

Spring


RESTful Web Service with Spring MVC

REST

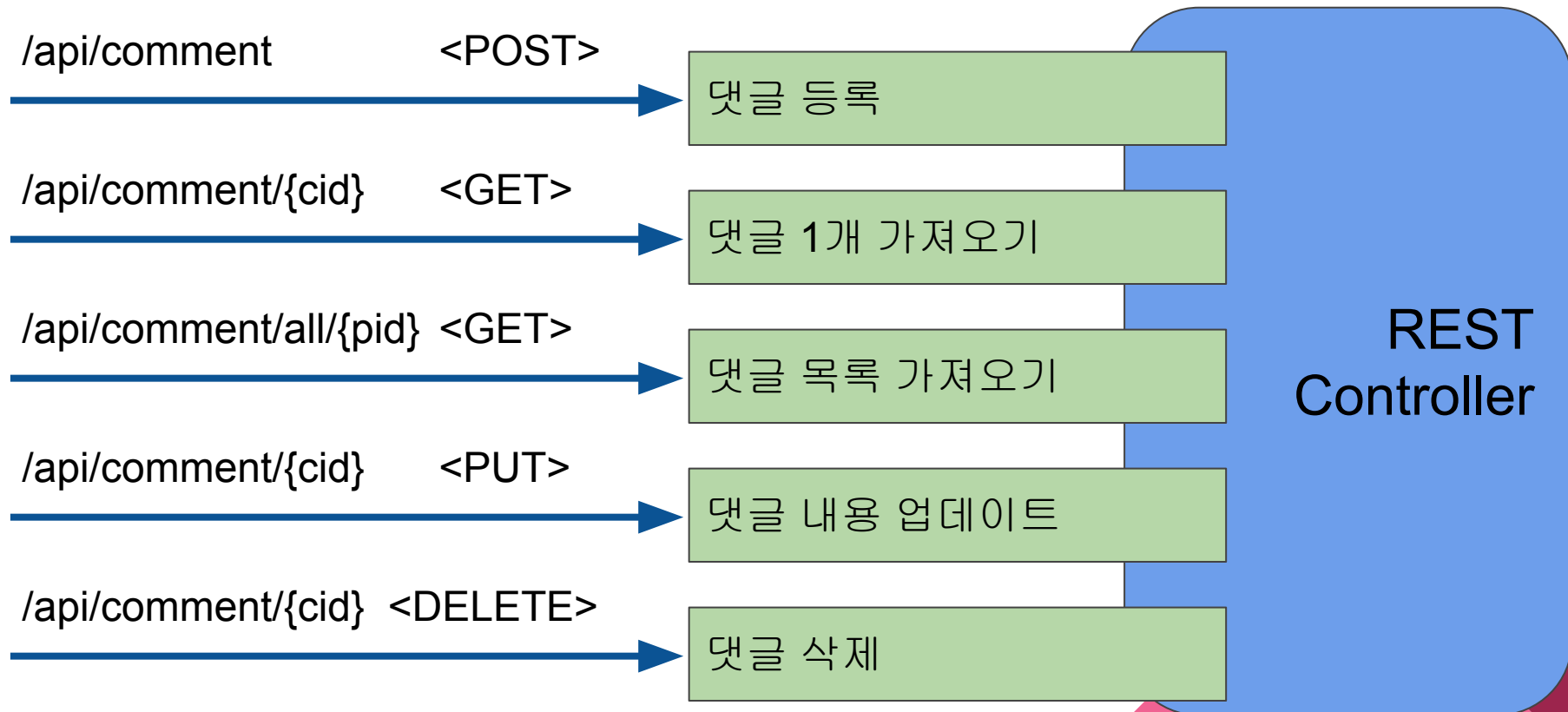
- REST: REpresentational State Transfer

- 하나의 URI는 하나의 고유한 Resource를 대표하도록 설계
- 특정 URI(Uniform Resource Identifier)는 그에 상응하는 데이터를 의미 → 데이터에 대한 처리는 HTTP 방식(GET, POST, PUT, DELETE 등)과 같은 추가적인 정보를 통해서 결정
- REST 서비스: 특정 URI를 통해서 사용자가 원하는 정보를 제공하는 방식
 - RESTful:
 - RESTful API: REST 방식으로 제공되는 외부 연결 URI

- REST와 HTTP methods

- GET: 리소스 또는 리소스 컬렉션을 얻어옴
 - POST: 리소스 생성
 - PUT: 리소스 업데이트
 - DELETE: 리소스 삭제
- 

REST: RequestMapping & RequestMethod



Spring REST Annotation

- **@ResponseBody**

- 메소드 또는 메소드의 리턴 타입에 사용하는 어노테이션
 - @Controller 어노테이션이 있는 클래스에서 정의된 메소드 (리턴타입)에 @ResponseBody 어노테이션을 사용하면, 클라이언트로 JSP가 아닌 데이터 자체가 서비스됨
- Spring MVC의 기본 처리 방식(Controller → JSP)과 달리, 메소드가 리턴하는 데이터를 스프링의 MessageConverter가 가공해서 클라이언트(브라우저)에게 전달
- Spring 3 버전부터 지원

- **@RestController**

- Controller 클래스에 사용되는 어노테이션
- JSP와 같은 View를 만들어 내는 것이 목적이 아닌, REST 방식의 데이터 처리를 위한 컨트롤러 클래스임을 선언하는 어노테이션
- @RestController의 모든 컨트롤러 메소드들은, @ResponseBody 어노테이션 없이, 뷰가 아닌 데이터 자체를 클라이언트(브라우저)에게 서비스(리턴)하는 메소드가 됨

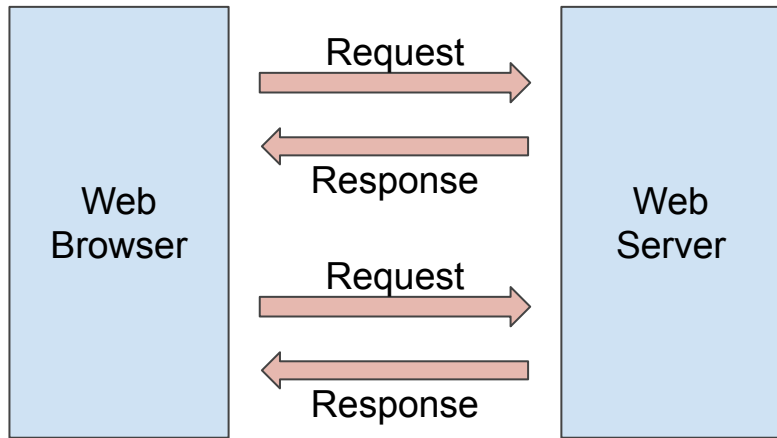
REST & Ajax

- Ajax(Asynchronous JavaScript And XML)
 - 요청(request)과 응답(response)가 비동기적(asynchronous)으로 이루어지는 통신 방식
 - 클라이언트(브라우저)는 서버에게 요청을 보내고, 응답을 기다리지 않고 다른 작업을 처리함. 응답을 받았을 때 그에 대한 처리(화면 업데이트 등)을 함.
- REST(Representational State Transfer)
 - HTTP URL과 방식(method: GET, POST 등)을 통해서 서버로부터 데이터를 전송받을 수 있는 방식
- Web Application 구현
 - Ajax를 이용하여 REST API를 호출
 - REST API의 결과 데이터를 브라우저가 처리해서 화면 업데이트



HTTP Client-Server 통신

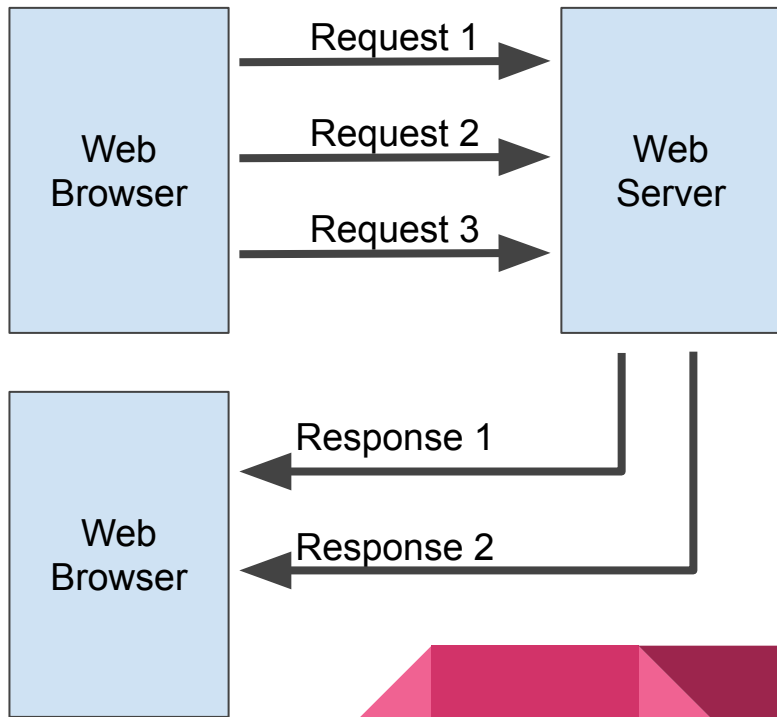
1. 웹 브라우저는 웹 서버에게 요청 (request)를 전송
2. 웹 서버는 클라이언트의 요청을 처리 후 응답(response)를 클라이언트에게 전송
3. 웹 브라우저는 서버의 응답 (response)을 받을 때까지 대기하다가 응답을 받은 후 화면을 업데이트



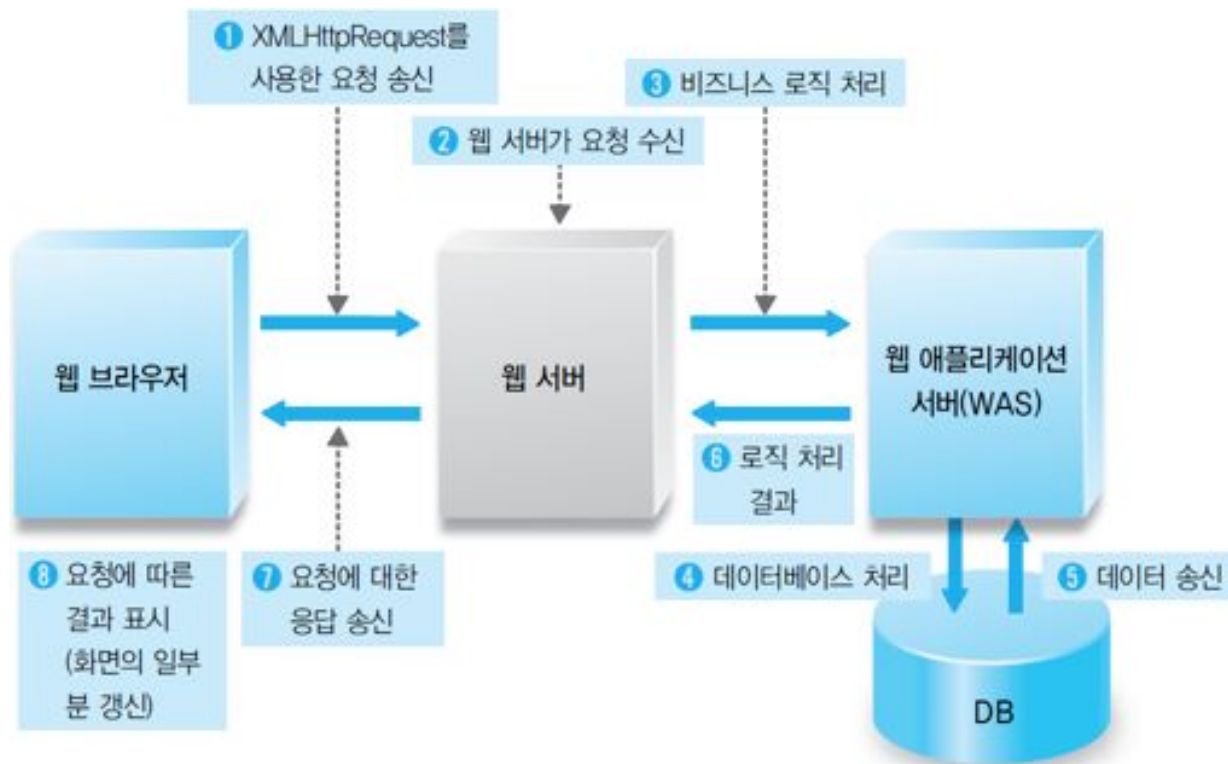
Ajax 통신 방식

Ajax: Asynchronous JavaScript And XML

1. 웹 브라우저는 웹 서버에게 요청(request)을 보냄
 - 브라우저는 응답(response)를 기다리지 않고, 다른 처리(요청)을 할 수 있음
2. 웹 서버는 요청에 대한 처리를 한 후 클라이언트에게 응답(response)을 보냄
3. 웹 브라우저는 응답이 도착했을 화면을 업데이트 함
 - 보통 화면의 일부만 갱신



Ajax(Asynchronous JavaScript and XML)



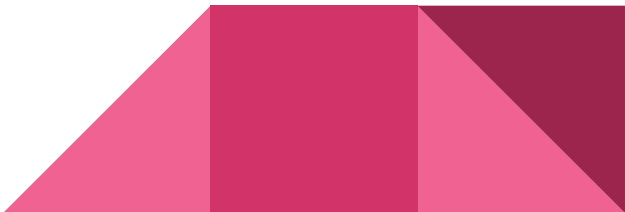
XMLHttpRequest Functions

함수	설명
open(method, url, async)	요청(request) 타입을 지정 <ul style="list-style-type: none">• method: 요청 타입 - GET/POST• url: 서버 (파일) 위치• async: 비동기 방식 사용 여부 - true/false
send()	GET 방식으로 요청을 서버로 전송
send(string)	POST 방식으로 요청을 서버로 전송 <ul style="list-style-type: none">• string: 요청 파라미터 이름과 값
setRequestHeader(header, value)	요청에 HTTP 헤더를 덧붙임 <ul style="list-style-type: none">• header: header 이름• value: header 값

XMLHttpRequest Properties

속성 (Properties)	설명
onreadystatechange	readyState 속성이 변경될 때 호출 될 함수를 정의
readyState	XMLHttpRequest의 상태 <ul style="list-style-type: none">• 0: request not initialized• 1: server connection established• 2: request received• 3: processing request• 4: request finished and response is ready
status	HTML 메시지 코드 <ul style="list-style-type: none">• 200: "OK"• 403: "Forbidden"• 404: "Page not found"
statusText	status-text를 리턴 (예: OK, Not found, ...)

Axios

- Getting Started
 - node.js와 브라우저를 위한 Promise 기반 HTTP 클라이언트
 - 브라우저: XMLHttpRequests 생성
 - node.js: http 요청 생성
 - Promise API를 지원
 - 요청 및 응답 인터셉트
 - 요청 및 응답 데이터 변환
 - 요청 취소
 - JSON 데이터 자동 변환
 - XSRF를 막기위한 클라이언트 사이드 지원
- 

Axios Example

```
// 지정된 ID를 가진 유저에 대한 요청
axios.get('/user?ID=12345')
  .then(function (response) {
    // 성공 핸들링
    console.log(response);
  })
  .catch(function (error) {
    // 에러 핸들링
    console.log(error);
  })
  .finally(function () {
    // 항상 실행되는 영역
  });
```

Axios Example

```
// async/await 사용을 원한다면, 함수 외부에 `async` 키워드를 추가.  
// async/await 는 ECMAScript 2017 문법  
async function getUser() {  
  try {  
    const response = await axios.get('/user?ID=12345');  
    console.log(response);  
  } catch (error) {  
    console.error(error);  
  }  
}  
  
getUser();
```