Lru cache android example

A tiny implementation of a secure memory cache that uses an LRU policy. This implementation prioritizes the simplicity of the API. If you want to use a rich API, use LruCache as part of Android. How to use the Cash Cash interface. It provides six techniques like fur. V get (K key) - Gets value for the specified key. V put (K key, V value) - Puts value in the cache for the specified key. V delete (K key) - Removes the entry for the key. The void is clear - clears all entries in the cache. int getMaxMemorySize () - Returns the maximum cache memory size. int getMemorySize () - Returns the current size of the cache memory. The API LruCache LruCache is certainly simple. Cash cache.put The default cache is ten. You can change the capacity with the help of a designer. Please override getValueSize if you control your memory size correctly. BitmapLruCache BitmapLruCache is a class that specializes in

Bitmap caching. Private static finale Bitmap A - Bitmap.createBitmap (1, 1, ALPHA_8); Cache cache - new BitmapLruCache cache.put Java 1.7 or more test support. Lru Cache should be used in applications where repetitive resource loads affect the smooth behavior of your application. For example, a photo gallery with large miniatures (128x128). Always be careful about the size of the Lru cache, as its too high setting can affect the application. Once Lru Cache is no longer useful, avoid any references to it so that the garbage collector can clear it of memory. For better performance, be sure to download resources such as bitmaps using best practices such as choosing the proper inSampleSize before adding it to the Lru cache. To add a resource to the cache, you need to provide a key and resource. First, make sure that the value is not in the cache of the already public void addResourceToMemoryCache (String Key, Bitmap resource) - if (memoryCache.get (key) - null memoryCache.put (key, resource); To get a resource from the cache, just translate the key of your resource (The Line in this example) to the public Bitmap getResourceFromMemoryCache - memoryCache.get (key); Lru Cache will store all added resources (values) for quick access until it reaches the memory limit, in which case it will drop the less used resource (value) to store the new one. To initiate the Lru cache, you need to maximize the value of the memory. This value depends on the application requirements and how important it is to keep your application running smoothly. The recommended value for a gallery of images, for example, will be 1/8 of the most accessible memory. Also note that Lru Cache works on the basis of key value. In the following example, the key is a line, and — Bitmap: int maxMemory (int) (Runtime.getRuntime().maxMemory() / 1024); int cacheSize - maxMemory / 8; LruCache<String, bitmap=> - памятьКаш - новый</String,> </String,> </String,> Bitmap'gt; (cacheSize) - protected int sizeOf (String Key, bitmap bitmap) - return bitmap.getByteCount (); SO Community 2016-11-02 2016-11-02 2017-05-22 Android Pedia LruCache public class expands java.lang.Object ↳ android.util.LruCache Cash, which contains strong links qlt;K, V'gt; on a limited number of values. Every time the value is accessed, it moves to the head of the queue. When the value is added to the full cache, the value at the end of this queue is evicted and may be suitable for garbage collection. If cached values hold resources that should be explicitly released, override the Removed record (boolean, K, V, V). If a cache blunder has to be calculated on demand for the relevant keys, override create (K). This simplifies the call code, allowing it to assume that the value will always be returned, even if there is a cache error. By default, the cache size is measured in the number of entries. Redefine the size of TheOf (K, V) for the size of the cache in different units. For example, this cache is limited to 4MiB bit cards: int cacheSize Nos. 4 and 1024 and 1024; 4MiB LruCache'lt;String, bitmap'gt; bitmapCache - new LruCache'lt;String, bitmap'gt; (cacheSize) - protected int sizeOf (String key, Bitmap value) - return value.getByteCount (); This class is a safe stream. We perform several cache operations atomically, synchronize in the cache: synchronized (cache) - if (cache.get (key) - null) - cache.put (key, value); This class does not allow you to use null as a key or value. The return value of zero from get (K), put (K, V) or delete (K) is unambiguous: the key was not in the cache. This class appeared in Android 3.1 (Honeycomb MR1); It's available as part of an Android support package for earlier releases. the final int createCount () Returns the number of times the creation (java.lang.Object) is returned. The final void is to evict The Clear cache, causing entryRemoved (boolean, K, V, V) on each deleted record. final int evictionCount () Returns the number of values that have been evicted. the final V key receives (K) returns the value of the key if it exists in the cache or can be created #create. the final int hitCount () Returns the number of times to receive (K) returned the value that was already present in the cache. For caches that don't override the sizeof (K, V), this returns the maximum number of entries in the cache. Final int missCount () Returns the number of times you get (K) returned zero or requires a new value to be created. the final value of V put (K key, V) Cash for the key. The final int putCount () Returns the number of times put (K, V) has been called. Final V Remove (K Key) Removes for the key, if it exists. in size (int maxSize) sets the size of the cache. For caches that don't override the size of TheOf (K, V), this returns the number of entries in the cache. The final card shot () Returns a copy of the card. current cache content, ordered from the least recent access to most recent access. the final toString line returns the view of the object line. invalid trimToSize (int maxSize) Remove senior entries until the total number of remaining entries is at or below the requested size. V create (K key) Called after skipping the cache to calculate the value for the corresponding key. Invalid entryRemoved (boolean evicted, K key, V oldValue, V newValue) called for records that were evicted or removed. int sizeOf (K key, V value) returns the entry size for the key and the value in the user-defined units. From the java.lang.Object Object class, the clone creates and returns a copy of this object. boolean (Object obj) indicates whether any other object is equal to this. invalid completion () Is called by the garbage collector at the facility when the garbage collection determines that there are no more references to the object. The final class of the getClass returns the time class of the subject. int hashCode () Returns the hash code value to the object. the final invalid to notify () will wake up one thread that is waiting on the monitor of this object. ToString returns the view of the object line. The final expectation of emptiness (long time out, int nanos) triggers anticipation of the current thread until another thread triggers the notification method () or the notifyAll method for that object, or some other thread interrupts the current thread, or a certain amount of real time has passed. The final expectation of emptiness (long time) triggers the wait for the current thread until another notification method () or notifyAll method has been triggered for that object, or a certain amount of time has passed. the final expectation of emptiness () causes the current thread to wait until another thread triggers the notification method () or the notifyAll method for that object. Public Designers of Public LruCache (int maxSize) MaxSize int Options: for caches that do not override the sizeof (K, V), this is the maximum number of entries in the cache. For all other caches, this is the maximum amount of entries in this cache. Public Public Final Methods int evictionCount () Returns the number of values that have been evicted. Public Finale V Get (K Key) Returns value to the key if it exists in the cache or can be created #create. If the value is returned, it moves to the queue chapter. This returns zero if the value is not cached and cannot be created. Public final int hitCount () Returns the number of times to receive (K) returned the value that was already present in the cache. Public final int maxSize () For caches that don't override the sizeof (K, V), this returns number of entries in the cache. For all other caches, this returns the maximum amount of entries in this cache. Public Finale Int MissCount () Returns Number get (k) returned zero or required a new value to be created. Public final V put (K key, V value) Cash value for the key. The value moves to the head of the queue. Returns V to the previous value displayed by the key. Public final int putCount () Returns the number of times put (K, V) has been called. Public Final V Delete (K Key) Removes the entry for the key if it exists. Returns V to the previous value displayed by the key. in size (int maxSize) sets the size of the cache. MaxSize int options: New maximum size. For caches that are not override by the size of TheOf (K, V), this returns the number of entries in the cache. For all other caches, this returns the amount of entries in this cache. The public final snapshot of the card returns a copy of the current cache content ordered from the least recent access to the latest access to the latest access. Typically, the toString method returns the line that textually represents that object. The result should be a short but informative presentation that is easy for a person to read. It is recommended that all subclasses override this method. The toString method for the class object returns a line consisting of the class name in which the object is a copy, a symbol on the 'q' sign, and an unsigned six-social representation of the object's hash code. In other words, this method returns a line equal to the value of getClass (.getName), the public void trimToSize (int maxSize) Remove the senior entries until the total number of remaining entries is at or below the requested size. MaxSize int options: the maximum cache size before returning. Maybe -1 to evict even 0-sized items. Protected V-key protection methods are called after a cache miss to calculate the value of the relevant key. Returns the calculated value or invalid if the value cannot be calculated. The default implementation returns zero. The method is called non-synchronized: other threads can access the cache while doing this method. If the key value exists in the cache when this method returns, the created value will be released with the recordRemoved (boolean, K, V, V) and discarded. This can happen when multiple threads request the same key at the same time (resulting in multiple values being created), or when one thread causes a bet (K, V) while the other creates a value for the same key. Protected Void EntranceRemoved (boolean evicted, Key K, V oldValue, V newValue) Is called for records that have been evicted or removed. This method is called when the value is removed to make space, is removed by a call to remove (K), or replaced by a call for (K, V). Implementing by default does nothing. The method is called non-synchronized: other zlt;/K, V'gt; access to the cache while this method is being implemented. Options evicted boolean: true if the record is removed to make the space false if the removal was caused by putting (K, V) or delete (K). key K oldValue v newValue V: a new value for the key if it exists. If not zero, this deletion was caused by putting (K, V) or get (K). Otherwise, it was caused by eviction or removal (K). The protected size of IntOf (K key, V value) returns the entry size for the key and the value in the user units. The default implementation returns 1, so size is the number of entries, and the maximum size is the maximum number of entries. The size of the record should not change while it is in the cache. Cache. disk lru cache example android