Servlets & Jsp

빌드 도구

- 실제 프로젝트 환경은 수십 개의 클래스와 여러 외부 라이브러리로 구성된다.
  이에 프로젝트의 소스를 체계적으로 컴파일하고 관련 라이브러리의 버전이나 종속성 관리를 쉽게 도와줄 방법이 필요해졌다.

- 이를 위해서 Java 빌드 도구가 생겨나게 되었다.
- 가장 오래된 빌드 도구로는 Ant가 있으며, 2004년에는 Maven이 등장했다.
  이후 오랜 기간 Maven은 거의 모든 프로젝트에서 사용하는 빌드 도구가 되었고,
  스프링 프레임워크 개발에서 기본 빌드 도구로 활용되었다.

- 시간이 지남에 따라 좀 더 유연하면서 복잡한 처리를 쉽게 하기 위해 2012년 Gradle이 나오게 되었다.

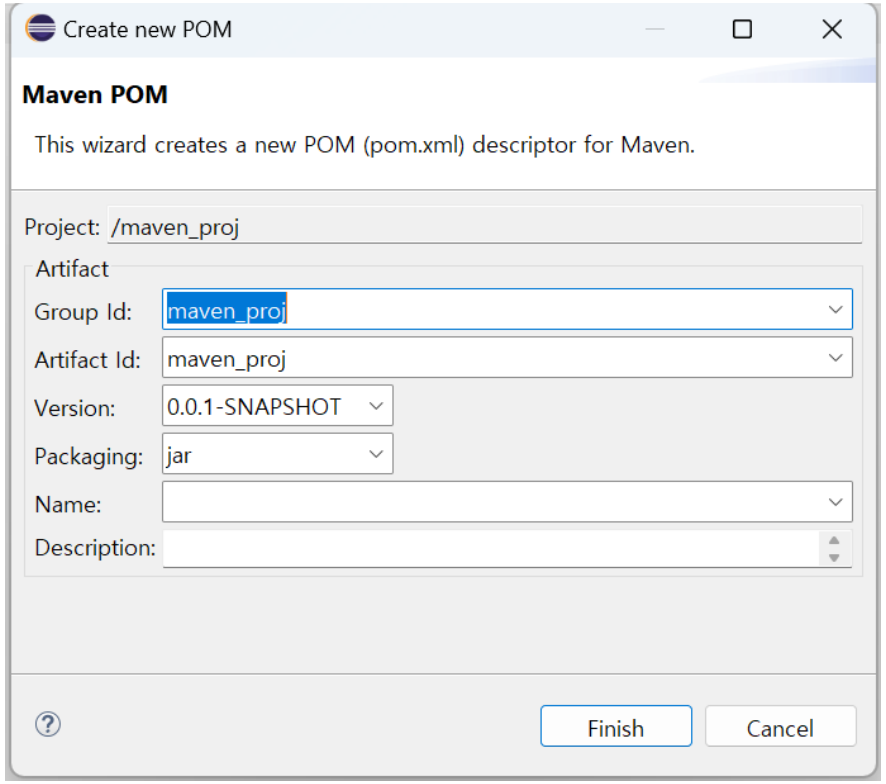
- Maven : 빌드 설정을 pom.xml 파일에 작성하게 된다.
- Gradle : Groovy라고 하는 JVM 기반 언어를 통해 프로그램 구조로 설정한다. (Maven에 비해 10~100배 성능 향상)

Maven 프로젝트 생성하기

- Dynamic Web Project 생성 [프로젝트 명: maven_proj]

- 프로젝트에서 마우스 우클릭 후, [Configure] [Convert to Maven Project]

- Group Id : 프로젝트 고유 식별자로 기본 패키지 이름 규칙에 따라 작성
- Artifact Id : 생성되는 jar(war) 파일의 이름으로 소문자로 작성
- Version : 버전 관리 번호
- Packaging : 빌드 산출물 형태 (jar 또는 war)



Maven 프로젝트 생성하기

- 자동으로 기존 프로젝트 구조에 몇몇 폴더 구조가 추가되며, pom.xml이 생성되어 오픈되어 있다.

- Maven을 사용하면 필요한 라이브러리를 쉽게 추가할 수 있다.

  https://mvnrepository.com/

Maven 프로젝트 생성하기

- JSTL과 LOMBOK 라이브러리를 추가해보자.

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.34</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Maven 프로젝트 생성하기

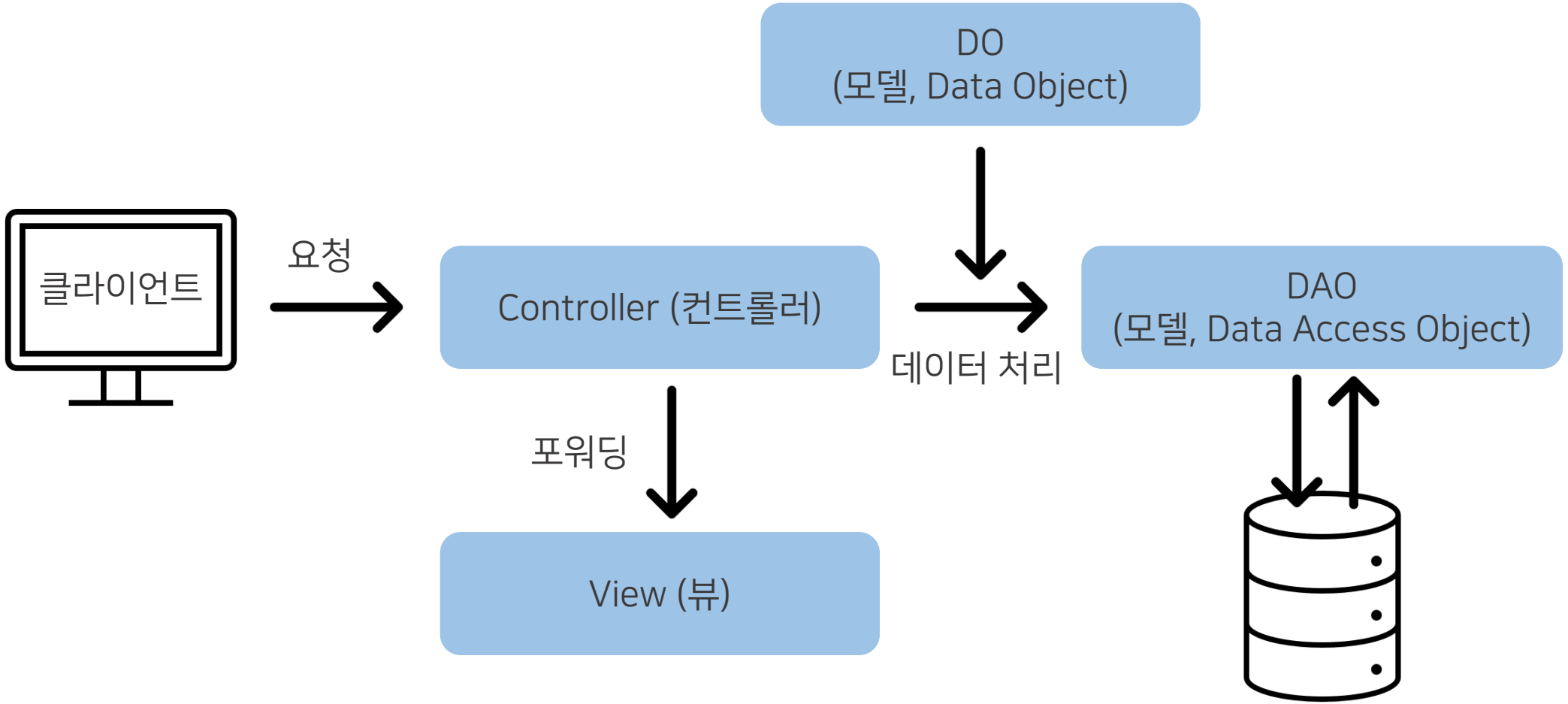
MVC 패턴

- MVC 패턴이란 Model - View - Controller의 약어로 주로 GUI 기반 애플리케이션 개발에 사용되는 디자인 패턴이다.
- 현재는 백엔드 기반의 웹 애플리케이션 개발의 기본 모델이 되었다.

- 모바일 앱 및 프론트앤드 기반의 웹 개발이 늘어나면서
  MVP(Model - View _ Presenter), MVVM(Model -View - ViewModel)과 같은 패턴도 사용되고 있다.

- 패턴들의 공통적인 목적은 화면과 데이터 처리를 분리하여 코드 간의 종속성을 줄이는 것에 있다.
- 즉, 구성 요소 간 역할을 명확하게 하여 코드를 쉽게 분리하고 협업을 용이하게 만드는 것이다.

MVC 패턴


DO
(모델, Data Object)
클라이언트
요청
Controller (컨트롤러)
데이터 처리
DAO
(모델, Data Access Object)
포워딩
View (뷰)

MVC 패턴

- 모델(Model) : 데이터를 처리하는 영역.
- 일반적으로 DB 연동을 위한 DAO 클래스와 데이터 구조를 표현하는 DO, 엔티티 클래스 등으로 구성된다.

  - DAO (Data Access Object)
    - 데이터베이스와의 상호작용을 담당하는 클래스
    - CRUD(Create, Read, Update, Delete) 작업을 수행
    - 데이터베이스의 데이터에 접근하고 조작하는 역할

  - DO (Data Object) / 엔티티(Entity) 클래스
    - 데이터베이스의 테이블과 매핑되는 클래스
    - 각 필드는 데이터베이스의 컬럼에 대응되며, 데이터 구조를 표현

- JPA를 사용하는 경우, DAO는 생략되거나 구현범위가 축소될 수 있으며,
  비즈니스 로직에 따라 DTO를 추가로 사용하기도 한다. (주로 서비스 영역)

MVC 패턴

- 뷰(View) : 화면 구성을 담당하는 영역.
- 뷰는 기본적으로 모델, 컨트롤러와의 종속성이 없도록 구현해야 한다.

- 뷰에서는 데이터를 직접 가져오는 방식보다, 주어진 데이터를 출력하는 용도로만 사용하는 것이 바람직하다.

- 뷰 영역의 구현을 위해 뷰 템플릿 엔진이 사용되며, 그 중 대표적인 것이 JSP이다.
- 스프링 프레임워크에서는 주로 타임리프, 프리마커 등이 대표적인 뷰 템플릿 엔진이다.

MVC 패턴

- 컨트롤러(Controller) : MVC 패턴의 핵심으로 모든 사용자 요청의 중심에 위치하는 영역.

- 사용자의 요청은 특정 뷰에 바로 전달되지 않고, 컨트롤러를 통해 전달된다.
- 컨트롤러는 사용자의 요청에 따라 모델을 통해 데이터베이스와 연동하여 데이터를 처리하고 뷰에 결과를 전달한다.

컨트롤러 설계

- 컨트롤러를 설계하기 위해서는
  클라이언트의 요청을 단일 컨트롤러에서 처리할 것인지, 개별 컨트롤러에서 처리할 것인지 결정해야 한다.

  /member/add
  /member/delete

- 위의 두 URL이 모두 GET 방식으로 요청이 들어온다면, 하나의 서블릿에서는 동일한 doGet( ) 메소드가 호출되기 때문에 각각의 URL 요청을 별도의 서블릿으로 구현해야 한다.

- 하지만 같은 단위의 업무라면 하나의 컨트롤러에서 처리하도록 하는 것이 구조적으로 관리가 쉽다.
- 사용자의 요청을 구분해서 하나의 서블릿에서 처리하기 위해서는 아래 두 가지의 방법을 이용할 수 있다.
  - URL의 파라미터를 이용
  - 프론트 컨트롤러 구현

컨트롤러 설계 - URL 파라미터를 이용

- URL 파라미터를 활용하는 방법은 앞서 작성한 URL 요청을 별도의 파라미터를 두어, 구분하는 방식이다.

  /member?action=add
  /member?action=delete

- 컨트롤러에서는 request.getParameter("action")을 통해 분기 처리를 진행할 수 있게 된다.
- 비교적 간단한 방법이지만, 파라미터 구조가 변경되면 HTML, JSP, 컨트롤러 모두 수정해야 한다는 단점이 있다.

컨트롤러 설계 - 프론트 컨트롤러 이용

- 프론트 컨트롤러는 모든 요청의 진입점이 되는 컨트롤러를 만들고, 여기에서 서브 컨트롤러를 호출하는 구조이다.

- 보다 복잡한 구조를 체계적으로 처리할 수 있다는 장점이 있다.

  /members/new-form
  /member/save

- 프론트 컨트롤러 패턴은 여러 구현에 응용되는 디자인 패턴이다.
- 중요한 것은 컨트롤러가 단순히 서블릿으로 요청을 처리하기만 하면 되는 것이 아니라 체계적인 구조로 이루어져야 한다는 것이다.