자와 리턴 된 검증자를 비교해야 한다. 클라이언트 검증자가 원서버의 검증자와 동등할 때는 가장 가까운 캐시는 304 (Not Modified)를 리턴한다. 그렇지 않으면 200 (OK) 응답으로 새로운 엔터티를 리턴한다.

요구에 no-cache 지시자가 포함되어 있으면 요구는 min-fresh, max-stale, 또는 max-age 를 포함해서는 안 된다.

네트워크 연결이 극도로 약할 때와 같은 경우에 클라이언트는 원서버와 갱신하거나 재검증하는 것이 아닌 현재 저장하고 있는 응답만을 리턴하기 위해 캐시를 원할 수 있다. 이를 위해서 클라이언트는 요구에 only-if-cached 지시자를 포함할 수 있다. 클라이언트가 이 지시자를 수신하면 캐시는 다른 응답의 제한 사항과 일치하는 캐시 엔트리를 사용하여 응답하던지 504 (Gateway Timeout) 상태로 응답할 수 있다. 그러나 캐시의 그룹을 안정된 내부 연결로 통합된 시스템에 사용할 때 이러한 요구는 해당 캐시 그룹 내에서 전달될 수 있다.

캐시가 서버에서 명시된 유효 시간을 무시하도록 설정될 수 있기 때문에 또한 클라이언트 요구가 max-stale 지시자를 포함할 수 있기 때문에(유사한 영향을 미친다) 규약은 원서버가 계속되는 캐시 사용에 대한 캐시 엔트리 검증을 요구할 수 있는 메커니즘을 포함하고 있다.

[Page 106]

Must-revalidate 지시자가 캐시가 수신한 응답에 포함되어 있고 캐시가 계속되는 요구에 응답하기는 낡아진 이후에 캐시는 먼저 원서버에 이를 재검증하기 전에는 엔트리를 사용해서는 안 된다. (예를 들어 캐시는 전적으로 원서버의 Expires 또는 max-age 값에 기초하여 캐시된 응답이 낡았으면 매번 end-to-end 검증을 실행해야 한다.)

Must-revalidate 지시자는 특정 규약 기능의 안정된 운영을 위해서 필요하다. 어떠한 경우이든 HTTP/1.1 캐시는 must-revalidate 지시자를 반드시 따라야 한다. 특히 캐시가 어떠한 이유이든 원서버에 도달할 수 없을 때는 반드시 504 (Gateway Timeout) 응답을 생성해야 한다.

서버는 아무런 표시 없이 실행되지 않은 재무 트랜잭션의 경우처럼 엔터티에 대한 재검증이 실패하여 부정확한 운영을 초래할 경우 반드시 must-revalidate 지시자를 발송해야 한다. 수신측은 결코 이 지시자를 위반하는 어떠한 자동화된 처리 방식을 갖고 있어서는 안 되며 재검증이 실패할 경우 자동적으로 검증되지 않은 엔터티 사본을 제공해서는 안 된다.

권하지는 않지만 극도로 악화된 연결 상태를 이용하는 사용자 에이전트는 이 지시자를 위반할 수는 있으나 사용자에게 반드시 검증되지 않은 응답을 제공했음을 명백하게 경고해야 한다. 경고는 검증되지 않는 접속 각각에 제공해야 하며 명백한 사용자 정보를 제공해야 한다.

Proxy-revalidate 지시자는 비 공유 사용자 에이전트 캐시에는 적용되지 않는다는 점을 제외하고는 must-validate 지시자와 동일한 의미를 갖고 있다. 사용자의 캐시가 응답을 저장하고 나중에 그것을 검증할 필요 없이 리턴할 수 있도록(그 사용자가 먼저 인증을 받았기 때문에) 하면서도 프락시에게 많은

사용자가 재검증하도록 요구하여(각 사용자가 인증되었음을 확실하게 하기 위해) 인증되지 않은 요구에 대한 응답으로 사용할 수 있다.

14.9.5 비 변경 지시어(No-Transform Directive)

가장 가까운 캐시의 구현자(프락시)는 특정 엔터티 본문의 media type 을 변환하는 것이 유용함을 발견할 수 있다. 예를 들어 프락시는 캐시 공간을 절약하거나 느린 링크 상의 트래픽 양을 줄이기 위해 이미지의 포맷을 변환할 수 있다. HTTP 는 오늘날까지 이러한 변환(transformations)에 대해서는 침묵을 지키고 있다.

[Page 107]

벌써 이러한 변환을 특정 종류의 애플리케이션에 사용할 엔터티 본문에 적용했을 때 심각한 운영 문제가 발생하였다. 예를 들어 의료 이미지 처리, 과학적 자료 분석 및 end-to-end 인증에 사용되는 애플리케이션은 모두 원서버의 entity-body 와 비트 단위까지 동일한 엔터티 본문을 수신하는 방식에 의존하고 있다.

따라서 응답이 no-transform 지시자를 포함하고 있으면 가장 가까운 캐시나 프락시는 13.5.2 절에 열거된 이러한 헤더들은 no-transform 지시자에 종속적이기 때문에 이들을 절대로 변경해서는 안 된다. 이것은 캐시 또는 프락시는 이러한 헤더가 명시한 어떠한 측면의 entity-body 도 변경하지 말아야 한다는 것을 의미한다.

14.9.6 캐시 제어 확장

Cache-Control 헤더 필드는 하나 또는 그 이상의 cache-extension 토큰을 이용하여 각각 선택적으로 부여된 값을 가지고 확장할 수 있다. 정보 확장(Informational extensions - 캐시 행태에 변화를 요구하지 않는)은 다른 지시자의 의미를 변화시키지 않고도 추가할 수 있다. 행태 확장(behavioral extensions)은 캐시 지시자의 기본 베이스에 대한 변경자의 역할을 수행하도록 디자인되었다. 새로운 지시자 및 표준 지시자 모두가 제공되어 새로운 지시자를 이해하지 못하는 애플리케이션은 표준 지시자가 명시한 행태에 기본적으로 따르며 새로운 지시자를 이해하는 애플리케이션은 이를 표준 지시자와 관련된 필요 조건의 변경으로 인식한다. 이러한 방식으로 지시자를 기본 규약에 대한 변경을 요구하지 않고도 확장할 수 있다.

확장 메커니즘은 원초 HTTP 버전에 정의된 모든 지시자와 특정 확장에는 따르지만 이해할 수 없는 모든 지시자를 무시하는 HTTP 캐시에 달려 있다.

예를 들어 "private" 지시자의 변경자 역할을 수행하는 "community"로 불리는 가설의 새로운 응답 지시자를 가정하자. 우리는 새로운 지시자를 모든 비 공유 캐시에 대한 추가로 값 내에 이름이 등록된 공동체 구성원만이 공유하는 응답에 대한 캐시를 의미하는 것으로 규정한다. "UCI" community 를 공유된 캐시의 private 응답에 사용하길 원하는 원서버는 다음을 포함하여 이를 수행할 수 있다.

```
Cache-Control: private, community="UCI"
```

이 헤더 필드를 만난 캐시는 캐시가 "community" cache-extension 을 이해할 수 없더라도 "private" 지시자를 보고 이해할 수 있어 안전한 행태의 기본 행태로 전환할 수 있기 때문에 정확하게 작동한다.

[Page 108]

인지할 수 없는 cache-directive 는 무시해야 한다. HTTP/1.1 캐시가 인지하지 못하는 모든 cache-directive 는 캐시가 확장을 이해하지 못하더라도 최소한도로 이러한 캐시 행태가 정확한 것으로 유지되도록 표준 지시자(또는 응답의 기본 캐시 가능성(chchability))와 결합되어 있다고 가정한다.

14.10 Connection

Connection 일반 헤더 필드는 발송측이 특정 연결이 원하는 선택 사항을 명시하는 데 사용하며 추가적인 연결 시 프락시를 통하여 통신해서는 절대 안 된다.
Connection 헤더는 다음과 같은 문법을 가지고 있다.

```
Connection-header      = "Connection" ":" 1#(connection-token)
connection-token       = token
```

HTTP/1.1 프락시는 메시지가 전송되기 전에 Connection 헤더를 반드시 분석하여야 하며 이 필드의 각각의 connection-token 에 대해 connection-token 과 동일한 이름을 가진 메시지에서 모든 헤더 필드를 삭제해야 한다. Connection 선택 사항은 해당 연결 선택 사항과 관련된 파라미터가 없으면 추가적인 헤더 필드가 발송되지 않기 때문에 관련 추가 헤더 필드가 아닌 Connection 헤더 필드에 connection-token 의 존재로 알 수 있다. HTTP/1.1 은 "close" 연결 선택 사항을 송신측이 응답을 완전히 받은 후에 연결이 종료된다는 것을 표시하는 데 사용한다.

예를 들어,

```
Connection: close
```

위와 같이 close 옵션이 요구나 응답 헤더 필드에 있으면 연결이 현재 의 요구/응답에 완성된 후에 'persistent' (8.1 절)로 간주되어서는 안 된다는 것을 표시한다.

persistent 연결을 지원하지 않는 HTTP/1.1 응용은 반드시 모든 메시지에 "close" 연결 선택 사항을 포함하고 있어야 한다.

14.11 Content-Base

Content-Base 엔터티 헤더 필드는 엔터티 내의 상대 URL 을 찾아내는 데 사용한다. 이 헤더 필드는 RFC 1808 에 Base 로서 기술되어 있으며 곧 개정될 것이다.

```
Content-Base           = "Content-Base" ":" absoluteURI
```

Content-Base 필드가 없으면 엔터티의 기본 URI 는 Content-Location (Content-Location URI 가 절대 URI 이면) 또는 요구를 시작하는 데 사용한 URI 에 의하여 동일한 우선 순위로 규정된다.

[Page 109]

그러나 entity-body 내 내용의 기본 URI 는 해당 entity-body 내에서 재규정될 수 있다는 점에 주의한다.

14.12 Content-Encoding

Content-Encoding entity-header 필드는 entity-body 에 대한 변경자로 사용한다. 이것이 있으면 그 값은 entity-body 에 어떠한 추가 Content coding 이 적용되었는지 표시하여 Content-Type 헤더 필드가 참조하는 media-type 을 얻기 위하여 어떠한 디코딩 메커니즘을 적용해야 하는지 표시한다.

Content-Encoding 은 주로 문서를 기저의 media type 의 정체(identity)를 상실하지 않고도 압축할 수 있도록 하는 데 사용한다.

```
Content-Encoding        = "Content-Encoding" ":" 1#content-coding
```

Content 코딩은 3.5 절에 규정되어 있다. 이의 사용 예는,

```
Content-Encoding: gzip
```

Content-Encoding 은 Request-URI 가 식별하는 엔터티의 특징이다. 전형적으로 entity-body 는 이 인코딩에 저장되며 표시 또는 유추 목적으로 사용하기 전에만 해독할 수 있다.

엔터티에 복수의 인코딩을 적용했으면 내용 코딩은 적용된 순서로 열거해야 한다.
인코딩 파라미터에 관한 추가 정보는 이 규격에서 규정하지 않은 entity-header 필드에서 제공한다.

14.13 Content-Language

Content-Language entity-header 필드는 포함하고 있는 엔터티 대상 청중자의 자연적인 언어를 기술하고 있다. Entity-body 내에서 사용된 모든 언어와 동등하지 않을 수도 있다는 점에 주의 한다.

```
Content-Language        = "Content-Language" ":" 1#language-tag
```

Language 태그는 3.10 절에 정의되어 있다. Content-Language 의 주된 목적은 사용자가 사용자 자신이 선호하는 언어에 따라 엔터티를 식별하거나 구별할 수 있도록 하는 것이다. 따라서 본문 내용이 덴마크어를 이해할 수 있는 청중을 위한 것이라면 적절한 필드는 다음과 같다.

```
Content-Language: da
```


Content-Language 가 명시되어 있지 않으면 기본은 내용이 모든 언어의 청중을 위한 것이다. 이는 송신측이 이것이 특정 자연적 언어에 한정적인 것이 아니거나 사용하고 하는 언어를 알 수 없다는 것을 의미한다.

복수 언어는 복수의 청중을 위한 내용을 열거할 수 있다.
예를 들어 "Treaty of Waitangi,"의 번역을 마오리(Maori)어 버전 및 영어 버전으로 번역하려면,

```
Content-Language: mi, en
```

그러나 엔터티에 복수의 언어가 존재한다는 것이 여러 언어를 사용할 수 있는 청중을 위한 것을 의미하는 것은 아니다. 이의 예는 "A First Lesson in Latin" 과 같은 초급자 언어 입문서이다. 이는 분명히 영어를 사용하는 청중을 위한 것이다. 이 경우 Content-Language 는 "en"만을 포함해야 한다.

Content-Language 는 모든 media type 에 적용할 수 있고 텍스트 문서에 한정된 것은 아니다.

14.14 Content-Length

Content-Length entity-header 필드는 message-body 의 크기를 수신측에 발송된 octets 의 십진수, HEAD method 의 경우에는 요구가 GET 이었을 경우 발송되었을 entity-body 의 크기를 표시한다.

Content-Length = "Content-Length" ":" 1*DIGIT

예는,
Content-Length: 3495

애플리케이션은 엔터티의 media type 에 관계 없이 이 필드를 전송하는 message-body 의 크기를 표시하는 데 사용한다. 수신측이 신뢰성 있게 entity-body 를 포함하고 있는 HTTP/1.1 요구의 종료를 결정할 수 있어야 한다. 예를 들어 요구가 유효한 Content-Length 필드를 가지고 Transfer-Encoding:chunked 또는 multipart body 를 사용할 수 있기 때문이다.

제로보다 크거나 동등한 모든 Content-Length 는 유효한 값이다. 4.4 절은 Content-Length 가 주어지지 않았을 때 message-body 의 길이를 결정하는 방법을 기술하고 있다.

[Page 111]

주의 : 이 필드의 의미는 해당되는 "message/external-body" content-type에서 사용되는 선택 필드인 MIME 규정과는 상당히 다르다. HTTP에서 메시지의 길이를 전송하기 전에 결정할 수 있으면 언제나 이 필드를 발송해야 한다.

14.15 Content-Location

Content-Location entity-header 필드는 메시지에 포함된 엔터티의 자원 위치를 제공하는 데 사용한다. 자원이 자신과 관련된 복수의 엔터티를 가지고 있고 각 엔터티가 사실상 개별적으로 접근하였을 때 구별된 위치를 가지고 있는 경우에는 서버는 리턴되는 특정 변형자(variant)에 대한 Content-Location 을 제공해야 한다. 또한 서버는 응답 엔터티에 상응하는 자원의 Content-Location 를 제공해야 한다.

Content-Location = "Content-Location" ":" (absoluteURI | relativeURI)

Content-Base 헤더 필드가 없으면 Content-Location 의 값은 엔터티의 URL 을 규정한다.(14.11 절 참조)

Content-Location 값은 원래 요청된 URI 의 대체물이 아니다. 이것은 요구를 방송한 시점의 특정 엔터티에 상응하는 자원의 위치를 표현하는 것일 뿐이다. 이후의 요구는 해당 특정 엔터티의 자원을 식별하는 것이 목적이라면 Content-Location URI 를 사용할 수 있다.

캐시는 Content-Location 을 조회하는 데 사용되는 URI 와 다른 Content-Location 을 가진 엔터티를 해당 Content-Location URI 의 추후 요구에 대한 응답에 사용할 수 있다고 가정해서는 안 된다. 그러나 Content-Location 은 13.6 절에서 기술한 대로 단일 요청 자원에서 조회한 복수의 엔터티를 차별화하는 데 사용할 수 있다.

Content-Location 이 상대적인 URI 이면 응답에서 제공하는 어떠한 Content-Location URI 에 상대적인 것으로 해석해야 한다. 아무런 Content-Base 도 제공되지 않았으면 상대적인 URI 은 Request-URI 에 상대적인 것으로 해석해야 한다.

[Page 112]

14.16 Content-MD5

RFC 1864 [23]에 규정된 바와 같이 entity-body 의 end-to-end 메시지 무결성을(end-to-end message integrity check (MIC)) 점검하기 위한 Content-MD5 entity-header 필드는 entity-body 의 MD5 digest 이다.(주의: MIC 는 전송되는 도중의 entity-body 에 대한 우발적인 변경을 탐지하는 데 유용하지만 악의적인 공격에 대한 증명은 아니다.)

```
Content-MD5   = "Content-MD5" ":" md5-digest
md5-digest    = <base64 of 128 bit MD5 digest as per RFC 1864>
```

Content-MD5 헤더 필드는 원서버가 entity-body 의 무결성을 확인하는 기능으로서 생성할 수 있다. 원서버만이 Content-MD5 헤더 필드를 생성할 수 있으며 프락시나 게이트웨이는 그 값을 end-to-end 무결성 점검으로 사용하지 못하도록 변질시킬 수 있기 때문에 절대도 이것을 생산하면 안 된다. 프락시나 게이트웨이를 포함한 어떠한 entity-body 의 수신측도 이 헤더 필드의 digest value 가 수신된 entity-body 의 digest value 와 일치하는지 점검할 수 있다.

MD5 digest 는 적용된 모든 Content-Encoding 을 포함하지만 message-body 에 적용되었을 수 있는 모든 Transfer-Encoding 은 포함하지 않는 entity-body 의 내용에 기초하여 산출할 수 있다. 수신된 메시지에 Transfer-Encoding 이 포함되어 있으면 해당 인코딩은 수신된 엔터티에 대한 Content-MD5 값을 점검하기 이전에 삭제하여야 한다. 이것은 digest 가 Transfer-Encoding 을 적용하지 않고 발송했을 때의 entity-body 와 정확하게 동일한 entity-body 의 octets 에서 산출되는 결과를 초래한다.

HTTP 는 RFC 1864 를 확장하여 digest 가 MIME 복합 media-type (예를 들어 multipart/* 및 message/rfc822)에서 산출될 수 있도록 허용한다. 그러나 이것이 이전 문장에서 규정한 digest 산출 방법을 변경하지는 않는다.

주의 : 이것은 몇몇 결과를 초래한다. 복합 유형의 entity-Body는 자신의 MIME 및 HTTP 헤더에 복수의 body-parts를 가질 수 있다.(Content-MD5, Content-Transfer-Encoding 및 Content-Encoding 헤더 포함) 만약 body-part 가 Content-Transfer-Encoding 또는 Content-Encoding 헤더를 가지고 있으면 body-part의 내용이 인코딩되었고 body-part가 현재 처럼(예를 들어 적용 이후) Content-MD5 digest에 포함되어 있다고 가정할 수 있다.

주의 : Content-MD5 규정이 RFC 1864의 MIME entity-bodies에서와 규정과 동일하기는 하지만 Content-MD5를 HTTP entity-bodies에 적용하는 것이 MIME entity-bodies에 적용하는 것과 다를 수 있는 몇 가지 경우가 있다.

[Page 113]

그 중의 하나가 MIME 과는 달리 HTTP 는 Content-Transfer-Encoding 을 사용하지 않지만 Transfer-Encoding 및 Content-Encoding 을 사용한다는 것이다. 다른 것은 HTTP 가 MIME 보다 더 자주 이진 내용 유형을 사용하기 때문에 digest 를 산출하는 데 사용된 바이트 순서는 해당 유형에 정의된 전송 바이트 순서라는 점에 주의할 필요가 있다. 마지막으로 HTTP 가 CRLF 로 정규화된 폼뿐만 아니라 어떠한 복수 라인 줄바꿈 관례에 따른 텍스트 유형이든 전송을 허용한다는 것이다. 실제로 전송된 텍스트에 사용된 줄바꿈(line break) 관례는 digest 를 산출할 때 변경하지 말아야 한다.

14.17 Content-Range

Content-Range entity-header 는 부분적 entity-body 와 함께 전송하여 전체 entity-body 의 어느 부분에 부분적 본문을 삽입해야 하는 가를 명시한다. 또한 이것은 전체 entity-body 의 크기를 표시하기도

한다. 서버가 클라이언트에게 부분적 응답을 리턴했을 때 서버는 응답이 차지하는 영역의 범위 및 전체 entity-body 의 길이를 기술해야 한다.

```
Content-Range              = "Content-Range" ":" content-range-spec
content-range-spec         = byte-content-range-spec
byte-content-range-spec    = bytes-unit SP first-byte-pos "-" last-byte-pos "/" entity-length
entity-length              = 1*DIGIT
```

Byte-ranges-specifier 값과는 달이 byte-content-range-spec 은 하나의 영역만을 명시할 수 있으며 영역의 처음 및 마지막 바이트의 절대 위치를 포함하고 있어야 한다.

Byte-content-range-spec whose Last-byte-pos 값이 first-byte-pos 값보다 적은 byte-content-range-spec 이나 entity-length 값이 last-byte-pos 값보다 적거나 동등한 것은 무효이다. 유효하지 않은 byte-content-range-spec 의 수신측은 이것과 이에 따라 전송되는 모든 내용을 반드시 무시해야 한다.

[Page 114]

엔터티가 전체 1234 바이트를 포함하고 있다고 가정하면 byte-content-range-spec 값의 예는 다음과 같다.

- The first 500 bytes: bytes 0- 499/1234
- The second 500 bytes: bytes 500- 999/1234
- All except for the first 500 bytes: bytes 500-1233/1234
- The last 500 bytes: bytes 734-1233/1234

HTTP 메시지가 단일 영역의 내용을 포함하고 있을 때 이 내용은 Content-Range 헤더 및 실제적으로 전송되는 바이트 수를 표시하는 Content-Length 헤더와 함께 전송된다.

예를 들면,

```
HTTP/1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021/47022
Content-Length: 26012
Content-Type: image/gif
```

HTTP 메시지가 복수 영역의 내용을 포함하고 있을 때(예를 들어 복수의 중첩되지 않는 영역에 걸친 요구에 대한 응답) 이 영역들은 multipart MIME 메시지로서 전달된다. 이 목적에 사용된 multipart MIME content-type 은 이 규격에서"multipart/byteranges"로 규정하고 있다. 규정에 관하여는 부록 19.2 절을 참조한다.

MIME multipart/byteranges 메시지를 해독할 수 없는 클라이언트는 단일 요구의 복수 byte-ranges 요청해야 한다.

클라이언트가 단일 요구에 복수의 byte-ranges 를 요청하면 서버는 요구에 나타난 순서 대로 이들을 되돌려 주어야 한다.

[Page 115]

서버가 byte-range-spec 이 무효이기 때문에 무시했으면 서버는 요구를 유효하지 않은 Range 헤더 필드가 존재하지 않는 것처럼 처리해야 한다.(대개의 경우 이것은 전체 엔터티를 포함하고 있는 200 메시지를 리턴함을 의미한다.) 이유는 클라이언트가 이러한 무효 요구를 하는 유일한 시간은 엔터티가 이전 유구에 의해 수신된 엔터티보다 작을 때이기 때문이다.

14.18 Content-Type

Content-Type entity-header 필드는 수신측에 발송한 entity-body 의 media type 을 표시하거나, 요구가 GET 이었으면 발송되었을 media type 를 표시한다.

```
Content-Type            = "Content-Type" ":" media-type
```

Media types 은 3.7 장에 규정되어 있으며 이 필드의 사용 예는 다음과 같다.

```
Content-Type: text/html; charset=ISO-8859-4
```

엔터티의 media type 을 식별하는 method 에 관한 토의는 7.2.1 절에 기술되어 있다.

14.19 Date

Date general-header 필드는 메시지가 생성되었을 때의 날짜와 시간을 표시하며 RFC 822 의 org-date 와 동일한 의미를 가진다. 필드 값은 3.3.1 절에 기술된 것처럼 HTTP-date 이다.

```
Date            = "Date" ":" HTTP-date
```

이의 사용 예는,

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
```

사용자 에이전트(요구의 경우)나 원서버(응답의 경우)와 직접적인 접속을 통하여 메시지를 수신하면 날짜는 수신측 끝의 현재 시간인 것으로 가정한다. 그러나 날짜가 캐시 응답을 평가하는 데 중요하기 때문에(원서버가 그렇다고 믿기 때문에) 원서버는 모든 응답에 Date 헤더 필드를 반드시 포함해야 한다. 클라이언트는 PUT 및 POST 요청의 경우처럼 entity-body 를 포함하고 있는 메시지의 Date 헤더 필드만을 발송해야 하기는 하지만 선택 사항이기도 하다. Date 헤더 필드를 가지고 있지 않은 수신 메시지는 수신측이 메시지를 캐시하거나 Date 를 요구하는 규약을 이용한 게이트웨이를 통과할 때 수신측이 하나를 지정해야 한다.

[Page 116]

이론상으로 날짜는 엔터티가 생성되지 바로 직전 순간을 표시해야 한다. 그러나 실제상으로 날짜는 의미 값에 영향을 미치지 않고 메시지 원문을 생성하는 동안 아무 시간에서나 생성될 수 있다.

Date 의 포맷은 3.3 절의 HTTP-date 가 규정하는 절대 날짜 및 시간이다. 반드시 RFC1123 [8]-날짜 포맷으로 발송해야 한다.

14.20 ETag

ETag entity-header 필드는 관련된 엔터티의 엔터티 태그를 정의한다. 엔터티 태그와 함께 사용하는 헤더는 14.20, 14.25, 14.26 및 14.43 절에 기술되어 있다. 엔터티 태그는 동일한 자원(13.3.2 절 참조)의 다른 엔터티와 비교하는 데도 사용할 수 있다.

ETag = "ETag" ":" entity-tag

예:
ETag: "xyzzy“
ETag: W/"xyzzy“
ETag: ""“

14.21 Expires

Expires entity-header 필드는 그 시간 이후 응답이 낡았다고 간주해야 하는 날짜/날짜를 제공한다. 캐시(프락시 캐시 또는 사용자 에이전트 캐시)는 대개 먼저 원서버(또는 엔터티의 신선한 복사본을 가지고 있는 가장 가까운 캐시)가 검증하지 않는 한 낡은 캐시 엔트리를 리턴하지 않는다. 유효일 모델에 관한 추가 논의는 13.2 절을 참조한다.

Expires 필드가 존재한다는 것이 그 시간 이전 또는 이후에 원래의 자원이 변경되거나 사라진다는 것을 의미하지는 않는다.

포맷은 3.3 절에서 정의한 HTTP-date 절대 날짜와 시간이다. 반드시 RFC1123-date 포맷이어야 한다.
Expires = "Expires" ":" HTTP-date

[Page 117]

이의 사용 예는 다음과 같다.
Expires: Thu, 01 Dec 1994 16:00:00 GMT

주의 : 응답이 max-age 지시자를 포함한 Cache-Control 필드를 포함하고 있으면 그 지시자는 Expires 필드를 무시한다.

HTTP/1.1 클라이언트와 캐시는 반드시 다른 유효하지 않는 날짜 포맷을, 특히 "0" 값을 포함하고 있는 날짜 포맷을 지나간 날짜로 취급해야 한다.(예를 들면 "벌써 만료된"으로)응답을 "벌써 만료된" 것으로 표시하기 위해서 원서버는 Expires 날짜를 Date 헤더 필드와 동일한 것으로 사용해야 한다.(13.2.4 절의 유효일 계산 원칙을 참조)응답을 "결코 만료되지 않는" 것으로 표시하기 위해서 원서버는 Expires 날짜를 대략 응답이 발송된 후 시점부터 1 년 후를 지정한다. HTTP/1.1 서버는 향후 1 년 이상 된 Expires 날짜를 발송하지 말아야 한다.

기본적으로 캐시할 수 없는 응답에 미래의 특정 시간의 시간 값과 함께 Expires 헤더 필드가 존재하면 Cache-Control 헤더 필드가(14.9 절 참조) 다른 식으로 표시하지 않는 한 응답을 캐시할 수 있다는 것을 표시한다.

14.22 From

From request-header 필드는, 존재한다면, 요청 사용자 에이전트를 통제하는 인간 사용자의 인터넷 전자우편 주소를 포함하고 있어야 한다. 주소는 RFC 822 의 우편함 (as updated by RFC 1123 에 의하여 갱신된 것처럼)이 규정한 것처럼 기계가 사용할 수 있는 것이어야 한다.

From = "From" ":" mailbox

사용 예는,

From: webmaster@w3.org

이 헤더 필드는 로깅(logging) 목적이나 무효이거나 원하지 않는 요구의 출처를 확인하는 수단으로 사용할 수 있다. 접속 금지의 불확실한 폼으로 사용해서는 안 된다. 이 필드는 요구가 주어진 사람(수행된 method 에 대한 책임을 지는 사람)을 대신하여 수행되고 있는 것으로 해석한다. 특히 로봇 에이전트는 이 헤더를 포함하여 수신측 끝에서 문제가 발생하였을 때 로봇을 운영하는 책임을 진 사람과 연락할 수 있도록 해야 한다.

[Page 118]

이 필드의 인터넷 전자우편 주소는 요구를 발송한 인터넷 호스트와 구별될 수 있다. 예를 들어 요구가 프락시를 통과할 경우 원서버의 주소를 사용할 수 있다.

주의 : 클라이언트는 사용자의 동의 없이는 그것이 사용자의 사생활 보호나 사이트의 보안 정책과 충돌할 수 있기 때문에 From 헤더 필드를 발송해서는 안 된다. 사용자는 요구를 발송하기 전 어떤 시점에라도 이 필드의 값을 무력화, 활성화 및 변경할 수 있도록 할 것을 강력히 추천한다.

14.23 Host

Host request-header 필드는 사용자나 참조하고자 하는 자원(보통 3.2.2 절에 기술한 HTTP URL)이 부여한 원래의 URL 에서 얻은 대로 요구 받고 있는 자원의 인터넷 호스트와 포트 숫자를 명시한다. Host 필드 값은 반드시 원서버나 원래의 URL 이 부여한 게이트웨이 네트워크 위치를 표시해야 한다. 이것은 원서버나 게이트웨이가 단일 IP 주소의 복수 호스트 이름에 사용되는 호스트의 루트 "/" URL 과 같이 내부적으로 모호한 URL 을 구별할 수 있도록 한다.

```
Host            = "Host" ":" host [ ":" port ]     ; 섹션3.2.2
```

뒤 따르는 포트 정보가 없는 "호스트"는 요구된 서비스의 기본 포트를 의미한다.(HTTP URL 의 "80"). 예를 들어 원서버의 <>에 대한 요청은 반드시 다음을 포함해야 한다.

```
GET /pub/WWW/ HTTP/1.1
Host: www.w3.org
```

클라이언트는 인터넷 상의 모든 HTTP/1.1 요구 메시지에 Host 헤더 필드를 반드시 포함해야 한다. 기존에 Host 필드가 존재하지 않으면 HTTP/1.1 프락시는 요구를 인터넷 상에서 전송하기 전에 요구 메시지에 Host 필드를 반드시 추가해야 한다. 인터넷을 기반을 둔 모든 HTTP/1.1 서버는 Host 헤더 필드가 없는 모든 HTTP/1.1 요구 메시지에 대하여 400 상태 코드로 반드시 응답해야 한다. Host 와 관련된 다른 필요 조건 사항에 대해서는 5.2 및 19.5.1 절을 참조한다.

14.24 If-Modified-Since

If-Modified-Since request-header 필드는 GET method 와 함께 사용하여 GET method 를 조건적으로 만든다. 요구된 변형자가 이 필드에 명시된 시간 이후에 변경되지 않았으면 엔터티는 서버로부터 리턴되지 않는다. 대신 304 (not modified) 응답이 message-body 없이 리턴 될 것이다.

[Page 119]

```
If-Modified-Since       = "If-Modified-Since" ":" HTTP-date
```

이 필드의 사용 예는,

If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

If-Modified-Since 헤더는 있지만 Range 헤더가 없는 GET method 는 식별된 엔터티가 If-Modified-Since 헤더에 표시된 날짜 이후로 변경되었을 때만 전송되도록 요구한다. 이것을 결정하는 알고리즘은 다음 경우의 수를 포함한다,

a) 요청이 보통 200(OK) 상태 이외의 것을 산출하거나 전달된 If-Modified-Since 날짜가 무효이면 응답은 일반적인 GET와 완전히 동일하다. 서버의 현재 시간보다 이후인 날짜는 유효하지 않다.

b) 변형자가 If-Modified-Since 날짜 이후에 변경되었으면 응답은 일반적인 GET와 완전히 동일하다.

c) 변형자가 유효한 If-Modified-Since 날짜 이후 변경되지 않았으면 서버는 반드시 304 (Not Modified) 응답을 리턴해야 한다.

이 기능의 목적은 트랜잭션 오버헤드(transaction overhead)를 최소화하면서 캐시된 정보를 효과적으로 갱신할 수 있도록 하는 것이다.

Range request-header 필드는 If-Modified-Since 의 의미를 변경한다는 점에 주의한다. 전체적인 내용은 14.36 절을 참조한다. 클라이언트와 동시화되지 않은 시계를 가진 서버가 If-Modified-Since 시간을 해석할 수 있음을 주의해야 한다.

클라이언트가 동일한 요구에서 If-Modified-Since 헤더를 가져 오는 대신 If-Modified-Since 헤더에 자의적인 날짜를 사용하였으면 클라이언트는 이 날짜가 서버의 시간 해석 방식에 의해 해석된다는 사실을 인지해야만 한다는 것에 주의해야 한다. 클라이언트는 동시화되지 않는 시계 및 클라이언트와 서버의 사이의 각기 다른 시간 인코딩으로 인한 반올림을 고려해야 한다. 이것은 처음 요구한 시간과 계속되는 요구의 If-Modified-Since 날짜 사이에서 문서가 변경되었을 경우 경쟁 상황(race conditions)이 발생할 가능성 및 If-Modified-Since 날짜를 클라이언트의 시계에서 서버 시계와 연결 없이 추출하였을 경우 틀린 시계와 관련된(clock-skew-related) 문제가 발생할 가능성을 모두 포함한다. 클라이언트와 서버 사이의 틀린 시간의 교정은 기껏해야 네트웍의 잠복기(network latency) 때문에 근사치일 뿐이다.

[Page 120]

14.25 If-Match

If-Match request-header 필드는 method 와 함께 사용하여 method 를 조건적으로 만든다. 이전에 자원에서 획득한 하나 또는 그 이상의 엔터티를 가진 클라이언트는 연관된 엔터티 태그의 목록을 If-Match 헤더 필드에 포함하여 이러한 엔터티 중의 하나가 현재의 것임을 증명할 수 있다. 이 기능의 목적은 트랜잭션 오버헤드(transaction overhead)를 최소화하면서 캐시된 정보를 효과적으로 갱신할 수 있도록 하는 것이다. 또한 요구를 갱신할 때 자원의 잘못된 버전에 대한 부주의한 변경을 방지하는 데 사용할 수 있다. 특별한 경우로 "*" 값은 자원의 모든 현재 엔터티와 일치한다.

If-Match = "If-Match" ":" ("*" | 1#entity-tag)

해당 자원에 유사한 GET 요구(If-Match 헤더 없이)에 대한 응답으로 리턴되었을 수 있는 엔터티의 엔터티 태그와 일치하는 어떠한 엔터티 태그나 "*"이 주어지고 해당 자원에 대한 현재 의 엔터티가 존재한다면 서버는 If-Match 헤더 필드가 존재하지 않는 것처럼 요구 받은 method 를 수행할 것이다.

서버는 If-Match 의 엔터티 태그를 비교하기 위하여 반드시 강한 비교 기능(strong comparison function (3.11 절 참조))을 사용해야 한다.

아무런 엔터티 태그도 일치하지 않거나 "*" 이 주어졌는데도 아무런 현재의 엔터티가 존재하지 않으면 서버는 요구 받은 method 를 절대 수행해서는 안되고 반드시 412 (Precondition Failed) 응답을 리턴해야 한다. 이 행태는 클라이언트가 PUT 과 같은 갱신 method 가 클라이언트가 마지막으로 조회한 이후 변경된 자원을 다시 변경하지 못하도록 한다.

If-Match 헤더 필드 없이 요구가 2xx 상태 이외의 어떤 것이라도 초래하게 되면 If-Match 를 무시해야 한다.

"If-Match: *" 의 의미는 원서버(또는 Vary 메커니즘을 이용한 캐시, 14.43 절 참조)가 선택한 표시 방법이 존재하면 method 를 반드시 수행해야 하고 그렇지 않다면 절대 수행해서는 안 된다는 것이다.

[Page 121]

자원을 갱신할 목적의 요구(예를 들어 PUT)는 If-Match 값(단일 엔터티 태그)에 상응하는 엔터티가 더 이상 해당 자원을 표시하는 않는다면 요구 method 를 절대로 적용해서는 안 된다는 것을 표시하기 위하여 If-Match 헤더 필드를 포함할 수 있다. 이것은 사용자가 자신이 인지하지 못하는 동안 자원이 변경되었으면 요구가 완료되는 것을 바라지 않음을 표시할 수 있도록 한다.

예를 들면,

```
If-Match: "xyzzy“
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz“
If-Match: *
```

14.26 If-None-Match

If-None-Match request-header 필드는 method 와 함께 사용하여 method 를 조건적으로 만든다. 이전에 자원에서 획득한 하나 또는 그 이상의 엔터티를 가진 클라이언트는 연관된 엔터티 태그의 목록을 If-None-Match 헤더 필드에 포함하여 이러한 엔터티 중의 하나가 현재의 것임을 증명할 수 있다. 이 기능의 목적은 트랜잭션 오버헤드(transaction overhead)를 최소화하면서 캐시된 정보를 효과적으로 갱신할 수 있도록 하는 것이다. 또한 요구를 갱신할 때 자원의 잘못된 버전에 대한 부주의한 변경을 방지하는 데 사용할 수 있다. 특별한 경우로 "*" 값은 자원의 모든 현재 엔터티와 일치한다.

```
If-None-Match           = "If-None-Match" ":" ( "*" | 1#entity-tag )
```

해당 자원에 유사한 GET 요구(If-Match 헤더 없이)에 대한 응답으로 리턴되었을 수 있는 엔터티의 엔터티 태그와 일치하는 어떠한 엔터티 태그나 "*"이 주어지고 해당 자원에 대한 현재의 엔터티가 존재한다면 서버는 요구 받은 method 를 절대로 수행해서는 안 된다. 대신 요구 method 가 GET 또는 HEAD 이면 서버는 일치하는 엔터티 중 하나의 캐시와 관련된 entity-header 필드(특히 ETag)를 포함하여 304 (Not Modified) 응답으로 응해야 한다. 다른 모든 요구 method 에 대해서 서버는 상태 412(Precondition Failed)로 응답해야 한다.

두 엔터티 태그가 일치하는지 결정하는 원칙에 대하여는 13.3.3 절을 참조한다. GET 또는 HEAD 요구에는 약한 비교 기능(weak comparison function)만을 사용할 수 있다.

어떠한 엔터티 태그도 일치하지 않고 "*"이 주어지고 해당 자원에 대한 아무런 현재 엔터티도 존재하지 않는다면 서버는 If-None-Match 헤더 필드가 존재하지 않는 것처럼 요구 받은 method 를 수행할 것이다.

[Page 121]

If-None-Match 헤더 필드 없이 요구가 2xx 상태 이외의 어떤 것이라도 초래하게 되면 If-None-Match 를 무시해야 한다.

" If-None-Match: *" 의 의미는 원서버(또는 Vary 메커니즘을 이용한 캐시, 14.43 절 참조)가 선택한 표시 방법이 존재하면 method 를 반드시 수행해야 하고 그렇지 않다면 절대 수행해서는 안 된다는 것이다. 이 기능은 PUT 처리 시 경쟁(race)을 방지하는 데 유용하다.

예:

```
If-None-Match: "xyzzy“
If-None-Match: W/"xyzzy“
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz“
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz“
If-None-Match: *
```

14.27 If-Range

클라이언트가 자신의 캐시에 엔터티의 부분적 사본을 가지고 있고 전체 엔터티의 최신 갱신 사본을 가지고 싶다면 조건적인 GET(If-Unmodified-Since 및 If-Match 중 하나나 둘 모두를 이용하여)의 Range request-header 를 사용할 수 있다. 그러나 엔터티가 변경되어 조건이 실패한다면 클라이언트는 현재의 전체 entity-body 를 획득하기 위해 2 차 요구를 할 수 있다.

If-Range 헤더는 클라이언트가 2 차 요구를 "단축(short-circuit)"할 수 있도록 한다. 약식으로 말하면 이것의 의미는 `엔터티가 변경되지 않았다면 내가 빠트린 부분만을 발송하고 그렇지 않다면 새로운 전체 엔터티를 발송하시오. '이다.

```
If-Range        = "If-Range" ":" ( entity-tag | HTTP-date )
```

클라이언트가 엔터티의 엔터티 태그를 가지고 있지 않으나 Last-Modified 날짜를 가지고 있으면 클라이언트는 그 날짜를 If-Range 헤더에서 사용할 수 있다. (서버는 2 문자 이내를 검사하여 유효한 HTTP-date 와 entity-tage 의 어떠한 폼도 구별할 수 있다.) If-Range 헤더는 Range 헤더와 함께만 사용할 수 있으며, Range 헤더를 포함하고 있지 않거나 서버가 하부-영역 운영(sub-range operation)을 지원하지 않으면 요구를 반드시 무시해야 한다.

[Page 123]

If-Range 내의 엔터티 태그가 엔터티의 현재 엔터티 태그와 일치하면 서버는 206 (Partial content)응답을 이용하여 자세한 엔터티의 하부-영역을 제공해야만 한다. 엔터티 태그가 일치하지 않으면 서버는 200 (OK) 응답을 이용하여 전체 엔터티를 리턴해야 한다.

14.28 If-Unmodified-Since

If-Unmodified-Since request-header 필드는 method 와 함께 사용하여 method 를 조건적으로 만든다. 요구된 자원이 이 필드에 명시된 시간 이후 변경되지 않았으면 서버는 If-Unmodified-Since 헤더가 존재하지 않는 것처럼 요구 받은 작업을 수행해야 한다.

요구 받은 변형자가 지정된 시간 이후에 변경되었으면 서버는 요구 받은 작업을 절대로 수행해서는 안 되며 반드시 412 (Precondition Failed)를 리턴해야 한다.

```
If-Unmodified-Since      = "If-Unmodified-Since" ":" HTTP-date
```

이 필드의 사용 예는,

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

요구가 대개(예를 들어 If-Unmodified-Since 헤더 없이) 2xx 상태 이외의 어떤 것이라도 초래하게 되면 If-Unmodified-Since 를 무시해야 한다.

명시한 날짜가 유효하지 않으면 헤더를 무시해야 한다.

14.29 Last-Modified

Last-Modified entity-header 필드는 원서버가 변형자가 마지막으로 변경되었다고 믿는 날짜와 시간을 표시한다.

```
Last-Modified   = "Last-Modified" ":" HTTP-date
```

사용 예는,

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

이 헤더 필드의 정확한 의미는 원서버의 구현 방식 및 원래 자원의 성격에 따라 달라진다. 파일의 경우 의미는 파일 시스템에서 마지막으로 변경된 시간을 역동적으로 부분을 포함하는 엔터티의 경우 의미는 부분 요소가 마지막으로 변경된 시간 세트가 될 수 있다. 데이터 베이스 게이트웨이의 경우에는 레코드의 최근 갱신 시간 스탬프(stamp)를 , 가상 객체의 경우에는 내부 상태가 마지막으로 변경된 시간일 수 있다.

[Page 124]

원서버는 절대 서버의 메시지 발생 시간보다 늦은 Last-Modified 날자를 발송해서는 안 된다. 이처럼 자원의 최근 변경이 미래의 특정 시간을 표시하는 경우 서버는 그 날짜를 메시지 발생 날짜로 대체해야 한다.

기본 서버는 엔터티의 Last-Modified 값을 응답의 Date 값을 생성한 시간과 가능한 한 가까운 것을 얻어야 한다. 이것은 수신측이 특히 엔터티가 응답이 생성된 시간에 가깝게 변경되었을 때 정확하게 엔터티의 변경 시간을 평가할 수 있도록 한다.

HTTP/1.1 서버는 가능할 때 마다 반드시 Last-Modified 를 발송해야 한다.

14.30 Location

Location response-header 필드는 요구의 완성 또는 새로운 자원의 식별을 위해 수신측의 방향을 Request-URI 가 아닌 다른 장소로 방향을 재설정하는 데 사용한다. 201 (Created) 응답의 경우

Location 은 요구에 의해 새롭게 생성된 자원의 위치이다. 3xx 응답의 경우 위치는 자원의 자동 방향 재설정을 위해 서버가 선호하는 URL 을 표시한다.

```
Location        = "Location" ":" absoluteURI
```

예는,

```
Location: http://www.w3.org/pub/WWW/People.html
```

주의 : Content-Location 헤더 필드(14.15절)는 Content-Location이 요구에 포함된 엔터티의 원래 위치를 식별한다는 점에서 Location과 다르다. 따라서 응답이 Location 및 Content-Location 헤더 필드를 모두 포함할 수 있다. 몇몇 methods의 캐시 필요 조건에 관해서는 13.10 절을 참조한다.

14.31 Max-Forwards

Max-Forwards request-header 필드는 TRACE method (14.31 절)와 함께 사용하여 다음의 들어오는 방향(inbound)의 서버에 요구를 전달할 수 있는 프락시나 게이트웨이의 숫자를 제한한다.

```
Max-Forwards            = "Max-Forwards" ":" 1*DIGIT
```


Max-Forwards 값은 이 요구 메시지가 전달될 수 있는 남은 횟수를 표시하는 십진수 정수이다.

Max-Forwards 헤더 필드를 포함하고 있는 TRACE 요구를 수신하는 각각의 프락시나 게이트웨이는 요구를 전달하지 이전에 그 값을 점검하고 갱신해야만 한다. 수신된 값이 제로(0)이면 수신측은 요구를 전달해서는 안 되며 대신 수신한 요구 메시지를 응답 entity-body(9.8 절에 기술한 바와 같이)로서 포함하는 200 (OK) 응답으로 마지막 수신측의 입장에서 응답해야 한다. 수신한 Max-Forwards 값이 제로보다 크면 전달된 메시지는 값이 1 만큼 감소된 갱신된 Max-Forwards 필드를 포함해야 한다.

Max-Forwards 헤더 필드는 이 규격에서 정의한 다른 모든 method 및 해당 method 정의의 일부분으로 명백하게 참조되지 않는 모든 확장 method 에서는 무시되어야 한다.

14.32 Pragma

Pragma general-header 필드는 request/response chain 을 따라 어떤 수신측에도 적용할 수 있는 구현 방식에 한정된 지시자(implementation-specific)를 포함하는 데 사용한다. 모든 pragma 지시자는 규약의 관점에서 선택 사항적 행태를 명시한다. 그러나 몇몇 시스템은 그 행태가 지시자와 일치할 것을 요구한다.

```
Pragma                  = "Pragma" ":" 1#pragma-directive
pragma-directive        = "no-cache" | extension-pragma
extension-pragma        = token [ "=" ( token | quoted-string ) ]
```

No-cache 지시자가 요구 메시지에 존재하면 애플리케이션은 요구되고 있는 것의 캐시 사본을 가지고 있다 하더라도 요구를 원서버에 전달해야 한다. 이 pragma 지시자는 no-cache cache-directive (14.9 절 참조)와 동일한 의미를 가지며 여기서는 HTTP/1.0 과의 호환성 유지를 위해 규정하였다. 클라이언트는 No-cache 요구가 HTTP/1.1 을 따르지 않는 것으로 알려진 서버로 전달되었을 때 두 헤더 필드를 모두 포함해야 한다.

Pragma 지시자는 애플리케이션에서 가지는 중요도에 관계없이 지시자는 request/response chain 을 따라 모든 수신측에 적용할 수 있기 때문에 반드시 프락시나 게이트웨이 애플리케이션을 통과해야 한다.

[Page 126]

HTTP/1.1 클라이언트는 Pragma request-header 를 발송해서는 안 된다. HTTP/1.1 캐시는 "Pragma: no-cache"를 클라이언트가 "Cache-Control: no-cache"를 발송한 것처럼 취급해야 한다. 더 이상의 새로운 Pragma 지시자는 HTTP 에 규정되지 않을 것이다.

14.33 Proxy-Authenticate

Proxy-Authenticate response-header 필드는 407 (Proxy Authentication Required) 응답의 일부분으로 반드시 포함해야 한다. 필드 값은 Request-URI 의 프락시에 적용할 수 있는 인증 scheme 이나 파라미터를 표시하는 인증 획득 시도로 구성되어 있다.

```
Proxy-Authenticate      = "Proxy-Authenticate" ":" challenge
```

HTTP 접근 인증 처리는 11 장에 기술되어 있다. WWW-Authenticate 와는 달리 Proxy-Authenticate 헤더 필드는 현재의 접속에만 적용되며 다운스트림(downstream) 클라이언트로 전달해서는 안 된다. 그러나 가장 가까운 프락시는 다운스트림(downstream) 클라이언트에 요청하여 자신의 증명서를 획득할 필요가 있을 수 있다. 어떤 상황에서는 프락시가 헤더 필드를 전송하는 것처럼 보일 것이다.

14.34 Proxy-Authorization

Proxy-Authorization request-header 필드는 클라이언트가 인증을 요구하는 프락시 자신(또는 자신의 사용자)을 식별시킨다. Proxy-Authorization 필드 값은 프락시 및/또는 요청되고 있는 자원의 영역에 관한 사용 에이전트의 인증 획득 정보를 포함하고 있는 증명서로 구성되어 있다.

```
Proxy-Authorization     = "Proxy-Authorization" ":" credentials
```

HTTP 접근 인증 획득 절차는 11 장에 기술되어 있다. Authorization 와는 달리 Proxy-Authorization 헤더 필드는 Proxy-Authenticate 필드를 이용한 인증 획득을 요구하는 다음의 외부로 향한(outbound) 프락시에만 적용된다. Chain 에서 복수의 프락시가 사용되었을 때 Proxy-Authorization 헤더 필드는 보증서를 수신할 예정인 외부로 향한 첫 프락시에 의해 사용된다. 프락시는 그것이 프락시가 주어진 요구를 상호 협조적으로 인증하는 메커니즘이라면 보증서를 클라이언트 요구에서 다음 프락시로 중계할 수 있다.

14.35 Public

Public response-header 필드는 서버가 지원하는 methods 세트를 열거한다. 이 필드의 목적은 엄격하게 수신측에 이례적인 methods 에 관한 서버의 능력을 알리는 데 있다.

[Page 127]

열거된 method 는 Request-URI 에 적용할 수 도 할 수 없을 수도 있다. Allow 헤더 필드(14.7 절)는 특정 URI 에 사용할 수 있는 method 를 표시하는 데 사용한다.

```
Public          = "Public" ":" 1#method
```

사용 예는,

Public: OPTIONS, MGET, MHEAD, GET, HEAD

이 헤더 필드는 클라이언트(예를 들어 연결 고리의 가장 가까운 이웃)에 직접적으로 접속된 서버에만 적용된다. 응답이 프락시를 통과한다면 프락시는 Public 헤더 필드를 삭제하든지 자신의 능력에 적용할 수 있는 것으로 대체해야 한다.

14.36 Range

14.36.1 Byte Ranges

모든 HTTP 엔터티는 연속적인 바이트로서 HTTP 메시지 내에 표현되기 때문에 바이트 범위의 개념은 모든 HTTP 엔터티에 의미가 있다.(그러나 모든 클라이언트나 서버가 byte-range 작업을 지원할 필요는 없다.)

HTTP 내의 바이트 영역 규격은 entity-body(반드시 message-body 와 동일할 필요는 없다.) 내의 바이트 연속에 적용된다.

바이트 영역 작업은 단일 바이트 영역이나 단일 엔터티 내의 영역 세트를 명시할 수 있다.

```
ranges-specifier          = byte-ranges-specifier
byte-ranges-specifier     = bytes-unit "=" byte-range-set
byte-range-set            = 1#( byte-range-spec | suffix-byte-range-spec )
byte-range-spec           = first-byte-pos "-" [last-byte-pos]
first-byte-pos            = 1*DIGIT
last-byte-pos             = 1*DIGIT
```

Byte-range-spec 의 first-byte-pos 값은 영역에서 첫 바이트의 byte-offset 을 제공한다. Last-byte-pos 값은 영역에서 마지막 바이트의 byte-offset 을 제공한다. 말하자면 명시된 바이트 위치는 포괄적인 것이다. 바이트 오프셋(byte offsets)은 0 부터 시작한다.

[Page 128]

Last-byte-pos 값이 존재하면 해당 byte-range-spec 의 first-byte-pos 보다 크거나 같아야 한다. 그렇지 않으면 byte-range-spec 은 유효하지 않다. 유효하지 않는 byte-range-spec 의 수신측은 이를 무시해야 한다.

Last-byte-pos 값이 없거나 entity-body 의 현재 길이보다 크거나 같으면 last-byte-pos 은 바이트 단위로 entity-body 의 현재 길이보다 작은 것과 동일한 것을 사용해야 한다.

클라이언트는 last-byte-pos 선택을 통하여 엔터티의 크기를 모른 채 수신한 바이트의 숫자를 제한할 수 있다.

```
suffix-byte-range-spec  = "-" suffix-length
suffix-length           = 1*DIGIT
```

Suffix-byte-range-spec 은 suffix-length 값이 부여한 길이의 entity-body 접미사를 명시하는 데 사용한다.(말하자면 이 폼이 entity-body 의 마지막 N 바이트를 명시한다.) 엔터티가 명시된 suffix-length 보다 짧으면 전체 entity-body 가 사용된다.

Byte-ranges-specifier 값의 사용 예이다.(entity-body 의 길이가 10000 이라고 가정했을 때)

- 첫 500 바이트 (byte offsets 0-499, inclusive):
  bytes=0-499
- 두 번째 500 바이트 (byte offsets 500-999, inclusive):
  bytes=500-999
- 마지막 500 바이트(byte offsets 9500-9999, inclusive):
  bytes=-500
- 또는
  bytes=9500-
- 처음과 마지막 바이트만(bytes 0 and 9999):
  bytes=0-0,-1

[Page 129]

- 규범적이지는 않지만 유효한 두 번째 500 바이트 명시 (byte offsets 500-999, inclusive):
  bytes=500-600,601-999
  bytes=500-700,601-999

14.36.2 Range Retrieval Requests

조건적 또는 무조건적 GET method 를 이용하는 HTTP 조회 요구는 Range 요구 헤더를 이용하여 요구의 결과로 리턴되는 엔터티에 적용할 수 있는 전체 엔터티 대신 하나 또는 그 이상의 엔터티의 하부 영역을 요청할 수 있다.

```
Range          = "Range" ":" ranges-specifier
```

서버는 Range 헤더를 무시할 수 있다. 그러나 HTTP/1.1 원서버 및 가장 가까운 캐시는 Range 가 부분적으로 실패한 전송을 효과적으로 복구하고 큰 엔터티의 효과적인 부분적 조회를 지원하기 때문에 가능하면 바이트 영역을 지원해야 한다.

서버가 Range 헤더를 지원하고 명시된 영역 이나 영역들이 엔터티에 적합하다면:

- 무조건적인 GET에 Range 헤더가 있으면 GET이 성공했을 때 리턴되는 것을 변경한다. 달리 표현하면 응답은 상태 코드 200 (OK) 대신에 206 (Partial Content)을 가지고 온다.

- 조건적인 GET(If-Modified-Since 및 If-None-Match 중 하나나 둘 모두, 또는If-Unmodified-Since 및 If-Match 중 하나나 둘 모두를 이용하는 요구)에 Range 헤더가 있으면 GET이 성공하고 조건이 참일 때 리턴되는 것을 변경한다. 이것은 조건이 거짓일 경우 리턴되는 304 (Not Modified) 응답에 영향을 미치지 않는다.

어떤 경우에는 Range 헤더에 첨가형 If-Range 헤더(14.27 절 참조)를 사용하는 것이 더 적절할 수도 있다.

Range를 지원하는 프락시가 Range 요구를 수신하고 안으로 향하는(inbound) 서버에 요구를 전달하고 이의 응답으로 전체 엔터티를 수신하면 프락시는 클라이언트에게 요구 받은 영역만을 리턴해야 한다. 프락시는 그것이 캐시 할당 정책과 부합된다면 전체 수신 응답을 자신의 캐시에 저장해야 한다.

[Page 130]

14.37 Referer

Referer[sic] request-header 필드는 클라이언트가 서버를 위해 Request-URI를 얻은("referrer", 헤더 필드의 스펠링이 틀렸다.) 자원의 주소(URI)를 명시하는 데 사용한다. Referer request-header는 서버가 취미, 로깅 또는 최적화된 캐시 등의 목적으로 자원에 대한 back-links 목록을 생성할 수 있도록 한다. 또는 폐기되었거나 타이핑을 잘못한 링크를 유지관리하기 위해 추적할 수 있도록 한다. Referer 필드는 사용자 키보드에서의 입력 등 자신의 URI를 가지고 있지 않는 출처에서 얻은 Request-URI를 절대로 발송해서는 안 된다.

```
Referer         = "Referer" ":" ( absoluteURI | relativeURI )
```

예:

```
Referer: http://www.w3.org/hypertext/DataSources/Overview.html
```

필드 값이 부분적 URI이면 값을 Request-URI에 상대적으로 해석해야 한다. URI는 절대로 파편을 포함해서는 안 된다.

주의 : 링크의 출처가 개인적인 정보이거나 개인적인 정보 출처를 누설할 수 있기 때문에 사용자는 Referer 필드를 발송할 것인지 여부를 선택할 수 있도록 할 것을 강력하게 추천한다. 예를 들어 브라우저 클라이언트는 공개/무명 브라우징(browsing)의 토글 스위치(toggle switch)를 가질 수 있으며 이는 각각 Referer 및 From 정보의 발송을 확성화/무력화한다.

14.38 Retry-After

Retry-After response-header 필드는 503 (Service Unavailable) 응답과 함께 사용하여 요청하는 클라이언트에게 얼마나 오랫동안 서비스를 사용할 수 있는지 표시할 수 있다. 이 필드의 값은 응답 시간 이후의 HTTP-date 또는 초의 정수(십진수)가 될 수 있다.

```
Retry-After     = "Retry-After" ":" ( HTTP-date | delta-seconds )
```

사용의 두 예는,

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

후자의 예에서 지연시간은 2 분이다.

[Page 131]

14.39 Server

Server response-header 필드는 요구를 처리하는 원서버가 사용하는 소프트웨어에 관한 정보를 포함하고 있다. 이 필드는 서버와 중요한 하부 제품을 식별하는 복수의 제품 토큰(3.8 절) 및 주석을 포함하고 있다. 제품 토큰은 애플리케이션을 식별하는 중요도의 순서에 따라 열거되어 있다.

```
Server          = "Server" ":" 1*( product | comment )
```

예:

```
Server: CERN/3.0 libwww/2.17
```

응답이 프락시를 통하여 전달된다면 프락시 애플리케이션은 절대 Server response-header 를 변경해서는 안 된다. 대신 프락시는 Via 필드(14.44 절에 기술된 대로)를 포함하여야 한다.

주의 : 서버의 특정 소프트웨어 버전을 누설하는 것은 서버가 보안 허점(security holes)을 가진 것으로 알려진 소프트웨어에 대한 공격에 더욱 취약하도록 만들 수 있다. 서버 구현자는 이 필드를 설정할 수 있는 선택 사항으로 만들 것을 권고한다.

14.40 Transfer-Encoding

Transfer-Encoding general-header 필드는 송신측과 수신측 사이에 메시지를 안전하게 전송하기 위해 어떤(적용되었다면) 유형의 변형이 메시지에 적용되었는지 표시한다. 이것은 전송 코딩(transfer coding)이 메시지의 특성이지 엔터티의 특성이 아니라는 점에서 Content-Encoding 과 다르다.

```
Transfer-Encoding       = "Transfer-Encoding" ":" 1#transfer-coding
```

Transfer codings 은 3.6 절에 규정되어 있으면 그 예는,

```
Transfer-Encoding: chunked
```

많은 이전 HTTP/1.0 애플리케이션은 Transfer- Encoding 헤더를 이해하지 못한다.

14.41 Upgrade

Upgrade general-header 는 클라이언트가 추가적으로 어떤 통신 규약을 지원하며 규약을 전환하는 것이 적절할 때 어떤 통신 규약을 사용하고자 하는지 명시할 수 있도록 한다.

[Page 132]

서버는 반드시 Upgrade 헤더 필드를 101 (Switching Protocols) 응답에 사용하여 어떤 규약이 전환되고 있는지 표시해야 한다.

```
Upgrade         = "Upgrade" ":" 1#product
```

예,

```
Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

Upgrade 헤더 필드는 HTTP/1.1 에서 다른 호환되지 않는 규약으로 이전하는 간단한 메커니즘을 제공할 목적으로 사용된다. 클라이언트가 비록 현재 의 요구를 HTTP/1.1 을 이용하여 작성하였다 하더라도 높은 주요 변경 버전 번호를 가진 HTTP 의 이후 버전과 같은 다른 규약을 사용하고 싶다는 것을 광고할 수 있도록 하여 이를 성취한다. 이것은 클라이언트가 사용할 수 있다면 "좀더 낳은" 규약을 사용하고 싶어한다는 것을 표시하면서도 클라이언트가 좀더 보편적으로 지원되는 규약에서 요구를 시작할 수 있도록 하여 호환되지 않는 규약간의 어려운 이전을 용이하게 한다. ("좀더 낳은" 규약은 가능하면 요구하고 있는 method 및/또는 자원의 기본적인 성격에 따라 서버가 결정한다.)

Upgrade 헤더 필드는 기존의 전송-계층 연결 위에서 애플리케이션-계층 규약을 전환하는 것에만 적용된다. Upgrade는 규약의 전환을 고집하는 데 사용할 수 없다. 전환의 수용 및 사용은 서버의 선택 사항이다. 규약 변경 후의 첫 작업은 Upgrade 헤더 필드를 포함하고 있는 첫 HTTP 요구에 대한 응답이어야 하지만 규약 변경 이후의 애플리케이션-계층 통신의 능력 및 기본적인 성격은 새롭게 선택된 규약에 전적으로 달려 있다.

Upgrade 헤더 필드는 오직 가장 가까운 연결에만 적용된다. 따라서 Upgrade 핵심어는 Upgrade 헤더 필드가 HTTP/1.1 메시지에 존재할 때 Connection 헤더 필드(14.10 절) 내에서만 제공하여야 한다.
Upgrade 헤더 필드는 다른 연결로의 규약 전환을 표시하는 데 사용할 수 없다. 이러한 목적에는 301, 302, 303 또는 305 방향 재설정 응답이 더 적절하다. 이 규격은 3.1 절의 HTTP 버전 규칙 및 이 규격의 향후 개정에서 규정한 것과 같이 하이퍼텍스트 전송 규약 집단에서 사용할 목적으로 오직 "HTTP"라는 이름의 규약만을 규정한다. 규약 이름을 위해 어떠한 토큰을 사용해도 되지만 클라이언트와 서버 모두가 해당 이름을 동일한 규약으로 연관시킬 때문에 유용할 것이다.

[Page 133]

14.42 User-Agent

User-Agent request-header 필드는 요구를 만들어 낸 사용자 에이전트에 관한 정보를 포함하고 있다. 이것은 통계 목적, 규약 위반의 추적, 특정 사용자 에이전트 한계를 피하기 위해 응답을 고칠 목적으로 사용자 에이전트를 자동 인지하기 위함이다. 사용자 에이전트는 요구에 이 필드를 포함해야 한다. 이 필드는 사용자 에이전트의 중대한 부분을 형성하는 에이전트, 모든 하부 제품을 식별할 수 있는 복수의 제품 토큰(3.8 절) 및 주석을 포함할 수 있다. 관례상 제품 토큰은 애플리케이션을 식별하는 중요도의 순서에 따라 열거되어 있다.

```
User-Agent      = "User-Agent" ":" 1*( product | comment )
```

예:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

14.43 Vary

Vary response-header 필드는 서버가 응답 엔터티를 서버가 주도하는 협상(12 장)을 이용한 이용 가능한 응답의 표시 방법에서 선택하였음을 표시하는 데 사용한다. Vary 헤더에 열거된 field-names은 request-headers의 field-names이다. Vary 필드 값은 주어진 헤더 필드 세트가 표시 방식이 변화할 수 있는 차원을 넘어선다는 것을 나타내거나, 변형의 차원이 명시되지 않아("*") 향후 요구의 어떠한 측면에서도 변형될 수 있다는 것을 나타낸다.

```
Vary            = "Vary" ":" ( "*" | 1#field-name )
```

HTTP/1.1 서버는 반드시 서버가 주도하는 협상에 종속되는 모든 캐시할 수 있는 응답에 적절한 Vary 헤더 필드를 포함해야 한다. 이렇게 하면 캐시가 해당 자원에 대한 향후 요구를 적절하게 해석할 수 있도록 하며 사용자 에이전트에게 해당 자원에 대한 협상이 존재함을 알릴 수 있다. 서버는 서버가 주도하는 협상에 종속되는 캐시할 수 없는 응답에 적절한 Vary 헤더 필드를 포함해야 한다. 이것이 사용자 에이전트에게 응답이 변형될 수 있는 차원에 관한 유용한 정보를 제공할 수 있기 때문이다.

Vary 헤더 필드 값에 의하여 명명되는 헤더 필드의 세트는 "selecting" request-headers로 알려져 있다.

캐시가 Request-URI 가 Vary 헤더를 포함한 하나 또는 그 이상의 캐시 엔트리를 명시하는 계속적인 요구를 수신했을 때 캐시된 Vary 헤더에 명명된 모든 헤더가 새로운 요구에 있거나 이전 요구의 모든 저장된 selecting request-headers 가 새로운 요구의 해당 헤더와 일치하지 않는 한 캐시는 절대 이러한 캐시 엔트리를 이용하여 새로운 요구에 대한 응답을 구성해서는 안 된다.

[Page 134]

두 요구는 메시지 헤더에 관한 4.2 절의 규칙에 따라 동일한 필드 이름의 복수 message-header 필드를 결합하거나 또한/또는 선형 공백문자(LWS)를 상응하는 BNF 가 허용하는 지점에 추가하거나 삭제하여 첫 요구의 selecting request-headers 가 두 번째 요구의 selecting request-headers 로 변환될 수 있을 때만 일치하는 것으로 규정된다.

"*" 의 Vary 필드 값은 아마도 request-header 필드 내용이 아닌(예를 들어 클라이언트의 네트워크 주소) 명시되지 않은 파라미터가 응답 표시 방법의 선택에 어떤 역할을 수행하고 있음을 표시한다. 해당 자원에 대해 계속되는 요구는 원서버에 의하여 적절히 해석될 수 있기 때문에 자원의 캐시된 신선한 응답을 가지고 있을 때도 캐시는 반드시 요구를(조건적일 수 있다.) 전송해야 한다. 캐시가 사용하는 Vary 헤더에 대해서는 13.6 절을 참조한다. Field-names 목록으로 구성된 Vary 필드 값은 응답을 위해 선택된 표시 방식이 최적의 표시 방법을 선택할 때 열거된 request-header 필드 값만을 고려하는 선택 알고리즘에 기초하고 있다는 것을 표시한다. 캐시는 열거된 필드 이름을 위해 동일한 값으로 응답이 신선한 시간 동안만 향후의 요구에서 동일한 선택을 하게 되리라 가정해도 된다.

주어진 field-names 은 이 규격에서 규정한 표준 request-header 세트에 한정된 것은 아니다. 필드 이름은 대소문자를 구별한다.

14.44 Via

게이트웨이나 프락시는 반드시 Via general-header 필드를 사용하여 요구를 만들었을 때는 사용자 에이전트와 서버 사이의, 응답을 수신했을 때는 원서버와 클라이언트 사이의 가장 가까운 규약 및 수신측을 표시해야 한다. 이것은 RFC 822 의 "Received" 필드와 유사하다. 또한 이것을 메시지 전달(message forwards)을 추적하고, 요구 루프(request loops)를 피하며 request/response chain 을 따라 모든 송신측의 규약 능력을 식별하는 데 사용하도록 계획되었다.

[Page 135]

```
Via                =  "Via" ":" 1#( received-protocol received-by [ comment ] )
received-protocol  = [ protocol-name "/" ] protocol-version
protocol-name      = token
protocol-version   = token
received-by        = ( host [ ":" port ] ) | pseudonym
pseudonym          = token
```

Received-protocol 은 request/response chain 의 각 부분(segment)을 따라 서버가 클라이언트가 수신하는 메시지의 규약 버전을 표시한다. Received-protocol 은 업스트림(upstream) 애플리케이션에 관한 정보를 모든 수신측이 볼 수 있도록 하기 위해 메시지가 전달되었을 때 Via 필드 값에 추가된다.

Protocol-name 은 그것이 "HTTP"일 때만 선택 사항이다. Received-by 필드는 보통 계속적으로 메시지를 전달하는 수신측 서버나 클라이언트의 호스트나 선택적인 포트 번호이다. 그러나 진짜 호스트가

민감한 정보를 가진 것으로 간주된다면 가명(pseudonym)으로 대체할 수 있다. 포트가 주어지지 않았으면 received-protocol 의 기본 포트인 것으로 가정할 수 있다.

복수의 Via 필드 값은 메시지를 전달한 각각의 프락시나 게이트웨이를 표시한다. 각각의 수신측은 마지막 결과가 전송한 애플리케이션의 순서에 따라 순서를 정할 수 있도록 자신의 정보를 반드시 추가해야 한다.

Via 헤더 필드에 주석을 사용하여 User-Agent 나 Server 헤더 필드와 유사하게 수신측 프락시나 게이트웨이의 소프트웨어를 식별할 수 있다. 그러나 Via 필드의 모든 주석은 선택적이며 메시지를 전달하기 이전에 수신측이 삭제할 수 있다.

예를 들어 HTTP/1.0 사용자 에이전트에서 요구 메시지를 HTTP/1.1 을 이용하여 "fred"라는 코드 이름이 붙은 내부 프락시로 전달할 수 있다. 내부 프락시는 요구를 nowhere.com 에 있는 공공 프락시로 전달하며 nowhere.com 은 요구를 에 있는 원서버에 전달하여 요구 처리를 완료한다면 가 수신한 요구는 다음의 Via 헤더 필드를 가지게 될 것이다.

```
Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
```

네트워크 방화벽(firewall)의 입구로 사용되는 프락시나 게이트웨이는 기본적으로 방화벽 영역 내의 호스트의 이름이나 포트를 전달해서는 안 된다. 이 정보는 명백하게 활성화되었을 때만 전파할 수 있다. 활성화되지 않았으면 방화벽 뒤의 모든 호스트의 received-by host 는 해당 호스트의 적절한 가명(pseudonym)으로 대체되어야 한다.

[Page 136]

내부 조직을 숨겨야 한다는 강한 사생활 보호 필요 조건을 가진 조직을 위해 프락시는 동일한 received-protocol 값을 가진 Via 헤더 필드 엔트리의 순서가 정해진 순차를 단일 엔트리로 결합
할 수도 있다.

예를 들면,

```
Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy
```

을 다음과 같이 축소할 수 있다.

```
Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

애플리케이션은 복수의 엔트리를 그것들이 동일한 조직 통제 밑에 있거나 호스트가 가명으로 대체되지 않는 한 결합해서는 안 된다. 애플리케이션은 상이한 received-protocol 값을 가지고 있는 엔트리를 절대로 결합해서는 안 된다.

14.45 Warning

Warning response-header 필드는 응답 상태 코드가 반영하지 않은 응답 상태에 관한 정보를 전송하는 데 사용한다. 이 정보는 비록 배타적이진 않지만 대개 캐시 작업에 발생할 수 있는 의미 투명성(semantic transparency)의 결여에 대한 경고를 하는 데 사용한다.

Warning 헤더는 다음을 이용하여 응답과 함께 발송된다.

```
Warning          = "Warning" ":" 1#warning-value
warning-value    = warn-code SP warn-agent SP warn-text
warn-code        = 2DIGIT
```

```
warn-agent          = ( host [ ":" port ] ) | pseudonym
                      ; Warning 헤더를 추가하는 서버의 이름 또는 별명
                      ; 디버깅에 사용한다.
warn-text           = quoted-string
```

하나의 응답은 하나 이상의 Warning 헤더를 포함할 수 있다.

Warn-text 는 응답을 수신하는 인간 사용자가 가장 잘 이해할 수 있는 자연적인 언어 및 문자 집합으로 표시해야 한다. 이에 대한 결정은 캐시나 사용자의 위치, 요구의 Accept-Language 필드, 응답의 Content-Language 필드 등 사용 가능한 어떤 정보에 기초해도 된다. 기본적인 언어는 영어이며 기본 문자 집합은 ISO-8859-1 이다.

ISO-8859-1 이외의 문자 집합이 사용되었으면 RFC 1522 [14]에 기술한 method 를 사용하여 warn-text 내에 인코딩해야 한다.

[Page 137]

어떠한 서버나 캐시도 응답에 Warning 헤더를 추가할 수 있다. 새로운 Warning 헤더는 모든 기존 Warning 헤더 다음에 추가해야 한다. 캐시는 응답과 함께 수신한 어떠한 Warning 헤더도 삭제해서는 안 된다. 그러나 캐시가 성공적으로 캐시 엔트리를 검증했으면 특정 Warning 코드가 명시한 경우를 제외하고는 해당 엔트리에 이전에 첨가된 모든 Warning 헤더는 삭제해야 한다. 그런 다음 캐시는 응답을 검증하는 동안 수신한 어떠한 Warning 헤더라도 추가할 수 있다. 달리 표현하면 Warning 헤더는 가장 최근에 관련된 응답에 추가된 헤더이다.

응답에 복수의 Warning 헤더가 첨부되었으며 사용자 에이전트는 응답에서 나타난 순서 대로 가능한 한 많은 Warning 헤더를 표시해야 한다. 모든 경고문을 표시할 수 없을 때 사용자 에이전트는 다음의 발견법(heuristics)에 따라야 한다.

- 답의 초기에 나타난 warnings이 나중에 나타난 것보다 우선권을 갖는다.
- 사용자가 선호하는 문자 집합에서 발생한 warning이 다른 문자 집합의 warnings보다 우선권을 갖는다. 그러나 warn-codes 및 warn-agents는 동일하다.

복수의 Warning 헤더를 생성하는 시스템은 이러한 사용자 에이전트의 행태를 염두에 두고 순서를 정해야 한다.

다음은 현재 정의된 warn-codes 이며 영어로 추천 warn-text 및 의미를 기술하고 있다.

110 Response is stale
리턴 된 응답이 낡을 때는 언제나 반드시 포함해야 한다. 캐시는 이 경고를 어떠한 응답에도 추가할 수 있지만 응답이 새로운 것으로 알려지기 전에는 절대 삭제해서는 안 된다.

111 Revalidation failed
서버에 도달하지 못하기 때문에 응답을 재검증하려는 시도가 실패하여 캐시가 낡은 응답을 리턴하게 되면 반드시 포함해야 한다. 캐시는 이 경고를 어떠한 응답에도 추가할 수 있지만 응답을 성공적으로 재검증하기 전에는 절대 삭제해서는 안 된다.

112 Disconnected operation
캐시가 의도적으로 일정기간 동안 나머지 네트워크로부터 단절되었을 때는 포함해야 한다.

113 Heuristic expiration

캐시가 발견법 상 신선한 기간을 24 시간 이상으로 선택하거나 응답의 경과 시간이 24시간 이상일 때는 반드시 포함해야 한다.

199 Miscellaneous warning

경고 텍스트는 인간 사용자에게 제공하기 위해 또는 로깅하기 위해 자의적인 정보를 포함할 수 있다. 이 경고를 수신한 시스템은 절대로 어떠한 자동 작업도 수행해서는 안 된다.

[Page 138]

214 Transformation applied

가장 가까운 캐시나 프락시가 이 Warning 코드가 응답에 포함되어 있지 않은 한 content-coding(Content-Encoding 헤더에 명시된 것처럼) 또는 응답의 media-type (Content -Type 헤더에 명시된 것처럼)에 변형을 가하는 변경 사항을 적용했을 때 반드시 추가해야 한다. 재검증 후에도 응답에서 삭제해서는 안 된다.

14.46 WWW-Authenticate

WWW-Authenticate response-header 필드를 반드시 401(Unauthorized) 응답 메시지에 포함해야 한다. 필드 값은 Request-URI 에 적용할 수 있는 인증 획득 scheme 및 파라미터를 나타내는 최소한 하나의 인증 시도(challenge)로 구성된다.

```
WWW-Authenticate      = "WWW-Authenticate" ":" 1#challenge
```

11 장에 HTTP 접근 인증 획득 절차가 기술되어 있다. 사용자 에이전트는 WWW-Authenticate 필드 값을 분석할 때 하나 이상의 WWW-Authenticate 헤더 필드가 있으면 인증 시도의 내용이 인증 파라미터의 콤마로 분리된 목록을 포함하고 있기 때문에 각별한 주의를 해야 한다.

15 보안에 대한 고려 사항

이 절에서는 이 문서가 기술한 바와 같이 애플리케이션 개발자, 정보 제공자 및 사용자에게 HTTP/1.1 의 보안 제한 사항을 알리고자 한다. 토론이 규정적인 해결 방안을 포함하지는 않지만 보안의 위험성을 감소시킬 수 있는 몇몇 안을 제시한다.

15.1 클라이언트의 인증

Basic authentication scheme 은 운반용기(carrier)로 사용되어 명백한 텍스트로 물리적인 네트워크 위에서 전송되는 안정된 사용자 인증 method 도 아니며 엔터티를 어떤 식이든 보호하지도 않는다. HTTP 는 보안을 증가 시키기 위하여 추가적인 인증 schemes 이나 암호화 메커니즘을 사용하는 것을 금지하는 않는다.(일회 암호 사용 scheme 처럼)

[Page 139]

Basic authentication 의 가장 심각한 단점은 근본적으로 물리적인 네트워크 위로 사용자의 암호가 명백한 텍스트로 전달된다는 것이다. Digest Authentication 이 언급하고자 하는 것이 바로 이 문제이다.

Basic authentication 이 암호의 명백한 텍스트 전송을 수반하기 때문에 민감하고 소중한 정보를 보호하기 위해서(기능 향상 없이는) 결코 사용해서는 안 된다.

Basic authentication 의 일반적인 용도는 식별(identification) 목적이다. 예를 들어 정확한 서버의 사용 빈도에 관한 통계를 수집하기 위해서 식별의 수단으로서 사용자의 이름과 암호를 사용자가 제공하도록 요구한다. 이런 식으로 사용했을 때 보호된 문서에 불법으로 접근하는 것이 주요 관심사가 아닌 경우 위험성이 없는 것으로 생각할 수 있다. 이것은 서버가 사용자 이름 및 암호를 사용자에게 발행할 때, 특히 사용자가 자신의 암호를 선택할 수 없도록 했을 때 올바른 생각이다. 순진한 사용자는 종종 복수의 암호를 유지하는 일을 피하기 위해 단일 암호를 재사용하기 때문에 위험이 발생한다.

서버가 사용자로 하여금 자신의 암호를 선택하게 한다면 위험성은 서버의 문서에 불법으로 접근하는 것뿐만 아니라 자신의 계정 암호를 사용하기로 결정한 모든 사용자의 계정에 불법으로 접근하는 것이다. 사용자가 자신의 암호를 선택하도록 허용한다는 것은 서버가 암호(아마도 암호화된)를 포함하고 있는 파일을 유지해야만 한다는 것을 의미한다. 많은 것들이 원격지 사이트의 계정 암호일 것이다. 이러한 시스템의 소유주 또는 관리인은 이러한 정보가 안전한 방법으로 유지되지 않으면 책임 발생을 초래할 수도 있다. Basic Authentication 는 또한 모조 서버에 의한 속임수에도 취약하다. 사실상 사용자가 적대적인 서버나 게이트웨이에 접속하고 있는데도 기본 인증에 의하여 보호 받는 정보를 포함하고 있는 호스트에 연결되어 있다고 사용자가 믿도록 하였으면 공격자는 암호를 요청하여 이것을 나중 용도를 위해 저장하고 에러인 것처럼 위장할 수 있다.

이런 유형의 공격은 Digest Authentication[32]에서는 가능하지 않다. 서버 구현자는 이와 같은 종류의 게이트웨이나 CGI 스크립트 위조 가능성에 대비하여야 한다. 특히 서버가 연결을 게이트웨이로 전달하는 것은 상당히 위험하다.

그런 다음 게이트웨이는 지속적인 연결 메커니즘을 이용하여 클라이언트가 탐지할 수 없는 방식으로 원서버인 것처럼 작동하면서 클라이언트와의 복수 트랜잭션에 참여할 수 있기 때문이다.

15.2 인증 scheme 을 선택할 수 있도록 함

HTTP/1.1 서버는 복수의 인증 시도(challenge)를 401(Authenticate) 응답으로 리턴하며 각각의 인증 시도는 별도의 scheme 을 사용할 수 있다.

[Page 140]

사용자 에이전트에게 리턴되는 인증 시도 순서는 서버가 선호하는 순서이다. 서버는 "가장 안전한" 인증 획득을 우선으로 하여 자신의 인증 시도 순서를 정해야 한다. 사용자 에이전트는 자신이 이해하는 것을 사용자가 처음 인증을 시도하는 것으로 선택해야 한다.

서버가 WWW-Authenticate 헤더를 이용하여 선택한 인증 획득 scheme 을 제공할 때 악의의 사용자가 일련의 인증 시도를 약탈하여 인증 획득 scheme 의 가장 약한 부분을 이용하여 자신을 인증하려 하기 때문에 인증 획득의 "안전" 만을 제공하는 것이다. 따라서 순서는 서버의 정보보다는 사용자의 증명서를 보호하는 데 사용된다.

사람이 중간이 낀 공격(man-in-the-middle (MITM) attack)은 약한 인증 획득 scheme 를 선택 사항 세트에 추가하여 클라이언트가 사용자의 증명서(예를 들면 암호)를 노출시키는 것을 하나 사용할 것이라는 기대하는 것이다. 이러한 이유 때문에 클라이언트는 항상 접수한 선택 사항 중 자신이 이해하는 가장 강한 scheme 을 사용해야 한다. 좀더 향상된 MITM 공격은 제공된 모든 선택 사항을 삭제하고 Basic authentication 을 요청하는 인증 시도를 삽입하는 것이다. 이러한 이유로 인하여 이러한 공격을 염려하는

사용자 에이전트는 서버가 요청했던 가장 강력한 인증 획득 scheme 을 기억하고 약한 것을 사용하기 전에 사용자의 확인을 요구하는 경고 메시지를 생성할 수 있다. 특히 이러한 공격을 가하려는 음흉한 방법은 속기 쉬운 사용자에게 "무료" 프락시 캐시를 제공하는 것이다.

15.3 서버 로그 정보의 남용

서버는 사용자의 읽는 유형 및 관심사를 알려 줄 수 있는 사용자의 요청에 관한 개인적인 데이터를 저장하는 입장에 있다. 이러한 정보는 기본 성격상 분명히 비밀이며 특정 국가에서는 이러한 정보의 처리가 법에 의하여 제한을 받고 있다. 데이터를 제공하기 위해 HTTP 규약을 사용하는 사람은 발행된 결과로 식별할 수 있는 이러한 자료가 개인의 허락 없이 배포되지 않도록 하는 책임을 갖고 있다.

15.4 민감한 정보의 전송

일반적인 자료 전송 규약처럼 HTTP 는 전송되는 데이터의 내용을 통제할 수 없으며 사전에 특정 요구의 문맥에서 정보의 특정 부분의 민감성을 결정할 방법이 없다. 따라서 애플리케이션은 이러한 정보에 대한 통제를 해당 정보의 제공자에게 가능한 한 많이 제공하여야 한다. 네 개의 필드가 이러한 의미에서 특별히 언급할 가치가 있다. - Server, Via, Referer 및 From.

[Page 141]

서버의 특정 소프트웨어 버전을 표시함으로 해서 서버가 보안 허점을 가진 것으로 알려진 소프트웨어에 대한 공격에 좀더 취약하도록 만들 수 있다. 구현자는 Server 헤더 필드를 설정할 수 있는 선택 사항으로 만들어야 한다.

네트워크 방화벽을 따라 입구의 역할을 하는 프락시는 방화벽 뒤의 호스트를 식별하는 헤더 정보의 전달에 관하여 특별한 주의를 기해야 한다. 특히 프락시는 방화벽 뒤에서 생성된 모든 Via 필드를 삭제하던지 청소된(sanitized) 버전으로 대체해야 한다.

Referer 필드는 읽기 유형을 연구하고 역 링크를 구성할 수 있도록 한다. 비록 매우 유용하기는 하지만 Referer 에 포함된 정보에서 사용자의 인적 사항을 분리하지 않으면 이러한 능력이 남용될 수 있다. 개인적 정보가 삭제된 이후에도 Referer 필드는 발행하는 것이 적절하지 않는 개인적 문서의 URI 를 표시할 수 있다. From 필드에서 전달하는 정보가 사용자의 사생활 보호나 사용자 사이트의 보안 정책과 충돌될 수 있다. 따라서 사용자가 필드의 내용을 무력화, 활성화 및 변경할 수 없을 때는 From 필드는 전달되어서는 안 된다. 사용자는 사용자 선택 사항이나 애플리케이션의 기본 환경 설정에서 이 필드의 내용을 설정할 수 있어야 한다.

우리는 비록 요구하지는 않지만 편리한 토글(toggle) 인터페이스를 사용자에게 제공하여 사용자가 From 및 Referer 정보를 활성화 또는 무력화 할 수 있도록 하는 것을 추천한다.

15.5 파일 및 경로 이름에 기초한 공격

HTTP 원서버의 구현 방식은 HTTP 응답이 리턴하는 문서를 서버 관리자가 의도한 것만으로 한정하도록 주의하여야 한다. 만약 HTTP 서버가 HTTP URI 를 파일 시스템 호출로 바로 해석하면 서버는 HTTP 클라이언트에게 제공하지 않으려는 파일이 제공되지 않도록 특별한 주의를 해야 한다. 예를 들어 UNIX, Microsoft Windows 및 다른 운영 체계는 현재 디렉토리 바로 위를 표시하기 위해 ".."를 경로 구성원으로 사용한다. 이러한 시스템에서 HTTP 서버는 이것이 HTTP 서버를 통하여 접근하도록 계획된 것 이외의 자원에 접근할 수 있도록 할 수 있기 때문에 Request-URI 에 이러한 구성 형식을 허용하지 말아야 한다.

마찬가지로 서버 내부 참조만을 위한 파일(접근 제어 파일, 환경 설정 파일 및 스크립트 코드 등)은 민감한 정보를 포함하고 있을 수도 있기 때문에 부적절한 조회에서 보호해야 한다. 경험에 따르면 이러한 HTTP 서버 구현 방식의 사소한 버그가 보안 허점으로 바뀔 수도 있음을 보여 주고 있다.

[Page 142]

15.6 개인적인 정보

HTTP 클라이언트는 종종 대량의 개인 정보(사용자의 이름, 위치, 우편 주소, 암호, 암호화 키 등)에 관계되어 있으며 이러한 정보가 HTTP 규약을 통하여 다른 출처로 원하지 않게 누출되는 것을 방지하도록 주의해야 한다. 우리는 이러한 정보의 배포를 통제할 수 있도록 사용자에게 편리한 인터페이스를 제공할 것과 디자이너와 구현자들이 이 분야에서 특히 조심할 것을 강력히 추천한다. 전례를 보면 이 분야에서의 에러는 종종 심각한 보안 및/또는 사생활 보호 문제가 되며, 또한 구현자 회사의 평판에 심각한 손상을 입힌다.

15.7 Accept 헤더와 연결된 사생활 보호의 이슈

Accept request-headers 는 사용자에 대한 정보를 접근하는 모든 서버에 노출시킬 수 있다. 특히 Accept-Language 헤더는 특정 언어의 이해는 특정 인종 그룹의 멤버십과 강하게 상호 연관되어 있기 때문에 사용자가 사적인 것으로 간주하는 정보를 노출할 수 있다. 요구를 발송할 때 마다 Accept-Language 헤더의 내용을 설정할 수 있는 선택 사항을 제공하는 사용자 에이전트는 설정 과정에 사용자가 사생활 보호를 상실할 수도 있음을 인지할 수 있도록 하는 메시지를 포함하도록 강력히 추천한다. 사생활 보호 손실을 제한할 수 있는 접근법은 사용자 에이전트가 기본적으로 Accept-Language 헤더 발송을 생략할 수 있게 하는 것이며 서버가 생성한 Vary response-headers 를 검색하여 이러한 헤더 발송이 서비스의 질을 향상시키는 것임을 알게 된다면 사용자에게 Accept-Language 헤더의 서버 발송을 시작할 것인지 질문하는 것이다.

서버는 모든 요구에 발송하는 정교하고 user-customized 된 Accept 헤더 필드를, 특히 이것이 품질 값을 포함하고 있다면, 비교적 신뢰할 수 있고 오래 지속되는 사용자 식별자로 이용한다. 이러한 사용자 식별자는 내용 제공자가 클릭-흔적 추적(click-trail tracking)을 할 수 있도록 하며 상호 협력하는 내용 제공자가 서버에 걸쳐 클릭-흔적을 일치시키거나 개인 사용자의 폼을 제출할 수 있도록 한다. 프락시를 사용하지 않는 많은 사용자의 경우 사용자 에이전트를 운영하는 호스트의 네트워크 주소를 오래 지속되는 사용자 식별자로 사용할 수 있음을 주의한다. 프락시를 사생활 보호를 향상시키기 위해 사용하는 환경에서 사용자 에이전트는 사용자에게 Accept 헤더를 설정할 수 있도록 할 때 상당히 조심해야 한다. 최대한도의 사생활 보호 조치로 프락시는 중계되는 요구에서 Accept 헤더를 여과시킬 수 있다. 높은 수준의 헤더 설정 방법을 제공하는 일반 목적의 사용자 에이전트는 사용자에게 사생활 보호의 손실을 경고해야 한다.

[Page 143]

15.8 DNS Spoofing(속이기)

HTTP 를 사용하는 클라이언트는 Domain Name Service 에 상당히 의존하고 있으며 일반적으로 교묘하게 IP 주소와 DNS 이름을 잘못 연관시킨 것에 기반을 둔 보안 공격에 취약하다. 클라이언트는 IP number/DNS name 상관 관계가 지속적으로 유효하다고 가정할 때 주의해야 한다. 특히 HTTP 클라이언트는 이전의 호스트 이름 조회 결과를 캐시하기보다는 이름 분석자(name resolver)가 IP number/DNS name 상관 관계를 확인하는 것에 의존해야 한다. 많은 플랫폼이 적절한 경우 호스트 이름 검색을 지역에 캐시할 수 있게 하며 그렇도록 환경을 설정해야 한다. 그러나 이러한 조회는 네임 서버(name server)가 보고한 TTL (Time To Live) 정보가 캐시된 정보에 따라 유용한 것으로 유지되고

있는 경우에만 캐시해야 한다. HTTP 클라이언트가 성능 향상을 위해 호스트 이름 조회 결과를 캐시하였으면 DNS 가 보고한 TTL 정보를 반드시 준수해야 한다.

HTTP 클라이언트가 이 원칙을 준수하지 않으면 이전에 접근한 서버의 IP 주소가 변경될 경우 속임을 당할 수 있다. 네트워크 주소 숫자의 변경이 점점 더 일반적인 것이 되고 있기 때문에 이러한 유형의 공격은 증가할 것이다. 이 필요 조건을 준수하는 것이 잠재적인 보안 취약성을 감소시켜 준다.

이 필요 조건은 또한 동일한 DNS 이름을 사용하는 중복된 서버의 클라이언트 행태의 load-balancing 을 향상시키며 그 전략을 사용하는 사이트에 접근했을 때 사용자가 실패를 경험할 가능성을 감소시켜 준다.

15.9 Location 헤더와 Spoofing(속이기)
단일 서버가 서로 신뢰하지 않는 복수의 조직을 지원하고 있다면 서버는 언급한 조직의 통제 하에서 생성되는 응답의 Location 및 Content-Location 헤더 값을 다른 조직이 권한이 없는 자원을 무효화하려 시도하지 말도록 하기 위해 점검해야 한다.

16 감사의 말

이 규격은 첨가된 BNF 및 David H. Crocker 이 RFC 822 에서 규정한 일반 구축법을(generic constructs) 많이 사용하고 있다. 마찬가지로 Nathaniel Borenstein 와 Ned Freed 의 MIME 이 제공하는 많은 규정을 재사용하였다. 우리는 이 규격에 MIME 을 포함하여 HTTP 와 인터넷 전자 우편 메시지 포맷과의 관계에 대한 이전의 혼란이 감소하리라 기대한다.

[Page 144]

HTTP 규약은 지난 4 년간 비약적인 발전을 해 왔다. 이 규약은 크고 활동적인 개발자 공동체의 혜택을 많이 받았으며 이 공동체가 HTTP 및 일반적인 World Wide Web 의 성공에 대한 찬사를 들어야 한다. Marc Andreessen, Robert Cailliau, Daniel W. Connolly, Bob Denny, John Franks, Jean-Francois Groff, Phillip M. Hallam-Baker, Hakon W. Lie, Ari Luotonen, Rob McCool, Lou Montulli, Dave Raggett, Tony Sanders, Marc VanHeyningen 등에게 이 규약의 첫 측면을 규정하는 데 노력한 데 대하여 특별한 감사를 표한다. 이 문서는 HTTP-WG 에 참가한 많은 사람들의 코멘트에서 많은 혜택을 받았다. 이전에 언급한 사람들 이외에 다음의 사람들이 이 규격에 공헌했다.

Gary Adams
Harald Tveit Alvestrand
Keith Ball Jean-Philippe
Brian Behlendorf
Paul Burchard
Maurizio Codogno
Mike Cowlishaw
Roman Czyborra
Michael A. Dolan
David J. Fiander
Alan Freier
Marc Hedlund
Greg Herlihy
Koen Holtman
Albert Lunde
John C. Mallery
Martin-Flatin
Larry Masinter
Mitra
David Morris
Gavin Nicol
Bill Perry
Jeffrey Perry
Scott Powers
Owen Rees
Luigi Rizzo
David Robinson
Marc Salomon

Alex Hopmann
Bob Jernigan
Shel Kaphan
Rohit Khare
John Klensin
Martijn Koster
Alexei Kosut
David M. Kristol
Daniel LaLiberte
Ben Laurie
Paul J. Leach
Daniel DuBois
Rich Salz
Allan M. Schiffman
Jim Seidman
Chuck Shotton
Eric W. Sink
Simon E. Spero
Richard N. Taylor
Robert S. Thau
Bill (BearHeart) Weinman
Francois Yergeau
Mary Ellen Zurko

이 규격의 대부분의 캐시 디자인에 관한 내용이나 프리젠테이션은 다음의 사람들이 준 제안 및 코멘트에 기초하였다. - Shel Kaphan, Paul Leach, Koen Holtman, David Morris, Larry Masinter.

[Page 145]

이 규격의 대부분의 영역에 관한 것은 처음 Ari Luotonen 이 행한 연구에 기초하고 있으며 Steve Zilles 로부터 추가적인 정보를 구했다. Palo Alto 의 "cave men"에게 감사한다. 당신은 당신이 누구인지 알고 있다.

Jim Gettys (이 문서의 현재 편집장) 은 이전 편집장인 Roy Fielding 에게 특별히 감사하며 John Klensin, Jeff Mogul, Paul Leach, Dave Kristol, Koen Holtman, John Franks, Alex Hopmann 및 Larry Masinter 에게 그들의 도움에 감사한다.

17 참고 문헌

[1] Alvestrand, H., "Tags for the identification of languages", RFC1766, UNINETT, March 1995.

[2] Anklesaria, F., McCahill, M., Lindner, P., Johnson, D., Torrey, D., and B. Alberti. "The Internet Gopher Protocol: (a distributed document search and retrieval protocol)", RFC 1436, University of Minnesota, March 1993.

[3] Berners-Lee, T., "Universal Resource Identifiers in WWW", A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, June 1994.

[4] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[5] Berners-Lee, T., and D. Connolly, "HyperText Markup Language Specification - 2.0", RFC 1866, MIT/LCS, November 1995.

[6] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0.", RFC 1945 MIT/LCS, UC Irvine, May 1996.

[7] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, Innosoft, First Virtual, November 1996.

[8] Braden, R., "Requirements for Internet hosts - application and support", STD 3, RFC 1123, IETF, October 1989.

[9] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[Page 146]

[10] Davis, F., Kahle, B., Morris, H., Salem, J., Shen, T., Wang, R., Sui, J., and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification", (v1.5), Thinking Machines Corporation, April 1990.

[11] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

[12] Horton, M., and R. Adams. "Standard for interchange of USENET messages", RFC 1036, AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[13] Kantor, B., and P. Lapsley. "Network News Transfer Protocol." A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego, UC Berkeley, February 1986.

[14] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, University of Tennessee, November 1996.

[15] Nebel, E., and L. Masinter. "Form-based File Upload in HTML", RFC 1867, Xerox Corporation, November 1995.

[16] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/ISI, August 1982.

[17] Postel, J., "Media Type Registration Procedure", RFC 2048, USC/ISI, November 1996.

[18] Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, USC/ISI, October 1985.

[19] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/ISI, October 1994.

[20] Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[21] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[22] ISO-8859. International Standard -- Information Processing -- 8-bit Single-Byte Coded Graphic Character Sets --

Part 1: Latin alphabet No. 1, ISO 8859-1:1987.

Part 2: Latin alphabet No. 2, ISO 8859-2, 1987.

Part 3: Latin alphabet No. 3, ISO 8859-3, 1988.

Part 4: Latin alphabet No. 4, ISO 8859-4, 1988.

[Page 147]

Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988.

Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987.

Part 7: Latin/Greek alphabet, ISO 8859-7, 1987.

Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988.

Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

[23] Meyers, J., and M. Rose "The Content-MD5 Header Field", RFC 1864, Carnegie Mellon, Dover Beach Consulting, October, 1995.

[24] Carpenter, B., and Y. Rekhter, "Renumbering Needs Work", RFC 1900, IAB, February 1996.

[25] Deutsch, P., "GZIP file format specification version 4.3." RFC 1952, Aladdin Enterprises, May 1996.

[26] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP Latency. Computer Networks and ISDN Systems, v. 28, pp. 25-35, Dec. 1995. Slightly revised version of paper in Proc. 2nd International WWW Conf. '94: Mosaic and the Web, Oct. 1994, which is available at HTTPLatency.html.

[27] Joe Touch, John Heidemann, and Katia Obraczka, "Analysis of HTTP Performance", <URL: >, USC/Information Sciences Institute, June 1996

[28] Mills, D., "Network Time Protocol, Version 3, Specification, Implementation and Analysis", RFC 1305, University of Delaware, March 1992.

[29] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3." RFC 1951, Aladdin Enterprises, May 1996.

[30] Spero, S., "Analysis of HTTP Performance Problems" <>.

[31] Deutsch, P., and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, Aladdin Enterprises, Info-ZIP, May 1996.

[32] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., and L. Stewart, "An Extension to HTTP : Digest Access Authentication", RFC 2069, January 1997.

[Page 148]

18 저자의 주소

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425, USA

Fax: +1 (714) 824-4056
EMail:

Jim Gettys
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail:

Jeffrey C. Mogul
Western Research Laboratory
Digital Equipment Corporation
250 University Avenue
Palo Alto, California, 94305, USA
EMail:

Henrik Frystyk Nielsen
W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail:

Tim Berners-Lee
Director, W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail:

[Page 149]

19 부록

19.1 Internet Media Type message/http

HTTP/1.1 규약을 규정하는 것과 더불어 이 문서는 Internet media type "message/http"에 대한 규격으로도 사용한다. 다음 사항을 IANA 에 등록해야 한다.

Media Type name: message
Media subtype name: http
Required parameters: none
Optional parameters: version, msgtype
version: 동봉된 메시지의 HTTP-Version 번호(예를 들어 1.1)가 없으면 버전은 본문의 첫 라인으로 결정한다.
msgtype: 메시지 유형 - "요구" 또는 "응답". 표시되지 않았으면 유형은 본문의 첫 라인으로 결정한다
Encoding considerations: "7bit", "8bit", 또는 "binary" 만 허용된다.
Security considerations: none

19.2 Internet Media Type multipart/byteranges

HTTP 메시지가 복수 영역의 내용(예를 들어 복수의 중첩되지 않는 영역에 대한 요구의 응답)을 포함하고 있을 때, 이것은 multipart MIME 메시지로서 전송된다. 이러한 목적의 multipart media type 을 "multipart/byteranges".라고 부른다.

Multipart/byteranges media type 은 둘 또는 그 이상의 부분을 포함하며 각각 자신의 Content-Type 과 Content-Range 필드를 가진다. 이 부분들은 MIME 경계 파라미터(boundary parameter)를 이용하여 구분한다.

Media Type name:	multipart
Media subtype name:	byteranges
Required parameters:	boundary
Optional parameters:	none
Encoding considerations:	"7bit", "8bit", 또는 "binary" 만 허용된다.
Security considerations:	none

[Page 150]

예를 들면,

HTTP/1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000

...첫 영역...

```
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000

...두 번째 영역
--THIS_STRING_SEPARATES--
```

19.3 Tolerant Applications

이 문서가 HTTP/1.1 메시지를 생성하는 데 필요한 조건들을 명시하고 있지만 모든 애플리케이션이 이것을 정확하게 구현하지는 않을 것이다. 따라서 우리는 실제적인 애플리케이션이 이러한 이탈(deviations)이 명확하게 해석될 수 없을 때는 언제나 이탈에 대해 관대할 것을 추천한다.

클라이언트는 Status-Line 을 분석하는 데 관대하고 서버는 Request-Line 을 분석하는 데 관대해야 한다. 특히 둘 모두 필드 사이에 비록 단 하나의 SP 만 필요하다 하더라도 SP 및 HT 문자의 수에 관계없이 이를 수용하여야 한다.

Message-header 의 라인 종결자는 연속적인 CRLF 이다. 그러나 우리는 애플리케이션이 이러한 헤더를 분석할 때 단일 LF 를 라인 종결자로 인식하고 앞에 있는 CR 을 무시할 것을 추천한다.

Entity-body 의 문자 집합은 해당 본문에 사용된 문자 집합의 최저 공동 명칭(lowest common denominator)으로 라벨을 붙여야 하나 예외적으로 US-ASCII 또는 ISO-8859-1 라벨에는 아무 것도 붙이지 않는 것이 좋다.

날짜의 분석과 인코딩 필요 조건에 관한 추가 규칙 및 날짜 인코딩과 관련된 다른 잠재적 문제점은 다음과 같다.

- HTTP/1.1 클라이언트와 캐시는 향후 50년 이후 이상의 RFC-850 날짜는 사실상 과거의 날짜라고 가정해야 한다.(이것이 "2000 년" 문제를 해결하는 데 도움을 준다.)

- HTTP/1.1 구현 방식은 내부적으로 적절한 값보다 이전으로 분석된 Expires 날짜를 표시할 수는 있으나 절대로 적절한 값보다 이후 날짜를 표시해서는 안 된다.

- 유효일에 관련된 모든 계산은 GMT로 해야 한다. 지역적인 시간대는 경과 시간이나 유효 시간을 계산하거나 비교하는 데 영향을 미쳐서는 안 된다.

- HTTP 헤더가 부정확하게 GMT 이외의 시간 대 날짜 값을 가지고 있다면 가장 조심스러운 환산 방법을 사용하여 GMT로 변환하여야 한다.

19.4 HTTP 엔터티와 MIME 엔터티의 차이점

HTTP/1.1 은 인터넷 메일(RFC 822) 및 다목적 인터넷 메일 확장(Multipurpose Internet Mail Extensions (MIME))을 위해 규정된 많은 구성물(construct)을 이용하여 엔터티가 공개된 다양한 표시 방식 및 확장 가능한 메커니즘을 통하여 전달될 수 있도록 한다. 그러나 MIME [7] 또한 전자 우편에 대해 논의하고 있으며 HTTP 에는 MIME 에서 기술된 것과 상이한 기능이 몇몇 있다. 이러한 차이점은 바이너리 접속의 성능을 최적화하고, 새로운 media type 을 사용하는 최대한의 자유를 허용하며, 날짜 비교를 용이하게 하며, 이전 HTTP 서버 및 클라이언트의 행태를 인정할 수 있도록 주의 깊게 선택하였다.

이 부록은 HTTP 와 MIME 의 차이를 기술하고 있다. 프락시에서 엄격한 MIME 환경으로의 게이트웨이는 이러한 차이점을 인식하고 필요하다면 적절한 변환을 해 주어야 한다. 프락시와 MIME 환경으로부터 HTTP 로의 게이트웨이 또한 약간의 변환이 필요할 수 있으므로 이 차이점을 인지할 필요가 있다.

19.4.1 규범적인 폼으로 변환

MIME 은 인터넷 메일 엔터티를 전송되기 전에 규범적인 폼(canonical form)으로 변환할 것을 요구한다. 이 문서의 3.7.1 절은 HTTP 로 전송될 때 폼에 사용할 수 있는 "text" media type 의 subtype 을 기술하고 있다. MIME 에서 텍스트 유형의 내용은 줄바꿈을 CRLF 로 표시해야 하고 줄바꿈 이외는 CR 또는 LF 를 사용하지 말아야 한다. HTTP 는 메시지가 HTTP 를 통하여 전달될 때 텍스트 내용 안에서 줄바꿈을 표시하기 위해 CRLF, 단일 CR 및 LF 를 허용한다.

가능하다면 프락시에서 엄격한 MIME 환경으로의 게이트웨이는 3.7.1 절에서 기술한 text media type 내의 모든 줄바꿈을 CRLF 의 MIME 정규 폼으로 해석해야 한다. 그러나 Content-Encoding 이 있으면 이 해석이 복잡해질 수 있으며 또한 HTTP 가 multi-byte 문자 집합의 경우처럼 octets 13 및 10 을 CR 및 LF 를 표시하는데 사용하지 않는 다른 문자 조합을 허용한다는 사실 때문에 복잡해질 수 있다.

[Page 152]

19.4.2 날짜 형식의 변환

HTTP/1.1 은 한정된 날자 포맷(3.3.1)을 사용하여 날짜 비교 과정을 단순화시켜 준다. 프락시에서 다른 규약의 게이트웨이는 확실하게 HTTP/1.1 포맷 중 하나를 따르는 메시지에 Date 헤더 필드가 있고 필요하다면 날짜를 재기입하도록 해야 한다.

19.4.3 Content-Encoding 소개

MIME 은 HTTP/1.1 의 Content-Encoding 헤더 필드와 동등한 개념을 아무것도 포함하고 있지 않다. 이것이 media type 에 변경자로서 작동하기 때문에 HTTP 에서 MIME 을 준수하는 규약으로의 프락시나 게이트웨이는 반드시 Content-Encoding 헤더 필드의 값을 변경하거나 메시지를 전달하기 전에 entity-body 를 해독해야 한다. (인터넷 메일을 위한 몇몇 실험적인 응용프로그램은 media type 파라미터 ";conversions=<content-coding>"를 Content-Encoding 과 동등한 기능 수행을 위해 사용한다. 그러나 이 파라미터는 MIME 의 일부분이 아니다.)

19.4.4 No Content-Transfer-Encoding

HTTP 는 MIME 의 Content-Transfer-Encoding (CTE) 필드를 사용하지 않는다. HTTP 에서 MIME 을 준수하는 규약으로의 프락시나 게이트웨이는 응답 메시지를 HTTP 클라이언트에게 배달하기 전에 반드시 모든 non-identity CTE ("quoted-printable" 또는 "base64") 인코딩을 제거해야 한다. HTTP 에서 MIME 을 준수하는 규약으로의 프락시나 게이트웨이는 메시지가 해당 규약에서 안전하게 전송될 수 있도록, 또한 정확한 포맷과 인코딩이 되도록 확실하게 해야 한다. 여기서 "안전한 전송"은 사용되고 있는 규약의 제한 사항에 의해 규정된다. 이러한 프락시나 게이트웨이는 그렇게 하는 것이 목적지 규약으로 안전하게 전송할 가능성을 높여준다면 데이타에 적절한 Content-Transfer-Encoding 라벨을 붙어야 한다.

19.4.5 Multipart Body-Part 의 HTTP 헤더 필드

MIME 에서 multipart body-parts 의 거의 모든 헤더 필드는 필드 이름이 "Content-"로 시작하지 않는 한 보통 무시된다. HTTP/1.1 에서는 multipart body-parts 가 해당 부분의 의미에 상당히 중요한 HTTP 헤더 필드를 포함할 수 있다.

[Page 153]

19.4.6 Transfer-Encoding 소개

HTTP/1.1 은 Transfer-Encoding 헤더 필드(14.40 절)를 소개하고 있다. HTTP/1.1 은 프락시/게이트웨이가 MIME 을 준수하는 규약을 통하여 메시지를 전달하기 전에 모든 전송 코딩을 제거해야만 한다는 점을 소개하고 있다.

덩어리 전송 코딩(3.6 절)을 해독하는 절차를 다음과 같이 유사 코드(pseudo-code) 형식으로 표현할 수 있다.

```
length := 0
read chunk-size, chunk-ext (if any) and CRLF
while (chunk-size > 0) {
        read chunk-data and CRLF
        append chunk-data to entity-body
        length := length + chunk-size
        read chunk-size and CRLF
}
read entity-header

while (entity-header not empty) {
        append entity-header to existing header fields
        read entity-header
}
Content-Length := length
Remove "chunked" from Transfer-Encoding
```

19.4.7 MIME-Version

HTTP 는 MIME 을 준수하는 규약이 아니다.(부록 19.4 절 참조). 그러나 HTTP/1.1 메시지는 단일 MIME-Version general-header 필드를 포함하여 메시지를 생성하기 위하여 어떤 MIME 규약 버전을 사용했는지 표시할 수 있다. MIME-Version 헤더 필드의 사용은 메시지가 MIME 규약에 전적으로 따르고 있다는 것을 표시한다. 프락시/게이트웨이는 HTTP 메시지를 엄격한 MIME 환경으로 전송할 때 완전한 규약 준수(가능 하다면)를 확실하게 할 책임이 있다.

```
MIME-Version            = "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

MIME 버전 "1.0"이 HTTP/1.1 에서 사용할 때 기본 값이다. 그러나 HTTP/1.1 메시지 분석 및 의미는 MIME 규격이 아닌 이 문서에 규정되어 있다.

19.5 HTTP/1.0 이후 변경 사항

이 절에서 HTTP/1.0 과 HTTP/1.1 사이의 주요 차이점을 요약하였다.

[Page 154]

19.5.1 복수의 홈을 가진 웹 서버를 단순하게 하기 위한 변경 사항 및 IP 주소 보존 클라이언트와 서버가 Host request-header 를 지원해야 한다는 요구 조건 때문에 Host request-header(14.23 절)가 빠졌으면 에러를 발생시킨다. 또한 절대 URI(5.1.2 절)는 이 규격이 규정한 변경 사항 중 가장 중요한 것이다.

이전 HTTP/1.0 클라이언트는 IP 주소와 서버의 일대일 관계(one-to-one relationship)를 가정했다. 요구를 발송하고자 하는 서버와 요구가 발송된 IP 주소를 구별할 확립된 메커니즘이 없었다. 위에 대략적으로 설명한 변경 사항은 인터넷이, 단일 호스트에 복수의 IP 주소를 할당하는 것이 심각한 문제를 발생시켰던 이전 HTTP 클라이언트가 더 이상 보편적인 것이 아닐 때, 단일 IP 주소가 복수의 웹 사이트를 지원하여 대량의 웹 서버 운영을 단순하게 할 것이다. 인터넷은 또한 단지 루트 수준 HTTP URl 에 사용할 특수 용도의 도메인 이름을 사용할 목적으로 할당되었던 IP 주소를 복구할 수 있게 될 것이다. 웹의 성장 속도 및 이미 배치된 서버의 숫자를 감안할 때 HTTP 의 모든 구현 방식(기존 HTTP/1.0 애플리케이션 갱신을 포함하여)이 다음의 필요 조건을 정확하게 구현하는 것이 매우 중요하다.

- 클라이언트와 서버 모드 반드시 Host request-header를 지원해야 한다.
- HTTP/1.1 요구에는 반드시 Host request-header를 사용해야 한다.
- HTTP/1.1 요구가 Host request-header를 포함하고 있지 않으면 반드시 에러 메시지400 (Bad Request)을 발생시켜야 한다.
- 서버는 반드시 절대 URI를 수용해야 한다.

[Page 155]

19.6 추가 기능

이 부록은 기존 HTTP 구현 방식이 사용하고 있는 규약 요소를 문서화하고 있지만 대부분의 HTTP/1.1 애플리케이션 전반에 걸쳐 일관성 있고 정확한 것은 아니다. 구현자가 이러한 기능을 인지해야 하지만 다른 HTTP/1.1 애플리케이션 내에서의 이러한 기능의 존재나 상호 운영성(interoperability)에 의존할 수는 없다. 이것들 중 몇몇은 제안된 실험적 기능을 기술하고 있으며 다른 것들은 현재 기본 HTTP/1.1 규격에 언급되었지만 실험적으로 배포했을 때 부족한 것으로 발견된 기능을 기술하고 있다.

19.6.1 추가적인 요구 method

19.6.1.1 PATCH

PATCH method 는 엔터티가 Request-URI 가 식별한 자원의 원래 버전과 PATCH 작업을 적용했을 때 희망하는 자원의 차이점 목록을 포함하고 있다는 것을 제외하고는 PUT 과 유사하다. 차이점 목록은 엔터티의 media type(예를 들어 "application/diff")에서 규정한 포맷이며 서버가 자원의 원래 버전을 희망하는 버전으로 변화하는 데 필요한 변경 사항을 재생성할 수 있도록 하는 충분한 정보를 반드시 포함해야 한다.

요구가 캐시를 통과하거나 Request-URI 가 현재 캐시된 엔터티를 식별하면 해당 엔터티는 반드시 캐시에서 삭제해야 한다. 이 method 에 대한 응답을 캐시할 수 없다.

패치한 자원을 배치하는 방법 및 선행자에 미치는 영향에 대한 실제적인 method 는 전적으로 원서버가 규정한다. 만약 패치하고 있는 자원의 원래 버전이 Content-Version 헤더 필드를 포함하고 있다면 요구 엔터티는 반드시 원래 Content-Version 헤더 필드에 상응하는 Derived-From 헤더 필드를 포함해야 한다. 애플리케이션은 이러한 필드를 버전 부여 관계 및 버전 충돌을 해결하는 데 사용하도록 추천한다.

PATCH 요구는 8.2 절에서 설정한 메시지 전송 필요 조건에 따라야 한다.

PATCH 를 구현하는 캐시는 13.10 절 PUT 에서 규정한 대로 캐시된 응답을 무효화해야 한다.

19.6.1.2 LINK

LINK method 는 Request-URI 가 식별하는 기존 자원과 다른 기존 자원의 하나 또는 그 이상의 Link 관계를 확립한다.

[Page 156]

LINK 와 자원 사이의 링크를 설정할 수 있도록 하는 다른 method 와의 차이점은 LINK method 는 어떠한 message-body 도 요구와 함께 발송하지 못하도록 한다는 것과 직접적으로 새로운 자원을 생성하지 않는다는 것이다.

요구가 캐시를 통과하거나 Request-URI 가 현재 캐시된 엔터티를 식별하면 해당 엔터티는 반드시 캐시에서 삭제해야 한다. 이 method 에 대한 응답을 캐시할 수 없다.

LINK 를 구현하는 캐시는 13.10 절 PUT 에서 규정한 대로 캐시된 응답을 무효화해야 한다.

19.6.1.3 UNLINK

UNLINK method 는 Request-URI 가 식별하는 기존 자원과 다른 기존 자원의 하나 또는 그 이상의 Link 관계를 삭제한다. 이러한 관계는 LINK 를 이용하거나 Link 헤더를 지원하는 다른 method 에 의하여 확립되었을 수 있다. 자원에 대한 링크를 삭제하는 것은 자원이 더 이상 존재하지 않는다거나 향후 참조를 위해 접근할 수 없다는 것을 의미하는 것은 아니다.

요구가 캐시를 통과하거나 Request-URI 가 현재 캐시된 엔터티를 식별하면 해당 엔터티는 반드시 캐시에서 삭제해야 한다. 이 method 에 대한 응답을 캐시할 수 없다.

UNLINK 를 구현하는 캐시는 13.10 절 PUT 에서 규정한 대로 캐시된 응답을 무효화해야 한다.

19.6.2 Additional Header Field Definitions

19.6.2.1 Alternates

Alternates response-header 필드를 원서버가 클라이언트에게 요구 받은 자원을 표시할 수 있는 다른 방식을 각각의 독특한 속성과 함께 알려 주는 수단으로 제의하였다. 이렇게 하여 사용자 에이전트가 사용자의 희망 사항에 더 적합한(12 장에서 에이전트가 주도하는 협상(agent-driven negotiation)으로 기술되었다.) 다른 표시 방식을 계속적으로 선택할 수 있는 신뢰할 수 있는 방안을 제공한다.

[Page 157]

Alternates 헤더 필드는 응답의 해석이나 사용할 수 있는 표시 방법에 영향을 미치지 않고 메시지에 둘 다 존재할 수 있다는 점에서 Vary 헤더 필드와 직교하고 있다. Alternates가 Vary 필드가 제공하는 유형 및 언어와 같이 공동 차원(common dimensions)에 걸쳐 변형될 수 있는 자원에 관한 서버 주도 협상(server-driven negotiation)에 대해 상당한 개선을 할 수 있을 것으로 기대되고 있다.

Alternates 헤더 필드는 향후 규격에서 규정될 예정이다.

19.6.2.2 Content-Version

Content-Version entity-header 필드는 진행되고 있는 엔터티의 해석에 관련된 버전 태그를 규정한다. 19.6.2.3 절에서 기술한 Derived-From 필드와 더불어 이것은 사람들이 작업을 반복적인 절차로 동시에 진행할 수 있도록 한다. 이 필드는 특정 작업의 진행이 파생된 작업이나 다른 표현 방법에 의한 해석이 아닌 단일 경로를 따를 수 있도록 하는 데 사용한다.

```
Content-Version          = "Content-Version" ":" quoted-string
```

Content-Version 필드의 사용 예는 다음과 같다.

```
Content-Version: "2.1.2“
Content-Version: "Fred 19950116-12:26:48“
Content-Version: "2.5a4-omega7“
```

19.6.2.3 Derived-From

Derived-From entity-header 필드는 발송측이 변경하기 전 상태에서 엔터티가 파생된 자원의 버전 태그를 표시하기 위해 사용한다. 이 필드는 또한 자원에 대한 계속적인 변경을, 특히 이러한 변경이 복수의 자원과 병행하여 이루어 질 때 혼합하는 과정을 관리할 수 있도록 한다.

```
Derived-From             = "Derived-From" ":" quoted-string
```

이 필드의 사용 예는,

```
Derived-From: "2.1.1“
```

PUT 및 PATCH 요구에는 발송되는 엔터티가 이전에 동일한 URI에서 조회된 것이고 마지막으로 조회했을 때 Content-Version 헤더를 포함하고 있었다면 Derived-From 필드가 필요하다.

[Page 158]

19.6.2.4 Link

Link entity-header 필드는 보통 요구 받은 자원과 다른 자원과의 관계인 두 자원과의 관계를 기술하는 수단을 제공한다. 엔터티는 복수의 Link 값을 포함할 수 있다. 메타 정보 수준의 링크는 대개 계서적 구조나 항해 경로(navigation paths)와 같은 관계를 표시한다. Link 필드는 의미상 HTML.[5]의 <LINK> 요소와 동등하다.

```
Link             = "Link" ":" #("<" URI ">" *( ";" link-param )
link-param       = ( ( "rel" "=" relationship )
                   | ( "rev" "=" relationship )
                   | ( "title" "=" quoted-string )
```

```
                    | ( "anchor" "=" <"> URI <"> )
                    | ( link-extension ) )
  link-extension  = token [ "=" ( token | quoted-string ) ]
  relationship    = sgml-name
                    | ( <"> sgml-name *( SP sgml-name) <"> )
  sgml-name       = ALPHA *( ALPHA | DIGIT | "." | "-" )
```

관계 값은 대소문자를 구별하며 sgml-name 구문법의 제한 사항 내에서 확장될 수 있다. "title" 파라미터는 링크의 목적지를 표시하는 데 사용하여 사람이 읽을 수 있는 메뉴에서 식별자로 사용할 수 있다. 앵커 파라미터(anchor parameter)는 이 자원 또는 제 삼의 자원의 한 단편과 같이 현재의 전체 자원이 아닌 소스 앵커(source anchor)를 표시하는 데 사용한다.

사용 예를 보면,

```
  Link: <>; rel="Previous“
  Link: <>; rev="Made"; title="Tim Berners-Lee“
```

첫 번째 예는 chapter2 가 논리적 운항 경로에서 이 자원의 이전 것임을 표시한다. 두 번째 예는 자원을 사용할 수 있도록 만드는 책임을 진 사람을 주어진 전자우편 주소로 식별한다는 것을 표시한다.

19.6.2.5 URI

이 규격의 이전 버전에서 URI 헤더 필드는 기존 Location, Content-Location, Vary 헤더 필드 및 향후 Alternates 의 결합체로 사용했었다.

[Page 159]

이 필드의 주요 목적은 이름 및 미러 위치(name and mirror location)를 포함하는 자원의 추가 URI 목록을 포함하는 것이었다. 그러나 이 단일 필드 내에 많은 별도의 기능을 결합하는 것은 이러한 기능을 일관성 있고 정확하게 구현하는 데 장애물이 되고 있다는 것이 명백해졌다. 더더욱 우리는 이름 및 미러 위치의 식별은 Link 헤더 필드를 통하여 더 잘 수행할 수 있다고 믿는다. 따라서 URI 헤더 필드는 그러한 필드를 선호하여 경시되고 있다.

```
  URI-header     = "URI" ":" 1#( "<" URI ">" )
```

19.7 추가 헤더 필드 정의

이전 버전에 따르도록 강제하는 것은 규약 규격의 범위를 벗어나는 것이다. 그러나 HTTP/1.1 은 이전 버전을 쉽게 지원하도록 정교하게 디자인 되었다. 이 규격을 작성하는 순간 우리는 상업적 HTTP/1.1 서버가 다음 사항을 수행할 것으로 기대하고 있음에 주의하기 바란다.

- HTTP/0.9, 1.0, 또는 1.1 요구의 Request-Line 포맷을 인식한다.
- HTTP/0.9, 1.0, 또는 1.1 포맷으로 된 어떠한 요구도 이해한다.
- 클라이언트가 사용하는 주요 버전에서 적절하게 메시지에 응답한다.

또한 우리는 HTTP/1.1 클라이언트가 다음을 수행하기를 기대한다.

- HTTP/1.0 및 1.1 응답의 Status-Line 포맷을 인지한다.
- HTTP/0.9, 1.0, 또는 1.1 포맷으로 된 어떠한 요구도 이해한다.

대부분의 HTTP/1.0 구현 방식은 각각의 연결은 요구가 발생되기 이전에 클라이언트가 설정하며 응답을 발송한 후 서버가 종료한다. 소수의 구현 방식은 19.7.1.1 에서 기술한 Keep-Alive 지속적 접속의 버전을 구현한다.

[Page 160]

19.7.1 HTTP/1.0 지속적인 연결과의 호환성

몇몇 클라이언트 및 서버는 이전 HTTP/1.0 클라이언트 및 서버의 지속적 연결과 호환성 유지를 원할 수 있다. HTTP/1.0 의 지속적 연결은 이것이 기본 행태가 아니기 때문에 반드시 명백하게 협상해야 한다. HTTP/1.0 지속적 접속의 실험적 구현 방법은 잘못이었으며 HTTP/1.1 은 이러한 문제를 인증하도록 디자인 되었다. 문제는 몇몇 기존 1.0 클라이언트가 Keep-Alive 를 Connection 을 이해하지 못하는 프락시 서버에 발송한다는 것이었다. Keep-Alive 를 발송한 후 이것을 다음의 내부를 향한 서버(inbound server)에 실수도 전달하여 Keep-Alive 연결을 설정하고 HTTP/1.0 프락시가 응답의 종료를 무한정 기다리는 결과를 초래하였다. 결과는 HTTP/1.0 클라이언트가 프락시와 통신할 때 Keep-Alive 를 사용하지 못하도록 하는 것이었다.

그러나 프락시와의 통신은 지속적인 접속의 가장 중요한 용도였기 때문에 이러한 금지 사항은 명백하게 수용할 수 없는 것이었다. 따라서 우리는 Connection 을 무시하는 이전 프락시와 통신할 때도 사용하기에 안전한 지속적인 접속을 바란다는 것을 표시하는 다른 메커니즘이 필요하다. 지속적인 연결(persistent connection)은 HTTP/1.1 메시지의 기본 값이다. 우리는 지속적이지 않은 연결을 위해 새로운 핵심어(Connection: close)를 소개한다.

다음은 원래의 HTTP/1.0 형식 지속적 접속을 기술하고 있다.
원서버와 연결되었을 때 HTTP 클라이언트는 Persist connection-token 에 추가하여 Keep-Alive connection-token 을 발송할 수도 있다.

```
Connection: Keep-Alive
```

그러면 HTTP/1.0 서버가 Keep-Alive connection token 으로 응답하고 클라이언트는 HTTP/1.0 (또는 Keep-Alive) persistent connection 으로 계속 진행할 것이다.

또한 HTTP/1.1 서버는 Keep-Alive connection token 을 수신하자마자 HTTP/1.0 클라이언트와의 지속적인 접속을 확립할 수 있다. 그러나 HTTP/1.0 클라이언트와의 지속적인 접속에는 덩어리 전송 코딩(chunked transfer-coding)을 활용할 수 없기 때문에 각 메시지의 종료 경계(ending boundary)를 표시하기 위해서는 반드시 Content-Length 를 이용하여 표시를 해야 한다. 클라이언트는 HTTP/1.0 프락시 서버가 Connection 헤더 필드를 분석하기 위한 HTTP/1.1 원칙을 따르지 않기 때문에 Keep-Alive connection token 을 프락시 서버에 발송하지 말아야 한다.

[Page 161]

19.7.1.1 The Keep-Alive Header

요구나 응답에 Keep-Alive connection-token 이 전송되었으면 Keep-Alive 헤더 필드가 포함될 수 있다. Keep-Alive 헤더 필드는 다음의 형식을 취한다.

```
Keep-Alive-header       = "Keep-Alive" ":" 0# keepalive-param
keepalive-param         = param-name "=" value
```

Keep-Alive 헤더 자체는 선택 사항이며 파라미터가 발송되었을 때만 사용된다. HTTP/1.1 은 어떠한 파라미터도 규정하지 않고 있다.

Keep-Alive 가 발송되었으면 상응하는 연결 토큰(corresponding connection token)도 반드시 전송되어야 한다. 연결 토큰 없이 수신되었으면 Keep-Alive 헤더를 무시해야 한다.

[Page 162]