**웹개발 기초**

웹개발은 브라우저와 서버 사이의 요청과 응답을 잇는다. 따라서 네트워크와 브라우저, 서버를 비롯한 컴퓨팅 환경 전반에 대한 이해가 필요하다. 개인이 브라우저와 네트워킹, 서버 개발 환경에 관한 지식을 하나부터 열까지 익히고 바닥부터 쌓아 올리는 일은 불가능에 가깝다. 그렇기 때문에 현대 웹 개발은 많은 지식이 축적, 추상화된 프로토콜, 툴, 프레임워크를 활용해 이뤄진다.

**웹 애플리케이션 서버(WAS)와 웹 서버**

웹 서버는 WAS와 웹 서버로 구분된다. 웹 애플리케이션 서버에는 서버 사이드 프로그램이라고 불리는 방식의 코드들이 동작한다. 클라이언트가 특정 데이터에 대한 동적인 반응을 원하면 WAS 서버는 해당 요청을 서버 사이드에 위치한 프로그램을 실행시켜 동적으로 변환한 후 응답한다. 자바는 서블릿과 JSP를 사용해 WAS 서버를 구성한다. 일반적인 웹 서버는 HTML, CSS, 이미지 파일 등의 변환이 불필요한 정적 파일을 보관하고 있다가 요청이 오면 전달한다.

자바는 이러한 웹개발 환경에 대응해 JavaEE라는 기술 스펙을 제공한다. 동시 요청 문제, 서버 장애, 데이터 전송 최적화, 분산 처리 등의 문제를 해결하기 위해 등장한 이 기술에서 일반적으로 가장 기본적인 기술이 서블릿과 JSP이다.

**서블릿**

서버에서 동적인 요청을 처리할 때 서블릿을 사용한다. 서블릿 클래스라고 불리는 클래스들은 고객의 요청에 대응하기 위한 데이터와 메서드를 가지고 있다. 이러한 서블릿 클래스는 해당 서블릿 클래스를 실행시켜줄 수 있는 프로그램을 통해 API로 기능한다. 이때 해당 서블릿 클래스의 코드를 클라이언트의 요청에 따라 실행시켜주는 프로그램이 톰캣이다. 톰캣과 같은 프로그램은 서블릿 컨테이너 또는 서블릿 엔진으로 불린다. 서블릿 컨테이너는 고객의 요청과 웹 애플리케이션 사이에서 서블릿 클래스나 JSP 파일의 실행을 담당한다. JSP는 근본적으로는 서블릿과 비슷하지만 좀 더 HTML에 특화된 방식으로 코드를 작성할 수 있도록 포매팅되어 있다. 이는 용도에 의한 구분으로 서블릿으로는 애플리케이션 로직을 처리하고 JSP로는 화면을 개발하기 위함이다.

일반적으로 자바 코드로 작성된 프로그램은 메인 함수를 통해 실행된다. 그러나 서블릿 클래스, JSP는 서블릿 컨테이너를 통해 실행되기 때문에 일반적인 자바 프로그램과 다음과 같은 차이가 생긴다.

1. 사용자가 아닌 컨테이너가 객체를 생성하거나 호출한다.
2. 서블릿 클래스에서 생성한 객체를 컨테이너가 관리한다.
3. 서블릿/JSP 코드의 개발은 기본적인 자바 API와 더불어 서블릿 API도 같이 사용해야 한다.

**JSP**

Java Server Page의 약자이다. HTML이 메인이고 자바 코드가 서브인 자바코드이다. JSP도 결국에는 서블릿 코드로 변환되어 컴파일되고 실행되기 때문에 자바 코드이다. 필요한 순간에 자바 파일로 생성되고 이를 컴파일 해 클래스 파일로도 만들어진다. 개발자의 목적과 편의성에 따라 순수 서블릿 코드와 JSP로 구분된다. JSP는 GET/POST 방식의 호출을 구분하지 않기 때문에 POST 방식으로 접근해야 하는 JSP도 GET 방식으로 호출할 수 있다는 문제가 있다. 또 JSP를 다른 JSP파일로 변경할 경우 주소의 변경 사항이 모든 액션 코드에 반영되어야 해서 유지 보수의 문제도 있다. 따라서 최근 JSP는 다음과 같이 제한적인 용도로 사용한다.

1. 쿼리 스트링이나 파라미터를 처리하지 않는다. – JSP 대신 서블릿을 통해 처리한다.
2. JSP는 입력 화면을 구성하거나 처리 결과를 보여주는 용도로만 사용한다.
3. 브라우저는 직접 JSP 경로를 호출하지 않고 서블릿 경로를 통해 JSP를 보는 방식으로만 사용한다.

위와 같은 문제를 해결하기 위해 나온 방식이 웹 MVC 방식이다. JSP는 결과만 출력하고 처리는 서블릿을 이용하는 방식으로 변화한 것이다.

**Web MVC 방식**

JSP는 GET/POST 구분 불가, URL 변경에 유연하지 못하다는 등의 단점이 있었다. 2000년대 중반부터 이런 단점을 극복하기 위해 MVC 패턴을 활용한 개발이 일반화되었다.

MVC는 위 문제를 해결하기 위해 다음과 같은 구조를 활용한다.

* **브라우저의 요청**
* **서블릿(컨트롤러)이 필요한 데이터(를 제공하는 객체 -> 모델) 처리 후 JSP로 전달**
* **JSP(뷰)는 EL을 이용해 최종 데이터 출력**

JSP를 통해 생성한 화면은 톰캣이 브라우저로 전송한다.

**PRG 패턴(Post-Redirect-GET)**

포스트와 리다이렉트를 결합해 활용하는 패턴이다. 사용자가 컨트롤러에 원하는 작업을 POST로 처리하기를 요청하면 POST를 컨트롤러에서 처리하고 브라우저는 다른 경로로 이동(GET)하라는 응답(리다이렉트)를 주게 된다. 브라우저는 GET 방식으로 이동하게 된다. 개발자 도구에서 응답 헤더의 location 경로에 해당한다.

**HttpServlet**

HttpServlet은 Http 메서드를 처리하는 메서드인 doGet, doPost등의 메서드를 제공한다. 개발자들은 이 메서드를 오버라이트 해서 각 메서드에 대한 반응을 작성할 수 있다. HttpServlet을 상속한 클래스는 톰캣과 같은 WAS의 내부에서 자동으로 객체를 생성하고 관리해 개발자가 신경 쓸 필요가 없다. HttpServlet은 또한 멀티 스레드에 의해 동시에 실행될 수 있도록 개발되기 때문에 개발자가 동시 요청에 대한 부담을 덜 수 있다.

HttpServlet은 Servlet, Serializable, ServletConfig을 구현하는 GenericServlet이라는 추상 클래스를 상속해 구현된다. 제네릭한 서블릿은 HttpRequest와 HttpResponse 타입이 나닌 일반적인 서블릿리퀘스트, 서블릿리스폰스라는 타입의 매개변수를 인자로 받는다.

라이프 사이클은 단순화하면 요청을 처리해 응답하는 과정이다. 이 과정에 톰캣이 개입해 객체의 생성과 관리를 돕는다. 브라우저가 특정 경로를 호출하면 톰캣은 서블릿에서 이를 처리해 해당 서블릿 클래스를 로딩하고 객체를 생성한다. 이때 init()이라는 메소드를 실행해 서블릿 객체가 동작하기 전에 수행할 일들을 처리할 수 있다. 서블릿 객체는 브라우저의 요청에 대한 정보를 분석해 GET/POST 등의 정보와 파라미터(쿼리 스트링의 내용)를 HttpServletRequest의 파라미터로 전달받는다. 서블릿 내부에서 doGet(), doPost() 등의 메소드를 실행하고 동일한 호출이 있을 때는 동일한 객체 하나만을 사용해 이를 처리하게 된다. 톰캣이 종료될 때는 서블릿의 destroy()메서드를 실행한다.

서블릿 객체는 경로에 맞게 하나만 생성되고 호출시에는 자동으로 doGet이나 doPost 등을 호출한다.

**모델**

모델은 컨트롤러를 담당하는 서블릿과 뷰를 담당하는 JSP를 제외한 나머지 부분이다. 크게 서비스(비즈니스 로직 처리) 계층, 영속(데이터 처리) 계층으로 분리된다.

**DTO(Data Transfer Object)**

3티어와 같이 계층을 분리하는 경우 반드시 계층이나 객체들 간에 데이터 교환이 이루어진다. 한 개 이상의 데이터를 전달할 경우가 많기에 묶어서 하나의 객체로 전달하게 되는데 이 객체를 DTO(Data Transfer Object)라고 한다.

특별한 규약이나 제약은 없지만 대부분 Java Beans 형태로 구성하는 경우가 많다. Java Beans는 다음과 같은 형식으로 구성된다.

* 생성자가 없거나 반드시 파라미터가 없는 생성자 함수를 가지는 형태
* 멤버 변수는 private로 작성
* Getter/setter의 제공