



실증 아카데미 리액트 특강

JavaScript와 웹 통신의 기본



일시 2021년 01월 11일

주최 동아대학교



CONTENTS—

01 JavaScript 자료형

- 원시 타입과 객체
- 원시 타입

02 JavaScript 객체

- 객체
- 비교 연산자
- 객체의 생성

03 JavaScript 함수

- 함수 객체
- 호이스팅, 스코프
- 다양한 함수 형태

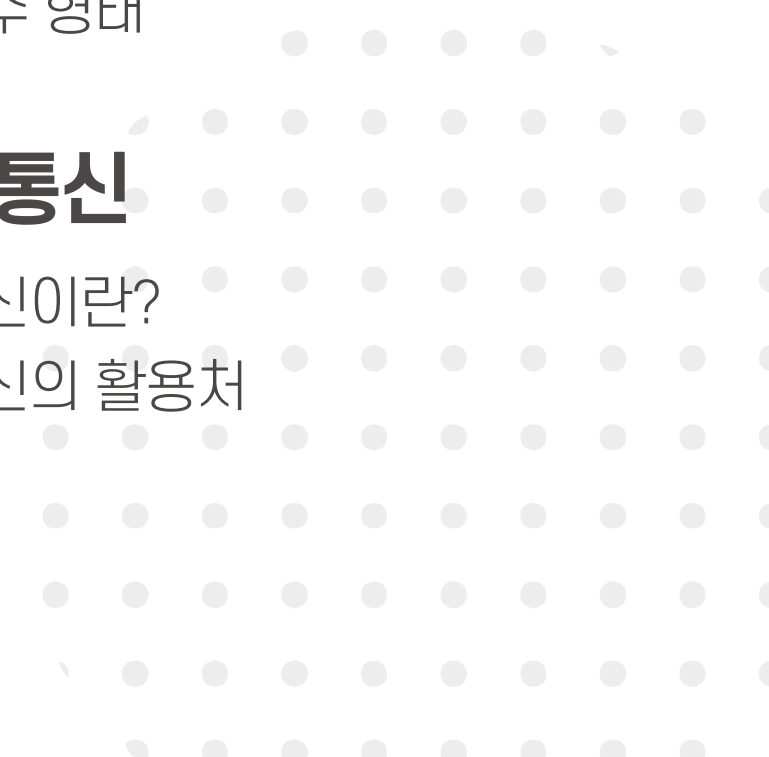
04 HTTP 통신

- HTTP 통신의 구조
- HTTP 요청 메소드
- HTTP 응답 구조

05 브라우저에서 요청

06 비동기 통신

- 비동기 통신이란?
- 비동기 통신의 활용처





01

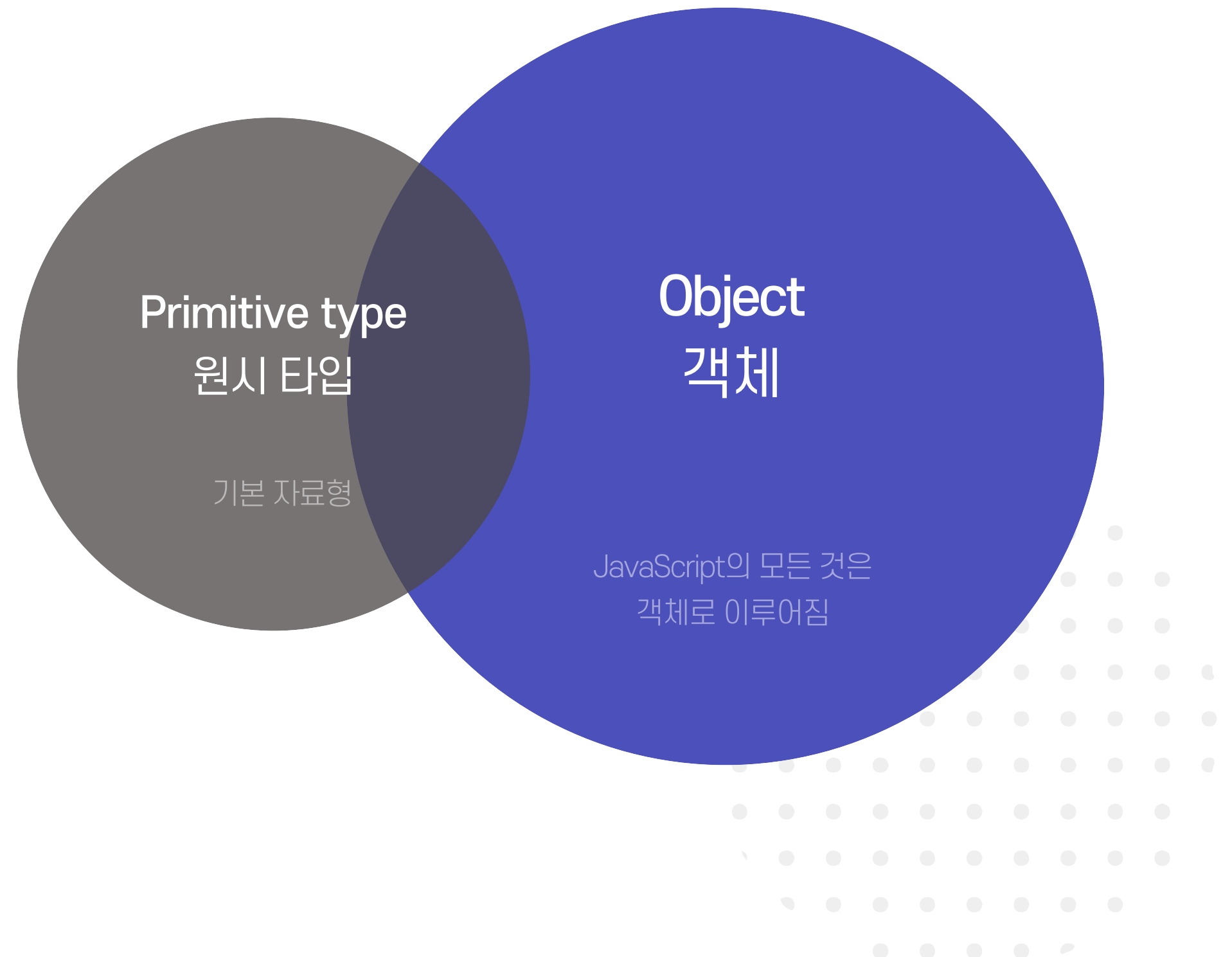
JavaScript 자료형

원시 타입과 객체
원시 타입

원시 타입과 객체

JavaScript는 7가지 원시 타입 존재

- 원시 타입은 변경 불가능한 값
- 원시 타입은 변수에 실제 값이 저장
- 원시 타입은 값에 의한 전달
- 객체는 변경 가능한 값
- 객체는 변수에 참조 값이 저장
- 객체는 참조에 의한 전달



원시 타입

원시 값은 변경 불가능한 값

- 변수 값이 아닌 원시 값이 변경 불가능

변수의 값은 재할당하면 다시 값을 넣을 수 있음

하지만 원시값에 $20 + 1$ 을 하면 메모리에는 20이라는 값과 21이라는 값이 동시에 존재함

- 문자열 또한 원시값으로 변경할 수 없음

다른 언어의 문자열과는 다르게 문자열 중간에 `s[0] = "d"`와 같이 할당을 하여도 전체 문자열은 변경되지 않음

- 원시 타입의 종류

boolean, number, string, null, undefined, object, symbol 7가지의 타입이 존재함



02

JavaScript 객체

객체

비교 연산자

객체의 생성

객체

JavaScript의 모든 것으로,
복잡한 자료구조

- 객체는 프로퍼티와 메서드로 이루어짐

프로퍼티는 key와 value의 쌍으로 이루어져 있으며,
`obj[key] = value;` 와 같이 사용함

메서드는 프로퍼티에 할당된 함수를 말하며,
`obj[key] = function () {};` 와 같이 사용함

- 객체는 변경 가능한 값

원시 타입과는 달리 `obj[key] = newValue;` 로 할당하면
값이 실제로 변경됨

거기에 더해 추가, 삭제 등도 가능함

- 객체는 참조 값

`obj1`과 `obj2`가 있으면, `obj2 = obj1`을 했을 때,
`obj1 === obj2`의 결과 값은 `true`

비교 연산자

JavaScript에서는 == 연산자와 === 연산자가 존재함

연산자	== 연산자	=== 연산자	!= 연산자	!== 연산자
의미	동등 비교	일치 비교	부동등 비교	불일치 비교
사례	x == y	x === y	x != y	x !== y
설명	x와 y의 값이 같음	x와 y의 값과 타입이 같음	x와 y의 값이 다름	x와 y의 값과 타입이 다름
객체에서	참조값을 비교함			

일반적으로 ==, != 연산자는 사용하지 않음



객체의 생성

함수, 배열, 정규 표현식 등 모든 것이 객체

- 객체는 다른 언어와 달리 new 키워드 없이 생성 가능
- 프로퍼티는 객체의 상태, 메서드는 객체의 동작
- 프로퍼티의 키는 문자열 또는 심벌 값으로 사용 가능
- 프로퍼티의 값은 JavaScript에서 사용 가능한 모든 값 사용 가능
- 프로퍼티에는 마침표, 대괄호를 통해 접근 가능

```
var counter = {  
  num: 0,  
  increase: function () {  
    this.num++;  
  }  
};
```

프로퍼티

메서드

- 객체의 생성 -



03

JavaScript 함수

함수 객체
호이스팅, 스코프
다양한 함수 형태

함수 객체

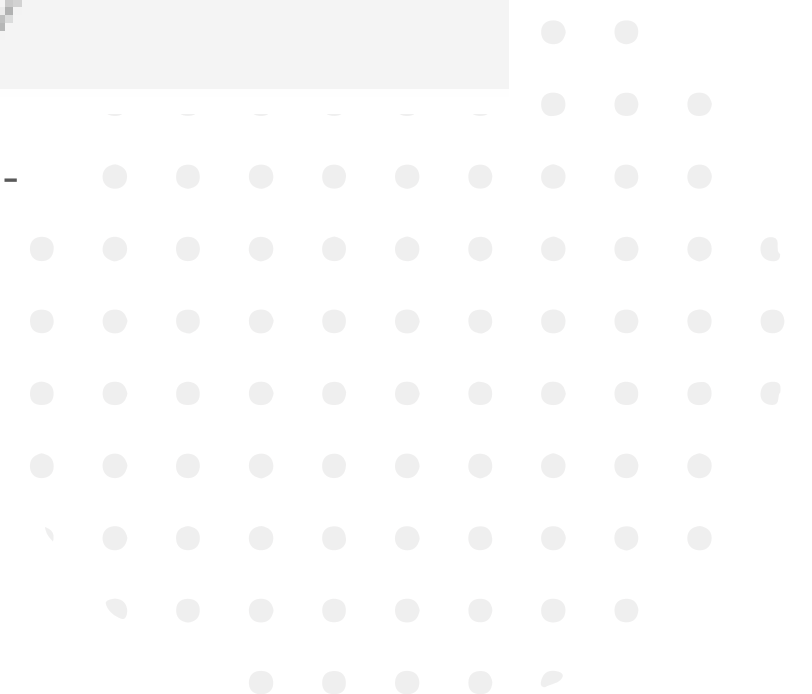
JavaScript에서 함수는 객체 타입의 값

- 함수는 일련의 과정을 감싼 하나의 실행 단위
- 매개변수(입력), 인수(입력), 반환 값(출력)이 있음
- 함수 정의를 통해 생성한 함수를 호출하여 사용
- 코드의 재사용이라는 측면에서 매우 유용
- JavaScript에서는 객체와 같이 동적으로 생성할 수 있음
- 함수 선언문은 호이스팅이라는 현상이 발생

```
// f(x, y) = x + y  
function add(x, y) {  
    return x + y;  
}
```

```
// f(2, 5) = 7  
add(2, 5); // 7
```

- 함수의 생성과 호출 -



호이스팅

함수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 JavaScript 고유의 특징

```
function add() {};  
add();  
plus();  
function plus() {}; // 함수 선언문
```

> 위 두 코드 모두 정상적으로 작동함

```
multi();  
division();  
var multi = function () {}; // 함수 표현식  
const division = function () {};
```

> 위 두 코드는 작동하지 않음



클로저

클로저는 함수 내부의 변수가 함수가 끝난 후에도 남아 있어 사용 가능한 매커니즘

```
const x = 1;
function outer() {
  const x = 10;
  const inner = function () { console.log(x); }
  return inner;
}
const innerFunc = outer();
innerFunc(); // 10
```

다음과 같이 outer 함수에서 x의 생명주기가 끝났음에도 외부에서 innerFunc를 호출하면 x에 여전히 참조할 수 있는 현상



다양한 함수 형태

01 즉시 실행 함수

```
(function () {  
    var a = 3;  
    var b = 5;  
    return a + b;  
})();
```

함수 정의와 동시에 즉시 호출되는 함수

※ 변수나 함수 이름 충돌 방지에 유용

03 콜백 함수

```
function repeat(n, f) {  
    for (let i = 0; i < n; i++) { f(i); }  
}  
  
const logAll = function (i) { console.log(i); };  
repeat(5, logAll); // 0 1 2 3 4
```

함수의 매개변수를 통해 다른 함수 내부로 전달되는 함수

※ 매개변수를 통해 함수의 외부에서 콜백 함수를 전달받은 함수를 고차함수라고 함

함수의 응용 방법

02 중첩 함수

```
function outer() {  
    function inner() {}  
    inner();  
}  
  
outer();
```

함수 내부에 정의된 함수

※ 호이스팅으로 인해 혼란이 발생할 수 있으므로 중첩함수 내에는
함수 선언문을 사용하지 않음



04

HTTP 통신

HTTP 통신의 구조

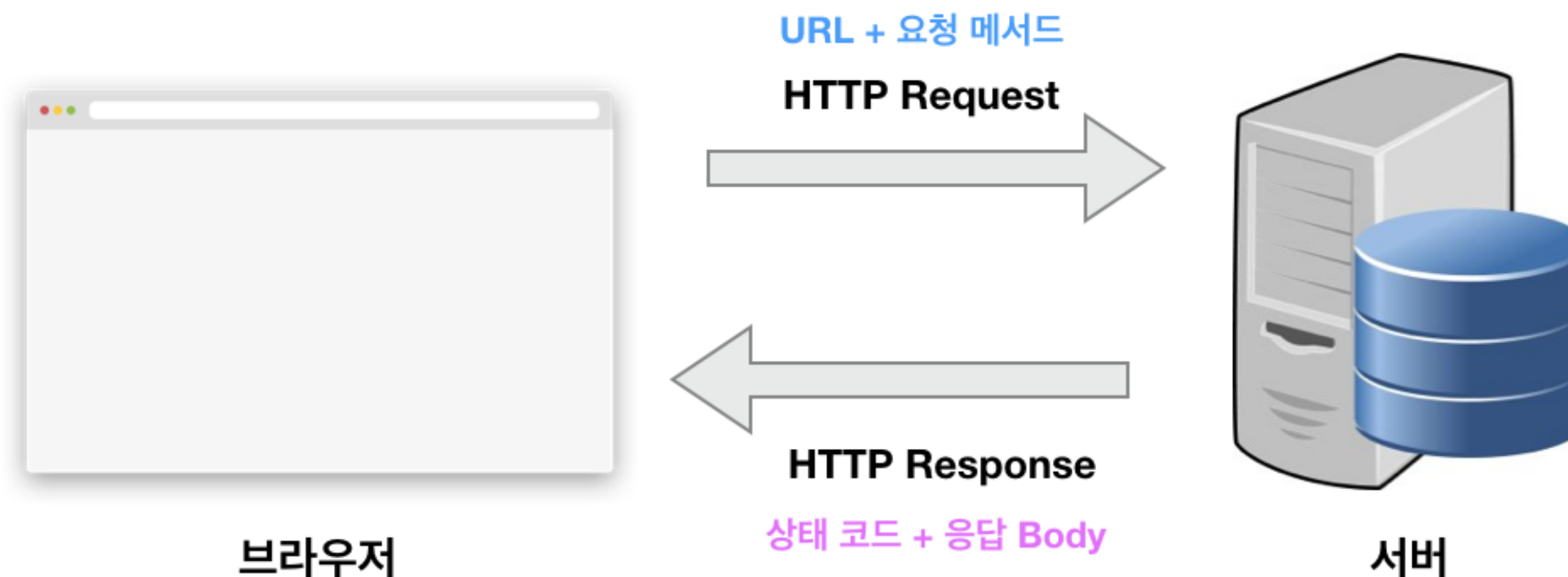
HTTP 요청 메소드

HTTP 응답 구조

HTTP 통신의 구조

HTTP는 요청과 응답으로 구성된 데이터 교환 규칙

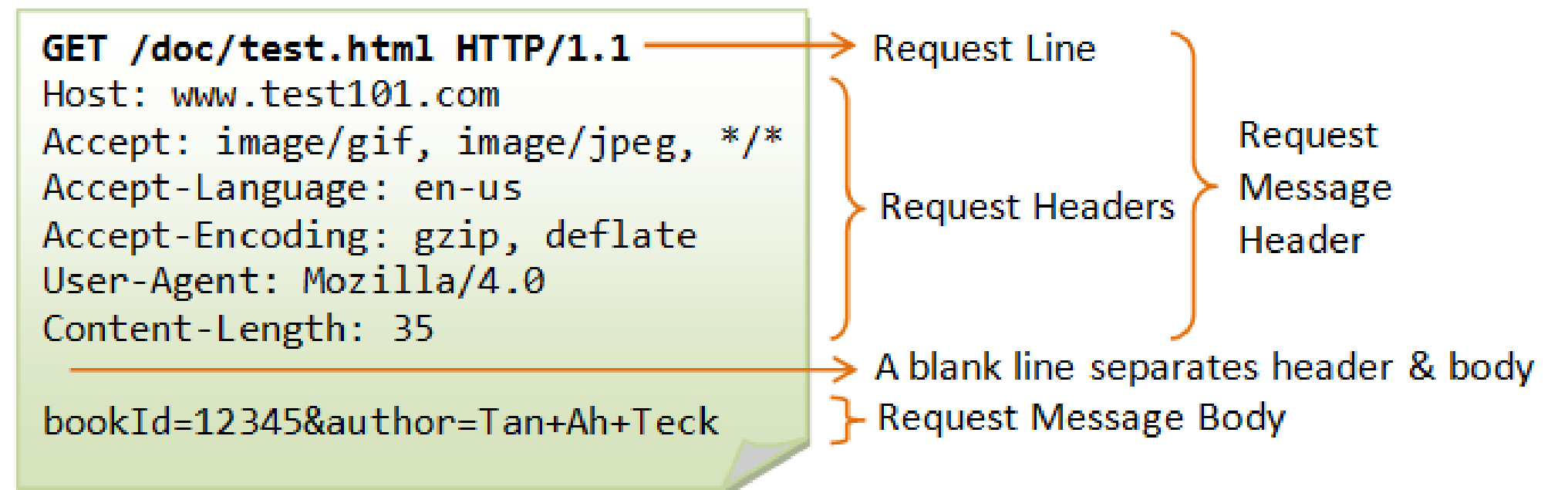
- 클라이언트가 HTTP Request를 서버에 보내면, 서버는 HTTP Response를 돌려주는 구조
- HTTP는 상태를 저장하지 않으며, 데이터가 필요하면 계속해서 데이터를 요청하고 응답받는 형태로 연결을 유지하지 않고 통신함
- HTTP는 크게 StartLine, Headers, Body 세 부분으로 구성되어 있음



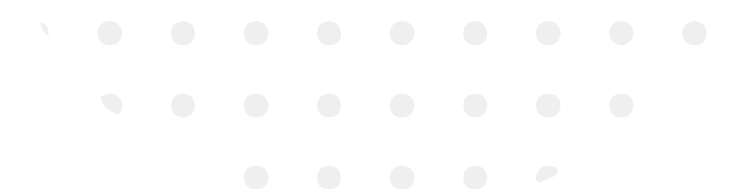
HTTP 요청 메소드

GET, POST, PUT, PATCH, DELETE, OPTIONS 등의 메소드가 있음

- HTTP 요청을 할 때, 해당 요청이 어떠한 역할을 정의하는지를 기술하는 이름을 말함
- GET 요청은 서버로부터 데이터를 받아(GET, read)올 때 사용하는 메소드
- 일반적으로 브라우저의 주소 창을 통해 접속하는 행위가 GET 메소드의 통신
- POST 요청은 데이터 생성/수정/삭제 등에 주로 사용되는 메소드
- 회원가입, 로그인 등의 중요 데이터가 포함될 때 많이 사용됨



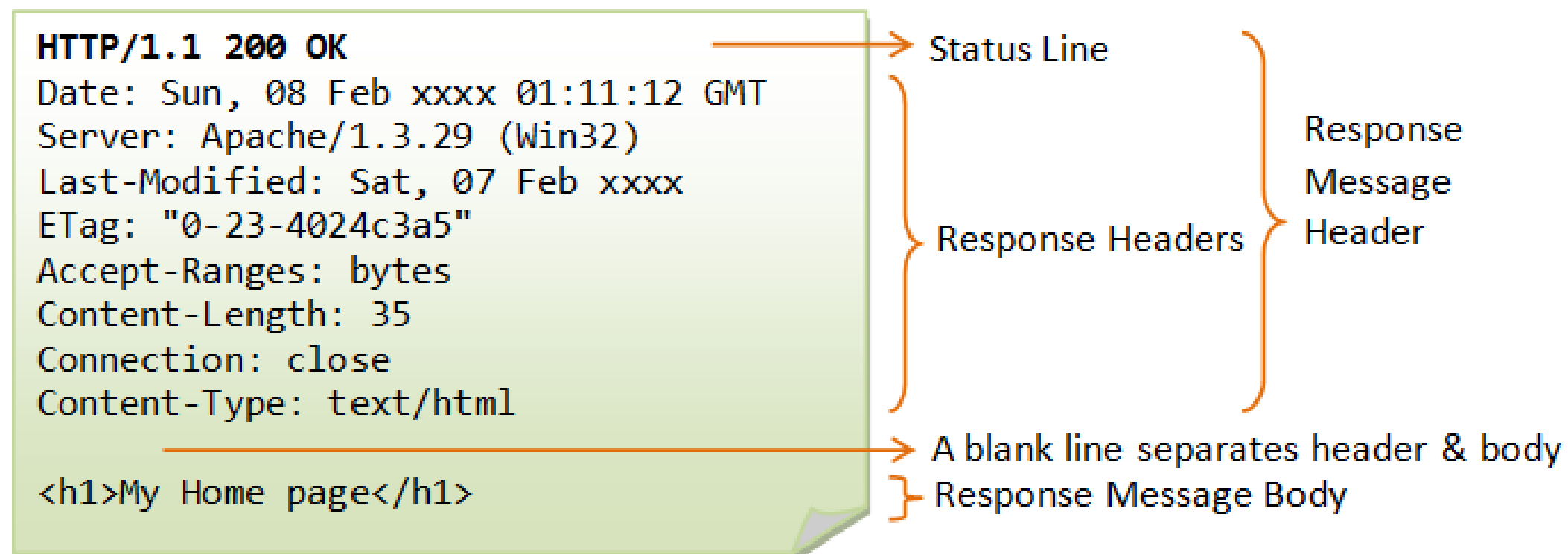
메소드 외에 Headers, Body를 통해
추가적인 데이터와 상태 등을 첨부하여 보낼 수 있음



HTTP 응답 구조

응답 상태 코드와 Headers로 통신에 관한 정보, body를 통해 데이터를 받아옴

- 응답에 대한 간략한 상태를 알려주는 응답 상태 코드가 존재함(20x ~ 50x)
- 브라우저가 응답 상태 코드만을 보고 판단하는 경우도 있음
- Headers에는 Server에서 저장시키는 값(쿠키, 세션 등)이 포함될 수 있음
- GET 방식으로 요청하면 body에는 HTML, CSS, JavaScript가 오는 것이 일반적
- 요청 메소드 및 응답 데이터에 따라 HTML이 아닐 수 있음





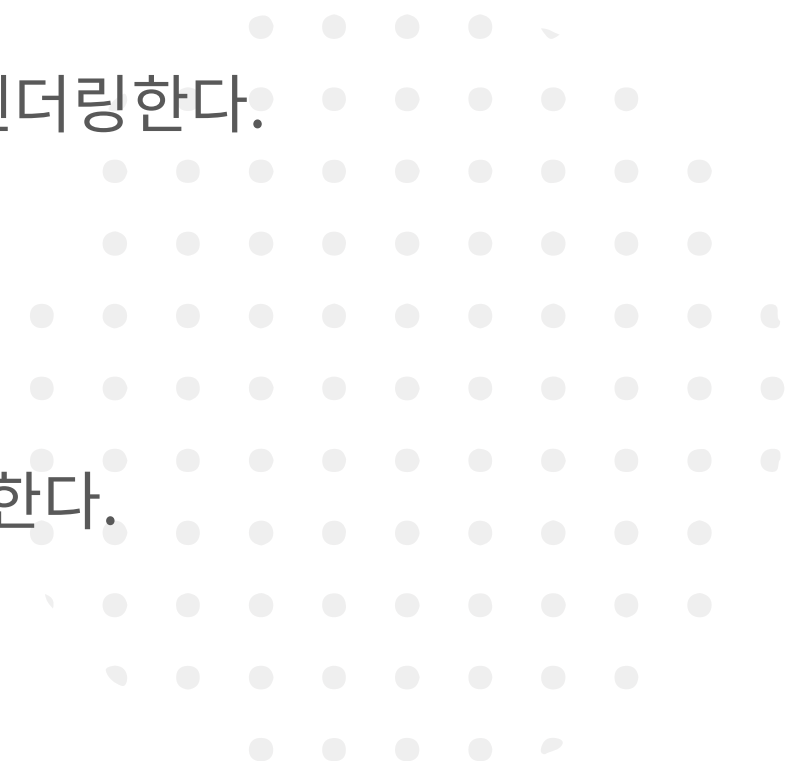
05

브라우저에서 요청

www.google.co.kr에 접속했을 때 브라우저의 행동

브라우저의 주소창에서 접속하는 행위는 GET 방식으로 통신하는 행위와 같음

1. 브라우저는 www.google.co.kr이라는 주소의 IP를 DNS라는 곳으로부터 받아온다.
2. 해당 주소의 IP로 HTTP/GET 메소드를 통해 요청한다.
3. Google 서버는 해당 요청에 대해 메인 페이지에 해당하는 HTML, CSS, JavaScript를 200 상태 코드로 응답한다.
4. 상태 코드 확인 후 브라우저는 받아온 HTML, CSS, JavaScript를 해석하여 뼈대를 구성하고 렌더링한다.
5. 사용자는 www.google.co.kr의 홈페이지를 확인한다.
6. 홈페이지에서 사용자의 행동에 따라 추가 데이터를 비동기적으로 요청하는 등의 이벤트를 수행한다.





06

비동기 통신

비동기 통신이란?

비동기 통신의 활용처

비동기 통신이란?

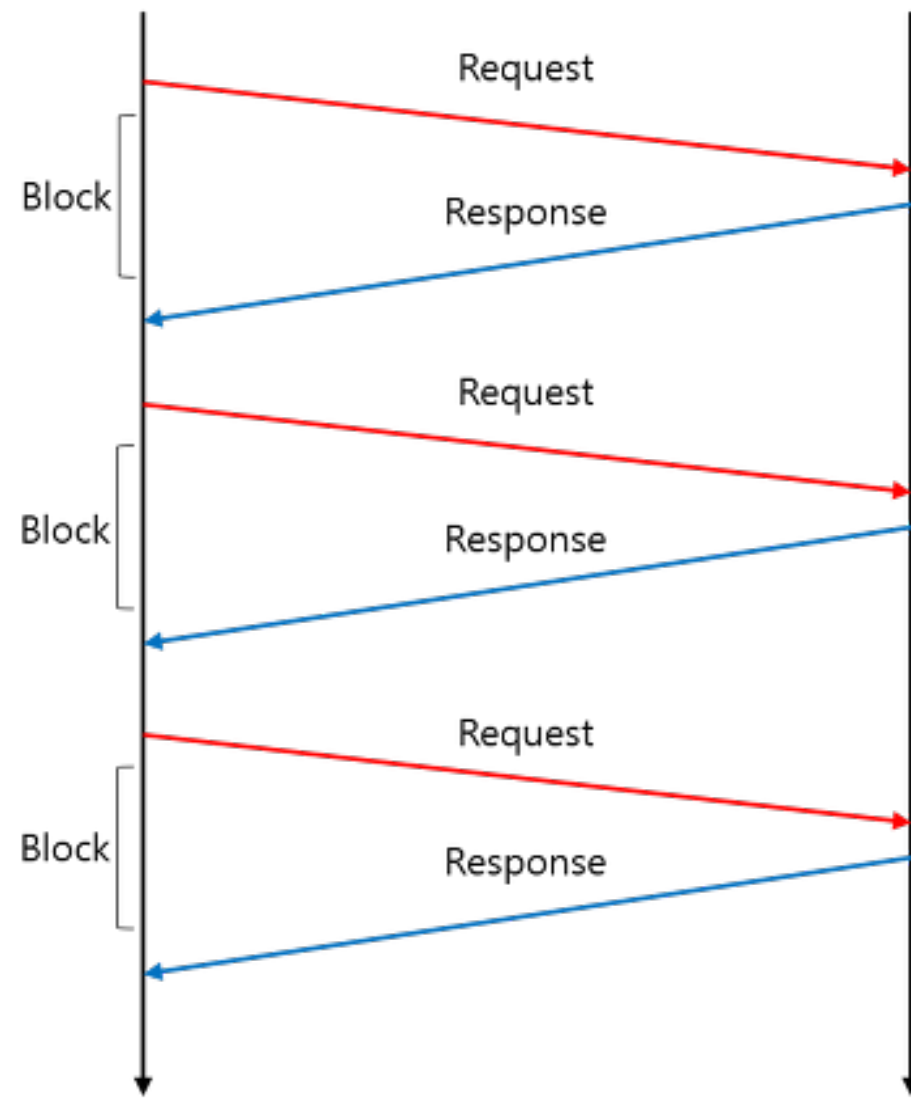
동기식 통신과 반대로 요청 후 Response가 올 때까지 기다리지 않는 통신(동시에 X)

- 기존의 HTTP 통신은 요청 후 응답이 올 때까지 응답을 기다리는 방식
- 때문에 어떤 홈페이지에 접속하면 로딩을하고 기다리는 경우가 일반적
- 하지만 이는 동시에 여러가지 일을 할 수 없는 병목현상을 유발함
- Non-Block 상태/통신이라고도 함
- 이를 활용해 페이지의 이동 없이 추가적인 정보를 요청하면서도 페이지 새로고침을 하지 않을 수 있게 됨
- JavaScript는 Ajax라는 기술이 등장하면서 급격히 발전하였음
- 이전에는 HTML에서 새로운 정보를 요청하기 위해서 페이지 이동이나 새로고침을 동기적으로 요청하여 업데이트하는 방식으로 이용되었음



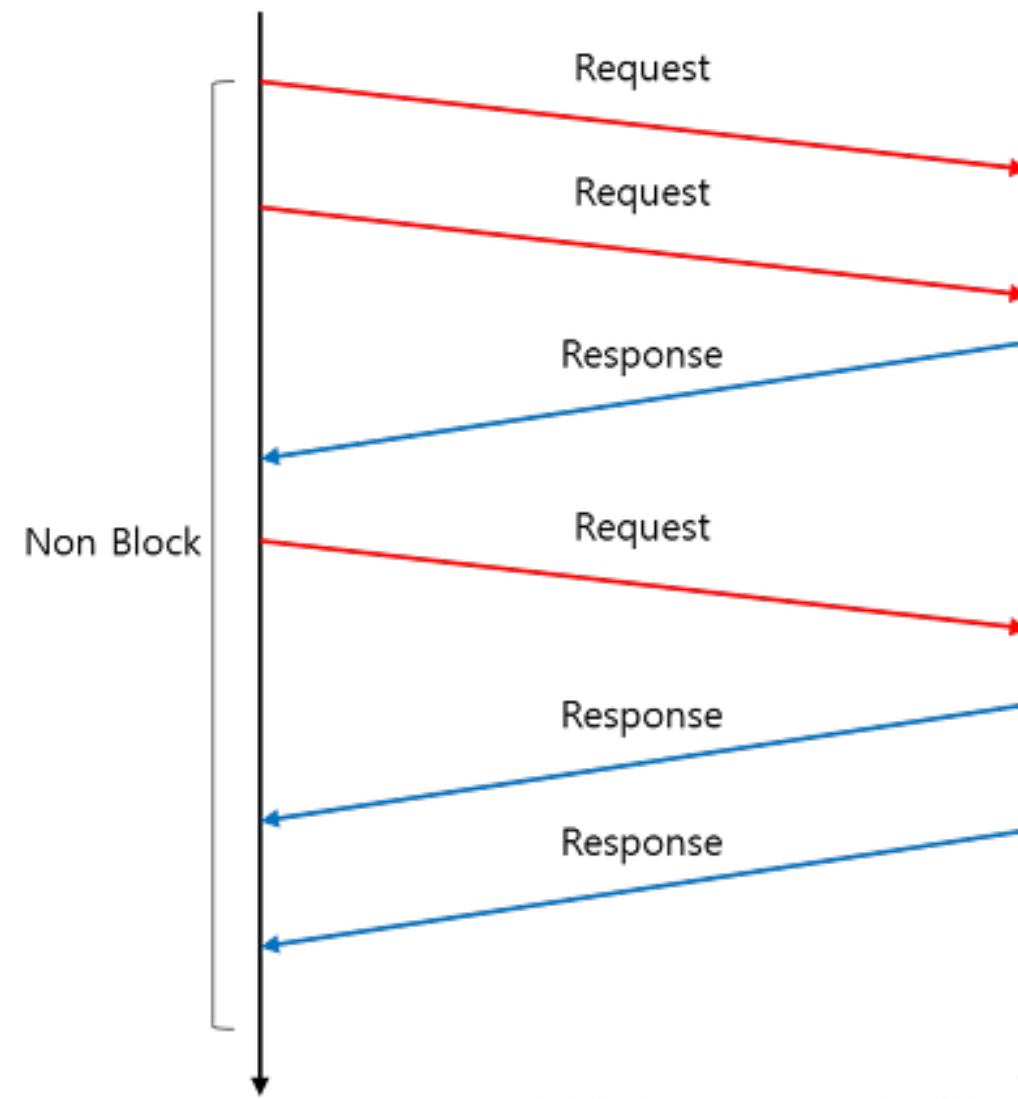
비동기 통신이란?

Synchronous (동기식)



※ 요청과 응답의 순서가 보장된다

Asynchronous (비동기식)

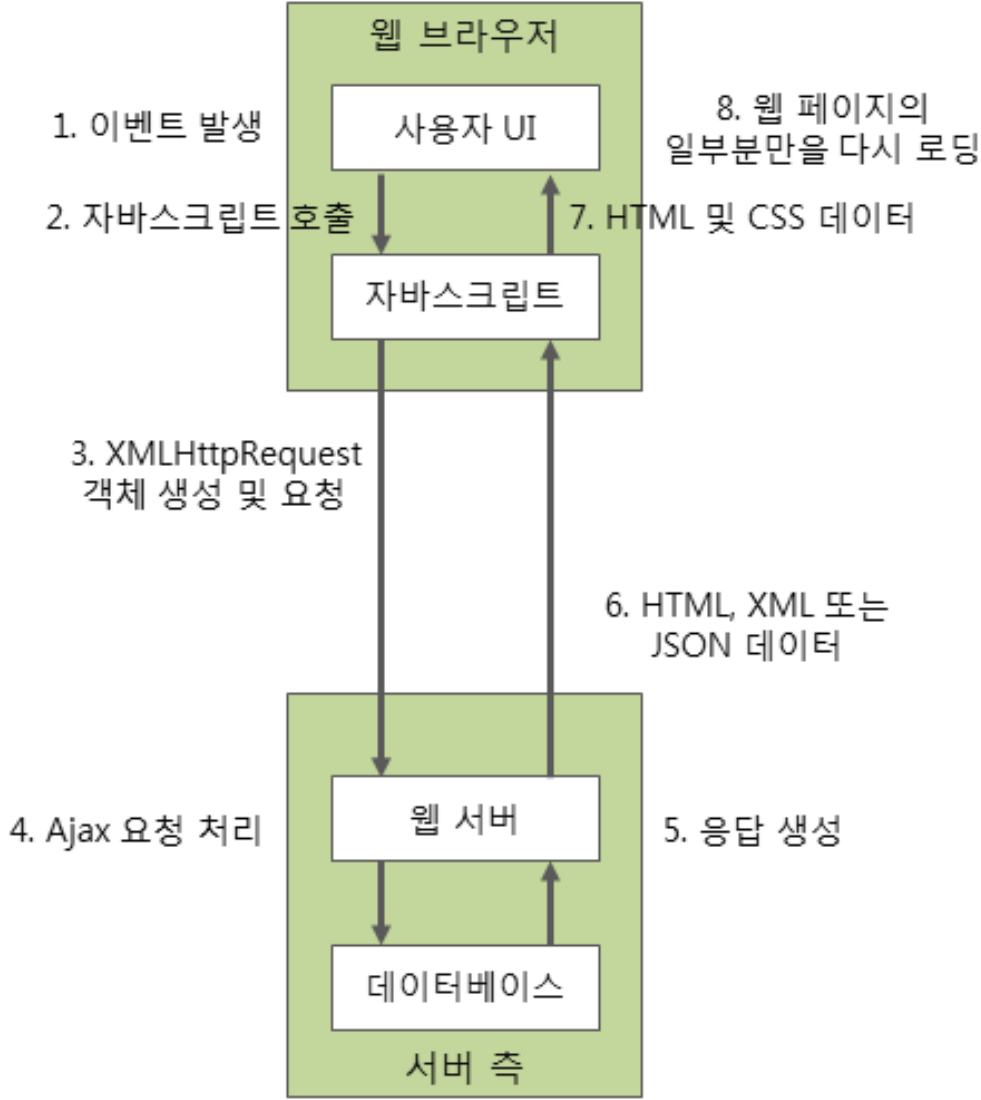


※ 요청과 응답의 순서가 보장되지 않는다

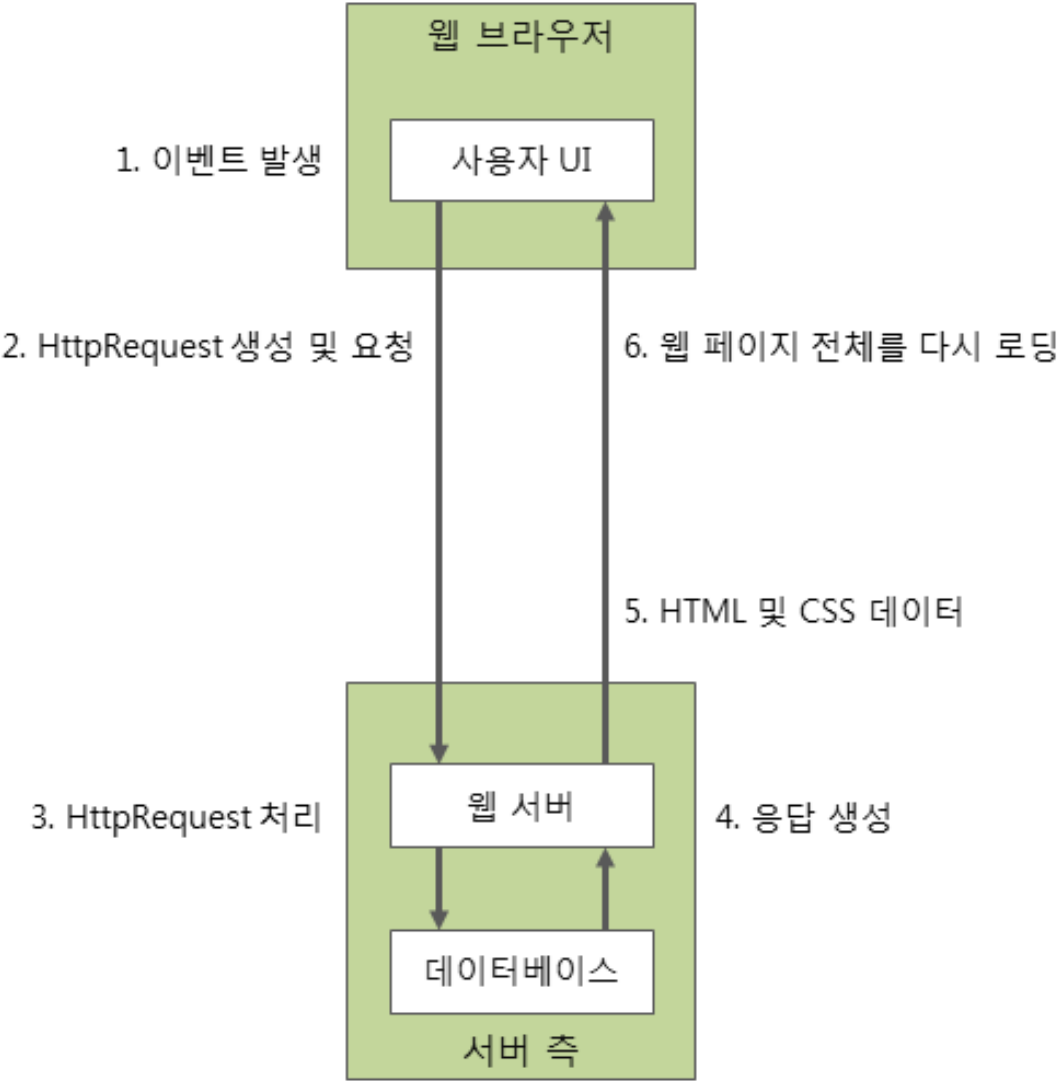


기존 웹 서비스와 비동기 웹 서비스의 비교

< Ajax를 이용한 웹 응용 프로그램의 동작 원리 >



< 기존 웹 응용 프로그램의 동작 원리 >



비동기 통신의 활용처

서버측 데이터의 변동이 잦은 경우에 굉장히 유리함

- 기존에는 서버 측의 데이터가 변경되어도 페이지를 직접 새로고침하여 새로운 데이터를 갱신 받아야 했음
- 이는 비연결 통신방식인 HTTP의 문제점으로 인해 발생
- 하지만 비동기 통신을 통해 새로고침하지 않고 데이터만 요청하여 기존에 렌더링된 화면 위에 데이터를 추가하는 방법을 사용할 수 있게 됨
- 웹 서비스의 속도 개선과 불편함 개선
- 이 기술이 발전되면서 웹 애플리케이션이라는 서비스가 더불어 발전하기 시작함
- 페이스북, 인스타그램과 같은 SNS는 페이지를 이동하지 않고 데이터만 요청하여 구성된 대표적인 SPA 서비스
 - ※ 과거에는 추천 기능을 누르면 페이지가 이동이 됐지만 현재는 페이지 내에서 좋아요 가능한 이유
- 요즘엔 과거의 웹 서비스인 카페, 커뮤니티 등지에서도 활용되고 있음(새로고침이 없어도 됨)
- 오늘날에는 거의 필수적인 기술
- 단점으로는 요청과 응답의 순서가 보장되지 않으므로 해당 부분에 대한 확실한 처리를 해줘야함



실증 아카데미 리액트 특강

THANK
YOU

(주) 인바이즈 박철현
(주) 인바이즈 개발팀