1장 웹 브라우저가 메시지를 만든다.

태그	성공과 실패를 결정하는 1% 네트워크 원리 스터디
생성일시	@2022년 6월 1일 오후 1:13
최종편집일시	@2022년 6월 2일 오후 2:39
다이어리 기록	
참고	https://github.com/Stacked-Book/BookRecord https://developer.mozilla.org/ko/docs/Web/HTTP/Methods https://avinetworks.com/glossary/subnet-mask/

▼ **목차**

HTTP 리퀘스트 메시지를 작성한다.

1. 탐험 여행은 URL 입력부터 시작된다.

> URL은 `http://` 로 시작하는 것이라고만 알아두면 되지만 더 자세히는 `file:` `ftp:` `mailto:` 등 다양하게 있다.

❓ 왜 여러개가 존재할까?

> 파일을 다운로드 하는 FTP(File Transfer Protocol)의 클라이언트 기능이나 메일의 클라이언트 기능도 가지고 있기 때문
> 즉 브라우저는 몇 개의 클라이언트 기능을 겸비한 복합적인 클라이언트 소프트웨어이다.

엑세스 하는 방법에는 여러가지가 있다.

- 웹 서버나 FTP 서버에 엑세스 하는 경우에는 서버의 도메인명이나 액세스 하는 파일의 경로명 등을 URL에 포함시킨다.
- 메일의 경우 보내는 상대의 메일 주소를 URL에 포함시킨다는 식
- 필요에 따라 사용자명이나 패스워드, 서버측 포트 번호 등을 쓸 수 있다.

```
// http://사용자명:패스워드@웹서버의 도메인명:포트번호/파일의 경로명
// 사용자명, 패스워드, 포트번호는 생략이 가능하다.
```

```
// HTTP 프로토콜로 웹 서버에 엑세스 하는 경우
http://user:password@www.cyber.co.kr:80/dir/file1.html

// FTP 프로토콜로 파일을 다운로드 하는 경우
ftp://user:password@ftp.cyber.co.kr:21/dir/file1.html

// 클라이언트 PC 자체의 파일에서 데이터를 읽어오는 경우
file://localhost/c:/path/file1.zip

// 메일을 송신하는 경우
mailto:tone@cyber.co.kr

// 뉴스그룹의 기사를 읽는 경우
new:comp.protocols.tcp-ip
```

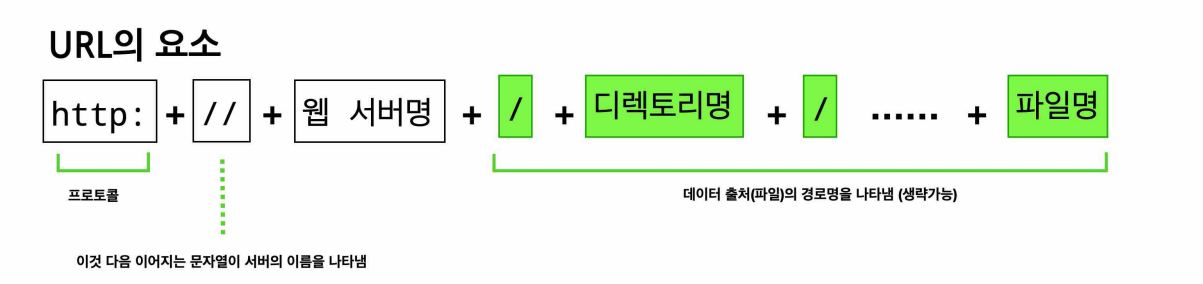
> 단 file:로 시작하는 URL과 같이 액세스할 때 네트워크를 사용하지 않는 것도 있으므로 프로토콜을 나타낸다고 단언할 수는 없다. 즉 액세스 하는 방법이라고 생각하는 것이 좋다.

❓ **프로토콜이란?**

> 통신 동작의 규칙을 정한것을 포로토콜이라고 부른다.

2. 브라우저는 먼저 URL을 해독한다.

> 브라우저가 처음 하는 일은 웹 서버에 보내는 리퀘스트에 메세지를 작성하기 위해 해당 URL을 해독하는 것이다.



`www.lab.cyber.co.kr/dir/file1.html`
→ 서버 주소인 `www.lab.cyber.co.kr` 에 가서 `dir/file1.html` 파일에 액세스한다.

3. 파일명을 생략한 경우

`http://www.lab.cyber.co.kr/dir/` 처럼 사용할 수 있다.

이때 끝이 `/` 로 끝난의미는 `dir/` 의 다음에 써야 할 파일명을 쓰지 않고 생략한다는 것이다.

이렇게 될 경우 어느 파일에 엑세스 해야할지 브라우저는 모르기에 이렇게 생략할 때를 대비해서 파일명을 미리 설정해놓습니다.

예) index.html, default.html

→ `http://www.lab.cyber.co.kr/dir/` → `http://www.lab.cyber.co.kr/dir/index.html`

4. HTTP의 기본 개념

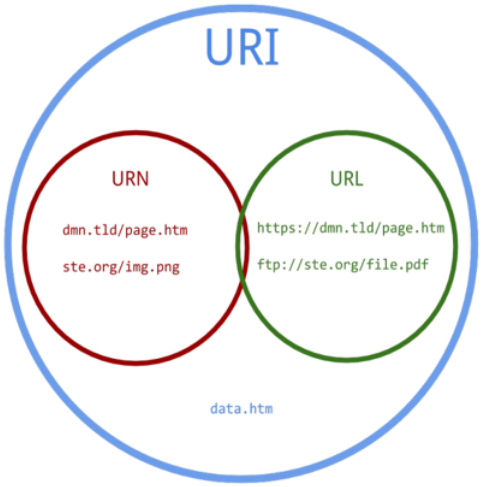
> URL을 해독하면 어디에 엑세스 해야 하는지 판명된다.

- HTTP 프로토콜은 클라이언트와 서버가 주고받는 메시지의 내용이나 순서를 정한것이다.
- 기본적인 개념은 먼저 클라이언트에서 서버를 향해 리퀘스트 메시지를 보낸다.
- 리퀘스트 메시지의 안에는 **무엇을 어떻게** 해서 하겠다는 내용이 써져있다.

❓ **무엇을? URI**

> Uniform Resource Identifier - 통합 자원 식별자의 약자이다.
> 페이지 데이터를 저장한 파일의 이름이나 CGI 프로그램의 파일명을 URI로 쓴다.

- 예시로 `dir/file1.html` 같은 식의 파일명을 말한다.
- URI는 위 내용뿐만 아니라 여기에 http:로 시작하는 URL도 그대로 쓸 수 있다.



❓ **어떻게 해서? HTTP메서드**

> 웹 서버에 어떤 동작을 하고 싶은지 전달

메소드	HTTP 1.0	HTTP 1.1	의미
GET	O	O	URI로 지정한 정보를 도출
POST	O	O	클라이언트에서 서버로 데이터를 송신
HEAD	O	O	GET과 비슷. 단, HTTP 메시지 헤더만 반송하고 데이터의 내용을 돌려보내지 않습니다.
OPTIONS		O	통신 옵션을 통지하거나 조사할 때 사용합니다
PUT	△	O	URI로 지정한 서버의 파일을 치환합니다. - 목적 리소스 모든 현재 표시를 요청 payload로 바꿉니다.
DELETE	△	O	목적 리소스를 삭제합니다.
TRACE		O	목적 리소스의 경로를 따라 메시지 loop-back 테스트를 합니다.
CONNECT		O	암호화한 메시지를 프록시로 전송할 때 이용하는 메소드입니다.

리퀘스트 메시지가 웹 서버에 도착하면 웹 서버는 그 속에 쓰여있는 내용을 해독한다.

결과 데이터를 응답 메시지에 저장한다.

맨 앞에부분에는 실행 결과를 나타내는 Status code(200 Ok, 404 Not Found 등) 포함돼있다.

이러한 응답 메시지가 클라이언트측에 도착하면 브라우저가 메시지를 해석하여 화면에 표시해주면 HTTP 동작은 끝마친다.

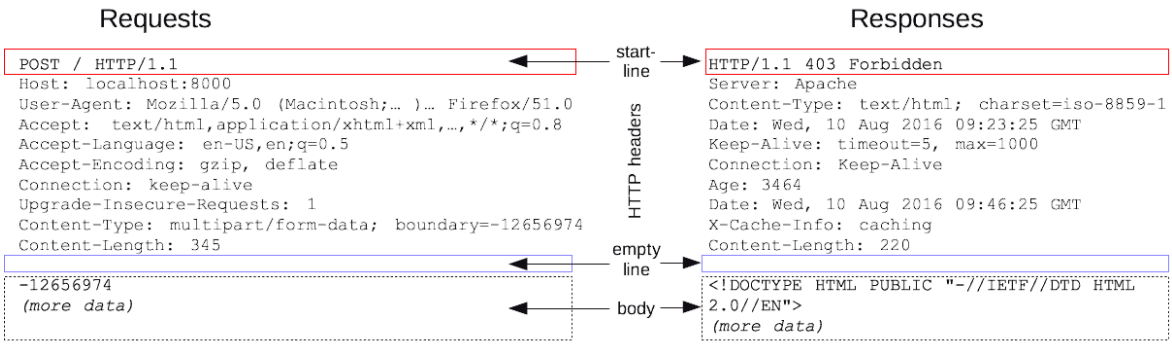
5. HTTP 리퀘스트 메시지를 만든다.

> 브라우저는 URL을 해독하고 웹 서버와 파일명을 판단하면 이것을 바탕으로 HTTP 메시지를 작성한다.

```
// 리퀘스트 메시지 포맷
<메소드><공백><URI><공백><HTTP 버전> // 리퀘스트 라인이며 메소드가 가장중요하다.
<필드명>:<필드값> // '메시지 헤더 부분'
...
...
...
<공백 행>
<메시지 본문> // 메시지 본문 : 클라이언트에서 서버에 송신하는 데이터

// 응답 메시지 포맷
<HTTP 버전><공백><스테이터스 코드><공백><응답 문구> //
<필드명>:<필드값>
...
...
...
<공백 행>
<메시지 본문>  // 메시지 본문 : 서버에서 클라이언트에 송신하는 데이터
```

- 시작 줄(start-line)에는 실행되어야 할 요청, 또는 요청 수행에 대한 성공 또는 실패가 기록돼 있다.
- 옵션으로 HTTP 헤더 세트가 들어간다. 여기에는 요청에 대한 설명, 혹은 메시지 본문에 대한 설명이 들어간다.
- 요청에 대한 모든 메타 정보가 전송됐음을 알리는 빈 줄(blank line)이 삽입된다.

- 요청과 관련된 내용(HTML 폼 콘텐츠 등)이 옵션으로 들어가거나, 응답과 관련된 문서(document)가 들어간다.
  본문의 존재 유무 및 크기는 첫 줄과 HTTP 헤더에 명시된다.



> HTTP 메시지의 시작 줄과 HTTP 헤더를 묶어서 요청 헤드라고 부르며, 이와 반대로 HTTP 메시지의 페이로드는 본문이라고 한다.

리퀘스트 메시지는 크게 3가지 요소로 구성돼 있다.

1. **리퀘스트 라인 - 시작줄**

   3가지 요소로 구성 돼 있다.

   - 메서드 : 브라우저의 동작 상태에 따라 이 메서드가 결정된다.
     - 브라우저의 URL 입력창에 URL을 입력
     - 웹 페이지 안의 하이퍼링크 클릭
     - 폼에 데이터를 기입하여 submit 등
   - URI : 파일이나 프로그램의 경로명을 쓰는 것이 보통이다.
   - HTTP 버전

2. **헤더**

   부가적인 자세한 정보를 제공하는 역할을 한다. 다양한 종류의 요청헤더가 있는데, 이들은 몇몇 그룹으로 나눠진다.

   - General 헤더 : 리퀘스트 메시지와 리스폰스 메시지 둘 다 사용되는 헤더

     ex) `Connection`, `Via`, `Cache-Control`
   - Request 헤더 : 리퀘스트 메시지에 사용되는 헤더

     ex) `Accept`, `Authorization`, `User-Agent`
   - Entity 헤더 : 요청 본문에 적용되는 헤더. 요청 본문이 없는 경우 entity 헤더는 전송되지 않는다.

     ex) `Content-Length`

3. **본문**

   메시지 헤더 뒤 공백 행을 넣고 그 뒤에 이어지는 부분이다.

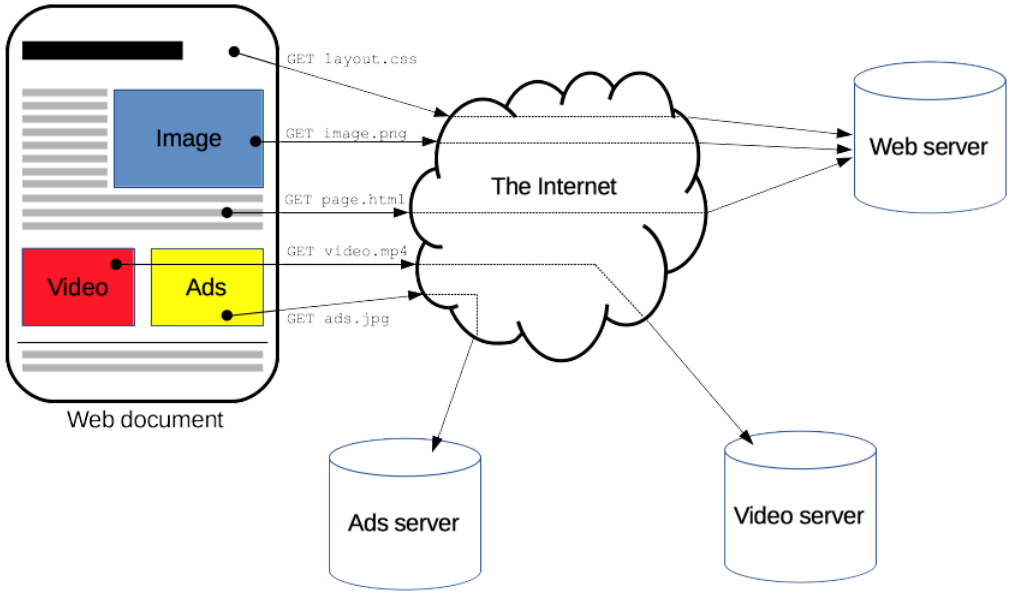
   메시지의 실제 내용이 되며 일반적으로 `Get` `Post` `Delete` `Options` 처럼 리소스를 가져오는 요청은 보통 본문이 필요없다.

6. 리퀘스트 메시지를 보내면 응답이 되돌아온다.

> 응답 메시지의 포맷도 기본적인 개념은 리퀘스트 메시지와 같다.
> 단 첫 행의 리퀘스트의 실행 결과를 나타내는 Status code와 응답문구를 갖고 있다.

code	description
1xx	처리의 경과 상황 등을 통지
2xx	정상 종료
3xx	무언가 다른 조치가 필요함을 나타냄
4xx	클라이언트측의 오류
5xx	서버측의 오류

한 개의 리퀘스트에 대해 한개의 응답만 돌려 보낸다.



> 위 처럼 한 페이지 내의 여러 컨텐츠를 포함한 경우 각각 요청을 보내야만 한다.

html 내의 이지 태그를 포함한 경우

1. '/sample1.html' 리퀘스트 메시지

```
GET /sample.htm HTTP/1.1
Accept: */*
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.lab.glasscom.com
Connection: Keep-Alive
```

2. '/sample1.html' 의 리스폰스 메시지

```
HTTP/1.1 200 OK
Date: Wed, 7 Oct 2020 13:00:00 GMT
Server: Apache
Last-Modified: Wed, 30 Sep 2020 13:00:00 GMT
ETag: "5a9da-27903c726b61"
Accept-Ranges: bytes
Content-Length: 632
Connection: close
Content-Type: text/html

<html>
<head>
<meta http-equiv-"Content-Type" content="text/html"; charset=utf-8">
<title> 인터넷 탐험 여행</title>
</head>

<body>
<img src="gazou.jpg"/>
</body>
</html>
```

3. '/gazou.jpg'에 해당하는 리퀘스트 메시지

```
GET /gazou.jpg HTTP/1.1
Accept: */*
Referer: http://www.lab.cyber.co.kr/sample1.htm
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.lab.cyber.co.kr
Connection: Keep-Alive
```

4. '/gazou.jpg' 리스폰스 메시지

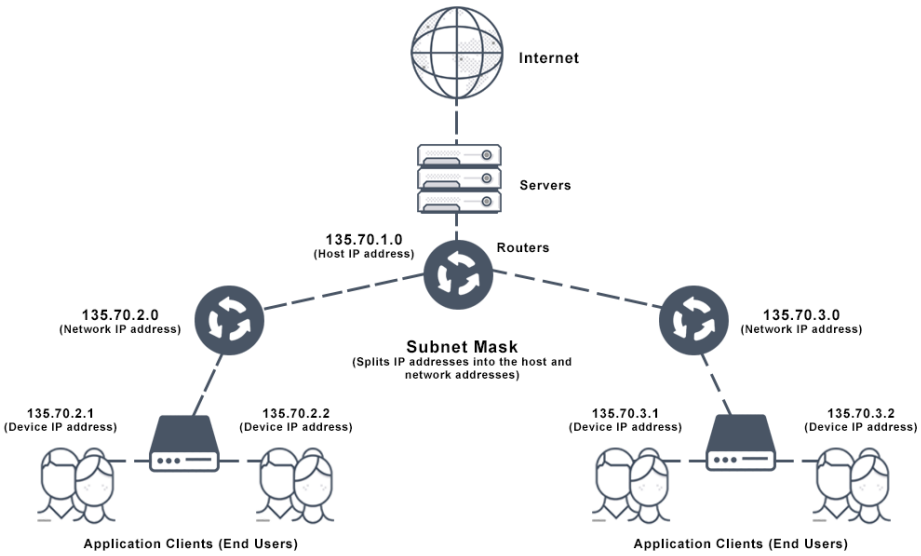
```
HTTP/1.1 200 OK
Date: Wed, 7 Oct 2020 13:00:00 GMT
Server: Apache
Last-Modified: Wed, 30 Sep 2020 13:00:00 GMT
ETag: "5a9da-1913-3aefa236"
Accept-Ranges: bytes
Content-Length: 6419
Connection: close
Content-Type: image/jpeg

이미지 바이너리 데이터
```

웹 서버의 IP 주소를 DNS 서버에 조회한다.

1. IP 주소의 기본

> 브라우저는 자체적으로 메시지를 네트워크에 송출하는 기능은 없고, OS에 의뢰하여 웹 서버에 송신하게 된다.
> 여기서 OS에 송신을 의뢰할 때 도메인 명으로 하는게 아닌 127.0.0.5 같은 IP 주소로 상대를 지정해야 한다.
> 따라서 IP 주소를 얻는 과정이 필요로 한다.

- **허브**를 중심으로 몇 대의 컴퓨터가 모여 **서브넷**을 이루고, 서브넷들은 라우터를 통해 **네트워크**를 이루게 된다.
- 서브넷에 할당된 주소를 **네트워크 번호**, 각 컴퓨터에 할당된 주소를 **호스트 번호**라고 한다.
  그리고 이 두 주소를 합쳐서 **IP 주소**라고 한다



송신측 → 허브 → 라우터 → 라우터 → 수신측

- 실제 IP 주소는 32비트의 디지털 데이터, 8비트씩 점으로 구분하여 10진수로 표기한다.
- IP 주소는 네트워크 번호와 호스트 번호 두 가지를 합쳐서 한다는 것만 결정되고 내용은 결정돼 있지 않다.
  - 그 내용을 **넷마스크**라고 한다.
- 넷마스크는 1인 부분은 네트워크 번호로 나타내고, 0인 부분은 호스트 번호를 나타낸다.
  IP 주소와 같이 8비트씩 구분하여 표기할 수도 있는데 너무 길어져서 한 부분의 비트 수를 10진수로 나타내고 IP 주소의 오른쪽에 병기할 수도 있다.

```
// a) ip 주소 본체의 표기 방법
10. 11. 12. 13

// b) ip 주소 본체와 같은 방법으로 네트워크를 표기하는 방법
// ip 주소 본체 / 넷마스크
10.11.12.13/255.255.255.0

// c) 네트워크 번호의 비트 수를 넷마스크를 표기하는 방법
10.11.12.13/24 // 24 -> 비트 수로 표현

// d) 서브넷을 나타내는 주소
10.11.12.0/24 // 호스트번호 부분의 비트가 모두 0인 것은 각 컴퓨터가 아니라 서브넷 자체를 나타냄
```

```
// e) 서브넷의 브로드캐스트를 나타내는 주소
10.11.12.255/24 // 255 -> 호스트 번호 부분의 비트가 모두 1인 것은 서브넷 전체에 대한 브로드 캐스트를 나타냄
```

2. 도메인명과 IP 주소를 구분하여 사용하는 이유

❓ ip주소 대신 도메인명을 사용하면 안되나?

> ip 주소는 4바이트분의 수치만 취급하면 된다.
> 하지만 도메인명은 수십 byte ~ 225bytes를 차지한다 → 라우터에 과부하가 걸린다.

- 기억하기 쉽게 하기 위해 도메인명을 사용한다.

  → 따라서 사람은 도메인명, 라우터는 IP주소를 사용한다.

3. Socket 라이브러리가 IP 주소를 찾는 기능을 제공한다.

IP 주소를 조사하는 방법

- 가장 가까운 DNS 서버에 특정 도메인 서버의 IP 주소를 질문하는 것이다.
- 그러면 DNS서버가 그 도메인 서버의 주소는 ~~ 입니다. 라는 식으로 알려준다.

> 🌟 **DNS란**
> Domain Name System의 약자로 서버명과 IP 주소를 대응시키기 위해 DNS를 가장 많이 사용하지만
> DNS는 메일 주소와 서버를 대응 시키는 것 등 다양한 정보를 이름에 대응해서 등록할 수 있습니다.

❓ 그럼 DNS 서버를 어떻게 조회하는 것일까?

> DNS 서버에 조회 메세지를 보내고, 거기에서 반송되는 응답 메시지를 받는다는 것이다.
> 여기에서 IP 주소를 조사하는 것을 **네임 리졸루션**(name resolution, 이름 확인) 이라고 한다.
> 이러한 리졸루션을 실행하는 것은 **리졸버**이다.

DNS 리졸버

- DNS 클라이언트에 해당한다.
- 네임 리졸루션을 실행하는 주체이다.
- Socket 라이브러리 내의 하나의 프로그램이다.

> 🌟 **Socket 라이브러리**
> 네트워크 기능 호출을 위한 프로그램 라이브러리이다.
> OS에 포함돼 있는 네트워크의 기능을 애플리케이션에서 호출하기 위한 부품을 모아놓은 것이고,
> 리졸버는 그 속에 들어있는 프로그램 부품의 하나이다.

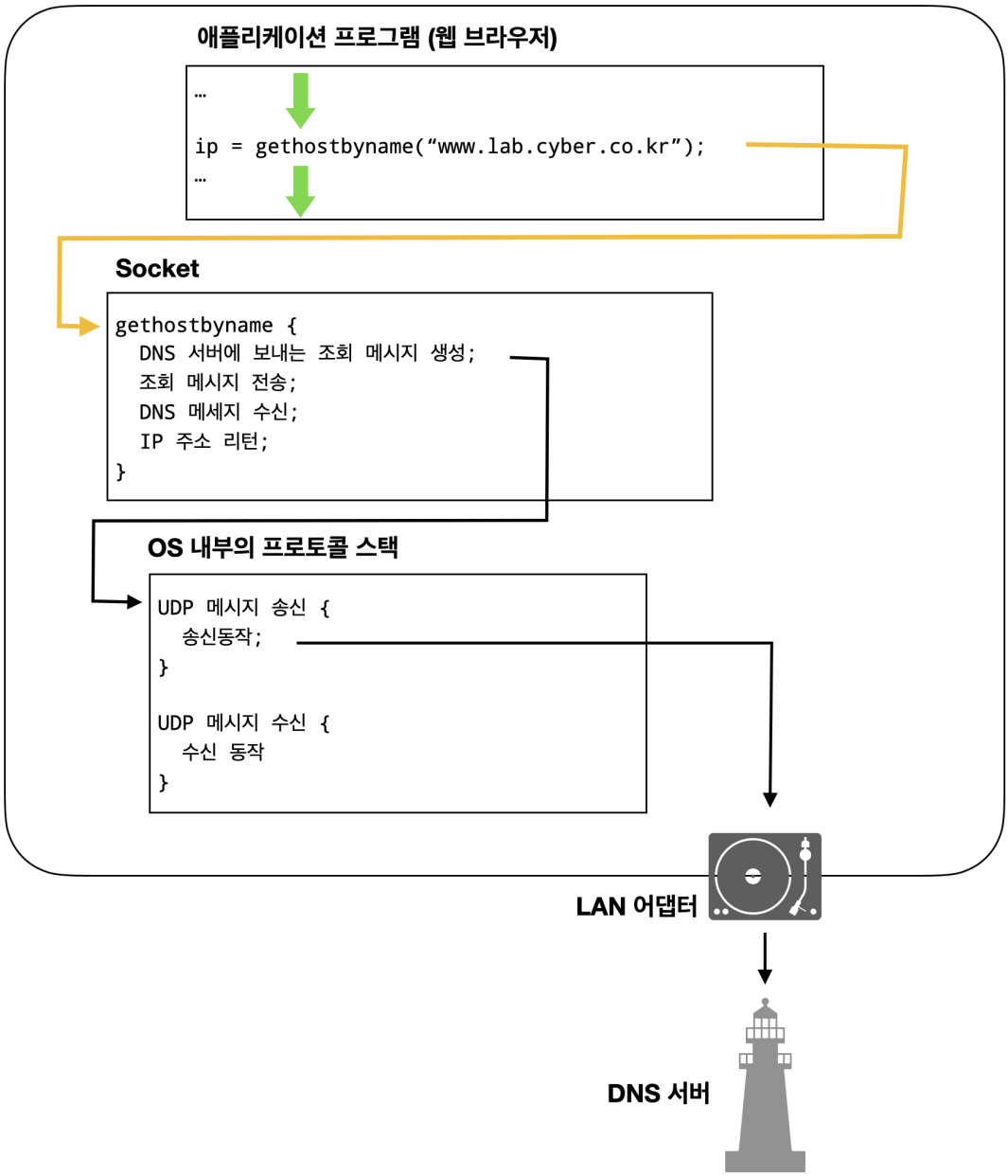
4. 리졸버를 이용하여 DNS 서버를 조회한다.

> 리졸버는 애플리케이션에서 Socket 라이브러리를 통해 간단히 호출이 가능하다.

```
class network_application_name(parameter) {
  ...
  // gethostbyname : 리졸버 프로그램명
  // ip : 메모리 영역
  ip = gethostbyname("www.lab.cyber.co.kr");
  ...
}
```

브라우저는 이런 형태로 리졸버를 이용해서 IP 주소를 얻고 웹 서버에 메시지를 보낸다.

5. 리졸버 내부의 작동



- 브라우저가 리졸버를 호출하게되면 **제어가 리졸버의 내부로 넘어간다.**
- 리졸버는 DNS에 문의하기 위한 메시지를 만든다.
- 리졸버는 직접 메시지를 송신하는 기능을 가지고 있는 것이 아니다.
- OS의 프로콜 스택을 호출하여 실행을 의뢰하게된다.
- 프로토콜 스택이 LAN 어댑터를 통해 DNS 서버로 메시지를 송신한다.
- DNS 서버가 IP주소를 조회해 응답메시지를 보내준다.

❓ **그럼 DNS 서버의 IP주소는 어떻게 아는거지?**

→ **컴퓨터에 미리 설정돼 잇는 값을 가져오는 것이다.**

전 세계의 DNS 서버가 연대한다.

1. DNS 서버의 기본 동작

> 리졸버로부터 DNS 서버가 받는 조회 메시지에는 아래와 같은 내용이 포함돼있다.

- **이름** : 서버나 메일 배송 목적지와 같은 이름

- **클래스** : 인터넷 이외에서의 네트워크 이용까지 검토하여 이것들을 식별하기 위한 정보였다.

  → **현재는 인터넷 이외의 네트워크는 소멸됐다. 즉 IN 값만 사용한다.**

- **타입** : 이름에 어떤 타입(종류)의 정보가 지원되는지를 나타냄

  → **A** : IP 주소 지원, **MX** : 메일 배송 목적지 지원 등등

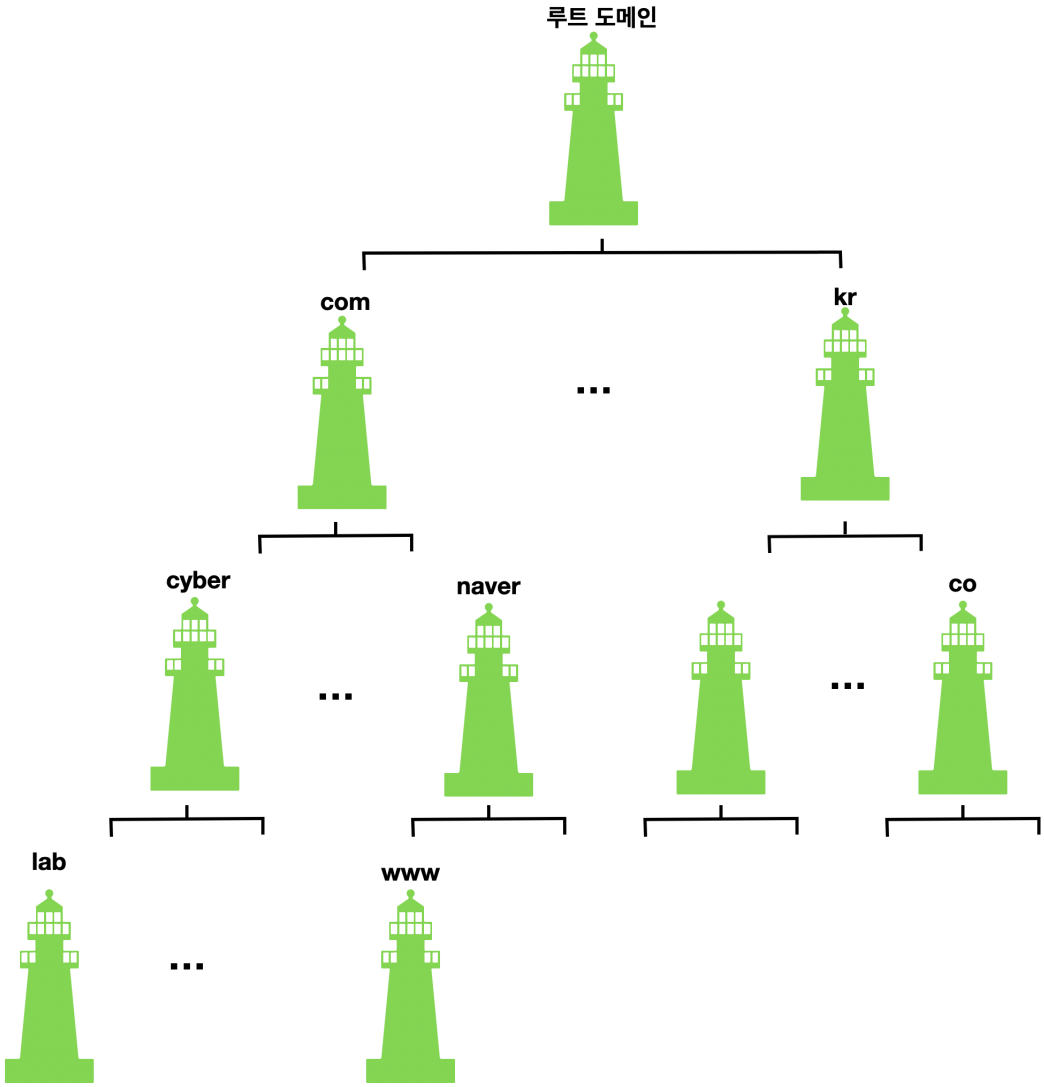
이름	클래스	타입	클라이언트 회답하는 항목
www.lab.cyber.co.kr	IN	A	192.0.2.226
cyber.co.kr	IN	MX	10 mail.cyber.co.kr
mail.cyber.co.kr	IN	A	192.0.2.227

- 이러한 항목들을 하나씩 리소스 레코드라고 부른다.
- 따라서 DNS 서버는 서버에 등록된 도메인명과 IP 주소의 대응표를 조사하여 IP 주소를 회답하게 된다.

2. 담당 DNS 서버를 찾아 IP주소를 가져온다.

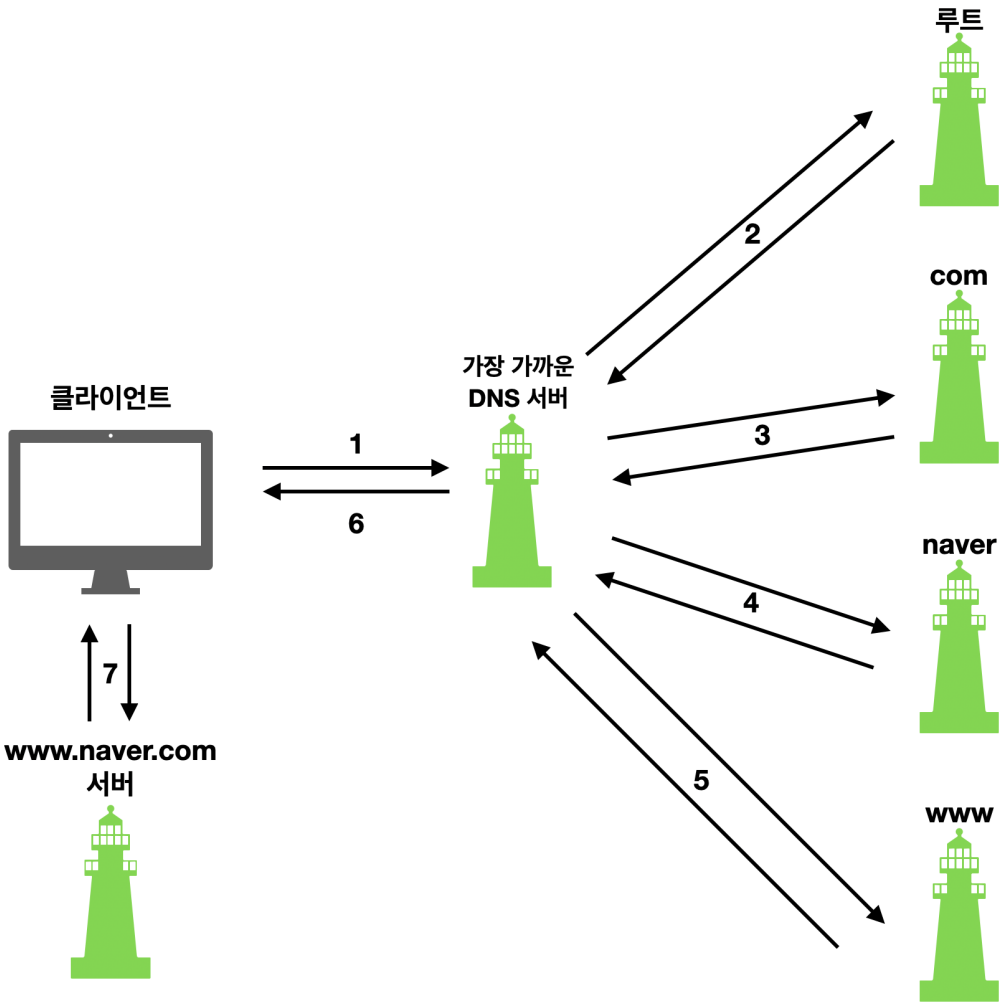
> 조회 메시지를 받은 DNS 서버에 해당 정보가 등록돼 있는 경우 찾아서 회답을 한다.
> 하지만 인터넷에는 우리가 상상할 수 없을 정도의 서버가 존재한다. 전부를 1대의 DNS 서버에 등록하는 것은 불가능하다.
> 따라서 조회 메시지를 받은 DNS 서버에 정보가 등록돼 있지 않은 경우를 살펴보자.

위에 설명한 것처럼 DNS 서버가 수 만대가 있으므로 일일이 모든 DNS 서버를 찾아갈 수 없다.
그러므로 DNS서버는 계층적 구조를 가지고 있다.



`www.naver.com` → com 도메인, naver 도메인, www 도메인 이러한 형태로 생각할 수 있다.

- kr과 com 같은 도메인을 최상위 도메인이라고 부른다.
  이들은 최상위 이지만 최상위 도메인을 연결하는 루트 도메인이 존재한다.



1. PC에 등록돼 있는 가까운 DNS 서버에 조회 메시지 전송
2. 해당 DNS 서버에는 등록돼 있지 않으므로 루트 도메인부터 조회 시작, 루트 도메인에도 등록돼 있지 않아 하위 도메인 com 도메인의 DNS 서버의 IP 주소를 응답
3. com DNS 서버에도 등록되어 있지 않아, 하위 도메인 naver DNS 서버 IP 주소 응답
4. naver DNS 서버에도 등록되어 있지 않아, 하위 도메인 www DNS 서버 IP 주소 응답
5. www DNS 서버에 최종적으로 www.naver.com 에 대한 IP 주소가 등록돼 있어 해당 웹 서버 IP 주소 응답
6. 최초로 조회 메시지를 받은 가장 가까운 DNS 서버는 이 IP 주소를 클라이언트에 응답
7. 웹 서버에 접그

3. DNS 서버는 캐시 기능으로 빠르게 회답할 수 있다.

> DNS 서버는 한 번 조사한 이름은 캐시에 기록할 수 있다.
> 캐시에 기록돼 있으면 그 정보를 응답해 더 빠르게 동작
> 조회한 이름이 존재지 않는다는 것도 캐시 가능

⛔ **주의점**

- 캐시에 저장한 후 등록 정보가 변경되는 경우 캐시정보는 잘못된 정보가 돼버린다.
- 따라서 캐시에 저장할 때 **유효기간을 설정**한다.

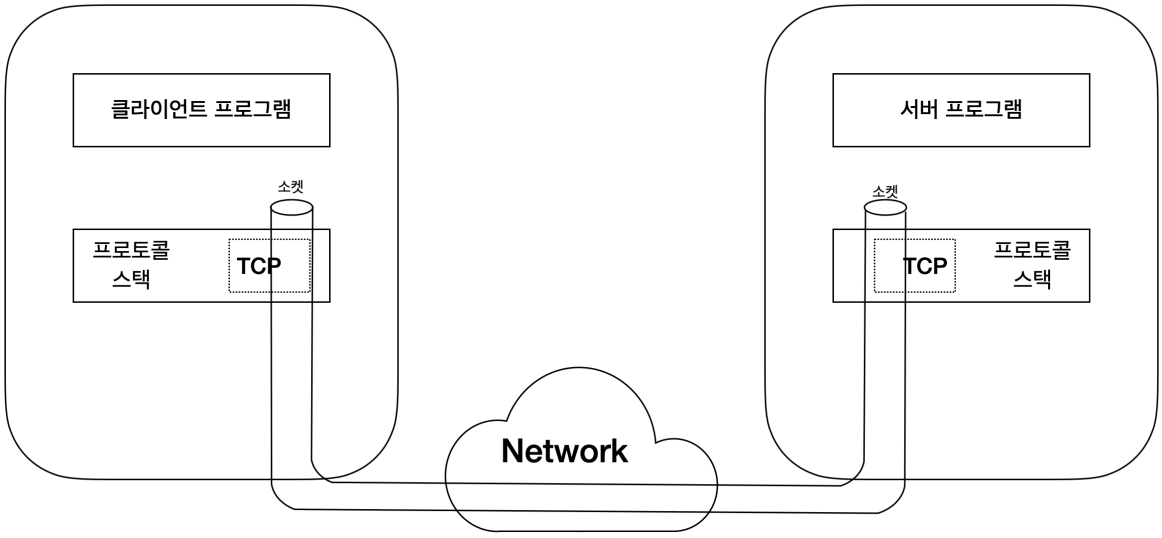
- 응답시 캐시 정보인지 등록처 DNS 서버에서 응답한 것인지 알려줘야 한다.

프로토콜 스택에 메시지 송신을 의뢰한다.

1. 데이터 송 수신 동작의 개요

> IP 주소를 조사했으면 IP 주소의 상태, 웹 서버에 메시지를 송신하도록 OS 내부에 있는 프로토콜 스택에 의뢰한다.

아래 송수신 동작 전체 사진을 보면 데이터 송수신은 기본적으로 데이터 통로를 통해 수행한다.



클라이언트와 서버 사이를 일종의 파이프로 연결하는 작업이 필요하다.

그 양 끝의 출입구를 socket(소켓) 이라고 부른다. 따라서 이 소켓을 만들고 연결을 하게 된다.

1. **소켓을 만든다.**
2. **서버측의 소켓에 파이프를 연결한다.**
3. **데이터를 송수신한다.**
4. **파이프를 분리하고 소켓을 말소한다.**

- Socket 라이브러리의 프로그램 부품을 이용해서, 위 단계들을 수행하게 되는데 실제로 이 프로그램 부품들의 메시지 내용을 그대로 포로토콜 스택에 절ㄴ달하는 중계 역할을 수행하고 실질적인 작업은 하지 않는다.
- 프로토콜 스택이 어떻게 파이프를 연결하고 데이터를 송수시하는지는 2장에 설명이 된다.

2. 소켓을 만든다.

> 소켓 라이브러리 안의 소켓이라는 프로그램 부품만 호출하면 된다.
> 소켓을 호출하면 그 내부에 제어가 넘어가 소켓을 만드는 동작을 실행하고 그 후 애플리케이션으로 제어가 돌아온다.

소켓이 생성되면 디스크립터라는 것이 생성되고 이 디스크립터는 애플리케이션에서 메모리에 기록을 해둔다.

❓ **디스크립터란?**

> 소켓을 식별해주는 식별자 역할을 한다.
> 컴퓨터 내부에서는 복수의 데이터 송수신 동작이 진행될 수 있으므로, 소켓들이 여러개 생기고 이 소켓들을 구별할 필요가 있기 때문이다. (브라우저에서 N개의 창을 띄워 다른 페이지에 접근할 때)

3. 서버측의 소켓에 파이프를 연결한다.

> 소켓을 서버측의 소켓과 접속하도록 프로토콜 스택에 의뢰한다.
> 이 때 애플리케이션의 소켓 라이브러리의 connect 라는 프로그램 부품을 호출하여 이 동작을 실해한다.

connect 프로그램 실행할 때 필요한 3가지

- 디스크립터 → 소켓에서 생성할 때 만들어진 디스크립터
- 서버 IP주소
- 포트 번호

❓ **포트 번호란?**

> IP 주소의 경우 네트워크에 존재하는 각 컴퓨터를 식별할 수 있는 값이고, 포트 번호는 그 컴퓨터의 소켓을 식별하는 값

- **디스크립터와의 차이**
  - 디스크립터는 소켓을 만드도록 의뢰한 애플리케이션을 건네주는 **값**일 뿐, 접속 상대에 건네주는 것이 아니다.
  - 즉 접속 상대측은 모르는 상태이다.
  - 서버측 소켓의 디스크립터는 클라이언트에서 알 수 없다.
- **서버측의 포트번호**
  - 서버측의 포트 번호는 애플리케이션의 종류에 따라 미리 결정된 값을 사용한다는 규칙이 있다.

    → **웹 : 80 메일 25 등등**
- **웹 서버측에서의 클라이언트 포트번호 식별방법**
  - 적당한 포트번호 값을 할당하고, 접속 동작을 실행할 때 서버측에 알려준다.

4. 데이터를 송수신한다.

> 소켓이 준비가 되면 데이터를 보낼 준비가 완료된 것이다.
> 애플리케이션은 송신 데이터(HTTP 리퀘스트 메시지)를 메모리에 준비한다.

송신과정

- 송신을 위해 라이브러리의 write 라는 프로그램 부품을 사용하여 프로콜 스택에 송신동작 의뢰
- 이 때는 디스크립터와 송신 데이터를 저장한다. (소켓은 이미 상대와 연결되어 있으므로 디스크립터만 사용)
- 송신 데이터는 네트워크를 통해 서버에 도착한다.
- 서버는 수신 데이터를 처리하고 응답 메시지를 보낸다.

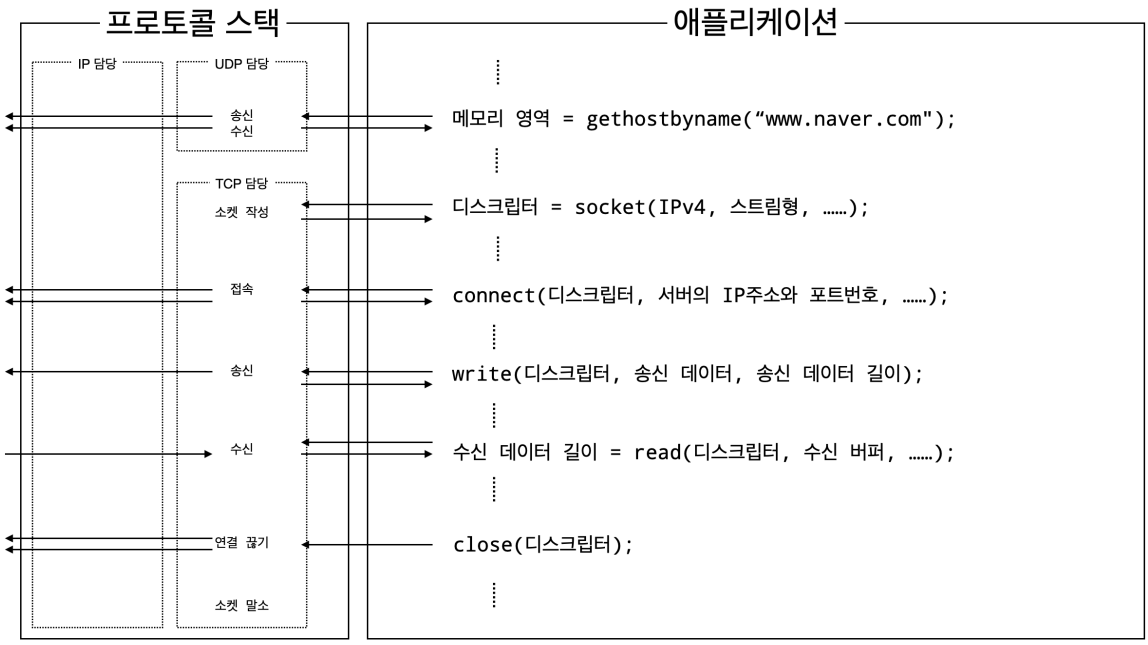
수신과정

- 소켓 라이브러리의 read 프로그램 부품을 통해 프로토콜 스택에 수신 동작을 의뢰
- 수신 응답 메시지를 수신 버퍼 라는 메모리 영역에 저장 → 애플리케이션 내부에 위치

5. 연결 끊기

> 브라우저가 데이터 수신을 완료하면 송수신 동작이 끝난다.
> 소켓 라이브러리의 close 라는 프로그램 부품을 호출하여 프로토콜 스택에 의뢰

- 웹 서버측은 응답 메시지의 송신을 완료했을 때 먼저 close를 호출하여 연결을 끊는다.
- 이것이 클라이언트측에 전달되고 클라이언트 소켓은 연결 끊기 단계로 들어간다.


프로토콜 스택
애플리케이션
IP 담당
UDP 담당
송신
수신
TCP 담당
소켓 작성
접속
송신
수신
연결 끊기
소켓 말소
메모리 영역 = gethostbyname(“www.naver.com");
디스크립터 = socket(IPv4, 스트림형, ……);
connect(디스크립터, 서버의 IP주소와 포트번호, ……);
write(디스크립터, 송신 데이터, 송신 데이터 길이);
수신 데이터 길이 = read(디스크립터, 수신 버퍼, ……);
close(디스크립터);