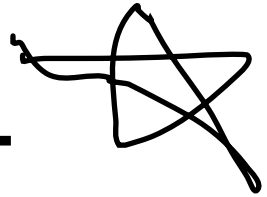
KB금융그룹 | 국민의 평생 금융파트너

vO객체를 사용하겠다ㅅ다.

2025년 상반기 K-디지털 트레이닝

MongoDB Java POJO 연동



[KB] IT's Your Life

KB 국민은행

MongoDB Java POJO 연동

POJO 매핑

- Document로 실제 데이터를 처리하는 경우 매우 번거로움
- Java POJO 클래스로 작업 직원 필요

DOCUMENT대신

자바 ㄱVO객체를 사용할 에정`

MongoDB Java POJO 연동

build.gradle

필요 의존성들

```
dependencies {
    compileOnly("org.projectlombok:lombok:1.18.32")
    annotationProcessor("org.projectlombok:lombok:1.18.32")
    testCompileOnly("org.projectlombok:lombok:1.18.32")
    testAnnotationProcessor("org.projectlombok:lombok:1.18.32")
    …
}
```

롬복도 추가!

연동하는 준비과정이
쉽지 않다

MongoDB Java POJO 연동

Database.java 유틸리티 수정하기

```
package app;
…
import org.bson.codecs.configuration.CodecProvider;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import static com.mongodb.MongoClientSettings.getDefaultCodecRegistry;
import static org.bson.codecs.configuration.CodecRegistries.fromProviders;
import static org.bson.codecs.configuration.CodecRegistries.fromRegistries;

public class Database {
    static MongoClient client;
    static MongoDatabase database;

    static {
        CodecProvider pojoCodecProvider = PojoCodecProvider.builder().automatic(true).build();
        CodecRegistry pojoCodecRegistry = fromRegistries(getDefaultCodecRegistry(), fromProviders(pojoCodecProvider));

        ConnectionString connectionString = new ConnectionString("mongodb://127.0.0.1:27017");
        client = MongoClients.create(connectionString);
        database = client.getDatabase("todo_db").withCodecRegistry(pojoCodecRegistry);
    }
```

추가 연동 준비

VO하고 컬렉션하고 매핑시켜주는 역할

후처리 디비 얻을때

MongoDB Java POJO 연동

Database.java

```
    public static void close() {
        client.close();
    }

    public static MongoDatabase getDatabase() {
        return database;
    }

    public static MongoCollection<Document> getCollection(String colName) {
        MongoCollection<Document> collection = database.getCollection(colName);
        return collection;
    }

    public static <T> MongoCollection<T> getCollection(String colName, Class<T> clazz) {
        MongoCollection<T> collection = database.getCollection(colName, clazz);
        return collection;
    }
}
```

아마 VO타입을 넣게쬬?

타입이 제네릭으로

해당 클래스에 대한 정보도 넘겨 생성자 호출을 위해서 new T()를 하기위해서 clazz.newInstance() 하기위해서

MongoDB Java POJO 연동

Todo.java VO객체 코드 쓰기

```
package app.domain;
…
import org.bson.types.ObjectId;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Todo {
    private ObjectId id;
    private String title;
    private String desc;
    private boolean done;
}
```

도메인 패키지 아래에

롬복 이용

그냥 id다

mysql연동할때 처럼

보통 자바 필드명으로

_로 시작하지 않는다.

get_id할때 햇갈리잖아
진자 이름인지
아닌지

MongoDB Java POJO 연동

Todo.java

```
package app.domain;
…
import org.bson.types.ObjectId;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Todo {
    private ObjectId id;
    private String title;
    private String desc;
    private boolean done;
}
```

MongoDB Java POJO 연동

App.java

```
package app;

import app.domain.Todo;
import com.mongodb.client.MongoCollection;
import org.bson.conversions.Bson;
import org.bson.types.ObjectId;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import static com.mongodb.client.model.Filters.eq;

public class App {
    public static void main(String[] args) {
        MongoCollection<Todo> collection = Database.getCollection("todo", Todo.class);
```

MongoCollectoin<Document>
MongoCollectoin<TOdo>
근본적인 차이
위는 비정형이네.
아래로쓰는 순간 정형이네.

정형이니까 형태가 고정됨.
몽고디비의 장점이 하나사라짐.
필드명을 유동적으로 바로 추가하거나
없애지 못함.

유연성 잃고 편리성 얻어.

내가 비정형데이터를할지 정형데이터를
다룰지에 따라 1번과 2번선택해야한다.

보통 데이터ㅓ들은 대부분 정형데이터이다

MongoDB Java POJO 연동

App.java

```
// insertOne
Todo newtodo = new Todo(null, "POJO", "POJO 테스트 확인", false);
collection.insertOne(newtodo);

// insertMany
List <Todo> newTodos = Arrays.asList(
        new Todo(null, "POJO2", "POJO2 테스트 확인", false),
        new Todo(null, "POJO3", "POJO3 테스트 확인", true),
        new Todo(null, "POJO4", "POJO4 테스트 확인", false)
);

collection.insertMany(newTodos);
```

MongoDB Java POJO 연동

App.java

```
        // find()
        List<Todo> todos = new ArrayList<>();
        collection.find().into(todos);
        for(Todo todo : todos) {
            System.out.println(todo);
        }

        // findOne()
        String id = "666a6296f4fe57189cd03eea";
        Bson query = eq("_id", new ObjectId(id));

        Todo todo = collection.find(query).first();
        System.out.println("==> findByIdResult : " + todo);

        Database.close();
    }
}
```

파인트 한것을 리스트todos로 담아라

findOne에 해당하는 동작

MongoDB Java POJO 연동

update와 delete는 기존 방식과 동일

하지만 이것도 여전히 조금 불편해^^ 스프링으로 가면 몽고디비와 연결하는 기술이 있다.
template이라고 불린다. 몽고디비 template을 사용하면
완전 자바 스타일로 사용가능

디비 기술을 어느 레벨에서 구현하느냐에 대하여
jdbc레벨. => 할만하냐? 원ㄴ초적이다. 프로그래밍 자체에는 딱히. 단순노동작업임. 중복되는 패턴이 있다.
좀더 편하게 작업을 할 디비처리 프레임워크를 사용하자.
아마jsbc프로그램잉은 다시 안할 것이다. 시간 노력 효율 다 별로.

디비처리 프레임워크 두가지 XRM, ORM기술.
xml로 데이터베이스를 매핑시키는 기술, xrm == xml rdb mapping xml로 디비매핑하겠다. 쿼리를 xml로 구성.
xml대표주자 mybatis. 쿼리를 xml로 구성하면 쿼리의 실행과 vo객체간 매핑은 알아서 다해준다.
쿼리레벨에서 get set할 필요없다. 대신 DDL파트는 직접해야함,.(인덱스 테이블 체크 등등)

ORM은 VO클래스를 만든 것처럼 자바스타일로 테이블을 정의한것이니 해당 클래스 정보를 보고
자동으로 DDL를 생성해줌. VO객체를 보고. DDL쿼리를 생성하고 앱이 기동될때 자동 실행시킴.
vo객체를 보고 현재디비 테이블을 조회하여 비교하여 없으면 생성하는 있으면 접근하는ddl쿼리를 생성및 실행.
O는 entity객체를 말함. ORM의 대표주자는 hibernate.

둘중뭘쓸지는 상부에서 지시해줌. 우리나라 흐름상과거에는 mybatis중심적. 왜냐 객체지향 전문가가 없었기때문.
객체로 디비를 바라보는 관점인 ORM을 만힝 안했었음. 그러다보니 sql중심인 xrm을 더 선호했었음.
우리나라 전자정부표준은 mybtis기준이이ㅓ쓰음. 하지만 시대상 빠른 생산력이 더 중요해짐.
최근에는 ORM으로 옮겨가ㄴ 있긴하다. 거의 양분되어있긴해. 빠른 서비스 생산. 옛날에 만들어진 서비스는
mybasts. 최근에는 ORM도 전자어부 프레임워크에 채댁됨. 어떤 서비스이냐에 따라 고르면 돼
이러한 기술을 통틀어서 JPA라고 하고. 제품명이 아닌 자바표준인터페이스다. orm을하기위한 규격.JPA맞춰개발된게hibernate
또한 이런 기준에 맞춰서 spring같은 프레임웤이 개발됨. mybatis도 마찬가지!!