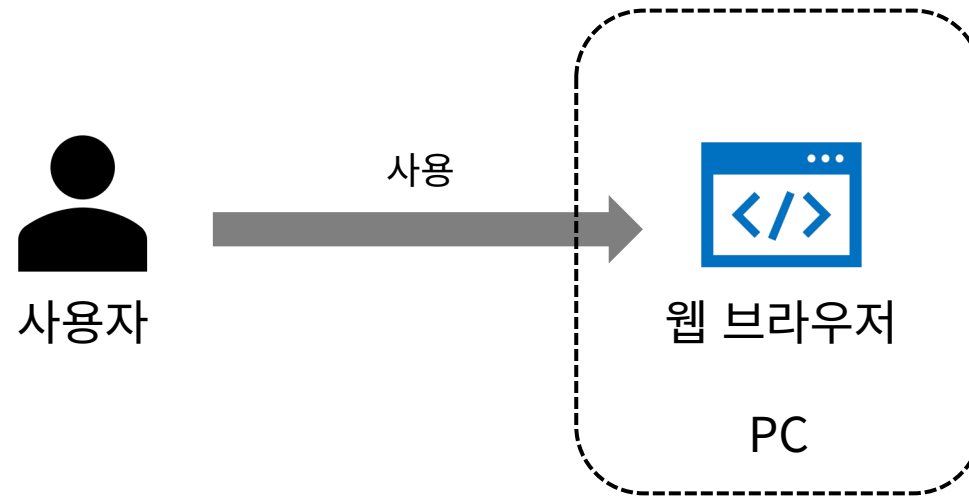


# 웹 애플리케이션 아키텍처

엄진영

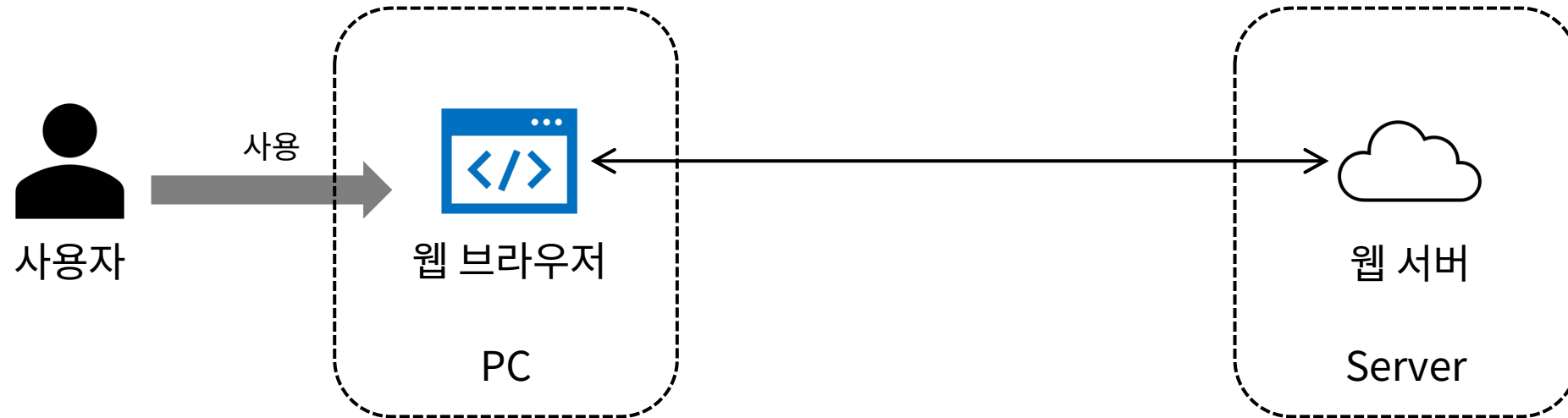
# 사용자와 웹 애플리케이션

- 웹 애플리케이션이란, 웹 브라우저를 통해 실행하는 프로그램이다.



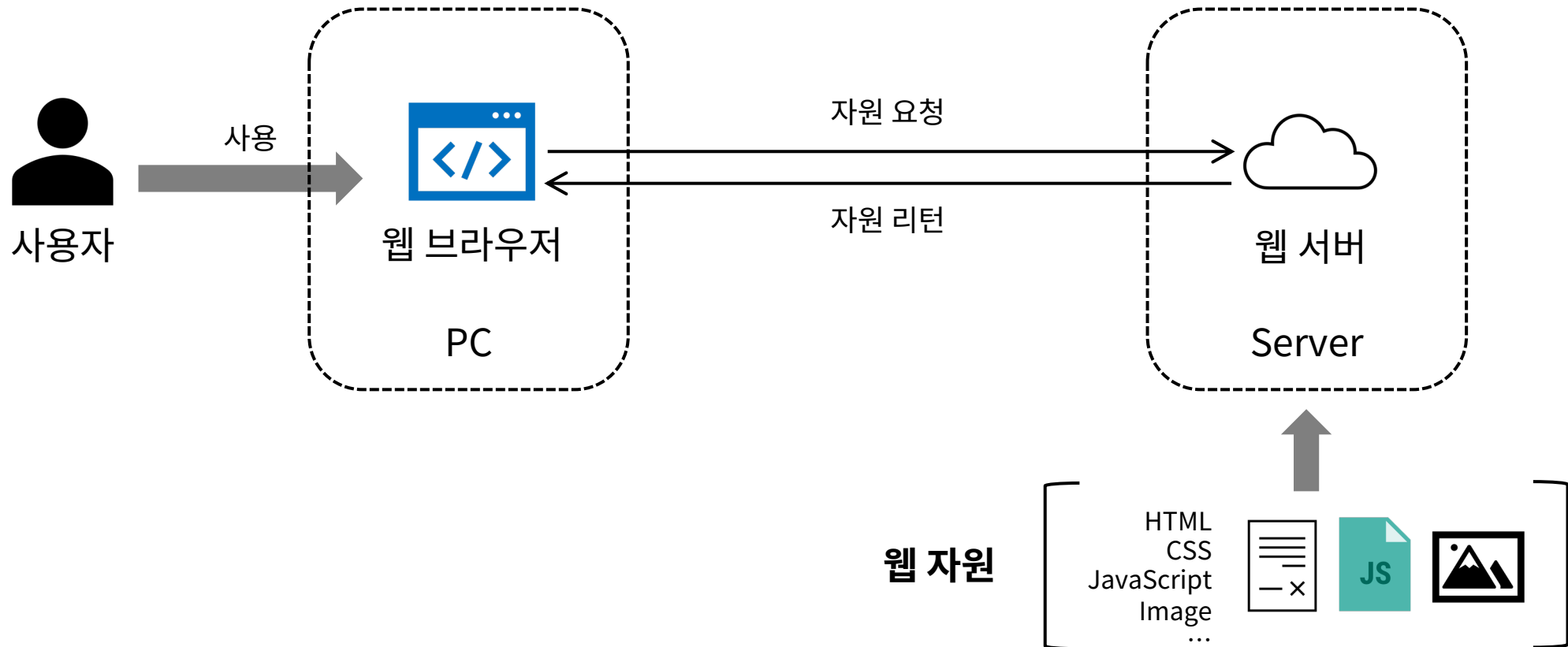
# 웹 애플리케이션 → 웹 브라우저 + 웹 서버

- 웹 애플리케이션은 웹 브라우저와 웹 서버의 협업으로 실행된다.
- 웹 브라우저는 로컬 컴퓨터에서 실행되고, 웹 서버는 원격 컴퓨터에서 실행된다.



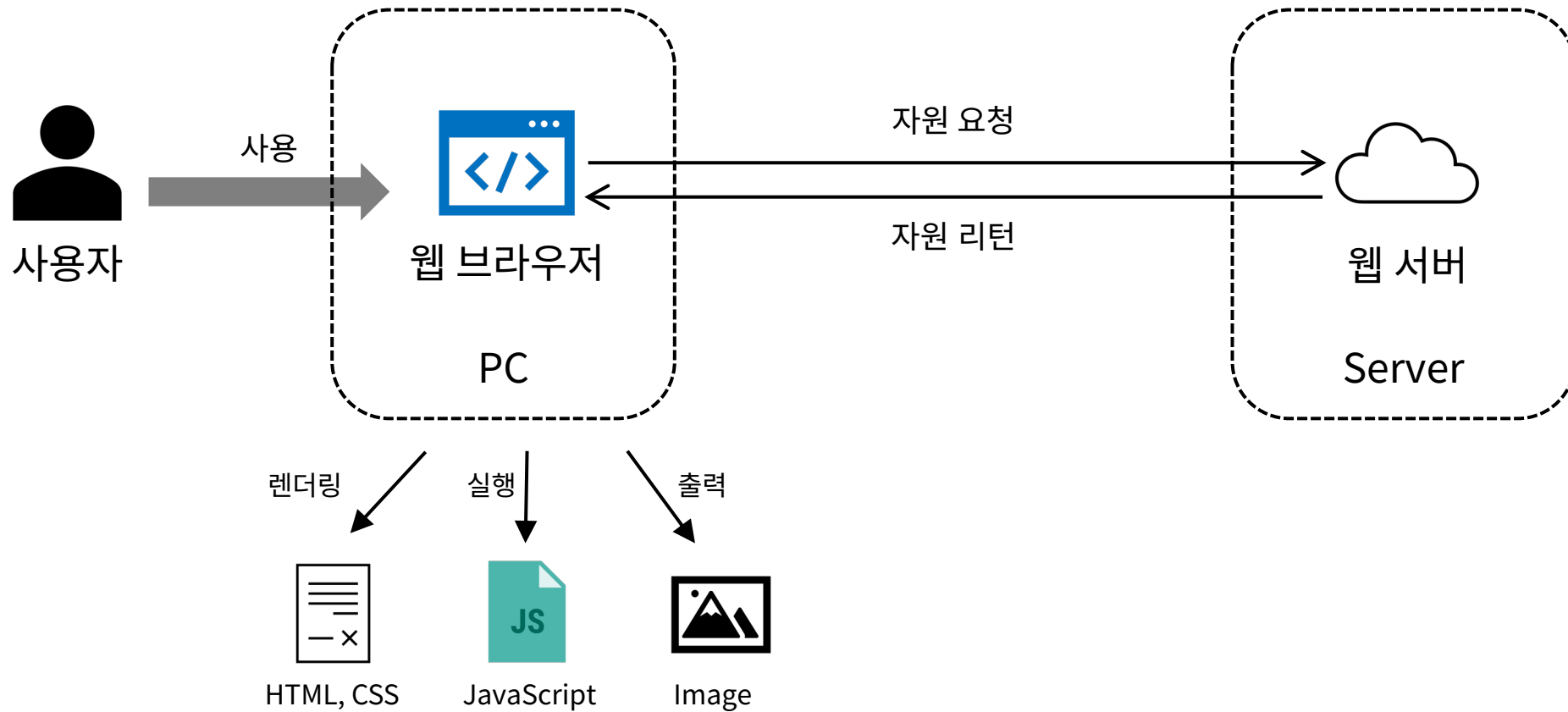
# 웹 서버의 역할

- 웹 서버는 웹 애플리케이션 자원을 갖고 있다.  
예) HTML, CSS, JavaScript, Image 등
- 웹 브라우저가 자원을 요청하면 해당 자원을 찾아 보내준다.



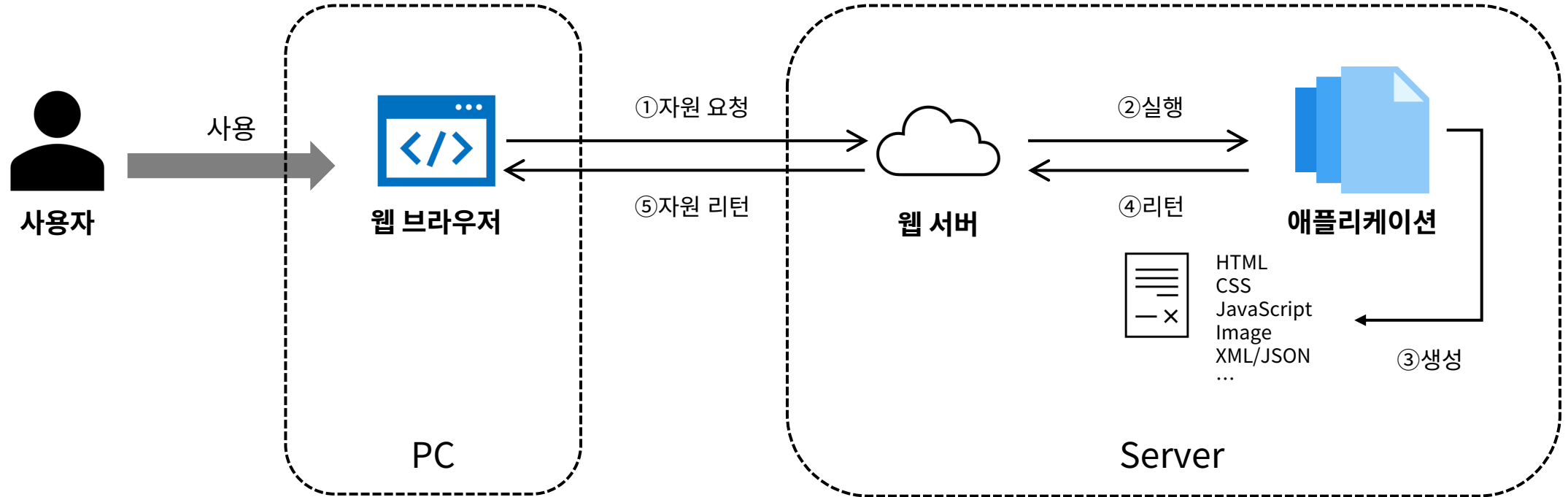
# 웹 브라우저의 역할

- 웹 브라우저는 서버에서 받은 자원을 실행한다.
- 자원이란? HTML, CSS, JavaScript, 이미지(PNG, JPG, SVG), 폰트 파일, PDF, 영상 파일 등을 뜻한다.



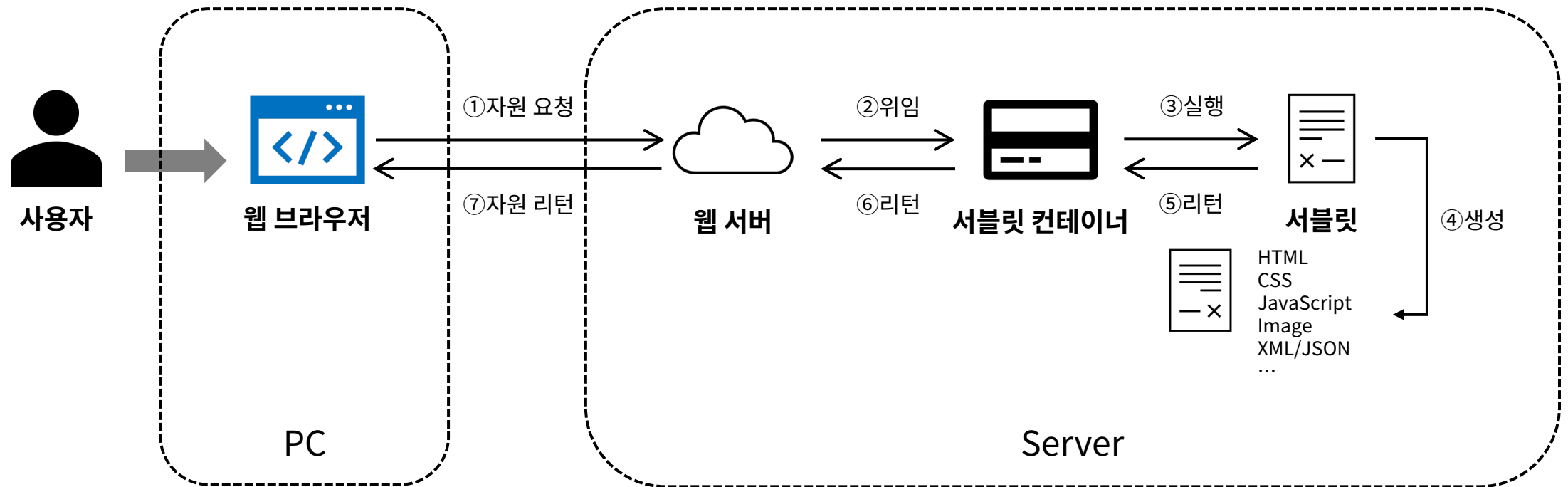
# 정적 자원과 동적 자원

- 정적 자원은 서버에 미리 존재하는 파일로, 요청이 들어오면 웹 서버는 가공 없이 그대로 읽어서 응답한다.  
예) HTML, CSS, JavaScript, 이미지 파일 등
- 동적 자원은 웹 서버 측에서 실행하는 프로그램이다. 요청이 들어오면 프로그램이 생성한 결과를 리턴 해준다.  
예) PHP, Python, Java Servlet/JSP 등



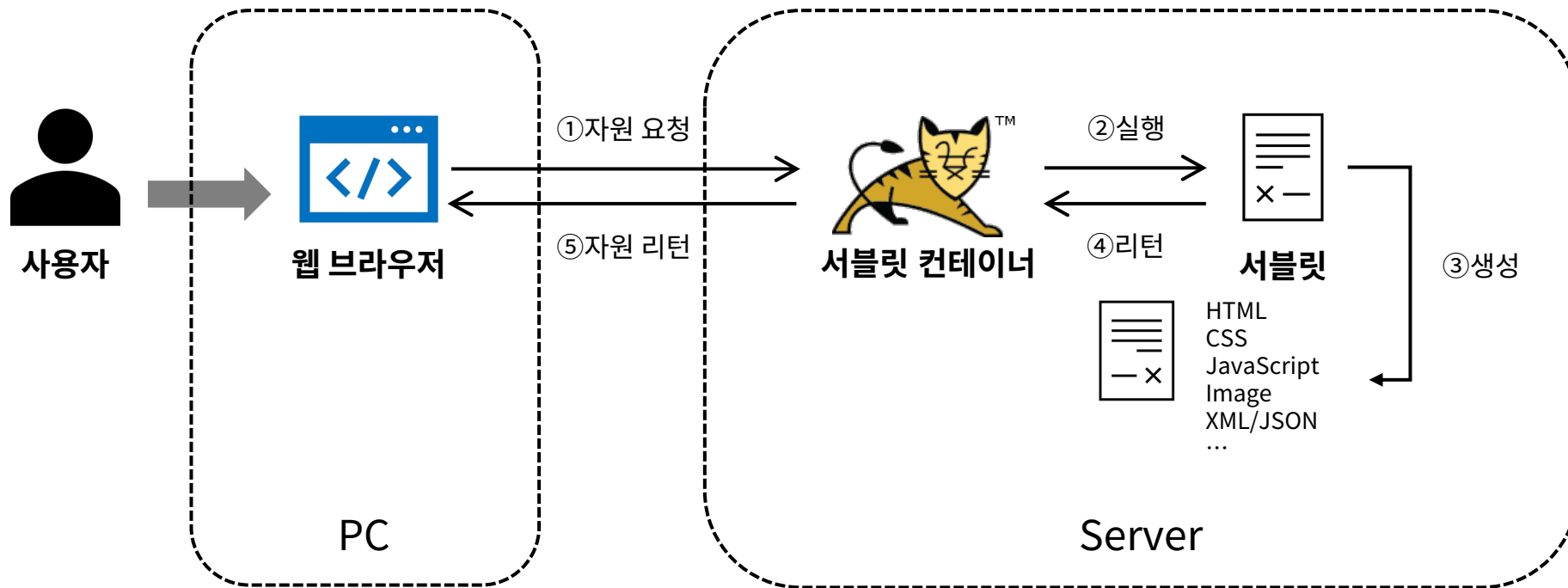
# 자바 웹 애플리케이션

- 자바 웹 애플리케이션은 Jakarta EE 명세(8 이하 버전은 Java EE)에 따라 자바로 작성한 프로그램이다.  
예) 서블릿, JSP, 필터, 리스너 등의 자바 컴포넌트
- Jakarta EE 구현 서버(또는 서블릿 컨테이너)에서 이들 컴포넌트를 관리하고 실행한다.



# Apache Tomcat 서버

- 자바 웹 애플리케이션을 실행할 수 있는 서버 프로그램이다.
- Jakarta EE 명세에서 웹 관련 기술 명세 일부를 구현한(해당 기술에 정의된 대로 동작하도록 만든) 서블릿 컨테이너다.  
예) Servlet, Pages, Expression Language, WebSocket 등
- HTTP/HTTPS 요청을 직접 수신하고 응답하는 기능이 내장되어 있어, 개발 과정에서 별도의 웹서버를 설치할 필요는 없다.





# Jakarta EE 기술 명세와 Apache Tomcat의 관계

기술 명세 및 버전		Jakarta EE	11	10	9.1	9	8
하위 기술 명세들	• Servlet		6.1	6.0	5.0	5.0	4.0
	• Server Pages		4.0	3.1	3.0	3.0	2.3
	• Expression Language		6.0	5.0	4.0	4.0	3.0
	• JSTL		3.0	3.0	2.0	2.0	1.2
	• Server Faces		4.1	4.0	3.0	3.0	2.3
	• Validation		3.1	3.0	3.0	3.0	2.0
	• WebSocket		2.2	2.1	2.0	2.0	1.1
	• Annotations		3.0	2.1	2.0	2.0	1.3
	• Enterprise Beans		4.0	4.0	4.0	4.0	3.2
	• JSON Processing		2.1	2.1	2.0	2.0	1.1
	• JSON Binding		3.0	3.0	2.0	2.0	1.0
	• ...		...	...	...	...	...
Java SE	• 최소 요구 버전		17+	11+	8+	8+	8+
서블릿 컨테이너	• Apache Tomcat		11 (Servlet 6.0)	10.1	10.1	10.1	9

# Spring Framework

- Spring Framework는 자바 기반의 엔터프라이즈 애플리케이션을 쉽고 유연하게 개발하기 위한 프레임워크다.
- 기존 EJB 중심 (구)Java EE 는 전용 서버에 강하게 의존해 기술 교체가 어려웠다.
- 스프링은 기술 종속성을 약화시켜, 최소한의 코드 변경으로 기술을 손쉽게 교체할 수 있다.

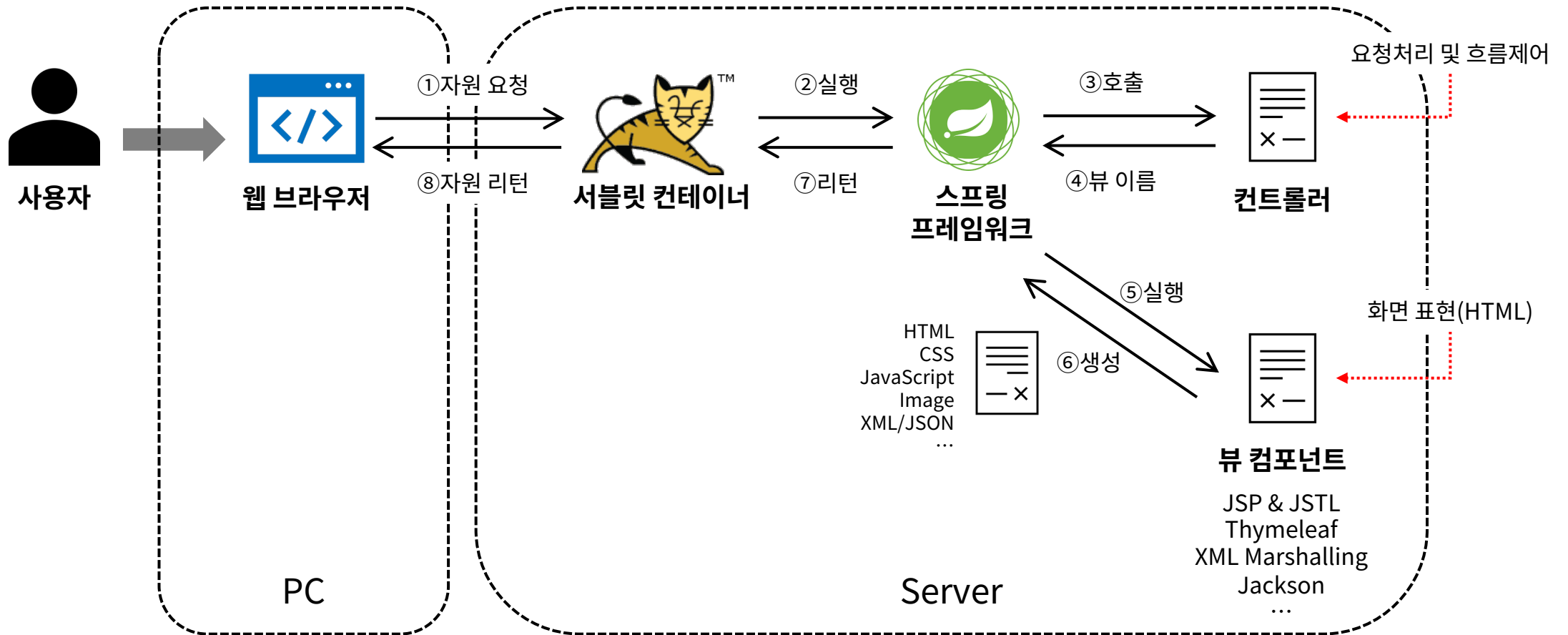


\* POJO(Plain Old Java Object)?

특정한 제약이나 기술에 종속되지 않은 순수한 자바 객체를 뜻한다.

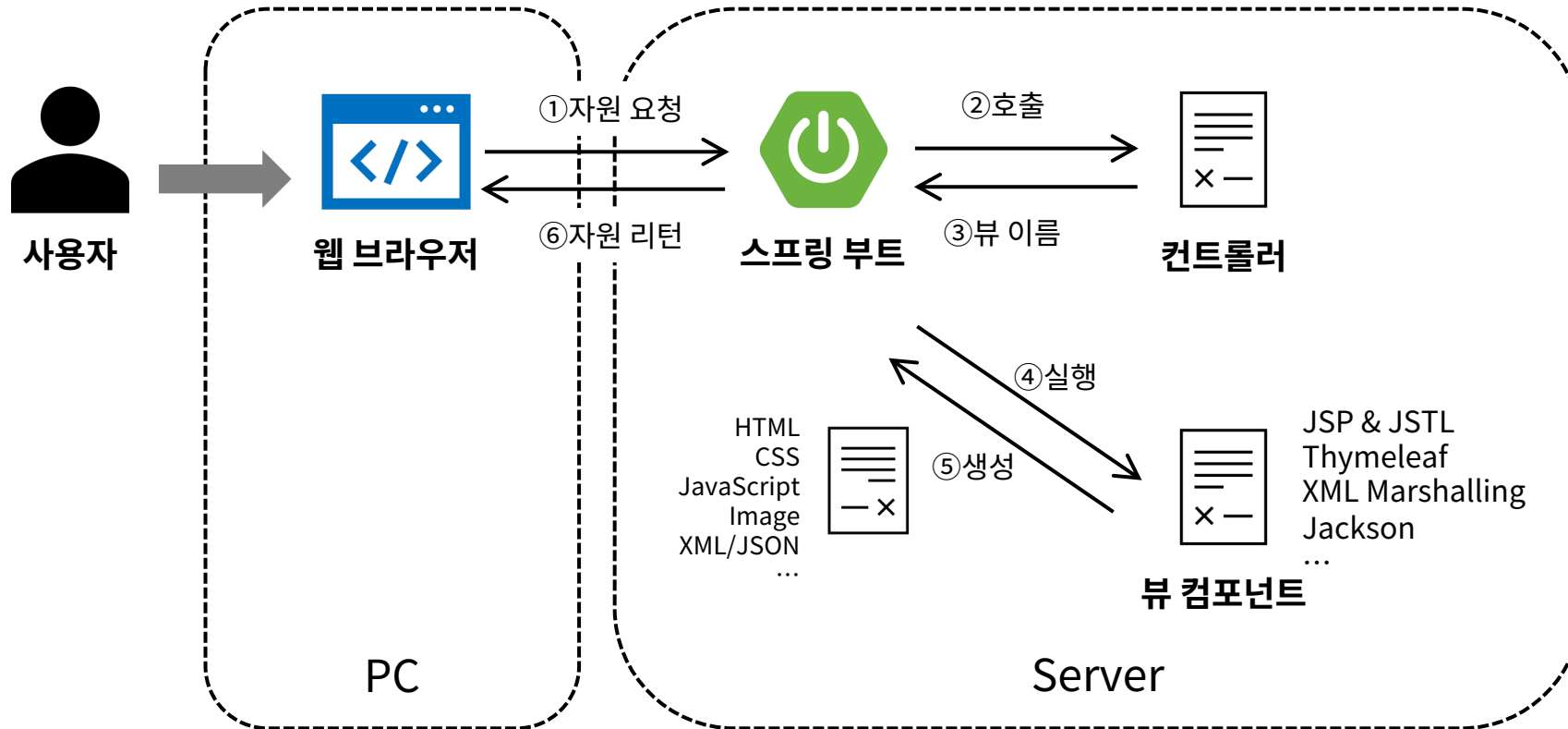
# Spring WebMVC

- Spring WebMVC는 Spring Framework가 제공하는 모듈 중 하나로, 웹 요청과 응답을 처리하는 역할을 담당한다.
- 웹 애플리케이션의 유지보수를 용이하게 하기 위해 역할을 분리하는 MVC(Model-View-Controller) 패턴을 사용한다.



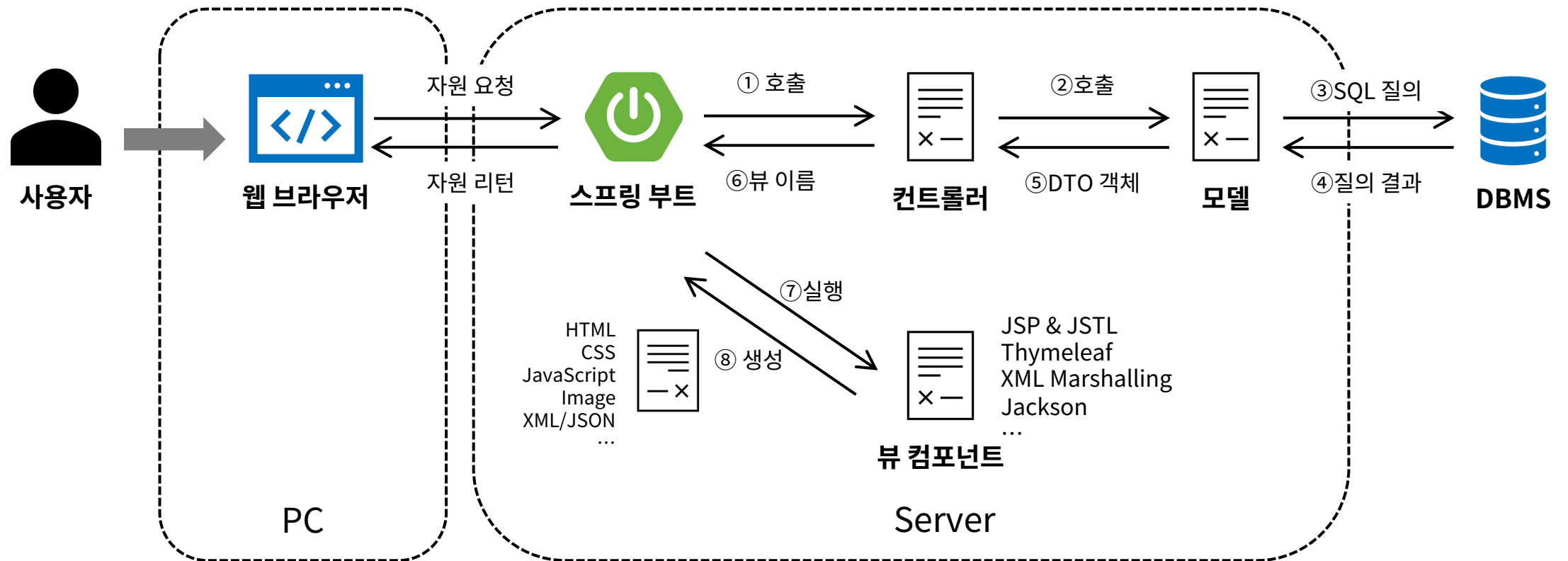
# Spring Boot = 서블릿 컨테이너 + 스프링 프레임워크

- Spring Boot는 Spring Framework를 더 쉽고 빠르게 사용하도록 만든 도구다.
- 서블릿 컨테이너를 내장하고 있어 따로 서버를 설치할 필요가 없고, 애플리케이션을 배치하고 테스트하기 쉽다.
- Starter를 이용하면 특정 기능을 쓰는 데 필요한 의존성을 묶음 단위로 관리할 수 있고, 각 라이브러리들의 버전이나 설정을 자동화할 수 있다.



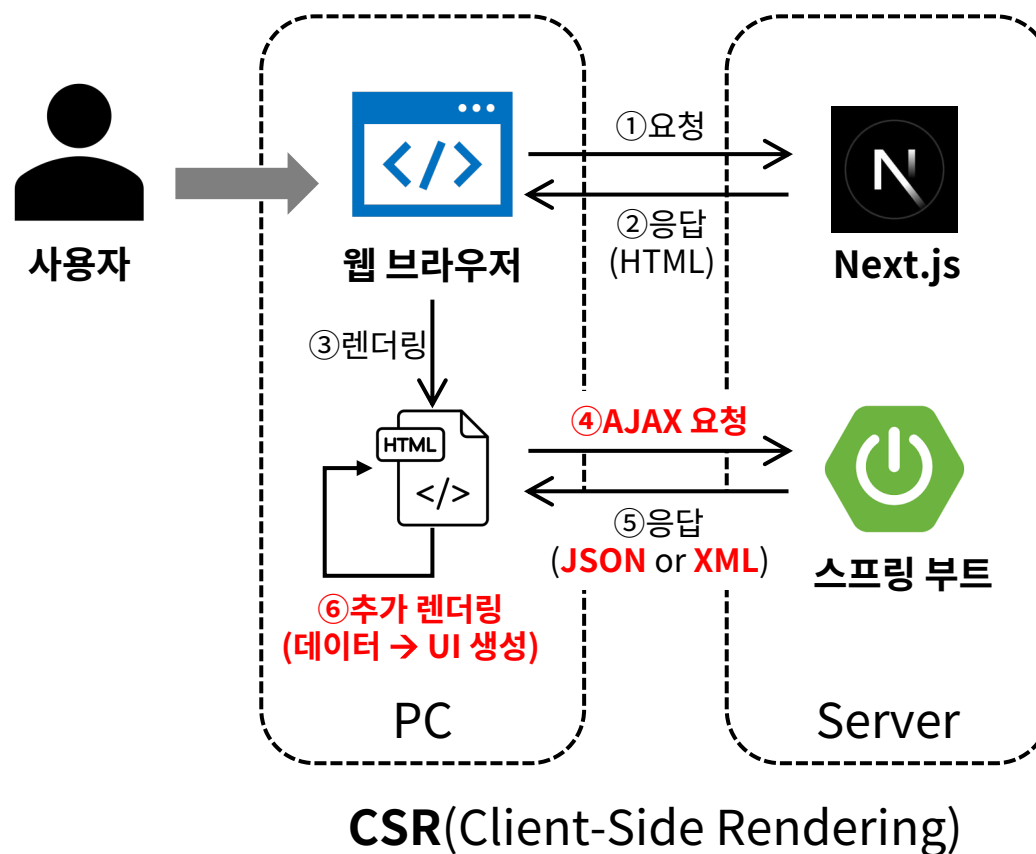
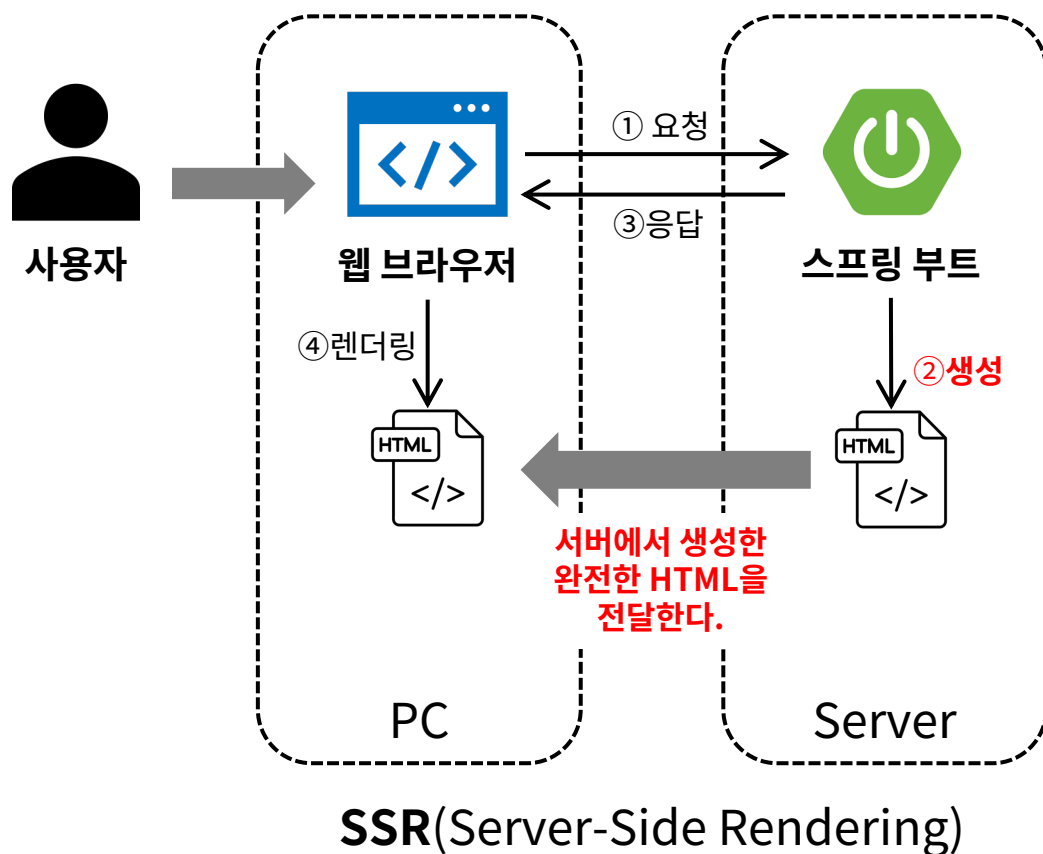
# DBMS와 Model 컴포넌트

- DBMS는 데이터를 저장/조회/수정/삭제하는 외부 시스템이다. 예) Oracle, MySQL, PostgreSQL, MongoDB 등
- MVC에서 **모델 컴포넌트**는 비즈니스 데이터와 규칙을 표현하고, 처리 결과를 전달하는 역할을 담당한다.
  - 예) 도메인/엔티티 객체(상태·규칙), 서비스 객체(유스케이스), DAO(DB 접근), DTO(뷰에 전달할 데이터)



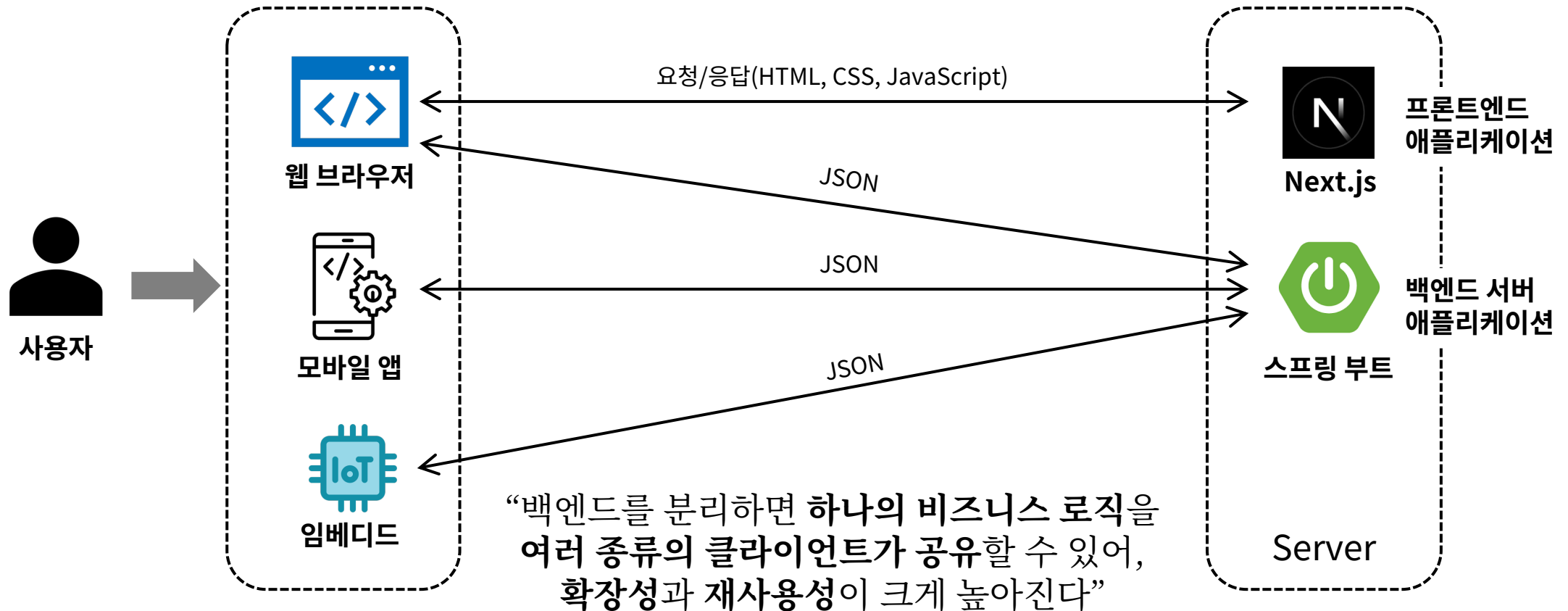
# SSR vs CSR

- **SSR**(Server-Side Rendering) 방식은 **서버에서 완성된 HTML**을 생성하여 브라우저로 전송한다. 일반적으로 사용자가 페이지를 요청할 때마다 서버가 HTML 응답을 생성한다.
- **CSR**(Client-Side Rendering) 방식은 최소한의 HTML과 JavaScript 번들을 받은 후, 브라우저에서 실행된 자바스크립트가 **서버로부터 데이터(예: JSON)**를 가져와 동적으로 DOM을 조작해 **화면**을 구성한다.



# 백엔드/프론트엔드 아키텍처

- UI(화면)와 서버 로직을 서로 다른 애플리케이션으로 분리하는 웹 아키텍처 스타일이다.
- **백엔드** 애플리케이션은 도메인로직, 인증인가, 데이터 영속성, API 제공의 역할을 담당한다.
- **프론트엔드** 애플리케이션은 UI 렌더링(주로 CSR), 상태 관리, 사용자 이벤트 처리 역할을 담당한다.

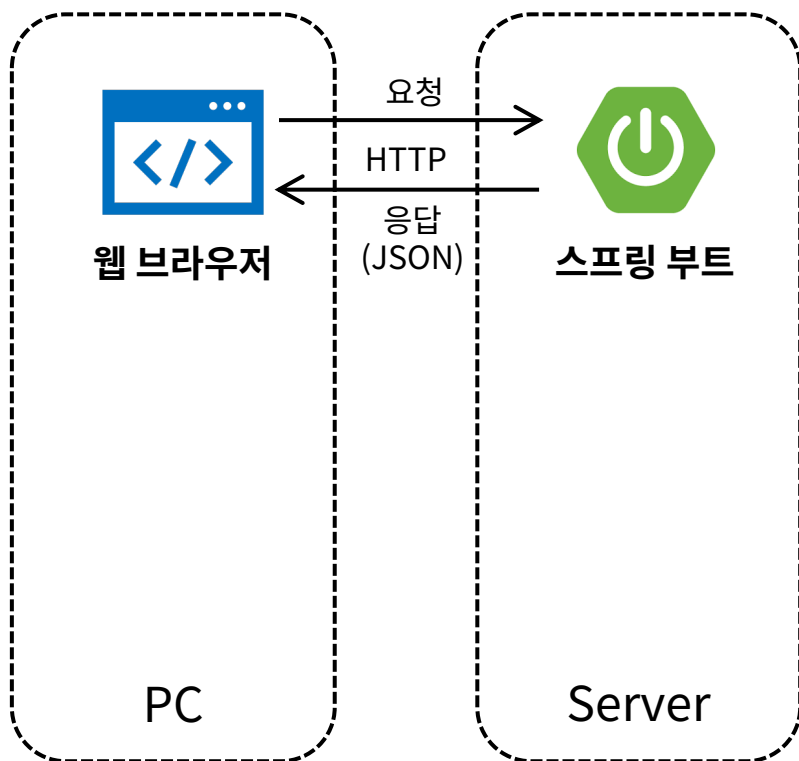


# Spring Boot 기반 백엔드 서버 구축



# RESTful API 서버

- RESTful API 서버는 클라이언트 요청에 대한 **자원의 상태**를 HTTP로 제공하는 서버이다.
- 'RESTful'은 REST 원칙을 잘 따르는 API나 시스템을 가리키는 말이다.
- REST(Representation State Transfer)는 웹에서 자원을 다루는 아키텍처 스타일이다. 즉, **웹 API를 설계하는 원칙**이다.



## REST 핵심 개념

### ① 자원(Resource)

URL로 표현한다. 예) /users, /orders/1

### ② HTTP Method

- GET → 조회
- POST → 생성
- PUT/PATCH → 수정
- DELETE → 삭제

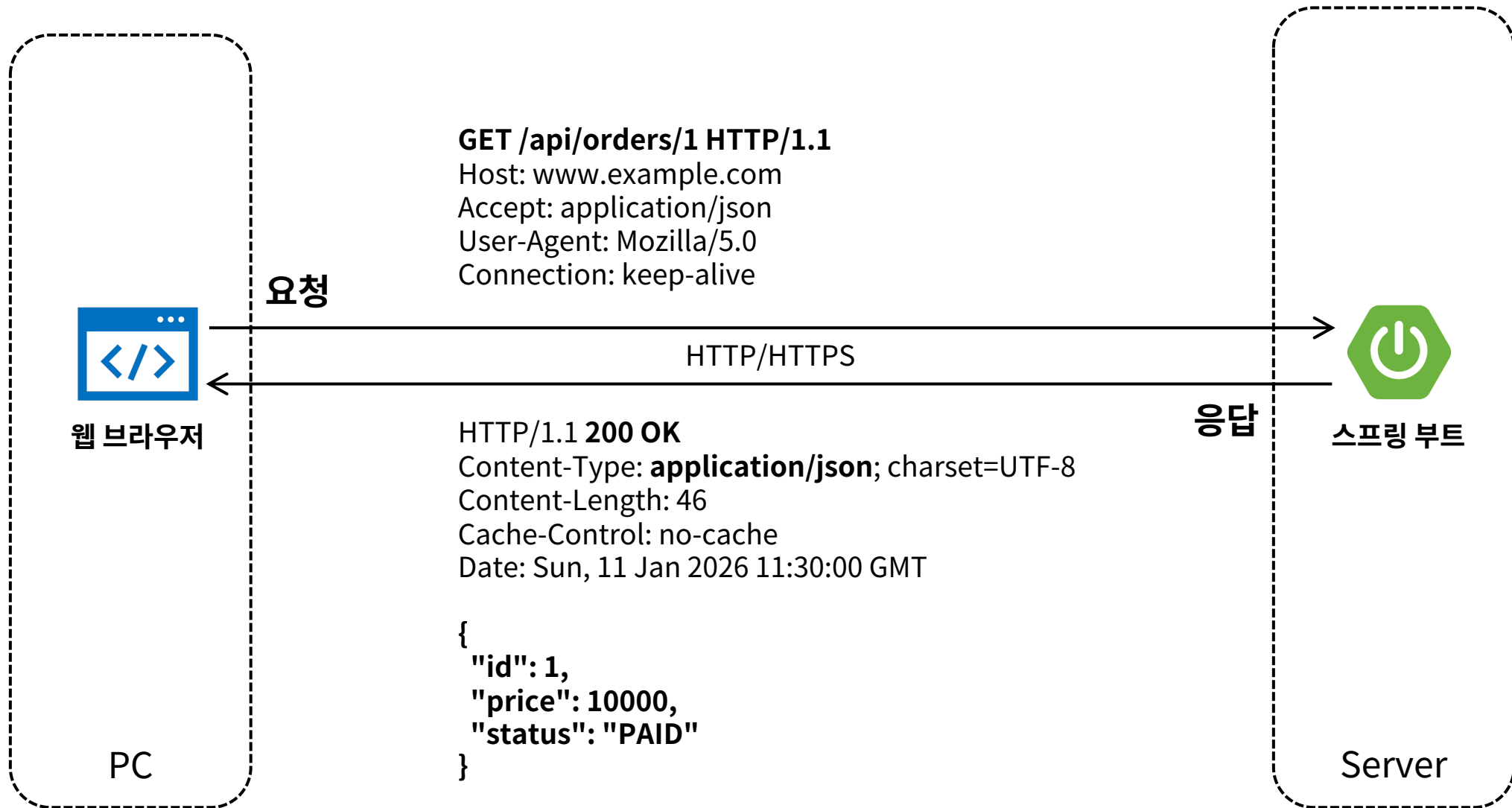
### ③ 무상태(Stateless)

요청 하나하나가 완전한 정보를 포함한다.  
서버는 이전 요청을 기억하지 않는다.

### ④ 표준 HTTP 응답 사용

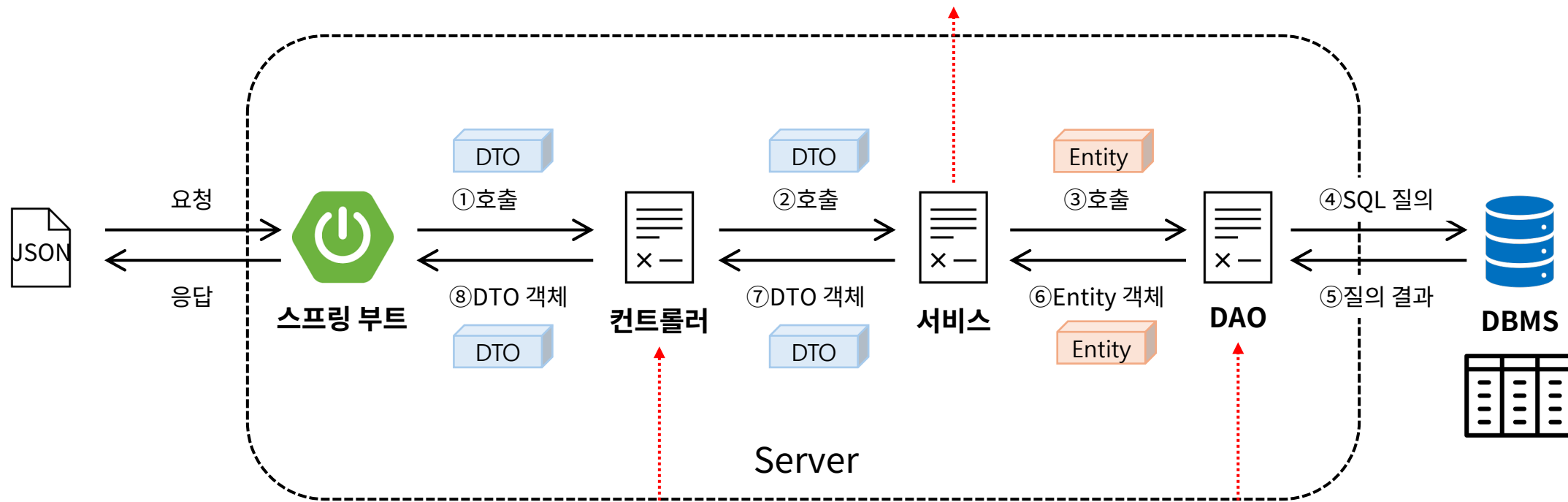
- 200 OK
- 201 Created
- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

# RESTful API 서버 – HTTP 요청/응답



# Spring Boot 백엔드 서버의 REST API 요청/응답

- 비즈니스 규칙과 유스케이스를 구현한다.
- 여러 Repository(DAO)를 조합한다.
- 트랜잭션 경계를 설정한다.



- HTTP 요청을 받는다.
- 요청 데이터를 해석·검증한다.
- 적절한 Service를 호출한다.
- 응답 형태(JSON/XML)를 결정한다.

- DBMS와 통신한다.
- 데이터를 저장·조회·수정·삭제한다.
- 도메인 객체와 DB 사이를 중개한다.

# 주요 기술 - 데이터베이스

유형	대표 제품	구조	장점	단점	사용 사례
관계형	PostgreSQL, MySQL, Oracle	테이블(행·열)	<ul style="list-style-type: none"><li>정합성(ACID) 보장</li><li>SQL 사용,</li><li>트랜잭션 강력</li></ul>	확장성	<ul style="list-style-type: none"><li>금융시스템</li><li>주문/결제 시스템</li><li>전통적인 업무 시스템</li></ul>
Key-Value	Redis, DynamoDB	Key-Value	<ul style="list-style-type: none"><li>속도</li></ul>	구조 제한	<ul style="list-style-type: none"><li>캐시</li><li>세션 저장</li><li>실시간 데이터</li></ul>
Document	MongoDB, CouchDB	JSON	<ul style="list-style-type: none"><li>유연성</li><li>중첩 구조에 강함</li></ul>	JOIN 불가	<ul style="list-style-type: none"><li>콘텐츠 관리</li><li>로그 데이터</li><li>마이크로서비스</li></ul>
Column-Family	Cassandra, HBase	컬럼	<ul style="list-style-type: none"><li>대용량</li><li>분산 처리에 강함</li></ul>	복잡	<ul style="list-style-type: none"><li>빅데이터</li><li>로그/이벤트 저장</li></ul>
Graph	Neo4j	노드 + 엣지	<ul style="list-style-type: none"><li>관계 탐색</li><li>복잡한 연결 분석</li></ul>	학습 비용	<ul style="list-style-type: none"><li>SNS</li><li>추천 시스템</li><li>관계 분석</li></ul>
In-Memory	Redis, Memcached	RAM	<ul style="list-style-type: none"><li>초고속</li></ul>	휘발성	<ul style="list-style-type: none"><li>캐시</li><li>실시간 처리</li></ul>

# 주요 기술 - 데이터 접근 기술

유형	특징	장점	단점	실무 사용
<b>JDBC 직접 사용</b>	<ul style="list-style-type: none"> <li>가장 저수준</li> </ul>	<ul style="list-style-type: none"> <li>완전한 제어 가능</li> <li>원리 명확</li> </ul>	<ul style="list-style-type: none"> <li>보일러플레이트 많음</li> <li>생성성 낮음</li> <li>실수 위험 높음</li> </ul>	낮음
<b>Spring JDBC (JdbcTemplate)</b>	<ul style="list-style-type: none"> <li>JDBC를 스프링이 감싸줌</li> </ul>	<ul style="list-style-type: none"> <li>반복 코드 크게 감소</li> <li>성능 예측 쉬움</li> <li>SQL 직접 통제</li> </ul>	<ul style="list-style-type: none"> <li>SQL과 매핑 코드를 계속 관리해야 함</li> </ul>	중간 (단순 CRUD/배치/레거시)
<b>MyBatis (SQL 매퍼)</b>	<ul style="list-style-type: none"> <li>SQL 직접 작성</li> <li>결과를 객체로 매핑</li> </ul>	<ul style="list-style-type: none"> <li>복잡한 조인/리포팅/튜닝에 강함</li> <li>SQL 통제력 최고 수준</li> </ul>	<ul style="list-style-type: none"> <li>SQL·매퍼 관리 비용</li> <li>CRUD 반복 작업 많아질 수 있음</li> </ul>	높음 (복잡한 쿼리/레거시 DB/리포팅 많은 조직에서 많이 사용)
<b>JPA(ORM) + Hibernate</b>	<ul style="list-style-type: none"> <li>객체(Entity) 중심으로 DB를 다루는 표준</li> </ul>	<ul style="list-style-type: none"> <li>도메인 중심 설계에 유리</li> <li>CRUD 생산성 높음</li> <li>변경 감지 등으로 코드량 감소</li> </ul>	<ul style="list-style-type: none"> <li>러팅 커브</li> <li>성능 튜닝 포인트 존재</li> </ul>	높음 (일반적인 비즈니스 서비스)
<b>Spring Data JPA</b>	<ul style="list-style-type: none"> <li>JpaRepository 기반</li> <li>CRUD, 페이징/정렬, 쿼리 메서드를 자동 제공</li> </ul>	<ul style="list-style-type: none"> <li>CRUD 개발 속도 빠름</li> <li>팀 표준화에 좋음</li> </ul>	<ul style="list-style-type: none"> <li>복잡 쿼리는 별도 전략 필요(QueryDSL, JPQL, MyBatis 등과 연동)</li> </ul>	매우 높음 (REST API 기본 선택지)
<b>QueryDSL</b>	<ul style="list-style-type: none"> <li>자바 코드로 SQL/JPQL을 타입 안전하게 작성</li> </ul>	<ul style="list-style-type: none"> <li>복잡한 검색 조건/동적 쿼리 유지보수에 좋음</li> <li>컴파일 타임 오류 검출</li> </ul>	<ul style="list-style-type: none"> <li>설정/코드 생성 단계 필요</li> <li>팀 숙련도 요구</li> </ul>	중간 ~ 높음 (검색/필터 조건 많은 서비스, Spring Data JPA와 조합)

# 주요 기술 – 인증(Authentication)기술: “누구인가?”

유형	특징	장점	단점	실무 사용
세션 기반 인증	<ul style="list-style-type: none"> <li>로그인 → 서버가 인증</li> <li>세션ID를 쿠키로 전달</li> <li>이후 요청마다 쿠키 자동 전송</li> <li>• <b>Stateful</b></li> </ul>	<ul style="list-style-type: none"> <li>구현 단순</li> <li>보안 개념 이해 쉬움</li> </ul>	<ul style="list-style-type: none"> <li>서버 확장 어려움 (세션 공유 필요)</li> <li>모바일 및 외부 클라이언트와 궁합 <b>×</b></li> </ul>	<ul style="list-style-type: none"> <li>SSR 기반 웹</li> <li>REST API 단독 서버에는 거의 사용 안함</li> </ul>
토큰 기반 인증 (JWT)	<ul style="list-style-type: none"> <li>로그인 → 서버가 토큰 발급</li> <li>클라이언트가 토큰을 저장</li> <li>Authorization: Bearer 요청 헤더로 토큰 전송</li> <li>• <b>Stateless</b></li> </ul>	<ul style="list-style-type: none"> <li>확장성 매우 좋음</li> <li>REST API, 모바일, SPA에 최적</li> </ul>	<ul style="list-style-type: none"> <li>토큰 탈취 시 위험</li> <li>만료·폐기 전략 필요</li> </ul>	<ul style="list-style-type: none"> <li>가장 많이 사용됨</li> <li>사실상 표준</li> </ul>
OAuth 2.0 (권한 위임 프레임워크)	<ul style="list-style-type: none"> <li>사용자를 대신해 “외부 서비스”에 접근할 수 있는 권한을 위임</li> <li>인증 자체라기보다 권한 위임 표준</li> <li>Access Token 사용</li> <li>• <b>Stateless</b></li> </ul>	<ul style="list-style-type: none"> <li>외부 로그인 연동 표준</li> <li>보안 검증이 잘 된 방식</li> </ul>	<ul style="list-style-type: none"> <li>개념 복잡</li> <li>설정 많음</li> </ul>	<ul style="list-style-type: none"> <li>외부 로그인 필수 서비스에 매우 흔함</li> </ul>
OpenID Connect (OIDC)	<ul style="list-style-type: none"> <li>OAuth 2.0 위에 인증 레이어 추가</li> <li>ID Token(JWT) 제공</li> <li>“로그인”을 표준화</li> <li>사용자 식별 정보 포함</li> <li>• <b>Stateless</b></li> </ul>	<ul style="list-style-type: none"> <li>OAuth + 인증 문제 해결</li> <li>SSO에 최적</li> </ul>	<ul style="list-style-type: none"> <li>OAuth보다 더 복잡</li> </ul>	<ul style="list-style-type: none"> <li>엔터프라이즈 및 SSO 환경에서 증가 중</li> </ul>
API Key 방식	<ul style="list-style-type: none"> <li>요청 헤더에 고정 키 포함 예) X-API-KEY: abc123</li> <li>단순한 인증</li> <li>• <b>Stateless</b></li> </ul>	<ul style="list-style-type: none"> <li>구현 매우 간단</li> <li>서버 간 통신에 적합</li> </ul>	<ul style="list-style-type: none"> <li>사용자 단위 인증 <b>×</b></li> <li>노출 시 위험</li> </ul>	<ul style="list-style-type: none"> <li>서버-to-서버</li> <li>내부 API</li> </ul>

# 주요 기술 – 인가·보안기술

## 인가(Authorization)기술

- ✓ Role 기반(RBAC)
  - `@PreAuthorize("hasRole('ADMIN')")`
  - 가장 일반적, 단순 명확
- ✓ Scope / Permission 기반
  - `@PreAuthorize("hasAuthority('ORDER:READ')")`
  - OAuth / API 권한 모델과 잘 맞음
  - 세밀한 권한 제어 가능

## 전송 보안

- ✓ HTTPS/TLS
  - 데이터 암호화
  - 토큰 탈취 방지

# 주요 기술 – Spring Security

- ✓ 인증·인가의 핵심 프레임워크
- ✓ Session, JWT, OAuth2, OIDC, API Key 등 지원
- ✓ Spring Boot 보안의 표준

## 실무

일반적인 REST API 서버	Spring Security + Role / Permission 인가 + JWT 인증(+ OAuth2/OIDC)
외부 로그인 포함 서비스	Spring Security + JWT 인증
SSR 웹 애플리케이션	Spring Security + Session / Cookie
서버 간 통신	API Key or OAuth2 Client Credentials



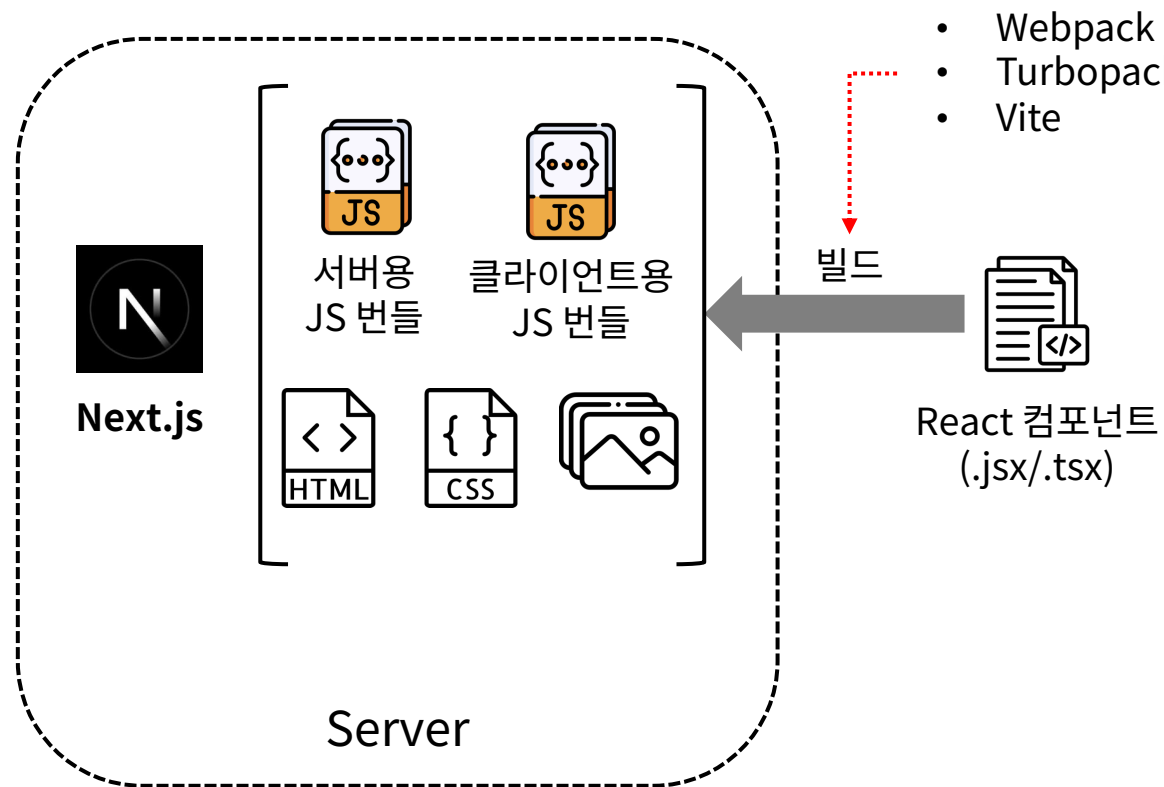
# 백엔드 서버 구축 - 기타 플랫폼

언어	런타임 / 서버	프레임워크	대표 프레임워크 특징	실무 사용
JavaScript	Node.js	<ul style="list-style-type: none"> <li>Express</li> <li><b>NestJS(대표)</b></li> <li>Fastify</li> </ul>	<ul style="list-style-type: none"> <li>모듈화/의존성 주입, 구조화된 아키텍처, 테스트 용이</li> <li>TypeScript 기반 엔터프라이즈 급 구조를 갖추</li> <li>Node.js 생태계와도 잘 어울림</li> <li>Express.js에 비해 <b>구조와 확장성이 커뮤니티에도 높게 평가</b> 받고 있음</li> <li>대규모 API, 마이크로서비스, 팀 개발에 많이 쓰임</li> </ul>	★★★★☆
TypeScript	Node.js	<ul style="list-style-type: none"> <li><b>NestJS</b></li> </ul>		★★★★★
Python	WSGI / ASGI	<ul style="list-style-type: none"> <li>Django REST</li> <li><b>FastAPI(대표)</b></li> </ul>	<ul style="list-style-type: none"> <li>타입 힌트를 통한 빠른 개발, 자동 OpenAPI 문서화 지원</li> <li><b>Django 보다 가볍고</b> API 중심 서비스에서 빠르게 성장중</li> <li>AI/ML API에서 클라우드 네이티브까지 두루 채택됨</li> </ul>	★★★★☆
Go	내장 HTTP 서버	<ul style="list-style-type: none"> <li><b>Gin(대표)</b></li> <li>Echo</li> <li>Fiber</li> </ul>	<ul style="list-style-type: none"> <li>간결·고성능, Goroutine 기반 동시성</li> <li><b>경량 API 서버와 고성능 마이크로서비스에</b> 자주 쓰임</li> <li>비교적 <b>단순하면서도 퍼포먼스가 필요한 시스템에</b> 적합</li> </ul>	★★★★☆
C#	.NET (Kestrel)	<ul style="list-style-type: none"> <li><b>ASP.NET Core</b> Web API</li> </ul>	<ul style="list-style-type: none"> <li>크로스 플랫폼, 고성능 서버, 풍부한 툴링</li> <li><b>MS 생태계와의 연계가 강점</b></li> <li>엔터프라이즈/클라우드 API에 최적화됨</li> <li>Windows/Cloud 서비스에서 강세</li> </ul>	★★★★☆
PHP	PHP-FPM	<ul style="list-style-type: none"> <li><b>Larabel</b></li> </ul>	<ul style="list-style-type: none"> <li>깔끔한 문법, ORM, 개발 생산성</li> <li>중소형 웹/백오피스 서비스에서 여전히 강력한 생산성 제공</li> </ul>	★★★☆☆

# Next.js 기반 프론트엔드 서버 구축

# Next.js 프론트엔드 서버

- **프론트엔드 서버**는 사용자에게 보여지는 **화면(UI)**를 전달하고, 브라우저에서 실행될 클라이언트 코드를 제공하는 서버다.
- **Next.js** 서버는 프론트엔드 서버 역할을 수행할 수 있는 **React 기반 웹 프레임워크**이다.
- SSR·CSR·SSG를 상황에 맞게 혼합해 사용할 수 있다.

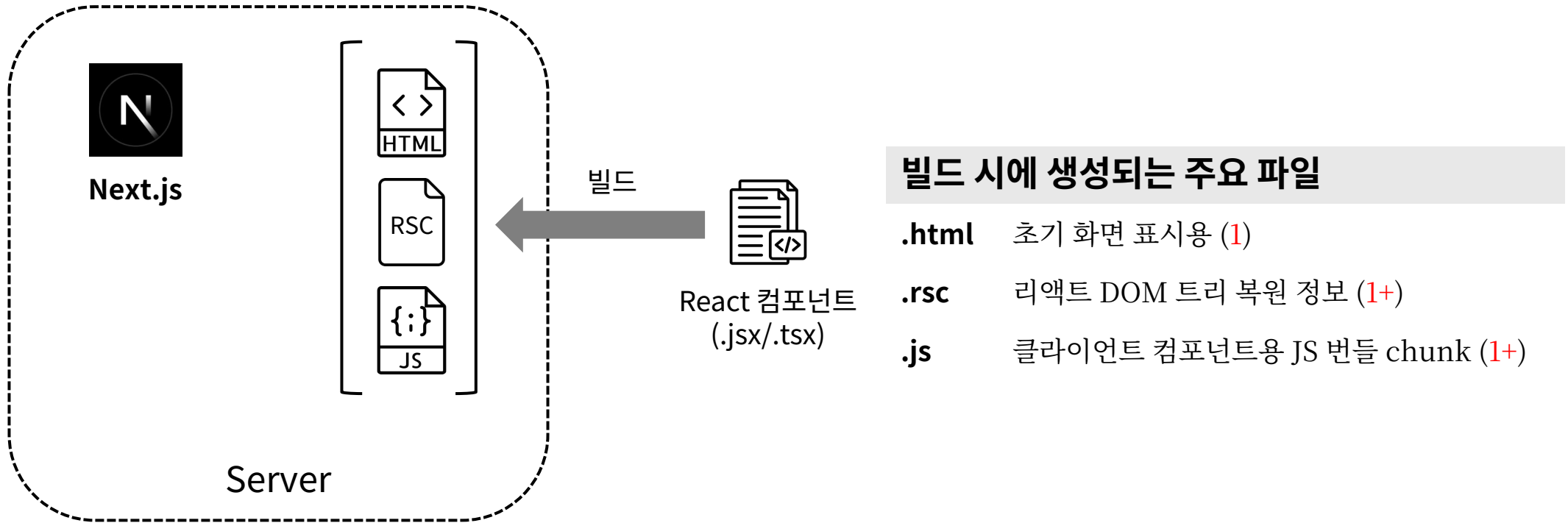


## React 빌드

- React 코드(TypeScript, JSX) → JavaScript 생성
- 최신 JS → 타킷 환경용 JS로 변환
- **JS 번들 생성**: 서버/클라이언트/공유 컴포넌트 분리
- SSG/ISR 페이지 → **HTML 생성**
- **최적화 수행**: 코드 압축, CSS 최적화 등

# Next.js 서버 – SSG

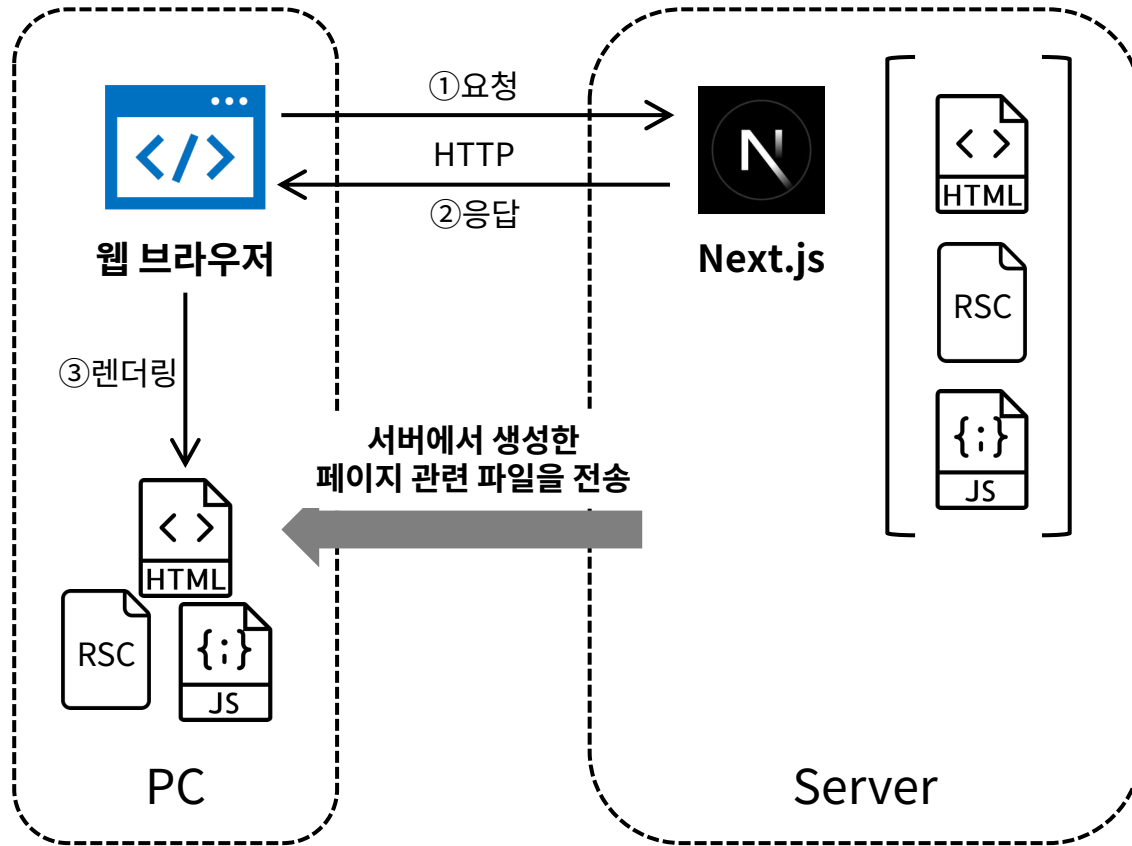
- 정적 사이트 생성(Static Site Generation) 방식은 프로젝트를 빌드할 때 미리 HTML 및 관련 파일을 생성한다.



## \* .rsc 파일

RSC(React Server Component)의 실행 결과를 **리액트 전용 포맷(Flight Protocol)**으로 직렬화한 “데이터 페이로드” 이다. 서버 컴포넌트의 렌더링 결과, 컴포넌트 트리 구조, 직렬화된 props, 클라이언트 컴포넌트 참조, Suspense/Streaming 메타 정보가 들어 있다. 모든 마크업 생성을 브라우저에게만 맡기지 말고, “서버에서 할 수 있는 일은 서버에서 끝내자”라는 생각에서 도입된 방식이다.

# Next.js 서버 - SSG 실행 흐름



## 실행 흐름

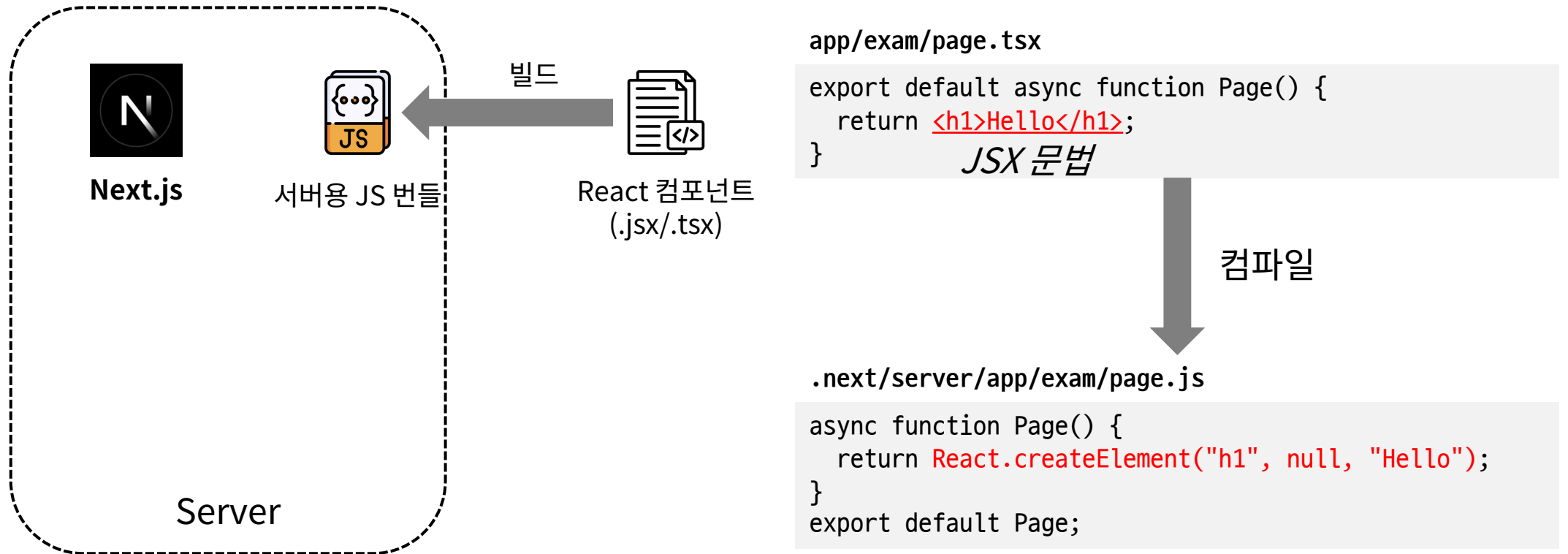
- ❶ 페이지 요청  
→ [서버] 미리 생성해 둔 HTML 응답
- ❷ 화면 표시  
→ HTML 파싱  
→ RSC Payload 다운로드(.rsc)
- ❸ 페이지와 관련된 JS 번들 다운로드(.js)
- ❹ RSC Payload 해석  
→ 서버 컴포넌트 트리 재구성
- ❺ React Hydration  
→ HTML과 React 트리를 연결
- ❻ 페이지 렌더링 완성  
→ 사용자 상호작용 가능

### \* Hydration

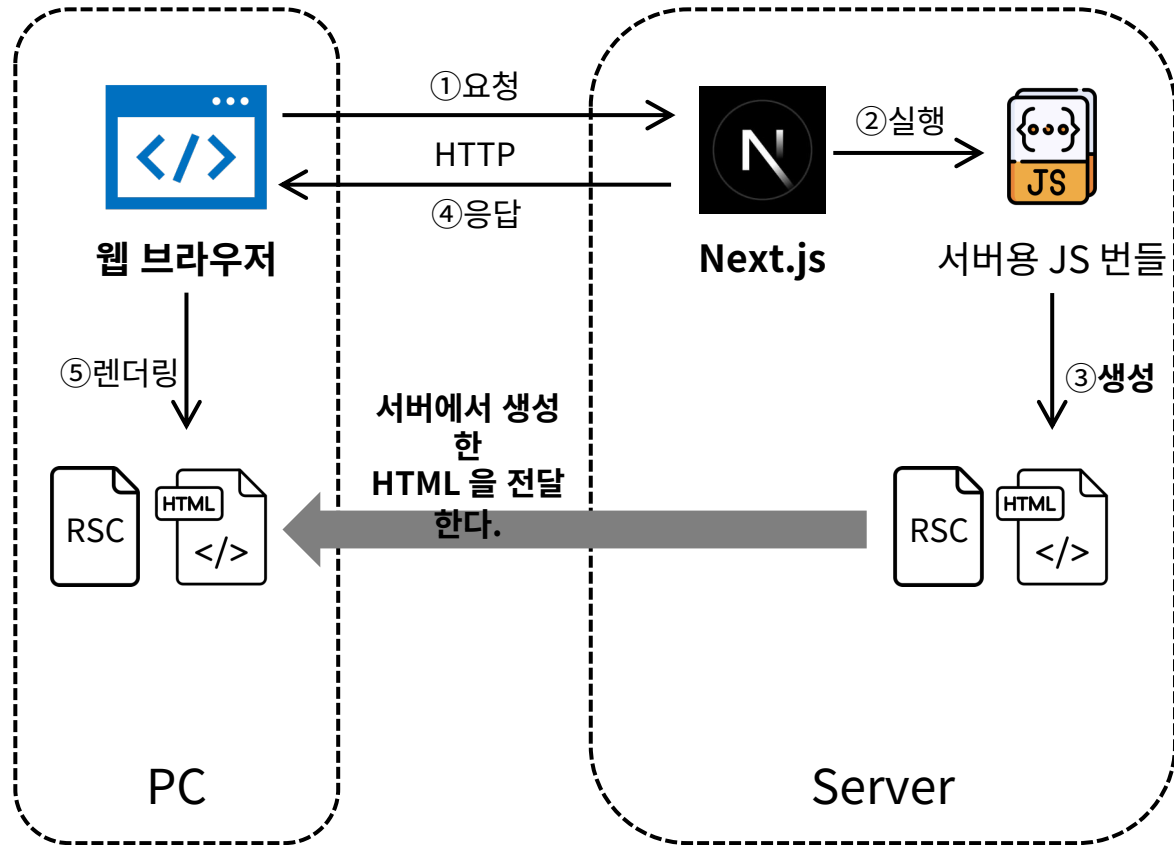
사전적 의미 → 물을 공급하다. 건조한 것에 수분을 더해 살아 있게 만들다. 예) 피부 hydration → 수분을 공급해 탄력 회복  
즉 이미 만들어진 **HTML**(건조한 상태)에 **React의 상태·이벤트·로직**을 주입해서 “**살아있는 앱**”으로 만드는 과정을 뜻한다.

# Next.js 서버 – SSR

- 서버 사이드 렌더링(Server-Side Rendering) 방식은 프로젝트를 빌드할 때 서버 실행용 JS 번들로 변환한다.
- 즉, TypeScript와 JSX 문법으로 작성된 리액트 코드를 순수 JavaScript 코드로 컴파일 한다.



# Next.js 서버 - SSR 실행 흐름

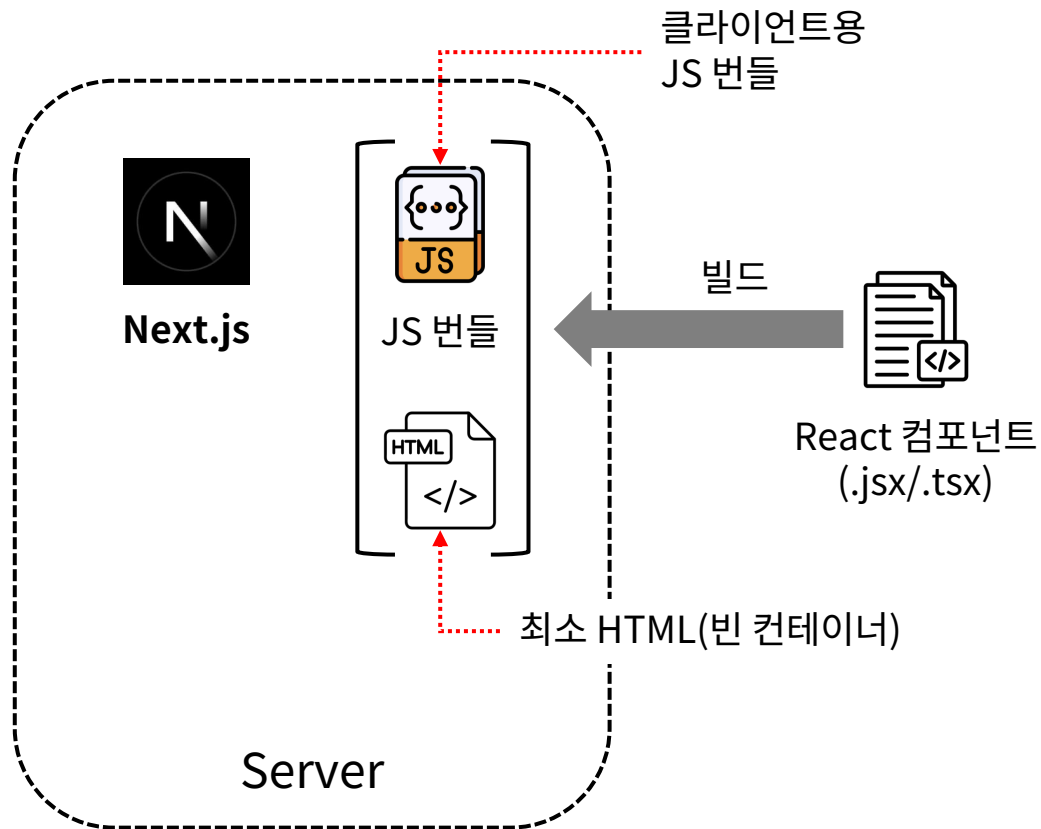


## 실행 흐름

- ① 페이지 요청
- ② [서버] 서버용 JS 번들 실행
  - HTML 및 RSC Payload 생성
  - 브라우저로 응답
- ③ HTML 표시
  - RSC Payload 해석
  - 서버 컴포넌트 트리 복원
- ④ 클라이언트용 JS 번들 로드
- ⑤ React Hydration
  - HTML과 React 트리를 연결
  - 이벤트 핸들러. 부착
  - 상태 활성화
- ⑥ 페이지 렌더링 완성
  - 사용자 상호작용 가능

# Next.js 서버 – CSR

- 클라이언트 사이드 렌더링(Server-Side Rendering) 방식은 프로젝트를 빌드할 때 클라이언트용 JS 번들로 패키징한다.
- 즉, TypeScript와 JSX 문법으로 작성된 리액트 코드를 순수 JavaScript 코드로 컴파일 한다.



app/exam/page.tsx

```
export default async function Page() {  
  return <h1>Hello</h1>;  
}
```

JSX 문법

컴파일

.next/static/chunks/framework.js

.next/static/chunks/main.js

.next/static/chunks/app/exam/page.js

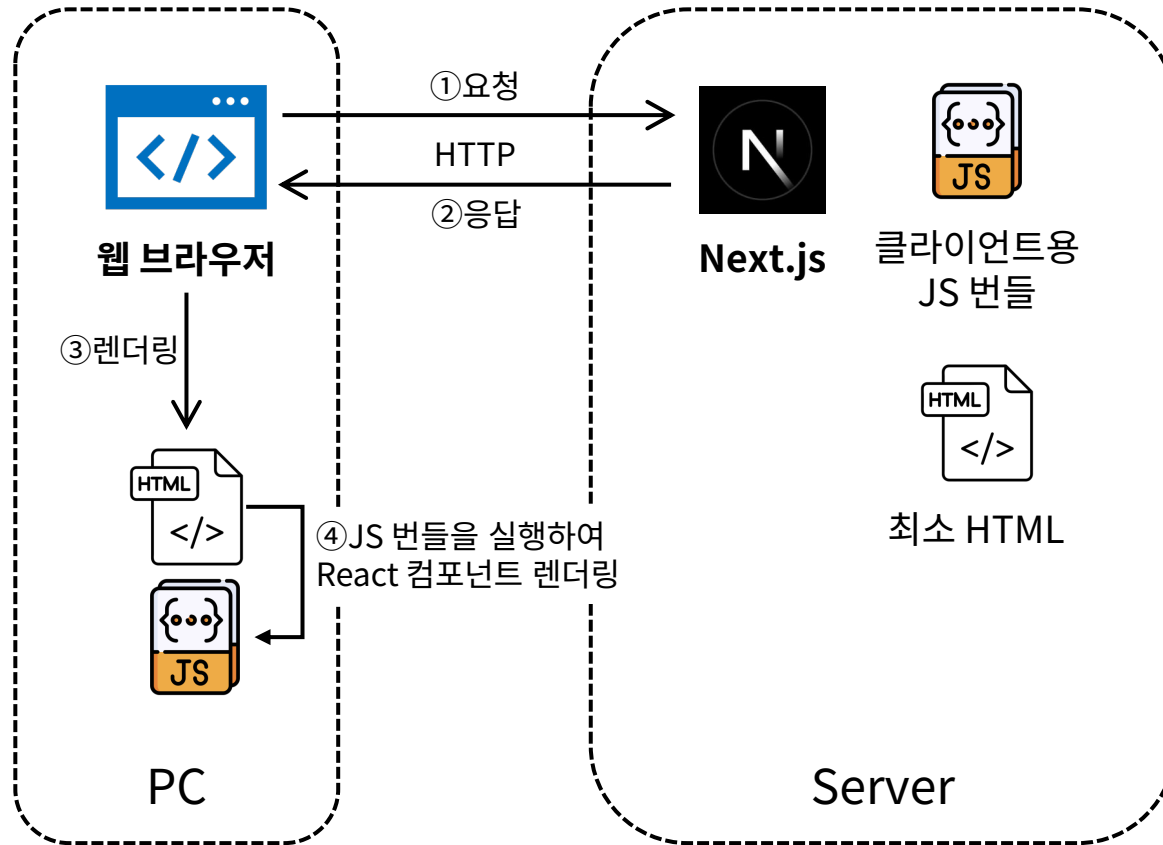
.next/server/app/exam.html (깍대기 수준)

빈 HTML 예: exam.html

```
<div id="__next"></div>
```



# Next.js 서버 - CSR 실행 흐름



## 실행 흐름

- ① 페이지 요청
- ② [서버] 빌드 시 생성해 둔 최소 HTML 응답
- ③ JS 번들 다운로드
  - `/_next/static/chunks/framework.js`
  - `/_next/static/chunks/exam/page.js`
- ④ JS 실행
  - .tsx 에서 컴파일된 JS가 처음 실행
  - React 앱 부트스트랩(JS 엔트리 포인트 실행)
- ⑤ React 렌더링
  - 컴포넌트 함수 실행
  - Virtual DOM 생성
- ⑥ 실제 HTML DOM 생성 및 삽입

# Next.js 서버 – SSG vs SSR vs CSR

항목	SSG	SSR	CSR
JS 번들 생성 (TypeScript/JSX → 순수 JS)	빌드 타임	빌드 타임	빌드 타임
HTML 생성	빌드 타임 → HTML 파일 생성	요청 → 서버에서 HTML 생성	껍데기 HTML 파일 → 브라우저 에서 추가 HTML 생성
RSC Payload 생성	✓	✓	✗
초기화면	즉시	즉시	JS 실행 후
SEO	매우 유리	유리	불리

# 주요 기술 – React UI 라이브러리

항목	특징	실무 포지션
Material UI (MUI)	<ul style="list-style-type: none"><li>가장 보편적인 선택지, 기업·실무 표준</li><li>구글 Material Design 기반, 컴포넌트 수가 매우 많음</li><li>테마/다크모드/접근성 잘 갖춰짐</li></ul>	<ul style="list-style-type: none"><li>대기업·엔터프라이즈 프로젝트에서 압도적</li><li>SSR 지원 안정적</li><li>문서 레퍼런스 풍부</li></ul>
Tailwind CSS + shadcn/ui	<ul style="list-style-type: none"><li>2024 ~ 2025년 가장 빠르게 확산 중인 조합</li><li>Tailwind → 유틸리티 기반 CSS</li><li>Shadcn/ui → 라이브러리가 아니라 복사해서 쓰는 컴포넌트</li><li>디자인 자유도 매우 높음, 불필요한 추상화 없음</li></ul>	<ul style="list-style-type: none"><li>스타트업·모던 서비스에서 폭발적 증가</li><li>App Router, Server Components와 잘 맞음</li><li>번들 가볍고 성능 우수</li></ul>
Ant Design (AntD)	<ul style="list-style-type: none"><li>관리자·백오피스·대시보드 최강자</li><li>완성도 높은 컴포넌트, 테이블·폼·모달·차트에 강함</li><li>디자인 커스터마이징은 다소 무거움</li></ul>	<ul style="list-style-type: none"><li>중국/아시아권 + 엔터프라이즈에서 매우 흔함</li><li>SSR 지원 가능</li><li>관리자 화면이면 AntD</li></ul>
Chakra UI	<ul style="list-style-type: none"><li>개발자 경험(DX)과 접근성 중심</li><li>props 기반 스타일링, 접근성 기본 내장</li><li>학습 곡선 완만</li></ul>	<ul style="list-style-type: none"><li>중소규모 서비스에 적합</li><li>SSR 안정</li><li>빠르게 무난하게 만들기 좋음</li></ul>
Mantine	<ul style="list-style-type: none"><li>MUI와 shadcn의 중간 성향</li><li>컴포넌트 + 훅 제공</li><li>디자인 완성도 높음, 커스터마이징 유연</li></ul>	<ul style="list-style-type: none"><li>유럽·개인 프로젝트에서 점점 증가</li><li>SSR 지원</li><li>아직은 MUI/AntD보다 점유율 낮음</li></ul>