Published on *Marco de Desarrollo de la Junta de Andalucía* (http://madeja.i-administracion.junta-andalucia.es/servicios/madeja)

Conceptos sobre la cache de objetos

- **Área:** Arquitectura Tecnológica
- **Carácter del recurso:** Recomendado

Código: RECU-0219
Tipo de recurso: Referencia

Descripción

Uno de los objetivos básicos en un enfoque de mejora de rendimiento es minimizar los accesos a disco de una aplicación. La caché es una memoria de acceso rápido, que con una buena configuración permite elevar el rendimiento de las aplicaciones de forma significativa. La caché puede ser utilizada de forma concurrente, aprovechando varios usuarios las mismas estructuras de datos que ya se han cargado.

Esta concurrencia, contrae también algunos inconvenientes ya que pueden provocar que los datos que utilizan los diferentes usuarios no estén relacionados provocando que la cache se desborde y este continuamente cargando y expulsando datos, siendo desfavorable para el rendimiento en esta situación.

El proceso habitual con le que trabaja una caché es el siguiente:

1. Un usuario realiza una operación que provoca un consulta.
2. Se realiza una verificación para comprobar que los objetos se encuentran en cache.
3. Si los encuentra, los devuelve directamente.Si no los encuentra, los carga desde un recurso externo y los registra en caché para futuros accesos.

Problemas redundantes con el manejo de cache

Cuando se diseña un sistema de caché es interesante considerar los problemas comunes que afectan al diseño:

Modificación concurrente de los datos

Este problema es muy frecuente, de hecho existe la recomendación de solo utilizar caches para accesos de lectura o para datos que se actualicen con poca frecuencia. Existen políticas para evitar la incoherencia en la cache.

En ocasiones, se realizan escaneos para comprobar el estado de los objetos. Se añaden bits a la cache que notificación que se han modificado los datos y que asegure que cada modificación se registra en disco. Sea como sea, lo que está claro es que necesitamos algún mecanismo para controlar este problema.

Sistemas con cache distribuida

Un problema potencialmente importante sucede en lo sistemas de caché distribuidos. Estas caches pueden ser utilizadas por separado por cualquier aplicación. Las caches distribuidas además de lanzar una búsqueda para comprobar si existe el objeto en su estructura interna, sino que emplea un tiempo en la sincronización de los nodos de la caché en las operaciones que se hayan realizado sobre ella.

Los sistemas que implementan un sistema de caché distribuidos pueden ser muy diversos. Una posibilidad, es replicar una caché en cada nodo, teniendo asumido el coste de la sincronización de los datos. Otra posibilidad es tener una caché repartida entre los nodos. En esta situación se asume el coste de solicitar y transportar los objetos de un nodo a otro nodo solicitante. Existe una tercera posibilidad, implementando caches independientes que se sincronizan con una caché común cada cierto tiempo. Sea cual sea la posibilidad elegida, aparecen una serie de costes asumidos en la comunicación, inexistentes previamente.

El coste asociado a las actualizaciones a realizar mediante el uso de las caches distribuidas, es importante. Si un nodo se actualiza, lo comunica al resto para que se actualicen también. Obviamente para realizar estas actualizaciones, será necesario realizar una serie de transacciones entre los nodos, decrementando el rendimiento. A pesar de introducir una caché distribuida en un sistema, no aseguramos un aumento del rendimiento (especialmente si es un sistema con constantes actualizaciones).

Actualización de los datos desde aplicaciones que no utilizan caché

Hay situaciones que aplicaciones que no están bajo el control del desarrollador acceden a los datos contenidos en la caché. Se recomienda que no se usen caches para este tipo de datos compartidos.

Hay un concepto informático básico que defiende que solo si podemos asegurar la información podemos realizar un tratamiento sobre la misma. Almacenar datos en caché de los que no somos propietarios, por lo que el mantenerlos en la

caché es potencialmente muy peligroso. Se corre el riesgo de crear una estructura de cache buena que no funcione por un acceso concurrente a este tipo de datos.

Arquitectura de la caché

El estándar de la arquitectura persistencia JPA divide la estructura de la cache en dos niveles.

Cache de primer nivel

La caché denominada de primer nivel se corresponde con el objeto de sesión. En esta caché se van almacenando los objetos que se recuperan del disco, de manera que, si vuelven a solicitarse , ya están en cache y no es necesario acceder al disco. La caché de primer nivel es el punto de acceso a los objetos, evitando que los cambios sobre dichos objetos no sean visibles. Con el control de la sesión se evita que los errores alcancen a todo el motor de persistencia y se queden dentro que la sesión.

Cuando se realizan operaciones de actualización de los datos (consultas, abrir, borrar...) se interactúa de forma transparente con la caché de primer nivel. Este tipo de operaciones no se ejecutan directamente, se almacenan en caché de primer nivel. Para ejecutar dichas operaciones a base de datos es necesario realizar una llamada al método **flush()** de la interfaz **Session** o al método **commit()** de la transacción abierta para esa sesión.

Si queremos eliminar estos objetos de la caché, se puede utilizar el método **evict()** que elimina de la caché el objeto pasado como parámetro. Si queremos eliminar todos los objetos que se encuentran alojados en cache podemos utilizar el método **clear()**. Ambos métodos son útiles para políticas de sincronización entre caches.

A través del pool de conexiones, cada vez que se inicia una sesión, se obtiene un conexión a base de datos. Es muy importante liberar esta conexión cuando se cierra la sesión. A medida que se abren sesiones, más recursos se consumen del pool y se corre el riesgo de agotarlo. Para evitar esta problemática, normalmente se permite hacer desconexiones momentáneas. De esta manera liberamos la conexión establecida con la base de datos facilitando un aumento de escalabilidad en el sistema. Se puede volver a conectar la sesión a la base de datos del pool de conexiones.

La manera de utilizar una caché dentro de una aplicación se puede resumir, de manera breve, de la siguiente manera: Una operación que se realiza con el servidor se denomina unidad de trabajo. Cuando se realizan varias operaciones, normalmente hablamos de una transacción. En el contexto de una aplicación se ejecutarán múltiples transacciones de aplicación. Tras esta definición, nos encontramos con los siguientes modelos de gestión de la sesión, es decir, de la caché del primer nivel: sesión global, sesión por aplicación, sesión por transacción y sesión por proceso.

Sesión Global

Es la posibilidad más simple. Se mantiene una sola sesión en el servidor y sobre ella se manejan todas las peticiones y se realizan las operaciones asociadas con la base de datos. Con este manejo de la sesión se mantiene la ventaja de que no se producen errores de actualización de concurrencia de caches, ya que obviamente sólo se mantiene una. Si se producen errores, se invalida la sesión y se crea una nueva. Por ello, es conveniente que se realicen periódicamente operaciones de **flush()** o **commit()**, ya que al mantener sólo una única sesión, procesamos una única transacción, y una excepción ocasionaría un **rollback()** y perderíamos todas las modificaciones hechas hasta el momento.

Como contrapunto, esta solución es poco escalable. Se corre el riesgo de que si la carga de trabajo crece en demasía, el sistema se vuelva inestable. Por lo tanto, sólo hay que considerar esta solución si está destinada para sistemas pequeños y con pocos usuarios. Además, es necesario que estos usuarios presenten muy poca concurrencia. Si se prevé que existirá un aumento de carga o de concurrencia en un futuro, es mejor no plantear esta solución.

Sesión por usuario o aplicación

Este modelo se basa en la idea de asignar una sesión por usuario o aplicación que interactúe con el servidor. La sesión está asociada al usuario que se conecta a la aplicación, así que cuando se termina con la ejecución de la aplicación, la sesión se cierra. Dada este relación, normalmente, los sistemas implementan algún método que controlo el tiempo límite de una sesión para informar, de forma periódica, al servidor del estado del cliente.

Este modelo exige un manejo de métodos que permitan conectar y reconectar sesiones en función de la inactividad de los clientes, porque si no se logra, existe un riesgo grande de agotamiento frecuente del pool de conexiones. Especialmente sensibles son las excepciones y errores, donde es necesario liberar las sesiones ya que pueden quedar latentes y ayudar al agotamiento del pool de conexiones. Con este método, se logra mantener activa la caché de usuario. Obviamente, si cada usuario mantiene su caché, existe un riesgo en la concurrencia de usuarios y sus actualizaciones individuales.

Sesión por unidad de trabajo

Este modelo se basa en la idea de crear sesiones por cada unidad de trabajo. Así, cada vez que se ejecuta una operación, se crea una sesión. Una vez finaliza la unidad de trabajo, la sesión se cierra. Con este sistema hay que tener en cuenta un riesgo potencial grave, se necesita que cada operación conlleve varios accesos a la base de datos porque si no, estaríamos en el caso de abrir y cerrar sesiones por cada acceso a la base de datos.

Con este modelo de gestión, se obtiene una mejor escalabilidad y una mejor tolerancia a fallos. Dado el carácter de una operación, normalmente las conexiones no tienen mucha duración por lo que se recogen y se devuelven con frecuencia al pool, evitando problemas de agotamiento. Las transacciones también suelen ser cortas, por lo que las necesidades de sincronización, bloqueos, etc. disminuyen considerablemente. Así mismo, las excepciones ya no son tan problemáticas. En el caso de un fallo, eliminan la sesión y se recupera una nueva. La operación de vuelta atrás sólo afecta a una unidad de trabajo, lo que permite una recuperación más sencilla y efectiva.

El mayor problema de este modelo es que desaprovecha las cualidades de la caché. La caché durará lo mismo que una unidad de trabajo, por tanto, sólo el tiempo de ejecución de unas operaciones por lo tanto cuando concluyan, lo que se haya

almacenado en la caché se pierde. Por lo tanto, ante nuevos accesos, será necesario volver a buscar los datos en la base de datos con la penalidad consecuente. Este aspecto, es principalmente peligrosos en las aplicaciones de escritorio, donde se espera una latencia poco significativa.

Esta solución es manejable en entornos que requieren de gran escalabilidad y que no necesiten almacenar muchos datos en caché. Aplicaciones con unidades de trabajo, que dispongan de varios accesos a la base de datos son candidatas a usar este modelo. Otra característica reseñable es la reducción de problemas de sincronización. Si las unidades de trabajo no son muy amplias, es complicado que aparezcan inconsistencias y no se causaría demasiada contención en los bloqueos.

En el caso de aplicaciones que realizan accesos frecuentes a estructuras complejas de datos, que requieren muchas consultas, es mejor solución implantar una caché de segundo nivel o utilizar alguno de los modelos anteriores.

Sesión por transacción de aplicación

Este modelo combina ideas de los anteriores. La sesión se mantiene mientras dure toda la transacción. Como una transacción esta compuesta de varias unidades de trabajo, en cada uno de ella se realiza un mantenimiento de la conexión (métodos de reconectar y desconectar) para mantener el pool de conexiones y la escalabilidad. Cuando se finaliza la transacción, se finaliza la sesión. Esta orientado principalmente al mundo web, donde las transacciones suelen requerir secuencias formales que incluyen gran variedad de pasos para completar su ejecución.

Como es una aproximación basada en varios modelos, intenta aprovechar al máximo las principales ventajas de cada uno de ellos. Por un lado la sesión se mantiene abierta durante el tiempo que dura la transacción mejorando la escalabilidad con respecto a las primeras aproximaciones de gestión de la sesión. Lo más lógico es que unidades de trabajo relacionadas, mantengan cierta relación en sus objetos, por lo que la reutilización de los objetos alojados en caché es alta y se encuentran dentro de la cache de primer nivel.

Aun así, este modelo también tiene sus desventajas. En el caso de que las transacciones sean cortas, se corre el riesgo de desaprovechar la caché de primer nivel. Si estamos en el caso de transacciones largas, se corre el riesgo de caer en muchas actualizaciones concurrentes. Para decidir si trabajar con este método hay que evaluar el tipo de transacciones que van a existir.

Caché de segundo nivel

Existe un problema grave con las actualizaciones concurrentes para cualquier sistema de caché. Da igual el modelo que utilice, que siempre existe un riesgo de errores en la sincronización. Con la introducción de una caché de segundo nivel, se reducen este tipo de problemas.

Se introduce una caché de segundo nivel se adhiere a la sesión, de manera que va incluyendo todos los fallos que se detectan en la misma. De esta manera se soluciona el problema de las actualizaciones concurrentes que se deriva del uso de varias sesiones.

Es muy útil usar una caché de segundo nivel si se realiza una gestión de la sesión basada en la transacción de aplicaciones o en el modelo de unidad de trabajo. El principal problema que adolecían estos modelos es el desaprovechamiento de la caché de primer nivel, que gracias al incluir la caché de segundo nivel se reducen considerablemente. De esta manera, cuando se cierran las sesiones ya no supone un problema tan grave, ya que, aunque haya que realizar las consultas recurrentes, los datos estarán en la caché de segundo nivel.

Estrategias de concurrencia

Resulta necesario establecer una estrategia de concurrencia que permita sincronizar la caché de primer nivel con la caché de segundo nivel, y ésta última con la base de datos. Existen cuatro modelos de estrategias de concurrencia predefinidas. A continuación aparecen listadas por orden de restricciones en términos de aislamiento transaccional.

- **transactional**: Garantiza un nivel de aislamiento hasta repetición de lectura (repeatable read), si se necesita. Es el nivel más estricto. Es conveniente su uso cuando no podamos permitirnos datos que queden desfasados. Esta estrategia sólo se puede utilizar en clusters, es decir, con caches distribuidas.
- **read-write**: Mantiene un aislamiento hasta el nivel de commited, utilizando un sistema de marcas de tiempo (timestamps). Su mayor utilidad se da en el mismo caso que para la estrategia transactional pero con la diferencia de en que esta estrategia no se puede utilizar en clusters.
- **nonstrict read-write**: No ofrece ninguna garantía de consistencia entre la caché y la base de datos. Para sincronizar los objetos de la caché con la base de datos se utilizan **timeouts,** de modo que cuando caduca el timeout se recargan los datos. Con esta estrategia, tenemos un intervalo en el cual tenemos el riesgo de obtener objetos desfasados. Cuando Hibernate realiza una operación de **flush()** en una sesión, se invalidan los objetos de la caché de segundo nivel. Aún así, esta es una operación asíncrona y no tenemos nunca garantías de que otro usuario no pueda leer datos erróneos. A pesar de todo esto, esta estrategia es ideal para almacenar datos que no sean demasiado críticos.
- **read-only**: Es la estrategia de concurrencia menos estricta. Ideal para datos que nunca cambian

Caches distribuidas

En ocasiones, cuando el número de usuarios es muy elevado, es muy posible que sea necesario montar una arquitectura distribuida, normalmente un cluster. Lo único que es necesario configurar es la caché de segundo nivel. Se puede montar una caché en un cluster de forma diversa. Si lo hacemos de forma sencilla, se monta sobre cada nodo del cluster una instancia del proveedor de cache. De esta manera se evitan problemas de sincronización entre nodos. El segundo modo, es instalar un proveedor de cache más avanzado que soporte la sincronización de las caches de los diferentes nodos.

Caché de consultas

El objeto de una caché de consultas es la reutilización de las mismas, con el fin de recuperar los datos de forma ágil. El mantenimiento de una caché de consultas sólo es útil en aplicaciones que realicen un número de consultas muy elevado y con un índice de repetición también muy elevado.

En este tipo de caches no se graban directamente los objetos, si no que se graban las claves de los mismos. Se hace así para evitar que se realice un consumo excesivo de memoria. El principal problema que se deriva, es que si realizamos una consulta y la repetimos tiempo después puede ocurrir que aunque esta consulta se encuentre en caché, quizás los objetos no estén en la caché de primer nivel ni en la de segundo nivel, ya que ambas pueden gestionar sus entidades de forma independiente a la caché de consultas. Como consecuencia de esto extraemos que aunque una consulta esté en la caché de consultas, no tiene por que evitar obligatoriamente que tengamos que acceder a la base de datos para recuperar los resultados.

Buenas prácticas y recomendaciones de uso

Como resumen de las principales recomendaciones establecidas dentro del recurso

- Establecer una estrategia de concurrencia que permita sincronizar la caché de primer nivel con la caché de segundo nivel
- Elegir adecuadamente un proveedor de la cache de segundo nivel
- Hacer uso de la cache distribuida cuando el número de usuario es muy elevado
- Hacer uso de la caché de consultas solo en aplicaciones que realicen un número de consultas muy elevado y con un índice de repetición también se muy elevado

Ejemplos

Ejemplo de interés

Enlaces externos

- Ejemplo de interés
- Pagina Wiki

Pautas

Área: Arquitectura » Arquitectura Tecnológica

Código	Título	Tipo	Carácter
LIBP-0073	Buenas prácticas en el manejo de la caché	Directriz	Obligatoria

Source URL: http://madeja.i-administracion.junta-andalucia.es/servicios/madeja/contenido/recurso/219