



Flask & OracleDB

강의 필수 지식

- 파이썬 기초문법
- 기타:HTML,CSS,javascript

필요 소프트웨어

- Anaconda:파이썬종합개발환경
- www.python.org
- `pip install flask`
- Pycharm(community edu):에디터
- Visual studio code

	1주차				2주차				3주차				4주차				5주차			
	과정내용	시간	교수법	장비사용유무	과정내용	시간	교수법	장비사용유무	과정내용	시간	교수법	장비사용유무	과정내용	시간	교수법	장비사용유무	과정내용	시간	교수법	장비사용유무
1교시	웹의 역사와 구조	1	이론 및 실습	X	jinja 제어와 반복 처리	1	이론 및 실습	X	Python-Oracle 연결	1	이론 및 실습	X	Ajax 개념	1	이론 및 실습	X	Plotly	1	이론 및 실습	X
2교시	웹프로토콜의 구조	1	이론 및 실습	X	jinja 템플릿 상속	1	이론 및 실습	X	Oracle SQL	1	이론 및 실습	X	Ajax DB연동	1	이론 및 실습	X	Plotly	1	이론 및 실습	X
3교시	Flask 개발 환경 구축	1	이론 및 실습	X	jinja 템플릿 재사용	1	이론 및 실습	X	Oracle SQL	1	이론 및 실습	X	Ajax DB연동	1	이론 및 실습	X	Plotly	1	이론 및 실습	X
4교시	라우팅	1	이론 및 실습	X	jinja 템플릿 필터 작성과 사용	1	이론 및 실습	X	Oracle SQL	1	이론 및 실습	X	Flask-SQLAlchemy	1	이론 및 실습	X	Plotly	1	이론 및 실습	X
점심시간	점심식사																			
5교시	요청과 응답	1	이론 및 실습	X	웹 폼 유효성 검사	1	이론 및 실습	X	DB Pool	1	이론 및 실습	X	Flask-SQLAlchemy	1	이론 및 실습	X	종합평가	1	이론 및 실습	X
6교시	html 기초	1	이론 및 실습	X	모듈별 협업하기(MVC 파일분할)	1	이론 및 실습	X	Oracle SQL	1	이론 및 실습	X	Flask-SQLAlchemy	1	이론 및 실습	X	종합평가	1	이론 및 실습	X
7교시	html 기초	1	이론 및 실습	X	파일 업로드 다루기 캐싱	1	이론 및 실습	X	Oracle SQL	1	이론 및 실습	X	Flask-SQLAlchemy	1	이론 및 실습	X	종합평가	1	이론 및 실습	X
8교시	연습과제	1	이론 및 실습	X	연습과제	1	이론 및 실습	X	연습과제	1	이론 및 실습	X	연습과제	1	이론 및 실습	X	종합평가	1	이론 및 실습	X
Total Time		8				8				8				8				8		

- 50~정각 10~15분 휴식
- 식사: 11:50~13:00
- 16:00 과제 및 풀이

개발환경

- anaconda:파이썬 종합개발환경
- pycharm(edu, community), vscode

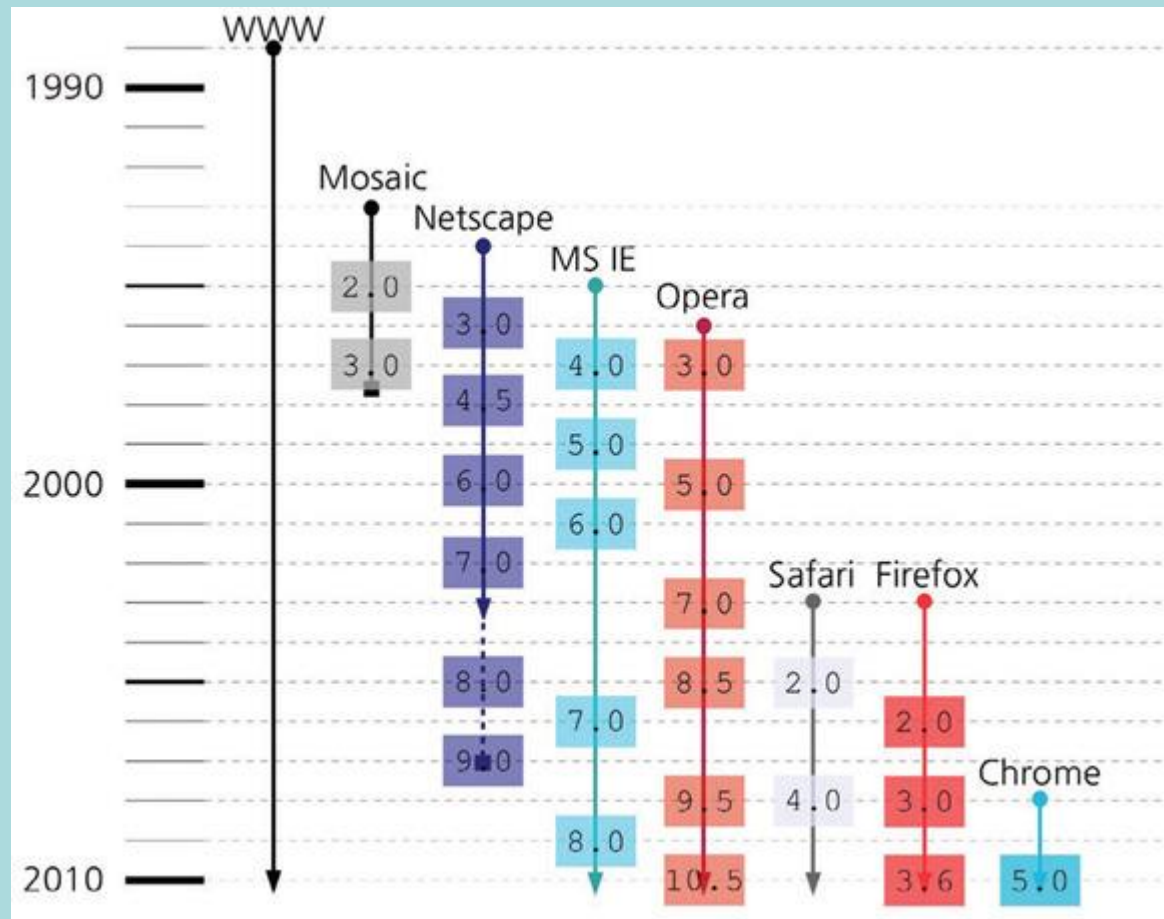
Flask 란

- flask는 python 기반의 Web Framework다.
- python 기반의 다른 web framework 는 Django, FastAPI 등이 있음
- 마이크로 웹 프레임워크 Flask(라이트한 개발목적)

웹의 역사및 구조



웹브라우저 발달역사



<http://evolutionofweb.appspot.com/>

웹브라우저 역사

10

- 최초의 웹브라우저는 **1990년 12월 25일** 영국의 소프트웨어 공학자인 팀 버너스리(**Tim Berners-Lee**)가 정보 공유 시스템용으로 개발한 **월드와이드웹(WorldWideWeb)**이었다. 처음에는 웹브라우저의 이름과 정보 공유 방법을 모두 월드와이드웹이라고 했으나, 후에 두 기능의 혼동을 방지하기 위하여 웹브라우저의 이름은 **넥서스(Nexus)**로, 정보 공유 방법은 **월드 와이드 웹(World Wide Web)** 또는 간단히 **웹(web)**으로 구분하였다.
- 넥서스는 단순한 웹브라우저로 그 기능이 텍스트를 처리하는 정도였다
- **모자이크(Mosaic)**은 미국 일리노이 대학 슈퍼컴퓨터 응용 연구소(**National Center for Supercomputing Applications: NCSC**)의 마크 앤드리슨에 의하여 개발되었다.
- 윈도 환경에서 사용할 수 있도록 멀티미디어 그래픽 사용자 인터페이스를 채용하였다.
- 마크 앤드리슨이 회사를 설립하고 모자익을 개량하여 **네스케이프 내비게이터(Netscape Navigator)**를 개발
- **1995년에 MS**사는 스파이글래스 모자익을 인터넷 익스플로러(**Internet Explorer: IE**)라는 이름으로 출시하였다. 이것이 바로 윈도**95**에 끼워 제공하던 **IE**이다.
- 윈도**98**과 함께 배포될 때에는 대폭적으로 성능이 개선되었다. 이 시점 이후로 **IE**는 네스케이프를 밀어내고 웹브라우저의 절대 강자로 군림하게 되었다.
- **1996년에** 개발된 **오페라(Opera)**는 작고 빠른 웹브라우저라는 모토 하에 꾸준히 명맥을 이어왔다.
- **2003년** 네스케이프의 몰락과 함께 모질라(**Mozilla**) 재단은 독자적으로 파이어폭스(**FireFox**)를 개발하여 지금에 이르고 있다.
- **2003년** 애플사는 자체적으로 사파리(**Safari**)를 개발하여 매킨토시에서 아이폰에 이르는 모든 제품의 웹브라우저로 제공해오고 있다.
- **2008년에** 들어와 구글 크롬(**Chrome**)이 출현하였다. 크롬의 가장 큰 특징은 페이지 탭별로 멀티태스킹이 가능하고 빠르다는 점이다. 이에 힘입어 크롬은 지금까지 꾸준히 점유율을 높여오고 있다

Html 역사

11

연월	사양	설명
1993년 6월	HTML 1.0	IETF Internet Draft
1995년 11월	HTML 2.0	RFC 1866
1997년 1월	HTML 3.2	W3C권고
1997년 12월	HTML 4.0	W3C권고
1998년 2월	XML 1.0	W3C권고
1998년 10월	DOM Level 1	W3C권고
1999년 12월	HTML 4.01	W3C권고
2000년 1월	XHTML 1.0	W3C권고
2000년 11월	DOM Level 1	W3C권고

HTML 의 발달사

12

넷스케이프 → W3C → WHATWG →

W3C: 팀버너스 리가 1994년 설립한 비영리 단체로
HTML4.01을 권고한 것을 1999년 일로 10년간 HTML 최
신버전으로 있었음

WHATWG(2004년) :모질라 재단과 오페라 소프트웨어
(모든 웹사이트에 플러그인(액티브엑스) 이 들어가면서 웹
사이트가 점점 무거워짐
익스플로러를 제외한 독자적으로 새로운 웹 표준기관을 설
립함

2004년 6월 HTML5 표준을 제정하는 WHATWG가 설립됨



HTML 역사

Rough Timeline of Web Technologies

1991 HTML
 1994 HTML 2
 1996 CSS 1 + JavaScript
 1997 HTML 4
 1998 CSS 2
 2000 XHTML 1
 2002 Tableless Web Design
 2005 AJAX
 2009 HTML 5

Web 1.0	HTML & CSS	마크업과 정보
Web 2.0	AJAX & Open API	애플리케이션 플랫폼.
HTML5	새로운 요소와 API	독립적인 웹 애플리케이션

HTML5 = 추가된 HTML 태그 + CSS3 +
 강력한 javascript API



HTML5 배경

- 웹이 정적 문서에서 동적 프로그램으로의 변화
- XHTML2.0의 기존 웹과의 비호환성과, 엄격한 XML 규칙의 적용으로 인한 제어의 어려움

WHATWG (Web Hypertext Application Technology Working Group)

모질라재단, 애플, 오페라 소프트웨어의 세 회사가 모여 현재의 HTML4.01 기술과 호환되면서 웹의 기능과 표현 범위를 확장하고자 하는 기술 표준을 작성



HTML5역사



VS



WHATWG

1. 차세대 웹표준으로 XHTML 추진
2. HTML/XHTML과의 호환성을 고려하지 않은 XHTML2.0

1. XHTML보다 HTML 업데이트
2. W3C에 제안을 거절당함
3. Apple, Mozilla, Opera 주도로 워킹그룹을 2004년 발족



HTML5역사



WHATWG

1. 2007년 W3C는 WG라는 워킹 그룹을 발족
2. WHATWG와 공동으로 HTML5의 제정에 함께하기로 함.
3. 2008년 1월 22일 W3C가 HTML5의 초안을 공개함.
4. XHTML 추진에 주력하던 W3C가 획기적으로 방침을 전환한 역사적인 사건
5. 2009년 10월 WHATWG가 HTML5 표준화 작업을 W3C로 넘김
6. 2009년 7월에 XHTML2의 사양 제정이 중지됨



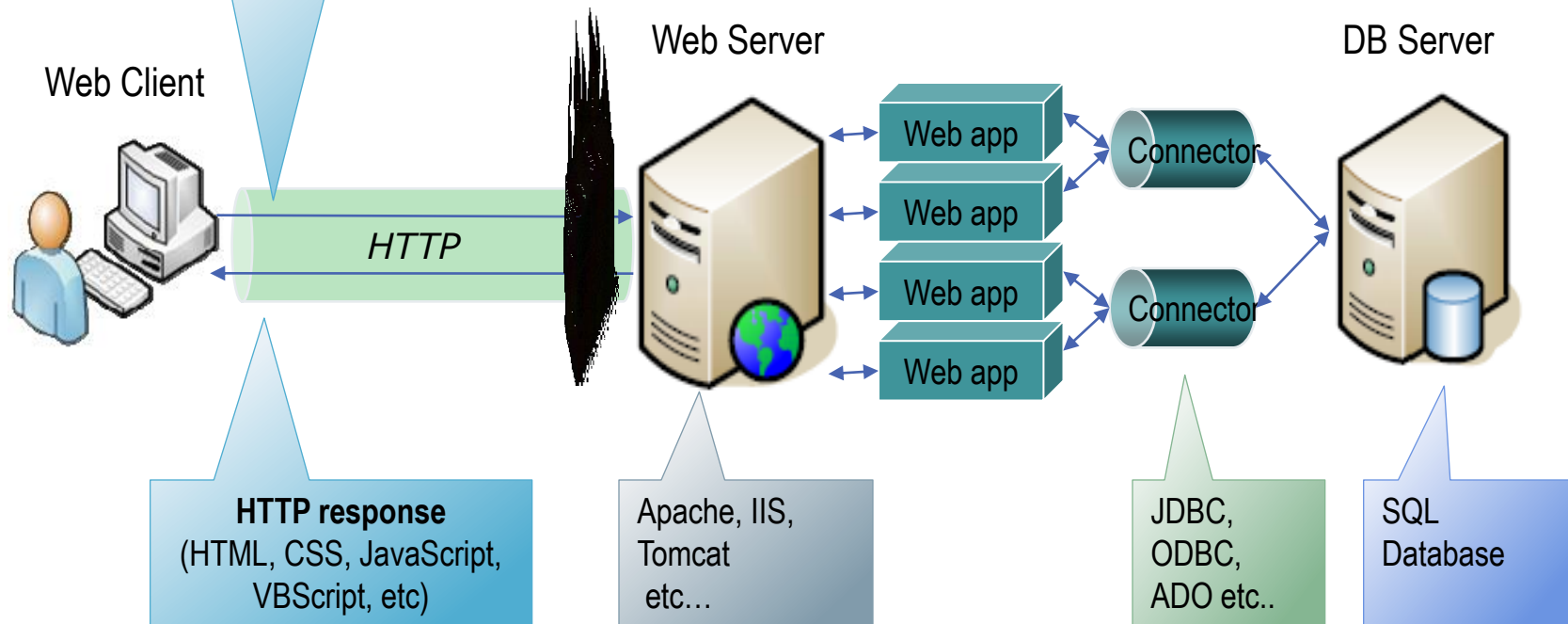
HTML5역사

연월	사왕	설명
2011년 5월	HTML5 최종 초안 (Last Call Working Draft)	
2012년 Q2	HTML5 후보 표준안 (Candidate Recommendation)	2개 이상의 브라우저에서 테스트 완료
2014년 Q1	HTML5 제안 표준안 (Proposed Recommendation)	브라우저 업체와 파드백 반영
2014년 Q2	HTML5 최종 표준안 (Recommendation)	

1.1 웹어플리케이션 아키텍처

18

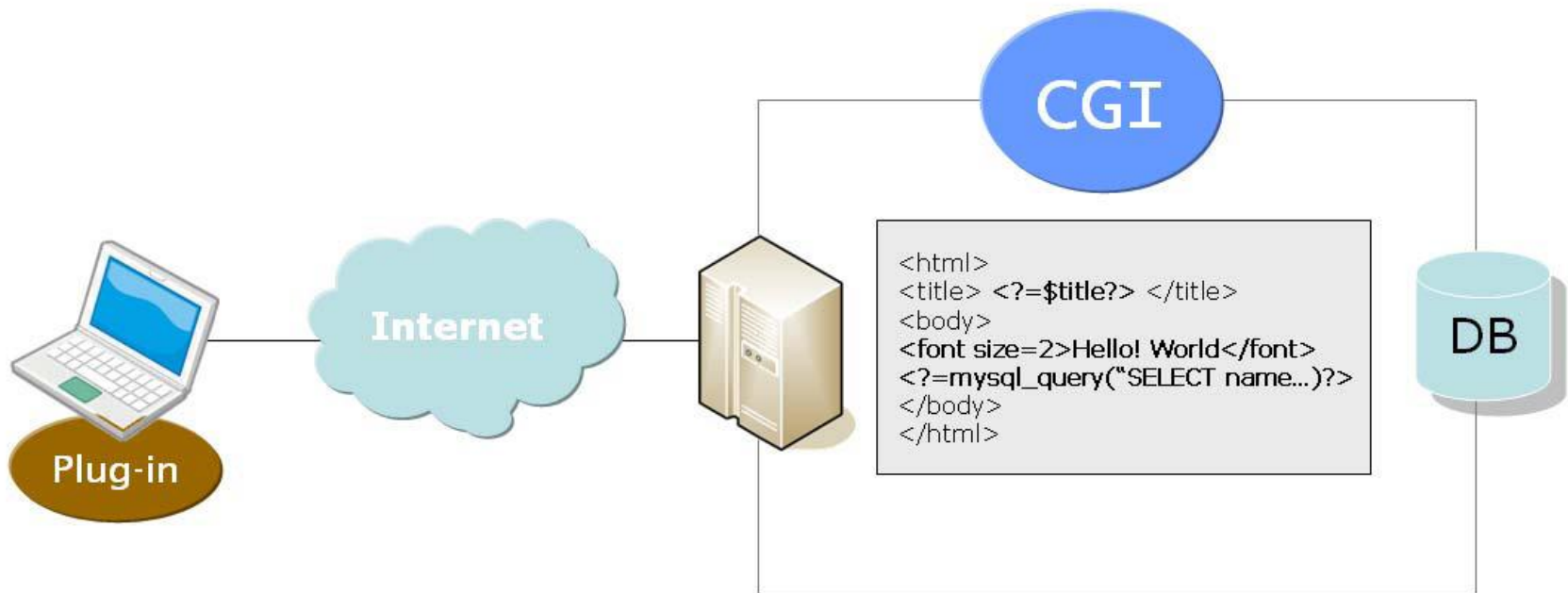
대부분이 2Tier로 분리되어 있을지만 3Tier가 존재할 수도 있습니다. 물론 소규모의 경우 1Tier에서 웹서버와 DB서버의 서비스를 제공할 수 있습니다. 물론 안정성을 위해 웹서버나 DB서버의 이중화가 이루어질 수 있습니다.



1.1.1 웹 문서 시대(1990년대)

19

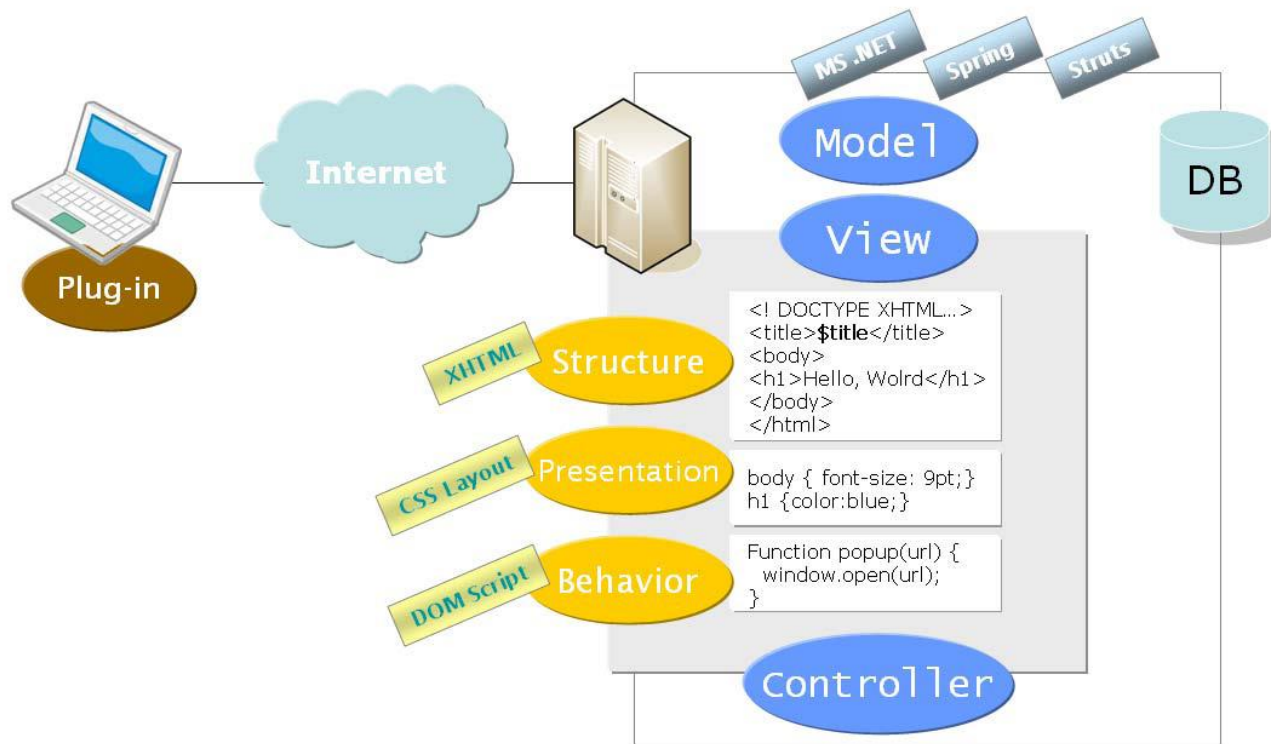
- 웹 서버와 웹 브라우저간 정적 HTML문서를 주로 보내거나 CGI(Common Gateway Interface)를 이용하여 개발하는 경우, 마크업과 프로그램 코드가 섞여있는 개발 방식을 사용했다.
- 이 때는 개발 직군간의 업무 분담이 전혀 이루어지지 않는 상태였습니다.



1.1.2 웹 표준 시대(2000년대 초반)

20

- 2000년대 초반으로 오면서 백엔드 개발에서는 이른바 **MVC 모델**이라는 기법을 통해 **데이터 모델**과 **템플릿** 그리고 **비즈니스 로직이 분리된 코드**를 통해 개발 생산성을 높이게 됩니다.
- **프론트엔드**에서도 웹 표준 개발 방법론을 통해 **구조(HTML)**, **표현(CSS)**, **동작(DOM Script)**를 분리하고 **CSS 레이아웃**과 **W3C 기반 DOM**을 통한 웹 개발 방식을 많이 이용하게 됩니다.



21

-
- The diagram illustrates the MVC architecture for a web application. On the left, a laptop represents the client, connected to the Internet cloud. The client side is divided into three yellow ovals: Structure, Presentation, and Behavior. A brown oval labeled 'Plug-in' is also shown. The server side is represented by a server rack icon, connected to the Internet cloud. The server side is divided into three blue ovals: Model, View, and Controller. The Model contains a JSON object:

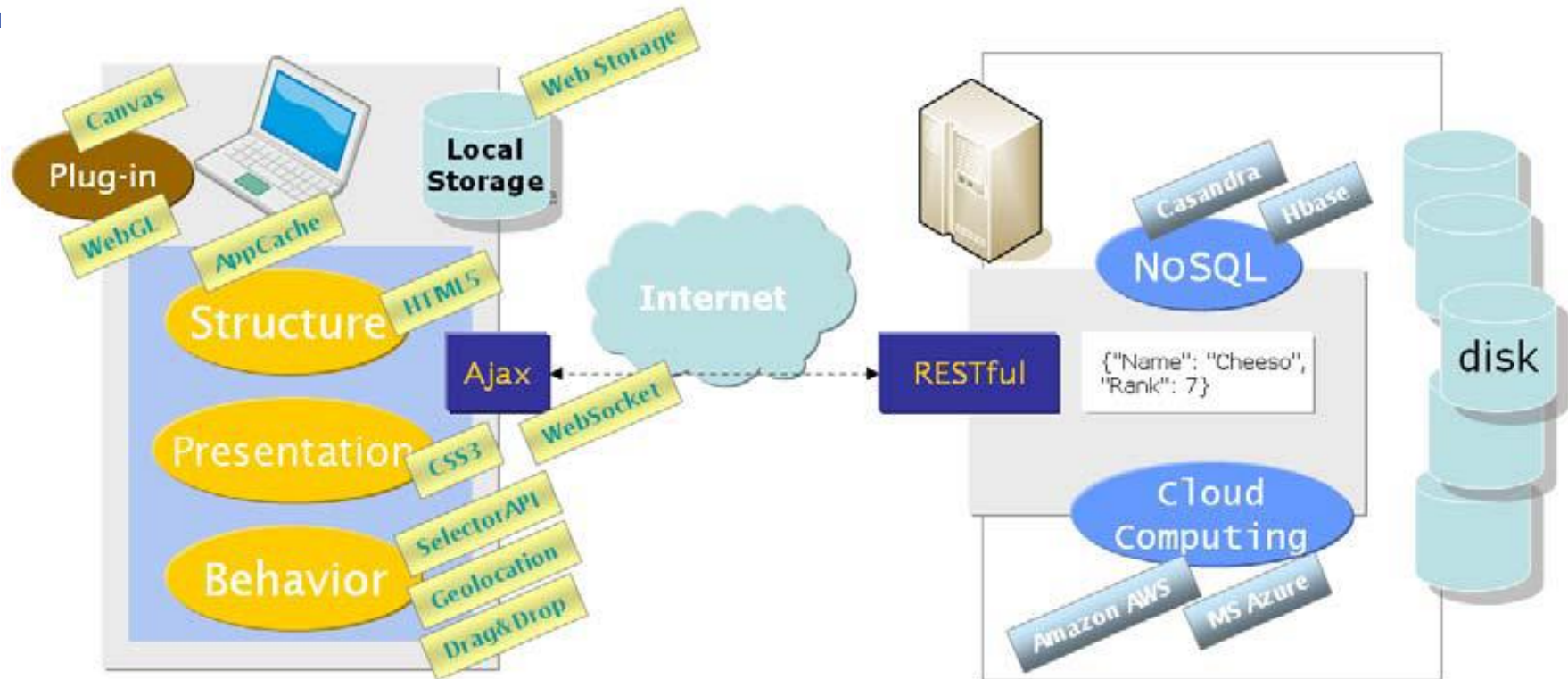
```
{ "Name": "Cheeso", "Rank": 7 }
```

. The View is connected to the Model. The Controller is connected to the View and the Internet cloud. The Controller is also connected to a database (DB) icon. The Controller is labeled 'RubyOnRails'. The Ajax and OpenAPI components are shown as blue boxes with double-headed arrows between them, indicating communication. The Ajax component is connected to the Internet cloud and the client side. The OpenAPI component is connected to the Internet cloud and the server side.

1.1.4 HTML5시대 (2010년대 초반)

22

- HTML5가 가져올 가장 큰 변화는 서버와 독립적인 웹 애플리케이션의 개발이 가능하다는 것입니다.
- 특히, 모바일 환경에서 오프라인 기능과 로컬 데이터베이스의 지원은 웹 서버와 독립할 수 있는 여건을 만들어 줍니다.

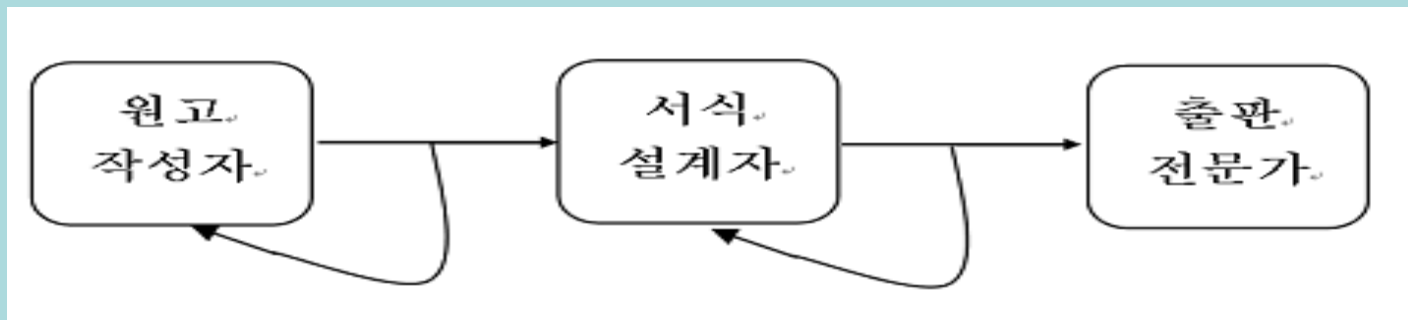


마크업(Markup)이란 ?

□ 전통적인 마크업

■ 전통적인 출판 단계

- 원고 작성자
 - 원고 작성
- 서식 설계사
 - 원고의 출판 형태를 정의(서체 종류 및 크기 등)
☞ 이러한 수작업을 마크업이라 한다
- 출판 전문가
 - 마크업된 지시대로 원고 데이터를 표현



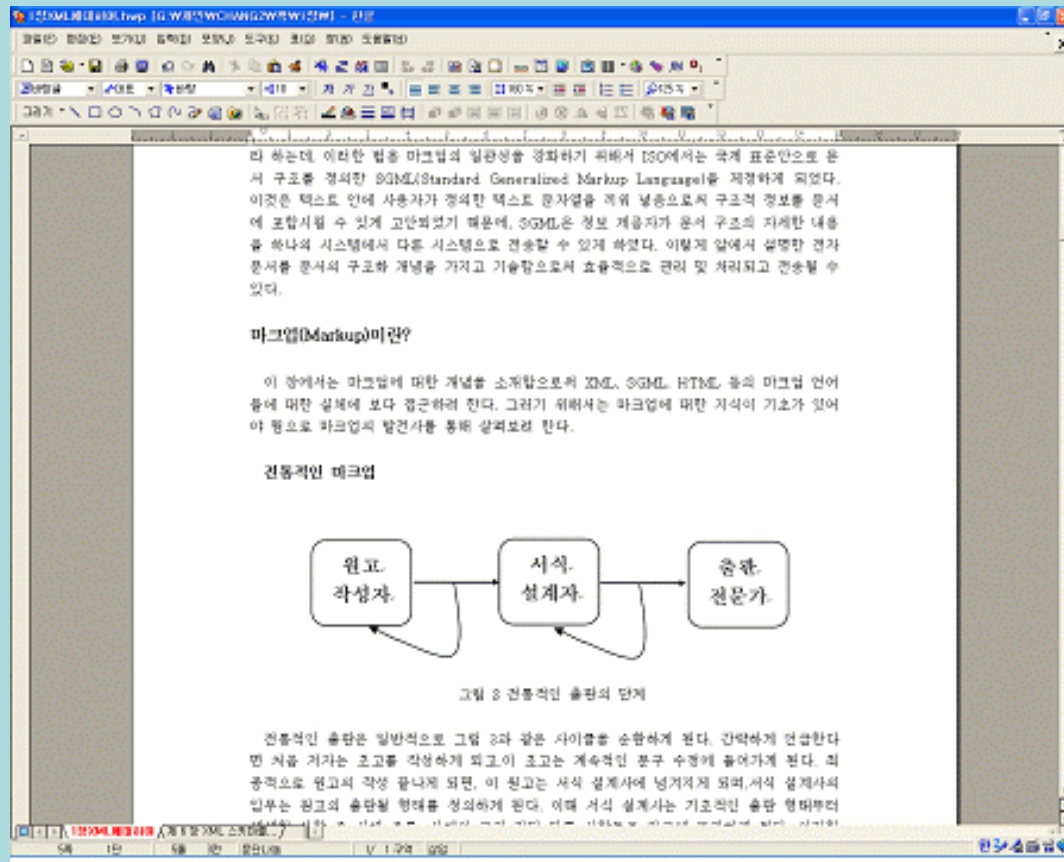


마크업(Markup) 이란 ?

- 전자적(**electronic**) 마크업
 - ▣ 컴퓨터 시스템을 이용한 마크업
 - ▣ 특수하게 정의된 구문을 가지고 응용에 의존적으로 다양한 형태로 표현
 - ▣ 응용 프로그램 마다 마크업 명령이 달라 이를 익히기 위해 많은 시간과 비용이 소요
 - ▣ 새로운 출력 장치가 기존의 장비에 추가될 때마다 마크업 변경이 요구
 - ▣ (종류) 절차적 마크업, 서술적 마크업, 범용 마크업

마크업(Markup)이란 ?

- 전자적(electronic) 마크업
 - 문서 작성 시스템



마크업(Markup)이란 ?

- 절차적(Procedural) 마크업 : **specific markup**
 - ▣ 단순히 문서의 정보와 속성만을 분석하여 이에 따라 처리하는 낮은 수준의 마크업
 - ▣ 어플리케이션에 의존적이므로 다른 어플리케이션에서 사용하기 위해 마크업 변환이 필요
 - ▣ 워드프로세서에서 텍스트의 외형과 위치를 지정하기 위해 사용
 - ▣ 텍스트와 같이 특정 코드로 저장
 - ▣ 문제점
 - 문서의 구조 정보를 기록하지 않음(문서 외형을 위해 사용)
 - 유연성이 부족(이식성 결여)
 - 처리가 느리고, 오류 발생이 많음

마크업(Markup)이란 ?

□ 절차적(Procedural) 마크업 : specific markup

- 마크업의 영향 범위에 따른 분류
 - 한 문자에 영향을 미치는 명령
 - 다음 명령까지 계속되는 명령
 - 페이지에 영향을 미치는 명령으로 새로운 라인이나 페이지 시작 시 동작
- 마크업 명령은 다음의 기본적인 방법 사용
 - 특정 분리된 라인에 위치
 - 특별한 구분자(delimiter)로 묶음

마크업(Markup) 이란 ?

□ 절차적(Procedural) 마크업

■ (예) troff 절차적 마크업

.bp.

.ps 20.

.ft I.

.ce.

문서 기술 언어 SGML.

.sp 2.

.ps 10.

.ft B.

SGML 은 "Standard generalized Markup Language"의 약자로, "문서 기술 언어"로 번역할 수 있다.

.sp 2.

-인쇄 제어 명령 마크업들이 특정 분리된 라인에 위치

-라인 맨 앞에 "."으로 시작 하여 순수 텍스트와 구분

-.bp : 새로운 페이지로 개행

-.ps 20 :폰트 크기를 20

-.ft I : 글자를 이탤릭체

-.ce :텍스트를 중앙정렬

-.sp 2 : 두 행 띄우기

포맷처리
결과

문서 기술 언어 SGML.

SGML 은 "Standard generalized Markup Language" 의 약자로, "문서 기술 언어"로 번역할 수 있을 것이다.

□ 절차적(Procedural) 마크업

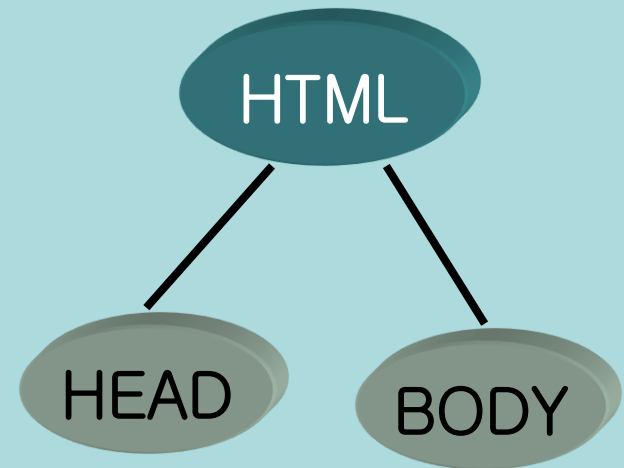
▣ (예) WORDSTAR 마크업 예

```
.PL 66.
.MT 6.
.MB 9.
.LH 12.
.UJ ON.
^A^BChapter 1.
SGML 소개^B.
SGML 은 문서를 표현하기 위한.
```

마크업(Markup) 이란 ?

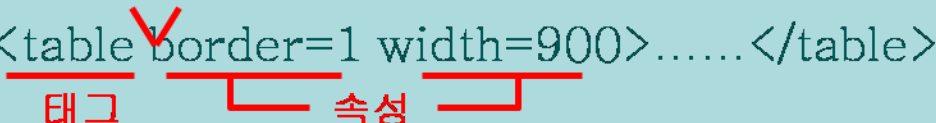
□ HTML(HyperText Markup Language)

- W3C의 명세
- 웹상에서 Hypertext 문서를 생성할 수 있는 간단한 마크업 언어
- SGML의 한 어플리케이션
- ASCII Text 양식의 문서
(일반 텍스트 편집기 작성 가능)
- HTML = SGML 선언 + a DTD
- HTML SGML



HTML

- Hyper Text Markup Language
- 웹(www)문서를 작성하기 위해 사용되는 언어
- Http 프로토콜 사용
- 한 쌍의 태그(tag)명령으로 이루어짐 – 한쌍이 아닌경우도 있음
 - ...,

- 태그 – 속성(attribute)로 옵션을 줌
 - 

```
<table border=1 width=900>.....</table>
```
- 대소문자 구별하지 않음
- 2 이상의 공백 혹은 enter 무시 – 별도의 특수문자 기호 필요
 - Enter-
, 공백-

웹 프로토콜

1. 웹 프로토콜 – HTTP Request(1)

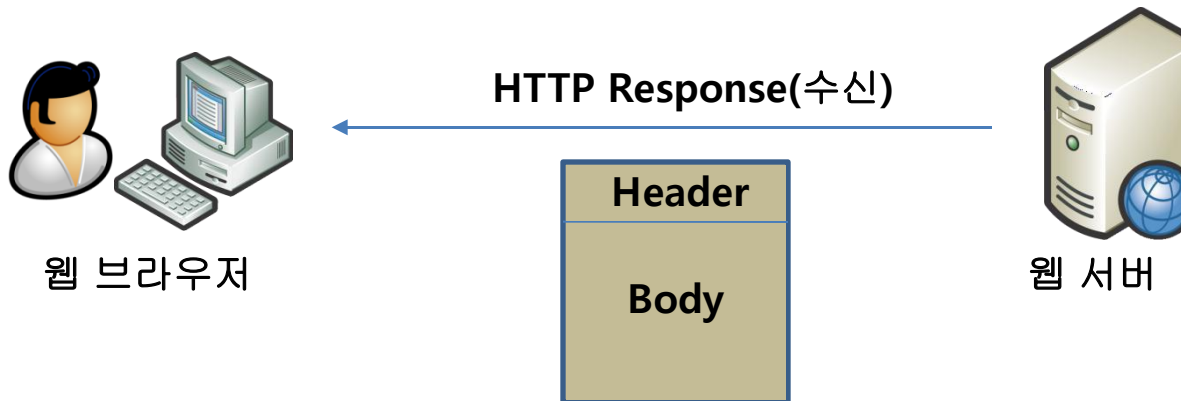
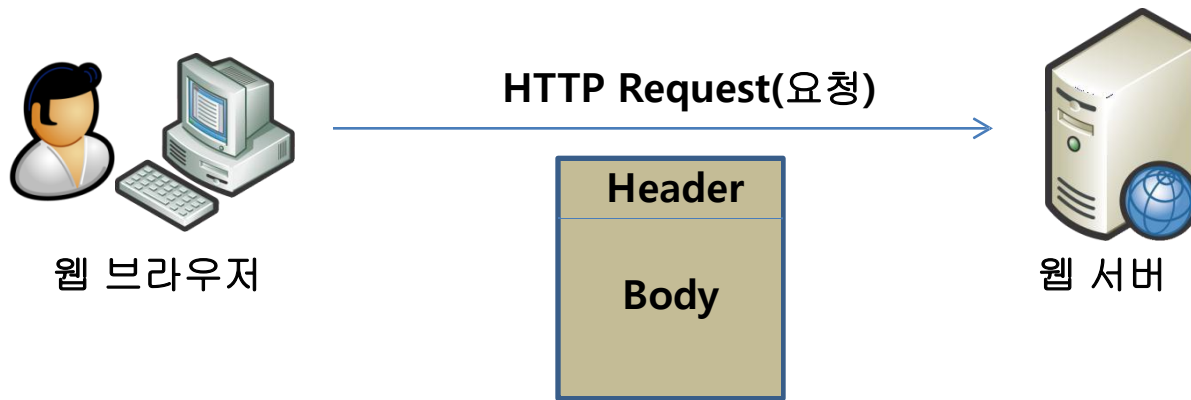


그림. HTTP 송, 수신

1. 프로토콜 – HTTP Request(2)

○ HTTP Request에는 무엇이 포함되어 있는가 ?

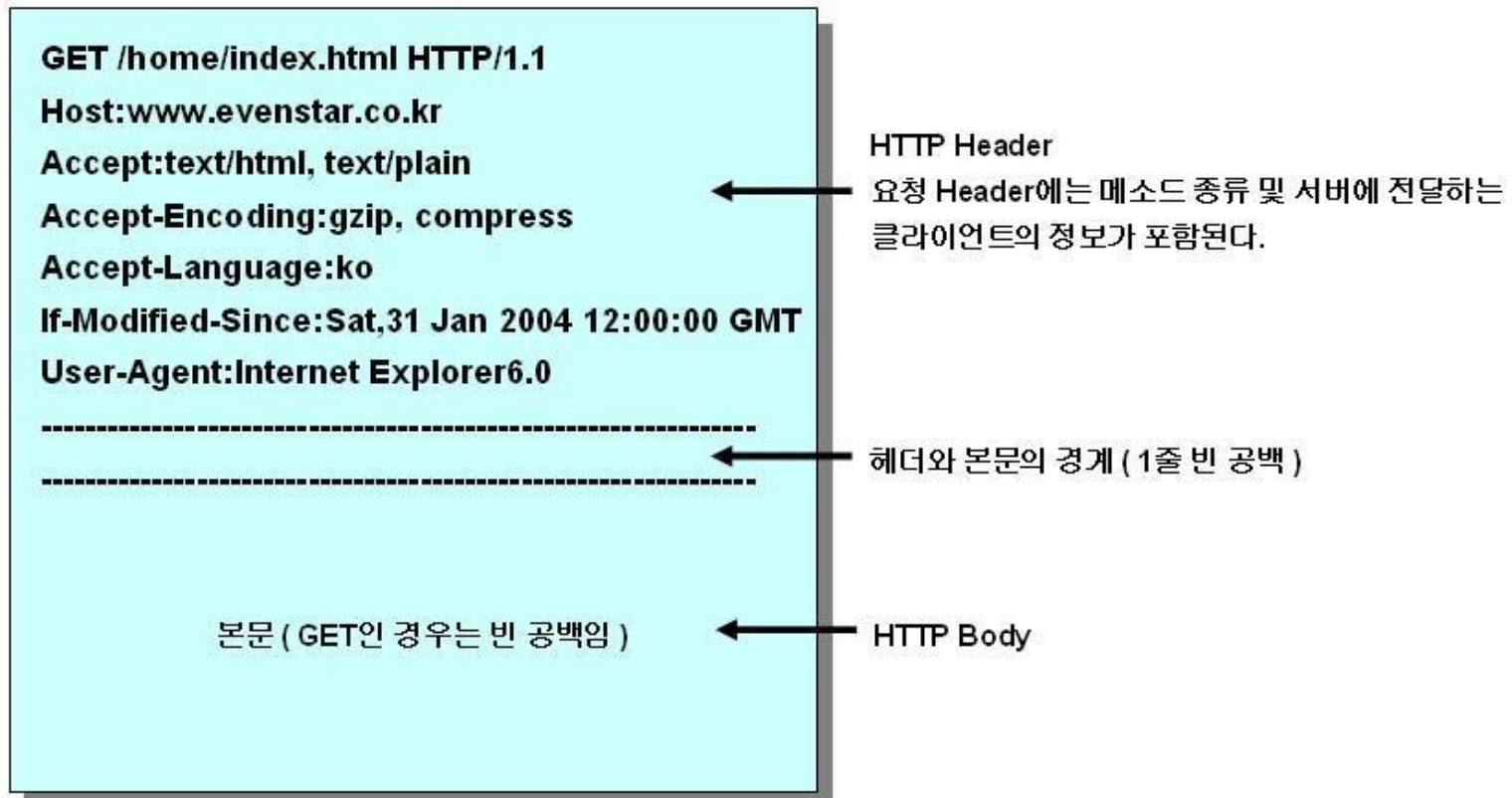
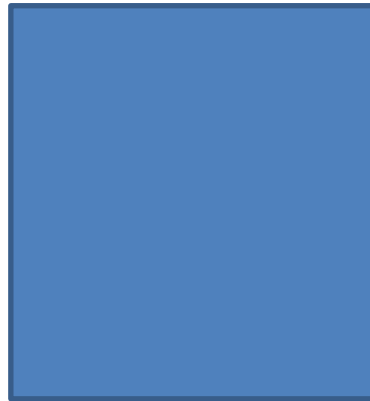
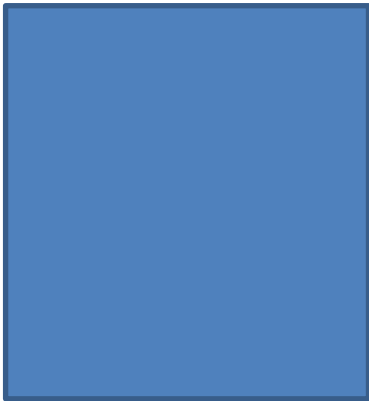


그림. HTTP Request 정보



1. 웹 프로토콜 – HTTP Request(3)

○ HTTP Request에 포함된 상세 정보

① GET /index.html HTTP/1.1	// 요청 URL 정보(메소드, 페이지) 및 HTTP 버전
② user-agent: MSIE 6.0; Windows NT 5.0	// 사용자 웹 브라우저 종류
③ accept: text/html; */*	// 요청 데이터 타입
④ cookie: name = value	// 쿠키(인증 정보)
⑤ referer: http://www.bbb.com	// 경유지 URL
⑥ host: www.aaa.co.kr	// 요청 도메인

1. 웹 프로토콜 - 방식

GET	지정된 리소스(URI)를 요청
POST	서버가 클라이언트의 폼 입력 필드 데이터의 수락을 요청. 클라이언트는 서버로 HTTP Body 에 Data 를 전송한다
HEAD	문서의 헤더 정보만 요청. 응답 데이터(body) 를 받지 않는다
PUT	클라이언트가 전송한 데이터를 지정한 uri 로 대체 한다 ftp 의 put 와 동일. 역시 클라이언트는 서버로 HTTP Body 에 Data 를 전송한다
DELETE	클라이언트가 지정한 URI 를 서버에서 삭제
TRACE	클라이언트가 요청한 자원에 도달하기 까지의 경로를 기록하는 루프백(loop back) 검사용. 클라이언트가 요청 자원에 도달하기 까지 거쳐가는 프록시나 게이트 웨이의 중간 경로부터 최종 수신 서버까지의 경로를 알아낼 때 사용.

1. 웹 프로토콜 – HTTP Request(6)

○ GET Method

2,083 정도의 길이 데이터만을 처리(게시판 글 입력 처리 불가 등)

Method	구조	설 명
GET	GET [request-uri]?query_string HTTP/1.1 Host:[Hostname] 혹은 [IP]	GET 요청 방식은 요청 URI(URL)가 가지고 있는 정보를 검색하기 위해 서버 측에 요청하는 형태이다.

○ HTTP GET 구조 (URI + Query String)

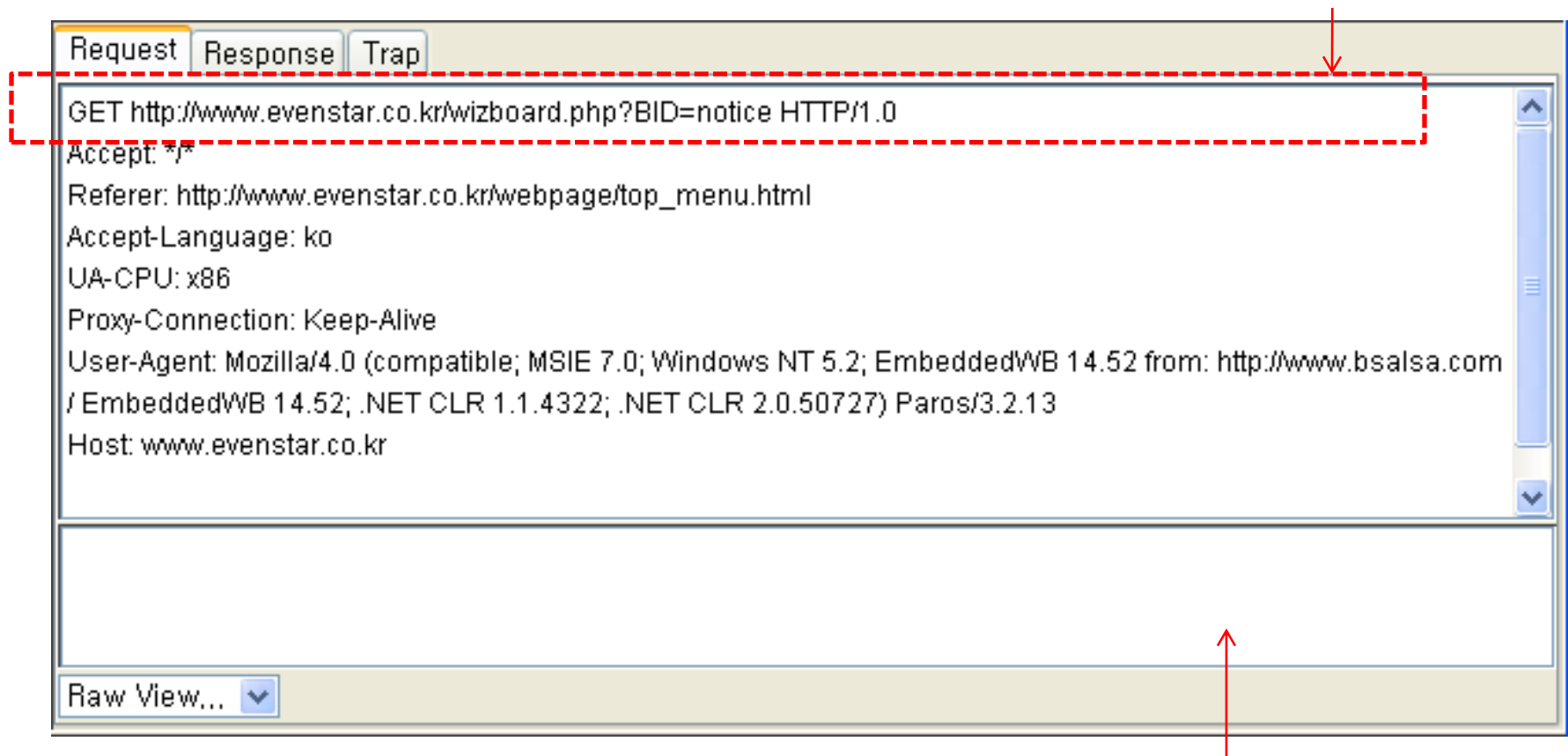
http://www.evenstar.co.kr/webpage/biglook_a.html (http header에 포함)
URI

<http://www.evenstar.co.kr/wizboard.php?BID=notice> (http header에 포함)
URI Query String

1. 웹 프로토콜 – HTTP Request(7)- 파로스 툴

○ GET Method

GET 요청



메시지(Body)는 없음

1. 웹 프로토콜 – HTTP Request(8)

○ POST Method

길이 제한이 없어 많은 입력 데이터를 처리(게시판 입력 글 처리 가능)

Method	전송 형태	설 명
POST	POST [request-uri] HTTP/1.1 Host:[Hostname] 혹은 [IP] Content-Length:[Bytes] Content-Type:[Content Type]	게시판 등과 같은 폼 데이터 및 CGI 프로그램으로 구성된 페이지를 위해 처리하기 위해 POST 방식으로 전송하게 되며, 웹 브라우저와 시스템 간 데이터 처리로 웹 브라우저에는 페이지 정보만을 확인할 수 있다.
	[query-string] 혹은 [데이터]	

○ HTTP POST 구조 (URI + Query String)

<http://www.aaa.co.kr/wizboard.php>
URI

(http header에 포함)

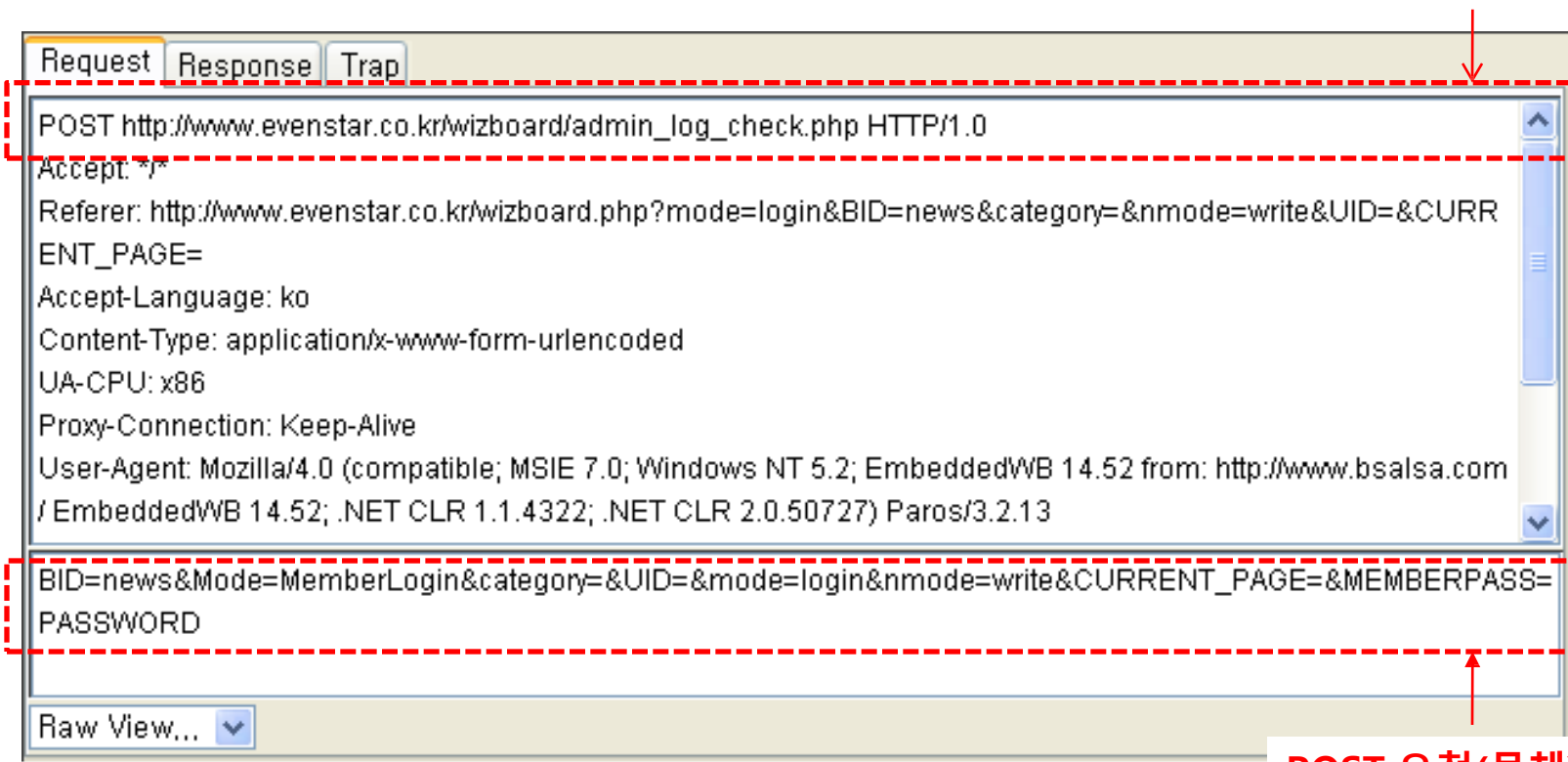
BID=notice
Query String

(http Body에 포함)

1. 웹 프로토콜 – HTTP Request(9)

○ POST Method

POST 요청(헤더)



POST 요청(몸체)

[주의] GET Method와의 차이는 무엇인가?

1. 웹 프로토콜 – HTTP Response(1)

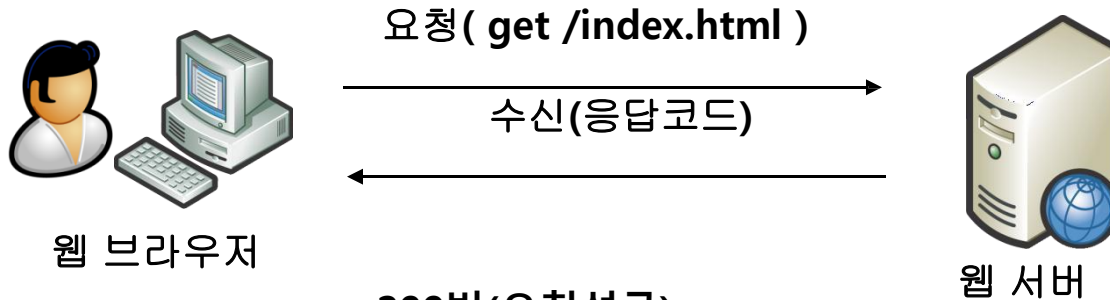
○ HTTP Response에 포함된 상세 정보

① HTTP/1.1 OK 200	// 프로토콜 버전 및 응답코드	Header
② Server: NCSA/1.4.2	// 웹 서버 배너 정보	
③ Content-type: text/html	// MIME타입	
④ Content-length: 107	// HTTP Body 사이즈	
빈 공백 1줄		
⑤ <html><head></head> <Title>http protocol</Title> <body> The understanding of http protocol </body></html>	// 페이지 구성 정보(HTML태그 등)	Body

- HTTP Header 포함 범위 : (1), (2), (3), (4)
- HTTP Body 포함 범위 : (5)

1. 웹 프로토콜 – HTTP Response(2)

HTTP Status Code(응답코드) 종류



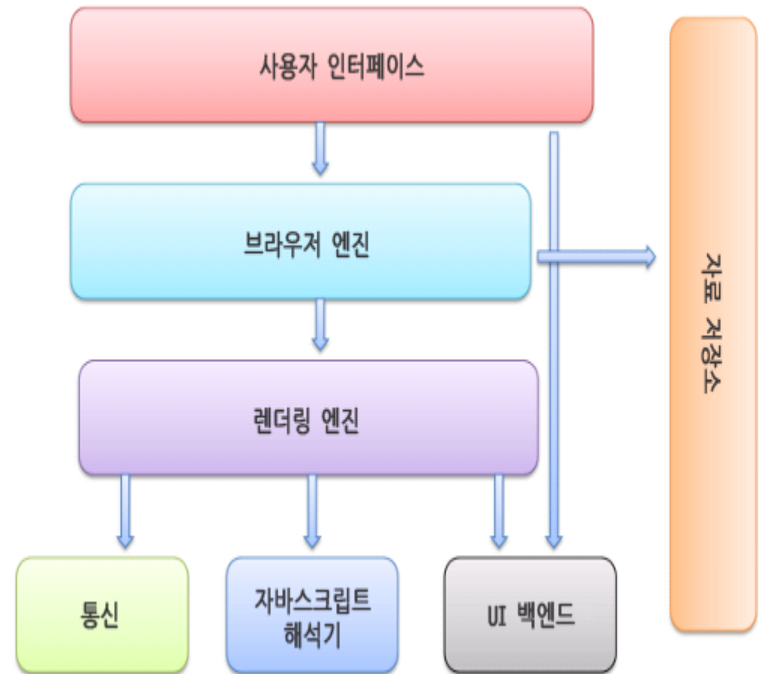
- 200번(요청성공)
- 201번(원격지서버에 파일 생성)
- 302번(페이지이동)
- 304번(로컬 캐쉬정보이용)
- 401번(인증실패)
- 403번(접근금지)
- 404번(페이지없음)
- 500번(서버에러)

브라우저 동작구조

1. 브라우저 주요 구성요소

1. Inside Browser

- **UI (User Interface)**
 - 각 브라우저의 UI는 앞서 설명한 바와 같이 주소바, 뒤로가기/앞으로가기 버튼, 북마킹 메뉴등을 공통적으로 포함합니다.
- 브라우저 엔진
 - 렌더링(rendering) 엔진에 작업을 요청하고 다루는 인터페이스 부분으로 볼 수 있습니다.
- 렌더링 엔진
 - 요청된 콘텐츠를 화면에 표시하기 위해 필수적인 부분입니다. 예를 들어 요청된 페이지가 HTML이라 할 때, 렌더링 엔진은 HTML과 CSS를 파싱하고 화면에 파싱된 콘텐츠를 표현하는 역할을 수행합니다.
- 네트워킹
 - HTTP request와 같은 네트워크 호출을 위해 필요한 부분입니다. 각 플랫폼에 의존적인 부분과 플랫폼에 독립적인 인터페이스 부분으로 구성됩니다.
- UI 백엔드
 - 콤보박스나 윈도우(window)와 같은 기본 위젯을 화면에 그리는 데(drawing) 필요한 부분입니다.
- 자바스크립트 해석기
 - 자바 스크립트 코드를 파싱하고 실행하는데 필요합니다.
- 데이터 스토리지
 - 퍼시스턴스 계층 (persistence layer). 브라우저는 쿠키등과 같이 데이터를 로컬영역에 저장할 공간이 필요합니다. HTML5 같은 경우에는 “web database”를 따로 정의해 놓았는데 브라우저 안에 완전한 형태의 데이터베이스가 존재해야 합니다. chrome의 경우에 기존에 sqlite와 같은 경량형 RDB를 사용하고 있으며 향후에는 LevelDB라는 Key/Value 기반 데이터베이스가 HTML5용으로 사용될 듯 합니다.



크롬은 다른 브라우저와는 다르게 렌더링 엔진의 인스턴스를 각 탭별로 사용합니다. 즉 각 탭별로 별도의 렌더링 엔진을 사용하여 화면을 표시합니다.

2 렌더링 엔진

1. Inside Browser

- 브라우저별 사용된 레이아웃 엔진의 종류.

- IE 9 - Trident
- 크롬 10 - Webkit
- 파이어폭스 4 - Gecko
- 사파리 5- Webkit
- 오페라 11 - Presto

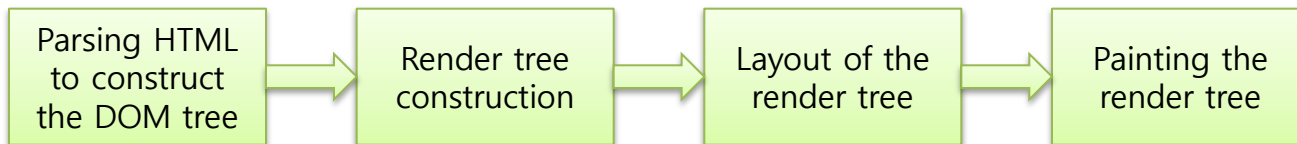
참고

- 각 브라우저에 사용된 자바스크립트 엔진
 - IE 9 - Chakra
 - 크롬 10 - V8 (Crankshaft)
 - 파이어폭스 4 - JaegerMonkey
 - 사파리 5- Nitro
 - 오페라 11 - Carakan

- 렌더링 엔진은 네트워크 계층으로부터 요청된 페이지의 콘텐츠를 얻어 오는데 대략 **8k** 크기의 청크 단위로 이를 수행합니다.

- 렌더링 엔진 기본 흐름

- 렌더링 엔진은 **HTML** 문서를 파싱 할 것이고 **"content tree"**라 불리는 트리형태로 **DOM** 노드를 생성하게 되며
- **HTML**의 스타일 정보들은 별도의 **render tree**로 구성됩니다. 렌더트리는 색상이나 **dimension**들을 포함하는 그래픽 속성들의 사각형을 포함하며 사각형들은 올바른 순서대로 화면에 표시됩니다.
- 렌더트리를 생성한 후에 렌더트리의 **"layout"** 과정을 수행합니다. 다시 말해서 각 노드들이 화면상에서 정확히 그들이 놓여있어야 할 곳에 나타나게 됩니다.
- 다음과정은 **"painting"**인데 이 과정에서 렌더 트리를 순회하면서 스타일 정보를 찾아서 **UI** 백엔드를 이용하여 각 노드들을 화면상에 표현하게 됩니다.



렌더링 엔진 기본 흐름

3.1 렌더링 엔진 종류

1. Inside Browser

- 트라이던트(Trident)
 - 마이크로소프트 윈도우 버전의 **IE가 채용**하고 있는 레이아웃 엔진
 - **HS HTML**로 알려져 있으며 이 엔진은 **1997년 IE 4.0**부터 도입되어 현재까지 새로운 기술을 탑재해 꾸준히 업그레이드 되고 있음
 - 트라이던트는 개발자가 자사 응용 프로그램에 웹 브라우징을 쉽게 추가할 수 있도록 모듈 방식으로 제공됨
 - **IE**의 시장 독점이 계속되어온 탓에 트라이던트 엔진 역시 오랫동안 독점적인 위치를 차지하게 되었고, 특히 한동안 **IE**의 개발을 하지 않았기에 역시 트라이던트 코어의 업데이트도 상당기간 이뤄지지 않았으며, 결국 **W3C** 표준과 맞지 않고 내부의 버그와 보안 문제들이 누적되어 브라우저의 호환성이나 성능도 매우 부족한 모습을 보였었음.
 - **IE7**에서 **MS**는 트라이던트 레이아웃에 큰 변화를 주어 웹 표준 대응을 개선하고 새로운 기술을 지원하게 되었지만, 상대적으로 다른 브라우저들에 사용된 엔진들에 비해서는 크게 뒤떨어진 편이라고 할 수 있음
 - **IE9**에는 **SVC, HHTML, HTML5** 지원이 추가된 트라이던트 **5.0** 버전이 적용되었음
- 웹킷(WebKit)
 - 웹킷은 **애플, 노키아, 어도비, 구글 등에 의해 공동 개발된 오픈 소스 기반의 웹 브라우저 엔진**
 - 웹킷은 **KDE의 KHTML**와 **KJS**를 기반으로 제작되었으며 단순히 레이아웃 엔진이라고 부르지 않는 이유는 바로 **HTML, SVG**를 위한 레이아웃, 렌더링, **DOM** 라이브러리인 **Webcore**와 웹킷 기능을 위한 자바스크립트 엔진인 **JavaScriptCore**, 그리고 자바스크립트 오류를 검진할 수 있는 **Drosera** 디버거를 함께 포함하고 있는 상위 개념이기 때문입니다.
 - 웹킷 엔진은 애플 사파리와 구글 크롬 외에도 맥용 옴니웹, 시이라, 아로라, 미도리, 유즈블, **iCab**, 어도비 통합 런타임 등에도 사용되고 있음
 - 특히 웹킷은 휴대폰 웹 브라우저에서 보다 많이 사용되고 있는데, 대표적으로 구글의 안드로이드, 애플의 아이폰, 그리고 노키아의 **Series 60**의 브라우저들이 모두 웹킷 엔진 기반으로 되어 있음
 - 또한 위젯 엔진에서의 사용도 증가하고 있어 차이나 텔레콤의 **BAE**, 애플의 대시보드, 노키아 **WRT** 역시 모두 웹킷 엔진이 적용되어 있음
 - <http://webkit.org>

3.2 렌더링 엔진

- 게코(Gecko)
 - 게코는 **오픈 소스 코드**이며 **C++로 만들어진 레이아웃 엔진**
 - 넷스케이프에서 개발했지만 넷스케이프의 몰락 이후 현재는 **모질라에서 유지-보수를 담당**하고 있으며 **파이어폭스에서 사용**되고 있음
 - 게코 엔진의 가장 큰 특징은 완벽한 오픈 소스 코드이기 때문에 전세계 개발자들이 이 코드를 사용해 기능을 추가할 수 있게 되면서 개발 수준이 비약적으로 향상되었다는 점
 - 또한 오픈 소스이기 때문에 게코 엔진을 사용하는 브라우저도 많으며, 이로 인해 시장에서 꾸준히 높은 점유율을 유지하고 있음
 - 현재 **게코엔진을 사용한 브라우저는 파이어폭스를 비롯해 넷스케이프 6-9, SeaMonkey, Camino, Mozilla, Flock, Galeon 등으로 다양하며 구글 가젯 엔진도 게코 엔진을 사용**하였음.
 - 이 외에 게코 엔진은 뛰어난 확장성을 바탕으로 윈도우즈, **BSD**, 리눅스, 맥 **OS X**에서 사용이 가능하며 참고로 파이어 폭스 4에는 **Gecko 2** 최신 버전이 적용되었음.
- 프레스토(Presto)
 - 프레스토는 오페라 소프트웨어가 **오페라 브라우저를 위해 개발한 레이아웃 엔진**.
 - 프레스토 엔진은 **2003년 오페라 7.0에 처음으로 적용**되었으며 **렌더링 속도의 최적화에 중점**을 두고 가장 빠른 속도를 제공해왔음
 - 하지만 프레스토 엔진은 오페라 브라우저 또는 관련된 제품만 사용이 가능하고 소스나 바이너리 **DLL**등은 공개적으로 사용할 수 없기 때문에 호환성이나 대중성에서는 다소 부족한 모습임
 - 프레스토는 다른 엔진에 비해 뛰어난 스크립트 처리 능력과 빠른 자바스크립트 실행 속도를 제공하지만, 상업용 엔진이기 때문에 사용되는 브라우저가 제한되어 개발 및 발전을 가로막고 있다고 평가를 받고 있음. 오페라 11에는 프레스토 2.7 버전이 사용되어 이전 2.6 버전에 비해 더 많은 마크업과 **API**를 지원함

* KDE : K Desktop Environment, 그놈과 유사한 프로젝트

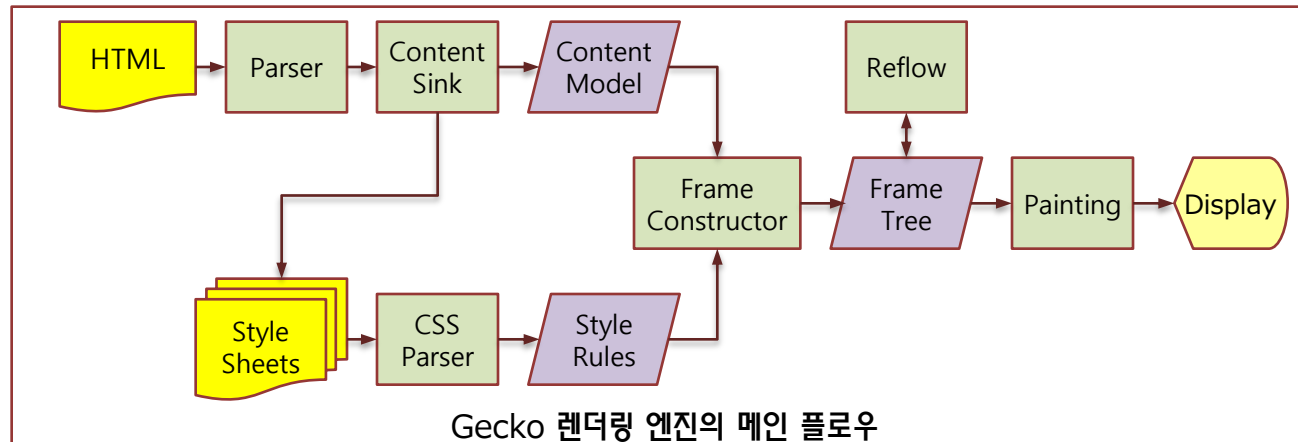
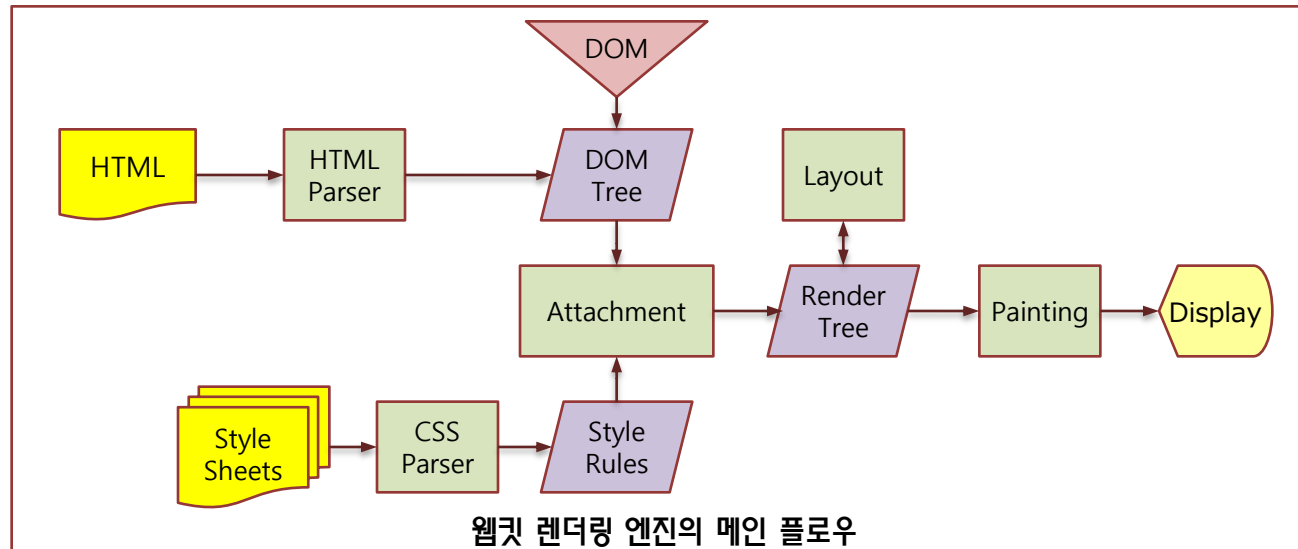
* KHTML : KDE 프로젝트가 개발한 HTML 레이아웃 엔진

* KJS : KDE 프로젝트가 개발한 JavaScript 엔진

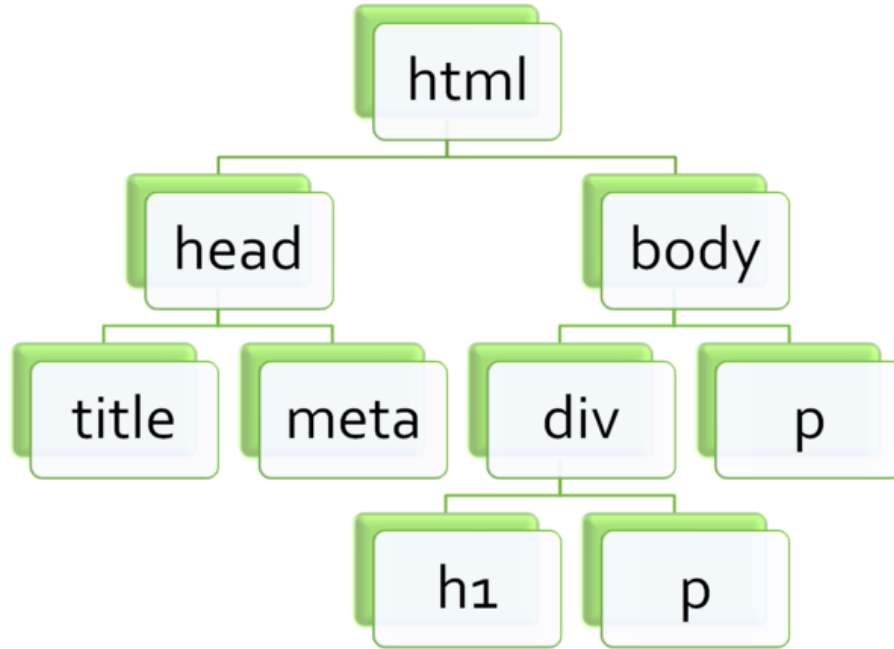
* SVG(Scalable Vector Graphics) : 2차원 벡터 그래픽을 표현하기 위한 XML기반의 파일 형식

4. 렌더링 엔진의 메인 플로우

1. Inside Browser



DOM(브라우저 렌더링에 의해 트리형태로 구성된 파싱문서)



자바스크립트
(DOM 트리 접근자)

1. 요소 노드 : 태그명 - 요소 노드는 다른 요소 포함 가능
2. 텍스트 노드 : 요소 안에 포함된 text
3. 속성 노드 : 요소에 대한 정확한 정보 표현. (항상 요소 노드 안에 포함)

Html 태그 종류

블락요소: hn, div, p, table, ul,li,ol, ...

- 개행:hn, p

- 개행없는: div, hr

인라인요소: span, input ,img, a, ...

iframe

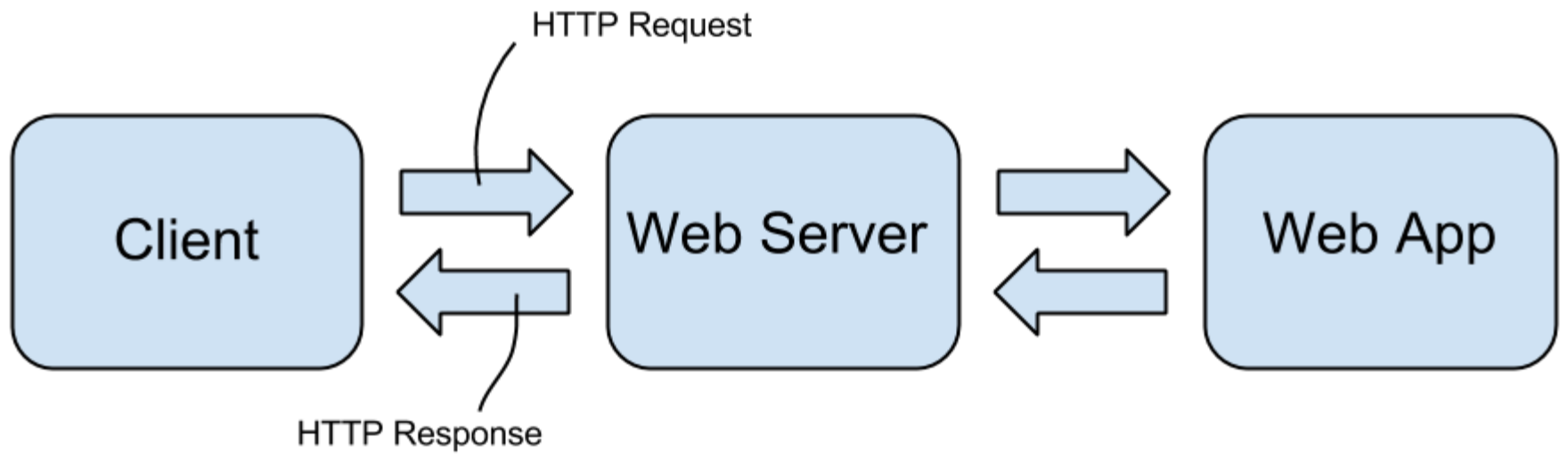
Flask 템플릿 jinja2

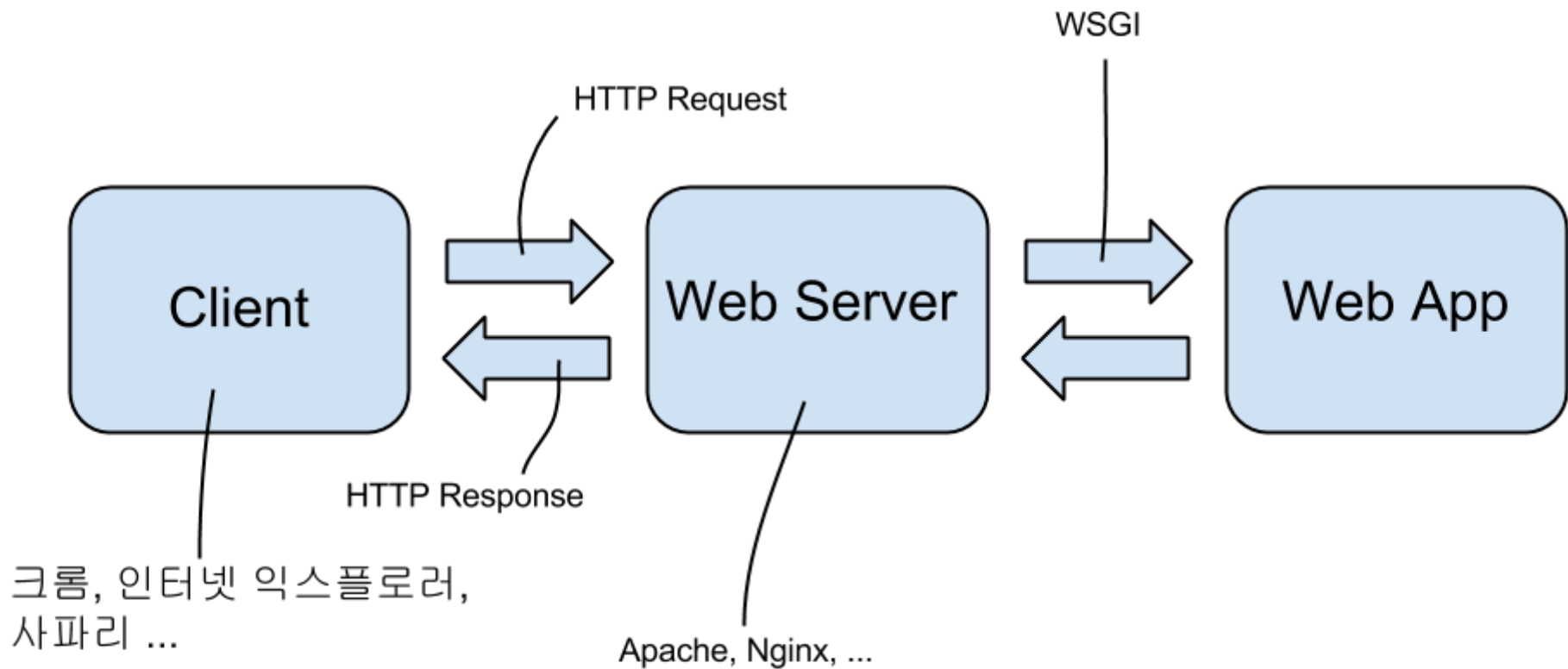
템플릿이 뭐예요?

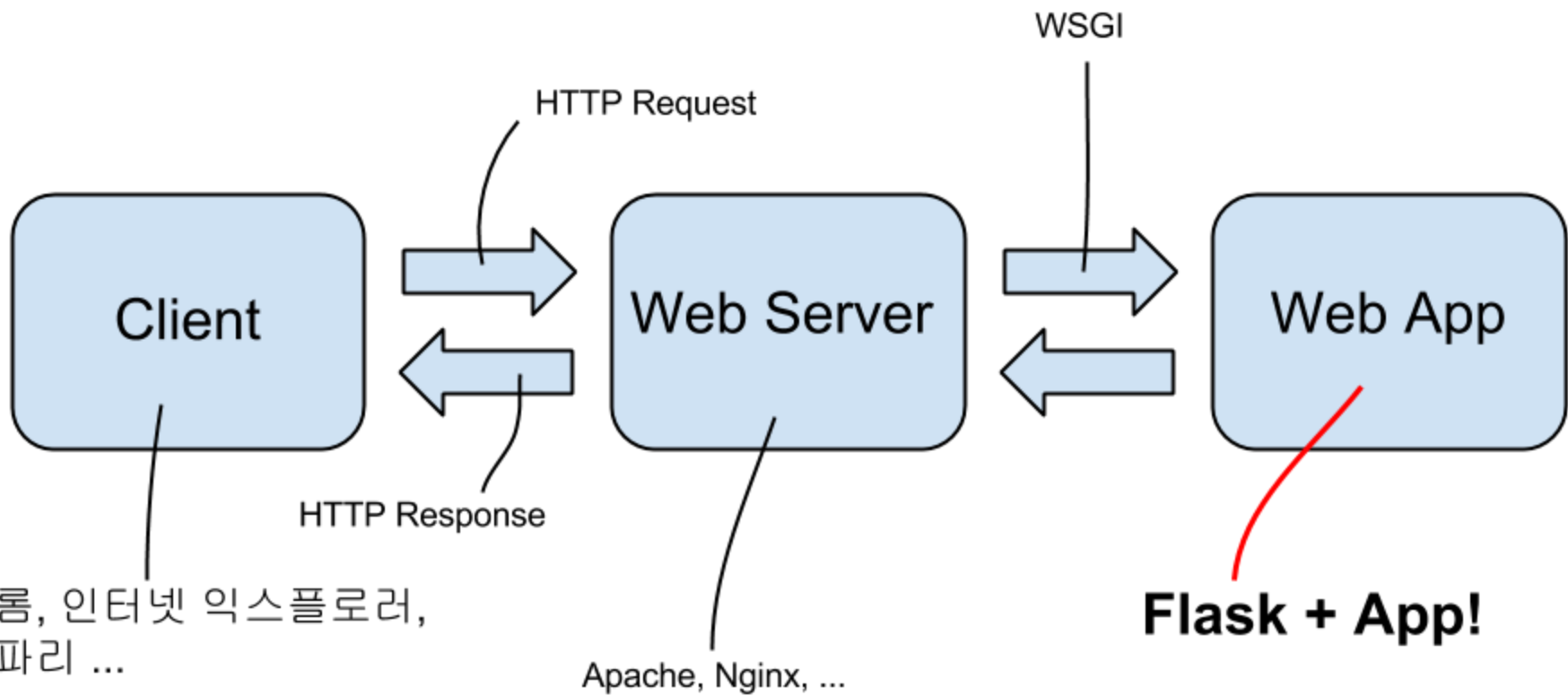
- **웹 개발**에서 템플릿은 데이터를 덧붙여서 HTML 파일을 지칭
- 웹 템플레이트 (Web Template)

???

- 웹 개발 간단 리뷰
- 일반적인 웹 서비스의 구조
 - 클라이언트
 - 웹 서버
 - 웹 애플리케이션 (+ 데이터베이스)







클라이언트 -> 웹서버

```
C:\coding\flask_test>python hello.py
* Running on http://127.0.0.1:5000/
* Restarting with reloader
127.0.0.1 - - [08/Jul/2013 23:20:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Jul/2013 23:20:23] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [09/Jul/2013 00:24:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2013 00:24:25] "GET /favicon.ico HTTP/1.1" 404 -
```

- Flask가 제공하는 개발 전용 웹서버
 - Apache 등을 설치할 필요 없이 개발 환경에서 간단한 웹서버를 구동
 - 클라이언트에서 받은 HTTP request 실시간 확인
 - 실제 서비스 용도는 아님!

웹서버 -> 웹 애플리케이션

- 웹서버로 온 HTTP Request -> 웹앱 전달
- 웹 애플리케이션은 request에 대응해서,
 - 데이터베이스에서 데이터를 가져온 후 가공
 - 데이터를 HTML에 입힌 후 전송 (response)
 - 데이터를 입히는 작업을 "렌더링" 혹은 "템플레이팅"이라고 부름
 - 이 작업을 하는 도구가 "템플릿 엔진"

Flask의 템플릿 엔진

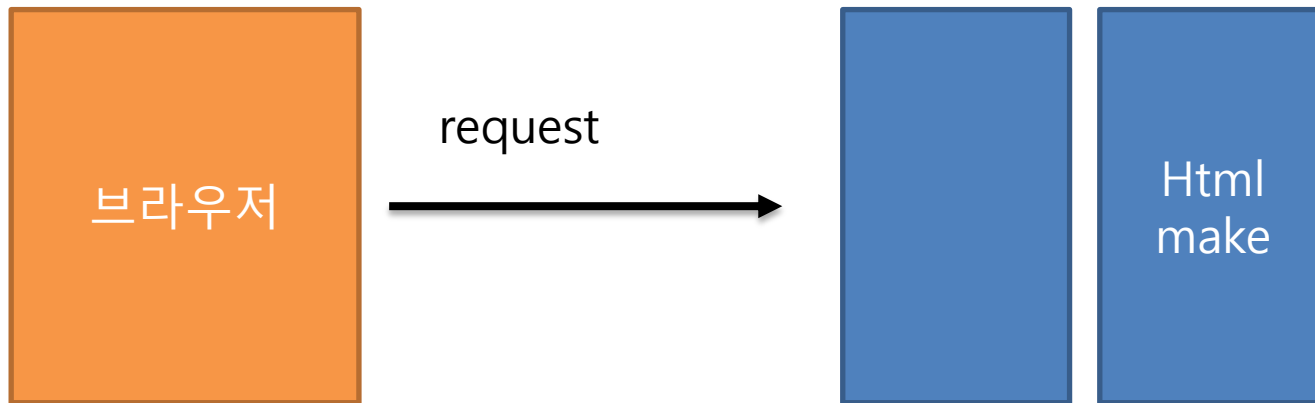
- Jinja2
 - Flask에 포함된 템플릿 엔진
 - 하지만 Flask와 독립된 프로젝트
 - Flask 프로젝트에서 Jinja2 아닌 다른 템플릿 엔진 사용 가능 (Genshi, Mako 등)
 - 그러나 Flask의 플러그인 호환성을 위해 Jinja2는 기본적으로 설치 해야 됨

Jinja2 기본 Syntax

- 표현식은 {% ... %}
- 변수는 {{ ... }}
- 표현식은 파이썬과 비슷
 - 예) team_list라는 리스트를 렌더링할때:
{% for team in team_list %}
{{ team }}
{% endfor %}

Bootstrap(css)라이브러리

- 반응형웹: 스마트폰, 태블릿, pc



10분 퀴즈

숫자1:

5



숫자2:

12



계산

합은: 17

템플릿 렌더링 예제

K리그에는 어떤 구단들이 있나요?

- 강원 FC
- 경남 FC
- 대구 FC
- 대전 시티즌
- 부산 아이파크
- FC 서울
- 성남 일화 천마
- 수원 삼성 블루윙즈
- 울산 현대
- 인천 유나이티드
- 전남 드래곤즈
- 전북 현대 모터스
- 제주 유나이티드
- 포항 스틸러스

플라스크 스터디 2013

템플릿 렌더링 예제

- 일반 HTML 페이지
- 데이터가 바뀔 때마다 HTML을 수정해야 됨

```
1  <!doctype html>
2  <html>
3    <head>
4      <title>플라스크 테스트</title>
5    </head>
6    <body>
7      <h1>K리그에는 어떤 구단들이 있나요?</h1>
8      <ul>
9        <li>강원 FC</li>
10       <li>경남 FC</li>
11       <li>대구 FC</li>
12       <li>대전 시티즌</li>
13       <li>부산 아이파크</li>
14       <li>FC 서울</li>
15       <li>성남 일화 천마</li>
16       <li>수원 삼성 블루윙즈</li>
17       <li>울산 현대</li>
18       <li>인천 유나이티드</li>
19       <li>전남 드래곤즈</li>
20       <li>전북 현대 모터스</li>
21       <li>제주 유나이티드</li>
22       <li>포항 스틸러스</li>
23     </ul>
24   </body>
25   <footer>
26     <span>플라스크 스터디 2013</span>
27   </footer>
28 </html>
```

템플릿 렌더링 예제

- 템플릿화 된 HTML
- HTML이 아닌 템플릿 언어로 프레젠테이션 로직을 표현
- 템플릿 렌더링을 통해, HTML 수정 없이 동적인 페이지를 제작

```
1  <!doctype html>
2  <html>
3    <head>
4      <title>플라스크 테스트</title>
5    </head>
6    <body>
7      <h1>{{ name }}</h1>
8      <ul>
9        {% for team in team_list %}
10       <li>{{ team }}</li>
11       {% endfor %}
12     </ul>
13   </body>
14   <footer>
15     <span>플라스크 스터디 2013</span>
16   </footer>
17 </html>
18
```

템플릿 렌더링 예제

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route("/")
5  def hello():
6      return open(r'C:\coding\flask_test\templates\hello.html', 'rb').read()
7
8  if __name__ == "__main__":
9      app.run(debug=True)
10
```

```
1  # -*- coding: utf-8 -*-
2  from flask import Flask, render_template
3  app = Flask(__name__)
4
5  @app.route("/")
6  def hello():
7      name = u'K리그에는 어떤 구단들이 있나요?'
8      team_list = [u'강원 FC', u'경남 FC', u'대구 FC', u'대전 시티즌', u'부산 아이파크',
9                  u'FC 서울', u'성남 일화 천마', u'수원 삼성 블루윙즈', u'울산 현대',
10                 u'인천 유나이티드', u'전남 드래곤즈', u'전북 현대 모터스', u'제주 유나이티드',
11                 u'포항 스틸러스']
12      return render_template('hello.html', name=name, team_list=team_list)
13
14  if __name__ == "__main__":
15      app.run(debug=True)
16
```

렌더링된 페이지

K리그에는 어떤 구단들이 있나요?

- 강원 FC
- 경남 FC
- 대구 FC
- 대전 시티즌
- 부산 아이파크
- FC 서울
- 성남 일화 천마
- 수원 삼성 블루윙즈
- 울산 현대
- 인천 유나이티드
- 전남 드래곤즈
- 전북 현대 모터스
- 제주 유나이티드
- 포항 스틸러스

플라스크 스테디 2013

템플릿 렌더링 예제

```
1  # -*- coding: utf-8 -*-
2  from flask import Flask, render_template
3  app = Flask(__name__)
4
5  @app.route("/")
6  def hello():
7      name = u'K리그에는 어떤 구단들이 있나요?'
8      team_list = [u'강원 FC', u'경남 FC', u'대구 FC', u'대전 시티즌', u'부산 아이파크',
9                  u'FC 서울', u'성남 일화 천마', u'수원 삼성 블루윙즈', u'울산 현대',
10                 u'인천 유나이티드', u'전남 드래곤즈', u'전북 현대 모터스', u'제주 유나이티드',
11                 u'포항 스틸러스']
12     return render_template('hello.html', name=name, team_list=team_list)
13
14 if __name__ == "__main__":
15     app.run(debug=True)
16
```

```
1  # -*- coding: utf-8 -*-
2  from flask import Flask, render_template
3  app = Flask(__name__)
4
5  @app.route("/")
6  def hello():
7      name = u'서울을 연고지로 하는 야구 구단들은 누구인가요?'
8      team_list = [u'LG 트윈스', u'두산 베어스', u'넥센 히어로즈']
9      return render_template('hello.html', name=name, team_list=team_list)
10
11 if __name__ == "__main__":
12     app.run(debug=True)
13
14
```

렌더링된 페이지

서울을 연고지로 하는 야구 구단들은 누구인가요?

- LG 트윈스
- 두산 베어스
- 넥센 히어로즈

플라스크 스테디 2013

- HTML 수정 없음!
- “내용”과 “프레젠테이션”의 분리
 - 내용은 서버에서 데이터를 가공
 - 프레젠테이션 로직은 템플릿 언어로 표현

Jinja2 기본 Syntax

- 다음과 같은 표현도 가능:
 - `{{ foo.bar }}`
 - `{{ foo['bar'] }}`
- 예제 1)
 - View에서 다음 dict object를 보내면,
`var = { 'name' : u'플라스크', 'date' : u'7/9/2012' }`
 - 템플릿에서 `{{ var.name }}`이나 `{{ var.date }}`로 사용 가능

Jinja2 기본 Syntax

- 예제 2)

```
class Study():  
    name = u'플라스크'  
my_study = Study()
```

- view에서 my_study를 넘겨준 후,
{{ my_study.name }}

Jinja2 기본 Syntax

- flow control:

```
{% if kenny.sick %}
```

```
    Kenny is sick.
```

```
{% elif kenny.dead %}
```

```
    You killed Kenny!  You bastard!!!
```

```
{% else %}
```

```
    Kenny looks okay --- so far
```

```
{% endif %}
```

Jinja2 기본 Syntax

- function:

```
{% for number in range(5) %}
```

```
<li class="empty"><span>...</span></li>
```

```
{% endfor %}
```

Jinja2 필터

- filter:

`{{ name|capitalize }}`

-> `capitalize(name)`과 같은 효과

- argument도 사용 가능

`{{ "%s - %s"|format("Hello?", "Foo!") }}`

-> "Hello? - Foo!"

Jinja2 필터

- 쉽게 custom filter를 등록해서 사용

- 예제:

```
from jinja2 import contextfilter
```

```
@contextfilter
```

```
def my_filter(context, value):
```

```
    result = do_something(value)
```

```
    return result
```

```
-> {{ name|my_filter }}
```

Jinja2 테스트

- 표현식 내에서 사용 가능한 간단한 테스트들

{% if loop.index **is divisibleby(3)** %}

- number(), string(), defined(), 등등...
- <http://jinja.pocoo.org/docs/templates/#builtin-tests>

Jinja2 템플릿 상속

- 예제) hello.html의 header와 footer 부분을 base.html로 떼어내기

```
1  <!doctype html>
2  <html>
3    <head>
4      <title>플라스크 테스트</title>
5    </head>
6    <body>
7      <h1>{{ name }}</h1>
8      <ul>
9        {% for team in team_list %}
10       <li>{{ team }}</li>
11       {% endfor %}
12     </ul>
13   </body>
14   <footer>
15     <span>플라스크 스터디 2013</span>
16   </footer>
17 </html>
18
```


Jinja2 템플릿 상속

- 예제) hello.html의 header와 footer 부분을 base.html로 떼어내기

```
1      <!doctype html>
2      <html>
3      <head>
4          <title>플라스크 테스트</title>
5      </head>
6      <body>
7          {% block body %}
8          {% endblock %}
9      </body>
10     <footer>
11         <span>플라스크 스터디 2013</span>
12     </footer>
13 </html>
```

Jinja2 템플릿 상속

- 예제) hello.html의 header와 footer 부분을 base.html로 떼어내기
- 분리된 hello.html:

```
1      {% extends 'base.html' %}
2      {% block body %}
3      <h1>{{ name }}</h1>
4      <ul>
5          {% for team in team_list %}
6          <li>{{ team }}</li>
7          {% endfor %}
8      </ul>
9      {% endblock %}
10
```

Jinja2 템플릿 상속

- 한 템플릿 안에 여러 {% block %} 생성 가능

- 예제)

Base.html:

```
<!doctype html>
```

```
<html>
```

```
{% block header %}{% endblock %}
```

```
{% block css %}{% endblock %}
```

```
{% block body %}{% endblock %}
```

```
{% block footer %}{% endblock %}
```

```
{% block javascript %}{% endblock %}
```

```
</html>
```

Jinja2 Autoescaping

- 경우에 따라, 렌더링하는 변수에 HTML이 포함되는 경우가 있음:
 description = <h1>Hello</h1>
 {{ description }}
- 만일 적절한 처리가 되어 있지 않다면, 악성 스크립트가 렌더링되는 경우가 발생 -> **보안 위험 (XSS 공격)**
- 따라서, HTML 특수 기호 <, > 등은 렌더링할때 HTML로 인식되지 않게 처리 -> "autoescape" (기본 설정 on)

Jinja2 Autoescaping

- 만일 HTML을 렌더링 하고 싶으면, autoescape 설정을 끄면 됨:

```
description = <h1>Hello!</h1>
```

```
{% autoescape false %}
```

```
{{ description }}
```

```
{% endautoescape %}
```

-> <h1>이 HTML 태그로 렌더링 됨

- 보안 위험 주의!**

Jinja2의 모든 기능

- 다양한 Jinja2의 filter, built-in test들은 공식 문서를 참고!
- <http://jinja.pocoo.org/docs/templates/>