Table of Contents
Introduction ........................................................................................................................................ 3
History ................................................................................................................................................ 3
Common Characteristics of persistence frameworks .............................................................................. 4  
  Rich Query / API Support .................................................................................................................. 4
  Code Reduction .................................................................................................................................. 4
  Database Portability ........................................................................................................................... 4
ORM – Object-Relational Mapping ....................................................................................................... 4
  Cache Support ..................................................................................................................................... 4
  Code Maintainability .......................................................................................................................... 5
Transaction Management .................................................................................................................... 5
Classification Parameters of Database Access Technologies ................................................................. 5
  Ease of Adaptation ............................................................................................................................. 5
  Ease of Development .......................................................................................................................... 5
  Scalability .......................................................................................................................................... 6
  Performance ....................................................................................................................................... 6
Reliability ............................................................................................................................................... 6
Licensing ............................................................................................................................................... 6
General Features .................................................................................................................................. 6
Different Data Base Access Technologies / Persistence Frameworks ....................................................... 6
  Hibernate ........................................................................................................................................... 6
  TopLink ............................................................................................................................................. 7
  IBatis (Mybatis) ............................................................................................................................... 7
  Open JPA .......................................................................................................................................... 8
  Apache Cayenne ............................................................................................................................... 8
  Torque .............................................................................................................................................. 8
  Spring Framework ............................................................................................................................ 8
  jOOQ ............................................................................................................................................... 9
Conclusion .......................................................................................................................................... 10

Disclaimer: The views, processes or methodologies published in this article are those of the author. They do not necessarily reflect Dell EMC’s views, processes or methodologies.
Introduction

There is always a need to persist information in the software world. As more and more software is being written, developers started thinking of ways to write software in ways that are more efficient and less time consuming, which gave rise to the world of persistence frameworks. These frameworks save developers from re-inventing the wheel. As more software and applications are being written in Java, there is increased demand for these readymade persistence frameworks, which can be integrated easily in the software projects.

The term Persistence Framework and ORM tools are sometimes used interchangeably but there is a difference between them, as per the definition of Wikipedia "A persistence framework is a middleware that assists and automates the storage of program data into databases, especially relational databases" which implies ORM Tools are persistence frameworks. However, there are frameworks for No-SQL Databases too, e.g. Hibernate OGM (Object Grid Mapper) is used for querying data from Mongo DB. These frameworks form the core part of the database access technology world and they are the backbone of most enterprise Java applications. Choosing the appropriate technology, however, is one of the most important decisions for an enterprise architect. More often than not, such a decision hit or miss. Mistakes made in selecting the appropriate technology results in performance bottlenecks, lack of scalability, unreliable transaction handling, and so on. This paper attempts to consolidate the persistence technologies relevant in today’s scenario and also list the feature/characteristics on the basis of which of these technologies can be chosen for Java projects.

History

The first version of Java was released in 1996. At that time, engineers at Java did not feel the need to having any specification for persistence. It took almost 10 years for Java Community to come out with the specification for the support of persistence in Java. JPA (Java Persistence API) 1.0 version came out in June 2006, followed later by JPA 2.0 under JSR-317. However by the time Java came out with the specs, there were already a number of persistence frameworks available in the market, and they matured over time. Some of the famous ones that were available before JPA came out are Toplink, Hibernate, and iBatis. Toplink project started in 90s, and later on (around 1998) it was customized for Java. iBatis project also started in 2001 and a stable version was released in 2004.

All of these frameworks matured over time in their own way. However, they were lacking common specifications between them, creating problems for the developers. Any project developed with one framework could not be ported to another framework because of tight coupling with the implementation, e.g. any project that is developed using Hibernate framework, could not be made to use iBatis because of tight coupling with the Hibernate APIs. The problem of portability is a main concern for software developers and it became one of the motivations for JPA. Members of the different expert groups came together and solved this problem by writing the JSR for JPA, and thus JPA saw the light of the day. Now, most persistence frameworks are JPA compliant, i.e. they conform to the specification provided by the JPA. This has provided software developers a lot of flexibility as they can easily change the underlying implementation just by doing some configuration changes. JPA primarily defines the specifications around the following three areas:

2018 Dell EMC Proven Professional Knowledge Sharing
Common Characteristics of Persistence Frameworks

Most of the Java persistence frameworks available today possess some common characteristics across them, though these characteristics are not essential or a “must-have” for any database access technology. Following are some notable characteristics of JPA frameworks.

Rich Query / API Support
New Database access technologies provide state-of-the-art querying capability. The developer doesn't need to be a SQL query expert, and can focus more on the business and object model. Database rows (in case of relational) are queried using Java objects. These tools themselves translate it into SQLs that the underlying database can understand. This again provides abstraction over the underlying database. For instance, We have different function syntax for querying the current timestamp; MySQL has the function current_timestamp() while Oracle has sysdate functions but the framework will provide asingle function which will work with both the databases.

Code Reduction
These persistence framework help significantly reduce the number of lines of code that need to be written. Developers can focus more on the business logic layer and don’t have to pay much attention to the database layer, which mostly deals with the repetitive codebase for doing CRUD operations on the database. All of the boilerplate / repetitive code / low-level details are managed by these frameworks.

Database Portability
Database portability is one of the key advantages of using persistence frameworks. As these frameworks provide the abstraction over the underlying database, and our application layer uses these abstracted APIs, it is easy to change the underlying database, all it requires is that framework should support that database.

ORM – Object-Relational Mapping
Most persistence frameworks are ORM tools. Object Relational Mapping (ORM) as the name implies is a technique which maps Model objects to the database tables. In Java, it maps the Java Objects to the database tables and vice-versa. Data in database tables can be easily manipulated using Java objects, providing freedom from basic SQLs. ORM tools themselves provide abstraction over an underlying database.

Cache Support
Most persistence framework provide support for caching. Java Objects that are mapped to a table row are cached in memory thereby reducing a load on the database. Cache keeps the representational copy of the database in the local memory or disk. In effect, cache is a local copy of the database. This local
copy (cache) is as close as possible to the database state and sits between our application code and database. It reduces the trips between a database and the application, thus increasing the application performance. Different persistence framework provide different caching strategies for enhancing performance.

**Code Maintainability**

It is easy to maintain the code base that uses the persistence frameworks. Over the long run, the code bases are passed to different teams and developers for maintenance and upgrades. Since these frameworks follow standard specs, it is easy to understand and debug code bases for developers.

**Transaction Management**

Support for Transaction Management, in general, is provided out of the box by these persistence frameworks. All object modifications are scoped to a transaction. There is an option to commit or roll back an entire transaction. Further, there are provisions for transaction isolation. JPA provides various transaction management strategies and these frameworks have the implementation of those strategies.

**Classification Parameters of Database Access Technologies**

We can broadly classify the database access technologies on the basis of the parameters mentioned below. These parameters help us in selecting from the choice of persistence frameworks per our needs.

**Ease of Adaptation**

This parameter judges the framework on its flexibility. Are there multiples ways to manipulate the database, i.e. apart from framework-specific query language, does it provide a way to execute all type of SQLs? What data type support is provided by default and how framework deals with the user-defined data type is also a key characteristics of the framework. Frameworks should also be evaluated on their ability to invoke interceptors on the data before any CRUD operation is being performed on them. In general, there are operations like audit logging, where business validation happens before persistence. Does framework provide provision for them and if yes, how easy or difficult is to do such adaptation.

**Ease of Development**

This criterion judges the framework on its developer friendliness. What type of support is provided for writing model object; a creation of database tables or mapping or both? Does framework have the ability to create the model object from existing database schema? This schema is helpful in some legacy projects being migrated towards new persistence framework. Similarly, we can check if there is an ability to create database schema on the basis of an existing object model. If there it is out of the box support provided by the framework, it makes the life of developer very easy. Apart from object model and database creation we should also check how easy or difficult it is to map between both. In general XML mappers or annotations are used to do the mapping, however, this should not be very difficult or complex to do. There are some frameworks which provide Graphical User Interface to do so and it is very easy to do such mapping on these tools.
Scalability

We need to check the frameworks on the scalability part. How do these frameworks react when the load tends to increase on them? How good is their own DB connection pooling, or how seamlessly can third party connection pools can be integrated? How does the persistence framework behave with the scaling database.

Performance

Sometimes these frameworks tend to add a lot of overhead and performance bottleneck start happening. It is always advisable to do a performance check on these frameworks. First of all, we should do analysis on our requirements and then check if the framework is not overkill for our need. These frameworks should be evaluated on the basis of Query optimization and caching. Caching is one of the important aspects of the JPA-compliant ORM tool, and different frameworks use different caching strategies. Check to see if caching is present and if so, are these caching strategies configurable?

Reliability

Frameworks should be checked to see how good the support is for Transaction Management. The transaction is one of the fundamental things in the world of Persistence. How good is the support of JTA integration? A lot of frameworks have built-in support for transaction management with different strategies and scope for the transaction management.

Licensing

Frameworks should be checked for their License information. Not all frameworks are free for commercial purpose. The nature of the application license information about the persistence framework should be checked.

General Features

Apart from features mentioned above, we can check for the special requirement in our application. There are the certain feature which persistence frameworks provide out-of-the-box.

- Support for Batch Processing
- Integration with JMX for runtime statistics
- Lazy Loading of Objects
- Sorting and Pagination support in the Queries
- Field-level encryption in the database table
- Declarative Security where different users have different query permissions

The parameters above specify only the broad set of characteristics to look out for when finalizing any of the database access technologies.

Different Database Access Technologies / Persistence Frameworks

This section list some of the most used persistence technologies in Java. Brief descriptions and key characteristics are mentioned for each. Although each persistence framework tends to differ from another, they all pose the same features, more or less.

Hibernate
It is most popular persistence framework in Java and it has become the de-facto standard in the persistence frameworks. In addition to its own native API, it provides implementation of JPA specifications, thus enabling projects that are using JPA standard to easily switch to Hibernate. It provides support for both SQL and No-SQL Databases. Started in 2001, it is still maintained and growing, and is one of the most stable and dependable persistence frameworks available. Hibernate uses HQL (Hibernate Query language) internally which is very robust and fully leverages object-oriented paradigm. Some of the advanced functionality like pagination and dynamic profiling can be very easily achieved by HQL as compared to SQL. It is very rich in terms of features and functionally with a very large user base.

Time Line – 1st Version


License: GNU Lesser General Public License, version 2.1 and Apache License V 2.0

TopLink

This persistence framework was initially started by The Object People Company and later acquired by Oracle in 2002. Part of the TopLink code was donated to Eclipse to create a product called EclipseLink, which is compliant with JPA Specifications. “Query By Example” and “graphical mapping editor” are some of the notable features of TopLink. Advance features like a generation of the data model from object model and vice versa and auto mapping of the object model and data model make it one of the preferred persistence frameworks among the developer community. It is used mostly in enterprise-grade application.

Time Line – 1st Version released in 1998

Current State: Oracle TopLink 12c (12.1.3)

License: Oracle License

IBatis (Mybatis)

This project started with the name of IBatis in 2001 by Clinton Begin, who after three years of development donated the codebase to Apache software foundation. ASF maintained it till 2010 and then renamed it MyBatis. Unlike other persistence frameworks, it focuses more on the use of direct SQLs. A key difference between IBatis and other frameworks is that IBatis maps the model objects directly with the SQLs, result sets and stored procedures, while other frameworks map model objects to database tables. It is said that IBatis is the best choice when you need full control of SQLs.

Time Line – 1st Version released in 2001

Current State: Only MyBatis is being maintained and 3.4.6-Snapshot in Aug 2017

License: Apache License 2.0
Open JPA

An implementation of JPA from Apache, Open JPA releases are synched with the JPA versions. It claims to be fully compliant with JPA specifications and pass the TCK (Technology Compatibility Kit), a kit that checks the accuracy of the implementation for any specification by the Java community. It comes bundled with a number of open source platforms, i.e. Apache Tomcat, Apache Geronimo, etc.

Time Line – 1st Version released in 2007
Current State: Stable 2.4 x release happened in Jan 2017
License: Apache License 2.0

Apache Cayenne

Cayenne is a Java object-relational mapping (ORM) framework. The most robust persistence framework from Apache software foundation, it provides additional components like Modeler—GUI mapping tool that supports reverse engineering of RDBMS schemas editing ORM projects. In Apache Cayenne, transaction management happens behind the scene, providing provision for expanding object graph as per demand. It has module like "cayenne-crypto" which allows field-level data encryption.

Time Line – 1st Version released in 2002
Current State: 3.1.2 November 2017
License: Apache License 2.0

Torque

An object-relational mapper and generates all the database resources required by the application, Torque also includes a runtime environment to run the generated classes and resources. It hides database-specific implementation details, thus making an application independent of a specific database, which limits its capabilities to leverage any specialized DB Specific feature.

Time Line – Stable 3.3 Version released in October 2004
Current State: Stable 4.0 x release happened in February 2013
License: Apache License 2.0

This section covers the data access technologies that are not the implementation of JPA, are lightweight in nature, and either interact with the database directly using plain JDBC or sits on top of other ORM framework providing another layer of abstraction.

Spring Framework
Almost all new Java projects have some element of Spring framework. Spring framework provides a host of data access technologies. Over time new projects/modules are released by the Spring team solving one or another problem. Notable frameworks/projects/modules released by Spring are:

**Spring-JDBC**: It simplifies the plain JDBC access for Java applications and provides the JDBC template class that takes care of boilerplate code so that developers can concentrate on the SQLs and query parameters. Convenient to use with a very small learning curve, it provides several different approaches to database access.

**Spring DAO**: Data Access Object (DAO) support in Spring is provided to make working with data access technologies like JDBC, Hibernate, JPA or JDO consistent. It also enables developers to switch between persistence technologies easily and it provides the abstraction over exceptions that are specific to each persistence technology. It is not a module but a convention that dictates how DAO layer should be written.

**Spring ORM**: Another of the Spring modules that integrate with persistence technologies, namely JPA, JDO, Hibernate and iBatis. Spring provides integration classes for each of these technologies, so that they can be used following Spring principles of configuration, and smoothly integrates with Spring transaction management.

**Spring Data**: An umbrella project that has common APIs defining how to access data (DAO + annotations) in a more generic way, covering both SQL and NoSQL data sources. It provides the interfaces for defining the JPA repositories, and if naming conventions defined by Spring are followed, developers don’t even need to write the finder method names or SQLs.

Time Line – 1st Version released In March 2004

Current State: Stable version 5.0.2 in November 2017

License: Apache License 2.0

**jOOQ**

A lightweight database software library that facilitates querying data efficiently using type-safe SQL. It also provides capabilities for code generation which can be easily integrated into application code base. Centered on Active Record Design Pattern (similar to entity paradigm of JAP), it is one of the database access technologies without the JPA implementation.

Time Line – 1st Version released in August 2010

Current State: Stable Version 3.10.4 released in January 2018

License: Apache License 2.0
Conclusion

In today's software world, it's hard to classify persistence frameworks as all major frameworks are equivalent. If a framework lacks any feature or characteristic from that found in a contemporary framework in one version, it catches it in next version. Old Generation frameworks which failed to compete got retired, and only legacy projects are using them. These database technologies are evolving at a very fast pace. Each new version of the framework bring in more convenience for developers, and efficiency for the overall project.

References

http://java.sys-con.com/node/2012570