스프링 입문

강사: 김영한

스프링 입문 - 스프링부트

정적 컨텐츠

스프링 부트에서 웹브라우저에서 요청을 했을 시에 Controller에서 찾고 없다면

정적 컨텐츠를 반환해준다.

MVC와 템플릿 엔진

View는 화면을 그리는데에 모든 역량을 집중해야 한다.

@ResponseBody

API 방식으로 정보만 받을때 body에 JSON 으로 거의 사용 통일.

객체가 온다면 Default 방식 = > JSON으로 parsing해서 리턴해준다.

[HttpMessageConverter]가 동작 -String,JSON 을 정해져서 컨버터가 보내준다.

Jackson라이브러리 -> 객체를 JSON형식으로 바꿔주는 라이브러리

테스트

개발 - > 테스트 일반

테스트틀을 먼저 만들고 개발에 들어가는 것 = > 테스트 주도개발 **TDD**

스프링 빈을 등록하는 방법

1. Component 스캔을 통해 자동 의존관계 설정
2. 자바 코드로 직접 스프링 빈 등록하기

Component 스캔이 작동하는 장소

Application 클래스가 포함되어 있는 패키지와 그 하위 패키지전부

장점 : 설정파일만 변경하면 된다 (서비스 변경시)

자바 코드로 직접 스프링 빈 등록

- 생성자 주입, 필드 주입, setter 주입 3가지가 있다 .

  [주로 생성자 주입 권장한다] - 의존 관계가 실행중에 동적으로 변경하는 경우는 거의 없다.

요청 순서

Controller 에 Mapping 부터 찾고 정적 리소스를 확인한다.

스프링 **DB**접근 기술

H2 사용

스프링 통합 테스트

- DB까지 통합하여 테스트

스프링과 관련되게 테스트를 진행하는 법

```
@SpringBootTest
@Transactional
```

+ 필드 주입으로 간편하게 사용 (실제 코드에선 필드주입 사용하는걸 피한다.)

@Transactional 를 사용하게 되면 트랜잭션을 Commit 하지않고 rollback해주게 된다.

즉, DB 연결 작업해서 실행했을때 테스트에서 행해진 트랜잭션을 롤백

여러번 테스트가 가능하다**. [** 테스트 마다 동작 **]**

스프링 **Jdbc Template**

- JDBC API에서 본 반복코드를 제거

```
private final JdbcTemplate jdbcTemplate;

@Autowired
public JdbcTemplateMemberRepository(DataSource dataSource){
 jdbcTemplate = new JdbcTemplate(dataSource);
}
```

생성자가 하나면 autowired 생략가능

JPA

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=none

설정

ORM 기술 Object Relation Mapping

```
@Entity
public class Member {

   @Id
   @GeneratedValue (strategy = GenerationType.IDENTITY)
   private Long id;
   @Column(name = "username")
   private String name;
```

값 추가

```
@Override
public Member save(Member member) {
   em.persist(member);
   return member;
}
```

값 조회

```
public class JpaMemberRepository implements MemberRepository{

   private final EntityManager em;

   public JpaMemberRepository(EntityManager em) {
       this.em = em;
   }

   @Override
   public Member save(Member member) {
```

```
        em.persist(member);
        return member;
    }

    @Override
    public Optional<Member> findById(Long id) {
        Member member = em.find(Member.class, id);
        return Optional.ofNullable(member);
    }

    @Override
    public Optional<Member> findByName(String name) {
        List<Member> result = em.createQuery("select m from Member m where m.name =
:name", Member.class).setParameter("name", name)
                .getResultList();
        return result.stream().findAny();
    }

    @Override
    public List<Member> findAll() {
        return em.createQuery("select m from Member m",
Member.class).getResultList();
    }
}
```

JPQL을 작성해 줘야한다.

스프링 데이터 **JPA**

인터페이스만으로 ORM 개발을 완료할 수 있다.

jpa 기술을 스프링이 더욱 편리하게 제공 할 수 있게 한다.

인터페이스만 구현하고 자동으로 구현체를 생성해주고, 컨테이너에 인젝션해 사용한다.

AOP

Aspect Oriented Programming

AOP ?

- 관점 지향 프로그래밍
- 필요상황 예시) 모든 메소드의 호출 시간을 측정하고 싶을 때

공통 관심 사항과 핵심 관리 사항을 분리해서 조율

시간 측정 로직을 분리하는것!

예제

```
@Aspect
@Component
public class TimeTraceAop {

   @Around("execution(* hello.hellospring..*(..))")
   public Object execute(ProceedingJoinPoint joinPoint) throws Throwable{
       long start = System.currentTimeMillis();
       System.out.println("START : " + joinPoint.toString());
       try {
           return joinPoint.proceed();
       } finally {
           long finish = System.currentTimeMillis();
           long timeMs = finish - start;
           System.out.println("END : " + joinPoint.toString() + " " + timeMs + "
ms");
       }
   }
}
```

Component로 자동 빈등록을 사용해도 되지만 수동등록으로 더 확실하게 사용한다는것을 알려주는 것이 좋다.

메소드 인터셉트 가능 핵심 관심사항을 깔끔하게 유지 가능 공통 관심사항을 Aspect로 묶어서 가능하다. `@Around("execution(* hello.hellospring..*(..))")`

범위 지정도 가능하다.

AOP를 적용하게 된다면 스프링 컨테이너에서 프록시를 앞세워서 의존 주입을 하고 관리하게 된다.