64일차 강의

* @ 의 의미

: 마치 함수 같은 것을 실행시키는 것과 비슷

이를 통해 의존 주입하는 것이기 때문

* 퍼시스턴스 프레임 워크는

자바언어와 sql언어를 분리시킨다

**[ 개발 코드의 순서 ]**

* **src/main/java : 자바와 관련있는 파일관리( 자바코드파일)**
  + m : vo. dto
  + c : controller, service, dao
* **src/main/resource : 자바 이외의 파일관리(자바 코드가 아닌 것들)**
  + mappers -> sql 언어 -> xml

: Mappers : sql을 작성하는 곳

(Mapper: 우린 볼 수 없지만 존재함. 2.0버전은 볼 수 있음)

* + properties -> 설정 (jdbc, was 등) -> properties -> JNDI(Java Naming Domain Interface)

: JNDI 를 사용하면 사용자 이름이나 정보 등을 숨길 수 있다

* + config (sqlMap을 config로 바꿈)

config ->

* + - 글씨를 바꿀 때 오탈자를 방지하기위해 항상 복붙하는 것을 염두에 둬라!!!
  + 로ㅓㄹㅊ
* resultmap이 필요한이유

: Vo의 인스턴스 변수명과 resultMap 의 컬럼명이 다를 경우

* resultMap에 id 를 등록하여 id를 참조하면,

위와 같이 다른경우라도 서로 맵핑되어 resultMap을 참조할수 있도록 해준다

* Dao (deta access object)
* @ Repository 를 적어주면(annotation driven) mvc의 모델과 같은 역할을 수행할 수 있도록 해준다 (트랜젝션이나 생성자를 넣어놓음)
* 레파지토리까지 왔다는 건 데이터베이스의 연결만을 위한 것

그 결과를 담는 메서드만 필요한 것

메서드안에 조작할 필요없는 것

* salSession 은
* @Sevice

: 비지니스 로직은 여기서 처리하겠다는 의미

비지니스 중에서도 서비스에 관련된 처리

* **인터페이스를 쓰는 이유?**

이러한 기능은 있을꺼야~ 납득을 하지만,

무슨 코드를 쓸지는 협업시 서로가 달라질수 있다(매칭이 안됨)

그래서 변하지 않는 기본적인 틀을 인터페이스로 만들고(인터페이스는 설계의 영역)

안에서 구현하는 방식은 니가 마음대로 해라 단지 이 뼈대만 바꾸지 말라(의 의미가 인터페이스)

안의 코드는 상관없이 return 되는 값은 정해져 있으므로

* 인터페이스는 클래스명과 메서드명은 정해지므로,

그것의 값의 리턴이 정해져있고,

따라서 형태만 지킨다면 그 안의 코드는 어떻게 만들던지 상관이 없어진다

* 인터페이스는 무조건 형태를 만들고, 이 틀을 모두 따르자고 약속을 하는 것
* 인터페이스는 마치 부모자식 처럼의 관계가 되는데

부모 자식을 완벽히 1:1로 맵핑시킨것

부모를 몰라도 자식를 쓰면되고 자동 형변환도 해준다

* 팀워크할때 인터페이스까지 만들고 팀원에게 줘야한다.

최초의 업무분담 상태이다

1. 설계하는 사람이랑 그걸 실제로 구현(코딩)하는 사람이 별개의 사람이기 때문에 메서드명과 매개변수, 출력 데이터타입을 인터페이스를 통해 정의 내려 놓으면 그 조건에 맞게 코딩하는 사람들이 보고 어떻게든 만들기 때문이다.
2. 코드 재사용을 통해 생산성을 높이고자 할 때

인터페이스를 통해 메서드명, 매개변수, 리턴 타입이 이미 결정되었으므로, Dao 구현체의 내용이 바뀌더라도 컨트롤러의 코드 내용은 바뀔 필요가 없다.

(예: DaoImpl의 내용이 수백번 바뀌어도 ServiceImpl의 내용이 바뀔 필요가 없고, 컨트롤러의 내용도 바뀔 필요가 없다

* 코드의 재사용성
* 다형성
* 인터페이스
* 하나를 테스트해서 되는 것을 확인하고, 그후

테스트에서는 반복문을 써서 전체를 조회해야한다

* List 의 특징
* 컨트롤로 구현
* 화면을 구현 -> views 폴더에서 jsp 파일을 만들자
* Webapp 가 루트폴더이다
* 런타임 순서

1. 사용자는 index를 통해 어떤 버튼을 눌렀다

Input 태그에 주소를 넣어놨을 것

그럼 컨트롤러를(만) 찾는다

(컨트롤러로 어노테이션 한것만 찾는다)

1. 리퀘스트맵핑을 통해서 사용자의 주소를 알려준다

* 더 이상 상대 경로를 적지않는다
* 절대경로만 사용한다
* 모든 확장자명은 어떻게 끝날지는 web.xml에서 정했다
* 메서드 하나당 리퀘스트 맵핑을 해주면된다
* 메서드의 접근제어자를 무조건 public해준다
  + 인스턴스 변수는 무조건 private 으로 한다!!
* 메서드는 사람들이 인식할수 있는 기준이 된다

보통 테이블 이름명을 활용한다

* 메서드명은 유일하고 계속 발전해 나갈때를 고려해서 메서드명을 정해야한다
* 메서드명에는 일관성이 있어야 한다

: 여기에서 전략이 생기기도

테이블명(crud를 구분하는 단어)

memberSelect

memberList

getMember (앞에 get을 붙인다거나)

memberListByNo (하나를 조회할 때 ByNo를 붙여서 구분)

* Form의 action에 경로를 설정하고 메서드 방식을 설정하면,

컨트롤러에서 그에 해당하는 경로와 메서드 설정이 일치한다면 받는다

* 405오류 : 방식을 잘못 설정했다 (코드에러)
* POJO의 힘? 평이하게 하던데로 자바를 짠다
* Src/main/resource/의 log4j.xml 로 가보자
* 로그를 남기는 기술
* Log

Slf4j (인터페이스)

log4j

logger 구현

**Log4j 레벨**

|  |  |
| --- | --- |
| **FATAL** | 아주 심각한 에러가 발생한 상태. 시스템적으로 심각한 문제가 발생해서 어플리케이션작동이 불가능할 경우가 해당하는데, 일반적으로는 어플리케이션에서는 사용할 일이 없음 |
| **ERROR** | 요청을 처리하는 중 문제가 발생한 상태를 나타냄 |
| **WARN** | 처리 가능한 문제이지만, 향후 시스템 에러의 원인이 될 수 있는 경고성 메시지를 나타냄 |
| **INFO** | 로그인, 상태변경과 같은 정보성 메시지를 나타냄 |
| **DEBUG** | 개발시 디버그 용도로 사용한 메시지를 나타냄 |
| **TRACE** | log4j1.2.12에서 신규 추가된 레벨로서, DEBUG 레벨이 너무 광범위한 것을 해결하기 위해서 좀 더 상세한 상태를 나타냄 |
| FATAL > ERROR > WARN > INFO > DEBUG > TRACE DEBUG 레벨로 했다면 INFO~FATAL까지 모두 logging이 되어집니다 | |

출처: <https://cofs.tistory.com/354> [CofS]

* Syso 도 속도가 느리므로 다지워야한다

Log를 쓰면 syso를 사용하지않고도 알수 있다

완성하고 배포할 때 의미없는 syso는 전부 지워야한다