JPA – Java Persistence API

Table of Contents

[1 Introduction to JPA 2](#_Toc89979436)

[1.1 Why learn JPA? 2](#_Toc89979437)

[1.2 Book Recommendations 2](#_Toc89979438)

[1.3 Why JPA? 2](#_Toc89979439)

1. Introduction to JPA
   1. Why learn JPA?

It is widely used in the Java Enterprise world. Most of the Java enterprise apps use relational databases and JPA makes it easy to work with such databases. An alternative is JDBC but it is a low level DB connection API and does not have many features like JPA does.

JPA can show in various different types of Java applications:

* **Spring Boot application with a DB**: the most common way to build large scale Java applications
* **Java / Jakarta EE application with DB**: also widely used in the enterprise applications, especially the old ones
* **Microservices (each service with a DB)**: each microservice might have a DB and could use JPA
* **Desktop and mobile apps**: not widely used but JPA can still be used for such cases because probably your desktop or mobile app would send requests to a server.
  1. Book Recommendations

In addition to this course, there are also a couple useful books:

* **Pro JPA 2 in Java EE 8**: Covers JPA in its essence, a reference book.
* **Java Persistence with Hibernate**
  1. Why JPA?

What’s the point? Well, the point is **ORM (Object Relational Mapping)**. To build a java app with a database, you need:

1. Diagram, schematic

   Description automatically generated**JDBC drivers** to connect to the database
2. **Data access services** to access the data in the database and read, write data.
3. The data “container” will be the **data access objects (DAO’s)**, basically data instances.
4. The **business logic / business server** will call the **data services** to perform the required operations on the database

JPA is concerned with the data service part of your application. But why? This picture was working for many years just fine. So what’s the problem?

* Diagram

  Description automatically generated with low confidenceThe problem is the **paradigm mismatch**. Class vs database table.
* A table would corresponding to a class. Each column in the table are fields in the class and each row is an instance of the class.
* This is called an **impedence mismatch**. Because in a RDB, the emphesis is always on the relationships between tables but classes does not have that.

Lets say we want to **read/fetch** a bunch of Employees.

1. Prepare a SQL query: SELECT \* FROM Employee WHERE …
2. Run the query: once you run it you will get a ResultSet. You can’t use the result set directly in your app. In your java app you want a collection of Employees.
3. You will loop through each row in the ResultSet and
4. create a new Employee instance per row.
5. Extract each column and assign it to the fields of your instance.
6. Once the mapping is done, put the instance into a collection
7. But you also have to handle the exceptions or edge cases and decide a fallback value.

This was just a typical **Read** operation, the **R** of **CRUD**. This is already a lot and the other operations would be even more complex when more tables, more relationships comes into play.

* Object Relational Mapping, mapping objects to rows has to be done with every operation, so JPA does this for you. With the Java persistence API you can define the relationships and all the mapping logic.
  1. Hibernate vs JPA

Oversimplifying it, one is an **interface/specification (JPA)** and one is an **implementation (Hibernate)**. Hibernate is an open-source ORM framework. So it know what to do with the tables and classes.

* Diagram

  Description automatically generatedHibernate existed even before JPA was created.

The steps to add **Hibernate** to a java project**:**

1. you would get the library, add it to the classpath and then
2. you would map your classes to the database tables
3. map your member variables to column names
4. then use Hibernates API to do CRUD operations on the java object instances

So, Hibernate would sit between JDBC and your application. Hibernate would create all the SQL queries which does the job for you.

These mappings in step 2 and 3 can happen in 2 ways:

* XML configuration
* Java annotations (preffered)

Hibernate is one of the multiple ORM solutions, the most popular. So, when a framework gets too popular in the Java community, people will fear depending on one framework too much. What happens when one day Hibernate disappears and they are forced to use a different framework? They then need to adjust the existing code which is really painful.

* To avoid that, the community creates an interface. They try to standardize the API’s which are used by the most common frameworks in the Java space. Java EE and Jakarta EE is also examples for that.
* So, JPA is a standardization of ORM API’s
  1. Diagram

     Description automatically generatedHow JPA works and its advantages

The steps to add JPA to a java project:

1. You need to add a JPA implementation to your java project. There are also other JPA implementations, but yes, the most popular is still Hibernate.
2. Now in your code you would call the JPA API’s and not Hibernate API’s

What’s the problem with using just SQL?

* **Productivity**: Lesser code to write, lesser errors, less time to maintain
* **Database independence**: some things are depended on the database that you use. With JPA, it handles those differences for you if you want to switch the database.
* **Caching and performance**: JPA does caching in some scenarios which can improve the performance.
* But be aware of the performance drawbacks that JPA can have in certain situations
  1. Setup JPA

Typically, JPA is used with Spring or Java EE projects but the problem is that those hide a lot of details of JPA.

* So, we will setup a barebone Java project with only JPA to better understand it.

Graphical user interface, text, application, email

Description automatically generatedSteps:

1. **Create a java project**: a simple maven project
2. **Add JPA libraries**: we will use Hibernate as our JPA implementation and we also require the JDBC driver for the database we will use
3. **Tell JPA where our database is**
4. **Do entity relation mapping**
5. **Use JPA API to persist an entity instance**
   * 1. Tell JPA where our database is

JPA requires you to specify a **persistence context**. A persistence context contains information about:

* What is the database?
* Where is it located?
* Credentials?
* How should it connect and behave when connection to the database?

Chart

Description automatically generated

I created a persistence.xml in the classpath. It probably don’t have to be in META-INF but I guess it is the convention.

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

Text

Description automatically generated