<?xml version="1.0"?><DocumentFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <GraphicCharterDefinitionId>0</GraphicCharterDefinitionId> <TemplateBaseTypeId>0</TemplateBaseTypeId> <CompanyId>1</CompanyId> <ConfidentialId>0</ConfidentialId> <ConfidentialDescription /> <CountryId>0</CountryId> <PageSizeId>1</PageSizeId> <PageOrientationId>1</PageOrientationId> <PrePrintedStationary>false</PrePrintedStationary> <Project /> <Reference>20181120-101127-GSE</Reference> <TemplateType>3</TemplateType> <CultureId>es-ES</CultureId> <LanguageId>3</LanguageId> <Customer /> <DocumentDate>2018-11-21T09:20:15.7473885+01:00</DocumentDate> <Saved>false</Saved> <IsValid>true</IsValid> <FirstPageCover>false</FirstPageCover> <IsNew>true</IsNew> <CurrentVersion>1.00</CurrentVersion> <DocumentType /> <DocumentTypeId>0</DocumentTypeId> <Entity /> <HasDistributionList>true</HasDistributionList> <HasForeword>false</HasForeword> <Title /> <Status>1</Status> <StatusDescription>Trabajo</StatusDescription> <SetEdition>false</SetEdition> <SetVersion>false</SetVersion></DocumentFile>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | |  |
|  |  |  |  | |  |
|  |  |  | | |  |
|  |  |  | | |  |
|  |  |  | | |  |
|  |  | Versión 1.00 del miércoles, 21 de noviembre de 2018 | | |  |
|  |  | Estado : Trabajo | | |  |
|  |  |  | |  |  |

Destinatario(s)

|  |  |
| --- | --- |
|  |  |

Historial

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Origen de la actualización** | **Redactado por** | **Validado por** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Índice

**No se encontraron entradas de tabla de contenido.**Ciertas aplicaciones empresariales tienen la necesidad de recopilar, procesar y transformar una gran cantidad de datos y por ello la información ha de ser guardada en algún lugar donde no se pierdan dichos datos.

A pesar de todas la nuevas tecnologías que se encargan de la gestión de datos, los diseñadores de aplicaciones han tenido problemas a la hora de almacenar y recuperar información de una base de datos, y todavía siguen teniendo problemas al tratar de mover de una manera más eficiente este tipo de información.

La plataforma Java ha tenido un gran éxito en el trabajo de sistema de base de datos, pero sigue teniendo dificultades a la hora de trasladar datos de un sistema de base de datos a otro y viceversa, es decir, la manipulación de la información contenida en una base de datos se volvía más difícil de lo necesario, y los desarrolladores debían invertir mucho tiempo a la hora de plasmar en objetos los datos de las filas y columnas de una tabla en base de datos.

Para poder solucionar dicho problema, apareció una solución estándar, la Java Persistence API (JPA), la cual fue introducida en la plataforma para servir de enlace entre los modelos de dominio orientados a objetos y los sistemas de bases de datos relacionales.

Este manual pretende presentar la versión 2.2 de la API de persistencia de Java como parte de Java EE 8, de manera que permita ofrecer a los desarrollares todo lo necesario para su uso.

La liberación de mantenimiento de JPA 2.2 comenzó en 2017 bajo la JSR 338 y fue finalmente aprobado el 19 de junio de 2017.

Aquí está la declaración oficial de la versión 2.2 de Java Persistence Maintenance:

"La especificación Java Persistence 2.2 mejora la API de Java Persistence con

soporte para la repetición de anotaciones; inyección en convertidores de atributos; soporte para mapear el java.time.LocalDate, java.time.LocalTime, java.time.

LocalDateTime, java.time.offsetTime, y java.time.offsetDateTime tipos; y

métodos para recuperar los resultados de Query y TypedQuery como streams.

Uno de los puntos fuerte de la API es que puede ayudar a proporcionar persistencia de una manera mucho más efectiva independientemente de la aplicación en la que se encuentre.

Antes de explicar a fondo en que consiste la API, vamos a ver el entorno en el que nos encontramos, y que tipo de problemas estamos tratando de resolver.

**SOPORTE JAVA PARA PERSISTENCIA**

Desde los primeros días, en la plataforma Java han existido interfaces de programación que han permitido hacer de puente entre la base de datos y han permitido abstraer muchos de los datos de dominio.

La API de persistencia de Java consta de cuatro áreas:

* La API de persistencia de Java
* La API de criterios de persistencia de Java
* El lenguaje de consulta
* Los metadatos de asignación del objeto-relacional

**SOLUCIONES PROPIETARIAS**

Las soluciones del mapeo objeto-relacional han existido durante mucho tiempo, incluso más tiempo que el propio lenguaje Java. Productos como Oracle TopLink se inició en el mundo de Smalltalk antes de hacer el cambio a Java. Una de las primeras implementaciones de los “entity beans” fue añadiendo una capa adicional de estos sobre objetos mapeados por TopLink.

Los dos propietarios de APIs de persistencia más populares fueron TopLink en el espacio comercial e Hibernate en la comunidad de código abierto. Productos comerciales como TopLink estuvieron disponibles desde los primeros días de Java, fue más tarde cuando el mapeo de objeto-relacional como Hibernate se hicieron más populares y convirtió el mapeo de objeto relacional como la solución preferida para muchas aplicaciones. Este cambió proporcionó a las aplicaciones todas las características de persistencia que necesitaban.

**MAPEADORES DE DATOS**

Un enfoque parcial para resolver el problema de objeto-relacional fue el uso de mapeadores de datos. Los mapeadores de datos permiten hacer de puente entre el JDBC simple, donde el desarrollador de la aplicación es el encargado de realizar las instrucciones de SQL en bruto para mapear los objetos de las tablas de base de datos, y una solución completa al mapeo de objeto-relacional, donde normalmente se utilizan marcos de trabajo para realizar la instrucción de SQL desde métodos de mapeo de datos y permite mapear el conjunto de parámetros y resultados de la instrucción SQL. El marco de trabajo más popular fue Apache iBatis (actualmente se llama MyBatis y se aloja en Google Code).

Una de las mayores ventajas de marcos de trabajo como MyBatis es que permite tener control total sobre la instrucción SQL que se envía a la base de datos. Sin embargo, la mayor desventaja de poder escribir SQL personaliza es que necesita mantenimiento, es decir, cualquier modificación en el modelo de datos puede tener repercusiones en la instrucción de SQL.

**JDBC**

La segunda versión de la plataforma Java, Java Development Kit(JDK)1.1, lanzada en 1997, marcó el inicio del primer gran apoyo para la persistencia en base de datos con JDBC. Se trataba de una versión más específica de Java de su predecesor genérico ODBC (Object Database Conectivity) el cual permitía acceder a cualquier base de datos relacional desde cualquier idioma o plataforma. JDBC permite a programas de Java una interacción completa con la base de datos. Esta interacción depende en gran medida de SQL, que permite a los desarrolladores la posibilidad de escribir consultas y declaraciones de manipulación de datos en lenguaje de base de datos pero ejecutando y procesando un modelo de programación en Java.

La desventaja de todo esto, es que el lenguaje de SQL no es portátil, ya que cada base de datos funciona de manera diferente dependiendo de la estructura de la consulta y en la mayoría de los casos requiere un ajuste del proveedor. Debido a esto no se debe tener la tentación de utilizar instrucciones SQL listas para ejecutar, ya que si llegase el día en que tu aplicación tiene que dar soporte a otro proveedor de base de datos, se puede dar el caso que las instrucciones SQL que se estaban empleando no son compatibles con el nuevo dialecto SQL.

**ENTERPRISE JAVABEANS**

La primera versión de la plataforma Java 2 Enterprise Edition (J2EE) introdujo una nueva solución para la persistencia, llamados “beans” de entidad, que son parte de la familia de componentes de Enterprise JavaBean (EJB), la cual introdujo un enfoque basado en una interfaz de una clase concreta llamada “bean” la cual nunca ha sido usada directamente por el código cliente. En su lugar, un compilador especializado de “beans” para facilitar la persistencia, la seguridad y la gestión de transacciones, delegando la lógica de negocia a la implementación de la entidad. Los “beans” de entidades se configuraron utilizando una combinación de descriptores de despliegue XML estándar.

Los “bean” de entidades fueron sobredimensionados para resolver el problema que se estaba teniendo, pero la primera versión de la tecnología carecía de muchas características necesarias para una lógica de negocio realista, pues las relaciones entre entidades debían ser gestionadas por la aplicación, lo que requería que los campos de clave externa sean almacenados y gestionados por la clase “bean”.

Con EJB 2.0 se introdujo que los “bean” de entidades sean gestionados por contenedores, en los que el “bean” se volvía abstracto y el servidor era el encargado de generar una subclase para gestionar los datos persistentes. A pesar de todas estas mejoras, seguía sin poder resolver el problema de la excesiva complejidad.

**OBJETO DE DATOS DE JAVA**

Ante la frustración de no tener una API de persistencia estandarizada satisfactoria se inventaron los Java Data Objects(JDO) que fueron inspirados principalmente por los proveedores de base de datos orientadas a objetos (OODB) y nunca fueron adoptados por la comunidad de programación principal.

JDO se llegó a convertir en una extensión del JDK pero nunca se convirtió en una parte integral de la plataforma Java. A pesar de todo esto, los principales proveedores comerciales no compartían la misma opinión sobre cómo se debe implementar un marco de persistencia, por tanto, rara vez se llegó a usar.

Una vez que el movimiento de persistencia de EJB 3.0 estaba en movimiento, y los principales proveedores se inscribieron para formar parte de la nueva empresa de persistencia se anunció que JDO se reduciría a la especificación de modo de mantenimiento y JPA cogería tanto de JDO como de los proveedores de persistencia y se convertiría en el único estándar del futuro.

**¿POR QUÉ OTRO ESTÁNDAR?**

Dicho estándar permitía al desarrollador podía tratar el objeto persistente como cualquier otro objeto Java y luego asignarlo a un almacén persistente y usar una API de persistencia para persistirlo. Debido a que los objetos eran objetos Java normales este modelo de persistencia llegó a ser conocido como Plain Old Java Object (POJO).

La necesidad de hacer otro estándar, es permitir una implementación completa sin la necesidad de acoplar a la aplicación un producto en concreto.

Vincular un estándar a un proyecto de código abierto como Hibernate sería problemático, para el estándar y probablemente aún peor para el proyecto Hibernate. Sin embargo, el uso de una tecnología estándar a la corporación cambiar de proveedor si la elección inicial no cumple con los requisitos necesarios