Trabajando con JPA

Introducción

Índice del capítulo

Un poco de historia

- El **framework Hibernate**, un conjunto de librerías que implementaba un mapeador ORM (Mapeador Objeto-Relacional), comenzó a ser desarrollado por **Gavin King y un grupo de colaboradores** a finales de 2001.
- Desde sus inicios se estableció como un **proyecto Java open source**.
- Pronto ganó popularidad y el grupo de desarrolladores fue contratado por JBoss, integrando el producto en el servidor de aplicaciones de la compañía.
- En la **actualidad JBoss ha sido adquirido por RedHat**, que ha incorporado su servidor de aplicaciones en algunas de sus distribuciones de Linux.

Un poco de historia

- En **paralelo** al desarrollo y popularización de Hibernate, la **especificación oficial de Java EE** también intentaba definir entidades persistentes.

- En concreto, se definía en la **arquitectura EJB** (Enterprise JavaBeans) el uso de entity beans, objetos persistentes distribuidos gestionados por contenedores. Junto a los entity beans, Sun también apoyó la **especificación de JDO** (Java Data Objects), otro framework alternativo de gestión de entidades persistentes que no requiere el uso de contenedores EJB.

- **Ninguno de los dos frameworks tuvo demasiado éxito**.

- Los EJB de entidad siempre fueron denostados por ser muy **poco eficientes y complejos** de utilizar. JDO, por otra parte, **tardó bastante** en ser implementado de una forma robusta y sencilla de manejar.

Un poco de historia

- En este contexto se crea en **Mayo de 2003 el grupo de trabajo** que definirá la siguiente (actual) especificación de EJB (EJB 3.0). En este grupo de trabajo pronto se tiene que adoptar un modelo para la gestión de entidades persistentes y se decide apostar por la solución que ya ha adoptado de hecho la comunidad: **el enfoque basado en POJOs de Hibernate**.

- Tras tres años de trabajo, en **Abril de 2006** se realiza la votación que aprueba la nueva especificación. En **declaraciones de Gavin King**, la especificación de JPA recoge el **95% de las funcionalidades de Hibernate**.

- En la actualidad Hibernate ofrece una **implementación de la especificación de JPA**.

- Una de las **diferencias fundamentales** con las versiones clásicas de Hibernate es que éstas utilizan **ficheros de configuración XML** para definir el mapeado de las entidades con la base de datos. JPA (y su implementación Hibernate) recurre a **anotaciones y a opciones por defecto** para simplificar la configuración.

JPA

- **Java Persistence API** (JPA) es la tecnología estándar de Java para gestionar entidades persistentes que se incluye en la última versión de Java EE (Java EE 5).

- La **descripción oficial** del estándar está definida en el JSR 220 en el que se especifica la arquitectura completa EJB 3.0. JPA es una parte del estándar EJB 3.0, aunque está especificado en un documento separado y autocontenido.

- Si quieres obtener una **visión completa y detallada** de lo que hace JPA, te será muy útil consultar este documento. Como la mayoría de los documentos que especifican las JSR, es un documento bastante legible, muy bien estructurado, muy conciso y con bastante ejemplos. Además, por ser la especificación original, es completo. Cualquier característica de JPA debe estar reflejada en este documento.

JPA

- https://jcp.org/en/jsr/summary?id=Persistence

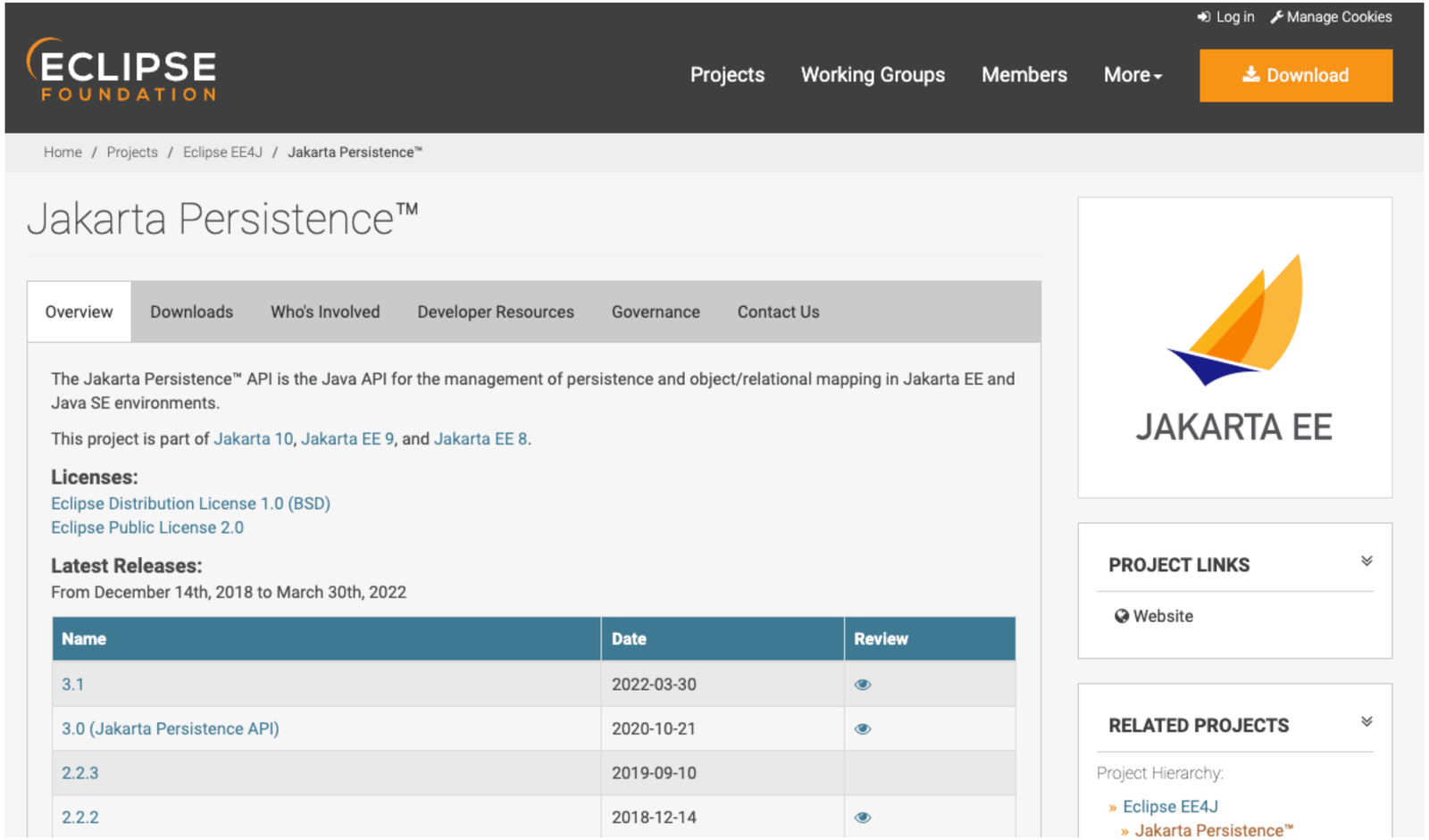
JSRs by Query - Persistence - 5 found

317	**Java™ Persistence 2.0**
Description:	The Java Persistence API is the Java API for the management of persistence and object/relational mapping for Java EE and Java SE environments.
Status:	Final
Latest Stage:	Final Release | Download page | Start: 2009-12-10
Spec Lead:	Linda Demichiel, Oracle

338	**Java™ Persistence 2.2**
Description:	The Java Persistence API is the Java API for the management of persistence and object/relational mapping in Java EE and Java SE environments.
Status:	Maintenance
Latest Stage:	Maintenance Release | Download page | Start: 2017-08-04
Spec Lead:	Linda Demichiel, Oracle
Spec Lead:	Lukas Jungmann, Oracle

JPA

- https://projects.eclipse.org/projects/ee4j.jpa



JPA

- La idea de trabajar con **entidades persistentes** ha estado presente en la Programación Orientada a Objetos desde sus **comienzos**.

- Este enfoque intenta aplicar las ideas de la **POO a las bases de datos**, de forma que las clases y los objetos de una aplicación puedan ser almacenados, modificados y buscados de forma eficiente en unidades de persistencia.

- Sin embargo, aunque desde **comienzos de los 80** hubo aplicaciones que implementaban bases de datos orientadas a objetos de forma nativa, la idea nunca ha terminado de cuajar. La **tecnología dominante** en lo referente a bases de datos siempre han sido los **sistemas de gestión de bases de datos relacionales** (RDBMS). De ahí que la solución propuesta por muchas tecnologías para conseguir entidades persistentes haya sido realizar un **mapeado del modelo de objetos al modelo relacional** (ORM, Object-Relational Mapping en inglés).

- **JPA es una de estas tecnologías**.

Entidades persistentes

- De la misma forma que en **Programación Orientada a Objetos** se trabaja con **clases e instancias**, en JPA se definen los conceptos de **clase entidad** (entity class) e **instancia de una clase entidad** a la que llamaremos instancia entidad (entity instance).

- Una **clase entidad** define un conjunto de atributos persistentes que van a compartir todas sus instancias. Por ejemplo, si estamos escribiendo una aplicación para una agencia de viajes será normal que usemos clases entidades como Hotel, Reserva o Vuelo. Las clases entidades se mapean directamente en tablas de la base de datos.

- Por otro lado, las **instancias entidad** son objetos concretos de la clase entidad (un hotel, o un vuelo) y en el **mapeado relacional** se corresponden con **filas concretas de las tablas definidas** por la clase entidad.

- A diferencia del anterior estándar de persistencia en Java EE (EntityBeans en EJB 2.1), las entidades JPA son **POJOs** (Plain Old Java Object), **objetos Java estándar** que se crean con una llamada a new, que pueden ser pasados como parámetros y que tienen un conjunto de métodos con los que acceder a sus atributos.

Relaciones entre entidades y ORM

- En JPA también es posible definir **relaciones entre entidades**, similares a las **relaciones entre tablas** en el modelo relacional.
- Así, una entidad puede estar relacionada con una o muchas otras entidades, definiendo relaciones **uno-a-uno**, **uno-a-muchos** o **muchos-a-muchos**.
- Estas relaciones se definen en JPA por **medio de variables de instancia** de las entidades y de métodos getters y setters que las actualizan.
- Por ejemplo, una propiedad de una entidad Reserva será el Vuelo sobre el que se ha hecho la reserva. Las instancias de Reserva tendrán asociadas instancias de Vuelo a las que podremos acceder fácilmente con un método como getVuelo. De esta forma, podremos obtener la(s) instancia(s) asociadas con una instancia dada llamando a un método de la entidad, sin tener que realizar ninguna consulta SQL.
- Esta es una de las **ventajas fundamentales de JPA**: es posible hacer de forma programática (llamando a métodos definidos en las entidades) lo que en el modelo relacional sólo se puede hacer mediante consultas SQL.

Relaciones entre entidades y ORM

- Otra de las **características principales de JPA** frente a otros frameworks que realizan un ORM es su **simplicidad**.
- Hasta ahora otros frameworks (como Hibernate) han utilizado **ficheros de configuración XML** para especificar el mapeado entre clases Java y tablas de la BD relacional. Esto hacía complicado la definición y el mantenimiento de las entidades.
- La **novedad de JPA en este sentido** es la utilización de **anotaciones**, una importante característica de Java aportada en su release 5.0.
- Veremos que el uso de **anotaciones** simplifica bastante la definición de entidades y de sus relaciones.

Entity Manager y transacciones

- El **Entity Manager** es el objeto de JPA que gestiona los **contextos de persistencia y las transacciones**.
- Una de las **características más importantes** de JPA es que no es **invisible**. Las entidades deben cargarse, borrarse, modificarse, etc. de forma activa por parte de la aplicación que está utilizando JPA (con la excepción, quizás, de las llamadas a los métodos set).
- Cuando una **entidad se obtiene de la base de datos** se guarda en una especie de **caché en memoria** que mantiene JPA. Esta caché se denomina **contexto de persistencia**, y se mantiene en el objeto EntityManager que utiliza la aplicación.

Implementaciones de JPA

- JPA es un **estándar que necesita ser implementado** por desarrolladores o empresas. Al ser una especificación incluida en **Java EE 5** cualquier servidor de aplicaciones compatible con Java EE debe proporcionar **una implementación de este estándar**.

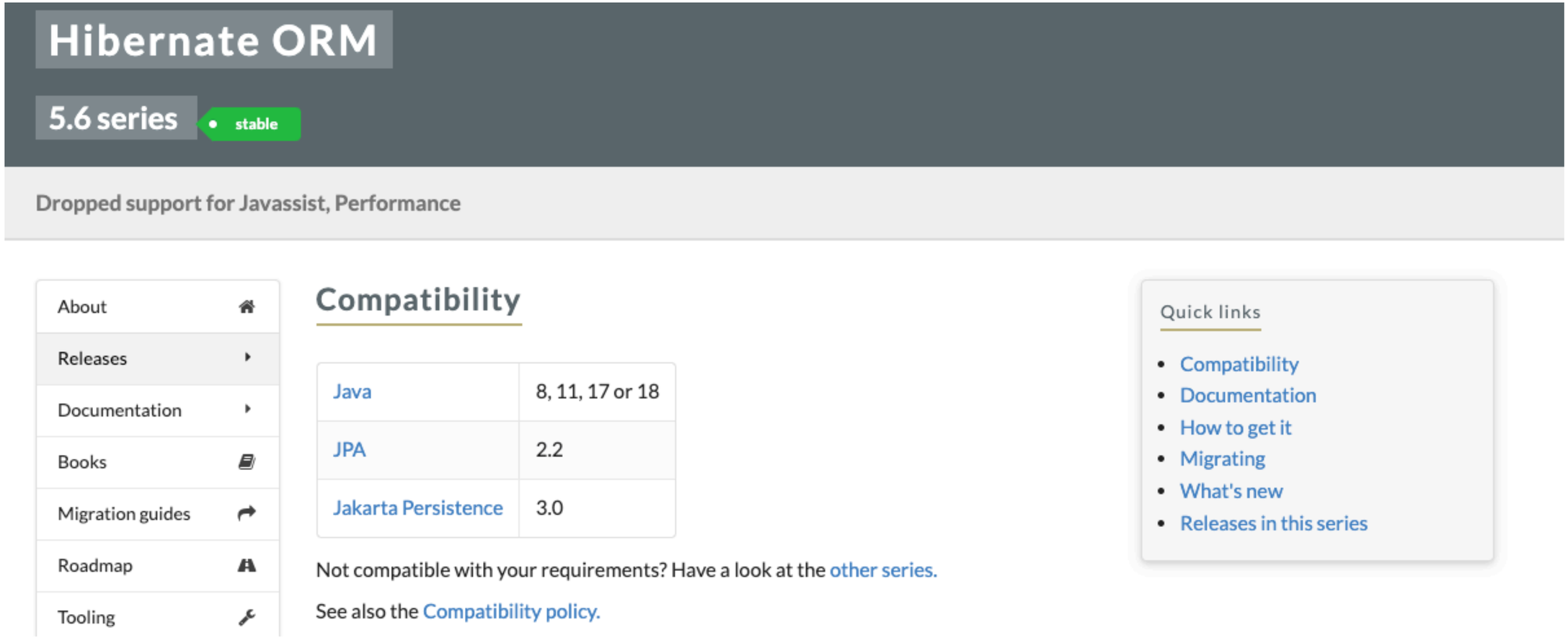
- Por otro lado, dado que también es posible utilizar JPA en Java SE, existen bastantes implementaciones de JPA en forma de librerías Java (ficheros JAR) disponibles para incluir en aplicaciones de escritorio o aplicaciones web.

- **La más popular es Hibernate**, que también se incluye en el servidor de aplicaciones JBoss. Otras implementaciones gratuitas son Apache OpenJPA (incluida en el servidor de aplicaciones Jeronimo) y Oracle TopLink (incluida en el servidor de aplicaciones GlassFish de Sun). La única implementación comercial de JPA existente en la actualidad es CocoBase PURE POJO.

- La **implementación de Hibernate es la más popular del estándar**. La gran aceptación de Hibernate en la comunidad de desarrolladores Java se refleja en que en la actualidad hay muchas empresas que utilizan Hibernate como capa de persistencia y no han dado todavía el salto a JPA. Es previsible que lo hagan próximamente.

Hibernate

- https://hibernate.org/orm/releases/5.6/



Hibernate

- https://hibernate.org/orm/releases/6.1/