UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Desarrollo web con Google App Engine y Java

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A

Gerardo Cardelas Gómez

DIRECTOR DE TESIS

Mtro. Juan José Carreón Granados

Asesores: M.C. Alejandro Velázquez Mena
M.I. Jorge Valeriano Assem
M.I Ángel Cesar Govantes Saldivar
Ing. Carlos Alberto Roman Zamitiz

Ciudad Universitaria, México Octubre 2012

Agradecimientos

Doy gracias a Dios por tantas cosas bellas con las que ha bendecido mi vida, por su amor que me expresa día con día en cada uno de los pasos que doy, por haber llegado hasta estos momentos, doy gracias porque me ha permitido vivir con unos padres maravillosos a quienes amo tanto, mi madre Clotilde Olivia Gómez Cruz y mi padre Andrés Cardelas Sánchez, quienes con su incansable esfuerzo me han alentado a seguir siempre adelante y quienes me han brindan su amor todos los días de mí vida.

Padres, les doy gracias por sus enseñanzas, su paciencia, su comprensión y apoyo a lo largo de todos estos años, muchas veces estuve cerca de abandonar el camino, pero ustedes siempre estuvieron ahí, brindándome su amor y el apoyo que me permitió seguir adelante.

Estar aquí ha sido un proceso que ha llevado mucho tiempo, durante el cual mi familia ha estado presente en cada paso, gracias a mis tías Julia y Martha por sus consejos, gracias a mi tío Jesús por su apoyo, a mis padrinos Eliseo y Virginia por su dedicación y cariño, gracias a mis primos Leonardo y Daniel quienes me han brindado su comprensión y apoyo todos estos años, gracias a cada uno de ustedes por alentarme a terminar este ciclo.

Angélica gracias por tu apoyo y comprensión a lo largo de este proceso, gracias por acompañarme en esta travesía aún en los momentos más difíciles, me alegra que nuestros caminos se hayan cruzado ya que me dio la oportunidad de descubrir el amor a tu lado, gracias por permitirme formar parte de tu vida.

Contenido

Introducción

El uso de la Web hoy en día es común a la mayoría de las personas, muchas de las actividades diarias involucran la interacción con ella: consultar el correo, realizar una búsqueda, ingresar a una red social. Todas estas actividades son vistas la mayoría del tiempo desde el punto de vista del usuario final, sin embargo, para el desarrollador de aplicaciones la comprensión de estas actividades resulta más profunda y desafiante.

Desarrollar una aplicación que pueda ser vista y usada por miles o millones de usuarios no es una tarea sencilla, debe de considerarse la carga en el servidor, los recursos de red, el almacenamiento de datos, los balanceadores de carga. Y adicionalmente a todas estas actividades el desarrollador de aplicaciones debe concentrarse en la lógica de negocio y la codificación de la aplicación.

Es verdad que todos estos aspectos forman parte del desarrollo de la aplicación, pero también es cierto que restan tiempo y evitan que el desarrollador sea más creativo, lo cual debería ser la cuestión principal.

Sólo si se dispone de los recursos necesarios, tal vez sea posible comprar los servidores y contratar al personal que se encargará de su administración óptima, y aún en este caso, es posible que los servidores sean incapaces de ir a la par de la carga recibida por la aplicación.

Este escenario ha sido una oportunidad para varias compañías, las cuales han comenzado a ofrecer todos estos servicios de administración de tal forma que los clientes puedan enfocarse en el desarrollo de su aplicación. Los servicios otorgados pueden agruparse en tres grupos principales:

- Infraestructura como servicio.
- Plataforma como servicio.
- Software como servicio.

La Infraestructura como servicio proporciona la mayor flexibilidad ya que permite configurar cada aspecto de los servidores y la base de datos. En esta etapa aún se requiere una persona encargada de la administración de los mismos, la ventaja se encuentra en que el usuario no tiene que mantener los servidores físicos ni el hardware necesario ya que puede adquirirlos sobre la marcha.

La plataforma como servicio se encuentra en un nivel superior a la infraestructura como servicio, en esta capa, el usuario ve una pila de software sobre la cual puede trabajar, la cual proporciona bloques de construcción. El usuario no tiene que preocuparse por configurar el software en los servidores proporcionados y el escalamiento es manejado de manera automática, a partir de la pila de software proporcionada el usuario construye su aplicación.

En el Software como servicio, los clientes no tienen conocimiento del hardware y capas de software en la solución que les es entregada, en su lugar se proporciona una interfaz que se comunica con las capas inferiores, en esta capa el usuario ve una solución proporcionada por el proveedor del servicio que puede o no ser propietaria, la cual provee de soluciones a problemas concretos.
Google App Engine (GAE) se encuentra en la segunda capa, la cual proporciona bloques de construcción en lugar de pilas de soluciones.

El trabajo tiene como objetivo el proporcionar una visión general del desarrollo de aplicaciones web con Google App Engine de tal forma que el lector sea capaz de comenzar con el desarrollo de sus aplicaciones y mostrar que Google App Engine resulta apropiado y viable para aquellas aplicaciones que cumplan las siguientes características:

- Presenten baja latencia.
- No realicen procesamiento sofisticado de los datos.

Para ello se presentarán algunas de las características que proporciona la plataforma así como sus alcances y limitaciones, el trabajo presentará las características por capítulos, los cuales contendrán ejemplos en código que ilustren el uso de la funcionalidad mencionada, se eligieron los temas teniendo en cuenta aquellos que el autor considera son comunes a una amplia gama de aplicaciones web, en base a su experiencia profesional.

La elección de Google App Engine está basada en los bloques de construcción que utiliza, los cuales son familiares al autor del presente trabajo, adicionalmente Google App Engine permite que cualquier persona pueda comenzar a trabajar en su entorno con una cantidad gratuita de recursos, a diferencia de otras opciones.

No se pretende que este trabajo sea exhaustivo, sino un punto de partida, explicando el proceso de desarrollo sobre la plataforma, ventajas, desventajas y nuevos retos que se imponen en el diseño de aplicaciones, de tal forma que el lector pueda tomar una decisión fundamentada en caso de elegir usar la herramienta.

El trabajo esta conformado por los siguientes capítulos:

Capítulo 1, Google App Engine y el cómputo en la nube:
Se muestra una descripción general de la arquitectura de Google App Engine, se describen las cuotas que afectan el sistema así como los límites en el servicio.

Capítulo 2 Manejo de recursos web
La arquitectura de Google App Engine es explicada con mayor detalle, se explica el trabajo realizando por los servidores de frontend como el realizado por los servidores estáticos, por último se explica como la aplicación puede mantener diferentes versiones.

Capítulo 3 Capa de persistencia
Se presenta una descripción de la capa de persistencia, el almacenamiento de los datos y las APIs disponibles para la consulta y manipulación de las mismas. El capítulo muestra las restricciones en la consulta de los datos así como una introducción a los índices.

Capítulo 4 El Memcache
Se presenta la memoria caché provista por Google App Engine, se introduce la api Jcache presente en la platafora y las opciones de configuración disponible.

Capítulo 5 Manejos de peticiones web
Google App Engine no permite la comunicación directa con otros hosts para ello provee del servicio URL Fetch, el capítulo explica el proceso utilizado para acceder a otros recursos web, las APIs provistas y algunos ejemplos que muestran la utilización de las mismas

Capítulo 6 Colas de tareas y tareas programadas
Se introducen las colas de tareas, las cuales son útiles para realizar actividades fuera de la petición original del usuario así como las tareas programadas, las cuales permiten realizan tareas periódicas.

Capítulo 7 Creando una aplicación
Se presenta una explicación paso a paso de como realizar una aplicación mínima con Google App Engine.

Capítulo 8 Caso de estudio. Integración del framework Struts 2
La posibilidad de integración se Google App Engine se muestra al presentar la configuración requerida para habilitar en uso del framework Struts 2 en la plataforma.

Capítulo 9 Instalación de la aplicacion
Una vez que se han terminado las pruebas de la aplicación en el servidor de desarrollo, el último paso es la instalación de la aplicación en los servidores de Google App Engine, el capítulo muestra como llevar a cabo dicho proceso.

Capítulo 1. Google App Engine y el cómputo en la nube

Google App Engine es un servicio de hospedaje de aplicaciones web, donde una aplicación web, es aquella que puede ser accedida a través de la Web generalmente por medio de un navegador.

App Engine permite el construir aplicaciones empresariales escalables en la misma infraestructura que utiliza Google. En este capítulo se mostrarán los conceptos básicos de App Engine y como está estructurado, se indicarán las características y beneficios que proporciona así como las consideraciones de diseño que se deben de tener en cuenta cuando se ejecutan aplicaciones en un entorno multihuésped.

1.1 Arquitectura de App Engine

App Engine difiere del diseño típico de un servidor web, en esencia restringe la aplicación de cualquier acceso a la capa física, evitando la apertura de sockets, ejecución de procesos en segundo plano (background), y la utilización de otras rutinas que los desarrolladores dan por hecho en otros ambientes.

App Engine está diseñado para ocuparse de las cuestiones de escalabilidad, confiabilidad y disponibilidad. Está construido sobre el concepto de escalamiento horizontal, lo cual, en esencia, significa que en lugar de ejecutar la aplicación en hardware más poderoso, esta se ejecuta en más instancias de hardware menos poderoso.

Cuando la aplicación se ejecuta en App Engine, se encuentra aislada de las demás aplicaciones, no obstante, comparte los recursos disponibles, mientras que los datos y la seguridad es única. La aplicación es capaz de utilizar algunos de los servicios de Google, como lo son URL Fetch, para ejecutar procesos en su beneficio. Debido a que no es posible el abrir sockets directamente dentro de la aplicación, la aplicación tiene que confiar en este servicio, de tal forma que Google abre un puerto y obtiene la URL solicitada.

Ejecutar todos estos servicios en beneficio de la aplicación no es algo que AppEngine maneje sin restricciones. La aplicación obtiene un límite diario en la cantidad de recursos que puede utilizar en cada tipo de petición, cada petición es rastreada y después se disminuyen los recursos utilizados en la petición de los recurso proporcionados cada día, esto evita que una aplicación monopolice los recursos o que agote sus recursos en un corto periodo de tiempo.

1.1.1 Visión General de App Engine

Google App Engine permite la construcción de aplicaciones web usando tecnologías estándar, como lo es Java, y las ejecuta en la infraestructura escalable de Google.

Una aplicación basada en Java se ejecuta en una Máquina Virtual de Java (JVM) versión 6, en un ambiente seguro conocido como “caja de arena” (sandbox). En este ambiente, la aplicación puede

ejecutar código, almacenar y solicitar datos al datastore de AppEngine, usar App Engine mail, ULR fetch services y examinar las peticiones web del usuario y preparar la respuesta.

Una aplicación en App Engine no puede realizar las siguientes actividades:

- Escribir en el sistema de archivos. Las aplicaciones deben usar el App Engine datastore para almacenar datos persistentes. Leer del sistema de archivos está permitido, y todos los archivos que fueron subidos juntos con la aplicación están disponibles.
- Abrir un socket o acceder a otro host de manera directa. Una aplicación puede utilizar el AppEngine URL fetch service para realizar peticiones HTTP o HTTPS a otros host en los puertos 80 y 443, respectivamente.
- Generar un sub-proceso o thread. Una petición web a la aplicación debe ser manejada en un sólo proceso dentro de unos pocos segundos. Procesos que toman demasiado tiempo para responder son terminados para evitar una sobrecarga en el servidor web.
- Ejecución de otro tipo de llamadas de sistema.

Mientras que App Engine no soporta enteramente la especificación Java EE [5], muchos de sus componentes individuales son soportados. Estos incluyen:

- Java Data Objects (JDO).
- Java Persistence API (JPA).
- Java Server Faces (JSF) 1.1 – 2.0
- Java Server Pages (JSP) + JSTL
- Java Servlet API 2.4
- JavaBeans Activation Framework (JAF)
- Java Architecture for XML Binding (JAXB)
- JavaMail
- XML processing APIs including DOM, SAX and XSLT.

1.2 Cuotas

Cada aplicación en App Engine obtiene una cantidad limitada de recursos de cómputo de manera gratuita, esto permite a los usuarios el comenzar con el desarrollo y prueba de la aplicación de inmediato. Es posible comprar recursos adicionales de cómputo si la aplicación lo requiere.

El ambiente en tiempo de ejecución supervisa los recursos de sistema usados por la aplicación y limita cuanto puede consumir de cada uno de ellos. Aquellos recursos por los cuales paga el usuario, tales como el CPU, el tiempo o el almacenamiento, los límites se establecen al asignarles un presupuesto determinado en la Consola de Administración. App Engine también refuerza varios limites que se aplican a todo el sistema, lo que protege la integridad de los servidores y su disponibilidad para atender muchas aplicaciones.

En el dialecto de App Engine, las “cuotas” son limites en los recursos que se actualizan al principio del día del calendario (medianoche en Tiempo del Pacífico). Es posible el supervisar el consumo diario de las cuotas usando la Consola de Administración en la sección “Quota Details”.

Debido a que Google podría cambiar estos límites a medida que el sistema es ajustado, no se indicarán valores específicos en los límites de los recursos, los cuales siempre podrán ser consultados en la documentación oficial.

Google App Engine proporciona recursos a la aplicación de manera automática siempre que el tráfico se incremente, de tal forma que pueda soportar varias peticiones de manera simultánea. Sin embargo, App Engine reserva automáticamente capacidades de escalamiento para aplicaciones con baja latencia, donde la aplicación responde a las peticiones en menos de un segundo. Aplicaciones con muy alta latencia (más de un segundo por petición para muchas peticiones) son limitadas por el sistema, y requieren una exención para poder tener un número mayor de peticiones simultáneas. Si la aplicación tiene una fuerte necesidad de un alto rendimiento en peticiones extensas, se puede solicitar una exención en el límite de las peticiones dinámicas simultáneas. La gran mayoría de las aplicaciones no requieren ninguna exención.

Las aplicaciones que están fuertemente limitadas por el CPU pueden también incurrir en alguna latencia adicional en orden de compartir eficientemente recursos con otras aplicaciones en el mismo servidor. Las peticiones de archivos estáticos están exentas de estos límites de latencia.

1.2.1 Cuotas facturables y cuotas fijas

AppEngine define dos tipos de cuotas, tal como se muestra en la tabla 1.1

Tabla 1.1 Tipos de cuota

Tipo de cuota	**Descripción**
Cuota facturable	Máximos establecidos por el usuario
	Basados en presupuesto
	Varían de acuerdo a la aplicación y pueden ser establecidos por el administrador.
Cuota fija	Máximos establecidos por App Engine
	Basados por sistema
	El mismo para todas las aplicaciones en App Engine

Los límites en las aplicaciones gratuitas son suficientes para una amplia gama de aplicaciones, al habilitar el cobro en la aplicación, se incrementan los límites de las cuotas más allá de lo que es provisto por la versión gratuita.

Las cuotas se reinician cada día a medianoche. Cualquier uso que se haya tenido comienza nuevamente al siguiente día (App Engine utiliza el Tiempo Pacífico Estándar para el cobro y la medición de las cuotas, de tal forma que podría no representar la medianoche de la localidad en la que el usuario se encuentre).

Adicionalmente a las cuotas diarias, App Engine mide algunas de ellas por minuto. Esto tiene como objetivo el proteger la aplicación de utilizar sus recursos diarios en un corto periodo de tiempo. Y, por supuesto, en un ambiente multi-huésped, también previene que otras aplicaciones en App Engine monopolicen cualquier recurso, lo cual afectaría el desempeño de la aplicación.

Si la aplicación consume un recurso muy rápido, la palabra "Limited" aparecerá a lado de dicho recurso en la pantalla Quota Details de la Consola de Administración de App Enigne. Una vez que un recurso a sido agotado, App Engine rechazará peticiones a ese recurso, regresando un estatus code de HTTP 403 Forbidden. Esto significa que la aplicación no funcionará más hasta que el recurso haya sido repuesto. Los siguientes recursos tienen este comportamiento:

- Peticiones
- Tiempo de CPU
- Ancho de banda entrante
- Ancho de banda de salida

Para otros recursos que son agotados, la aplicación lanzará una excepción.

Sin embargo, existen recursos que no son renovables al final de cada día, estos son fijos y se manejan como un todo para la aplicación, entre ellos se encuentran:

- El almacén de datos (Datastore)

1.2.2 Cuotas de seguridad y límites en los recursos facturables

Las cuotas de seguridad son establecidas por Google para proteger la integridad del sistema App Engine. Estas cuotas aseguran que ninguna aplicación puede consumir recursos en exceso que causen detrimento de otras aplicaciones. Las cuotas en los recursos facturables son establecidas por el administrador de la aplicación en la pestaña "Billing Settings" de la consola de administración. Estas cuotas permiten a los administradores el manejar el desempeño y costo de la aplicación.

1.2.3 Cuotas de seguridad

Las cuotas de seguridad incluyen *cuotas por día* y *cuotas por minuto:*

*Cuotas por día**:*** son actualizadas todos los días a la media noche tiempo del Pacífico. Las aplicaciones que tienen habilitada la facturación pueden exceder esta cuota libre hasta agotar su presupuesto.

Cuotas por minuto: protegen la aplicación de consumir todos sus recursos en un periodo corto de tiempo, y previenen a otras aplicaciones de monopolizar determinado recurso. Si la aplicación consume un recurso de manera muy rápida y agota una de las cuotas por minuto, la palabra "Limited" aparece a lado de la cuota correspondiente en la pantalla Quota Details en la consola de administración. Las peticiones por recursos que han alcanzado su número máximo por minuto serán denegadas.

Las aplicaciones que soportan la facturación tienen cuotas diarias y por minuto mayores a las de las aplicaciones gratuitas. Cuando una aplicación excede esta cuota, continua usando el recurso hasta que su presupuesto es agotado. Estas cuotas son estrictas por seguridad y representan un extremo que la mayoría de las aplicaciones no encontrará en uso normal.

Una aplicación puede utilizar los siguientes recursos, los cuales están sujetos a cuotas. Los recursos medidos contra fondos económicos son indicados como “(facturable)”. Los recursos representan una asignación sobre un periodo de 24 horas.

1.2.4 Peticiones

Varios límites que aplican al sistema en general especifican como se comportan las peticiones. Esto incluye el tamaño y número de peticiones por un periodo de tiempo, y el ancho de banda consumido por el tráfico entrante y saliente.

Un límite importante en las peticiones es el tiempo de respuesta. Una aplicación tiene 30 segundos para entregar la respuesta a una petición.

Cerca del final de los 30 segundos, el servidores lanzan una excepción que la aplicación puede atrapar para propósitos de terminar limpiamente o regresar al usuario un mensaje más amigable. Si el manejador de la petición no ha regresado una respuesta o ha excedido los 30 segundos, el servidor termina el proceso y regresa un error genérico (código 500 de HTTP) al cliente.

El límite de 30 segundos se aplica a toda petición, incluyendo peticiones del usuario, tareas encoladas, tareas programadas y otros ganchos web.

El tamaño de la petición está limitado a 10 megabytes al igual que el tamaño de la respuesta.

1.2.5 Ancho de banda de salida (facturable)

La cantidad de datos enviado por la aplicación en respuesta a una petición.

Esto incluye:

- Datos servidos en respuesta tanto a peticiones seguras como no seguras por los servidores de aplicaciones, los servidores estáticos o el Blobstore.
- Datos enviados en los mensajes de correo.
- Datos enviados a través de XMPP o la Channel API.
- Datos de salida a peticiones HTTP del servicio URL Fetch.

1.2.6 Ancho de banda de entrada (facturable)

La cantidad de datos recibidos por la aplicación a través de peticiones. Cada petición entrante HTTP no puede ser mayor de 32 MB.

Esto incluye:

- Datos recibidos por la aplicación en peticiones tanto seguras como no seguras.
- La cantidad de datos agregados al servicio Blobstore
- Datos recibidos en respuesta a peticiones HTTP por el servicio de recuperación de URLs.

1.2.7 Límites de CPU

La CPU es compartida por múltiples manejadores de peticiones en cierto servidor al mismo tiempo. App Engine previene que un manejador de petición utilice la CPU de tal forma que afecte el desempeño de otras aplicaciones al asignar más recursos cuando se requieran, tal como el servir las aplicaciones desde servidores alternos. También está limitado para evitar que picos en su uso causen problemas.

El uso de la CPU puede ser medido como el número de ciclos de procesador que le lleva realizar su trabajo, en unidades como megaciclos. Sin embargo, la velocidad de la CPU puede variar de un servidor de aplicaciones a otro. Mientras la misma cantidad de trabajo es realizada en cierta CPU para un número dado de megaflops, el impacto total en el desempeño podría variar (tal vez en una forma que no sea percibida por el usuario).

De tal modo que en lugar de megaflops, App Engine mide el uso de CPU en “minutos de CPU”. Un minuto de CPU es el número de megaflops que pueden ser realizados por un procesador estándar en un minuto. El procesador estándar en este caso es un procesador Intel x86 a 1.2 Ghz.

App Engine trata de balancear la carga de tal forma que cada aplicación tenga una tasa consistente de atención de la CPU. Pero si un manejador de peticiones usa demasiado tiempo de CPU, el servidor de aplicaciones puede estrangular la asignación de CPU. En ese caso, el manejador de peticiones consume los mismos minutos de CPU, sólo que requiere más ciclos de reloj para realizarlo.

La cuota de CPU incluye dos tipos de recursos: tiempo de CPU usado en el servidor de aplicaciones, y tiempo de CPU usado en el almacén de datos. Otros servicios no contribuyen a la cuota de tiempo de CPU; ellos manejan los recursos de CPU en otras formas. Un servidor de aplicaciones no consume tiempo de CPU mientras espera la respuesta de una llamada a un servicio.

1.2.8 Almacén de datos (Datastore)

La cuota del Almacén de Datos se refiere a los datos almacenados por la aplicación en el Datastore Maestro/Esclavo, la cola de tareas y el Blobstore. La cuota High Replication Storage se refiere a los

datos almacenados por la aplicación en el Datastore High Replication. Otras cuotas en la sección "Datastore" en la pantalla Quota Details de la consola de Administración se refieren a servicios del Datastore específicamente.

1.2.9 Datos almacenados

El número total de datos almacenados en el datastore de entidades e índices correspondientes, datos almacenados en la cola de ejecución y en el Blobstore.

Es importante el hacer notar que los datos almacenados en el datastore pueden incurrir en una ligera sobrecarga. Esta sobrecarga depende del número y tipo de las propiedades asociadas, e incluyen el espacio usado por los índices automáticos y los personalizados. Cada entidad almacenada en el datastore requiere los siguientes metadatos:

- La llave de la entidad, incluyendo el tipo, el ID o el nombre de la llave y las llaves de los ancestros de la entidad.
- El nombre y valor de cada propiedad. Debido a que el datastore carece de esquema, el nombre de cada propiedad debe ser almacenada con la propiedad y el valor de la misma para cada una de las entidades.
- Cualquier índice automático o personalizado de algún renglón (row) que se refiere a esta entidad. Cada tupla contiene el tipo de la entidad, y cualquier número de valores de propiedad dependiendo de la definición del índice y la llave de la entidad.

1.2.10 Número de índices

El número de índices en el datastore que existen para la aplicación. Estos incluyen índices creados en el pasado y que ya no aparecen en la configuración de la aplicación pero que no han sido eliminados utilizando el comando `vacuum_indexes` de AppCfg.

Nota: Como se mencionó anteriormente, el Datastore es una constante, no se otorga una nueva cantidad de espacio a la medianoche.

1.2.11 Url Fetch

App Engine puede comunicarse con otras aplicaciones o acceder otros recursos en la red (tal como web services) por medio de la extracción (*fetch*) de URLs. Una aplicación puede utilizar este servicio para solicitar peticiones HTTP o HTTPS y recibir respuestas.

La cuota se encuentra dividida en tres categorías:

- *Llamadas a la API URL Fetch*. El número total de veces que la aplicación accede al servicio de URL Fetch para realizar peticiones HTTP o HTTPS.

- *Datos enviados a URL Fetch*. La cantidad de datos enviados al servicio URL Fetch en las peticiones. Esto también cuenta en la cuota de Ancho de Banda de Salida.

- *Datos recibidos por el URL Fetch.* La cantidad de datos recibidos en respuesta al servicio URL fetch. Esto también cuenta en el Ancho de Banda de Entrada.

1.3 Límites de servicio

Cada servicio tiene su propio conjunto de cuotas y límites. Tal como los límites que se aplican al sistema en general, algunos pueden ser incrementados utilizando una cuenta de facturación y un presupuesto, tales como el número de destinatarios a los cuales se les puede enviar emails. Otros límites existen para proteger la integridad del servicio, tal como el número de transformaciones de imágenes realizado por el servicio de imágenes.

No se enlistarán los límites de los servicios aquí, éstos pueden ser consultados en la documentación oficial.

Dos límites particulares aplican a todos los servicios: el tamaño de la llamada al servicio y el tamaño de la respuesta del servicio, ambos están limitados a 1 megabyte. Esto impone un límite inherente en el tamaño de las entidades del datastore y los valores en el memcache. No obstante las peticiones entrantes pueden contener hasta 10 megabytes, solamente 1 megabyte de los datos puede ser almacenado usando una única llamada al datastore o al memcache.

Los límites en el tamaño de las llamadas a los servicios también aplican a la API de lotes del datastore. Llamadas de extracción en lotes o de almacenaje de múltiples entidades en una única llamada al servicio. El tamaño total de una llamada en lote debe de ajustarse dentro del límite de 1 megabyte a la llamada de servicio.

1.3.1 Límites en el despliegue

Dos límites afectan el tamaño y estructura de los archivos de nuestra aplicación. Un archivo de aplicación no puede ser mayor a 32 megabytes. Esto aplica a archivos de recursos (código, configuración) así como archivos estáticos. También, el número total de archivos para una aplicación no puede ser mayor a 10,000, incluyendo los archivos de recursos y los archivos estáticos. Adicionalmente el tamaño total de todos los archivos no puede exceder 150 megabytes.

Las aplicaciones en Java tienen una solución para reducir el conteo de archivos del código de la aplicación: los JARs. Si la aplicación tiene muchos archivos .class, es posible el ponerlos en un archivo JAR utilizando la utilidad jar incluida con el kit de desarrollo de Java.

También, con Java, se debe asegurar el uso de las directivas `<static-files>` y `<resource-files>` en el archivo *appengine-web.xml* para excluir los archivos apropiados. Por defecto, todos los archivos fuera de WEB-INF/ pertenecen a ambos grupos, y son contados *dos veces*, una por cada grupo. El límite de conteo de archivo es el total del conteo de ambos grupos.

Existe un límite en el número de veces que la aplicación es subida al servidor por parte del desarrollador. La cuota actual es 1000 por día.

Capítulo 2. Manejo de recursos web

Una aplicación web es una aplicación que responde a peticiones a través de la Web. Idealmente la aplicación responde de manera rápida, realizando la menor cantidad de trabajo requerido para regresar una respuesta. Muchas de las aplicaciones interactúan con los usuarios en tiempo real, por lo que una respuesta rápida significa que el usuario tiene que esperar menos tiempo para que una acción sea completada.

Un beneficio adicional de una aplicación que responde de manera rápida se encuentra en su facilidad de escalamiento. Entre menos cantidad de trabajo realiza una aplicación por petición más fácil resulta el distribuirla entre varios servidores.

AppEngine fue diseñado para aplicaciones web que responden de manera rápida a las peticiones. Una aplicación que responde dentro de diez milisegundos lo está haciendo bastante bien. Ocasionalmente, una aplicación podría llevar más tiempo en entregar una respuesta, por ejemplo, cuando realiza actualizaciones en la base de datos o realiza consultas que podrían regresar varios resultados. Si una aplicación toma un largo tiempo en entregar una respuesta de manera rutinaria, App Engine relega las peticiones más lentas en favor de las que responden más rápido.

2.1 La arquitectura de AppEngine

La primera parada de una petición entrante es el frontend de AppEngine. Una balanceador de carga, el cual es un sistema dedicado para la distribución de peticiones de manera óptima, se encuentra distribuido entre varias máquinas, se encarga de encaminar la peticiones a uno de muchos servidores frontend. El frontend determina la aplicación para la cual está dirigida la petición a partir del nombre de dominio, ya sea un dominio de Google Apps o un subdominio de appspot.com. Después de lo cual consulta la configuración de la aplicación y determina el siguiente paso.

La configuración de la aplicación describe como deben los frontend el tratar la petición basado en la trayectoria de la URL. Una URL puede ser a un archivo estático que debe ser entregado al cliente de manera directa, tal como una imagen o un archivo JavaScript. O una URL puede dirigirse a una manejador de peticiones, código de aplicación que es invocado para determinar la respuesta a la petición. El archivo de configuración es subido junto con el código de nuestra aplicación.

Si la URL de una petición no encuentra ninguna coincidencia en nuestro archivo de configuración, los frontend regresan un código HTTP 404 “Not Found” al cliente. Los frontend regresan un mensaje genérico. Si se desea que los clientes reciban un mensaje personalizado o por lo menos más descriptivo al ingresar a un recurso no válido, es posible el realizar un mapeo de todas las URL a un manejador de peticiones en la aplicación que se encargue de determinar si la petición es válida y de no ser así indicarlo al usuario.

Si la URL de la petición encuentra alguna coincidencia en los archivos estáticos de la aplicación, los frontend dirigen la petición a los manejadores de archivos estáticos. Los servidores están dedicados a manejar la entrega de contenido estático, con una topología de red y mecanismo de caché optimizado

para la entrega rápida de recursos que no cambian de manera seguida. Se le indica a App Engine de los recursos estáticos de la aplicación en el archivo de configuración de la misma. Cuando la aplicación se sube, éstos archivos son insertados en los servidores estáticos.

Si la URL de la petición coincide con un patrón de los manejadores de peticiones de nuestra aplicación, los frontend envían la petición a los servidores de aplicación. El pool de servidores de aplicaciones comienza una nueva instancia de la aplicación en el servidor o reutiliza una instancia existente si existe alguna presente. El servidor invoca la aplicación al llamar al manejador de peticiones correspondiente con la URL de la petición, de acuerdo al archivo de configuración.

El servidor invoca el manejador de peticiones y espera por la respuesta. El servidor maneja los recursos locales disponibles para la aplicación, incluyendo los ciclos de CPU, memoria y tiempo de ejecución, y se asegura que la aplicación no consuma los recursos del sistema en una manera que interfiera con las demás aplicaciones.

El código de aplicación se ejecuta dentro de un *ambiente en tiempo de ejecución*, una abstracción por encima del hardware y sistema operativo que provee acceso a los recursos del sistema y otros servicios. El ambiente en tiempo de ejecución es una “caja de arena”, una pared de arena que permite a la aplicación el utilizar solamente las características del servidor que pueden escalar sin interferir con otras aplicaciones. Por ejemplo, la caja de arena previene a la aplicación de escribir en el sistema de archivos, o el realizar conexiones de red innecesarias a otros hosts.

El manejador de peticiones prepara la respuesta, después de lo cual regresa y termina. La aplicación no envía ningún tipo de dato al cliente hasta que el manejador de peticiones termina, por lo cual no puede mandar flujos de datos o mantener una conexión abierta por un largo tiempo. Cuando el manejador termina, el servidor de aplicaciones regresa la respuesta, y la respuesta es completa.

Los frontend, los servidores de aplicaciones y los servidores estáticos son gobernados por una “aplicación maestra”. Entre otras cosas, la aplicación maestra es responsable de instalar nuevas versiones del software de la aplicación, configuración y actualizar la versión “por defecto” que es presentada por el dominio de la aplicación. Las actualizaciones de una aplicación se propagan de manera rápida, pero no son atómicas en el sentido de que solamente el código de una versión de la aplicación se está ejecutando en un momento dado. Si la versión por defecto es cambiada, todas las peticiones que comenzaron antes del cambio son permitidas de completarse usando esa versión del software.

2.2 Configurando los frontend

El control de como los frontend encaminan las peticiones a la aplicación se realiza usando archivos de configuración. Estos archivos residen junto al código de la aplicación y archivos estáticos en el directorio de la aplicación. Cuando se instala la aplicación, todos los archivos son subidos juntos como una entidad lógica.

Una aplicación Java consiste de archivos empaquetados en un formato estándar llamado WAR (Web Application Archive). El estándar WAR especifica la disposición de la estructura de un directorio para la aplicación web de Java, incluyendo la ubicación de varios archivos de configuración, clases compiladas de Java, archivos JAR, archivos estáticos y otros archivos auxiliares. Las herramientas de

App Engine generalmente esperan que el WAR sea un directorio en el sistema de archivos.

Las aplicaciones servlet de Java utilizan un archivo llamado “descriptor de instalación” (deployment descriptor) para especificar como el servidor invoca la aplicación. Este archivo usa un formato XML, el cual es parte de la especificación del estándar servlet. En un WAR, el descriptor de instalación es un archivo llamado *web.xml* el cual reside en un directorio llamado WEB-INF, el cual se encuentra dentro del directorio raíz del WAR.

El descriptor de instalación le dice a los servidores de App Engine la mayoría de lo que necesitan saber, pero no todo. Para el resto, App Engine utiliza un archivo llamado *appengine-web.xml*, también en el directorio WEB-INF, el cual también usa una sintaxis XML.

2.2.1 App IDs y Versiones

Cada aplicación App Engine tiene un ID que distingue de manera única a la aplicación de las demás. Para registrar un ID para una nueva aplicación se utiliza la Consola de Administración. Una vez que se tiene un ID, se agrega al archivo de configuración de la aplicación de tal forma que las herramientas de desarrollo tengan conocimiento que los archivos en la carpeta raíz de la aplicación pertenecen a la aplicación con ese ID.

La configuración de la aplicación también incluye un identificador de versión. Tal como el ID de la aplicación, el identificador de versión es asociado con los archivos de la aplicación cuando esta es subida. App Engine mantiene un conjunto de archivos y configuraciones de los frontends para cada una de las distintas versiones usadas durante una subida de la aplicación. Si no se cambia la versión de la aplicación en la configuración cuando es subida, los archivos existentes de esa versión son reemplazados.

Cada versión distinta de la aplicación es accesible en su propio dominio, de la siguiente forma:

version-id.latest.app-id.appspot.com

Cuando se tienen múltiples versiones de una versión subida en AppEngine, se puede usar la consola de Administración para seleccionar cual de las versiones es la que deseamos que el público ingrese. La Consola llama a esto la versión “por defecto”. Cuando un usuario visita el dominio de Google Apps (y subdominio configurados), o el dominio appspot.com sin el identificador de versión, el usuario ve la versión por defecto.

Todas las versiones de una aplicación acceden el mismo almacén de datos, memcache, y otros servicios y todas las versiones comparten el mismo conjunto de servicios. Es posible el tener hasta 10 versiones activas. Se pueden eliminar versiones previas utilizando la Consola de Administración.

Los Ids de la aplicación y los identificadores de versión pueden contener números, letras en minúsculas y guiones.

El ID de la aplicación y el identificador de versión aparecen en el archivo *appengine-web.xml*. El ID de la aplicación es especificado en el elemento XML `<application>`, y el identificador de versión es especificado con `<version>`.

2.2.2 Manejadores de peticiones

Una aplicación web Java mapea patrones URL a servlets utilizando el descriptor de instalación *(web.xml)*. La configuración de un servlet se realiza en dos pasos: la declaración del servlet y el mapeo del servlet.

El elemento <servlet> declara el servlet. Incluye un `<servlet-name>`, y un nombre para los propósitos de referirse al servlet en otro lugar en el archivo, y el `<servlet-class>`, el nombre de la clase que implementa el servlet.

Para mapear un servlet a un patrón URL, se usa el elemento `<servlet-mapping>`. Un mapeo incluye el `<servlet-name>` que coincide con la declaración de un servlet y un `<url-pattern>`.

El patrón URL coincide con la trayectoria URL. Se permite utilizar el carácter * al principio o final del patrón para representar cero o más de coincidencias de cualquier carácter.

El orden en el cual aparecen los mapeos de URL no es significante. El patrón “más específico” gana, determinado por el menor número de comodines en el patrón.

2.3 Archivos estáticos y archivos de recursos

La mayoría de las aplicaciones web tienen un conjunto de archivos que son entregados tal cual a todos los usuarios, y no cambian mientras la aplicación es utilizada. Estos pueden ser imágenes, hojas de estilo CSS, código JavaScript o páginas HTML que no tienen contenido dinámico. Para acelerar la entrega de estos archivos y mejorar el tiempo de entrega de la página, App Engine usa servidores dedicados para contenido estático.

Tanto el proceso de instalación como los frontend deben estar enterados de cuales de los archivos de la aplicación son estáticos. El proceso de instalación envía los archivos estáticos a servidores dedicados. Los frontend recuerdan cuales URL se refieren a archivos estáticos de tal forma que puedan encaminar las peticiones de esas trayectorias a los servidores apropiados.
Para salvar espacio y reducir la cantidad de datos necesarios cuando se instala una nueva aplicación, los archivos estáticos no son guardados en los servidores de aplicaciones. Esto significa que el código de aplicación no puede acceder a los contenidos de archivos estáticos usando el sistema de archivos.

Los archivos que son guardados en los servidores de aplicación se conocen como “archivos de recursos” (resources files). Estos pueden incluir archivos de configuración de la aplicación , plantillas de páginas web, u otro tipo de contenido estático que es necesario para la aplicación pero que no

requiere entregarse a los usuarios de manera directa. El código de aplicación puede acceder a estos archivos leyéndolos del sistema de archivos. El código en si mismo es también accesible de esta forma.

La estructura del directorio WAR para una aplicación web de Java mantiene todo el código de la aplicación, JARs y la configuración en un directorio llamado WEB-INF. Típicamente, los archivos fuera de WEB-INF representan recursos que el usuario puede acceder de manera directa, incluyendo archivos estáticos y JSP. Las trayectorias URL para estos recursos son equivalentes a las trayectorias de estos archivos dentro del WAR.

Por ejemplo, un WAR tiene los siguientes archivos:

```
main.jsp
forum/inicio.jsp
forum/registro.jsp
imagenes/logo.png
imagenes/okbutton.png
imagenes/border.png
home/about.html
```

Esta aplicación tiene cuatro archivos estáticos: tres imágenes PNG y un archivo HTML llamado about.html. Cuando la aplicación es subida, estos cuatro archivos son guardados en los servidores estáticos. Los frontend saben como encaminar trayectorias URL equivalentes a estas trayectorias de archivos (tales como /imagenes/logo.png) a los servidores de archivos estáticos.

Por defecto, *todos* los archivos en un WAR son guardados en los servidores de aplicación, y son accesibles por los servidores de aplicación por medio del sistema de archivos. Esto incluye los archivos que son identificados como estáticos y son guardados en servidores estáticos. En otras palabras, todos los archivos son considerados como archivos de recursos, y todos los archivos excepto los JSPs y el directorio WEB-INF son considerados como archivos estáticos.

Se puede especificar cuales archivos son considerados como archivos de recursos y cuales son considerados archivos estáticos usando el archivo *appengine-web.xml*, con las etiquetas `<resource-files>` y `<static-files>` respectivamente. Estos elementos pueden contener elementos `<include>` y un elemento `<exclude>` que modifica el comportamiento por defecto de incluir todos los archivos. Por ejemplo

```
<resource-files>
      <exclude path=”/imagenes/**” />
</resource-files>
```

Esto excluye los contenidos del directorio /imagenes y todos los subdirectorios del conjunto de archivos de recursos. Esto reduce la cantidad de datos que son guardados en los servidores de aplicación cuando se comienza una nueva instancia de la aplicación, a expensas de no ser capaces de acceder a estos archivos dentro de la aplicación. El patrón ** coincide con cualquier número de caracteres en archivos y nombres de directorios, incluyendo subdirectorios.

Otro ejemplo:

```
<static-files>
```

```
        <exclude path="/**.xml" />
        <include path="/sitemap.xml"/>
    </static-files>
```

Esto excluye todos los nombres que terminan en .xml del conjunto de archivos estáticos, excepto *sitemap.xml*. Tal vez los archivos XML sólo están destinados para la aplicación, pero queremos asegurarnos que los motores de búsqueda puedan ver el mapa del sitio.

Los archivos contenidos en el directorio WEB-INF/ son siempre considerados archivos de recursos. Ellos no pueden ser incluidos como archivos estáticos o excluidos del conjunto de archivos estáticos.

Los navegadores se apoyan en los servidores web para decirles el tipo de archivos que le serán entregados. Los servidores estáticos determinan el contenido de tipo MIME de un archivo en base a la extensión del nombre de archivo. Por ejemplo, un archivo cuyo nombre termina en .jpeg es servido como un tipo MIME de image/jpeg. El servidor tiene un conjunto de mapeos de extensiones de archivo a tipos MIME. Podemos especificar mapeos adicionales usando el elemento `<mime-mapping>` en el descriptor de instalación (*web.xml*).

Los navegadores también requieren el saber si es seguro el almacenar un archivo en la caché y por cuanto tiempo. El servidor estático puede sugerir la duración de expiración de la caché cuando el archivo es entregado (no obstante el navegador puede ignorar esta sugerencia). Se puede especificar que un conjunto de archivos estáticos deben permanecer en caché por un periodo de tiempo al incluir el atributo *expiration* en el elemento `<include>` de *appengine-web.xml*:

```
    <static-files>
        <include path="images/**" expiration="30d" />
    </static-files>
```

El valor de *expiration* es la duración especificada como un número, las unidades son d, h,m y s, donde d es días, h horas, m minutos y s segundos. Se pueden agregar valores de múltiples unidades al especificarse las mismas separadas por espacios, por ejemplo: 3d 12h.

Capítulo 3. Capa de persistencia

El diseño de aplicaciones altamente escalables puede ser truculento. Se requiere de varios servidores que sean capaces de atender las peticiones de cientos de miles de usuario, lo cual crea una sobrecarga en la replicación de los datos en cada uno de ellos así como un incremento en el tiempo de respuesta de cada petición.

Google App Engine se encarga de todas las tareas relacionadas a la disponibilidad, sincronización y garantía de escritura de los datos, liberando al usuario de esta responsabilidad. App Engine es soportado principalmente por dos servicios: Bigtable y Google File System (GFS).

En un sistema de base de datos relacional, una forma de optimizar las consultas es creando un *índice* en la(s) columna(s) involucradas. App Engine utiliza una solución basada en índices, lo cual le permite buscar la respuesta en una lista de respuestas previamente calculadas; App Engine puede realizar esto ya que conoce con antelación el tipo de preguntas que serán realizadas a la aplicación.

App Engine mantiene una lista de *todas* las consultas que la aplicación realizará. Debido a que el almacén de datos requiere realizar una simple búsqueda de un índice para cada consulta, la aplicación regresa los datos con prontitud. Y para grandes cantidades de datos, App Engine puede distribuir los datos y los índices a través de varias máquinas, y obtener los resultados de regreso de cada una de ellas sin necesidad de realizar una operación costosa de agregación.

El uso de índices no es una solución para todos los escenarios posibles, como muchas de las tecnologías tiene ventajas y desventajas inherentes, las cuales deben de ser analizadas para el escenario que deseamos resolver, por ejemplo, esta estrategia no resulta adecuada para aplicaciones que requieren realizar un procesamiento sofisticado en los datos (como en la minería de datos) o que prefieran consultas poderosas aunque lentas. Sin embargo, gran parte de las aplicaciones web saben con antelación el tipo de escenarios en los que serán utilizadas y son adecuadas para su uso con App Engine.

3.1 Introducción al almacén de datos (Datastore)

El almacén de datos de App Engine provee de almacenaje robusto y escalable para aplicaciones web, con énfasis en el desempeño de lecturas y consultas. Este almacén carece de esquema por lo que es responsabilidad de la aplicación el garantizar el modelo de datos que requiera la aplicación.

App Engine provee dos diferentes opciones las cuales se diferencian por su disponibilidad y la consistencia que garantizan en los datos:

- En la opción de *Replicación con Alta Disponibilidad (High Replication)*, los datos son replicados a través de los diferentes centros de datos usando un sistema basado en el algoritmo Paxos. Esta opción provee una muy alta disponibilidad para lecturas y escrituras (al costo de alta latencia en las escrituras). La mayoría de las consultas son eventualmente consistentes. Los costos de los recursos de almacenaje son iguales a los encontrados en la opción

Maestro/Esclavo.

- La opción *Maestro/Esclavo (Master/Slave)* utiliza un sistema de replicación maestro-esclavo, el cual de manera asíncrona replica los datos mientras son escritos a un centro de datos físico. Ya que solamente un centro de datos es el maestro para la escritura en cualquier momento, esta opción ofrece una alta consistencia para todas las lecturas y consultas, al costo de periodos temporales en los cuales se encuentra fuera de servicio debido a problemas en el centro de datos o periodos de mantenimiento planificados.

Una de las diferencias principales del almacén de datos de App Engine con el de otras base de datos relacionales se encuentra en como describe la relaciones entre objetos de datos. Dos entidades del mismo tipo pueden tener diferentes propiedades. Diferentes entidades pueden tener propiedades con el mismo nombre, pero tener diferentes tipos de datos. Mientras la interfaz del almacén de datos tiene muchas de las características de las bases de datos relacionales, las características únicas del almacén de datos implican una forma diferente de diseñar y manejar los datos para tomar ventaja de la habilidad que proporciona para escalar de manera automática.

El almacén de datos provee una API de bajo nivel con operaciones simples en las entidades, incluyendo `get`, `put`, `delete` y `query`. Es posible el utilizar la API de bajo nivel para implementar otras interfaces adaptadoras o simplemente utilizarla de manera directa en nuestras aplicaciones. Adicionalmente a la API de bajo nivel, App Engine provee implementaciones de las interfaces *Objetos de Datos Java* (JDO, Java Data Objects) y de la *API de Persistencia de Java* (JPA, Java Persistence API) para el manejo de los objetos de datos.

3.2 Entidades y propiedades

Un objeto de datos en el almacén de datos de App Engine es conocido como una *entidad*. Una entidad tiene una o más *propiedades*, valores con nombre de uno de varios tipos de datos.

El almacén de datos proporciona una cantidad considerable de tipos de datos, los cuales internamente son soportados utilizando ocho tipos de datos básicos, sin contar los tipos no indexados (text y blob) . El almacén de datos soporta los tipos de datos adicionales utilizando alguno de los tipos de datos básicos, después de lo cual maneja de manera automática la conversión entre la representación interna y la que es vista por la aplicación. Por ejemplo, un valor de date-time es de hecho almacenado como un entero.

La tabla 4.1 describe los ocho tipos de datos indexables soportados. Los tipos son listados en su orden relativo, del primero al último.

Tabla 4.1 Tipos de datos soportados por Google App Engine

Tipo de dato	Tipo de dato en Java	Ordenamiento
El valor null	`Null`	-
Integer y date-time	Long (otros tipos de datos enteros se amplían), `java.util.Date`, `datastore.Rating`	Numeric (en el caso de datetime se realiza de manera cronológica)

Boolean	Boolean (true o false)	False, después true
Byte string	Datastore.ShortBlob	Byte order
Unicode string	`java.lang.String`, `datastore.Category`, `datastore.Email`, `datastore.IMHandle`, `datastore.Link`, `datastore.PhoneNumber`, `datastore.PostalAddress`	El orden determinado por Unicode
Número de punto flotante (floating-point)	Double	Numeric
Geographical point	Datastore.GeoPt	Por latitud, después por longitud (números de punto flotante)
Una cuenta de Google	Users.User	Por dirección de email, ordenamiento Unicode
Entity key	Datastore.key	Kind (byte string), después ID (numeric) o por name (byte string)

Cada entidad tiene también una *llave* que identifica de manera única a la entidad. La más simple de las llaves tiene un *tipo* y un *ID de entidad* provisto por el almacén de datos. El tipo categoriza la entidad, de tal forma que podamos consultarla de manera más fácil. El ID de la entidad también puede ser una cadena provista por la aplicación.

Una aplicación puede obtener una entidad del almacén de datos utilizando su llave o al realizar una consulta que coincida con las propiedades de la entidad. Una consulta puede regresar cero o más entidades, y puede regresar los resultados ordenados por los valores de las propiedades. Una consulta también puede limitar el número de resultados regresados por el almacén de datos para conservar memoria en tiempo de ejecución.

3.3 Consultas y tipos

App Engine provee dos mecanismos generales para realizar operaciones en la capa de datos: una interfaz de bajo nivel que permite utilizar todas las características de BigTable y otra que proporciona un nivel de abstracción más alto por medio de dos interfaces estándar: Java Data Objects (JDO) y Java Persistance API (JPA).

En el caso de la API de bajo nivel, es posible realizar consultas por medio de las llaves (Key) de cada entidad, sin embargo, la mayoría de las veces uno desconoce la llave de las entidades y lo que conoce son los criterios que deben de cumplir, en el caso de los alumnos de un colegio, podríamos buscar todos aquellos alumnos que tengan cierto apellido o que su edad se encuentre dentro de algún rango.

Una *consulta* (*query*) obtiene entidades del almacén de datos que cumplen el criterio especificado por una serie de condiciones. La consulta especifica un tipo de entidad, cero o más *filtros* basados en los valores que tienen las propiedades de la entidad y cero o más *criterios de ordenamiento*. Cuando una consulta es ejecutada, obtiene todas las entidades del tipo especificado que satisfacen todos los filtros dados, ordenadas por el criterio especificado.

El almacén de datos Maestro-esclavo y el almacén de datos de replicación de alta disponibilidad tienen

diferentes garantías cuando se trata de la consistencia de las consultas. Por defecto:

- El almacén de datos maestro-esclavo es fuertemente consistente para todas las consultas.
- El almacén de datos de replicación de alta disponibilidad es fuertemente consistente por defecto para las consultas dentro de un *grupo de entidad*. Con la replicación de alta disponibilidad, las consultas sin ancestro son siempre eventualmente consistentes.

Las consultas basadas en los valores de las propiedades solamente pueden regresar entidades de un sólo tipo (*kind*). Este es el principal propósito de los tipos: el determinar cuales entidades son consideradas juntas como posibles resultados a una consulta. En la práctica los tipos corresponden a la noción intuitiva de que cada entidad del mismo tipo nominal representa el mismo tipo de datos, por ejemplo, cuando uno piensa en el concepto de alumno (tipo) uno espera que la consulta regrese solamente alumnos, no departamentos, mesas y otro tipo de entidades. A diferencia de otros sistemas de bases de datos, es opción de la aplicación el reforzar esta consistencia si es deseado, y la aplicación puede divergir de esto si lo considera apropiado.

3.3.1 Las APIs de consulta en Java

La API Java de bajo nivel provee una clase `Query` para permitir la construcción de consultas y una clase `PreparedQuery` para la extracción y devolución de entidades del almacén datos. Las interfaces de alto nivel proporcionan cada una su propio lenguaje de consulta los cuales son JDOQL (Java Data *Objets Query Language*) en el caso de JDO y JPQL (*Java Persistance Query Language*) en el caso de JPA.

Para realizar una consulta, se crea una instancia de la clase `Query` (del paquete `com.google.appengine.api.datastore`), proporcionando el nombre del tipo de las entidades a consultar como un argumento al constructor. Se llaman métodos del objeto Query para agregar filtros y criterios de ordenamiento. Para ejecutar la consulta, se pasa el objeto Query a un método de la instancia `DatastoreService`. Éste método regresa un objeto `PreparedQuery`, el cual se puede manipular para obtener los resultados.

La consulta es desempeñada hasta que se intenta consultar los resultados usando el objeto `PreparedQuery`. Si se consultan los resultados usando un iterador por medio de los métodos `asIterable()` o `asIterator()`, el acto de iterar causa que la API obtenga los resultados en grupos.

Los métodos antes mencionados aceptan un argumento opcional, un objeto `FetchOptions`, que controla que resultados serán regresados. Las opciones pueden incluir un *desplazamiento*, un número de resultados a descartar antes de cualquier resultado, un *límite* y un máximo número de resultados a regresar.

Nota: El proceso de desplazamiento y especificación del máximo número de resultados realizado por este servicio es bastante ineficiente, ya que realiza el desplazamiento y extracción de los resultados una vez que ha obtenido todos los resultados de la consulta.

A diferencia de la interfaz `iterator`, la cual obtiene los resultados en grupos, este método regresa

todos los resultados con una sola llamada al servicio. El método requiere que un límite sea especificado usando `FetchOptions`, hasta un máximo de 1000 entidades.

Si es probable que una consulta regrese solamente un resultado, o si solamente se está interesados en el primer registro, llamando al método `asSingleEntity()` se regresa el resultado y regresa un objeto `Entity`, o `null`.

3.3.2 Consultas de Llaves solamente

Podemos realizar consultas por las llaves de las entidades que coincidan con la consulta en lugar de las entidades completas usando la API de bajo nivel de Java del almacén de datos. Para indicar que la consulta debe de regresar solamente las llaves, se realiza una llamada al método `setKeysOnly()` en el objeto `Query`:

```
Query q = new Query("Alumno");
q.setKeysOnly();
```

Cuando una consulta es establecida solamente para regresar las llaves, los resultados de la consulta son objetos `Entity` sin ninguna propiedad establecida. Se pueden obtener las llaves de estos objetos utilizando el método `getKey()`:

```
PreparedQuery pq = ds.prepare(q);
for (Entity result : ps.asIterable()) {
      Key k = result.getKey();

      // ....

}
```

3.4 Introducción a los índices

Para cada consulta que la aplicación desempeña, App Engine mantiene un índice, una tabla única de posibles respuestas para una consulta. Específicamente, mantiene un índice para un conjunto de consultas que usan los mismos filtros y criterios de ordenamiento, posiblemente con diferentes valores para los filtros.

Considérese la siguiente consulta:

```
SELECT * FROM Alumno WHERE nombre = 'oscar'
```

Para realizar esta consulta, App Engine utiliza un índice el cual contiene las llaves de cada entidad `Alumno` y los valores de la propiedad `nombre` de la entidad, ordenados por los valores de la propiedad `nombre` en orden ascendente.

Para encontrar todas las entidades que cumplen las condiciones de la consulta, App Engine encuentra el primer renglón en el índice que coincida, después escanea la tabla hacía abajo hasta encontrar el primer

renglón que no coincida. Regresa las entidades mencionadas en todos los renglones en este rango (sin contar los renglones que no coinciden), en el orden que aparecen en el índice. Debido a que el índice ya se encuentra ordenado, se garantiza que todos los resultados de la consulta se encuentran en renglones consecutivos de la tabla.

App Engine utilizará este mismo índice para desempeñar otras consultas con estructura similar pero con diferentes valores, tal como la siguiente consulta:

```
SELECT * FROM Alumno WHERE nombre = 'javier'
```

Este mecanismo de consulta es rápido, aún con un número muy grande de entidades. Las entidades y los índices son distribuidos sobre múltiples equipos, y cada equipo escanea su propio índice en paralelo con los demás. Cada máquina regresa los resultados a App Engine mientras escanea su propio índice, y App Engine entrega el conjunto de resultados finales, en orden, tal como si todos los resultados se hubiesen encontrado en un único índice de tamaño mayor.

Otra razón por la cual las consultas son rápidas tiene que ver con la forma en que el almacén de datos encuentra el primer renglón coincidente. Debido a que los índices se encuentran ordenados, el almacén de datos puede usar un algoritmo eficiente para encontrar el primer renglón con la coincidencia. En el caso común, encontrar el primer renglón toma aproximadamente la misma cantidad de tiempo sin tomar en consideración el tamaño del índice. En otras palabras, la velocidad de una consulta no se ve afectada por el tamaño del conjunto de datos.

App Engine actualiza todos los índices relevantes cuando el valor de una propiedad cambia. En este ejemplo, si la aplicación regresa una entidad `Alumno`, cambia el `nombre`, y después salva la entidad con una llamada al método `put()`, App Engine actualiza el renglón apropiado en el índice previo. También mueve el renglón si es necesario de tal forma que el orden de los índices se preserve. La llamada a `put()` no regresa hasta que todos los índices apropiados son actualizados.

De manera similar, si la aplicación crea una nueva entidad `Alumno` con una propiedad `nombre` o borra una entidad `Alumno`, con una propiedad `nombre`, App Engine actualiza el índice. En contraste, si la aplicación actualiza un `Alumno` pero no cambia la propiedad `nombre`, crea o borra un `Alumno` que no tiene una propiedad `nombre`, App Engine no actualiza el índice `nombre` debido a que no es necesaria la actualización.

App Engine mantiene dos índice similares al ejemplo anterior para cada nombre de propiedad y tipo de entidad, uno con los valores de las propiedades ordenados en orden ascendente y otro con los valores en orden descendente. App Engine también mantiene un índice de las entidades de cada tipo. Estos índices satisfacen algunas consultas sencillas, y App Engine también los usa internamente para propósitos internos de mantenimiento.

Para otras consultas, se debe indicar a App Engine que índices preparar. Lo cual se realiza usando un archivo de configuración, el cual es subido junto con el código de la aplicación. En el caso de Java, este archivo es *WEB-INF/datastore-indexes.xml*. El servidor web de desarrollo que viene con el SDK añade automáticamente sugerencias a este archivo si encuentra consultas que aún no tiene configuradas y que no son posibles de realizar por los índices automáticos.

3.4.1 Índices automáticos y consultas sencillas

Como se ha mencionado anteriormente, App Engine mantiene dos índices por cada propiedad de cada tipo, una con los valores en orden ascendente y otra con los valores en orden descendente. App Engine construye estos índices de manera automática, ya sea que ellos estén o no mencionados en el archivo de configuración de índices. Estos índices automáticos satisfacen los siguientes tipos de consultas utilizando renglones consecutivos:

- Una consulta sencilla con todas las entidades de cierto tipo, sin filtros o criterios de orden.
- Un filtro en una propiedad utilizando el operador de igualdad (=).
- Filtros usando los operadores mayor que o menor que (>, >=, <, <=) en propiedades sencillas.
- Un criterio de orden, ascendente o descendente, y sin filtros, o filtros solamente en la misma propiedad usada con el criterio de orden.
- Filtros o criterios de orden en las llaves de la entidad.
- Consultas sin tipo con o sin filtros sobre la llave.

3.4.2 Todas las entidades de una tipo

La más simple de las consultas solicita todas las entidades de cierto tipo en cualquier orden. Utilizando GQL[1], una consulta de todas las entidades del tipo `Alumno` se ve del siguiente modo:

```
SELECT * FROM Alumno
```

App Engine mantiene un índice que convierte los tipos a llaves de entidad. Este índice es ordenado utilizando un orden determinístico para las llaves de las entidades, de tal forma que la consulta regresa un “orden de llave”. El tipo de una entidad no puede ser cambiada una vez que ha sido creada, de tal forma que este índice es actualizado solamente cuando las entidades son creadas o borradas.

3.4.3 Un filtro de igualdad

Considere la siguiente consulta, la cual pregunta por cada entidad `Alumno` con una propiedad `noCuenta` con una valor entero de 1500:

```
SELECT * FROM Alumno WHERE noCuenta = 1500
```

Esa consulta utiliza un índice de las entidades `Alumno` con la propiedad `noCuenta`, ascendente (uno de los índices automáticos). Utiliza un algoritmo eficiente para encontrar el primer renglón con un `noCuenta` igual a 1500. Después sigue escaneando hacia abajo los índices hasta que encuentre el primer renglón con un `noCuenta` que no es igual a 1500. Los renglones consecutivos desde la primer coincidencia hasta la última coincidencia representan todas las entidades `Alumno` con una propiedad `noCuenta` igual al entero

1 GQL es un lenguaje similar a SQL que permite recuperar entidades o las llaves de las mismas del almacén de datos de App Engine, una descripción más completa se encuentra en [13]

3.4.4 Filtros Mayor-Que y Menor-Que

La siguiente consulta pregunta por cada `Alumno` con una propiedad `noCuenta` cuyos valores sean mayores que el entero 500

```
SELECT * FROM Alumno WHERE noCuenta > 500
```

Esto requiere el uso de un índice de las entidades `Alumno` con la propiedad `noCuenta`, ascendente, también un índice automático. Tal como un filtro de igualdad, encuentra el primer renglón en el índice cuyo `noCuenta` es mayor a 500. En el caso de un mayor-que, debido a que la tabla se encuentra ordenada por `noCuenta` en orden ascendente, cada renglón a partir de este punto hasta el fondo de la tabla es un resultado para la consulta.

Similarmente, la siguiente consulta pregunta por cada `Alumno` con un `noCuenta` menor que 800:

```
SELECT * FROM Alumno WHERE noCuenta < 800
```

App Engine usa el mismo índice (`noCuenta`, ascendente), y la misma estrategia: encuentra el primer renglón que coincide con la consulta, en este caso el primer renglón. Esto escanea hasta el siguiente renglón que no coincide con la consulta, el primer renglón cuyo `noCuenta` es mayor que o igual a 800. Los resultados son representados por cada uno de los registros por debajo de ese renglón.

Finalmente, considere una consulta con valores de `noCuenta` entre 300 y 650:

```
SELECT * FROM Alumno WHERE noCuenta > 300 AND noCuenta < 650
```

De nuevo, el mismo índice y estrategia prevalecen: App Engine realiza un barrido de arriba hacia abajo, encontrando la primera coincidencia y los siguientes renglones que no coinciden, regresando las entidades representadas por todo lo que se encuentra en medio.

Si los valores usados con los filtros no representan un rango válido, tal como `noCuenta < 500 AND noCuenta > 1000`, el planeador de la consulta se da cuenta de esta situación y no se molesta en realizar la consulta, ya que sabe que no tiene resultados.

3.4.5 Un criterio de ordenamiento

La siguiente consulta pregunta por todas las entidades `Alumno`, ordenas por el `noCuenta`, de menor a mayor:

```
SELECT * FROM Alumno ORDER BY noCuenta
```

Como antes, la consulta usa un índice de las entidades `Alumno` con las propiedades `noCuenta` ordenadas de manera ascendente. Si tanto esta consulta como la consulta previa de igualdad fueran

desempeñadas por la aplicación, ambas consultas utilizarían el mismo índice. Esta consulta usa el índice para determinar el orden en el cual regresar las entidades `Alumno`, comenzando en la parte superior de la tabla y moviéndose hacia abajo hasta que la aplicación para de obtener resultados, o hasta el final de la tabla. Recordando que cada entidad `Alumno` con una propiedad `noCuenta` es mencionada en esta tabla.

La siguiente consulta es similar a la anterior, pero pregunta por las entidades ordenadas por el `noCuenta` del mayor al menor:

```
SELECT * FROM Alumno ORDER BY noCuenta DESC
```

Esta consulta no puede utilizar el mismo índice como antes, porque los resultados están en el orden incorrecto. Para esta consulta, los resultados deben comenzar con la consulta con el `noCuenta` más alto, de tal forma que la consulta necesita un índice donde este resultado está en el primer renglón. App Engine provee un índice automático para propiedades sencillas en orden descendente para este propósito.

Si la consulta con un criterio de ordenamiento en una propiedad sencilla también incluye filtros en esa propiedad, y sin otros filtros, App Engine aún requiere solamente el índice automático para llevar a cabo la consulta.

3.4.6 Consultas sin tipo

Adicionalmente a realizar consultas en las entidades de un cierto tipo, el almacén de datos nos permite realizar un conjunto limitado de consultas en entidades de todos los tipos. Las consultas sin tipo no pueden usar filtros o criterios de ordenamiento en las propiedades. Ellas pueden, sin embargo; usar filtros de igualdad y desigualdad en las llaves (Ids o nombres).

3.4.7 Restricciones en las consultas

La naturaleza del mecanismo de consulta por medio de índices impone ciertas restricciones sobre lo que una consulta es capaz de hacer. A continuación se describen tales restricciones.

3.4.7.1 El filtrar u ordenar una propiedad requiere que dicha propiedad exista

Si una propiedad tiene una consulta con filtro o criterio de ordenamiento, la consulta regresa solo aquellas entidades en el almacén de datos que tienen un valor (incluyendo `null`) para esa propiedad.

Como se mencionó anteriormente, el almacén de datos carece de esquema, por lo cual dos entidades del mismo tipo pueden tener propiedades diferentes. Un filtro en una propiedad puede solamente coincidir con un valor por esa propiedad. Si una entidad no tiene valor para una propiedad usada en el filtro o criterio de orden, la entidad es omitida del índice construido para la consulta.

3.4.7.2 No usar filtros para buscar entidades que carecen de cierta propiedad

No es posible el realizar una consulta por aquellas entidades que carecen de una propiedad dada. Una alternativa es crear un modelo fijo con todas las propiedades a utilizar y establecer sus valores en `null` por defecto, después de lo cual se crea un filtro para las entidades con un valor `null` en dicha propiedad.

3.4.7.3 Filtros de desigualdad son permitidos en una propiedad solamente

Una consulta puede utilizar solamente filtros de desigualdad (<, <=, >=, >, !=) en una propiedad en todos sus filtros. Los filtros pueden combinar comparaciones de igualdad (==) para diferentes propiedades en la misma consulta, incluyendo filtros con una o más condiciones de desigualdad en una propiedad.

3.4.8 Índices personalizados y Consultas Complejas

Todas las consultas que no son cubiertas por los índices automáticos deben de tener un índice correspondiente definido en el archivo de configuración de índices de la aplicación. Nos referimos a estos como "*índices personalizados*", en contraste con los "*índices automáticos*". App Engine requiere estas sugerencias ya que construir cada posible índice para cada posible combinación de propiedades y criterios de ordenamiento tomaría una cantidad prohibitiva de espacio y tiempo, y es muy probable que la aplicación sólo requiera una fracción de estas posibilidades.

En particular, las siguientes consultas requieren índices personalizados:

- Una consulta con múltiples criterios de ordenamiento.
- Consultas con criterios de ordenamiento descendente sobre las llaves.
- Consultas con filtros de desigualdad y filtros en los ancestros.
- Consultas con uno o más filtros de desigualdad en una propiedad y uno o más filtros de igualdad en las demás propiedades.

3.4.9 Múltiples criterios de ordenamiento

Debido a la forma en que se construyen los índices automáticos, estos son suficientes para cumplir con un criterio de ordenamiento. Si dos entidades tienen el mismo valor para una propiedad, dichas entidades aparecerán en renglones consecutivos en el índice ordenadas por el valor de su llave. Si se requiere ordenar estas entidades utilizando otro criterio se requiere de un índice con más información.

La siguiente consulta pregunta por todos las entidades `Alumno`, ordenados primero por la fecha de nacimiento en orden ascendente y después por su número de cuenta en orden descendente.

```
SELECT * FROM Alumno ORDER BY fecha_nacimiento ASC, no_cuenta DESC
```

El índice que esta consulta requiere no es construido de manera automática por App Engine, por lo cual es un índice personalizado que requiere una tabla de llaves `Alumno`, valores de `fecha_nacimiento` y `no_cuenta` ordenados de acuerdo a la consulta. En Java el archivo de configuración tendrá las siguientes líneas:

```
<datastore-index kind="Alumno" ancestor="false" source="auto">
    <property name="fechaNacimiento" direction="asc"/>
    <property name="noCuenta" direction="desc"/>
</datastore-index>
```

El orden en el cual las propiedades aparecen en el archivo de configuración importa. Este es el orden en el cual los renglones son ordenados: primero por `fechaNacimiento` de manera ascendente, después por `noCuenta` de manera descendente.

3.4.10 Filtros en múltiples propiedades

La siguiente consulta, pregunta por cada `Alumno` con número de cuenta menor al entero 500 y un nombre igual a la cadena 'Oscar'

```
SELECT * FROM Alumno WHERE nombre='oscar' AND noCuenta < 5000
```

Para que sea posible el escanear un conjunto de resultados contiguos que cumplan con los criterios de ambos filtros, el índice debe contener columnas con los valores para estas propiedades. Las entidades deben ser ordenadas primero por `nombre`, después por `noCuenta`, el archivo de configuración para esta consulta será.

```
<datastore-index kind="Alumno" ancestor="false" source="auto">
    <property name="nombre" direction="asc"/>
    <property name="noCuenta" direction="asc"/>
</datastore-index>
```

Es importante el hacer notar que la secuencia de ordenamiento es importante, ya que los resultados deben de aparecer en renglones consecutivos, si se ordena primero por `noCuenta` y después por nombre existiría la posibilidad de tener resultados en renglones no consecutivos.

El mantener los índices ordenados para consultas con filtros de igualdad y desigualdad requiere que sigamos ciertas reglas, las cuales se mencionan a continuación.

La primera Regla de Filtros de Desigualdad: Si una consulta utiliza filtros de desigualdad en una propiedad y filtros de igualdad en una o más propiedades, el índice debe ser primero ordenado por las propiedades usadas en los filtros de igualdad, después por las propiedades utilizadas en los filtros de desigualdad.

Esta regla tiene un corolario para aquellas consultas que tienen tanto filtros de desigualdad como criterios de ordenamiento. Considere la siguiente consulta posible:

```
SELECT * FROM Alumno WHERE noCuenta < 2000 ORDER BY fechaNacimiento DESC
```

La consulta anterior requiere un índice personalizado, el cual requiere de las columnas `noCuenta` y `fechaNacimiento`, pero ¿cuál de las dos columnas debe ser ordenada primero para satisfacer la consulta?

La Primera Regla implica que `noCuenta` debe ser ordenada primero, pero la consulta solicita que los resultados sean ordenados por `fechaNacimiento` de modo descendente. Si los índices fueran ordenados primero por fecha de nacimiento y después por número de cuenta, los resultados podrían no encontrarse de manera adyacente.

Para evitar la confusión, App Engine requiere que el orden correcto sea establecido de manera explícita en la consulta:

```
SELECT * FROM Alumno
WHERE noCuenta < 2000
ORDER BY noCuenta, fechaNacimiento DESC
```

La Segunda Regla de Filtros de Desigualdad: Si una consulta usa filtros de desigualdad en una propiedad y criterios de ordenamiento en una o más propiedades, los índices deben ser ordenados primero por la propiedad usada en los filtros de desigualdad (en cualquier dirección), después por los criterios de ordenamiento. Para evitar confusión, la consulta debe establecer los criterios de ordenamiento explícitamente.

Considere ahora una entidad `Empleado`, la cual tiene un `nivel` y un `noEmpleado`. La siguiente consulta intenta obtener todos los empleados que tengan un nivel menor a 25 y un `noEmpleado` menor a 2500

```
SELECT * FROM Empleado WHERE nivel < 25 AND noEmpleado < 2500
```

Esta consulta requiere un índice personalizado ordenado primero por nivel y después por `noEmpleado`, sin embargo, no existe un índice posible que pueda satisfacer esta consulta completamente usando renglones consecutivos. Esta no es una consulta válida en App Engine.

La Tercera Regla de Filtros de Desigualdad: Una consulta no pueda usar filtros de desigualdad en más de una propiedad.

Una consulta *puede* usar múltiples filtros de desigualdad en la misma propiedad, de tal forma que pueda probar un rango de valores.

3.4.11 Múltiples Filtros de Igualdad

Para aquellas consultas que ocupan solamente filtros de igualdad, es fácil el imaginar índices personalizados que los satisfagan: Por ejemplo:

```
SELECT * FROM Empleado WHERE nombre = 'ruben' AND noEmpleado = 50
```

Un índice personalizado que contenga estas propiedades, ordenadas en cualquier secuencia y dirección,

satisfará los requerimientos de la consulta. Pero App Engine tiene otro truco debajo de la manga para este tipo de consultas. Para consultas que utilizan solamente filtros de igualdad y no criterios de ordenamiento, en lugar de escanear una sola tabla con todos los valores, App Engine puede escanear los índices automáticos de propiedades sencillas para cada propiedad, y regresar los resultados tal como los encuentra. App Engine puede realizar una "*mezcla de unión (merge join)*" de los índices en propiedades-únicas que satisfacen este tipo de consultas.

En otras palabras, el almacén de datos no requiere un índice personalizado para realizar consultas usando solamente filtros de igualdad y sin criterios de ordenamiento. Si se agrega un índice personalizado apropiado al archivo de configuración, el almacén de datos lo utilizará. Pero un índice personalizado no es requerido, y la característica de configuración de índices automáticos del servidor de desarrollo no lo agregará si no existe.

Más específicamente, el almacén de datos *usualmente* no requiere un índice automático para tales consultas. El algoritmo que ensambla los resultados se desempeña bastante bien en casos típicos. Pero en casos extremos no puede llevar a cabo la consulta en una cantidad razonable de tiempo. Si el algoritmo no ha ensamblado el número de resultados solicitados durante la cantidad de tiempo permitida la consulta regresará un error.

Para comprender la problemática en casos extremos, considere como el algoritmo realizará la siguiente consulta usando solamente índices en propiedades-únicas:

```
SELECT * FROM Kind WHERE a=1 AND b=2 AND c=3
```

Recuerde que cada una de estas tablas contiene un renglón para cada entidad con el conjunto de propiedades, con campos para cada llave de la entidad y los valores de la propiedad. La tabla es ordenada primero por el valor, y después por la llave. El algoritmo toma ventaja del hecho que los renglones con el mismo valor son consecutivos, y dentro de ese bloque consecutivo, los renglones son ordenados por llave.

Para realizar la consulta, el almacén de datos usa los siguientes pasos:

- El almacén de datos revisa el índice de `a` para el primer renglón con el valor de 1. La entidad cuya llave esta en este renglón es una candidata, pero todavía no es un resultado confirmado.
- Después revisa el índice de `b` del primer renglón cuyo valor es 2 *y* cuya llave es mayor que o igual a la llave candidata. Otros renglones con un valor de 2 podrían aparecer por encima de este renglón en el índice de `b`, pero el almacén de datos sabe que no son candidatos porque el primer escaneo en `a` determino el candidato con la llave mínima.
- Si el almacén de datos encuentra la llave candidata en la región que coincide con b, esa llave es aún una candidata, y el almacén de datos procede con una revisión similar en el índice para `c`. Si el almacén de datos no encuentra el candidato en el índice de `b` pero encuentra otra llave con un valor mayor que coincide con el resultado, esa llave se convierte en la nueva candidata, y procede a revisar la nueva candidata en el índice `c`. (Eventualmente regresará a revisar `a` con la nueva candidata antes de decidir si es un resultado. Si no se encuentra ni la llave candidata o un renglón que cumpla con el criterio con una llave mayor, la consulta es completa.
- Si un candidato es encontrado que cumpla todos los criterios en todos los índices, el candidato es regresado como resultado. El almacén de datos comienza la búsqueda por un nuevo candidato, usando la llave candidata previa como la llave mínima.

Una característica principal de este algoritmo es que encuentra resultados en el orden en el cual son regresados: ordenados por la llave. El almacén de datos no requiere el compilar una lista completa de posibles resultados para esta consulta -posiblemente millones de entidades- y después ordenarlos para determinar cual debe ir primero. También, el almacén de datos puede detener la búsqueda tan pronto tiene suficientes resultados para completar la consulta, la cual es siempre un número de entidades limitadas.

El caso problema, el cual no puede ser manejado por este algoritmo antes de que el almacén de datos exceda el tiempo de operación, surge cuando el algoritmos tiene que rechazar demasiados candidatos. Cada escaneo toma tiempo, y un candidato rechazado representa varias comparaciones que no producen un resultado. Si pasa demasiado tiempo antes de que el algoritmo encuentre suficientes resultados para satisfacer la consulta, el almacén de datos termina la consulta.

Utilizar un índice personalizado para una consulta con solo filtros de igualdad y sin criterios de ordenamiento mejora la velocidad de la consulta, pero aumenta el tiempo de actualización de las propiedades mencionadas en la consulta. Si una consulta lanza `NeedIndexError`, se debe utilizar un índice personalizado.

3.5 Propiedades no establecidas y no indexadas

Anteriormente se ha mencionado que los tipos `text` y los valores `blob` no son indexados. Otra forma de decir esto es que, para el propósito de índices, una propiedad de tipo `text` o un valor `blob` es tratada como si no fuera establecida. Si una aplicación realiza una consulta usando un filtro o criterio de ordenamiento en una propiedad que siempre es establecida para ser `text` o un valor `blob`, la consulta siempre regresará sin resultados.

Algunas veces es útil el almacenar propiedades de otros tipos de datos, y excluirlas de los índices. Esto salva espacio en las tablas de índices, y reduce la cantidad de tiempo que toma el salvar la entidad.

En la API de Java podemos establecer que una propiedad permanece sin indexar usando el método `setUnindexedProperty()` del objeto `Entity`, en lugar del método `setProperty()`. Una entidad puede tener sólo una propiedad con cierto nombre, de tal forma que una propiedad que no esta indexada sobrescribe una que esta indexada y vice versa. También es posible declarar que las propiedades permanezcan sin indexar utilizando las interfaces JDO y JPA.

Si se necesita que una entidad califique como resultado para cierta consulta, pero no tiene sentido en nuestro modelo de datos el establecer cada propiedad que utilizamos en la consulta, usamos el valor `null` para representar el caso “no-valor”, y siempre es establecido. Las interfaces Java JDO y JPA hacen sencillo el asegurar que las propiedades siempre tienen valores.

3.6 Consultas y propiedades multivaluadas

El almacén de datos de App Engine puede almacenar más de un valor para una única propiedad. Con propiedades *multivaludadas* (MVP por sus siglas en inglés Multi Valued Properties) es posible el modelar una propiedad como un conjunto de datos.

Una de las características más útiles de las propiedades multivaluadas es como satisfacen un filtro de igualdad en una consulta. El motor de consultas del almacén de datos considera una propiedad multivaluada igual al valor de un filtro si *cualquiera* de las propiedades de los valores es igual al valor del filtro. Esta habilidad para determinar la pertenencia significa que las MVPs son útiles para representar conjuntos.

Las propiedades multivaluadas mantienen el orden de valores, y pueden repetir elementos. Los valores pueden ser de cualquiera de los tipos del almacén de datos, y una propiedad única puede tener valores de diferentes tipos.

Es posible resumir las características de las propiedades multivaluadas como sigue:

1. Una propiedad multivaluada aparece en un índice con un renglón por valor.
2. Todos los renglones en un índice son ordenados por valores, posiblemente distribuyendo los valores de las propiedades para una única entidad a través del índice.
3. La primera ocurrencia de una entidad en una búsqueda de un índice determina su lugar en un conjunto de resultados para una consulta.

Juntos, estos hechos explican que ocurre cuando una consulta ordena sus resultados por una propiedad multivalor. Cuando los resultados son ordenados por una propiedad multivaluada en orden ascendente, el valor más pequeño de la propiedad determina su ubicación en los resultados. Cuando los resultados son ordenados en orden descendente, el valor más grande para una propiedad determina su ubicación.

3.6.1 MVPs y el Planeador de Consultas

El planeador de consultas intenta el ser inteligente e ignorar los aspectos de la consulta que son redundantes o contradictorios. Por ejemplo, `a = 3 AND a = 4` normalmente no regresarían resultados, de tal forma que el planeador de consultas atrapa estos casos y no se molesta en realizar trabajo que no es necesario. Sin embargo, la mayoría de estas técnicas de normalización no aplican a propiedades multivaluadas. En este caso, la consulta podría estar preguntado, “Esta MVP tiene un valor que es igual a 3 y otro valor igual a 4?”. El almacén de datos recuerda cual propiedades son MVPs (incluso aquellas que terminan con uno o cero valores), y nunca toma atajos que producirían resultados incorrectos.

Pero existe una excepción. Una consulta que tiene tanto filtros de igualdad como criterios de ordenamiento desechará el criterio de ordenamiento. Si una consulta pregunta por `a = 3 ORDER BY a DESC` y `a` es una propiedad con valor único, el criterio de ordenamiento no tiene efecto porque todos los valores en el resultado son idénticos. Para un MVP, sin embargo, `a = 3` probará por pertenencia, y dos MVPs que cumplen esa condición no son necesariamente idénticas.

El almacén de datos descarta el criterio de ordenamiento en este caso de cualquier forma. El no hacerlo requeriría mucha información del índice y resultaría en una explosión de índices en casos en los que de otro modo sobrevivirían. Como siempre, el criterio de ordenamiento es determinístico, pero no será el orden solicitado.

3.7 Configuración de índices

Una aplicación específica los índices personalizados que requiere en un archivo de configuración. Cada definición de índice incluye un tipo, y los nombres y criterios de ordenamiento de las propiedades a incluir. Un archivo de configuración puede tener cero o más definiciones de índices.

La mayoría de las veces, se puede dejar el mantenimiento de este archivo al servidor web de desarrollo. El servidor de desarrollo observa las consultas que la aplicación realiza, y si una consulta requiere un índice personalizado y tal índice no está definido en el archivo de configuración, el servidor agrega las configuraciones apropiadas de manera automática.

El servidor de desarrollo no removerá la configuración de los índices. Si está seguro que la aplicación no requerirá más un índice, puede editar el archivo manualmente y removerlo.

Es posible el deshabilitar la característica de configuración automática de índices. Hacer esto causa que el servidor web de desarrollo se comporte como App Engine: si una consulta no tiene un índice y requiere uno, la consulta falla.

La configuración de índices es global a todas las versiones de la aplicación. Todas las versiones de una aplicación comparten el mismo almacén de datos, incluyendo los índices. Si se instala una versión de la aplicación y el archivo de configuración ha cambiado, App Engine usará la nueva configuración de los índices para todas las versiones.

3.7.1 Configuración de índices para java

Para las aplicaciones Java, se agrega una configuración de índices en un archivo llamado *datastore-indexes.xml*, en el directorio war/WEB-INF/ . Este es un archivo XML con un elemento raíz llamado `<datastore-indexes>`. Este contiene cero o más elementos `<datastore-index>`, uno para cada índice.

Cada `<datastore-index>` especifica el tipo usando el atributo `kind`. También tiene un atributo `ancestor`, el cual es `true` si el índice soporta consultas con filtros en los ancestros, y `false` de otro modo.

Un `<datastore-index>` contiene uno o más elementos `<property>`, uno para cada columna en el índice. Cada `<property>` tiene un atributo name (el nombre de la propiedad) y un atributo direction (`asc` por ascendente, `desc` por descendente). El orden de los elementos es significativo: el índice es

ordenado por la primera columna primeramente, después por la segunda columna y así sucesivamente.

El elemento raíz `<datastore-indexes>` tiene un atributo llamado `autogenerate`. Si es `true`, o si la aplicación no tiene un archivo *datastore-indexes.xml*, el servidor Java de desarrollo generará una nueva configuración de índices cuando sea necesario para una consulta. Si es `false`, el servidor de desarrollo se comporta como App Engine: si una consulta necesita un índice que no es definido, la consulta falla.

El servidor de desarrollo no modifica el archivo *datastore-indexes.xml*. En su lugar, genera un archivo separado llamado *datastore-indexes-auto.xml*, en el directorio *WEB-INF/appengine-generated*. El archivo de configuración completo es el total de los dos archivos de configuración.

El servidor Java nunca removerá la configuración de un índice del archivo automático, de tal modo que si se requiere el borrar un índice, se deberá realizar del archivo automático.

3.8 Transacciones

El almacén de datos de App Engine soporta *transacciones*. Una transacción es una operación o conjunto de operaciones que se realizan de manera atómica, esto es, todas las operaciones ocurren o ninguna de ellas.

En App Engine una transacción opera en una o más entidades. Se garantiza que cada transacción es atómica, lo cual significa que las operaciones nunca son aplicadas de manera parcial, si la operación fracasa, todos los cambios registrados desde el inicio de la transacción hasta ese momento son descartados. Una operación podría fallar cuando:

- Muchos usuarios tratan de modificar un grupo de entidades de manera simultánea.
- La aplicación alcanza los límites de sus recursos.
- El almacén de datos encuentra un error interno.

En todos estos casos, la API del almacén de datos lanza una excepción, sin embargo no todas las excepciones significan que la operación ha fallado, se podrían recibir las siguientes excepciones `DatastoreTimeoutException`, `ConcurrentModificationException` O `DatastoreFailure Exception` en casos donde las transacciones han sido solicitadas y eventualmente serán aplicadas de manera satisfactoria.

Las transacciones son una característica opcional, no son requeridas para realizar operaciones en el almacén de datos.

3.8.1 Grupos de entidades

Cada entidad pertenece a un *grupo de entidad*, un conjunto de una o más entidades que pueden ser manipuladas en un sola transacción, este grupo forma una relación padre hijo que determina la forma

en que deben ser almacenadas físicamente las entidades para que sea posible el uso de transacciones, es decir, App Engine almacenará todas las entidades que pertenecen al mismo grupo en la misma parte de la red distribuida.

Cuando una aplicación crea una entidad, puede asignarle otra entidad como el *padre* de la entidad recién creada, eso coloca a la nueva entidad en el mismo grupo que su padre.

Una entidad que carece de padre es una entidad *raíz*. Una entidad que sea la entidad padre de otra entidad puede a su vez tener otra entidad como padre. Una cadena de entidades padres desde la entidad actual hasta la entidad raíz se conoce como la *trayectoria* de la entidad y todos los miembros de la trayectoria son los *ancestros* de la entidad. El padre de una entidad es definido cuando la entidad es creada y no puede ser modificado posteriormente.

Cada entidad con una entidad raíz dada como ancestro se encuentra en el mismo grupo de entidades. Todas las entidades en un grupo son almacenadas en el mismo nodo del almacén de datos. Una simple transacción puede modificar múltiples entidades del mismo grupo agregando nuevos elementos o modificando las propiedades de entidades existentes.

3.8.2 Grupos-cruzados de Transacciones (Cross-Group Transactions)

Las aplicaciones que usan el Almacén de Alta Disponibilidad (High Replication Datastore HRD) pueden realizar transacciones en entidades que pertenecen a diferentes grupos de entidades. Esta característica es llamada *grupos-cruzados* (Cross-Group), o *transacciones XG*.

Las transacciones XG pueden ser usadas a través de 5 grupos de entidades como máximo. Una transacción de este tipo tendrá éxito siempre y cuando ninguna transacción que se ejecute de manera concurrente toque alguna de los grupos de entidad usados en la transacción.

Las transacciones XG se comportan de manera similar a aquellas que son de un único grupo de entidades y difieren de estas por el uso de opciones en la transacción.

3.8.3 Que puede ser realizado en una transacción

El almacén de datos impone varias restricciones en lo que puede ser realizado dentro de una transacción.

Todas las operaciones ejecutadas en el almacén de datos dentro de una transacción deben operar en entidades del mismo grupo de entidad si la transacción es sobre un único grupo o en entidades de hasta máximo cinco grupos. Hay que tener en cuenta que cada entidad raíz pertenece a grupos de entidades diferentes, de tal modo que una única transacción no puede crear u operar en más de una entidad raíz a menos que sea una transacción XG.

Cuando dos o más transacciones intentan de manera simultánea el modificar entidades en uno o más

grupos de entidades comunes, sólo una de ellas puede tener éxito. Mientras una aplicación está aplicando cambios a las entidades de uno o más grupos de entidades, todos los demás intentos en el grupo o grupos fallaran en tiempo de ratificación (commit). Debido a este diseño, usar grupos de entidades limita el número de escrituras concurrentes que es posible realizar en cualquier entidad de los grupos.

Cuando una transacción comienza, App Engine usa *control de concurrencia optimista* al revisar el tiempo en que se realizó la última actualización de los grupos de entidades en la transacción. Cerca de ratificar una transacción para los grupos de entidades, App Engine revisa nuevamente el tiempo en que se realizó la última modificación de los grupos de entidades, si este ha cambiado desde la revisión inicial, App Engine lanza una excepción.

Una aplicación puede realizar una consulta durante una transacción, pero sólo si incluye un filtro de ancestro. (De hecho es posible el realizar una consulta sin filtro de ancestro, pero los resultados no reflejarán ningún estado en particular que sea consistente con la transacción). Una aplicación también puede obtener entidades por medio de su llave durante la transacción.

3.8.4 Aislamiento y Consistencia

El nivel de aislamiento del almacén de datos fuera de una transacción es cercano a `READ_COMMITTED`. Dentro de una transacción, el nivel de aislamiento es `SERIALIZABLE`, específicamente una forma de aislamiento de instantánea.

En una transacción, todas las lecturas reflejan el estado actual y consistente del almacén de datos al momento en que la transacción comenzó. Esto no incluye `puts` y `deletes` previos dentro de la transacción. Las consultas y `gets` dentro de una transacción son garantizadas a ver una única y consistente instantánea del almacén de datos al comienzo de la transacción. Las entidades e índices de los renglones en los grupos de la transacción son completamente actualizados de tal modo que las consultas regresan el resultado completo y correcto de las entidades, sin los falsos positivos o falsos negativos que pueden ocurrir en consultas que se ejecutan fuera de las transacciones.

Esta vista consistente de instantánea también se extiende a lecturas después de escrituras dentro de la transacción. A diferencia de la mayoría de base de datos, las consultas y gets dentro de una transacción en el Datastore *no* ven resultados de escrituras previas dentro de esta transacción. Específicamente, si una entidad es modificada o borrada dentro de una transacción, una consulta o `get` regresa la versión *original* de la entidad, tal como se encontraba al comienzo de la transacción o nada si la entidad aún no existía.

Capítulo 4. El Memcache

El almacenaje de los datos de manera permanente requiere un medio que mantenga los datos incluso después de una falla de luz o de un reinicio del equipo. Los discos duros son un medio de almacenaje permanente, lo cual permite consultar los datos en cualquier momento, sin embargo, el consultar los datos lleva tiempo, se requiere buscar la ubicación de los datos en el disco y colocar la cabeza de lectura correspondiente, si bien el tiempo es relativamente rápido, existen aplicaciones web en las que este tiempo puede ser prohibitivo.

Las aplicaciones web escalables de alto desempeño utilizan a menudo algún tipo de caché distribuido, una memoria caché utiliza un medio *volátil*, a menudo la memoria RAM de los equipos en los cuales se ejecuta, la cual le permite responder de manera rápida y eficiente a las peticiones más comunes de lectura y escritura sin necesidad de realizar un acceso a la base de datos, con el costo de no ver los datos más actuales.

El desarrollo de un caché eficiente, escalable, seguro y altamente concurrente requiere de un sólido conocimiento en concurrencia y en un conocimiento profundo de la aplicación, sin embargo, App Engine proporciona un servicio de memoria caché de manera automática.

El caché puede ser visto como un mapa donde los valores se almacenan en pares llave-valor. Un valor puede tomar hasta un megabyte en tamaño. Una llave es de hasta 250 bytes, la API acepta valores mayores, utilizando un algoritmo hash para convertirlo a un valor de 250 bytes.

El establecer un valor en el caché es una operación atómica: la llave obtiene el nuevo valor o conserva el que tenía (o permanece sin establecerse). El memcache no soporta transacciones como el almacén de datos. Si se obtiene un valor, se trata de establecer un nuevo valor basado en lo que se lee, se podría terminar trabajando con un valor que ya no existe o que ha cambiado desde que se obtuvo. App Engine incluye la habilidad de incrementar y decrementar valores numéricos como una operación atómica.

Una forma común de utilizar el memcache con el almacén de datos es el guardar en el caché las entidades por medio de sus llaves. Cuando se desea el obtener una entidad por su llave, primero se revisa el memcache por la existencia de la llave, y se obtiene el valor asociado (esto se conoce como un acierto de caché), si no se encuentra en el memcache (una falla de caché), se obtiene del almacén de datos y se coloca en el memcache de tal forma que en futuros intentos la puedan encontrar ahí[2].

Debido a que no existe una forma de actualizar tanto el almacén de datos como el memcaché en una sola transacción, existe la posibilidad de que el caché mantenga datos que no estén actualizados. Para minimizar la duración que el memcaché tendrá datos no actualizados, se puede establecer un valor de expiración cuando son guardados en el mismo. Cuando el tiempo límite se cumple, el caché elimina la llave, y una lectura subsecuente resulta en una falla de caché y lanza una búsqueda en el almacén de datos.

2 Esto funciona si la entidad es utilizada de manera constante, de lo contrario podríamos terminar fallando todas las búsquedas en el caché y obteniendo una nueva entidad del almacén de datos en cada consulta, con lo cual el desempeño de la aplicación será peor que si hubiéramos realizado la busqueda al almacén de datos de manera directa.

4.1 La API Java de Memcache

Al igual que ocurre con el almacén de datos, App Engine provee dos mecanismos para acceder al servicio de Memcache, una interfaz de bajo nivel y una interfaz de alto nivel a través de la API JCache. La interfaz de bajo nivel provee de los servicios MemcacheService y AsyncMemcacheService, ésta última obtiene los resultados de manera asíncrona y permite realizar otras operaciones mientras se completa nuestra petición.

El uso principal de una memoria caché es el acelerar el tiempo que tarda la aplicación en realizar ciertas consultas. Si múltiples peticiones realizan las mismas consultas, y los datos que son consultados son pocas veces actualizados, es posible el mejorar el desempeño de la aplicación al almacenar los resultados de dichas consultas en el caché, las peticiones subsecuentes pueden verificar el memcache y realizar la consulta solamente si los resultados no existen o si estos han expirado.

El servicio de la API de Java permite el uso de cualquier objeto que implemente la interfaz *Serializable* como llave o valor.

4.2 Como expiran los datos del caché

Por defecto, los valores almacenados en el memcache se mantienen tanto como es posible. Los valores podrían ser eliminados del caché cuando un nuevo valor es agregado si la memoria disponible en la caché es poca. Cuando esta situación ocurre, los valores que menos se han ocupado recientemente son eliminados primero.

La aplicación puede proveer de un periodo en el cual los datos deben de expirar, el cual es proporcionado cuando los valores son guardados. Este valor puede ser expresado como el número de segundos relativos al momento en que el valor fue agregado o como una media de tiempo Epoch Unix en el futuro (el número de segundos después de la media noche del 1 de Enero de 1970). El valor no permanecerá más allá de este punto, aunque podría ser expulsado por otras razones.

App Engine no garantiza que los valores permanecerán hasta el preciso momento indicado por el usuario ya que los valores podrían expirar por otras razones adicionales a la presión en memoria tales como una falla en el servicio.

Como regla general, una aplicación no debe esperar que los valores en el caché estén siempre disponibles.

4.3 Utilizando Jcache

App Engine soporta Jcache, una propuesta de interfaz estándar para las memorias caché. La propuesta todavía no es oficial.

Con Jcache se permite el guardar y obtener valores, controlar cuando los valores deberán expirar,

inspeccionar el contenido del caché y obtener estadísticas del mismo; de hecho, se pueden utilizar *"listeners"* para agregar funcionamiento adicional cuando un valor es agregado o eliminado del caché.

La API de Jcache es un estándar aún en desarrollo, mientras que App Engine trata de implementar un subconjunto de la misma tan fielmente como es posible, es probable que se desee el utilizar la interfaz de bajo nivel para acceder a más características que las proporcionadas por Jcache.

4.3.1 Obteniendo una instancia del Caché

Resulta posible el utilizar una implementación de la interfaz `net.sf.jsr107cache.Cache` para interactuar con el caché. Se obtiene un caché usando un `CacheFactory`, el cual se obtiene por medio de un método estático del `CacheManager`.

El método `createCache()` del objeto `CacheFactory` toma un `Map` con las opciones de configuración, si las opciones por defecto son suficientes se puede pasar al método un mapa vacío.

4.3.2 Colocando y obteniendo valores

El caché se comporta como un Mapa: se guardan las llaves y valores usando el método `put()`, y se obtienen valores usando el método `get()`. Se puede utilizar cualquier objeto que implemente la interfaz `Serializable` ya sea tanto para la llave como para el valor.

Para colocar múltiples valores, se puede llamar el método `putAll()` con un mapa como su argumento.

Para remover un valor del cache (y expulsarlo de forma inmediata), se llama al método `remove()` con la llave como su argumento. Para eliminar todos los valores del caché, se llama al método `clear()`.

El método `containsKey()` toma una llave, y regresa un valor boolean (true o false) para indicar si un valor con esa llave existe en el caché. El método `isEmpty()` comprueba si el caché está vacío. El método `size()` regresa el número de valores actualmente en el cache.

4.3.3 Configuración del tiempo de vencimiento

Es posible el especificar una fecha de vencimiento para los valores guardados en el caché, la cual es establecida utilizando propiedades de configuración al momento en el cual se crea una instancia del caché. Todos los valores establecidos en dicha instancia usan la misma política de vencimiento.

Las siguientes propiedades controlan el valor de expiración:

- `GCacheFactory.EXPIRATION_DELTA` : los valores expiran dada la cantidad de tiempo especificada relativa al momento en que estos fueron agregados, el valor es un entero que indica el número de segundos.

- `GCacheFactory.EXPIRATION_DELTA_MILLIS` : igual que la opción anterior, sólo que ahora el tiempo es expresado en milisegundos.
- `GCacheFactory.EXPIRATION`: Los valores expiran en la hora y fecha indicada por la instancia `java.util.Date`.

4.3.4 Configurando las opciones al colocar un valor

Por defecto, al establecer un valor en el caché se agrega el valor con la llave dada, y reemplaza un valor si la llave dada existe previamente. Podemos configurar el caché para que solamente agregue el valor (protegiendo valores existentes) o sólo reemplace valores (sin añadirlos).

Las siguientes propiedades controlan la política al intentar agregar un nuevo valor:

- `MemcacheService.SetPolicy.SET_ALWAYS`: inserta el valor si no existe una llave con el valor dado, reemplaza el valor si existe una llave con el valor dado previamente, esta es la configuración por defecto.
- `MemcacheService.SetPolicy.ADD_ONLY_IF_NOT_PRESENT`: agrega el valor si no existe una llave con ese valor previamente y no realiza nada en caso de que la llave exista.
- `MemcacheService.SetPolicy.REPLACE_ONLY_IF_PRESENT`: no realiza nada si no existe una llave con el valor dado, reemplaza un valor existente si una llave con el valor dado existe.

4.4 Características no soportadas de JCache

La API de escucha de Jcache es parcialmente soportada para escuchas (listeners) que pueden ejecutarse durante el procesamiento de llamadas a la API de la aplicación, tales como las funciones de escucha `onPut` y `onRemove`. Los escuchas que requieren procesamiento en segundo plano tal como `onEvict` no son soportados.

Las siguientes son algunas consideraciones a tomar en cuenta acerca del memcache.

- Una aplicación puede comprobar si el caché contiene cierta llave, pero no puede comprobar si el caché contiene un cierto valor (`containsValue()` no está soportado).
- Una aplicación no puede volcar los contenidos de las llaves del caché o sus valores.
- Una aplicación no puede manualmente restablecer las estadísticas del caché.
- El método `put()` no regresa el valor previo para una llave, siempre regresa `null`.

El listado 4.1 muestra un ejemplo del uso de memcache:

Listado 4.1 Uso de memcache

```
package mx.unam.store;

import java.io.IOException;
import java.util.Collections;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.tools.ant.types.resources.selectors.Date;


import com.google.appengine.api.memcache.MemcacheService;
import com.google.appengine.api.memcache.jsr107cache.GCacheFactory;

import net.sf.jsr107cache.Cache;
import net.sf.jsr107cache.CacheException;
import net.sf.jsr107cache.CacheFactory;
import net.sf.jsr107cache.CacheManager;


@SuppressWarnings("serial")
public class CacheServlet extends HttpServlet {

      @Override
      protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {

            try {
                  CacheFactory factory =
CacheManager.getInstance().getCacheFactory();

                  // configurando el cache
                  Map<Object, Object> propiedades = new HashMap<Object, Object>();

                  // los valores expiraran pasada una hora
                  propiedades.put(GCacheFactory.EXPIRATION_DELTA, 3600);

                  // colocar solo si el valor no existe previamente

      propiedades.put(MemcacheService.SetPolicy.ADD_ONLY_IF_NOT_PRESENT, true);

                  // crear el cache
                  Cache cache = factory.createCache(propiedades);

                  // guardar valores en el cache
                  String key = "idValor";
                  String valor = "valor";


                  // podemos colocar en el cache cualquier objeto que implemente
                  // la interfaz Serializable
```

```
                cache.put(key, valor);

                // obtener valor del cache
                String objFromCache = (String)cache.get(key);

                // mostrar el valor
                resp.setContentType("text/plain");
                resp.getWriter().println(objFromCache);

            } catch (CacheException e) {
                e.printStackTrace();
            }

        }

}
```

Capítulo 5. Manejo de peticiones web

Las aplicaciones web hoy en día raramente pueden verse como elementos auto contenidos, muchas de ellas requieren el comunicarse con otros recursos en la web para obtener la información que necesitan y poder llevar a cabo sus funciones, si las aplicaciones se alojan en el mismo servidor y utilizan la misma base de datos, la comunicación es trivial, pero este escenario es raro y generalmente las aplicaciones se comunican utilizando peticiones web de manera mutua.

La caja de arena (sandbox) dentro de la cual se ejecuta la aplicación no permite abrir puertos arbitrarios a través de sockets, de tal forma que la comunicación con otras aplicaciones externas requiere de otro mecanismo para llevarse a cabo. Para tal efecto App Engine proporciona la API de Extracción de URLs (URL Fetch), la cual permite la comunicación con otros hosts en internet por medio de peticiones HTTP o HTTPS, el servicio utiliza la infraestructura de red de Google para propósitos de eficiencia y escalabilidad.

El servicio soporta la extracción de URL usando el protocolo HTTP, y usando HTTP con SSL (HTTPS). Otros métodos algunas veces asociados con URLs (tal como FTP) no son soportados. En el caso de las conexiones HTTPS, el servicio no puede autentificar el destino de la conexión, ya que no existe una cadena de certificados de confianza. El servicio acepta todos los certificados. Esto significa que el protocolo de conexión en si mismo no puede protegerse ante ataques de “hombre en medio”, no obstante el tráfico se encuentra encriptado.

Debido a que el servicio está basado en la infraestructura de Google, el servicio hereda unas cuantas restricciones que fueron puestas en el diseño original del servicio HTTP. El servicio soporta los cinco tipos más comunes de acciones HTTP (`GET`, `POST`, `PUT`, `HEAD` y `DELETE`) pero no permite otros o el uso de acciones que no son estándar. También, sólo puede conectarse a puertos TCP estándar para cada método (80 para HTTP, y 443 para HTTPS). El proxy utiliza HTTP 1.1 para conectarse al host remoto.

5.1 Extrayendo URLs utilizando java.net

Es posible el utilizar la clase `java.net.URLConnection` y las clases relacionadas de la librería estándar de Java para realizar conexiones HTTP y HTTPS desde nuestra aplicación. App Engine implementa esta interfaz utilizando el servicio URL Fetch, por lo cual la aplicación no realiza conexiones de manera directa.

Una forma simple de obtener los contenidos de una página que se encuentra en cierta URL es creando un objeto `java.net.URL`, después llamar a su método `openStream()`. El método maneja los detalles de creación de conexión, realiza una petición HTTP GET y recupera los datos de la respuesta.

Nótese que el servicio URL Fetch ya ha almacenado en un buffer la respuesta entera de la aplicación para el momento en que la aplicación comienza a leer. La aplicación lee los datos de respuesta de la memoria, no de un flujo de red proporcionado por el socket o el servicio.

Para peticiones más sofisticadas, se puede llamar el método `openConnection()` del objeto URL para

obtener un objeto `URLConnection` (ya sea un `HttpURLConnection` o `HttpsURLConnection`, dependiendo de la URL). Se puede preparar este objeto con información adicional antes de emitir la petición.

Es posible el utilizar otras características de la interfaz `URLConnecion`, siempre y cuando operen dentro de la funcionalidad proporcionada por la API del servicio. Notablemente, el servicio URL Fetch no mantiene peticiones persistentes dentro del host remoto, de tal forma que las características que regresen tal conexión no trabajarán.

5.1.1 Realizando peticiones

Una aplicación puede solicitar una URL por medio de HTTP o HTTPS. La URL determina el esquema a utilizar: http//... o https://... .

La URL a ser extraída puede utilizar cualquier número de puerto en los siguientes rangos: 80-90, 440-450, 1024-65535. Si el puerto no es mencionado en la URL, se deduce del esquema: http://... es puerto 80, https://... utiliza el puerto 443.

Como una medida de seguridad contra solicitudes accidentales que ocasionen un ciclo en la aplicación, el servicio de extracción de URLs no puede extraer una URL que corresponda al manejador de peticiones que realiza la extracción. No obstante existe la posibilidad de crear una recursión infinita por otros medios por lo que se requiere de mucha precaución en su uso.

Podemos indicar un tiempo límite para una petición, la cantidad de tiempo máximo que el servicio esperará por una respuesta. Por defecto, el tiempo límite que espera el servicio una respuesta del host remoto es de 5 segundos. El tiempo máximo es de 60 segundos para peticiones HTTP y de 10 minutos para peticiones en la cola de tareas o tareas que han sido programadas por medio de un cronómetro. Cuando se utiliza la interfaz `URLConnection`, el servicio usa el tiempo especificado en la conexión (`setConnectTimeout()`) más el tiempo de lectura (`setReadTimeout()`) como tiempo máximo.

El servicio URL Fetch soporta tanto peticiones síncronas como peticiones asíncronas. Con una petición síncrona, la llamada de la API para extraer una URL espera hasta que el host remoto regresa un resultado, después de lo cual regresa el control a la aplicación. La aplicación puede especificar la cantidad máxima de tiempo a esperar cuando realiza la llamada. Si la cantidad de tiempo de espera es excedida, la llamada lanza una excepción.

Una petición asíncrona al servicio URL Fetch comienza una petición y regresa inmediatamente con un objeto. La aplicación puede realizar otras tareas mientras la ULR es extraída. Cuando la aplicación necesita los resultados, llama a un método en el objeto, el cual espera a que la petición termine si es necesario y después regresa los resultados. La aplicación puede tener hasta 10 llamadas asíncronas al servicio URL Fetch. Si cualquiera de las peticiones se encuentra pendiente mientras el manejador de la petición exista, el servidor de aplicaciones espera por todas las peticiones restantes hasta que regresen o alcancen su tiempo límite antes de regresar una respuesta al usuario.

En Java, la interfaz asíncrona solamente está disponible cuando se usa la API de bajo nivel de manera directa. El método `fetchAsync()` regresa un objeto `java.util.concurrent.Future<HTTP`

`Response> .`

5.1.2 Encabezados de la petición

Una aplicación puede establecer encabezados HTTP para la petición saliente.

Cuando se envía una petición HTTP POST, si el encabezado Contet-Type no es establecido explícitamente, el encabezado es establecido a `x-www-form-urlencoded`. Este es el tipo de contenido usado por las formas web.

La petición de salida puede contener parámetros de URL, un cuerpo de petición y encabezados HTTP. Unos cuantos encabezados no pueden ser modificados por razones de seguridad, lo cual significa principalmente que una aplicación no puede realizar una petición mal formada, tal como peticiones cuyo encabezado Content-Length no refleje de manera precisa el valor actual el contenido de la longitud del cuerpo de la petición.

A continuación se listan los encabezados que no pueden ser modificados por la aplicación:

- `Content-length`
- `Host`
- `Vary`
- `Via`
- `X-Forwarded-For`
- `X-ProxyUser-IP`

Estos encabezados son establecidos a valores precisos por App Engine, según corresponda. Por ejemplo, App Engine calcula el encabezado Content-Length a partir de los datos de la petición y lo agrega a la petición antes de enviarla.

Para establecer un encabezado HTTP en la petición de salida, se llama al método `setRequestProperty()` del objeto `HttpURLConnection`.

```
connection.setRequestProperty("X-MyApp-Version", "2.7.3");
```

5.2 Respuestas

El servicio URL Fetch regresa todos los datos de la respuesta, incluyendo los códigos de respuesta, los encabezados y el cuerpo.

Por defecto, si el servicio URL Fetch recibe una respuesta con un código de redirección, el servicio seguirá el redireccionamiento. El servicio seguirá hasta un máximo de 5 respuestas de redirección, después de los cuales regresará el recurso final. Se puede usar la API para decir al servicio URL Fetch que no siga redireccionamientos y sólo regresar una respuesta de redirección a la aplicación.

Por defecto, si la respuesta entrante excede el límite de tamaño máximo para una respuesta, el servicio lanzará una excepción. Se puede solicitar a la API que trunque la respuesta en lugar de lanzar una excepción.

5.3 Servicio URL Fetch y el servidor de desarrollo

Cuando la aplicación se ejecuta en el servidor de desarrollo, las llamadas al servicio URL Fetch son manejadas de manera local. El servidor de desarrollo obtiene las URLs al contactar el host remoto directo desde la computadora donde se ejecuta el entorno, usando cualesquiera configuración de red que la computadora esté utilizando para acceder a internet.

Cuando se prueban las características de la aplicación para extraer URLs, se debe asegurar que la computadora pueda acceder a hosts remotos.

5.3.1 Redireccionamientos

Por defecto, HttpURLConnection seguirá las redirecciones HTTP. El servicio URL Fetch seguirá hasta un máximo de 5 redirecciones.

Para deshabilitar esta configuración, se llama al método `setInstanceFollowRedirects()` del objeto `HttpURLConnection`.

```
connection.setInstanceFollowRedirects(false);
```

El listado 5.1 muestra una petición utilizando la interfaz `java.net` que ocupa la API REST de twitter para buscar aquellos tweets que cumplan el criterio de búsqueda especificado [13]:

Listado 5.1 Uso del servicio URL Fetch y la API REST de Twitter

```
package mx.unam.store;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
public class URLFetchServlet extends HttpServlet {

      @Override
      protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {

            URL url = new URL("http://search.twitter.com/search.json?q=blue
                        %20angels&rpp=5&include_entities=true&result_type=mixed");
```

```
            BufferedReader reader = new BufferedReader(new
                                    InputStreamReader(url.openStream()));

            resp.setContentType("text/plain");

            String line;
            while ((line = reader.readLine()) != null) {
                  resp.getWriter().println(line);
            }
      }
}
```

5.4 Características de la API de bajo nivel

El servicio URL Fetch limita el tamaño de los datos para una petición de salida y una respuesta de entrada. Cuando se utiliza la API `java.net`, los datos mayores que el límite son truncados de manera silenciosa. La API de bajo nivel permite especificar si los datos se truncan de manera silenciosa, o si al exceder el límite se lance una excepción.

El listado 5.2 muestra un ejemplo utilizando la interfaz de bajo nivel:

Listado 5.2 Utilización de la interfaz de bajo nivel de URL Fetch

```
package mx.unam.store;

import java.io.IOException;
import java.net.URL;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.urlfetch.FetchOptions;
import com.google.appengine.api.urlfetch.HTTPMethod;
import com.google.appengine.api.urlfetch.HTTPRequest;
import com.google.appengine.api.urlfetch.HTTPResponse;
import com.google.appengine.api.urlfetch.URLFetchService;
import com.google.appengine.api.urlfetch.URLFetchServiceFactory;

@SuppressWarnings("serial")
public class URLFetchLowServlet extends HttpServlet {

      @Override
      protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {

            URL url = new URL("http://search.twitter.com/search.json?q=blue
                        %20angels&rpp=5&include_entities=true&result_type=mixed");

            FetchOptions options =
                                    FetchOptions.Builder
```

```
                                .doNotFollowRedirects().disallowTruncate();
            HTTPRequest request = new HTTPRequest(url, HTTPMethod.GET, options);

            URLFetchService service = URLFetchServiceFactory.getURLFetchService();
            HTTPResponse response = service.fetch(request);
            String content = new String(response.getContent());

            resp.setContentType("text/plain");
            resp.getWriter().println(content);
        }
}
```

Capítulo 6 Colas de tareas y tareas programadas

App Engine esta optimizado para aplicaciones web que responden de manera rápida al usuario, algunas aplicaciones no obstante requieren realizar trabajos que no encajan en el modelo de respuesta rápido. En lugar de realizar el trabajo mientras el usuario está esperando, es usualmente aceptable el recordar el trabajo que necesita realizarse y enviar una respuesta rápida al usuario.

También existen trabajos que requieren ser realizados de manera independiente de las acciones del usuario: el extraer y poner en caché datos frescos obtenidos de fuentes remotas, enviar mensajes de correo con el estado del sistema, realizar la actualización de datos que cumplan cierta condición de acuerdo a la fecha actual. Para este trabajo, resulta útil el configurar un gestor de tareas que se ejecute de manera periódica, y que realice las tareas independientemente del tráfico web de los usuarios.

App Engine provee estas facilidades por medio de dos servicios: *colas de tareas* y *tareas programadas*. Con las colas de tareas, un manejador de peticiones invocado por una petición del usuario puede encolar una unidad de trabajo (una *tarea*) y regresar una respuesta al usuario y salir. App Engine ejecuta las tareas encoladas a la tasa de rendimiento configurada. Si la tarea falla, App Engine reintenta la tarea, asegurándose que las tareas que fallen debido a condiciones transitorias (tal como contenciones en el almacén de datos) son realizadas de manera exitosa eventualmente.

Una aplicación también puede proveer tareas programadas de ser realizadas de manera regular cada cierto tiempo, éste tipo de tareas no son realizadas de manera continua hasta que concluyan con éxito, pero una tarea programada puede usar una cola de tareas para tomar ventaja de su comportamiento de repetición hasta completarse de manera exitosa.

Con la API de la Cola de Tareas, las aplicaciones pueden realizar trabajo iniciado por una solicitud del usuario, fuera de esa petición. Si la aplicación necesita el ejecutar alguna tarea en segundo plano, puede utilizar la API de la Cola de Tarea para organizar el trabajo en pequeñas unidades discretas, llamadas *tareas*. La aplicación inserta estas tareas en una o más *colas* basado en la configuración de las mismas.

App Engine provee dos diferentes configuraciones de colas:

- *Colas de inserción (push queue)*: procesan tareas basados en la tasa de procesamiento configurada en la definición de la cola. App Engine automáticamente escala las capacidades de procesamiento para igualar la configuración de la cola y el volumen de procesamiento, y también elimina las tareas después de procesarlas. Las colas de inserción son la configuración por defecto.
- *Colas de extracción (pull queue)*: permiten a un consumidor de tareas (ya sea la aplicación o código externo a la misma) el manejar tareas en un momento específico para procesarlas dentro de una ventana de tiempo específico. Las colas de extracción otorgan más control de en qué momento las tareas son procesadas, y también permite el integrar la aplicación con código que no pertenece a App Engine usando la API experimental de Cola de Tareas REST. Cuando se utilizan colas de extracción, la aplicación necesita el manejar el escalamiento de las instancias basado en el volumen de procesamiento y también requiere eliminar las tareas después de procesarlas.

6.1 Conceptos de colas

Las colas de tareas son una herramienta eficiente y poderosa para el procesamiento en segundo plano; ya que permiten a la aplicación el definir tareas, agregarlas a la cola y después usar la cola y procesarlas en grupos. El nombrado de las colas y la configuración de sus propiedades se realiza por medio del archivo de configuración *queue.xml*.

Las colas de inserción funcionan solo dentro del ambiente de App Engine. Estas colas son la mejor opción para aplicaciones cuyas tareas funcionan solamente con las herramientas y servicios de App Engine. Con las colas de inserción, se puede simplemente configurar una cola y agregar tareas a ella. App Engine maneja el resto. Las colas de inserción son fáciles de implementar, pero están restringidas a ser utilizadas dentro de App Engine.

Si se desea el usar un sistema diferente para consumir tareas, las colas de extracción son la mejor opción. En las colas de extracción, un consumidor de tareas (ya sea nuestra aplicación App Engine, un backend, o código utilizado fuera de App Engine) se encarga de un número específico de tareas obtenidas de una cola específica por un plazo determinado. Después de hacerse cargo de las tareas, el consumidor de la tarea es responsable de eliminarla. Si las tareas se consumen dentro de App Engine, se pueden usar llamadas del paquete `com.google.appengine.api.taskqueue`. Si se consumen tareas fuera de App Engine, se requiere el usar la API de Cola de Tareas REST. Las colas de extracción otorgan más poder y flexibilidad de en qué momento y donde son las tareas procesadas, pero requieren que el desarrollador maneje el escalamiento basado en el volumen de procesamiento. El consumidor de tareas también requiere el borrar tareas después de procesarlas.

6.1.1 La cola por defecto

Por conveniencia, App Engine provee una cola de inserción por defecto para todas las aplicaciones (no existe una cola de extracción por defecto). Si no es nombrada una cola para la tarea, App Engine inserta automáticamente en la cola por defecto. Se puede utilizar ésta cola inmediatamente sin ninguna configuración adicional.

La cola por defecto está configurada con una tasa de rendimiento de 5 invocaciones de tarea por segundo. Si se desean cambiar los valores predeterminados, se define simplemente una cola con el nombre de *default* en el archivo *queue.xml*. El código puede siempre insertar una nueva tarea en la cola por defecto, pero si deseamos el deshabilitar la ejecución de estas tareas, podemos realizarlo al oprimir el botón Pausar Cola en la pestaña Task Queues de la Consola de Administración.

6.1.2 Colas con nombres

Mientras la cola por defecto hace fácil el encolar tareas sin configuración. Se puede también el crear colas personalizadas al definirlas en *queue.xml*. Las colas personalizadas permiten manejar eficazmente el procesamiento de tareas al agrupar las tareas similares. Es posible el configurar la tasa de procesamiento (y otras características) basados específicamente en el tipo de tarea de cada cola.

6.1.3 Las colas de tareas en la Consola de Administración

Se puede el manejar las tareas para una aplicación utilizando la pestaña *Task Queue* de la Consola de Administración, la cual lista todas las colas que tiene la aplicación. Al dar click en el nombre de una cola muestra la página *Task Queue Details* donde se pueden ver todas las tareas programadas para ejecutarse en una cola y se puede borrar manualmente cada tarea o purgar todas las tareas de la cola. Esto es útil si una tarea en la cola de inserción no puede ser completada de manera exitosa y esta atorada esperando a ser realizada nuevamente. Se puede también el pausar y reanudar una cola en esta página.

Se pueden ver los detalles de las tareas individuales al dar click en el nombre de la tarea de la lista de tareas en la página *Task Queue Details*. Esta página permite depurar porqué una tarea no se ejecutó de manera exitosa. También se puede ver información acerca de la ejecución de la tarea previamente realizada así como el cuerpo de la tarea.

Nota: La cola por defecto aparece en la Consola sólo después de que la aplicación ha encolado una tarea por primera vez.

6.2 Conceptos de tareas

Una *tarea* es una petición a un manejador de peticiones. La tarea especifica una trayectoria URL de la aplicación a ser llamada, y también puede incluir datos que App Engine pasa al manejador de peticiones como parámetros HTTP. La URL de una aplicación en red llamada por un sistema para realizar una tarea es conocida como *gancho web*.

App Engine invoca un manejador de peticiones para una tarea en el mismo ambiente en el que se ejecuta un manejador de una petición realizada por un usuario. Esto significa que el manejador se ejecuta con la misma caja de arena y restricciones de recursos, incluyendo un tiempo límite de 30 segundos. Una tarea puede encolar otra tarea, permitiendo una unidad mayor de trabajo el ser iniciada por una única tarea y realizarla en lotes. Al tratar los manejadores de tareas como manejadores de peticiones, App Engine puede paralelizar y escalar las tareas usando la misma infraestructura que utiliza para escalar las peticiones de los usuarios.

El manejador de peticiones para la URL de la tarea debe realizar el trabajo que le fue conferido y regresar un código HTTP 200 de respuesta, indicando que el trabajo fue realizado de manera exitosa. Si el manejador regresa un código de respuesta diferente, App Engine asume que la tarea fallo completamente y reinserta la tarea a la cola para intentar realizarla de nuevo posteriormente. Una tarea que falla es reencolada con un retraso que se vuelve exponencialmente mayor entre más veces la tarea haya sido reintentada de tal forma que los problemas ocasionados por consumo de recursos y contención pueden ser aliviados automáticamente, hasta un retraso máximo de una hora.

El reintentar una tarea garantiza el éxito eventual de la tarea, asumiendo que es posible que la tarea termine de manera exitosa. Si una tarea es encolada de manera exitosa, permanece en la cola hasta que

se completa con éxito. Las tareas son reintentadas indefinidamente y nunca expiran.

6.2.1 Nombres de tareas

Adicionalmente a los contenidos de una tarea, se puede declarar el nombre de la tarea. Una vez que una tarea con nombre *N* es escrita, cualquier intento subsecuente de insertar una tarea con nombre *N* fallará.

La unicidad del nombre de la tarea es generalmente verdadero, sin embargo, en casos raros, múltiples llamadas a crear un tarea del mismo nombre podrían tener éxito. También es posible en casos excepcionales para una tarea el ejecutarse más de una vez (aun cuando fue creada solamente una vez).

Todos los proyectos, colas y nombres de tareas deben ser una combinación de uno o más dígitos, letras a-z, guiones bajos y/o diagonales, satisfaciendo por tanto la siguiente expresión regular.

```
[0-9a-zA-Z\-\_]+
```

Si una tarea de inserción es creada de manera satisfactoria, eventualmente será borrada (a lo más, siete días después de que la tarea se ejecutó de manera exitosa). Una vez eliminada, su nombre puede ser reusado.

Si una tarea de extracción es creada de manera exitosa, la aplicación debe eliminarla después de procesarla. El sistema podría tardar hasta siete días para reconocer que una tarea ha sido eliminada; durante este tiempo, el nombre de la tarea deja de estar disponible. Los intentos por crear otra tarea durante este tiempo con el mismo nombre resultarán en un error del tipo "*item exists*". El sistema no ofrece métodos para determinar si alguno de los nombres de tareas eliminadas se encuentran aún en el sistema. Para evitar estos inconvenientes, se recomienda dejar que App Engine genere los nombres de las tareas de manera automática.

6.2.2 Eliminando tareas

Existen tres formas de eliminar una tarea:

1. De manera individual utilizando la consola de administración
2. Eliminando todas las tareas para la cola completa, utilizando la consola de administración.
3. Programáticamente, usando la operación `purge()`, la cual elimina todas las tareas de la cola especificada.

Al sistema le toma cerca de 20 segundos el darse cuenta que la cola ha sido purgada. En las colas de inserción, las tareas continúan ejecutándose durante este tiempo.

Advertencia: El crear tareas cercanas al tiempo en el cual se realiza una llamada a `purge` ocasionará que las nuevas tareas también sean purgadas.

Programáticamente utilizando la opción `deleteTask()` para eliminar tareas individuales.

Nota: Podría tomar varias horas el reclamar la cuota usada para las colas purgadas de manera programática o a través de la consola de Administración.

6.3 Uso de colas de inserción en Java

Una aplicación en Java configura las colas utilizando el archivo de configuración *queue.xml*, en el directorio WEB-INF que se encuentra dentro del WAR. Cada aplicación tiene una cola de inserción llamada *default* con algunas características por defecto.

Para encolar una tarea, se obtiene una cola usando el `QueueFactory`, después de lo cual se llama a su método `add()`. Se puede obtener una cola cuyo nombre haya sido especificado en el archivo *queue.xml* usando la opción `getQueue()` del factory, o se puede obtener la cola por defecto usando `getDefaultQueue()`. Se puede llamar al método `add()` con una instancia de `TaskOptions` (producido por `TaskOptions.Builder()`), o llamarlo sin argumentos para crear una tarea con las opciones por defecto para la cola.

6.3.1 Ejecución de Tareas

App Engine ejecuta las tareas de inserción al llamar a la URL específica de la aplicación con los manejadores de eventos para estas tareas. Estas URL deben ser locales al directorio raíz de nuestra aplicación y especificadas relativas a la URL. Podemos agregar esta URL a la definición de tareas utilizando el parámetro url de la clase `TaskOptions`.

Nota: Si una tarea realiza operaciones sensibles (tales como modificar datos importantes), se puede asegurar la URL del trabajador para prevenir a usuarios externos maliciosos de llamarla de manera directa.

Una tarea debe terminar de ejecutar y mandar una respuesta HTTP con valores entres 200-299 dentro de los 10 minutos contados a partir de la petición original. Este plazo se encuentra separado de las peticiones del usuario, las cuales tienen un plazo de 60-segundos. Si la ejecución de la tarea se encuentra cerca del límite, App Engine lanza un `DeadlineExceededException` la cual se puede atrapar para salvar el trabajo o guardar en un log el progreso antes de que el plazo se exceda. Si la tarea falla al ejecutarse, App Engine reintenta realizarla basada en los criterios que se hayan configurado.

6.3.2 Encabezados de Peticiones de Tareas

Las peticiones del servicio de Cola de Tareas contienen los siguientes encabezados HTTP:

- `X-AppEngine-QueueName` el nombre de la cola (posiblemente default).
- `X-AppEngine-TaskName` el nombre de la tarea, o un ID único generado por el sistema si el

nombre no fue especificado.

- `X-AppEngine-TaskRetryCount` el número de veces que esta tarea ha sido intentada; para el primer intento este valor es 0.
- `X-AppEngine-FailFast` especifíca que las tareas que se ejecuten en un segundo plano fallen inmediatamente en lugar de esperar una cola pendiente.
- `X-AppEngine-TaskETA` el tiempo de ejecución de la tarea, especificado en microsegundos desde Enero 1 de 1970.

6.3.3 El orden de ejecución de las tareas

El orden en el cual las tareas son ejecutadas depende de varios factores:

- **La posición de la tarea en la cola.** App Engine intenta el procesar las tareas en un orden FIFO (first in, first out, primero en entrar primero en salir). En general las tareas son insertadas en el fondo de la cola y ejecutadas desde la cabeza de la cola.
- **Los atrasos de las tareas en la cola.** El sistema intenta el tener la latencia más baja posible para cualquier tarea a través de notificaciones especializadas optimizadas por el planificador. Por tanto, en el caso de que una cola tenga varias tareas con atraso, el planificador del sistema podría “saltar” nuevas tareas al final de la cola.
- **El valor de la propiedad etaMillis de las tareas.** Esta propiedad especifica el tiempo más corto en el cual se puede ejecutar una tarea. App Engine siempre espera hasta después del ETA especificado para procesar tareas de inserción.
- **El valor de la propiedad countdownMillis de las tareas.** Esta propiedad especifica el número mínimo de segundos a esperar antes de ejecutar la tarea. Countdown y eta son mutuamente exclusivos, si se especifica uno no se puede especificar el otro.

6.3.4 Reintento de tareas

Si el manejador de una petición de tarea de inserción regresa un código HTTP con estatus dentro del rango 200-209, App Engine considera que la tarea se ha realizado de manera satisfactoria. Si la tarea regresa un código de estatus fuera de este rango, App Engine reintenta la tarea hasta que se realiza. El sistema retrocede de manera gradual para evitar desbordar la aplicación con demasiadas peticiones, pero planifica los reintentos para las tareas que fallaron a que recurran a un máximo de uno por hora.

También se puede configurar un esquema personalizado para el reintento de tareas usando el elemento `retry-parameter` en el archivo *queue.xml*.

Cuando se implementa código para las tareas (tales como trabajadores URL en la aplicación), es importante el considerar si la tarea es idempotente. La API de la Cola de Tareas de App Engine está diseñada para invocar una tarea sólo una vez; sin embargo, es posible que en circunstancias excepcionales una tarea se ejecute más de una vez (tales como el caso improbable de una falla mayor en el sistema). Por lo tanto, nuestro código debe asegurarse que no existen efectos colaterales dañinos en el caso de ejecuciones múltiples.

6.3.5 Puntos finales de URL

Las colas de inserción hacen referencia a su implementación por medio de URL. Por ejemplo, una tarea que extrae y analiza un RSS podría usar una URL de trabajador llamada /app_worker/fetch_feed. Se puede especificar la URL de este trabajador o utilizar la que es dada por defecto. En general, se puede utilizar cualquier URL como trabajador de una tarea, siempre y cuando se encuentre dentro de la aplicación; todas las URL de trabajador deben especificarse de manera relativa.

Si no se especifica una URL de trabajador, la tarea usa una URL de trabajador por defecto nombrada después de la cola:

`/_ah/queue/`*queue_name*

La URL por defecto para la cola es usada si y sólo si una tarea no tiene una URL de trabajador propia. Si una tarea tiene su propia URL de trabajador, entonces es solamente invocada para la URL del trabajador y nunca en otra. Una vez insertada en la cola, su punto final de URL no puede ser cambiado.

Advertencia: Si una tarea no tiene una URL de trabajador, entonces la tarea es invocada con la URL por defecto de la cola, aún si actualmente no existen manejadores definidos para la URL por defecto de la cola. Esto resulta en una respuesta HTTP 404 la cual estará disponible, junto con la URL exacta que fue intentada, en los logs de la aplicación. Debido a el estatus de falla, el sistema salva la tarea y la reintenta hasta que es eventualmente exitosa. Se pueden limpiar (o purgar) las tareas que no puedan completarse de manera satisfactoria usando la consola de Administración.

Para aterrizar todo lo que se ha mencionado de las colas de inserción a continuación se muestra una configuración básica de una cola de inserción cuyo manejador sólo imprime un mensaje en el log.

Listado 6.1 Configuración del archivoweb.xml para la cola de inserción

```
<servlet>
      <servlet-name>pushQueue</servlet-name>
      <servlet-class>mx.unam.store.PushQueue</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>pushQueue</servlet-name>
      <url-pattern>/pushQueue</url-pattern>
</servlet-mapping>
<servlet>
      <servlet-name>pushQueueHandler</servlet-name>
      <servlet-class>mx.unam.store.PushQueueHandler</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>pushQueueHandler</servlet-name>
      <url-pattern>/pushHanlder</url-pattern>
</servlet-mapping>
```

El servlet `pushQueue` se encarga de agregar la tarea a una cola de inserción, mientras que el servlet `pushQueueHandler` es la clase encargada de manejar la tarea correspondiente.

Listado 6.2 Utilización de una cola de inserción

```
package mx.unam.store;

import java.io.IOException;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.taskqueue.Queue;
import com.google.appengine.api.taskqueue.QueueFactory;
import com.google.appengine.api.taskqueue.TaskOptions;

@SuppressWarnings("serial")
public class PushQueue extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		// obtener la cola definida en el archivo queue.xml
		Queue queue = QueueFactory.getQueue("fasttest");
		TaskOptions taskOptions = TaskOptions.Builder.withUrl("/pushHanlder");

		queue.add(taskOptions);

		log.info("Push Queue");
	}

	private static Logger log = Logger.getLogger(PushQueue.class.getName());
}
```

La clase PushQueue utiliza la cola de inserción con nombre “fasttest”, esta se declara en el archivo de configuración *queue.xml*, el cual se encuentra en la carpeta WEB-INF, a continuación se muestra su contenido

Listado 6.3 Archivos de configuración queue.xml

```
<queue-entries>
	<queue>
		<name>fasttest</name>
		<rate>1/m</rate>
		<bucket-size>10</bucket-size>
	</queue>
</queue-entries>
```

Es importante el notar que este archivo xml no contiene un DTD.

A continuación se muestra la clase PushQueueHandler, la cual se encarga de manejar la tarea ingresada en la cola de inserción fastest.

Listado 6.4 Manejador de las tareas presentes en la cola de inserción

```
package mx.unam.store;

import java.io.IOException;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
public class PushQueueHandler extends HttpServlet {

      @Override
      protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {
            log.info("Push Queue Handler");
      }

      private static Logger log =
Logger.getLogger(PushQueueHandler.class.getName());
}
```

6.4 Uso de Colas de extracción

Las colas de extracción permiten diseñar sistemas propios para consumir tareas de App Engine. El consumidor de tareas puede ser parte de la aplicación App Engine (tal como un proceso en segundo plano o un sistema fuera de App Engine (utilizando la API REST de Cola de Tareas). El consumidor de tareas se encarga de un número específico de tareas por un periodo específico de tiempo, después las procesa y borra antes de que el tiempo de arrendamiento termine.

El uso de colas de extracción requiere que la aplicación maneje algunas características que son automáticas en las colas de inserción:

- La aplicación requiere escalar el número de trabajadores basado en el volumen de procesamiento. Si la aplicación no maneja el escalamiento, existe el riesgo de malgastar recursos de cómputo si no existen tareas que procesar; también está el riesgo de latencia si se tienen demasiadas tareas que procesar.
- La aplicación también requiere que se borren de manera explícita las tareas después de procesarlas. En las colas de inserción, App Engine elimina la tarea por nosotros. Si la aplicación no elimina tareas en la cola de extracción después de procesarlas, otro trabajador podría reprocesar la tarea. Esto gasta recursos de cómputo y tiene riesgos de errores si las tareas no son idempotentes.

Las colas de extracción requieren una configuración específica en el archivo *queue.xml*.

6.4.1 Visión General de la Cola de Extracción

Las colas de extracción permiten a un consumidor de tareas el procesar tareas fuera del sistema de procesamiento por defecto de App Engine. Si un consumidor de tarea es parte de la aplicación App Engine, se pueden manipular las tareas utilizando llamadas simples a la API del paquete `com.google.appengine.api.taskqueue`. Los consumidores fuera de App Engine pueden extraer las tareas utilizando la API REST de la Cola de Tareas.

Este proceso trabaja del siguiente modo:

- El consumidor adquiere tareas, ya sea vía la API de Cola de Tareas (si el consumidor es interno a App Engine) o utilizando la API REST de Cola de Tareas (si el consumidor es externo a App Engine).
- App Engine envía los datos de la tarea al consumidor.
- El consumidor procesa las tareas. Si la tarea falla al ejecutares antes de que el tiempo de arrendamiento expire, el consumidor puede arrendar nuevamente. Esto cuenta como un reintento, y se puede configurar el número máximo de intentos antes de que el sistema elimine la tarea.
- Una vez que la tarea se ejecuta exitosamente, el consumidor de tarea debe borrarla.
- El consumidor de tareas es responsable de escalar las instancias basado en el volumen de procesamiento.

6.4.2 Agregando Tareas a la Cola de Extracción

Para agregar tareas a la cola de extracción, simplemente obtenemos la cola usando el nombre definido en el archivo *queue.xml*, y usando `TaskOptions.Method.PULL`.

6.4.3 Arrendando Tareas

Una vez que se han agregado tareas a la cola de extracción, se utiliza `leaseTasks()` para hacerse cargo de una o más tareas. Podría haber un ligero retraso antes de que las tareas recientemente agregadas usando `add()` estén disponibles por medio de `leaseTasks()`. Cuando se solicita un arrendamiento, se especifica el número de tareas a arrendar (hasta un máximo de 100 tareas) y la duración del arrendamiento en segundos (hasta una semana como máximo). La duración del arrendamiento requiere ser lo suficiente de modo que puede asegurarse que la tarea más lenta tendrá tiempo de terminar antes de que el periodo de arrendamiento expire. Se puede extender el arrendamiento utilizando `modifyTaskLease()`.

Arrendar una tarea hace que deje de estar disponible para procesamiento por otro trabajador, y permanece inasequible hasta que el periodo de arrendamiento expira. Si se arrenda una tarea individual, la API selecciona la tarea de la parte frontal de la cola. Si tal tarea no está disponible una lista vacía es regresada.

Note: `leaseTasks()` opera solamente en colas de extracción. Si se intenta arrendar una tarea agregada

a la cola de inserción, App Engine lanza una excepción. Se puede cambiar una cola de inserción a una cola de extracción al cambiar su definición en el archivo *queue.xml*.

6.4.4 Eliminación de tareas

En general, una vez que un trabajador completa una tarea, requiere eliminar la tarea de la cola. Si se observan tareas restantes en una cola después de que el trabajador termina de procesarlas, es muy probable que el trabajador haya fallado; en este caso, la tarea necesita ser procesada por otro trabajador.

Se puede eliminar una tarea individual de la lista usando `deleteTask()`. Se debe conocer el nombre de la tarea para poder eliminarla. Si se están eliminando tareas de una cola de extracción, se pueden encontrar los nombres de las tareas en el objeto `Task` regresado por `leaseTasks()`.

A continuación se muestra una configuración básica de una cola de extracción, primero se mostrará la declaración del archivo *queue.xml*

Listado 6.5 Archivo de configuración queue.xml para una cola de extracción

```
<queue-entries>

      <!-- cola de extraccion -->
      <queue>
            <name>pullQueueTest</name>
            <mode>pull</mode>
      </queue>

</queue-entries>
```

La clase encargada de insertar trabajos en la cola de extracción es `PullQueue`, la cual extiende de `HttpServlet`, ambos métodos `doGet` y `doPost` realizan el mismo trabajo, su definición se muestra a continuación

Listado 6.6 Utilización de una cola de extracción

```
package mx.unam.store;

import java.io.IOException;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.taskqueue.Queue;
import com.google.appengine.api.taskqueue.QueueFactory;
import com.google.appengine.api.taskqueue.TaskOptions;

@SuppressWarnings("serial")
public class PullQueue extends HttpServlet {
```

```
	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		// obtener la cola definida en el archivo queue.xml
		Queue queue = QueueFactory.getQueue("pullQueueTest");
		TaskOptions taskOptions = TaskOptions.Builder
				.withMethod(TaskOptions.Method.PULL)
				.payload("Ejemplo pull queue");
		queue.add(taskOptions);
	}

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		log.info("Pull Queue");

		doPost(req, resp);
	}


	private static Logger log = Logger.getLogger(PullQueue.class.getName());
}
```

En el ejemplo la clase encargada de procesar las tareas de la cola de extracción es `PullQueueHandler`, la cual muestra en el log el nombre de la tarea, y la carga útil (payload) que haya definido después de lo cual elimina la tarea de la cola.

Listado 6.7 Manejador de tareas de la cola de extracción

```
package mx.unam.store;

import java.io.IOException;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.taskqueue.Queue;
import com.google.appengine.api.taskqueue.QueueFactory;
import com.google.appengine.api.taskqueue.TaskHandle;

@SuppressWarnings("serial")
public class PullQueueHandler extends HttpServlet {

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		// manejar las tareas de la cola de extraccion
		Queue queue = QueueFactory.getQueue("pullQueueTest");
```

```
            List<TaskHandle> tasks = queue.leaseTasks(10, TimeUnit.SECONDS, 10);

            log.info("Pull Queue Handler, post method");

            for (TaskHandle t : tasks) {
                  String payload = new String(t.getPayload());
                  String taskName = t.getName();

                  log.info("Task name: " + taskName);
                  log.info("Payload: " + payload);

                  // delete task
                  queue.deleteTask(taskName);
            }
      }

      @Override
      protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {
            log.info("Pull Queue Handler");


            doPost(req, resp);
      }

      private static Logger log =
Logger.getLogger(PullQueueHandler.class.getName());

}
```

Por último se muestra la definición en el archivo web.xml para las clases antes presentadas

Listado 6.8 Configuración del archivoweb.xml para la cola de inserción

```
<!-- Pull Queue -->
      <servlet>
            <servlet-name>pullQueue</servlet-name>
            <servlet-class>mx.unam.store.PullQueue</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>pullQueue</servlet-name>
            <url-pattern>/pullQueue</url-pattern>
      </servlet-mapping>

      <!-- Pull Queue Handler -->
      <servlet>
            <servlet-name>pullQueueHandler</servlet-name>
            <servlet-class>mx.unam.store.PullQueueHandler</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>pullQueueHandler</servlet-name>
            <url-pattern>/pullQueueHandler</url-pattern>
      </servlet-mapping>
```

Capítulo 7. Creando una aplicación

En esta sección se mostrará el uso del SDK de Google App Engine, el cual facilita el desarrollo de aplicaciones web, elaborando una aplicación que muestre un mensaje de bienvenida al usuario.

El SDK de App Engine se proporciona como un plugin para el entorno de desarrollo Eclipse, nuestro ejemplo se ejecuta sobre una distribución Debian Linux de 64 bits, con el SDK de Java versión 7.2 y Eclipse índigo JEE.

1. Una vez que comienza eclipse, se elige la opción **Help > Install New Software** ..., en la ventana de diálogo que aparece, ingrese la siguiente URL del sitio de actualización en el campo de texto **Work with**: `http://dl.google.com/eclipse/plugin/3.7`

Y oprima la tecla enter.

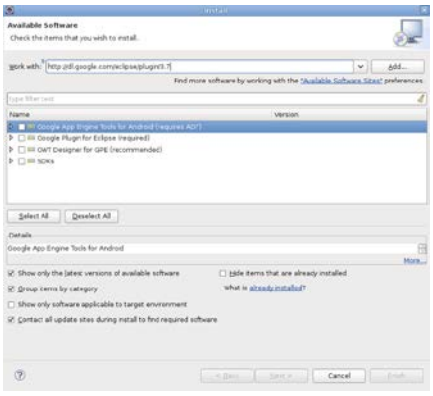

Figura 7.1 Software disponible

2. Deberá ver una caja de texto al centro la cual contiene las opciones Plugin y SDKs. Seleccione el checkbox que se encuentra junto a ambas opciones. Esto instalará el plugin, el SKD Java de Google App Engine y el SDK Google Web Toolkit. Oprima la opción Next.

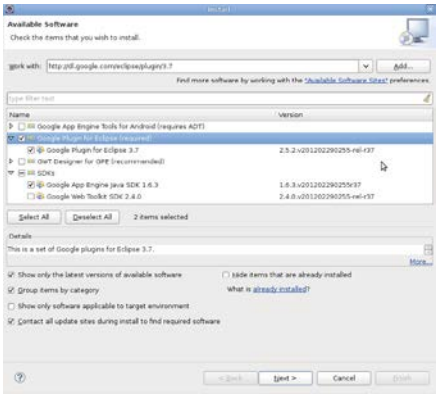

Figura 7.2 Software disponible

3. Revise las características que esta a punto de instalar. De click en Next.

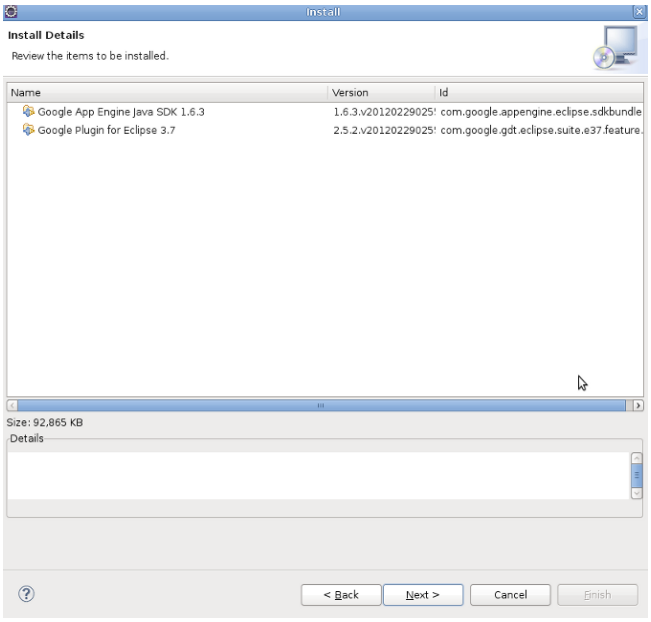

Figura 7.3 Detalles de instalación

4. Lea el acuerdo de licencia y si esta de acuerdo, seleccione la opción “I accept the terms of the license agreements”. De click en Finish.

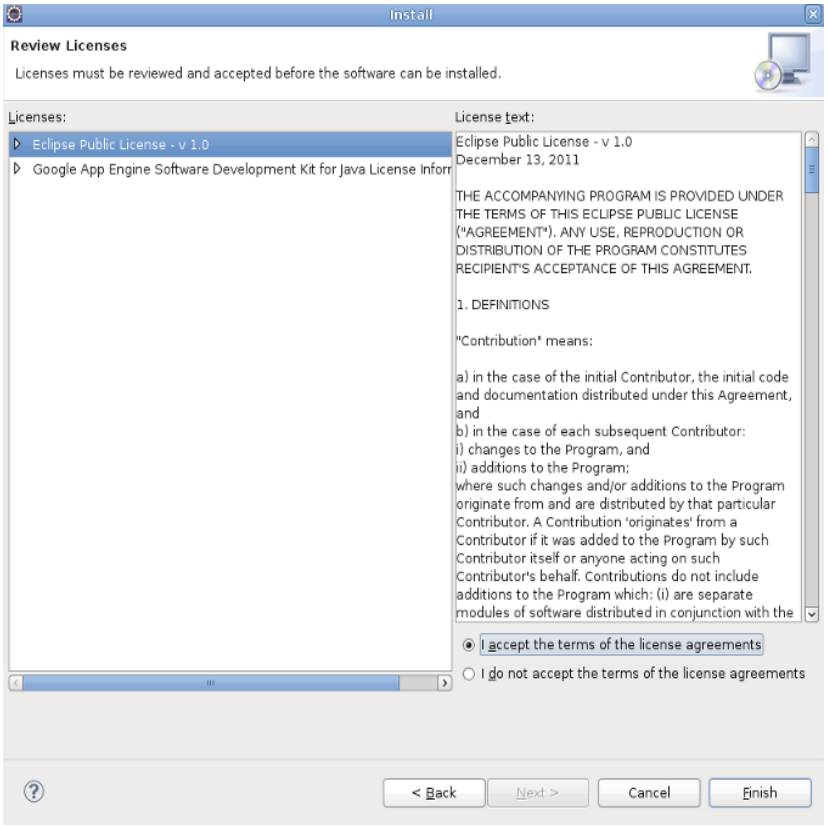

Figura 7.4 Acuerdo de licencia

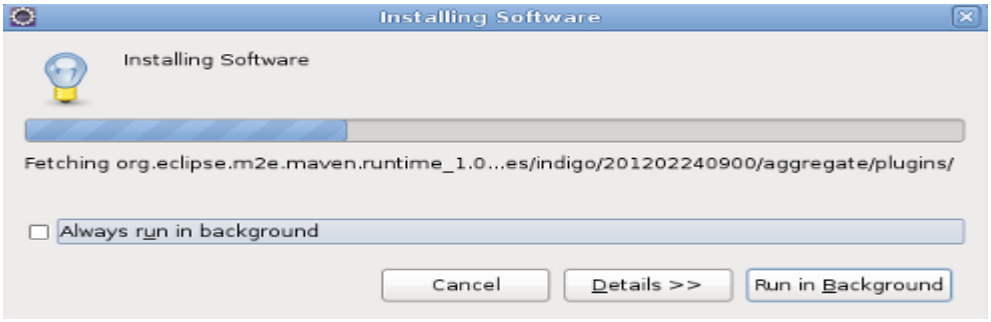

Figura 7.5 Progreso instalación

Figura 7.6 Contenido sin firmar

5. Al final de la instalación se le preguntará si desea reiniciar Eclipse. De click en **Restart Now**.

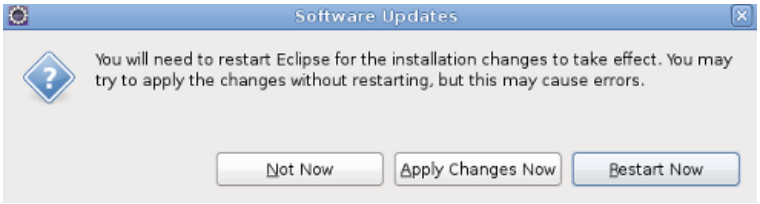

Figura 7.7 Reinicio de Eclipse

Ahora se encuentra en condiciones para continuar con el desarrollo de la aplicación. Es importante mencionar que Google App Engine proporciona un entorno Java version 6 en tiempo de ejecución, al momento de realizar este trabajo el SDK Java proporcionado por Oracle se encuentra en la version 7, la pregunta que surge es ¿existirán problemas si ejecuto la aplicación de manera local utilizando Java 7?, veremos que ocurre.

1. El SDK de Google es capaz de crear el esqueleto de una aplicación básica que muestra un mensaje de bienvenida al usuario, para realizarlo elija **File** > **New** > **Web Application Project** del menú principal.

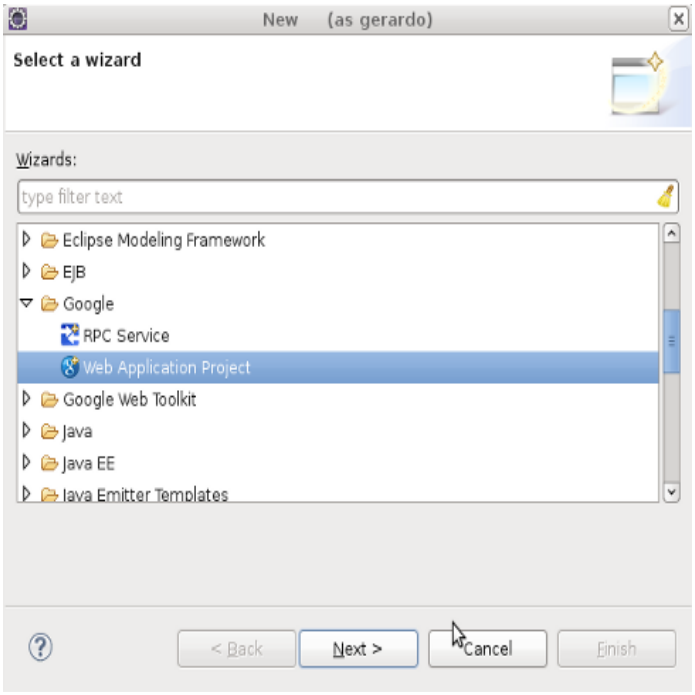

Figura 7.8 Nuevo proyecto de aplicación web

2. Escriba el nombre del proyecto y seleccione la opción **Use Google App Engine**

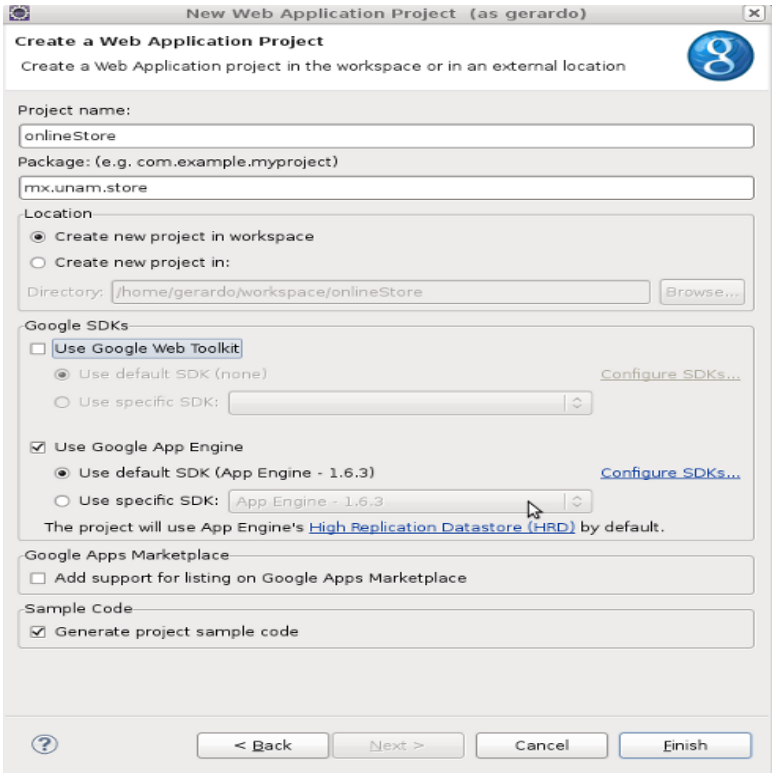

Figura 7.9 Datos del proyecto

3. Para ejecutar la aplicación de click derecho sobre la carpeta raíz del proyecto, y elija **Debug As > Web Application**, al realizarlo el servidor de desarrollo le presentará el siguiente mensaje.

```
java.lang.RuntimeException: Unable to restore the previous TimeZone
        at com.google.appengine.tools.development.DevAppServerImpl.restoreLocalTimeZone(DevAppServerImpl.java:228)
        at com.google.appengine.tools.development.DevAppServerImpl.start(DevAppServerImpl.java:164)
        at com.google.appengine.tools.development.DevAppServerMain$StartAction.apply(DevAppServerMain.java:173)
        at com.google.appengine.tools.util.Parser$ParseResult.applyArgs(Parser.java:48)
        at com.google.appengine.tools.development.DevAppServerMain.<init>(DevAppServerMain.java:120)
        at com.google.appengine.tools.development.DevAppServerMain.main(DevAppServerMain.java:96)
Caused by: java.lang.NoSuchFieldException: defaultZoneTL
        at java.lang.Class.getDeclaredField(Class.java:1899)
        at com.google.appengine.tools.development.DevAppServerImpl.restoreLocalTimeZone(DevAppServerImpl.java:222)
        ... 5 more
```

Figura 7.10 Error en la configuración de zona horaria

4. Para solucionarlo se requiere el pasar un argumento adicional a la máquina virtual, para ello seleccione **Debug > Debug Configurations > Arguments** y agregue la siguiente opción a la máquina virtual de Java *-Dappengine.user.timezone=UTC*

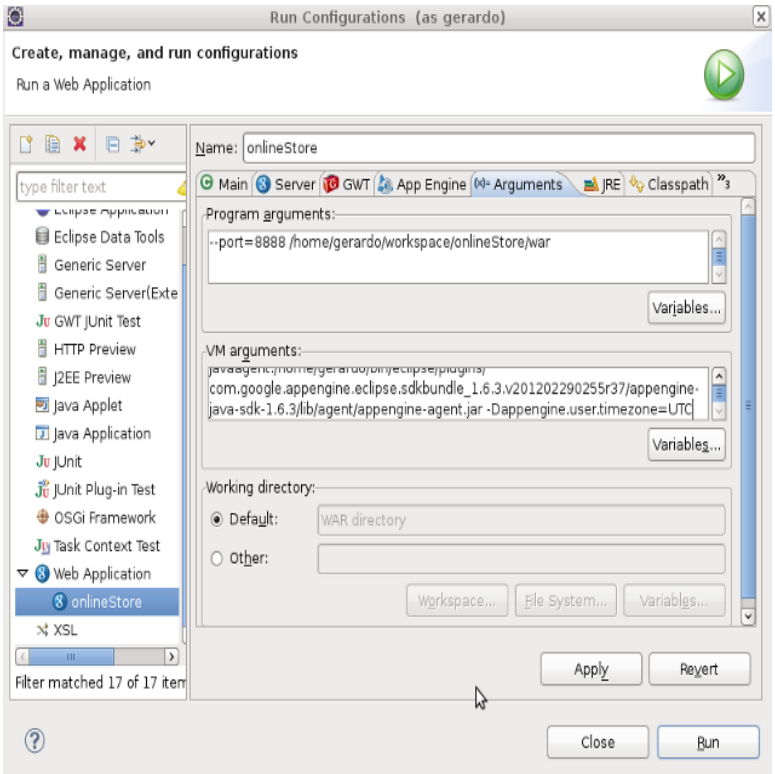

Figura 7.11 Configuración de la zona horaria

5. Ahora ejecute nuevamente la aplicación, esta vez se ejecutará sin problemas. Para visualizarla abra su navegador web e ingrese la url http://localhost:8888/proyectosimple

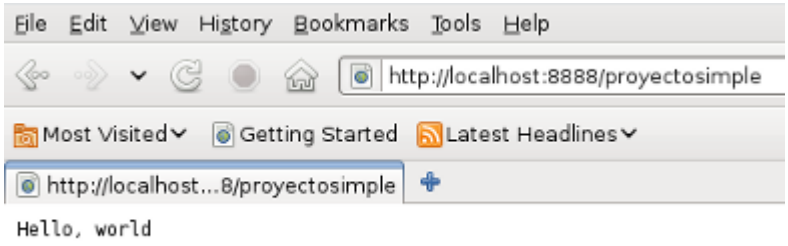

Figura 7.12 Ejecución del proyecto inicial

Con esto se termina la primera aplicación con Google App Engine.

Capítulo 8 Caso de estudio. Integración del framework Struts 2.

Apache Struts 2 es un framework para la creación de aplicaciones Java web empresariales, el cual implementa el patron de diseño Modelo-Vista-Controlador. El framework utiliza la especificación Java Servlet creando una abstracción que permite un manejo más limpio y dinámico de las aplicaciones web.

No se ahondará en las características y descripción de Struts 2 ya que se encuentra fuera del alcance del trabajo, sin embargo, se mostrarán los pasos necesarios para su integración con Google App Engine [10] [11].

Como primer paso, se creará un nuevo proyecto, para ello en eclipse elija **File > New project > Web Project**

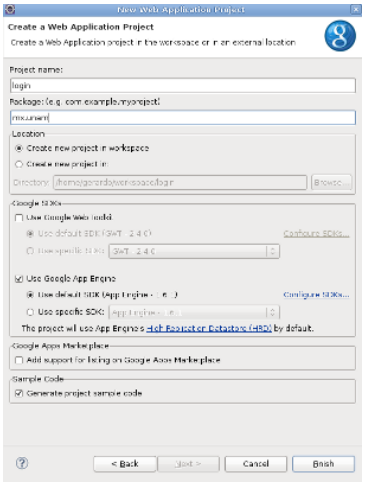

Figura 8.1 Nuevo proyecto web

El plugin de Google para Eclipse creará un proyecto con la estructura mínima de una aplicación.

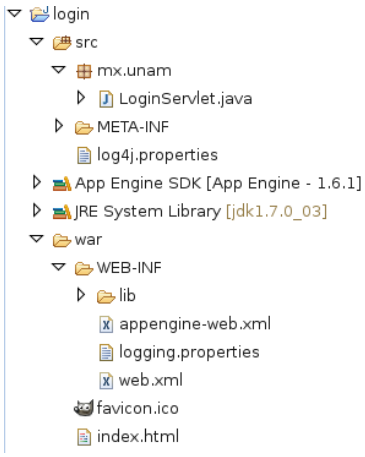

Figura 8.2 Estructura mínima proyecto web

El siguiente paso será el agregar las librerías mínimas que requiere un proyecto que utilice Struts 2, la versión que se ocupará será Struts 2 – 2.3.1.2 y los siguientes jar

- struts2-core.jar
- xwork.jar
- ognl.jar
- freemarker.jar
- commons-logging.jar
- commons-fileupload.jar
- commons-lang.jar (no indicado en la página)
- commons-io.jar
- javassist.jar (no indicado en la página)

Nota: El nombre de los jar omite la versión de cada uno de ellos.

Una vez que tenga las librerías requeridas, agreguelas a la carpeta *war/WEB-INF/lib* del proyecto

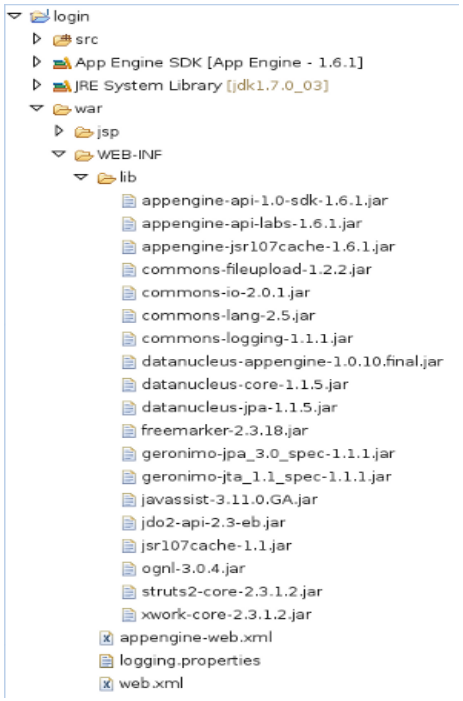

Figura 8.3 Librerías mínimas proyecto con Struts 2

Eclipse necesita conocer que estas librerías serán parte del classpath del proyecto para ello, seleccione la carpeta raíz, seleccione **Properties > Java Build Path > Libraries** y elija la opción **Add Jars**, eligiendo las librerías que agrego previamente al proyecto.

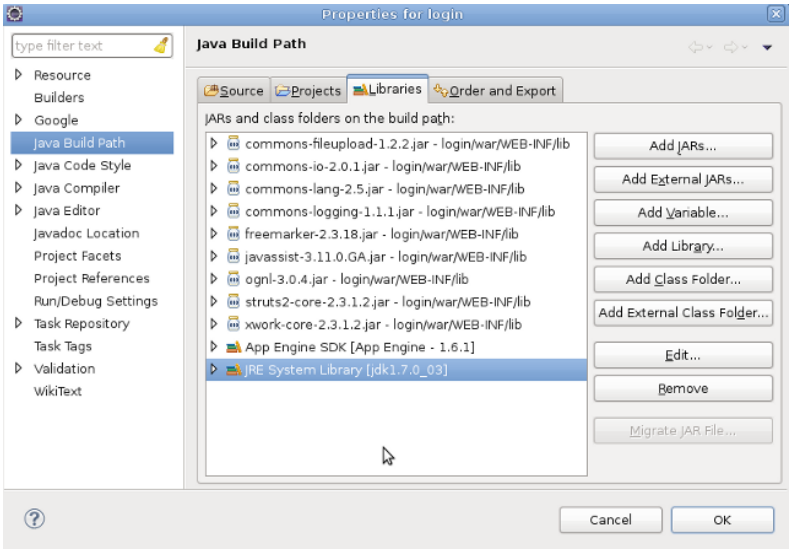

Figura 8.4 Agregar las liberías al classpath del proyecto

En struts 2 cualquier clase puede encargarse de manejar una petición siempre y cuando exponga funciones que no reciban parámetros y regrese una cadena como resultado, no se requiere el extender o implementar una clase en particular. El listado 8.1 muestra una clase sencilla encargada de manejar los datos de login del usuario.

Listado 8.1 Clase encargada de manejar los datos de login del usuario

```
package mx.unam.action;

import java.util.logging.Logger;

public class ActionLogin {

	public String validarUsuario() {
		log.info("!validarUsuario");

		return "success";
	}

	public String getUsername() {
		return username;
	}

	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
```

```
            this.password = password;
        }


        private static Logger log = Logger.getLogger
                    (ActionLogin.class.getCanonicalName());

        private String username;
        private String password;
}
```

Ahora genere la página de login y la página de bienvenida que se mostrará una vez que el usuario haya ingresado su nombre y contraseña, para ello cree la carpeta *jsp* dentro de la carpeta war, y cree el archivo login.jsp con los contenidos del listado 8.2

Listado 8.2 Formulario utilizado por el usuario para ingresar los datos de login

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head></head>
<body>

<s:form action="ValidateLogin">
  <s:textfield name="username" label="Usuario"/>
  <s:password name="password" label="Contraseña"/>
  <s:submit/>
</s:form>

</body>
</html>
```

y la página de bienvenida, la cual solo desplegará un mensaje al usuario

Listado 8.3 Página de bienvenida welcome.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head></head>
<body>

<h4>Hello <s:property value="username"/></h4>

</body>
</html>
```

Sólo resta indicarle a Google App Engine de la existencia de struts 2, para ello se requiere:

- Configurar el archivo *web.xml* para declarar el filtro de struts 2
- Declarar los Action (clases encargadas de manejar las peticiones) en el archivo *struts.xml*

El archivo *web.xml* con las modificaciones pertinentes se muestra en el listado 8.4

Listado 8.4 Archivo de configuración web.xml

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
 <display-name>Struts 2 - App Engine integration</display-name>

 <filter>
       <filter-name>struts2</filter-name>
       <filter-class>
        org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
       </filter-class>
 </filter>

   <filter-mapping>
       <filter-name>struts2</filter-name>
       <url-pattern>/*</url-pattern>
   </filter-mapping>


 <welcome-file-list>
       <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
```

Y el archivo de *struts.xml*

Listado 8.5 Archivo de configuración struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

    <constant name="struts.devMode" value="true" />
    <package name="/" namespace="/Registration" extends="struts-default">

        <action name="ValidateLogin"
                method="validarUsuario"
                class="mx.unam.action.ActionLogin">
         <result name="success">jsp/welcome.jsp</result>
        </action>

        <action name="Login">
         <result>jsp/login.jsp</result>
        </action>

    </package>

</struts>
```

Si se ejecuta la aplicación por medio de **Run > Debug As > Web application** en un entorno java 1.7 se obtendrá el siguiente error en la consola

```
java.lang.RuntimeException: Unable to restore the previous TimeZone
        at com.google.appengine.tools.development.DevAppServerImpl.restoreLocalTimeZone(DevAppServerImpl.java:228)
        at com.google.appengine.tools.development.DevAppServerImpl.start(DevAppServerImpl.java:164)
        at com.google.appengine.tools.development.DevAppServerMain$StartAction.apply(DevAppServerMain.java:164)
        at com.google.appengine.tools.util.Parser$ParseResult.applyArgs(Parser.java:48)
        at com.google.appengine.tools.development.DevAppServerMain.<init>(DevAppServerMain.java:113)
        at com.google.appengine.tools.development.DevAppServerMain.main(DevAppServerMain.java:89)
Caused by: java.lang.NoSuchFieldException: defaultZoneTL
        at java.lang.Class.getDeclaredField(Class.java:1899)
        at com.google.appengine.tools.development.DevAppServerImpl.restoreLocalTimeZone(DevAppServerImpl.java:222)
        ... 5 more
```

Figura 8.5 Excepción debido a la zona horaria

Para solucionarlo se selecciona *Run* > *Run configurations* > *Arguments* y se agrega la opción **`-Dappengine.user.timezone=UTC`** a la máquina virtual de java, se guardan las modificaciones y se ejecuta nuevamente la aplicación [12]. Esta vez no mostrará ningún error en la consola, se abre un navegador y se ingresa la siguiente URL: *http://localhost:8888/Login.action*

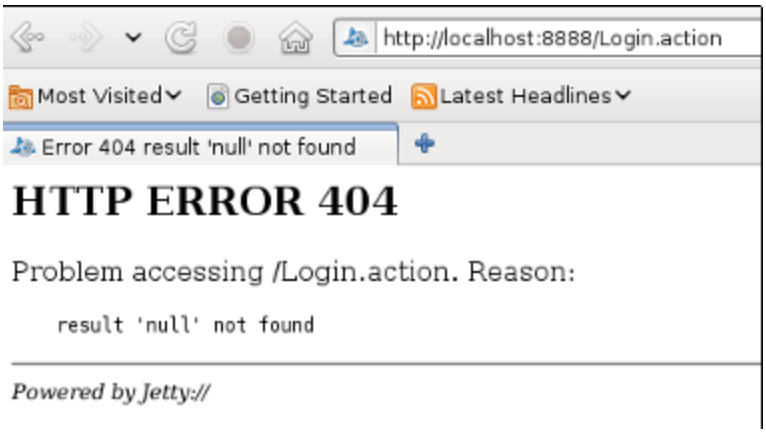

Figura 8.6 Resultado nulo debido a restricciones de seguridad de OGNL

El navegador indica que no existe el resultado nulo, éste es un punto muerto ya que la configuración de struts 2 es correcta y la consola del servidor de desarrollo no muestra ningún mensaje. El error que tenemos se debe al manejador de seguridad de OGNL, el cual se debe establecer en `null` para que struts 2 sea capaz de encontrar el resultado de la petición, para ello se crea el `Servlet ContextListener`, con los contenidos del listado 8.6.

Listado 8.6 Listener que establece en `null` el manejador de seguridad de OGNL.

```
package mx.unam.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import ognl.OgnlRuntime;

public class OgnlListener implements ServletContextListener {

	@Override
	public void contextDestroyed(ServletContextEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void contextInitialized(ServletContextEvent arg0) {
		OgnlRuntime.setSecurityManager(null);

	}

}
```

Se debe declarar el listener en el archivo web.xml, para ello agregue las líneas del listado 8.7

Listado 8.7 Declaración del listener en el archivo de configuración web.xml

```
<listener>
    <listener-class>mx.unam.listener.OgnlListener</listener-class>
</listener>
```

Si se ejecuta de nuevo la aplicación, se obtendrá el siguiente resultado

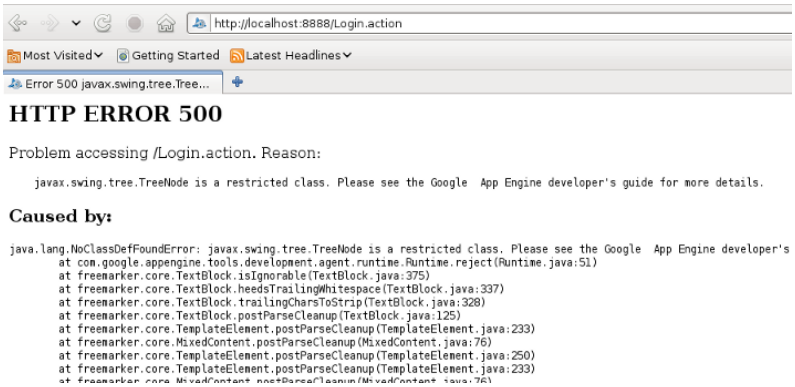

Figura 8.7 Excepción debido a freemarker

Para corregirlo se debe crear el paquete *freemarker.core* y crear la clase `TextBlock`, debido a la longitud del código, su contenido se presenta en el apéndice A, con la clase recién agregada se ejecuta nuevamente el proyecto

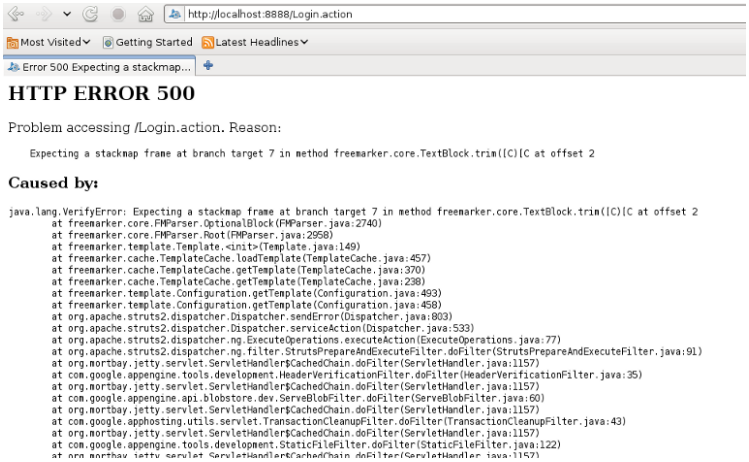

Figura 8.8 Error en la versión de la máquina virtual de Java

Esta vez se tiene un error más críptico con un mensaje similar a "*Expecting a stackmap frame at branch target 7 ...*", sin embargo su solución es mucho más sencilla, para ello se modifica la versión del entorno Java que se esta utilizando, cambiando de la versión 1.7 a la 1.6, en eclipse se da click derecho en la carpeta raíz y se elije la opción *Build path* > *Configure build path* > *Libraries*, se edita la opción *JRE System Library [JavaSE-1.7]*, en *Execution environment* se selecciona la version 1.6 y se da click en *Finish*. Al ejecutar la aplicación se obtiene la siguiente pantalla de login.

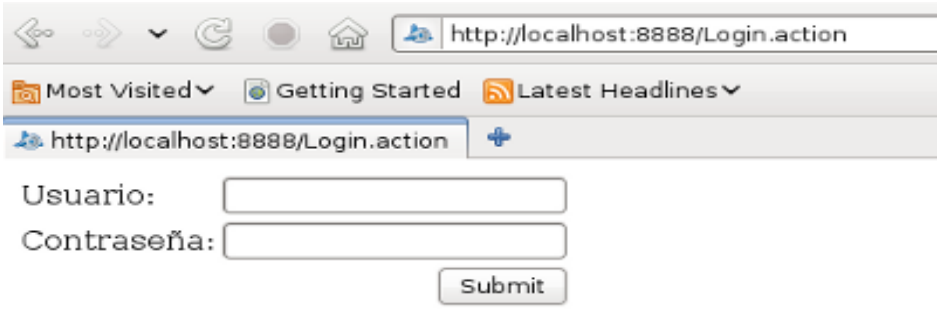

Figura 8.9 Formulario de entrada

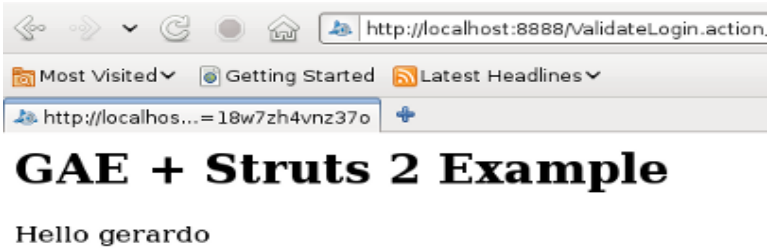

Figura 8.10 Mensaje de bienvenida

Con lo cual termina la configuración de un entorno que utiliza Struts 2.

Capítulo 9. Instalación de la aplicación

El subir la aplicación a App Engine es una tarea sencilla, sólo se requiere el dar click en un botón, después de lo cual se ingresa la cuenta de email y la contraseña.

Lo primero que se tiene que realizar es registrar un identificador único para la aplicación, después de lo cual se podrá subirla ya sea mediante el plugin de Eclipse o ejecutando una utilidad en línea de comandos del SDK.

Registrando la aplicación

Las aplicaciones web App Engine se crean y manejan desde la Consola de Administración de App Engine, a la cual se puede acceder desde la siguiente URL:

https://appengine.google.com

Ingrese utilizando alguna cuenta de correo de Gmail. La primera vez que se ingresa se deberá registrar un número de celular en el cual Google App Engine mandará un mensaje de texto con un código de activación, una vez que se tiene el código correspondiente Google App Engine presentará una pantalla en la cual se registrará el ID de la aplicación así como algunas características adicionales tales como el tipo de autentificación que se desea utilizar.

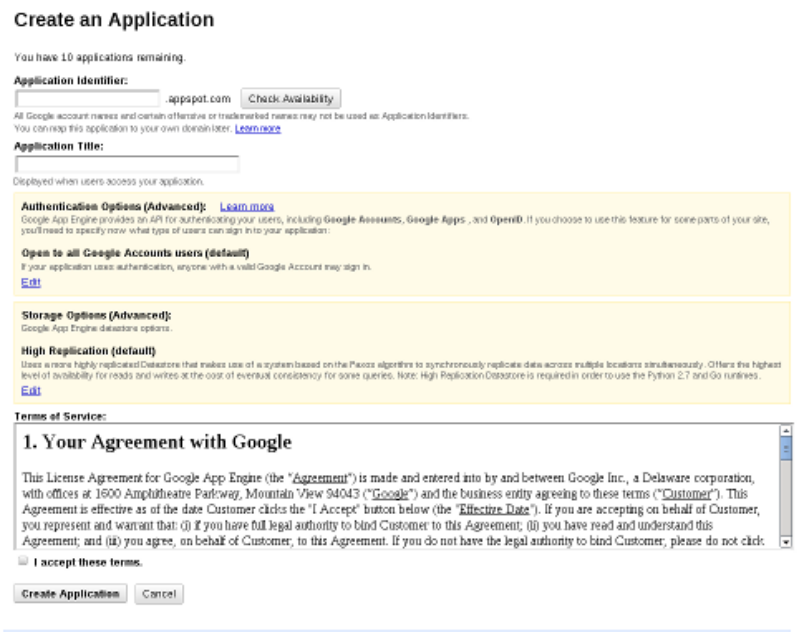

Figura 9.1 Registro de una aplicación en Google App Engine

Una vez que se tenga el ID, se deberá actualizar el archivo *appengine-web.xml*, donde se copiará el ID y se indicará el número de versión.

Una vez que se haya actualizado el archivo correspondiente se usará el SDK de Google App Engine para Eclipse con el objeto de subir la aplicación, para ello se dará click derecho sobre el proyecto y se elegirá *Google >Deploy to App Engine*

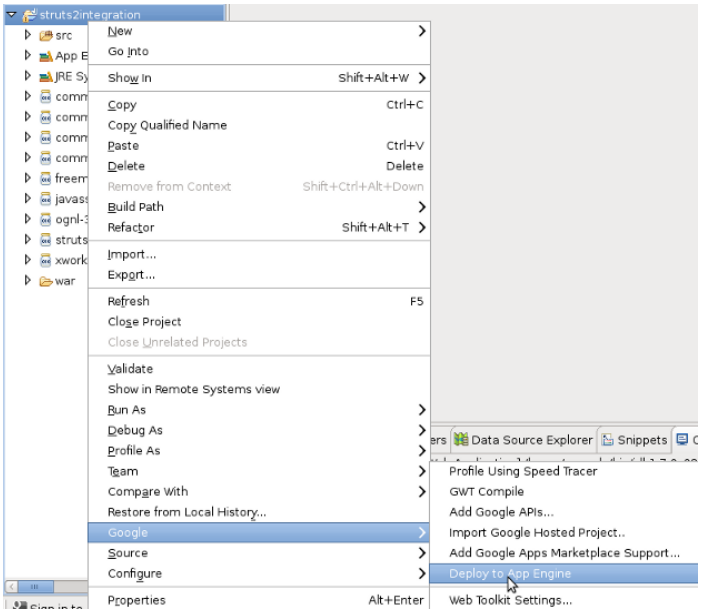

Figura 9.2 Instalación de la aplicación en los servidores de Google App Engine

El SDK presentará una pantalla donde se ingresará la cuenta de gmail que se dio de alta en *https//appengine.google.com*, en esta pantalla se ingresará la cuenta y contraseña:

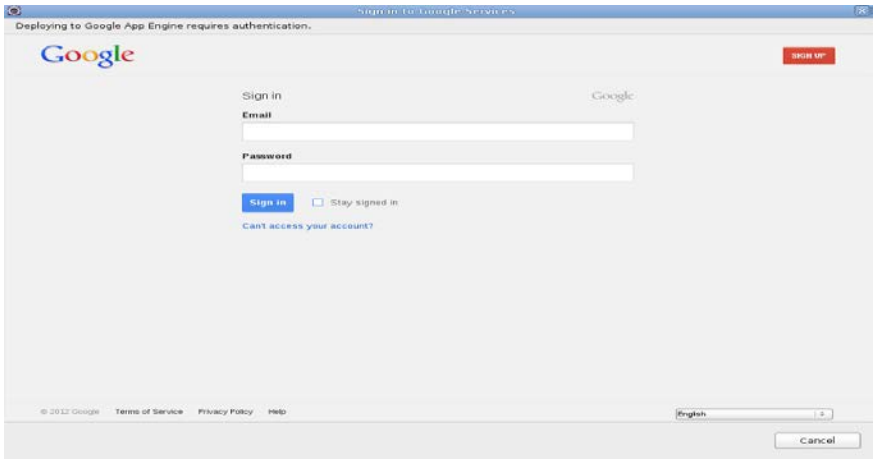

Figura 9.3 Autentificación de Google App Engine

Una vez que se hayan ingresado los datos correctamente, el plugin de Google App Engine para Eclipse solicitará permisos para administrar la aplicación y otros elementos relacionados a la misma, se elije la opción Permitir acceso.

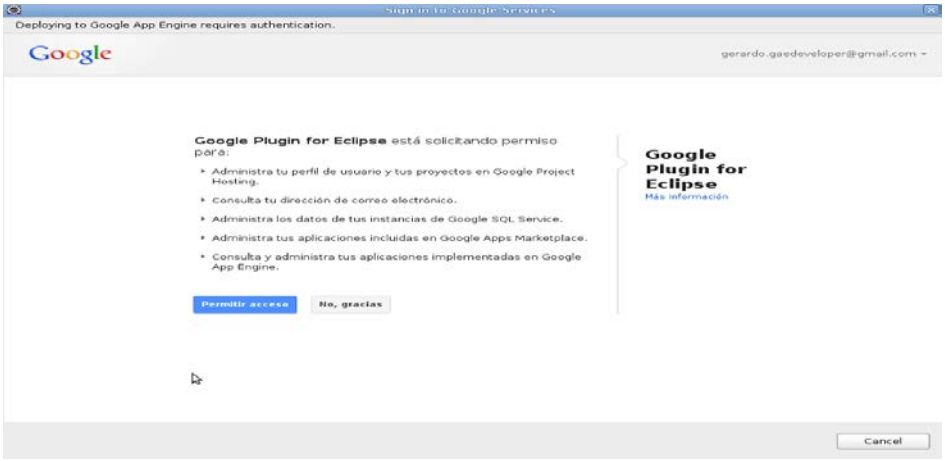

Figura 9.4 Otorgamiento de permisos al plugin de Eclipse

El plugin presentará una pantalla donde se podrá elegir el nombre del proyecto que se desea subir, una vez elegido se da click en Deploy

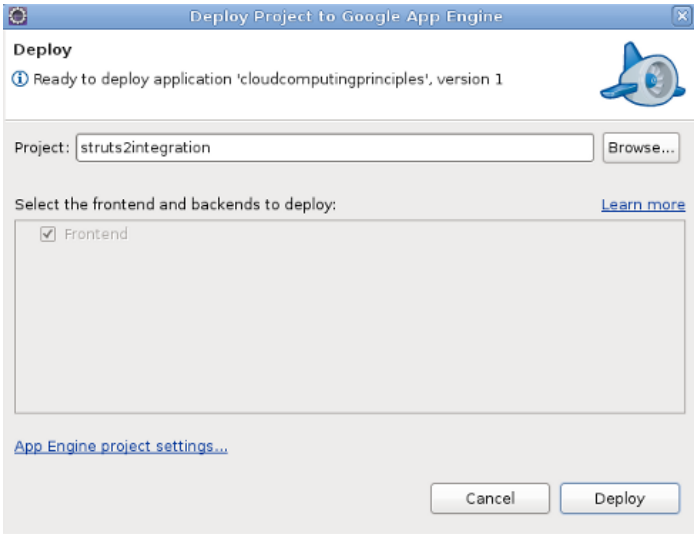

Figura 9.5 Selección del proyecto a subir

Terminados los puntos anteriores el plugin se encargará de subir la aplicación a Google App Engine

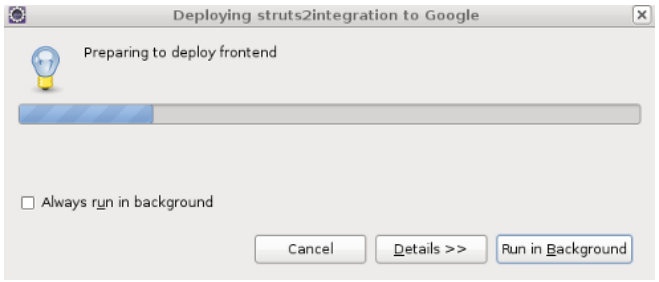

Figura 9.6 Progreso de la instalación de la aplicación en los servidores de Google App Engine

Una vez que el plugin haya terminado de instalar la aplicación, mostrará un resumen del número de archivos subidos y el estatus de la operación.

```
------------ Deploying frontend ------------

Preparing to deploy:
        Created staging directory at: '/tmp/appcfg7643093325916236710.tmp'
        Scanning for jsp files.
        Compiling jsp files.
        Scanning files on local disk.
        Initiating update.
        Cloning 2 static files.
        Cloning 48 application files.

Deploying:
        Uploading 12 files.
        Uploaded 3 files.
        Uploaded 6 files.
        Uploaded 9 files.
        Uploaded 12 files.
        Initializing precompilation...
        Sending batch containing 12 file(s) totaling 41KB.
        Deploying new version.

Verifying availability:
        Will check again in 1 seconds.
        Will check again in 2 seconds.
        Will check again in 4 seconds.
        Will check again in 8 seconds.
        Closing update: new version is ready to start serving.

Updating datastore:
        Uploading index definitions.

Deployment completed successfully
```

Figura 9.7 Estatus de la operación de instalación de la aplicación

Después de lo cual podremos consultar nuestra aplicación en los servidores de Google App Engine

Figura 9.8 Consulta de la aplicación en los servidores de Google App Engine

Con lo cual se termina la instalación de la aplicación.

Conclusiones

A lo largo del trabajo se han presentado diversas características de Google App Engine, las cuales resultan de importancia para aquellos desarrolladores que deseen utilizarla como una plataforma para el diseño de aplicaciones web. El trabajo no pretendió el abarcar cada aspecto de la plataforma debido a que la cantidad de información sería excesiva y además de que el trabajo siempre permanecerá inconcluso ya que constantemente se agregan nuevas características, la información más actualizada siempre se podrá consultar en la documentación en línea.

El autor ha elegido el entorno Java de Google App Engine porque está familiarizado con él y no porque lo considere mejor al proporcionado por Python o Go (el cual al momento en que se realizó éste trabajo se encuentra en fase experimental), además Java es una plataforma estable utilizada por muchas compañías a nivel mundial, de la cual se crean nuevas librerías y frameworks cada día, lo cual permite integrar novedosas y elegantes características a las aplicaciones desarrolladas.

Los características de la plataforma que se han presentado son aquellas que el autor considera relevantes a una amplia gama de aplicaciones web y son aquellas que son necesarias para comenzar a utilizar Google App Engine, pero no significa que sean necesarias todas ellas para realizar una aplicación web ni tampoco que sean las únicas características que existen, por ejemplo, no se consideró la utilización de OAuth para autentificar a los usuarios, lo cual permite que usuarios de otros servicios, por ejemplo, Facebook o Twitter ingresen al sistema o que la aplicación sea capaz de utilizar los datos de estas redes sociales para otorgar nuevos servicios al usuario.

Al inicio del trabajo se busco también el demostrar que Google App Engine es una plataforma apropiada y viable para aquellas aplicaciones que cumplen los siguientes requisitos:

- Presentan una baja latencia
- No realizan un procesamiento intensivo de los datos

Teniendo en cuenta lo mencionado en el trabajo podemos concluir lo siguiente:

1. Las aplicaciones que presentan una alta latencia verán disminuida su capacidad de escalamiento por dos cuestiones fundamentales, por una parte Google App Engine tiene preferencia por aquellas aplicaciones que responden de manera rápida a peticiones del usuario, relegando aquellas con tiempos de respuesta más lento, adicionalmente las peticiones tienen un tiempo límite de respuesta, si la petición tarda mucho en procesarse, App Engine terminará la petición con código de error, de forma que la aplicación nunca tendrá un progreso real y siempre se agotará su tiempo de respuesta antes de entregar una respuesta al usuario.

2. El modelo de persistencia proporcionado por App Engine requiere que se cree un índice de cada consulta posible que pueda realizar la aplicación, los índices son fijos y no se crean nuevos una vez que la aplicación se ejecuta, por tanto no es posible el realizar consultas que no hayan sido previstas con anterioridad lo cual limita la exploración de datos que puede realizar el usuario. Si bien la aplicación puede almacenar una cantidad significativa de datos la exploración de los mismos esta limitada en tiempo y tamaño, toda consulta tiene un tiempo límite, si dicho tiempo es excedido App Engine lanzará una excepción, adicionalmente, la cantidad de información que puede obtenerse del almacén esta limitada en tamaño lo cual impide el extraer volúmenes

grandes de información.

El autor del presente trabajo considera en base a lo antes expuesto que la plataforma es un buen punto de partida para un grupo considerable de aplicaciones web, pero no lo recomendaría para aplicaciones que ocupan la minería de datos, ni tampoco para aplicaciones cuyos tiempos de respuesta son considerables.

Los alcances del trabajo son limitados y queda corto en dos cuestiones importantes, por una parte no se presenta una comparación con otros productos de Plataforma como servicio, lo cual permitiría realizar un análisis cualitativo y cuantitativo de Google App Engine, por otra, el trabajo no explora las capacidades de la plataforma con una aplicación concreta lo cual permitiría el realizar métricas y verificar el comportamiento ante cargas reales. Estas cuestiones sin embargo requerirían un enfoque completamente diferente del trabajo presentado.

El caso de estudio que corresponde a la integración del framework Struts 2 con Google App Engine tiene como objeto el facilitar el desarrollo de aplicaciones web, sin embargo, este framework no es la única posibilidad, también es posible el utilizar Spring o algún otro framework basado en Java. Al final estas decisiones toman en cuenta otros aspectos, pero siempre es bueno el saber las posibilidades que existen.

El autor espera que el lector tenga al momento de finalizar esta obra una visión más clara de la plataforma y que la información contenida en el trabajo le permita comenzar a desarrollar sus propias aplicaciones.

Al momento de realizar el trabajo, Java liberó la versión 1.7 de su lenguaje, además existe la especificación Java Servlet 3.0, el autor espera que estas características sean integradas a la plataforma de modo que el desarrollador de aplicaciones tome ventajas de las nuevas características que otorgan.

Apéndice A

A continuación se muestra el código fuente de la clase freemarker.core.TextBlock.java, utilizado para integrar Struts 2 con Google App Engine.

```
/*
 * Copyright (c) 2003 The Visigoth Software Society. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution, if
 *    any, must include the following acknowledgement:
 *       "This product includes software developed by the
 *        Visigoth Software Society (http://www.visigoths.org/)."
 *    Alternately, this acknowledgement may appear in the software
itself,
 *    if and wherever such third-party acknowledgements normally
appear.
 *
 * 4. Neither the name "FreeMarker", "Visigoth", nor any of the names
of the
 *    project contributors may be used to endorse or promote products
derived
 *    from this software without prior written permission. For written
 *    permission, please contact visigo...@visigoths.org.
 *
 * 5. Products derived from this software may not be called
"FreeMarker" or "Visigoth"
 *    nor may "FreeMarker" or "Visigoth" appear in their names
 *    without prior written permission of the Visigoth Software
Society.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE VISIGOTH SOFTWARE SOCIETY OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
```

```
 *
=====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Visigoth Software Society. For more
 * information on the Visigoth Software Society, please see
 * http://www.visigoths.org/
 */
package freemarker.core;

import java.io.IOException;

/**
 * A TemplateElement representing a block of plain text.
 *
 * @version $Id: TextBlock.java,v 1.17 2004/01/06 17:06:42 szegedia Exp $
 */
public final class TextBlock extends TemplateElement {
        private static final char[] EMPTY_CHAR_ARRAY = new char[0];
        static  final  TextBlock  EMPTY_BLOCK  =  new  TextBlock(EMPTY_CHAR_ARRAY,
false);
        // We're using char[] instead of String for storing the text block because
        // Writer.write(String) involves copying the String contents to a char[]
        // using String.getChars(), and then calling Writer.write(char[]). By
        // using Writer.write(char[]) directly, we avoid array copying on each
        // write.
        private char[] text;
        private final boolean unparsed;

        public TextBlock(String text) {
            this(text, false);
        }

        public TextBlock(String text, boolean unparsed) {
            this(text.toCharArray(), unparsed);
        }

        private TextBlock(char[] text, boolean unparsed) {
            this.text = text;
            this.unparsed = unparsed;
        }

        /**
         * Simply outputs the text.
         */
        public void accept(Environment env) throws IOException {
            env.getOut().write(text);
        }

        public String getCanonicalForm() {
            String text = new String(this.text);
            if (unparsed) {
                  return "<#noparse>" + text + "</#noparse>";
            }
            return text;
        }

        public String getDescription() {
            String s = new String(text).trim();
```

```
    if (s.length() == 0) {
          return "whitespace";
    }
    if (s.length() > 20) {
          s = s.substring(0, 20) + "...";
          s = s.replace('\n', ' ');
          s = s.replace('\r', ' ');
    }
    return "text block (" + s + ")";
}

TemplateElement postParseCleanup(boolean stripWhitespace) {
    if (text.length == 0)
          return this;
    int openingCharsToStrip = 0, trailingCharsToStrip = 0;
    boolean deliberateLeftTrim = deliberateLeftTrim();
    boolean deliberateRightTrim = deliberateRightTrim();
    if (!stripWhitespace || text.length == 0) {
          return this;
    }
    if (parent.parent == null && previousSibling() == null)
          return this;
    if (!deliberateLeftTrim) {
          trailingCharsToStrip = trailingCharsToStrip();
    }
    if (!deliberateRightTrim) {
          openingCharsToStrip = openingCharsToStrip();
    }
    if (openingCharsToStrip == 0 && trailingCharsToStrip == 0) {
          return this;
    }
    this.text = substring(text, openingCharsToStrip, text.length
                - trailingCharsToStrip);
    if (openingCharsToStrip > 0) {
          this.beginLine++;
          this.beginColumn = 1;
    }
    if (trailingCharsToStrip > 0) {
          this.endColumn = 0;
    }
    return this;
}

/**
 * Scans forward the nodes on the same line to see whether there is a
 * deliberate left trim in effect. Returns true if the left trim was
 * present.
 */
private boolean deliberateLeftTrim() {
    boolean result = false;
    for (TemplateElement elem = this.nextTerminalNode(); elem != null
                && elem.beginLine == this.endLine; elem = elem
                .nextTerminalNode()) {
          if (elem instanceof TrimInstruction) {
                TrimInstruction ti = (TrimInstruction) elem;
                if (!ti.left && !ti.right) {
                      result = true;
                }
                if (ti.left) {
```

```
                        result = true;
                        int lastNewLineIndex = lastNewLineIndex();
                        if (lastNewLineIndex >= 0 || beginColumn == 1) {
                              char[] firstPart = substring(text, 0,
                                          lastNewLineIndex + 1);
                              char[] lastLine = substring(text, 1 +
                                          lastNewLineIndex);
                              if (trim(lastLine).length == 0) {
                                    this.text = firstPart;
                                    this.endColumn = 0;
                              } else {
                                    int i = 0;
                                    while (Character
                                          .isWhitespace(lastLine[i])) {
                                          i++;
                                    }
                                    char[] printablePart=substring(lastLine,
                                                i);
                                    this.text = concat(firstPart,
                                                printablePart);
                              }
                        }
                  }
            }
            if (result) {
            }
            return result;
      }

      /**
       * Checks for the presence of a t or rt directive on the same line.
Returns
       * true if the right trim directive was present.
       */
      private boolean deliberateRightTrim() {
            boolean result = false;
            for (TemplateElement elem = this.prevTerminalNode(); elem != null
                        && elem.endLine == this.beginLine; elem = elem
                        .prevTerminalNode()) {
                  if (elem instanceof TrimInstruction) {
                        TrimInstruction ti = (TrimInstruction) elem;
                        if (!ti.left && !ti.right) {
                              result = true;
                        }
                        if (ti.right) {
                              result = true;
                              int firstLineIndex = firstNewLineIndex() + 1;
                              if (firstLineIndex == 0) {
                                    return false;
                              }
                              if (text.length > firstLineIndex
                                          && text[firstLineIndex - 1] == '\r'
                                          && text[firstLineIndex] == '\n') {
                                    firstLineIndex++;
                              }
                              char[] trailingPart = substring(text,firstLineIndex);
                              char[] openingPart = substring(text, 0,
                                          firstLineIndex);
```

```
                        if (trim(openingPart).length == 0) {
                              this.text = trailingPart;
                              this.beginLine++;
                              this.beginColumn = 1;
                        } else {
                              int lastNonWS = openingPart.length - 1;
                              while  (Character.isWhitespace(text[lastNonWS]))
{
                                    lastNonWS--;
                              }
                              char[] printablePart = substring(text, 0,
                                    lastNonWS + 1);
                              if (trim(trailingPart).length == 0) {
                                    // THIS BLOCK IS HEINOUS! THERE MUST BE A
                                          BETTER
                                    // WAY! REVISIT (JR)
                                    boolean trimTrailingPart = true;
                                    for (TemplateElement te =
                                                this.nextTerminalNode();
                                          te != null && te.beginLine ==
                                                this.endLine;
                                          te = te.nextTerminalNode()) {
                                          if (te.heedsOpeningWhitespace()) {
                                                trimTrailingPart = false;
                                          }
                                          if (te instanceof TrimInstruction
                                                      && ((TrimInstruction)
                                                      te).left) {
                                                trimTrailingPart = true;
                                                break;
                                          }
                                    }
                                    if (trimTrailingPart)
                                          trailingPart = EMPTY_CHAR_ARRAY;
                              }
                              this.text = concat(printablePart,trailingPart);
                        }
                  }
            }
      }
      return result;
}

/*
 * private String leftTrim(String s) { int i =0; while (i<s.length()) { if
 * (!Character.isWhitespace(s.charAt(i))) break; ++i; } return
 * s.substring(i); }
 */
private int firstNewLineIndex() {
    String content = new String(text);
    int newlineIndex1 = content.indexOf('\n');
    int newlineIndex2 = content.indexOf('\r');
    int result = newlineIndex1 >= 0 ? newlineIndex1 : newlineIndex2;
    if (newlineIndex1 >= 0 && newlineIndex2 >= 0) {
          result = Math.min(newlineIndex1, newlineIndex2);
    }
    return result;
}
```

```
        private int lastNewLineIndex() {
            String content = new String(text);
            return Math.max(content.lastIndexOf('\r'), content.lastIndexOf('\n'));
        }

        /**
         * figures out how many opening whitespace characters to strip in the
         * post-parse cleanup phase.
         */
        private int openingCharsToStrip() {
            int newlineIndex = firstNewLineIndex();
            if (newlineIndex == -1 && beginColumn != 1) {
                  return 0;
            }
            ++newlineIndex;
            if (text.length > newlineIndex) {
                  if (newlineIndex > 0 && text[newlineIndex - 1] == '\r'
                              && text[newlineIndex] == '\n') {
                        ++newlineIndex;
                  }
            }
            if (new String(text).substring(0, newlineIndex).trim().length() > 0) {
                  return 0;
            }
            // We look at the preceding elements on the line to see if we should
            // strip the opening newline and any whitespace preceding it.
            for (TemplateElement elem = this.prevTerminalNode(); elem != null
                        && elem.endLine == this.beginLine; elem = elem
                        .prevTerminalNode()) {
                  if (elem.heedsOpeningWhitespace()) {
                        return 0;
                  }
            }
            return newlineIndex;
        }

        /**
         * figures out how many trailing whitespace characters to strip in the
         * post-parse cleanup phase.
         */
        private int trailingCharsToStrip() {
        String content = new String(text);
        int lastNewlineIndex = lastNewLineIndex();
        if (lastNewlineIndex == -1 && beginColumn != 1) {
            return 0;
        }
        String substring = content.substring(lastNewlineIndex +1);
        if (substring.trim().length() >0) {
            return 0;
        }
// We look at the elements afterward on the same line to see if we should
// strip any whitespace after the last newline
        for (TemplateElement elem = this.nextTerminalNode();
             elem != null && elem.beginLine == this.endLine;
             elem = elem.nextTerminalNode())
        {
            if (elem.heedsTrailingWhitespace())
            {
                return 0;
```

```
        }
    }
    return substring.length();
}
    boolean heedsTrailingWhitespace() {
        if (isIgnorable()) {
              return false;
        }
        for (int i = 0; i < text.length; i++) {
              char c = text[i];
              if (c == '\n' || c == '\r') {
                    return false;
              }
              if (!Character.isWhitespace(c)) {
                    return true;
              }
        }
        return true;
    }

    boolean heedsOpeningWhitespace() {
        if (isIgnorable()) {
              return false;
        }
        for (int i = text.length - 1; i >= 0; i--) {
              char c = text[i];
              if (c == '\n' || c == '\r') {
                    return false;
              }
              if (!Character.isWhitespace(c)) {
                    return true;
              }
        }
        return true;
    }

    boolean isIgnorable() {
        if (text == null || text.length == 0) {
              return true;
        }
        if (!isWhitespace()) {
              return false;
        }
        // do the trick
        boolean atTopLevel = true;
        TemplateElement prevSibling = previousSibling();
        TemplateElement nextSibling = nextSibling();
        return ((prevSibling == null && atTopLevel)
                    || nonOutputtingType(prevSibling))
                    && ((nextSibling == null && atTopLevel)
                    || nonOutputtingType(nextSibling));
    }

    private boolean nonOutputtingType(TemplateElement element) {
        return (element instanceof Macro || element instanceof Assignment
                    || element instanceof AssignmentInstruction
                    || element instanceof PropertySetting
                    || element instanceof LibraryLoad || element instanceof
                          Comment);
```

```
        }

        private static char[] substring(char[] c, int from, int to) {
            char[] c2 = new char[to - from];
            System.arraycopy(c, from, c2, 0, c2.length);
            return c2;
        }

        private static char[] substring(char[] c, int from) {
            return substring(c, from, c.length);
        }

        private static char[] trim(char[] c) {
            if (c.length == 0) {
                  return c;
            }
            return new String(c).trim().toCharArray();
        }

        private static char[] concat(char[] c1, char[] c2) {
            char[] c = new char[c1.length + c2.length];
            System.arraycopy(c1, 0, c, 0, c1.length);
            System.arraycopy(c2, 0, c, c1.length, c2.length);
            return c;
        }

        boolean isWhitespace() {
            return text == null || trim(text).length == 0;
        }
}
```

Referencias

[1] Kyle Roche and Jeff Douglas ,
Beginning Java Google App Engine, Primera Edición,
Apress 2009, 236 páginas.

[2] Dan Sanderson
Programming Google App Engine, Primera edición,
O'Reilly Google Press 2010, 370 páginas

[3] http://code.google.com/intl/en/appengine/articles/cf1-text.html
Google App Engine Campfire One - April 2008,
24/Septiembre/2012

[4] https://developers.google.com/appengine/?hl=es,
Google App Engine (Sitio oficial),
27/Septiembre/2012

[5] http://www.oracle.com/technetwork/java/javaee/overview/index.html,
Sitio oficial de oracle con la descripción de la plataforma JEE,
27/Septiembre/2012

[6] https://developers.google.com/appengine/docs/java/runtime?hl=en,
Entorno Java Servlet presente en Google App Engine,
27/Septiembre/2012

[7] http://code.google.com/p/googleappengine/wiki/WillItPlayInJava,
Lista los niveles de compatibilidad de diferentes tecnologías Java con Google App Engine,
22/Septiembre/2012

[8] http://struts.apache.org/2.1.8/docs/simple-setup.html,
Configuración básica de un proyecto que ocupe Struts 2,
27/Septiembre/2012

[9] http://struts.apache.org/2.x/,
Sitio oficial del proyecto Struts 2,
22/Septiembre/2012

[10] http://www.mkyong.com/google-app-engine/google-app-engine-struts-2-example/,
Ejemplo de integración de Struts 2 y Google App Engine,
26/Septiembre/2012

[11] http://whyjava.wordpress.com/2009/08/30/creating-struts2-application-on-google-app-engine-gae/,
Creación de un aplicación Struts 2 utilizando Google App Engine,
26/Septiembre/2012
[12] http://code.google.com/p/googleappengine/issues/detail?id=6928,
Error en la configuración de la zona horaria del Plugin Eclipse de Google App Engine,

27/Septiembre/2012

[13] https://dev.twitter.com/docs/api
API REST de Twitter,
27/Septiembre/2012

[14] https://developers.google.com/appengine/docs/python/datastore/gqlreference,
Descripción del lenguaje GQL,
25/Septiembre/2012