Need of Spring data

module

SQL DB s/w contains

db tables having fixed

strucutre and schema

of

Different types Db s/ws

(a) SQL DB s/ws (eg:oracle, mysql, postgreSQL and etc..)

(b) No SQL DB s/ws (eg: mongoDB, cassendra, couchbase, neo4j and etc..)

n

=>if App is dealing with formatted data having fixed amout of attributes then prefer SQL db s/ws to stor that data.

of

eg:: Employee having max of 30 details

(fixed structure information)

Spring data is the main module name

and it is having lots of sub modules like => spring data jpa (for SQL dbs) => spring data mongoDB for NO SQL => spring data cassendra and etc,,

DBs

nearly 30+ sub modules are there

customer having max of 20 details

is

=>if one db table having multiple records with different count of attributes or col values but allocating for memory even unfilled col values is meaning less.

having

eg: db table in oracle for Employee 20 cols =1st record with 4 details

Here each record size can not be increaed

=>2nd record with 6 details => 3rd record with 20 details

or decreased dynamically.

not

=> 4th record with 30 details (possible) data

data

data

=>if App is getting unstructured dynamically growing and not having fixed schema then prefer using NO SQL Db s/ws.

NOSQL DB s/w does not maintain db table and records.. infact they maintain

documents,graphs, trees and etc.. which do not need any structure my fixed schema.

Records in MongoDB will be stored as documents

->doc1 with 5 details of customer1 ->doc2 with 10 details of customer2 ->doc3 with 100 details of customer3 ->doc4 with 5 details of customer4

Here docs are representing

unstructured and uneven dynamically chaging data with out fixed attributes,

Conclusion:: prefere storing data in SQL DB s/ws only when the data is having fixed attributes and fixed structure (eg: employees of a deprt having fixed no.of max attributes) prefere storing data in NO SQL DB s/ws only when the data is not having fixed attributes and fixed structure (eg: Flipkart product details

eg:: employee information in a software company (fixed structure -- use SQL Db s/w)

eg:: products inforamation in an e-commerce store (dynamic structure --- use NO SQL DB s/w)

--> eletronic products will have 10 different attributes --> colthing products will have 20 different attributes (all these details can be stroed in the form of json docs in NOSQL DB s/w though they have different attributes)

note:: json docs are the docs that maintain the information in the form of key -value pairs

Need of spring data module Spring boot data module

Before arrival of spring data module to spring/spring boot framework

Spring app

spring JDBC (abstraction on jdbc) (jdbc style)

RDBMS DB s/w or SQL DB s/w

Spring app

ORM

(abstron hibernate) -->RDBMS DB s/w or SQL DB s/w (o-r mapping style)

MongoDB api + monogo Driver

Spring App ----(native API)

---> MongoDB s/w (NOSQL DB s/w)

cassendra api +cassendra driver

Spring App -----(native API)

----> Cassendra DB s/w (NO SQL DB s/w)

=>The way of writing persistence logic in spring before the arrival of with spring data jpa is like working multiple remotes to operate multiple devices in our home..

=>Before the arrival of spring data module there is no module in spring f/w to interact with NO SQL DB s/w i.e we need to use

the NOSQL DB s/w specific apis and drivers directly for communication

of

note:: spring is not having any module to interact with NO SQL Db s/w before the arrival spring data module note:: Before arrival of spring data module .. there is no single unified mechanism to talk both

SQL and No SQL DB s/ws from spring i.e we need to use different apis to interact with different types of SQL or NO SQL DB s/ws.

Spring Data provides single unified model to interact both SQL and NO SQL DB s/w..by providing lots of sub modules..

Spring Data

Abstraction

Spring Data

JIPA

JDBC"

CouchDB MongoDB

relational

Cassandra Neo4j non-relational

Solr

Redis

(SQL DB s/ws)

(NO SQL DB s/w)

The way we write persistence logic in spring /spring boot after the arrival of spring data module is like working with single universal remote to operate all kinds of eletronic devices.

=>Spring data module provides abstraction on multiple technologies and frameworks to simplify

the interaction with both SQL and NO SQL DB s/ws in the unified model env.. (It is like universal Remote)

Important sub modules of spring data module

a) spring data jpa ---> provides abstraction on ORM f/ws like hibenrate, ibatis and etc..

b) spring data JDBC --> provides abstraction on JDBC Technology

(different from spring JDBC)

c) spring data MongoDB --->provides abstraction on MongoDB api

and etc....

=>if ur learning spring data jpa then there is no need of learning

Nearly 30+ sub modules are given in spring data module for interacting with both SQL and NoSQL DB s/w

spring jdbc module, spring orm module seperately..

Main modules

Spring Data Commons - Core Spring concepts underpinning every Spring Data module.

Spring Data JDBC - Spring Data repository support for JDBC.

Spring Data JDBC Ext - Support for database specific extensions to standard JDBC including support for Oracle RAC fast connection failover, AQ JMS support and support for using advanced data types. Spring Data JPA - Spring Data repository support for JPA.

Spring Data KeyValue - Map based repositories and SPIs to easily build a Spring Data module for key-value stores.

Spring Data LDAP - Spring Data repository support for Spring LDAP.

Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.

Spring Data Redis - Easy configuration and access to Redis from Spring applications.

Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

Spring Data for Apache Cassandra - Easy configuration and access to Apache Cassandra or large scale,

highly available, data oriented Spring applications.

Spring Data for Apache Geode - Easy configuration and access to Apache Geode for highly consistent, low latency, data oriented Spring applications.

Spring Data for Pivotal GemFire - Easy configuration and access to Pivotal GemFire for your highly consistent, low latency/high through-put, data-oriented Spring applications. Community modules

Spring Data Aerospike - Spring Data module for Aerospike.

Spring Data ArangoDB - Spring Data module for ArangoDB.

Spring Data Couchbase - Spring Data module for Couchbase.

Spring Data Azure Cosmos DB - Spring Data module for Microsoft Azure Cosmos DB.

Spring Data Cloud Datastore - Spring Data module for Google Datastore.

Spring Data Cloud Spanner - Spring Data module for Google Spanner.

Spring Data DynamoDB - Spring Data module for DynamoDB.

Spring Data Elasticsearch - Spring Data module for Elasticsearch.

Spring Data Hazelcast - Provides Spring Data repository support for Hazelcast.

Spring Data Jest - Spring Data module for Elasticsearch based on the Jest REST client.

Spring Data Neo4j - Spring-based, object-graph support and repositories for Neo4j.

Oracle NoSQL Database SDK for Spring Data - Spring Data module for Oracle NoSQL Database and Oracle NoSQL Cloud Service.

Spring Data for Apache Solr - Easy configuration and access to Apache Solr for your search-oriented Spring applications.

Spring Data Vault - Vault repositories built on top of Spring Data KeyValue.

(As of now we are learning spring data jpa to interact with SQL DB s/w and spring data mongoDB to interact with MongoDB No SQL DB s/w)

Spring App /spring Boot App

(0-r mapping persistence logic)

uses

Spring data JPA

uses

spring App/spring boot App (jdbc style persisstence logic)

uses

ORM f/w with JPA

(like Hibernate)

uses

uses

spring data JDBC

what is JPA?

=>It is s/w specification providing rules and guidelines

to develope ORM s/ws. like hibernate, iBatis and etc..

Sun Ms/oracle corp

JPA (Java Persistence API) (Specification)

RedHat/SoftTree

uses

spring JDBC +ORM framework Thibernate)

hiberante

uses

oracle corp toplink

Eclipse

Link

apache

IBatis

JDBC Technology

uses

JDBC driver s/w

=>spring data jpa intacts with

SQL DB s/w in o-r mapping style by ternally using hibenrate f/w (Best)

talks with Db s/w DB s/w

(SQL DB s/w)

=>spring data JDBC interact with DB s/w in o-r mapping + jdbc style by internally using hibenrate + jdbc

SQL

What is difference b/w ORM(o-r mapping) and JPA?

=>0-r mapping a style of persistence logic development.. For that style of programming JPA provides rules and gudelines vendor companies to creates ORM s/w..

like

=>Business in Life Insurance (ORM)

like

=> IRDA (Insurance Regularity Developement Authrority) JPA]

LIC

SBI Life

ORM Framerworks ..

Tata AIG

=>JPA is the theory or plan

=> ORM software is the software that is given based on the JPA theory.

=>All insurance companies in india controlled by IRDA becoz the IRDA gvies the guidelines to insurance companies to do business in Insurance domain

plain JDBC code (JDBC Technology code)

==================

=>Load jdbc driver class

common logics

=>Esablish the connection

=>create statement object

=> send and execute SQL query to DB s/w

=>Gather SQL query results and process them

=>handleExceptions

common logics

=>Perfrom TxMgmt

=>close jdbc objs

App specific logics

=>ORM is concept of Persistence logic development

=>JPA Programmings and guidelines for ORM persistence logic devleopment =>ORM f/w3 are really s/ws or tools that allows to implement ORM persinstence logic

to

according JPA rules and guidelines..

JPA (Theory) ----> ORM framworks (praticals)

like course [faculaties conducting classes

broucherl

based on course broucher details]

persistence

In java,we have the following possibilities to develop the logics (SQL) Λ

a) using plain JDBC b) Using Spring JDBC c)Using Plain Hibernate

d) Using Spring ORM e) spring data jpa (best)

note:: Develop should write both common logics and app specific logics in plain JDBC technology based Persistence logic development

(Boilerplate code problem)

The code that repeates across the multiple parts of the projects either with no changes or with minor changes is called boilter plate code..

plain hibernate code (ORM f/w code)

=======================

=>create Configuration class obj =>Create Session factory obj

=> creates SEssion obj (con++)

(objects based persistence logic)

Session obj = jdbc con obj++

common logics

(To activate the HB framework)

=> Write persistnece logic using Session obj app specific logics

and Entity class objs (Java bean class objs)

=>Perform Tx Mgmt (Commit or rollback activities) common logics

=> Close Session, SessionFactory objs

=> Configuration obj activates the HB f/w

=> SEssionFactory obj is factory for SEssion objs

=> Session obj= jdbc con object ++

note:: Develop should write both common logics and app specific logics (Boilerplate code problem)

The code that repeates across the multiple parts of the projects either with no changes or with minor changes is called boilter plate code..

Spring ORM code (providing abstraction on plain ORM f/w)

note1:: Plain JDBC code is SQL Queries based Persistence logic (DB s/w dependent logic) note2: Plain Hibernate Code is Objects based Persistnece logic (DB s/w independent logic)

like hibernate

=>create Hibernate Template class obj (takes care of common logics of ORM style persistence logic)

=> Develop Persistence logic using the objects of Entity class. (App specific logics)

note:: Boilerplate code problem is solved..

Limitation with spring ORM module

=====================

========

to

=> if project is having 500 db tables and we are looking perform CURD operations

on all db tables then we should develop 500 DAO Interfaces, 500 DAO impl classes

and 500* 6 methods having common persistence logic activies + addtional methods

class

in each DAOImpl specific to project requirement.

Spring data JPA Code

=====

==========

is

(this another kind of

boiler plate code program)

This problem is also there while working with plain hibernate and plain jdbc

(different types of Repository interfaces from the =>Just create Repository/DAO interface extending pre-defined Repository Inteface

are there) (Decl of CURD operation methods) (10 to 12)

with this our task is done in spring data jpa

coding conventions

=>if needed, decl some custom methods by following

advantage with spring data JPA

(note:: now for 500 db tables .. we just need to take 500 custom Repositoiry Interfaces having optional

custom method decls)

the

for the

=>Impl classes for CustomRepository interfaces will be generated dynamically providing persistence logics common methods inherited from pre-defined Repository Interfaces.. and also for custom methods decls.. (All these operations are taken care by Spring data JPA using InMemory Proxy classes)

compile

Normal Java class (.java file saved in hard disk)

.java(HDD)

e-> .class(HDD)

proxy

HDD -- Hard Disk Drive

of

JVM Loads .class file (JVM Memory RAM) ----> execution class file (JVM Memory ofRAM)

InMemory class (Everything happens at JVM memory of RAM)

====

====

of proxy class

proxy class

Conclusion

plain jdbc app :: Boilerplate code problem plain hibernate app :: boilerplate code problem spring ORMAPP one kind of boilerplate code problem is avoided but another kind of boilerplate code is added

(For 500 tables, we need to develop 500 DAO interfaces and 500 DAO impl classes having commonly used logics+ special logics)

Spring /spring boot data jpa app:

==>All kinds of boilerplate code problems are solved

output goes to console

==>For 500 tables we just take 500 DAO/repository interfaces extending from pre-defined repository interfaces (All logics will be generated in the dynamically generated proxy classes→→→→

Here the .class, .java files- will remain permanent as they allocate memory on HDD

› JVM Loads .class file (JVM Memory RAM) ----> execution class file (JVM Memory ofRAM)

output goes to console

HEre the proxy class related

Run the Application----> source code generation (JVM Memory of RAM) --------> compilation (JVM Memory of RAM) (normal app)

note:: spring data JPA uses Proxy DP to generate Impl classes of programmer supplied DAO/Repository interfaces as with InMemory Proxy classes dynamically at runtime.. i.e while working spring data ..the

persistence layer just contains DAO/Repository interfaces having few custom methods decl... becoz the implementation classes will be generated dynamically at runtime as the proxy classes

code

source code, compiled will be vanished at the end of the App's execution

The classes whose source code generation and compilation,later execution takes place

in the JVM memory where that java apps code runs is called InMemory Proxy class (JVM Memory of the RAM)

Sample Custom Repository/DAO interface of spring data jpa

inteface

public IEmployee Repo extends CrudRepository<Employee,Integer>{

}

-->Entity class pointing to db table (emp db table)

pre-defined Repository(1)

|---> @Id field (pointing to PK column) type

(12 methods)

PK column :: Primary Key column

All these Interfaces

will extend from

Pre-defined Repository Interfaces are

once Common interface

called Repository(1)

Interface Hierarchy

=> CrudRepository (12 methods)

=> JpaRepository (15 methods)

=>PagingAndSortingRepository (2 methods)

=> MongoRepository (4 methods)

and etc..

In spring boot 2.x

Repository<T,ID> (Empty Interface/ Marker interface)

these are called

common CrudRepository<T,ID> Repository interfaces

ReactiveCrudRepository<T,ID>

RxJava2CrudRepository<T,ID> RevisionRepository<T,ID,N>

PagingAndSortingRepository<T,ID>

ava2 Sorting Repository<T,ID>

Reactive SortingRepository<T,ID>

Specific to SQL DB s/w

extends

JpaRepository <T,ID>

extends

MongoRepository<T,ID> (Specific to MongoDB DB s/w)

In Spring boot 3.x

========

Repository <T,ID> (Empty Interface)

extends

(12) CrudRepository<T,ID>(I)

exstends

ListCrudRepository<T,ID>(I)

(2)

PagingAndSortinRepository<T,ID> (1)

Common Repository Interfaces

extends

ListPagingAndSortingRepository<T,ID>(I)

Specific to

extends (15) JpaRepository<T,ID>(I)

SQL DB s/w

Specific to

MongoRepository<T,ID>

MongoDB DB s/w

(No SQL DB s/w)

note:: All Repository Interfaces in the spring data jpa extends from the empty Repository (I) to make all the Repository interfaces as the same type/cult of interfaces