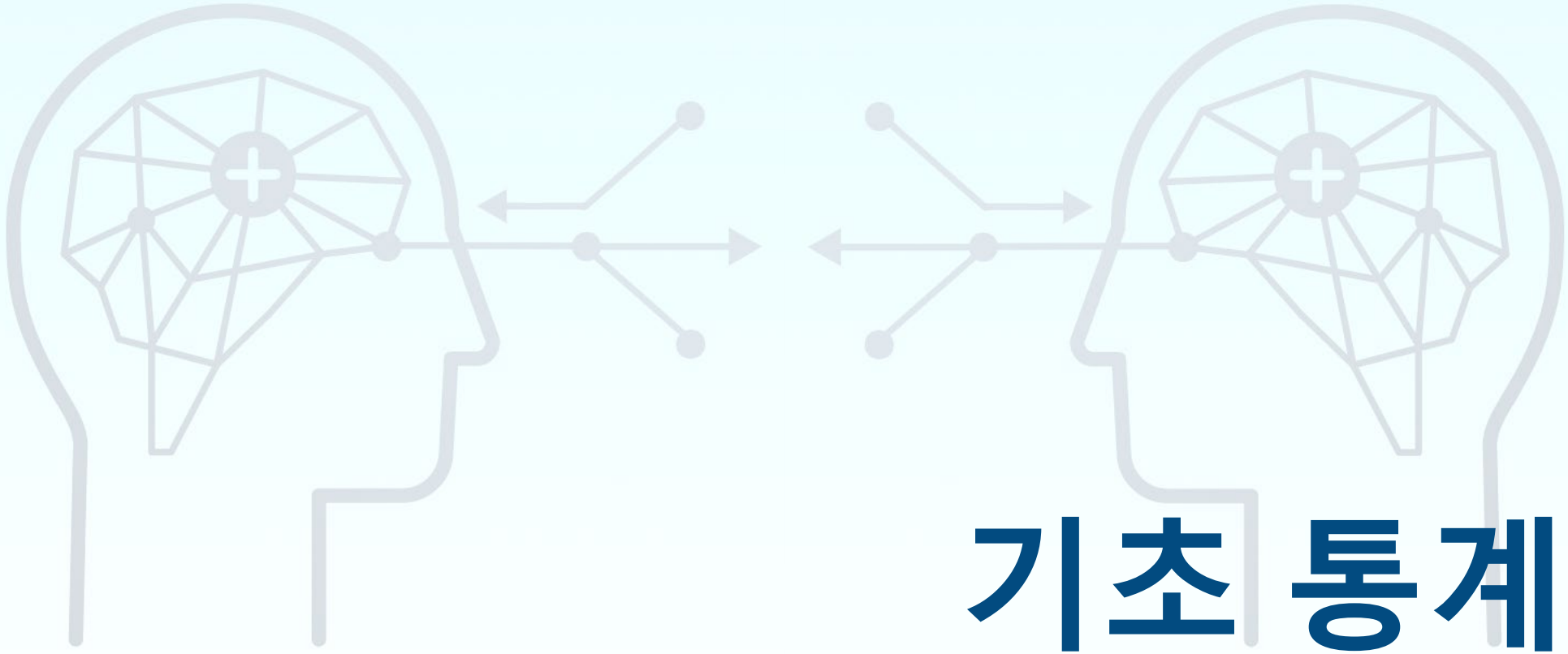
[한국폴리텍대학 성남캠퍼스 인공지능소프트웨어과]

JSP Servlet

2023. 5

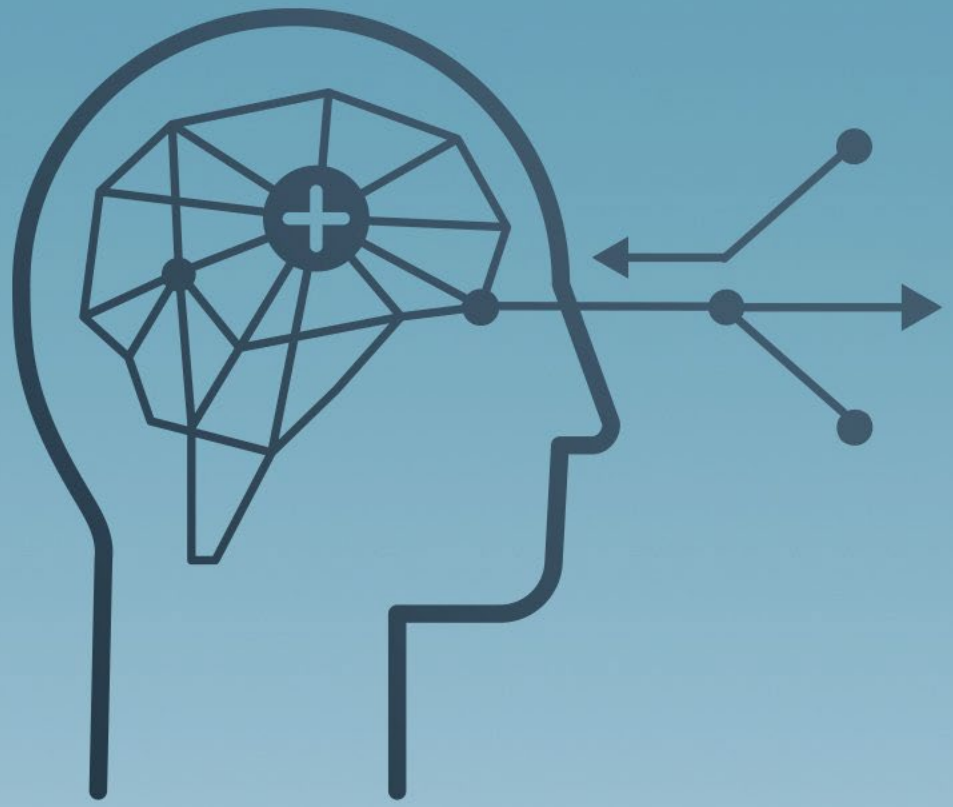
김 형 오



기초 통계

DEEP
LEARNING

MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS



DEEP
LEARNING

MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS

목차

A table of Contents

Part 1,

인터넷과 웹

참고문헌

한빛미디어, IT CookBook, 쉽게 배우는 JSP 웹 프로그래밍

TCP/IP

인터넷과 웹 프로그래밍>> 네트워크

TCP/IP

OSI 7계층	TCP/IP 4계층	
응용 계층	응용 계층	• 네트워크를 사용하는 WWW, FTP, 텔넷, SMTP 등의 응용 프로그램으로 구성.
표현 계층		
세션 계층	전송 계층	• 도착지까지 데이터를 전송 • 각각의 시스템을 연결 • TCP 프로토콜을 이용하여 데이터를 전송
전송 계층		
네트워크 계층	인터넷 계층	• 데이타를 정의 및 경로 지정 • 정확한 라우팅을 위해 IP 프로토콜을 사용 • IP 주소가 위치하는 계층
데이터 링크 계층		
물리 계층	물리 계층	• 물리적 계층 즉 이더넷 카드와 같은 하드웨어

IP주소

인터넷과 웹 프로그래밍>> 네트워크

IP 주소

- 네트워크에 연결된 컴퓨터를 구분하기 위해 사용
- 4개로 구분된 10진수를 사용함.
- 사설 IP는 NAT(Network Access Translator) 등을 이용해서 인터넷 접속 시 공인 IP로 매핑됨(일부 인터넷 서비스에 제약이 있을 수 있음)
- IP 주소 부족 문제를 해결하기 위해 IPV6가 논의됨.

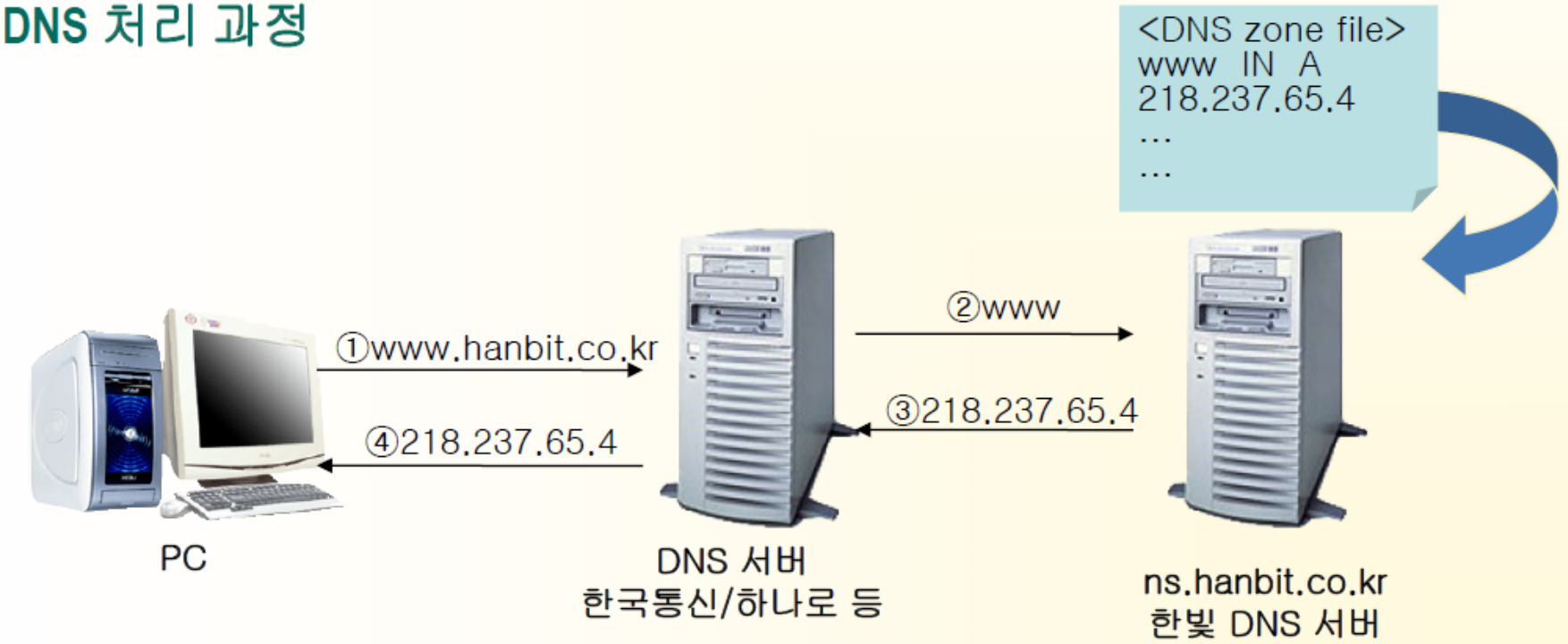
구분	범위	사용 목적
클래스 A	1.0.0.0~127.0.0.0	대형 통신망
클래스 B	128.0.0.0~191.255.0.0	중형 통신망, 주소 65536개 할당
클래스 C	192.0.0.0~223.255.255.0	소형 통신망, 주소 256개 할당
클래스 D	-	멀티 캐스트용으로 예약, 배포 중지
클래스 E	-	실험 목적, 배포 중지1

도메인 이름

도메인 이름

- IP 주소를 알기 쉬운 이름으로 바꾼 것
- DNS(Domain Name System) 서버가 필요함.

DNS 처리 과정



WWW

인터넷과 www

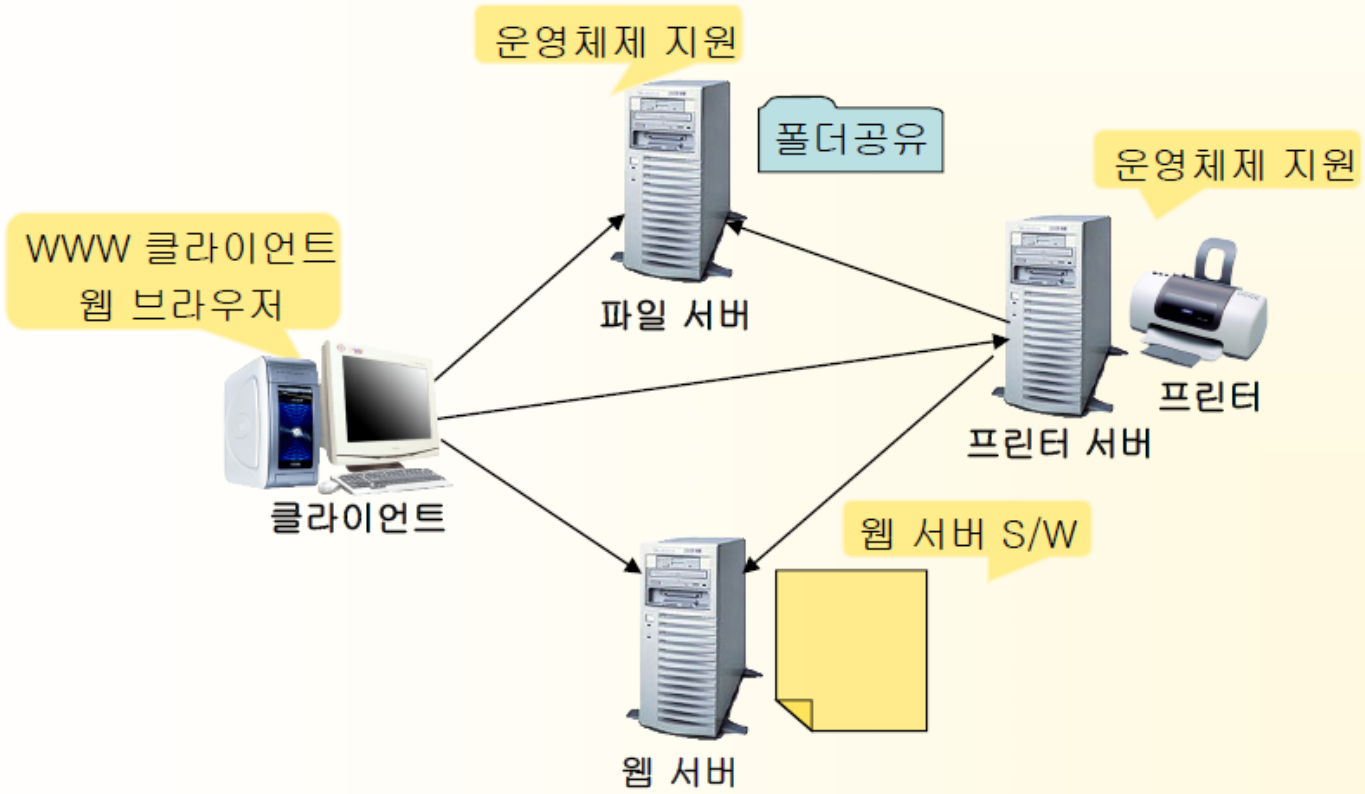
- 인터넷은 TCP/IP 기반의 네트워크가 전세계적으로 확대되어 하나로 연결된 '네트워크의 네트워크'
- 인터넷 = www가 아님. www는 인터넷 기반의 서비스 중 하나

이름	프로토콜	포트	기능
www	http	80	웹 서비스
Email	SMTP/POP3/IMAP	25/110/114	이메일 서비스
FTP	ftp	21	파일 전송 서비스
telnet	telnet	23	원격 로그인
DNS	DNS	83	도메인 이름 변환 서비스
News	NNTP	119	인터넷 뉴스 서비스

웹 서버와 클라이언트

웹 서버와 클라이언트

- 서버: 네트워크에서 서비스를 제공하는 컴퓨터
- 클라이언트: 네트워크에서 서비스를 제공받는 컴퓨터
  최근 클라이언트와 서버의 하드웨어적인 구분이 없어지고 있음

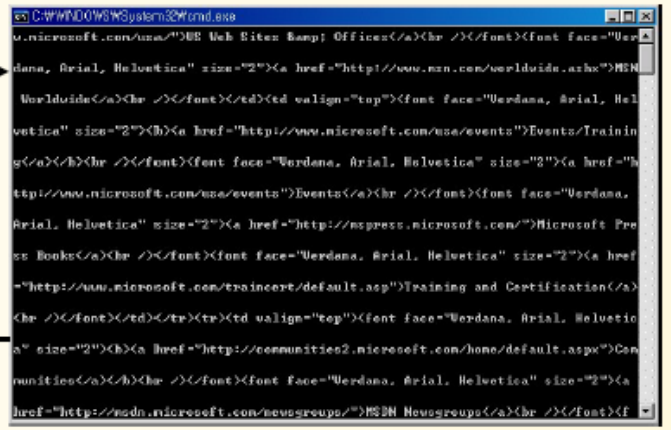


HTTP

HTTP(Hyper Text Tranfer Protocol)

- 프로토콜: 네트워크에 연결된 컴퓨터가 서로 통신(대화)하기 위한 규약
- HTTP는 www 서비스를 위한 통신 규약
- 웹 서버와 클라이언트는 HTTP를 이용해 통신
- HTTP 동작 원리

```
c:₩> telnet www.microsoft.com 80
....
GET /index.html
```



Part 1

클라이언트 스크립트

HTML과 클라이언트 스크립트 기술

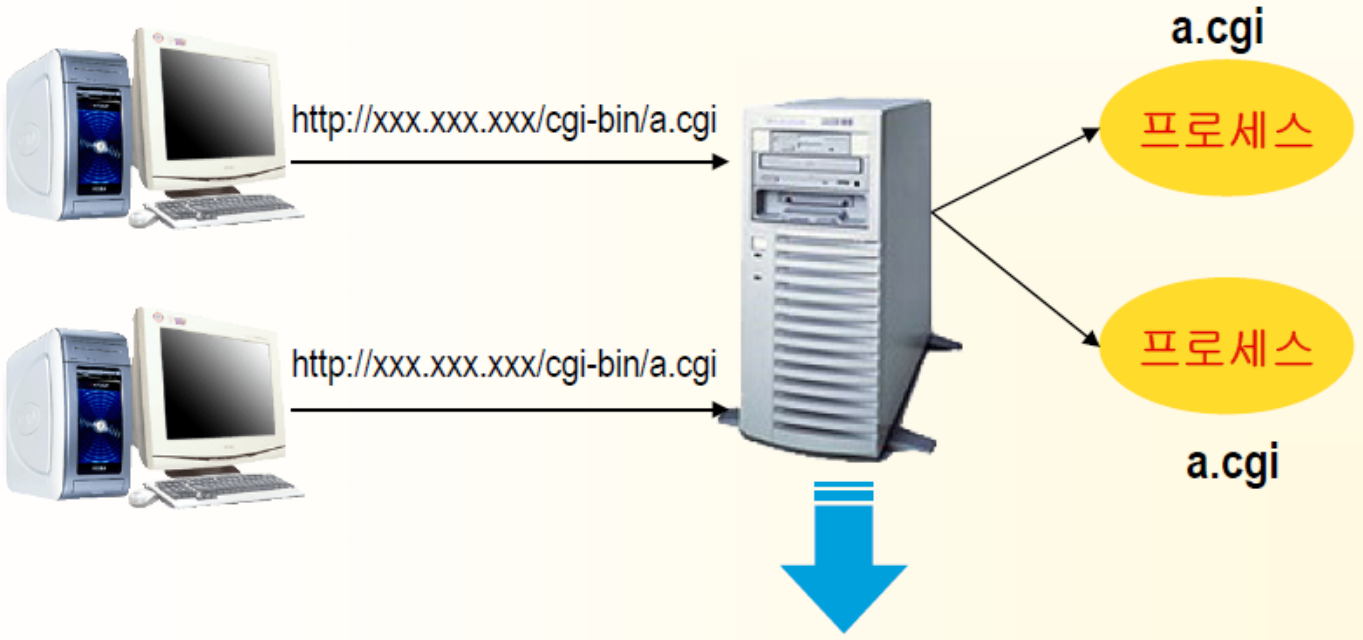
- HTML: www 서비스를 표현하기 위해 사용하는 언어
- www를 통해 서비스하는 모든 내용은 HTML로 표현되어야 함
- HTML은 텍스트 파일로 정적인 정보만 처리 가능
  → 동적으로 변하는 정보를 처리할 수 없음)
- 동적인 컨텐츠 처리하기 위해 CGI, Fast CGI, PHP, ASP, JSP 등의 기술이 사용

클라이언트 스크립트

- 자바스크립트가 대표적.
- 웹 브라우저가 스크립트 해석의 주체
- 웹 브라우저 핸들링은 가능하지만 서버 연동은 불가능

CGI(Common Gatewary Interface)

- 초기 웹 프로그래밍에 사용된 기술
- 프로세스 단위로 실행되기 때문에 사용자가 증가하면 급격히 성능 저하



사용자 증가에 따른
시스템 성능의 급격한 저하

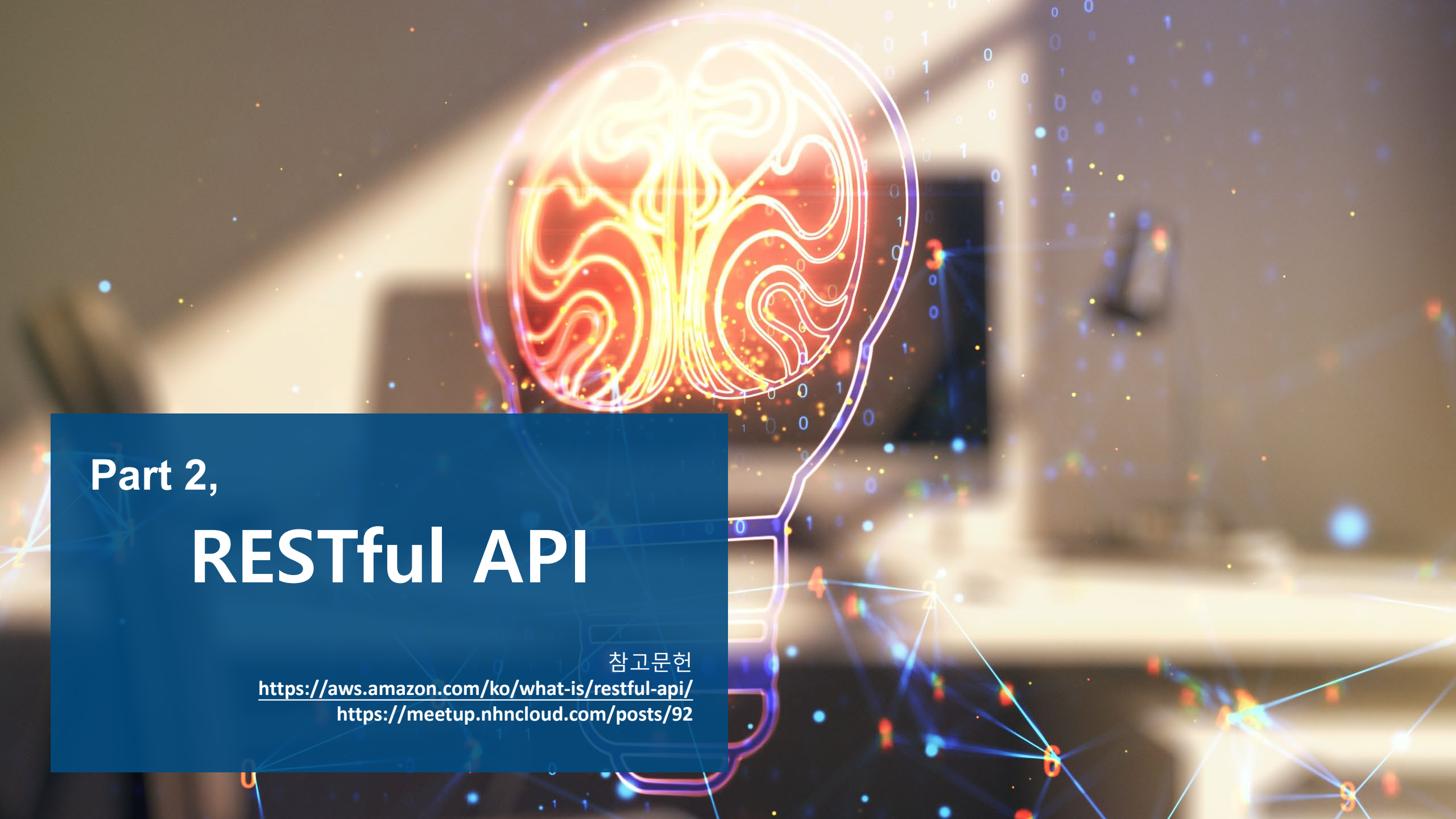
서버 스크립트

서버 스크립트 기술

- HTML과 스크립트 언어를 함께 사용할 수 있는 기술로 웹 서버에서 해석.
- 데이터베이스 연동 처리 등 다양한 구현이 가능
- 별도의 컴파일 과정없이 HTML 태그 수정 가능

서버 스크립트로 구현한 예(JSP)]

```
<%@ page contentType="text/html;charset=euc-kr" %>
<HTML>
<HEAD><TITLE>Hello World</TITLE></HEAD>
<BODY><H2>Hello World : 헬로월드</H2>
오늘의 날짜와 시간은 : <%= new java.util.Date() %>
➔ asp 의 경우 <%=now %>
</BODY>
</HTML>
```



Part 2,

RESTful API

참고문헌

https://aws.amazon.com/ko/what-is/restful-api/

https://meetup.nhncloud.com/posts/92

Part 2

RESTful API

RESTful API란 무엇인가요?

RESTful API는 두 컴퓨터 시스템이 인터넷을 통해 정보를 안전하게 교환하기 위해 사용하는 인터페이스입니다. 대부분의 비즈니스 애플리케이션은 다양한 태스크를 수행하기 위해 다른 내부 애플리케이션 및 서드 파티 애플리케이션과 통신해야 합니다. 예를 들어 월간 급여 명세서를 생성하려면 인보이스 발행을 자동화하고 내부의 근무 시간 기록 애플리케이션과 통신하기 위해 내부 계정 시스템이 데이터를 고객의 뱅킹 시스템과 공유해야 합니다. RESTful API는 안전하고 신뢰할 수 있으며 효율적인 소프트웨어 통신 표준을 따르므로 이러한 정보 교환을 지원합니다.

Part 2

API

API란 무엇인가요?

애플리케이션 프로그래밍 인터페이스(API)는 다른 소프트웨어 시스템과 통신하기 위해 따라야 하는 규칙을 정의합니다. 개발자는 다른 애플리케이션이 프로그래밍 방식으로 애플리케이션과 통신할 수 있도록 API를 표시하거나 생성합니다. 예를 들어, 근무 시간 기록 애플리케이션은 직원의 전체 이름과 날짜 범위를 요청하는 API를 표시합니다. 이 정보가 수신되면 내부적으로 직원의 근무 시간 기록을 처리하고 해당 날짜 범위에서 근무한 시간을 반환합니다.

웹 API는 클라이언트와 웹 리소스 사이의 게이트웨이라고 생각할 수 있습니다.

클라이언트

클라이언트는 웹에서 정보에 액세스하려는 사용자입니다. 클라이언트는 API를 사용하는 사람이거나 소프트웨어 시스템일 수 있습니다. 예를 들어 개발자는 날씨 시스템에서 날씨 데이터에 액세스하는 프로그램을 작성할 수 있습니다. 또는 사용자가 날씨 웹 사이트를 직접 방문할 때 브라우저에서 동일한 데이터에 액세스할 수 있습니다.

리소스

리소스는 다양한 애플리케이션이 클라이언트에게 제공하는 정보입니다. 리소스는 이미지, 동영상, 텍스트, 숫자 또는 모든 유형의 데이터일 수 있습니다. 클라이언트에 리소스를 제공하는 시스템을 서버라고도 합니다. 조직은 API를 사용하여 리소스를 공유하고 보안, 제어 및 인증을 유지하면서 웹 서비스를 제공합니다. 또한 API는 특정 내부 리소스에 액세스할 수 있는 클라이언트를 결정하는 데 도움이 됩니다.

REST

REST란 무엇인가요?

Representational State Transfer(REST)는 API 작동 방식에 대한 조건을 부과하는 소프트웨어 아키텍처입니다. REST는 처음에 인터넷과 같은 복잡한 네트워크에서 통신을 관리하기 위한 지침으로 만들어졌습니다. REST 기반 아키텍처를 사용하여 대규모의 고성능 통신을 안정적으로 지원할 수 있습니다. 쉽게 구현하고 수정할 수 있어 모든 API 시스템을 파악하고 여러 플랫폼에서 사용할 수 있습니다.

API 개발자는 여러 아키텍처를 사용하여 API를 설계할 수 있습니다. REST 아키텍처 스타일을 따르는 API를 REST API라고 합니다. REST 아키텍처를 구현하는 웹 서비스를 RESTful 웹 서비스라고 합니다. RESTful API라는 용어는 일반적으로 RESTful 웹 API를 나타냅니다. 하지만 REST API와 RESTful API라는 용어는 같은 의미로 사용할 수 있습니다.

REST는 Representational State Transfer라는 용어의 약자로서 2000년도에 로이 필딩 (Roy Fielding)의 박사학위 논문에서 최초로 소개되었습니다. 로이 필딩은 HTTP의 주요 저자 중 한 사람으로 그 당시 웹(HTTP) 설계의 우수성에 비해 제대로 사용되어지지 못하는 모습에 안타까워하며 웹의 장점을 최대한 활용할 수 있는 아키텍처로써 REST를 발표했다고 합니다.

Part 2

REST 아키텍쳐 스타일

균일한 인터페이스

균일한 인터페이스는 모든 RESTful 웹 서비스 디자인의 기본입니다. 이는 서버가 표준 형식으로 정보를 전송함을 나타냅니다. 형식이 지정된 리소스를 REST에서 표현이라고 부릅니다. 이 형식은 서버 애플리케이션에 있는 리소스의 내부 표현과 다를 수 있습니다. 예를 들어, 서버는 데이터를 텍스트로 저장하되, HTML 표현 형식으로 전송할 수 있습니다.

균일한 인터페이스에는 4가지 아키텍처 제약 조건이 있습니다.

1. 요청은 리소스를 식별해야 합니다. 이를 위해 균일한 리소스 식별자를 사용합니다.
2. 클라이언트는 원하는 경우 리소스를 수정하거나 삭제하기에 충분한 정보를 리소스 표현에서 가지고 있습니다. 서버는 리소스를 자세히 설명하는 메타데이터를 전송하여 이 조건을 충족합니다.
3. 클라이언트는 표현을 추가로 처리하는 방법에 대한 정보를 수신합니다. 이를 위해 서버는 클라이언트가 리소스를 적절하게 사용할 수 있는 방법에 대한 메타데이터가 포함된 명확한 메시지를 전송합니다.
4. 클라이언트는 작업을 완료하는 데 필요한 다른 모든 관련 리소스에 대한 정보를 수신합니다. 이를 위해 서버는 클라이언트가 더 많은 리소스를 동적으로 검색할 수 있도록 표현에 하이퍼링크를 넣어 전송합니다.

Part 2

REST 아키텍쳐 스타일

무상태

REST 아키텍처에서 무상태는 서버가 이전의 모든 요청과 독립적으로 모든 클라이언트 요청을 완료하는 통신 방법을 나타냅니다. 클라이언트는 임의의 순서로 리소스를 요청할 수 있으며 모든 요청은 무상태이거나 다른 요청과 분리됩니다. 이 REST API 설계 제약 조건은 서버가 매번 요청을 완전히 이해해서 이행할 수 있음을 의미합니다.

계층화 시스템

계층화된 시스템 아키텍처에서 클라이언트는 클라이언트와 서버 사이의 다른 승인된 중개자에게 연결할 수 있으며 여전히 서버로부터도 응답을 받습니다. 서버는 요청을 다른 서버로 전달할 수도 있습니다. 클라이언트 요청을 이행하기 위해 함께 작동하는 보안, 애플리케이션 및 비즈니스 로직과 같은 여러 계층으로 여러 서버에서 실행되도록 RESTful 웹 서비스를 설계할 수 있습니다. 이러한 계층은 클라이언트에 보이지 않는 상태로 유지됩니다.

Part 2

REST 아키텍처 스타일

캐시 가능성

RESTful 웹 서비스는 서버 응답 시간을 개선하기 위해 클라이언트 또는 중개자에 일부 응답을 저장하는 프로세스인 캐싱을 지원합니다. 예를 들어 모든 페이지에 공통 머리글 및 바닥글 이미지가 있는 웹 사이트를 방문한다고 가정해 보겠습니다. 새로운 웹 사이트 페이지를 방문할 때마다 서버는 동일한 이미지를 다시 전송해야 합니다. 이를 피하기 위해 클라이언트는 첫 번째 응답 후에 해당 이미지를 캐싱하거나 저장한 다음 캐시에서 직접 이미지를 사용합니다. RESTful 웹 서비스는 캐시 가능 또는 캐시 불가능으로 정의되는 API 응답을 사용하여 캐싱을 제어합니다.

온디맨드 코드

REST 아키텍처 스타일에서 서버는 소프트웨어 프로그래밍 코드를 클라이언트에 전송하여 클라이언트 기능을 일시적으로 확장하거나 사용자 지정할 수 있습니다. 예를 들어, 웹 사이트에서 등록 양식을 작성하면 브라우저는 잘못된 전화번호와 같은 실수를 즉시 강조 표시합니다. 서버에서 전송한 코드로 인해 이 작업을 수행할 수 있습니다.

Part 2

RESTful API 이점

확장성

REST API를 구현하는 시스템은 REST가 클라이언트-서버 상호 작용을 최적화하기 때문에 효율적으로 크기 조정할 수 있습니다. 무상태는 서버가 과거 클라이언트 요청 정보를 유지할 필요가 없기 때문에 서버 로드를 제거합니다. 잘 관리된 캐싱은 일부 클라이언트-서버 상호 작용을 부분적으로 또는 완전히 제거합니다. 이러한 모든 기능은 성능을 저하시키는 통신 병목 현상을 일으키지 않으면서 확장성을 지원합니다.

유연성

RESTful 웹 서비스는 완전한 클라이언트-서버 분리를 지원합니다. 각 부분이 독립적으로 발전할 수 있도록 다양한 서버 구성 요소를 단순화하고 분리합니다. 서버 애플리케이션의 플랫폼 또는 기술 변경은 클라이언트 애플리케이션에 영향을 주지 않습니다. 애플리케이션 함수를 계층화하는 기능은 유연성을 더욱 향상시킵니다. 예를 들어, 개발자는 애플리케이션 로직을 다시 작성하지 않고도 데이터베이스 계층을 변경할 수 있습니다.

독립성

REST API는 사용되는 기술과 독립적입니다. API 설계에 영향을 주지 않고 다양한 프로그래밍 언어로 클라이언트 및 서버 애플리케이션을 모두 작성할 수 있습니다. 또한 통신에 영향을 주지 않고 양쪽의 기본 기술을 변경할 수 있습니다.

Part 2

RESTful API 동작 방식

RESTful API의 기본 기능은 인터넷 브라우징과 동일합니다. 클라이언트는 리소스가 필요할 때 API를 사용하여 서버에 접속합니다. API 개발자는 서버 애플리케이션 API 문서에서 클라이언트가 REST API를 어떻게 사용해야 하는지 설명합니다. 다음은 모든 REST API 호출에 대한 일반 단계입니다.

1. 클라이언트가 서버에 요청을 전송합니다. 클라이언트가 API 문서에 따라 서버가 이해하는 방식으로 요청 형식을 지정합니다.
2. 서버가 클라이언트를 인증하고 해당 요청을 수행할 수 있는 권한이 클라이언트에 있는지 확인합니다.
3. 서버가 요청을 수신하고 내부적으로 처리합니다.
4. 서버가 클라이언트에 응답을 반환합니다. 응답에는 요청이 성공했는지 여부를 클라이언트에 알려주는 정보가 포함됩니다. 응답에는 클라이언트가 요청한 모든 정보도 포함됩니다.

REST API 요청 및 응답 세부 정보는 API 개발자가 API를 설계하는 방식에 따라 약간씩 다릅니다.

Part 2

RESTful API 구성요소

- **자원(RESOURCE)** - URI
- **행위(Verb)** - HTTP METHOD
- **표현(Representations)**

고유 리소스 식별자

서버는 고유한 리소스 식별자로 각 리소스를 식별합니다. REST 서비스의 경우 서버는 일반적으로 URL(Uniform Resource Locator)을 사용하여 리소스 식별을 수행합니다. URL은 리소스에 대한 경로를 지정합니다. URL은 웹페이지를 방문하기 위해 브라우저에 입력하는 웹 사이트 주소와 유사합니다. URL은 요청 엔드포인트라고도 하며 클라이언트가 요구하는 사항을 서버에 명확하게 지정합니다.

Part 2

RESTful API 구성요소

GET

클라이언트는 GET을 사용하여 서버의 지정된 URL에 있는 리소스에 액세스합니다. GET 요청을 캐싱하고 RESTful API 요청에 파라미터를 넣어 전송하여 전송 전에 데이터를 필터링하도록 서버에 지시할 수 있습니다.

POST

클라이언트는 POST를 사용하여 서버에 데이터를 전송합니다. 여기에는 요청과 함께 데이터 표현이 포함됩니다. 동일한 POST 요청을 여러 번 전송하면 동일한 리소스를 여러 번 생성하는 부작용이 있습니다.

PUT

클라이언트는 PUT을 사용하여 서버의 기존 리소스를 업데이트합니다. POST와 달리, RESTful 웹 서비스에서 동일한 PUT 요청을 여러 번 전송해도 결과는 동일합니다.

DELETE

클라이언트는 DELETE 요청을 사용하여 리소스를 제거합니다. DELETE 요청은 서버 상태를 변경할 수 있습니다. 하지만 사용자에게 적절한 인증이 없으면 요청은 실패합니다.

Part 2

RESTful API 디자인 가이드

[참고]HTTP METHOD의 알맞은 역할

POST, GET, PUT, DELETE 이 4가지의 Method를 가지고 CRUD를 할 수 있습니다.

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

1) URI는 정보의 자원을 표현해야 한다. (리소스명은 동사보다는 명사를 사용)

```
GET /members/delete/1
```

위와 같은 방식은 REST를 제대로 적용하지 않은 URI입니다. URI는 자원을 표현하는데 중점을 두어야 합니다. delete와 같은 행위에 대한 표현이 들어가서는 안됩니다.

RESTful API 디자인 가이드

2) 자원에 대한 행위는 HTTP Method(GET, POST, PUT, DELETE 등)로 표현
위의 잘못 된 URI를 HTTP Method를 통해 수정해 보면

```
DELETE /members/1
```

으로 수정할 수 있겠습니다.
회원정보를 가져올 때는 GET, 회원 추가 시의 행위를 표현하고자 할 때는 POST METHOD를 사용하여 표현합니다.

회원정보를 가져오는 URI

```
GET /members/show/1     (x)
GET /members/1          (o)
```

회원을 추가할 때

```
GET /members/insert/2 (x)  - GET 메서드는 리소스 생성에 맞지 않습니다.
POST /members/2       (o)
```

RESTful API 디자인 가이드

4-2. URI 설계 시 주의할 점

1) 슬래시 구분자(/)는 계층 관계를 나타내는 데 사용

```
http://restapi.example.com/houses/apartments
http://restapi.example.com/animals/mammals/whales
```

2) URI 마지막 문자로 슬래시(/)를 포함하지 않는다.

URI에 포함되는 모든 글자는 리소스의 유일한 식별자로 사용되어야 하며 URI가 다르다는 것은 리소스가 다르다는 것이고, 역으로 리소스가 다르면 URI도 달라져야 합니다. REST API는 분명한 URI를 만들어 통신을 해야 하기 때문에 혼동을 주지 않도록 URI 경로의 마지막에는 슬래시(/)를 사용하지 않습니다.

```
http://restapi.example.com/houses/apartments/ (X)
http://restapi.example.com/houses/apartments  (0)
```

RESTful API 디자인 가이드

3) 하이픈(-)은 URI 가독성을 높이는데 사용

URI를 쉽게 읽고 해석하기 위해, 불가피하게 긴 URI경로를 사용하게 된다면 하이픈을 사용해 가독성을 높일 수 있습니다.

4) 밑줄(_)은 URI에 사용하지 않는다.

글꼴에 따라 다르긴 하지만 밑줄은 보기 어렵거나 밑줄 때문에 문자가 가려지기도 합니다. 이런 문제를 피하기 위해 밑줄 대신 하이픈(-)을 사용하는 것이 좋습니다.(가독성)

5) URI 경로에는 소문자가 적합하다.

URI 경로에 대문자 사용은 피하도록 해야 합니다. 대소문자에 따라 다른 리소스로 인식하게 되기 때문입니다. RFC 3986(URI 문법 형식)은 URI 스키마와 호스트를 제외하고는 대소문자를 구별하도록 규정하기 때문이지요.

```
RFC 3986 is the URI (Unified Resource Identifier) Syntax document
```

RESTful API 디자인 가이드

6) 파일 확장자는 URI에 포함시키지 않는다.

```
http://restapi.example.com/members/soccer/345/photo.jpg (X)
```

REST API에서는 메시지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI 안에 포함시키지 않습니다. Accept header를 사용하도록 합시다.

```
GET / members/soccer/345/photo HTTP/1.1 Host: restapi.example.com Accept: image/jpg
```

Part 2

RESTful API 구성요소

HTTP 헤더

요청 헤더는 클라이언트와 서버 간에 교환되는 메타데이터입니다. 예를 들어, 요청 헤더는 요청 및 응답의 형식을 나타내고 요청 상태 등에 대한 정보를 제공합니다.

데이터

REST API 요청에는 POST, PUT 및 기타 HTTP 메서드가 성공적으로 작동하기 위한 데이터가 포함될 수 있습니다.

파라미터

RESTful API 요청에는 수행해야 할 작업에 대한 자세한 정보를 서버에 제공하는 파라미터가 포함될 수 있습니다. 다음은 몇 가지 파라미터 유형입니다.

- URL 세부정보를 지정하는 경로 파라미터.
- 리소스에 대한 추가 정보를 요청하는 쿼리 파라미터.
- 클라이언트를 빠르게 인증하는 쿠키 파라미터.

Part 2

RESTful API 서버응답

상태 표시줄

상태 표시줄에는 요청 성공 또는 실패를 알리는 3자리 상태 코드가 있습니다. 예를 들어, 2XX 코드는 성공을 나타내고 4XX 및 5XX 코드는 오류를 나타냅니다. 3XX 코드는 URL 리디렉션을 나타냅니다.

다음은 몇 가지 일반적인 상태 코드입니다.

- 200: 일반 성공 응답
- 201: POST 메서드 성공 응답
- 400: 서버가 처리할 수 없는 잘못된 요청
- 404: 리소스를 찾을 수 없음

Part 2

RESTful API 서버응답

상태코드	
200	클라이언트의 요청을 정상적으로 수행함
201	클라이언트가 어떠한 리소스 생성을 요청, 해당 리소스가 성공적으로 생성됨(POST를 통한 리소스 생성 작업 시)

상태코드	
400	클라이언트의 요청이 부적절 할 경우 사용하는 응답 코드
401	클라이언트가 인증되지 않은 상태에서 보호된 리소스를 요청했을 때 사용하는 응답 코드
	(로그인 하지 않은 유저가 로그인 했을 때, 요청 가능한 리소스를 요청했을 때)
403	유저 인증상태와 관계 없이 응답하고 싶지 않은 리소스를 클라이언트가 요청했을 때 사용하는 응답 코드
	(403 보다는 400이나 404를 사용할 것을 권고. 403 자체가 리소스가 존재한다는 뜻이기 때문에)
405	클라이언트가 요청한 리소스에서는 사용 불가능한 Method를 이용했을 경우 사용하는 응답 코드

Part 2

RESTful API 서버응답

상태코드	
301	클라이언트가 요청한 리소스에 대한 URI가 변경 되었을 때 사용하는 응답 코드
	(응답 시 Location header에 변경된 URI를 적어 줘야 합니다.)
500	서버에 문제가 있을 경우 사용하는 응답 코드

Part 2

RESTful API 서버응답

메시지 본문

응답 본문에는 리소스 표현이 포함됩니다. 서버는 요청 헤더에 포함된 내용을 기반으로 적절한 표현 형식을 선택합니다. 클라이언트는 데이터 작성 방식을 일반 텍스트로 정의하는 XML 또는 JSON 형식으로 정보를 요청할 수 있습니다. 예를 들어, 클라이언트가 John이라는 사람의 이름과 나이를 요청하면 서버는 다음과 같이 JSON 표현을 반환합니다.

'{"name":"John", "age":30}'

헤더

응답에는 응답에 대한 헤더 또는 메타데이터도 포함됩니다. 이는 응답에 대한 추가 컨텍스트를 제공하고 서버, 인코딩, 날짜 및 콘텐츠 유형과 같은 정보를 포함합니다.

Part 2

RESTful API 서버응답

메시지 본문

응답 본문에는 리소스 표현이 포함됩니다. 서버는 요청 헤더에 포함된 내용을 기반으로 적절한 표현 형식을 선택합니다. 클라이언트는 데이터 작성 방식을 일반 텍스트로 정의하는 XML 또는 JSON 형식으로 정보를 요청할 수 있습니다. 예를 들어, 클라이언트가 John이라는 사람의 이름과 나이를 요청하면 서버는 다음과 같이 JSON 표현을 반환합니다.

'{"name":"John", "age":30}'

헤더

응답에는 응답에 대한 헤더 또는 메타데이터도 포함됩니다. 이는 응답에 대한 추가 컨텍스트를 제공하고 서버, 인코딩, 날짜 및 콘텐츠 유형과 같은 정보를 포함합니다.

Part 3,

자바와 서블릿

참고문헌

한빛미디어, IT CookBook, 쉽게 배우는 JSP 웹 프로그래밍

자바와 서블릿

자바와 서블릿

- 썬마이크로시스템즈에서 개발한 객체 지향 언어
- 운영체제와 하드웨어의 독립적(휴대폰에서 매킨토시까지)

서블릿

- 서블릿은 자바 언어로 웹 프로그래밍 하기 위해 개발된 기술
- 자바의 모든 기능 사용 가능
- 멀티스레드 방식의 서버 운영으로 인해 빠른 처리 속도 보장

자바와 서블릿

JSP

- JSP는 Java Server Page 약자
- 썬에서 만든 자바 서블릿 기반의 서버측 스크립트 기술

자바와 서블릿

JSP 특징

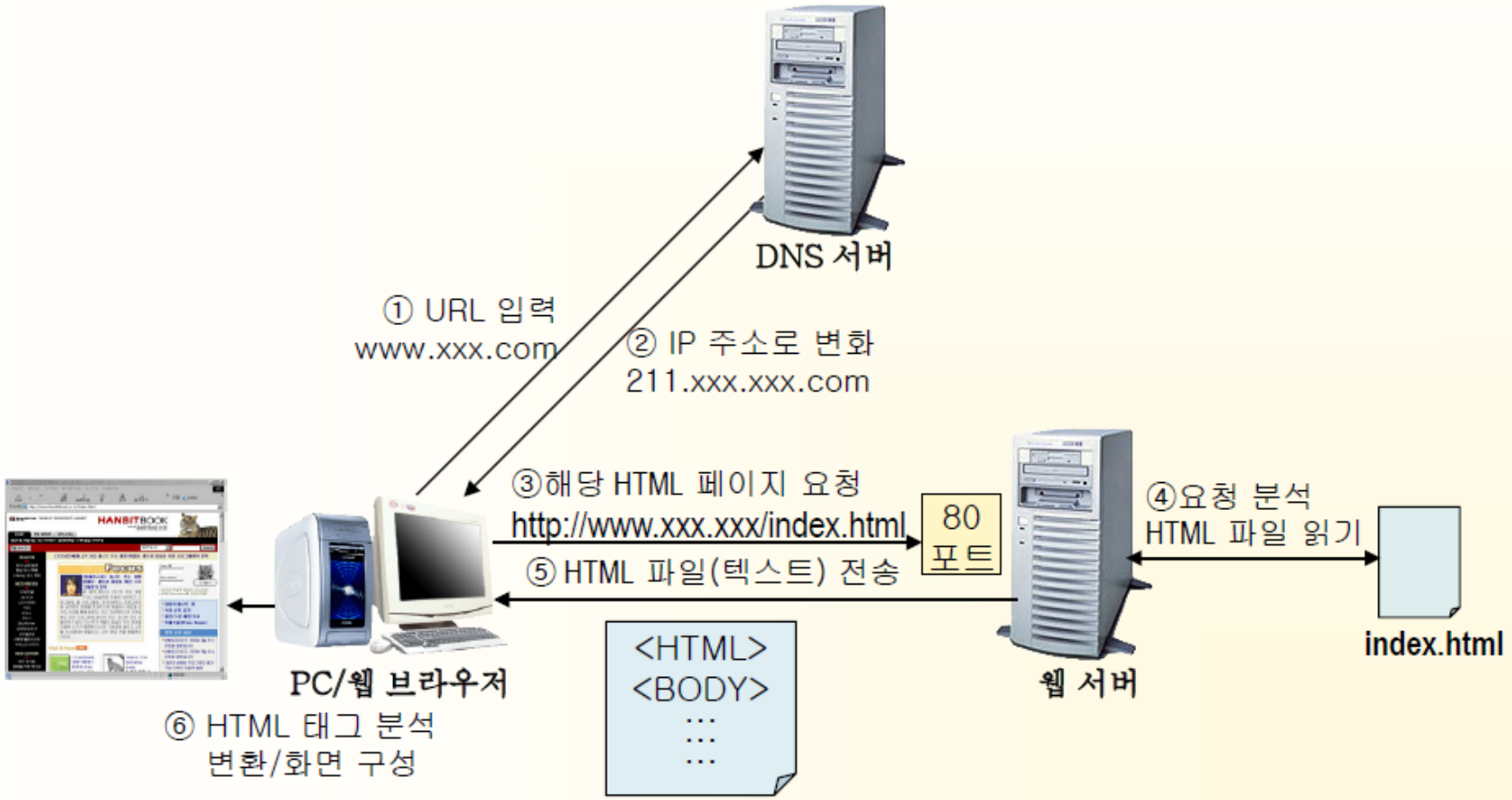
- 빈즈라고 하는 자바 컴포넌트를 사용할 수 있다.
- 최초의 서블릿으로 컴파일 된 후에는 메모리에서 처리

  → 많은 사용자 접속도 원활히 처리할 수 있다.
- JSP나 다른 서블릿 간의 쉬운 데이터 공유

  → page, request, session, application scope 으로 가능
- 자바의 모든 기능을 사용할 수 있다. → 무한한 확장성
- IBM, 오라클, 썬, BEA 등에서 강력히 지원
- 사용자 태그를 만들어 사용할 수 있다.
- JSTL(JSP Standard Tag Library)과 같은 다양한 기능의 태그 라이브러리 이용 가능
- 다양한 운영체제와 여러 회사의 JSP 개발/실행 환경을 이용할 수 있다.

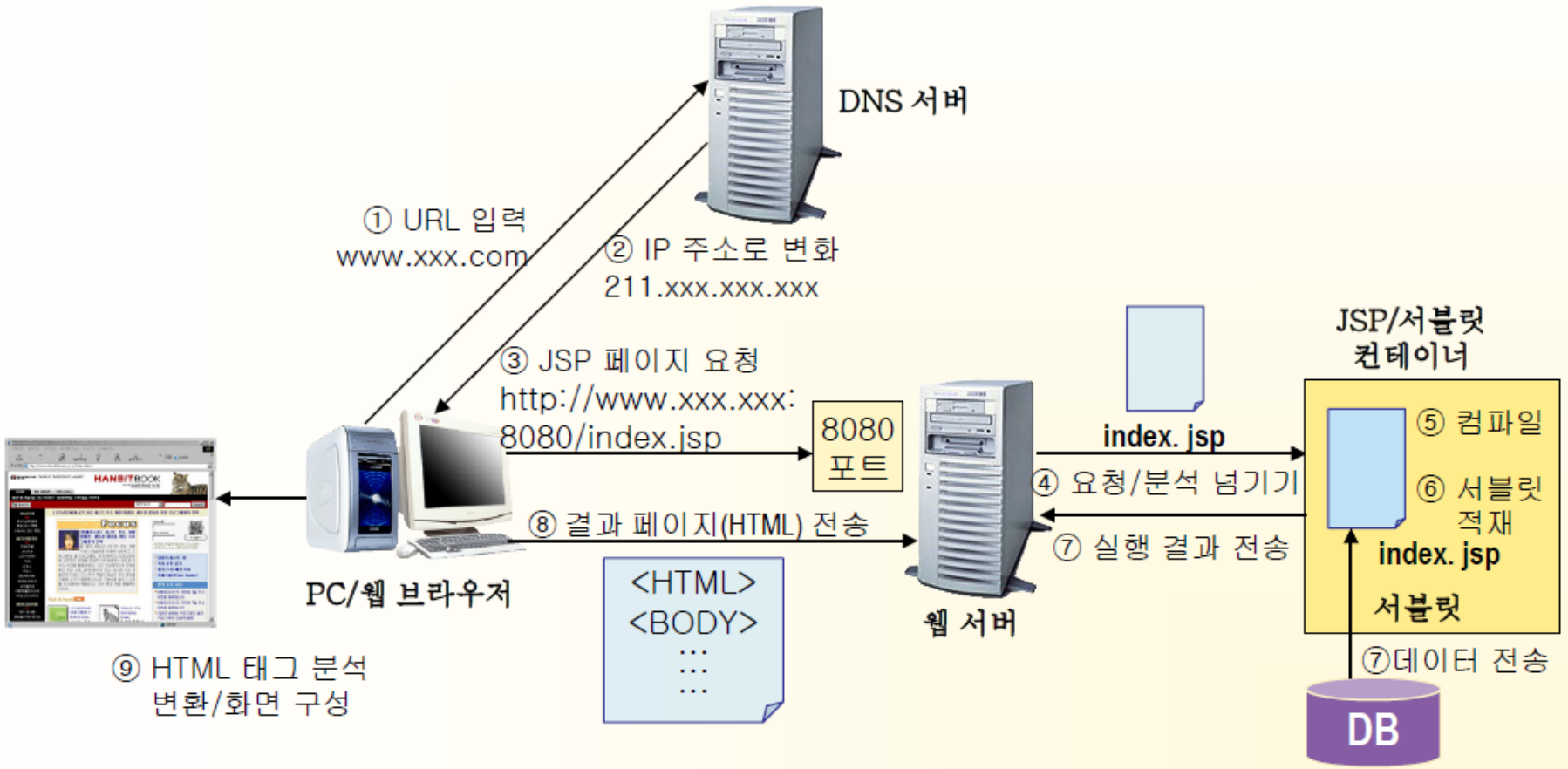
www 서비스

일반적인 www 서비스의 동작 과정



WWW 서비스 – JSP연동

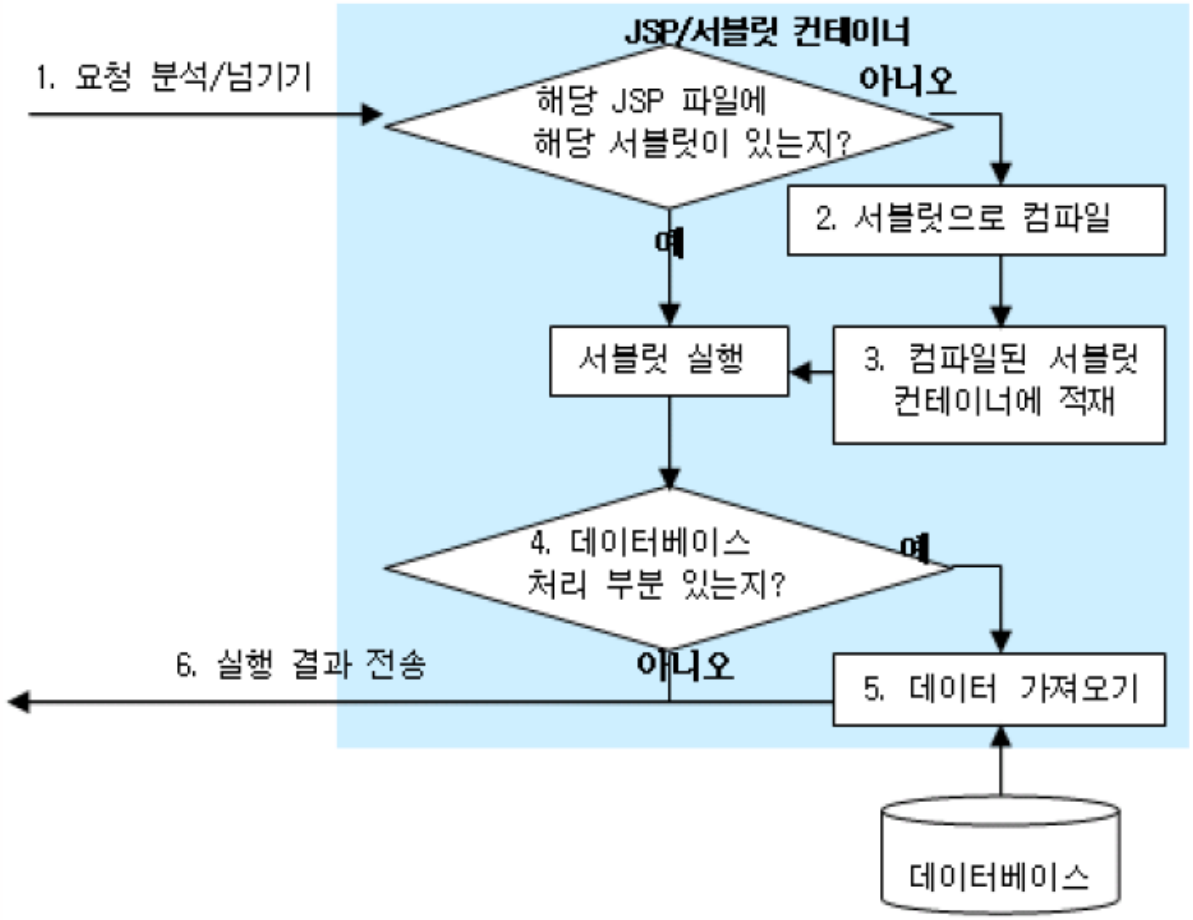
JSP 동작 과정



Part 3

JSP 서블릿 처리과정

JSP 서블릿 컴파일과 처리 과정



JSP를 위한 스킬

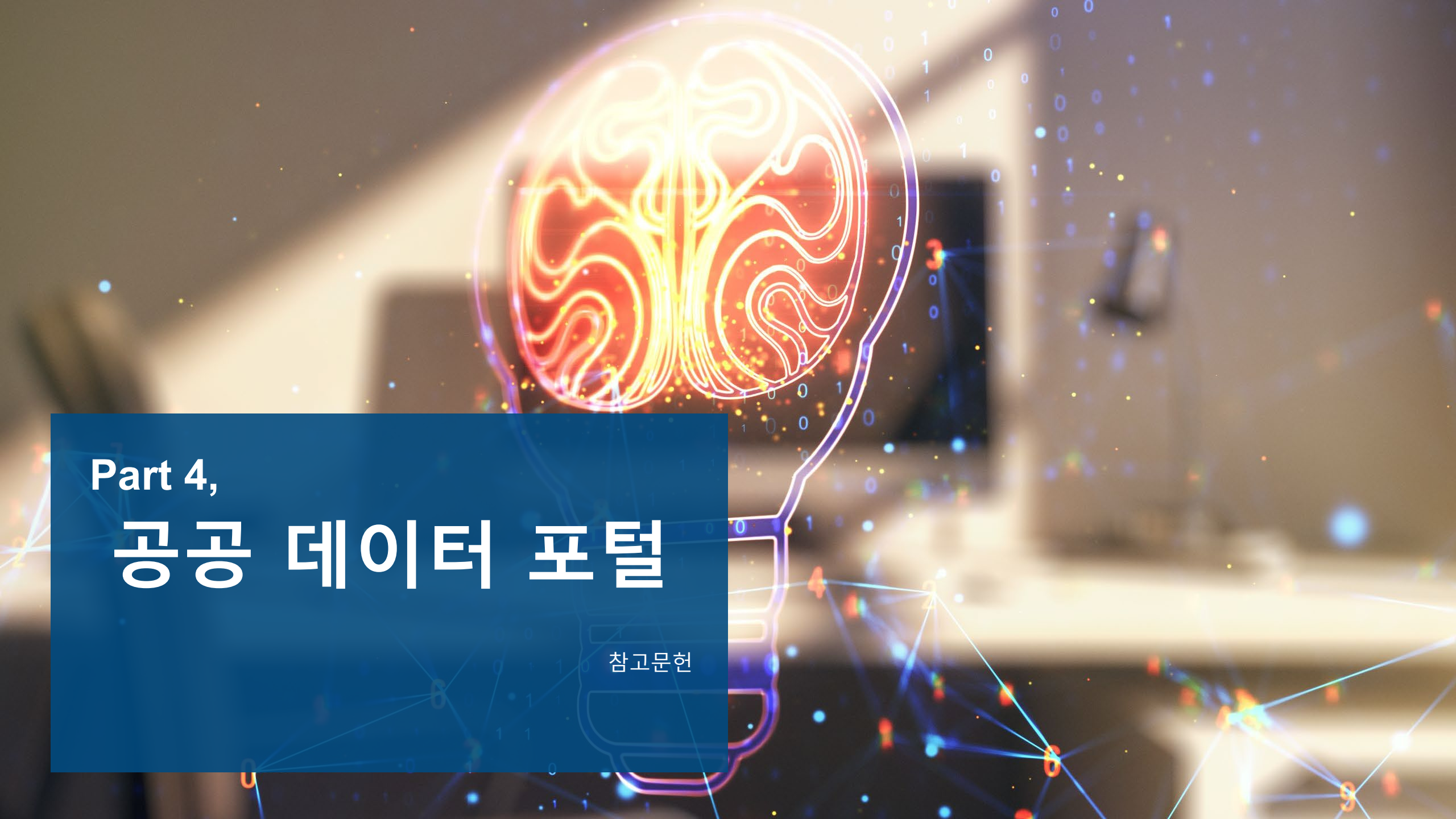
JSP 학습을 위한 필수 기술

필수 기술	프로그램 경험	비 고
자바	자바 언어 기본, 객체지향 개념 상속, 오버로딩, 오버라이딩 인터페이스 구현 java.util, java.io 패키지 쓰레드, 예외 핸들링	패키지와 클래스 이해 클래스 DOC을 참조하여 프로그래밍이 가능한 수준 JDK 설치 및 환경설정
JDBC	JDBC 드라이버 세팅 ResultSet, PreparedStatement 데이터 핸들링	오라클, MySQL 등 원격지 데 이터베이스 연결 처리 경험
서블릿	서블릿 기초 프로그래밍 request, response 처리 GET/POST 처리	서블릿 생명주기 이해

JSP를 위한 스킬

JSP를 배우는데 도움이 되는 기술

관련 기술	프로그램 경험	최소 요구사항
HTML	HTML 기초 태그 사용 FORM 관련 태그 사용	수작업으로 코딩이 가능한 수준. CSS, 레이어 이해
자바스크립트	메서드 만들기 FORM 연계 이벤트 처리	필요한 기능을 함수로 구현 가능한 수준.
데이터베이스	기본 SQL 문의 사용 데이터베이스 연계 프로그래밍 경험	테이블 생성과 키에 대한 이해 키 관계 설정
웹 프로그래밍	웹 서버 세팅 CGI, ASP, PHP등 웹 프로그래밍 경험	유닉스에서 웹 서버 세팅 경험



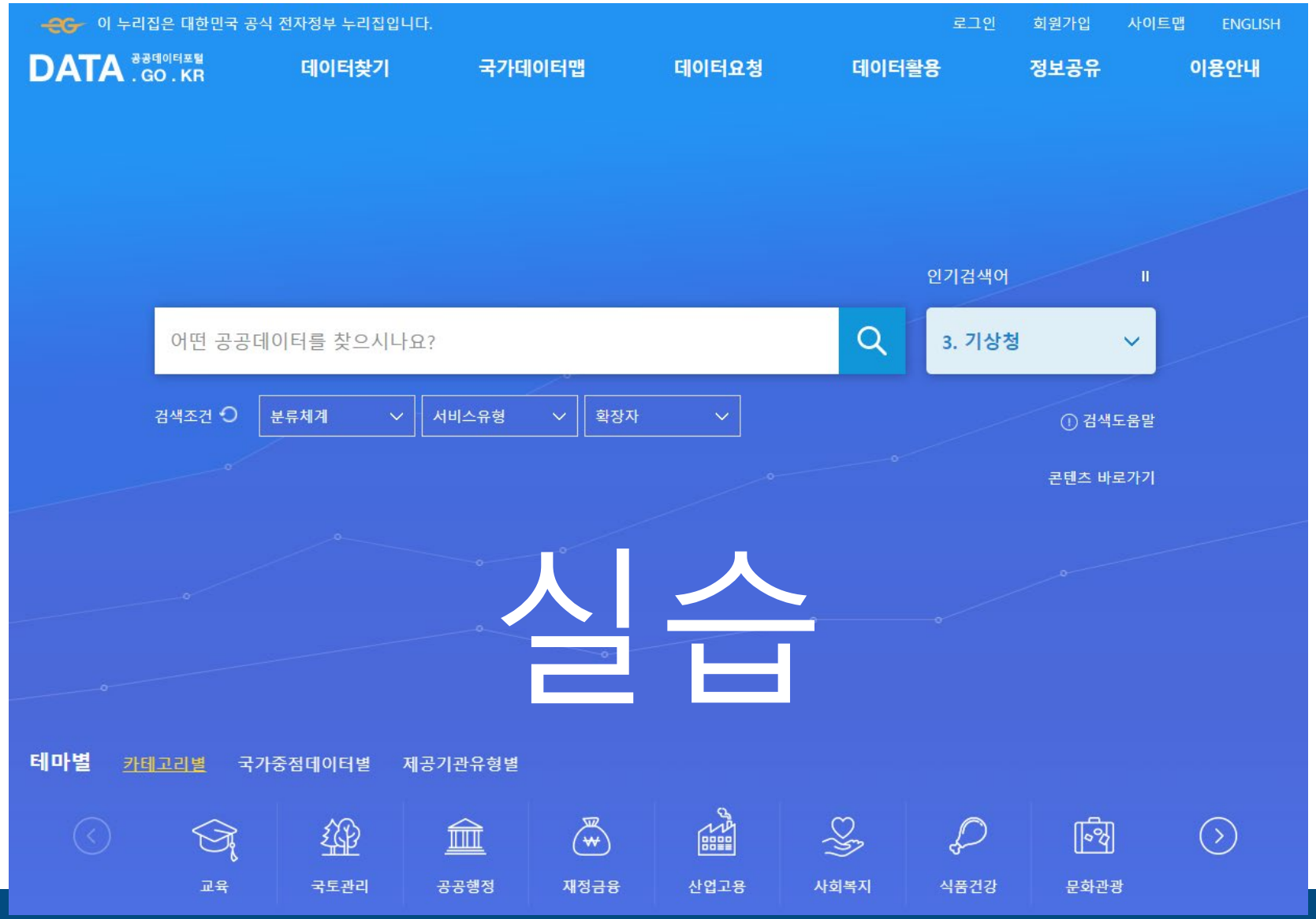
Part 4,

공공 데이터 포털

참고문헌

Part 4

공공 데이터 포털



경청해주셔서 감사합니다.