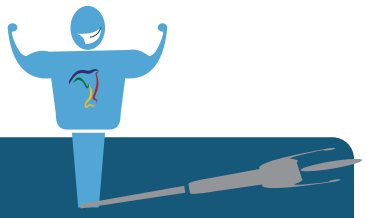
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

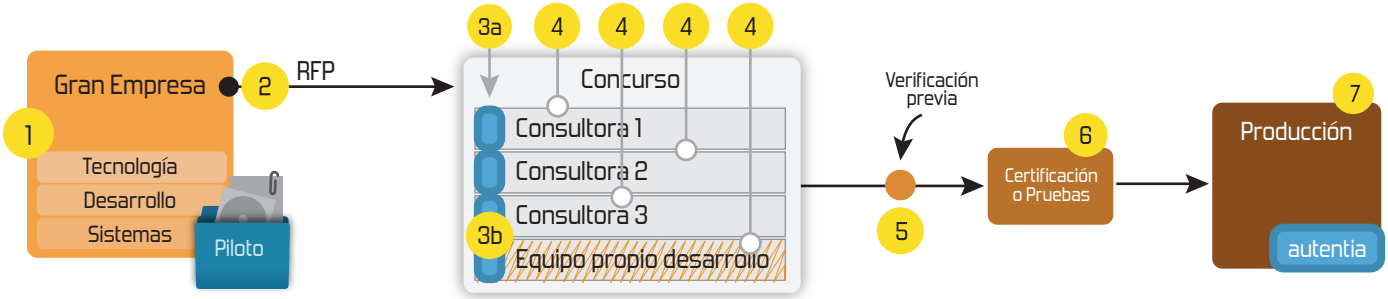
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Hosting patrocinado por enredados autentia

Inicio Quienes somos Tutoriales Formación Comparador de salarios Comentar libro Charlas Más

Estas en: **Inicio** **Tutoriales** Cómo utilizar el datastore de google App Engine con su API de nivel inferio...

Ultimas Noticias

» Cambio de fecha charla Hibernate
» ¡¡¡Vuelven las Charlas de Autentia!!!
» Nuestros tutoriales alcanzan la cifra de 10 millones de visitas!!!
» Publicado el primer libro de TDD en castellano de Carlos Blé.
» Comentando: Esta no es mi empresa: El desapego de los profesionales del siglo XXI de Ignacio Muro Benayas.
» Impresiones y video de asistentes al primer coding-dojo de agilismo.es
» Tarta y 7º aniversario de AdictosAlTrabajo.com
» Formación por calendario
» Comentando el libro: Eneagrama y éxito personal de Ginger Lapid-Bogda

+Noticias Destacadas

» ¡¡¡Vuelven las Charlas de Autentia!!!
» Impresiones y video de asistentes al primer coding-dojo de agilismo.es
» "Informática profesional: Las reglas no escritas para triunfar en la empresa."Pincha para comprar..
» Autentia patrocina el primer Agile Open Spain

+Comentarios Cómic

+Enlaces

Tutorial desarrollado por

Raúl Expósito Díaz

Consultor tecnológico de desarrollo de proyectos informáticos

Ingeniero en Informática por la Universidad Carlos III de Madrid, especialidad en IA

http://raulexposito.com/ - Perfil XING.

Puedes encontrarme en Autentia, somos expertos en Java/J2EE

Catálogo de servicios de Autentia

Descargar (6,2 MB)

Descargar en versión comic (17 MB)

AdictosAlTrabajo.com es el Web de difusión de conocimiento de Autentia.

autentia real business solutions

Catálogo de cursos

Descargar este documento en formato PDF: datastoreAPIBajoNivel.pdf

Fecha de creación del tutorial: 2010-02-17

CÓMO UTILIZAR EL DATASTORE DE GOOGLE APP ENGINE CON SU API DE NIVEL INFERIOR

1 Introducción

En un tutorial anterior os explicamos qué son el cloud computing y Google App Engine, donde os mostramos en qué consisten estas tecnologías, de qué capas están formadas y cuáles son las ventajas que nos aportan. Aquel introductorio tutorial nos servirá como base para este otro, donde nos centraremos en un apartado del Google App Engine en concreto: **su almacén de datos**, también como conocido como **el datastore**.

Todas las aplicaciones web necesitan algún tipo de soporte de almacenamiento de datos, es por ello que conocer el funcionamiento del datastore es realmente importante si quieres publicar tus aplicaciones utilizando la plataforma de Google App Engine.

El texto que vas a leer a continuación va a intentar que entiendas cómo funciona el datastore. Habrá una primera parte puramente teórica con la cual podremos situarnos y, tras ello, una parte práctica en la cual podreis descargar una aplicación de ejemplo. Dicha aplicación está escrita en java y hace uso del API de nivel inferior del datastore.

2 ¿Qué es el datastore?

El datastore es una parte vital de Google App Engine, destinada únicamente al **almacenamiento de datos**. Eso significa que todos los datos que necesitemos almacenar o recuperar los tendremos allí.

Es importante destacar que bajo el datastore **no hay una base de datos relacional**. Sirve para almacenar datos de un manera **totalmente consecuente** pero no funciona como las bases de datos tipo MySQL u Oracle. Aunque el concepto de almacenar y recuperar datos es el mismo, las tecnologías y la manera de trabajar son diferentes, lo que implica que tendrán distintos puntos fuertes y distintos puntos débiles.

Pero, ¿hasta qué punto las tecnologías son distintas?, ¿por qué los chicos de Google no han basado sus tecnologías en otras ya existentes?. Como mencionamos en el tutorial de introducción al cloud computing y a Google App Engine, Google posee unas tecnologías propias basadas en GFS y en BigTable. El primero es un sistema de ficheros propio, mientras que el segundo es un **sistema de almacenamiento de información distribuido** diseñado para almacenar cantidades de datos del tamaño de **petabytes** en miles de ordenadores.

¿Y qué diferencia hay entre unas tecnologias y otras?. Tanto GFS como BigTable han sido diseñadas específicamente para poder funcionar de manera distribuida en datacenters con cientos y miles de ordenadores en forma de grid donde, además, se entiende que la mayoría de operaciones que se realizan son lecturas en un espacio de almacenamiento infinito sobre un hardware que puede fallar en cualquier momento.

El datastore es, simplemente, una capa que se sitúa sobre BigTable. Todas las operaciones que realicemos sobre este sistema de almacenamiento tendrán que ser invocadas contra el datastore.

Catálogo de servicios Autentia (PDF 6,2MB)

En formato comic...

Acceso de usuarios registrados:

E-mail:
Contraseña:
Entrar

Deseo registrarme

He olvidado mis datos de acceso

Registra tu empresa:

Descubre las ventajas de registrar tu empresa en AdictosAlTrabajo...

Registrar mi empresa

Listado de empresas ya registradas

Google

Web
www.adictosaltrabajo.com
Buscar

Últimos tutoriales

2010-02-17
Cómo utilizar el datastore de google App Engine con su API de nivel inferior

2010-02-16
Aprendiendo Objetive-C desarrollando para nuestro Iphone 3Gs

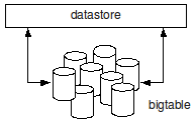
2010-02-11
Introducción a JCL.

2010-02-09
Creando la Baraja de SCRUM de Autentia como aplicación para el IPhone 3G.

2010-02-08
Cómo generar versiones imprimibles de páginas web

2010-02-04
Como cambiar el tamaño de las fuentes en Xcode (el entorno de desarrollo para Mac e iPhone)

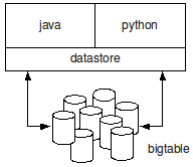
2010-02-04
Primeros pasos con Enterprise Architect y UML 2.x



BigTable fue diseñado pensando, entre otras cosas, en la **escalabilidad**. Es por ello que conceptos propios de las bases de datos relacionales como la normalización o las claves foráneas no existen. En BigTable cuando existe una relación entre datos y se guarda uno de ellos lo que en realidad se hace es guardar los dos juntos como si ambos fueran uno solo. Esto permite escalar y simplificar las búsquedas, ya que al ir a buscar uno de los datos se recuperan los dos sin obligar a BigTable a lanzar una segunda búsqueda utilizando una clave foránea.

3 ¿Qué podemos utilizar para almacenar y recuperar información del datastore?

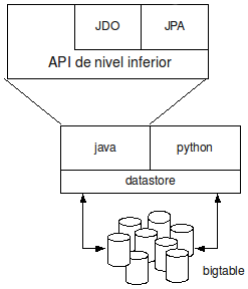
Actualmente Google da soporte para dos lenguajes diferentes en su Google App Engine: **java** y **python**. Es por ello que todas las operaciones que queramos realizar contra el Google App Engine en general y contra el datastore en particular tendrán que utilizar alguno de estos dos lenguajes:



Nosotros, a lo largo de estos tutoriales, nos vamos a centrar en el desarrollo de aplicaciones mediante el lenguaje de programación **java**. Utilizando este lenguaje dispondremos de tres maneras de poder operar con el datastore:

- JDO (*Java Data Objects*)
- JPA (*Java Persistence API*)
- El API de nivel inferior

Donde API son las siglas de *Application Programming Interface*.



Como podeis ver en la imagen, tanto JDO como JPA utilizan el API de nivel inferior para poder hacer uso del datastore debido a que JDO y JPA poseen un mayor nivel de abstracción. Eso no es obstáculo para que nosotros como desarrolladores también podamos acceder al API de nivel inferior para construir nuestras aplicaciones.

Este tutorial gira en torno al API de nivel inferior puesto que mantiene una relación más estrecha con el datastore y porque comprender cómo funciona este API nos permitirá saber mejor cómo funciona el datastore.

3.1 API de nivel inferior

Es el *interfaz de programación de aplicaciones de nivel inferior*. Se dice que es de nivel inferior ya que es la capa de almacenamiento de datos que se encuentra más cerca del datastore.

Este API opera con datos y con operaciones relativamente primitivos. Al programar con este API hay que ser explícito ya que es necesario indicar no sólo **qué** quieres almacenar o recuperar sino **cómo** quieres hacerlo.

Este API, al igual que el propio datastore, opera con **entidades**. Las entidades son objetos de datos y se explicarán en el apartado 4 de este mismo tutorial. Si conoces JDO o JPA debes tener en cuenta que **las entidades del datastore no son como las entidades de JDO o JPA**. Las entidades del datastore son lo que en java se conoce como **mapas, pares atributo-valor**.

La ventaja de trabajar contra esta API es que permite tener un mayor control a costa de tener que comprender mejor cómo funciona el datastore.

3.2 JPA

Es el *'API de persistencia para Java'*, o *'interfaz de programación de aplicaciones de persistencia para Java'*.

Este interfaz define un **estándar de almacenamiento de datos** en una base de datos relacional que, en este caso, se encuentra adaptado para poder almacenar datos en el datastore (recordemos que el datastore no es una base de datos relacional). Debido a esta adaptación, **el soporte de Google App Engine para JPA no admite algunas funciones de JPA.**

Del estándar JPA existen varias versiones, de las cuales de momento sólo la 1.0 funciona en Google App Engine y, como ya decimos, con restricciones.

Pero, ¿para qué sirve este estándar y qué ventaja podemos sacar de usar JPA en nuestras aplicaciones?, sirve para definir un marco de trabajo con un nivel de abstracción alto que, mediante la anotación de objetos Java, permita que la recuperación de objetos a través de consultas, que el almacenamiento de las mismas y que la interacción con una base de datos a través de transacciones sea bastante sencilla. Gracias a ello los desarrolladores pueden centrarse más en el **qué** quieren almacenar o recuperar que en el **cómo** tienen que hacerlo.

3.3 JDO

2010-02-04
Creación de un componente JSF, basádonos en un plugin de jQuery, con el soporte de RichFaces.

2009-02-03
Sincronizando el Mail de Mac con Gmail, el correo de Google

2010-02-03
Integración de jQuery en RichFaces.

2010-02-02
AjaxSingle: el partialSubmit de RichFaces.

2010-02-01
Introducción a RichFaces.

2010-01-29
Transformación de mensajes en SOA con OpenESB

2010-01-26
JMeter. Uso de funciones.

2010-01-18
Autenticando los usuarios de Sonar contra un LDAP

2010-01-18
Introducción a jQuery UI.

2010-01-18
jQuery: cómo crear nuestros propios plugins.

2010-01-18
Cómo consumir un servicio web RESTful con el soporte de Ajax y JSON de jQuery.

2010-01-18
Introducción a jQuery.

2010-01-17
Introducción a Tapestry 5

2010-01-14
JMeter. Gestión de usuarios

2010-01-14
Patrón Visitor con commons-collections y sus Closures

2010-01-12
Creación de servicios web RestFul, con soporte a persistencia, en NetBeans.

2010-01-11
JMeter y JSF. Extracción del parámetro ViewState

2010-01-07
Importar el correo de Microsoft Outlook al cliente de correo de Mac OS.

2010-01-07
Monitor de Hudson para Eclipse.

2010-01-07
Patrones de diseño de XML Schema

2010-01-04
Procesador Inteligente de Eventos (IEP) con OpenESB

Son los *'objetos de datos de java'* y también consiste en una especificación de almacenamiento de objetos en Java.

Tiene muchas similitudes y algunas diferencias con JPA. La idea de definir un marco de trabajo a un alto nivel de abstracción se mantiene, pero JDO **no** se centra en almacenar datos en bases de datos relacionales. JDO encaja, por tanto, mejor que JPA con el datastore ya que el éste no es relacional. Sin embargo tambien se encuentra limitado y no podremos utilizar todas las funciones de JDO en nuestra aplicación si esta tiene que funcionar en Google App Engine.

Del estándar JDO también existen varias versiones, funcionando la 2.3 en Google App Engine y, como ya decimos, con algunas restricciones.

4 ¿Qué son las entidades?

Anteriormente comentamos que el datastore opera con lo que se denominan **entidades**, y también comentamos que este concepto de *entidad* es diferente al concepto de *entidad* que existe en JDO y JPA.

Las entidades del datastore son **objetos de datos con propiedades**, pudiendo estas propiedades ser casi cualquier cosa: textos, números, valores binarios, conjuntos de datos, etc.

Lo *exótico* del asunto es que **las propiedades pueden tener uno o varios valores**, y además las entidades de un mismo tipo no tienen por qué tener las mismas propiedades, ya que **cada entidad puede definir qué propiedades tiene**. Esto se debe a que las entidades en el datastore son pares atributo-valor.

De esta manera podriamos definir entidades de tipo persona que tuvieran sólo coche y casa y entidades de tipo persona que tuvieran sólo mascota y edad. Los atributos son totalmente diferentes, pero para el datastore son entidades del mismo tipo.

Esto en un modelo relacional o en las entidades de JDO o JPA no es posible. Se podría intentar simular pero no se podría conseguir de manera nativa con facilidad.

Las entidades del datastore mantienen una clave que las identifica de manera **única**. Estas claves tienen varios valores, siendo uno de ellos precisamente el tipo de la entidad que representa.

Por tanto, cuando utilicemos el API de nivel inferior operaremos con entidades del datastore y podremos añadir todas las propiedades que necesitemos con todos los valores que hagan falta. Al utilizar JDO o JPA ganaremos en facilidad pero nos limitaremos a lo que nos permitan perdiendo estas novedosas características.

5 Transaccionalidad

Una transacción, según la wikipedia, es:

"[...] una interacción con una estructura de datos compleja, compuesta por varios procesos que se han de aplicar uno después del otro. La transacción debe ser equivalente a una interacción atómica. Es decir, que se realice de una sola vez y que la estructura a medio manipular no sea jamás alcanzable por el resto del sistema hasta que haya finalizado todos sus procesos."

http://es.wikipedia.org/wiki/Transacción_(informática)

Según la misma fuente, las transacciones deben cumplir 4 propiedades para poder ser consideradas como tal. A estas propiedades se las conoce como **ACID**

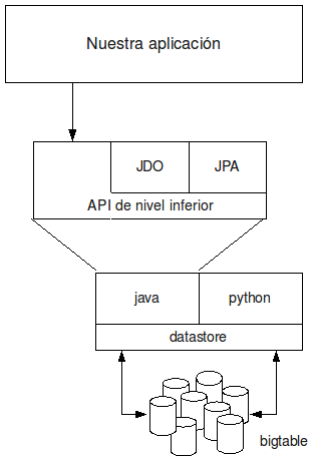
- *"Atomicidad (**A**tomicity): es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.*
- *Consistencia (**C**onsistency): es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto, se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.*
- *Aislamiento (**I**solation): es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información nunca generará ningún tipo de error.*
- *Permanencia (**D**urability): es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema."*

El concepto de transaccionalidad tambien existe en el datastore de Google App Engine, ya que vamos a ser capaces de lanzar varias operaciones en una misma transacción y recuperar la transacción entera si falla cualquiera de las operaciones. Recordemos que en el apartado 2 dijimos que el datastore era totalmente consecuente.

Pero, ¿cómo poder definir qué entra y qué no entra en una misma transacción?. Anteriormente mencionamos que el datastore opera con entidades. Pues bien, si necesitamos operar con varias entidades a la vez de manera transaccional es necesario que todas ellas pertencezcan a un mismo **grupo de entidades**. De este modo grupos de entidades distintos operan con transacciones independientes.

Otro aspecto a tener en cuenta es que el almacén de datos utiliza una política **optimista** para gestionar las transacciones. Esto es lógico, ya que tanto GFS como BigTable han sido diseñados para que sobre ellos se realicen muchas más búsquedas que escrituras.

6 Ejemplo con API de nivel inferior



A continuación os vamos a mostrar un ejemplo realizado con el API de nivel inferior. Podeis bajar el código fuente del mismo pulsando en este enlace.

La aplicación será un gestor de tutoriales para adictosaltrabajo.com, en el cual podreis publicar nuevos tutoriales y consultar todos los que ya existen.

2010-01-04
PHP Vs Java

2009-12-29
Tutorial de BPEL con OpenESB (II)

2009-12-29
Tutorial de BPEL con OpenESB (I)

2009-12-28
Pruebas funcionales de servicios web con soapUI

2009-12-28
SoapUI: jugando con web services

2009-12-17
¿Qué son el cloud computing y google app engine?

2009-12-14
JavaBean Datasource Ireport

2009-12-11
Contract-First web services con Visual Studio 2008

2009-12-09
Integrando Sonar con Hudson

2009-12-09
Apache + Tomcat: Balanceo de carga y alta disponibilidad

2009-12-08
MySQL: Replicación de bases de datos en MySQL

2009-12-07
Analizando la calidad del código Java con Sonar

2009-12-03
Instalar OpenESB 2.1 e Introducción

2009-11-25
Tutorial de Google Forms

2009-11-23
Alfresco - Modificando y eliminando contenido desde nuestras aplicaciones Java

2009-11-23
Alfresco - Añadiendo contenido desde nuestras aplicaciones Java

2009-11-23
Haciendo funcionar Google Chromium Operating System sobre Windows XP

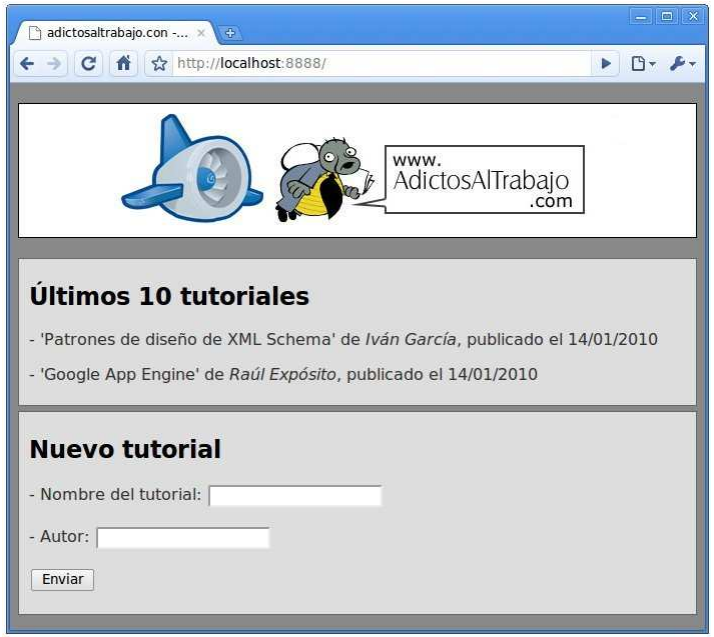
2009-11-23
Redimensionar el tamaño de una partición de VirtualBox

2009-11-20
El Arte de las Presentaciones. Siguiendo la Senda Zen.

2009-11-18
Tutorial basico de google wave bots

2009-11-13
Introducción a Escritorios Animado (Winamp y MilkWave)

2009-11-12



6.1 Cómo instalar el SDK para Java y el plugin de Eclipse

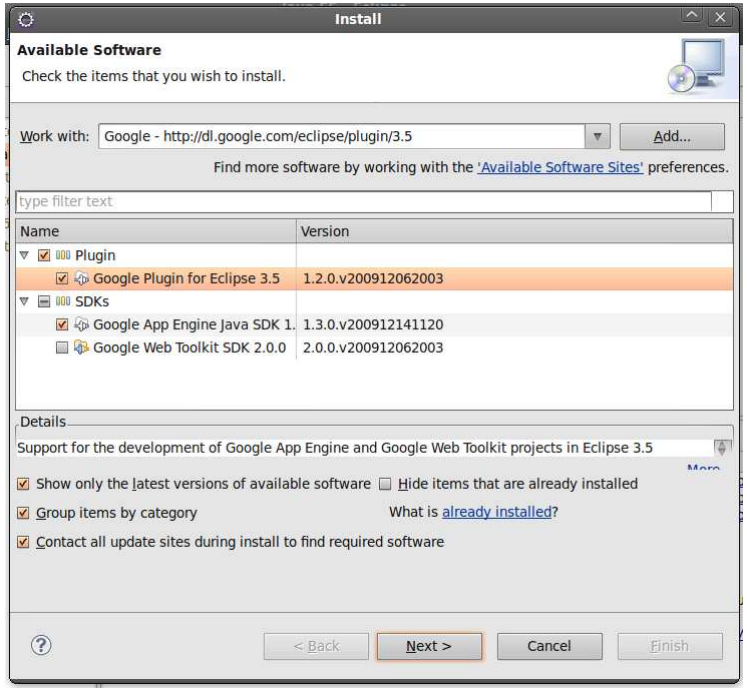
Lo primero que necesitamos es tener Java instalado en nuestro equipo, y también necesitaremos tener instalado Eclipse para poder desarrollar. El fuente de este tutorial ha sido generado con la versión 3.5 de Eclipse, también llamada Galileo.

Necesitaremos instalar el plugin de Eclipse. Para ello nos basaremos en el siguiente enlace:

http://code.google.com/intl/es/appengine/docs/java/tools/eclipse.html#Installing_the_Google_Plugin_for_Eclipse

IMPORTANTE: Si estás usando Eclipse Galileo la ruta que deberás utilizar es**http://dl.google.com/eclipse/plugin/3.5**

Maven Assembly Plugin: empaquetando aplicaciones con Maven para la ejecución de procesos batch.



Tras ello descomprimimos el código fuente del ejemplo. Si todavia no lo has bajado puedes hacerlo pulsando eneste enlace.

Una vez tengas:

- Java,
- Eclipse y
- el plugin de Eclipse

podemos importar el proyecto en nuestro workspace de Eclipse. Para ello vamos a [File > Import > General > Existing Projects into Workspace] y una vez ahí seleccionamos el directorio con el ejemplo descomprimido.

Para ejecutarlo seguiremos las indicaciones publicadas en esta dirección:http://code.google.com/intl/es/appengine/docs/java/tools/eclipse.html#Running_the_Project

6.2 Estructura del proyecto

Nos vamos a encontrar dos capas bien diferenciadas: la primera va a ser la capa con la lógica y la persistencia mientras que la segunda se va a encargar de gestionar la interfaz.

La capa con la lógica y la persistencia está formada por tres clases:

- La factoría encargada de crear servicio del datastore, llamada **DSF**. Es un singleton para asegurarnos de que no hay más que una factoria, ya que crear el servicio del datastore es computacionalmente bastante costoso.

  http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/DatastoreServiceFactory.html http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/DatastoreService.html

  view plain print ?

```
01. package com.autentia.adictosaltrabajo.gae.persistence;
02.
03. import com.google.appengine.api.datastore.DatastoreService;
04. import com.google.appengine.api.datastore.DatastoreServiceFactory;
05.
06. public class DSF {
07.
08.     private static final DatastoreService INSTANCE = DatastoreServiceFactory
09.             .getDatastoreService();
10.
11.     public static DatastoreService getDatastoreService() {
12.         return INSTANCE;
13.     }
14.
15.     private DSF() {
16.     }
17. }
```

- La entidad **Tutorial**, la cual es una clase simple que incluye en su interior una entidad del datastore. En este caso la clase *Tutorial* actua como envoltorio para simplificarnos el desarrollo. Observad que la clase Tutorial opera con la entidad como si fuese un simple mapa o una tabla hash.

  http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/Entity.html http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/Key.html

view plain print ?

```
01. package com.autentia.adictosaltrabajo.gae.persistence;
02.
03. import java.text.SimpleDateFormat;
04. import java.util.Date;
05.
06. import com.google.appengine.api.datastore.Entity;
07.
08. public class Tutorial {
09.
10.     public static final String TUTORIAL_ENTITY = "Tutorial";
11.
12.     public static final String AUTOR = "autor";
13.
14.     public static final String TUTORIAL = "tutorial";
15.
16.     public static final String FECHA = "fecha";
17.
18.     private static SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
19.
20.     private Entity entity = new Entity (TUTORIAL_ENTITY);
21.
22.     // ---------------------------------- //
23.     // constructores                      //
24.     // ---------------------------------- //
25.
26.     public Tutorial(final String autor, final String tutorial) {
27.         entity.setProperty(AUTOR, autor);
28.         entity.setProperty(TUTORIAL, tutorial);
29.         entity.setProperty(FECHA, new Date());
30.     }
31.
32.     public Tutorial(final String autor, final String tutorial, final Date fecha) {
33.         entity.setProperty(AUTOR, autor);
34.         entity.setProperty(TUTORIAL, tutorial);
35.         entity.setProperty(FECHA, fecha);
36.     }
37.
38.     // ---------------------------------------- //
39.     // getters de los atributos de la entidad  //
40.     // y de la propia entidad                  //
41.     // ---------------------------------------- //
42.
43.     public String getAutor() {
44.         return (String) entity.getProperty(AUTOR);
45.     }
46.
47.     public String getFecha() {
48.         return formatter.format((Date) entity.getProperty(FECHA));
49.     }
50.
51.     public String getTutorial() {
52.         return (String) entity.getProperty(TUTORIAL);
53.     }
54.
55.     public Entity getEntity () {
56.         return entity;
57.     }
58. }
```

- La clase encargada de almacenar y recuperar entidades, **TutorialUtils**. En esta clase hay un ejemplo sencillo de cómo lanzar una búsqueda. Las búsquedas son la base primordial datastore y es por ello que posee un API muy potente en ese sentido, donde puedes indicar no sólo qué búsqueda quieres lanzar sino, además, de qué manera quieres recuperar los resultados:

  http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/Query.html http://code.google.com/intl/es/appengine/docs/java/javadoc/com/google/appengine/api/datastore/FetchOptions.html

```
view plain  print  ?
01.  package com.autentia.adictosaltrabajo.gae.persistence;
02.
03.  import java.util.ArrayList;
04.  import java.util.Date;
05.  import java.util.List;
06.
07.  import com.google.appengine.api.datastore.DatastoreService;
08.  import com.google.appengine.api.datastore.Entity;
09.  import com.google.appengine.api.datastore.FetchOptions;
10.  import com.google.appengine.api.datastore.Query;
11.  import com.google.appengine.api.datastore.FetchOptions.Builder;
12.  import com.google.appengine.api.datastore.Query.SortDirection;
13.
14.  public class TutorialUtils {
15.
16.      private static final int FETCH_MAX_RESULTS = 10;
17.
18.      /**
19.        * Almacenamiento de un nuevo tutorial
20.        * @param autor nombre del autor
21.        * @param tituloTutorial titulo del tutorial
22.        */
23.      public static void insert(final String autor, final String tituloTutorial) {
24.          // recuperacion del datastore
25.          final DatastoreService datastoreService = DSF.getDatastoreService();
26.
27.          // creamos un nuevo tutorial y los insertamos en el datastore
28.          final Tutorial tutorial = new Tutorial(autor, tituloTutorial);
29.          datastoreService.put(tutorial.getEntity());
30.      }
31.
32.      /**
33.        * Recuperación de los últimos 10 tutoriales (FETCH_MAX_RESULTS)
34.        * @return una lista con los últimos 10 tutoriales
35.        */
36.      public static List<tutorial> getEntries() {
37.          // recuperación del datastore y configuracion de la consulta
38.          final DatastoreService datastoreService = DSF.getDatastoreService();
39.          final Query query = configureQuery();
40.          final FetchOptions fetchOptions = configureFetchOptions();
41.
42.          // declaracion de un listado donde volcar los resultados
43.          final List<tutorial> tutoriales = new ArrayList<tutorial>();
44.
45.          // lanzamiento de la consulta, la cual recupera entidades
46.          for (Entity entity: datastoreService.prepare(query).asList(fetchOptions)) {
47.              // conversion de las entidades a tutoriales
48.              tutoriales.add(convertEntityToTutorial (entity));
49.          }
50.          return tutoriales;
51.      }
52.
53.      /**
54.        * Conversion de entidad a tutorial
55.        * @param entity entidad a ser convertida en tutorial
56.        * @return tutorial procedente de una entidad
57.        */
58.      private static Tutorial convertEntityToTutorial (final Entity entity) {
59.          final String autor = (String) entity.getProperty(Tutorial.AUTOR);
60.          final String tutorial = (String) entity.getProperty(Tutorial.TUTORIAL);
61.          final Date fecha = (Date) entity.getProperty(Tutorial.FECHA);
62.          return new Tutorial(autor, tutorial, fecha);
63.      }
64.
65.      /**
66.        * Confiauración de la consulta a lanzar contra el datastore
67.        * @return una consulta configurada
68.        */
69.      private static Query configureQuery () {
70.          final Query query = new Query(Tutorial.TUTORIAL_ENTITY);
71.          query.addFilter(Tutorial.FECHA, Query.FilterOperator.NOT_EQUAL,null);
72.          query.addSort(Tutorial.FECHA, SortDirection.DESCENDING);
73.          return query;
74.      }
75.
76.      /**
77.        * Configuración de las opciones de recuperacion de datos
78.        * @return unas opciones de recuperacion de datos configuradas
79.        */
80.      private static FetchOptions configureFetchOptions () {
81.          return Builder.withLimit(FETCH_MAX_RESULTS);
82.      }
83.  }
84.
85.  </tutorial></tutorial></tutorial>
```

Por otro lado tenemos la capa encargada de gestionar el interfaz del ejemplo, la cual no vamos a mostrar en el tutorial ya que carece de interés por ser totalmente independiente al funcionamiento del datastore.

7 Conclusiones

Una necesidad constante en el desarrollo de aplicaciones web es la de disponer de un soporte sobre el cual poder almacenar y recuperar datos. Es por ello que si vamos a dejar nuestras aplicaciones en la nube de Google necesitaremos conocer cómo poder realizar estas tareas sobre sus soportes de almacenamiento.

Como hemos podido ver en este tutorial el soporte de almacenamiento de Google para Google App Engine se conoce como datastore, el cual se apoya a su vez en BigTable, tecnología que no consiste en ser una base de datos relacional aunque es totalmente consecuente. Esto responde a las necesidades de escalabilidad que requieren las aplicaciones de Google, ya que el modelo relacional posee un punto que no es posible escalar: las propias relaciones.

También hemos conocido algunos de los aspectos propios del datastore, como son las entidades, sus propiedades y las transacciones, las cuales se delimitan creando grupos de entidades.

Hemos visto que para la recuperación y almacenaje de información utilizando el soporte para Java tenemos dispone de 3 alternativas: el API de nivel inferior, JDO y JPA. Hemos situado cada una de estas tecnologías y hemos dejado un ejemplo que utiliza la primera de ellas.

El uso del API de nivel inferior es relativamente simple. Nos otorga flexibilidad, nos da todo el control y es muy potente a la hora de lanzar consultas, aparte de que como habeis podido ver no es dificil de utilizar, ya que podemos pensar que las entidades son mapas o tablas hash que se guardan en base de datos.

8 Referencias

¿Qué son el Cloud Computing y Google App Engine?

Google I/O 2008 - App Engine Datastore Under the Covers

Google I/O 2009 - Java Persistence & App Engine Datastore

El API Java de almacén de datos

El API de nivel inferior

Instalación del SDK de Java

Instalación del complemento de Google para Eclipse

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Cómo utilizar el datastore de google App Engine con su API de nivel inferior	En un tutorial anterior os explicamos qué son el cloud computing y Google App Engine, donde os mostramos en qué consisten estas tecnologías, de qué capas están formadas y cuáles son las ventajas que nos aportan. Aquel introductorio tutorial nos servirá co	2010-02-17	21	-	-	
Patrón Visitor con commons-collections y sus Closures	En este tutorial vamos a ver cómo podemos usar la librería de Apache commons-collections para implementar de forma sencilla un Visitor que se recorra todos los elementos de una colección.	2010-01-14	505	-	-	
¿Qué son el cloud computing y google app engine?	El presente tutorial trata de responder a muchas preguntas centrándose en una nube muy de moda, el Google App Engine	2009-12-17	1814	Muy bueno	5	
Apache + Tomcat: Balanceo de carga y alta disponibilidad	Este tutorial trata de cómo configurar un conjunto de servidores para que las peticiones de los usuarios a los servicios se distribuyan entre los servidores	2009-12-09	1909	-	-	
Instalación de Glassfish 2.1	En este tutorial nos veremos cómo instalar el servidor de aplicaciones GlassFish. Además veremos los primeros pasos, como entrar en la consola de administración del servidor, y desplegar una aplicación EAR (Enterprise Application)	2009-11-11	2943	Muy bueno	2	
EJB 3.0 y pruebas unitarias con Maven, JUnit 4 y Apache Open EJB.	Continuamos buscando una buena solución para llevar a cabo test de EJBs, ahora con OpenEJB.	2009-09-23	2519	-	-	
EJB 3.0 y pruebas de persistencia con Maven, JUnit 4 y Embedded JBoss sobre Java 6.	Damos continuidad al tutorial EJB 3.0 y pruebas unitarias con Maven, JUnit 4 y Embedded JBoss Java 6, probando el llevar a cabo un test de persistencia de un EJB de entidad, bajo soporte de EJB de servicio que implementa el patrón dao, en el mismo entorno	2009-09-14	2763	Muy bueno	1	
Instalación de Liferay en Tomcat existente	En este tutorial vamos a solventar el problema de instalar Liferay cuando ya contamos con un Tomcat existente y necesitamos que Liferay conviva con el resto de aplicaciones que corren en el servidor.	2009-09-12	3044	Bueno	4	
Joomla 1.5. Instalación y configuración	Veamos en detalle cómo instalar Joomla 1.5 y aplicar algunas configuraciones posteriores de las disponibles en este CMS	2009-08-14	9637	-	-	
Instalación de VirtualBox PUEL	En este tutorial os enseñamos a instalar VirtualBox como alternativa a VMWare para la virtualización de sistemas operativos.	2009-08-03	4199	Muy bueno	2	