**자바 웹 개발 워크북(5~)**

**스프링 부트**

엔터프라이즈급 애플리케이션 개발을 위해 스프링 프레임워크를 지원하기 위한 개발 도구를 함께 제공하는 프로젝트이다. 서브 프로젝트에서 시작해 메인 프로젝트로 거듭난 개발 도구이다. 엄밀히 말하면 다르지만 스프링과 스프링 부트는 거의 동의어로 처리되기도 한다. 스프링 부트의 중요한 특징은 Auto Configuration이다. 관련 모듈만 추가하면 자동으로 관련 설정을 찾아 실행하는 간편화 기능이 다양하게 제공된다. 부수적이고 잡다한 모듈 설정이 필요하지 않고 설정 자체도 비교적 단순하다. 또한 내장 톰캣과 단독 실행이 가능한 도구이다. 별도의 서버 설정 없이도 개발/실행이 가능하며 이를 이용해 스프링 부트 프로젝트를 실행 가능한 jar파일로 만들고 다른 운영체제에서 실행하는 등의 작업이 가능하다.

**기존 개발에서 달라지는 점들**

톰캣이 내장된 채로 프로젝트가 생성되기 때문에 WAS의 추가 설정이 필요하지 않다. 또 빈 설정을 XML 대신 자바로 설정해 더 간단해진다. 스프링 MVC에서처럼 JSP도 사용 가능하지만 일반적으로는 템플린 엔진인 Thymeleaf를 사용한다. 화면을 구성하지 않고 데이터만을 제공하는 API 서버라는 형태를 사용하기도 한다. 스프링 웹 MVC의 기능과 설정을 사용할 수 있지만 간편하게 만들었다고 생각하면 된다.

**스프링 부트 프로젝트 생성 방식**

이니셜라이저를 사용해 자동 생성 하거나 Gradle같은 빌드 툴을 이용해 직접 생성할 수 있다. 보편적으로는 1번이 선호된다. 이니셜라이저가 프로젝트의 기본 템플릿 구조를 만들어주고 호환성 면에서 유리하기 때문이다.

스프링 부트는 main()을 실행해 프로젝트를 실행한다. 메인 에플리케이션을 실행하면 내장된 톰캣을 통해 단독으로 실행이 가능하다. 라이브러리만으로 설정을 인식하는 특성을 자동 설정(Auto Configuration)이라고 하는데 스프링 부트는 자동 설정이 적용된 프레임워크이다. 추가적인 설정을 위해서는 프로젝트 생성시 만들어진 application.properties 파일을 이용하거나 application.yml(YAML)을 이용하면 된다. 파일 설정을 피하고 싶으면 @Configuration이 있는 클래스 파일을 만들어 필요한 설정을 추가할 수도 있다.

초기에 application.properties에서 DB 드라이버 설정을 잡고 main()을 통해 실행하면 8080포트로 톰캣이 실행된다. 스프링 부트는 별도로 HikariCP 라이브러리를 가져오거나 HikariConfig 객체를 구성하지 않아도 기본적으로 이를 이용한다. 8080 포트가 점유 중이라면 application.properties에서 server.port를 지정해 변경할 수 있다.

롬복 라이브러리는 초기 설정에 추가하면 자동으로 추가되지만 테스트 환경에서는 빠져있다. Build.gradle에서 test관련 의존성을 추가해주면 된다.

Log4j2도 기본적으로 추가되어 있다. 프로퍼티에서 간단하게 로그 설정만 해두면 된다.

**스프링 부트에서 웹 개발**

Web.xml이나 servlet-context.xml과 같은 웹 설정 파일이 없기 때문에 이를 대신하는 클래스를 작성해야 한다는 점이 전과 다르다.

**JSON 데이터 만들기**

API서버는 JSP나 Thymeleaf처럼 화면과 관련된 내용을 만들어내지 않고 순수한 데이터만 전송하는 방식의 서버를 말한다. 대표적으로 모바일에서는 화면을 웹이 아닌 앱이 구성하기에 API 서버를 많이 사용한다. 과거에는 데이터 전송에 XML을 많이 사용했지만 최근에는 JSON을 사용하는 방식이 일반적이다.

JSON은 JavaScript Object Notation의 약자이다. 구조를 가진 데이터(객체)를 자바스크립트의 객체 표기법 기법으로 표현한 순수 문자열이다. 문자열이기 때문에 프로그래밍 언어에 독립적이라는 장점이 있다. 스프링에서 Jackson-databind라는 라이브러리를 통해 개발할 수 있지만 스프링 부트는 web을 추가할 때 자동으로 포함되어 별도 설정이 필요없다.

스프링 부트는 웹 개발에 있어 JSON과 Thymeleaf 형태를 많이 이용한다. 따라서 두 경우 모두 다 잘 익혀놔야 한다.

**Thymeleaf**

JSP와 비슷하게 서버에서 동작하지만 ‘템플릿 엔진’이기 때문에 정해진 템플릿에 맞춰서 데이터를 보여주는 방식이다. JSP는 그와 달리 직접 데이터를 생성한다. Thymeleaf는 HTML에 자신의 템플릿을 적용해 끼워맞춘다.

**타임리프의 특별한 기능들**

타임리프는 JSTL과 비슷한 것 같지만 인라인 처리와 레이아웃에 강점을 보인다. 특히 상황에 따라 같은 데이터를 다르게 출력해주는 인라인 기능은 자바스크립트를 사용할 때 편리하다.

레이아웃은 별도의 라이브러리가 필요해 의존성 설정을 추가해야 한다. 레이아웃을 활용하면 레이아웃을 만들어 놓고 필요할 때 특정한 부분만 추가해 활용할 수 있다.

**Spring Date JPA**

ORM은 Object Relational Mapping의 약어로 DB를 객체지향으로 구성한 시스템에 매핑하는 기법을 의미한다. JPA(Java Persistence API)는 ORM 기법을 도입해 적은 양의 객체지향 코드로 많은 일을 할 수 있는 라이브러리이다.

JPA의 핵심 아이디어는 프로그램 코드 따로 SQL 따로 생각하지 않고 객체지향 개념을 데이터(엔티티 객체)에 적용해 이를 바로 DB와 연동해 관리할 수 있게끔 하는 것이다. 이때 엔티티 객체는 기본키를 가지는 자바 객체가 된다. 엔티티 객체는 고유 식별을 위해 @Id를 이용해 객체를 구분하고 관리하게 된다. Spring Data JPA는 엔티티 객체를 이용해 JPA를 이용하는 편리한 방법들을 제공하는 스프링 관련 라이브러리이다. 자동으로 객체를 생성하고 이를 통해 예외 처리 또한 자동으로 처리하는데 이를 위해 제공하는 인터페이스가 JpaRepository이다.

**JpaRepository**

마이바티스를 쓸 때 매퍼 인터페이스만을 선언하고 매핑xml을 사용하던 것처럼 JpaRepository 인터페이스를 상속하는 인터페이스를 선언하는 것만으로 CRUD와 페이징 처리를 모두 완료할 수 있다. save(), findById()

일반적으로 엔티티 객체는 최소한의 변경이나 변경이 없는 불변 객체로 설계하는 게 좋다. 강제적인 건 아니다.

**Pageable 타입**

페이징 기능을 쓰고 싶으면 페이징 로직을 구현하지 않아도 Pageable 타입을 파라미터로 설정해 쓸 수 있다. Pageable은 pageRequest.of() 메서드를 지원하는데 이 메서드를 이용해 페이지 설정이 가능하다.

**쿼리메소드와 @Query**

SQL에서 사용하는 키워드와 칼럼을 결합해 메소드 이름을 구성하면 그 자체로 JPA가 사용하는 쿼리가 된다. 메소드 이름은 일반적으로 findBy… or get…으로 시작하고 칼럼명과 키워드를 결합해 사용한다. 하지만 요구사항과 조건이 많아질수록 메소드의 길이가 너무 길어져서 가독성을 저해시킬 수 있다. 단순한 쿼리를 작성할 때는 유용하지만 실제 개발에서는 많이 사용되지 않는다.

쿼리 메소드와 유사한 @Query를 사용하면 JPQL을 이용할 수 있기 때문이다. JPQL은 @Query의 어노테이션의 value로 작성하는 문자열을 말한다. 여기서 JPQL은 SQL과 유사한 JPA의 쿼리 언어이다. SQL은 객체지향에 최적화된 언어가 아니므로 이를 추상화 해 JPA에 맞는 객체지향 쿼리로 만든 게 JPQL이다. JPQL은 테이블 대신 엔티티 타입을 사용하고 칼럼 대신 엔티티 속성을 이용한다. SQL과 상당히 유사하지만 **조인**, **원하는 속성만 추출**해 **Object[] 혹은 DTO로 처리**하는 기능, **nativeQuery 속성값을 true로 지정**해 **특정 DB에서 동작하는 SQL을 사용하는 기능**들을 **수행할 수** **없다.**

**Querydsl을 이용한 동적 쿼리 처리**

JPQL은 어노테이션에 붙박이로 작성해서 정적이다. 검색 기능이 필요할 경우 다양한 경우의 수를 모두 미리 JPQL로 작성해둬야 이에 대비할 수 있다. 이 문제를 해결하기 위해 사용되는 방식이 Querydsl이다.

Querydsl은 JPA의 구현체인 Hibernate 프레임워크가 사용하는 HQL(Hibernate Query Language)을 동적으로 생성할 수 있는 프레임워크이다. 자바 코드를 사용해 작성하기 때문에 타입 안정성을 유지하면서 쿼리를 작성할 수 있다. Querydsl을 이용하기 위해 Q도메인이라는 존재가 필요한데 Q도메인은 Querydsl의 설정을 통해 기존의 엔티티 클래스를 Querydsl에서 사용하기 위해 별도의 코드로 생성하는 클래스이다.

Querydsl 설정을 마치면 Q도메인이 소스 파일로 생성된다. 구현을 원한다면 Search인터페이스와 Search 인터페이스와 QuerydslRepositorySupport를 상속하는 Impl클래스를 만들어 구현하면 된다. super(instance.class) 명령을 통해 QuerydslRepositrySupport의 생성자 매개변수로 Querydsl을 이용할 엔티티 클래스를 넣어주면 된다. 그리고 인터페이스의 메소드를 구현해준다.

Querydsl은 ‘타입’기반 ‘코드’이용 ‘쿼리’생성을 목표로 한다. 이때 코드를 만드는 클래스가 Q도메인 클래스이다.

페이징을 원하면 Pageable을 매개변수로 받아 구현한 뒤 Page<T>타입을 반환하게 하면 되는데 Querydsl에서는 이를 직접 처리해야 하는 불편함이 있다. Spring Data JPA가 이를 위해 지원하는 PageImpl 클래스를 이용하면 간편하다. PageImpl은 3개의 파라미터로 Page<T>를 생성할 수 있다.

* List<T>: 실제 목록 데이터가 들어감
* Pageable: 페이지 관련 정보를 가진 객체
* long: 전체 개수

Querydsl은 JDBC나 마이바티스에 비해 객체지향 코드로 쿼리를 간편하게 작성할 수 있고 동적으로 활용할 수 있다는 강점을 가진다.

**JPARepository**

JPA에서 리포지토리는 JDBC에서 DAO와 같은 역할을 한다. 앞서 보았지만 단순히 상속만 해도 JPA의 기본적인 CRUD를 가져다 쓸 수 있다. 여기에 추가적인 커스텀 쿼리를 만들고 싶다면 쿼리를 표현할 인터페이스와 Impl 클래스를 만들어 JPARepository에 더해 인터페이스를 상속시키면 된다.

JDBC를 이용할 때 서비스 클래스에서 로직을 구성하고 해당 로직에서는 DAO를 이용해 DB에 접근했던 것처럼 Spring Data JPA에서는 Mapper와 Repositoy를 통해 DB에 접근할 수 있다. DB에 직접 접근해 가져와야 하므로 JDBC를 사용할 때처럼 DTO를 VO인 Entity로 매핑해주는 과정이 필요하다.

**@Transactional**

트랜잭셔널 어노테이션을 사용하면 스프링이 해당 객체를 감싸는 별도의 클래스를 생성한다. 여러 번의 데이터 연결이 있을 수 있으므로 트랜잭션 처리는 기본으로 적용해두는 것이 좋다.