Manejo básico de record store RMS en aplicaciones móviles con J2ME

2008

Por: Rubén Dario Orozco
(drincast@hotmail.com)

Esta obra está bajo una licencia Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 Colombia de Creative Commons. Para ver una copia de esta licencia, visite http://creativecommons.org/licenses/by-nc-sa/2.5/co/ o envie una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Creative Commons License Deed

Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 Colombia

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

- **Reconocimiento**. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial**. No puede utilizar esta obra para fines comerciales.
- **Compartir bajo la misma licencia**. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia iéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Advertencia

Este resumen no es una licencia. Es simplemente una referencia práctica para entender el Texto Legal (la licencia completa) es un redactado inteligible por cualquiera de algunos de los términos clave de la licencia. Se trata de una interficie amigable del Texto Legal que hay debajo. Este resumen por si mismo no tiene valor legal, y su contenido no aparece en la auténtica licencia.

Creative Commons no es un bufete de abogados y no ofrece servicios legales. La distribución, la muestra, o el enlace de este Resumen no crea ningun vínculo abogado-cliente.

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.

Esto es un resumen fácilmente legible del texto legal (la licencia completa).

Almacenamiento de datos en aplicaciones móviles con J2ME

Primero que todo quiero decir que esta guía es muy básica, es una introducción al almacenamiento de datos en aplicaciones móviles. Además los ejemplos están basados en dos artículos de escritos por Eric Giguere. Para comenzar con esta guía veamos un poco de teoría.

En las aplicaciones móviles se pueden almacenar datos, estos datos son almacenados en lo que se llama Records, un Record o registro es un dato individual de cualquier tipo de dato (string, array, imagen, etc.), para utilizar esta capacidad entra el concepto de RMS que no es más que el objeto que nos provee el almacenamiento y asigna un identificador único, algo importante es que un record o registro no es lo mismo que los RMS.

Pero con los records solamente guardamos un dato de cualquier tipo, y deberías estar creando un record por cada dato, para solucionar esto o dar un poco más de capacidad se utilizan los Recod Stores, que en conceptos simples no es más que una colección de records.

Para utilizar Records Stores o records hay que tener en cuenta las capacidades del dispositivo, en el cual vamos a utilizarlos, así que ten cuidado al crear un Record Store que almacene muchos datos en un dispositivo con muy poca memoria.

Una aclaración antes de que la pase por alto los RMS son considerados como Base datos locales en una aplicación móvil, pero esto no tiene que ver nada con enlace a base datos como SQLPostgres. Otra cosa un poco importante es que un record store generado en una PDA o PAL tiene una extensión .pde (“no recuerdo muy bien la extención”), en cambio en un celular es un archivo plano que no tiene extensión y se almacena en el mismo directorio de la aplicación.

A nivel de API un record store es representado por una instancia de la clase `javax.microedition.rms.RecordStore`, y todas las clases para el manejo de RMS están definidas en `javax.microedition.rms`.

Por ser una guía rápida no creare una aplicación como tal pero si se escribirán pedazos de código que muestran como utilizar un record store.

Como descubrimos un RecorStore almacenado en un dispositivo.

De nuestra MIDlet podemos obtener una lista de todos los record stores que almacena la aplicación, utilizando esta función `RecordStore.listRecordStores()`, esta función retorna una lista de string que contiene el nombre de los record stores que existen en la MIDlet. El nombre de un record store, este nombre se compone de 1 a 32 caracteres unicode y debe se único dentro de una MIDlet.

Miremos un ejemplo:

```
Public void analizarTodosRMS(){
      String[] nombres = RecordStores.listRecordStores();

      for(int i=0; nombre != null && nombres.length; i++){
            System.out.println(name[i]);
      }
}
```

La salida se muestra por consola esta función imprime los nombre de los record store almacenados en la MIDlet.

Como Abrimos y Cerramos un record store

El metodo RecordStore.openRecordStore(), se utiliza para abrir y opcionalmente crear un record store, pues de lógica este método retorna un objeto RecordStore. El primer parámetro de entrada es el nombre del record store, y el segundo es un dato tipo boolean (true y false), para indicarle si creamos un record store, si este no existe.

Cuando termine con un record store, se debe cerrar con el llamado del método `RecordStore.closeRecordStore().`

Miremos el ejemplo:

```
public void analizar( String rsNombre ){
    RecordStore rs = null;

    try {
        rs = RecordStore.openRecordStore( rsNombre, false );
    } catch( RecordStoreException e ){
        //nombre de record store muy largo
        logger.exception( rsNombre, e );
    } finally {
        try {
            rs.closeRecordStore();
        } catch( RecordStoreException e ){
        }
    }
}
```

Hasta el momento todo fácil no, bueno ya sabemos como obtener todos los record stores de una MIDlet, abrir, cerrar, y crear un record store, ahora aprenderemos a adicionar y actualizar datos.

Adicionando un registro

Para adicionar un registro utilizamos el método `RecordStore.addRecord`(), este recibe 3 parámetros, el primero es el dato aguardar, el segundo el inicio de donde vamos a empezar a guardar (comienzo de offset), el tercero es el total de bytes a almacenar (final del offset), el método retorna un dato entero que es el identificador del registro en donde se almaceno los datos.

...

```
// creamos un record store
RecordStore rs = null;

try {
    rs = RecordStore.openRecordStore( "rsDatos", true );
} catch( RecordStoreException e ){
    // no se puede crear o abrir
}

byte[] datos = new byte[]{ 0, 1, 2, 3 };
int    registroID;

registroID = rs.addRecord( registro, 0, datos.length );
...
```

Sencillo, ahora actualicemos un registro.

Actualizar o modificar un registro

Para realizar esta operación utilizamos el método RecordStore.setRecord(), este método recibe 4 parámetros, el primero es el identificador del registro (el que retorna rs.addRecord()), el segundo parámetro es el objeto que se va aguardar, el tercero y cuarto, es el inicio y final de lo que vamos a guardar, continuando con el ejemplo anterior tenemos.

```
...
byte[] nuevoDato = new byte[]{ 0, 10, 20, 30 };

// remplaza el registro con valores 10 y 20
rs.setRecord( registroID, nuevoDato, 1, 2 );

//o podemos reemplazarlo por completo
rs.setRecord( registroID, nuevoDato, 0, nuevoDato.length);
...
```

Lectura de registros de un record store

Para leer de un record store utilizamos la función `RecordStore.getRecord`(), recibe como parámetro el identificador de registro que se va a leer y nos retornara un array de bytes.

```
byte[] dato = rs.getRecord( registroID );
```

Otra forma para cargar un registro seria esta:

```
byte[] dato = new byte[ rs.getRecordSize( registroID ) ];
rs.getRecord( registroID, datos, 0 );
```

La función `getRecordSize(int identificador)`, nos retorna el tamaño del array de bytes del registro especificado por el identificador, como la función getRecord esta sobre cargada, esta recibe tres parametros, el primero es el identificador de registro, el segundo es el array de bytes en donde se cargara la información del registro, y el tercero es el comienzo de donde se empezara a cargar el array.

Recorriendo varios registros en un record store

Para recorrer todos los registros de un record store, primero debemos obtener el número total de registros, esto lo hacemos por medio del método `getNextRecordID()`, el cual retorna un dato entero con un valor del total de registros almacenados en el record store. Luego debemos obtener el tamaño de un registro en especifico, esto lo hacemos con el método `getRecordSize(int id)`, el cual recibe como parámetro el identificador de un registro y retorna el tamaño del registro, y por ultimo obtenemos el registro cargándolo en un array de bytes, pues fácil, veámoslo en un ejemplo.

```
...
int numRegistrosID = rs.getNextRecordID();     //obtenemos el numero de registros
byte[] dato = null;

for( int id = 0; id < numRegistrosID; ++id ){//recorremos cada uno de los registros
    try {
        int size = rs.getRecordSize( id ); //obtenemos el tamaño de un registro

        if( dato == null || dato.length < size ){
            dato = new byte[ size ];
        }

        rs.getRecord( id, dato, 0 );      //obtenemos los datos del registro
    } catch( InvalidRecordIDException e ){
       //no hay record
    } catch( RecordStoreException e ){
        handleError( rs, id, e );         // call an error routine
    }
}
...
```

Borrando un Registro o un record store

Para eliminar un registro de un record store utilizamos el método `RecordStore.deleteRecord(int id)`, recibe como parámetro el identificador del registro a eliminar, el record debe existir si no se genera una excepción `InvalidRecordIDException`.

Para borrar un record store utilizamos `RecordStore.deleteRecordStore()`, el cual recibe como parámetro el nombre del record store a eliminar.

```
try {
    RecordStore.deleteRecordStore( "rsDatos" );
} catch( RecordStoreNotFoundException e ){
    // no se encuentra el record store
} catch( RecordStoreException e ){
    // algo paso
}
...
```

Fácil, ya podemos crear, recorrer, eliminar record stores y crear, recorrer, modificar y eliminar records, pero solo podemos almacenar datos tipo byte, pero si sabes bastante de conversión de datos te será fácil pasar a datos de diferentes tipos a un arreglo de bytes, por el momento veremos algún mapeo de datos básico.

Mapeo de tipos de datos

Pasando de Array de bytes a Streams

```
byte[] data = new byte[]{ 1, 2, 3 };
ByteArrayInputStream bin = new ByteArrayInputStream( data );

int b;

while( ( b = bin.read() ) != -1 ){
    System.out.println( b );
}

try {
    bin.close();
} catch( IOException e ){
    // never thrown in this case
}
...
```

El proceso inverso

```
...
ByteArrayOutputStream bout = new ByteArrayOutputStream();

bout.write( 1 );
bout.write( 2 );
bout.write( 3 );

byte[] data = bout.toByteArray();

for( int i = 0; i < data.length; ++i ){
    System.out.println( data[i] );
}

try {
    bout.close();
} catch( IOException e ){
    // never thrown in this case
}
...
```

Conversión de Datos primitivos

Con la clase DataInputStream podemos convertir stream a tipos de datos primitivos

```
...
InputStream entrada = ... // un stream de entrada
DataInputStream dEntrada = new DataInputStream(entrada);

try {
    int vendedorID   = dEntrada.readInt(); //leemos un entero
    String nombre  = dEntrada.readUTF(); //leemos un string
    String apellido = dEntrada.readUTF();
    long timestamp   = dEntrada.readLong(); //leemos un long

    din.close();
}
catch( IOException e ){
    // ocurrio un error
}
...
```

Con la clase DataOutPutStream podemos escribir string o datos primitivos

```
...
OutputStream salida = ... // an output stream
DataOutputStream dSalida = new DataOutputStream(salida);

try {
    dSalida.writeInt( 100 );
    dSalida.writeUTF( "Pedro" );
    dSalida.writeUTF( "Perez" );
    dSalida.writeLong( System.currentTimeMillis() );

    dSalida.close();
}
catch( IOException e ){
    // ocurrió un error
}
...
```

Guardando datos en el record store

Para almacenar datos de tipos primitivos, primero debemos escribir los datos en un Stream y luego convertir esto a un array de bytes.

```
...
RecordStore rs = ... // a record store
ByteArrayOutputStream arraySalida= new ByteArrayOutputStream();
DataOutputStream dSalida = new DataOutputStream(arraySalida);

try {
    dSalida.writeInt( 100 );
    dSalida.writeUTF( "Pedro" );
    dSalida.writeUTF( "Perez" );
    dSalida.writeLong( System.currentTimeMillis() );
    dSalida.close();

    byte[] datos = arraySalida.toByteArray();
    rs.addRecord( datos, 0, datos.length );
}
catch( RecordStoreException e ){
    // ocurrio un error en RMS
}
catch( IOException e ){
    // ocurrio un error de E/S
}
...
```

Leyendo datos de un record store

```
...
RecordStore rs = ... // un record store
int registroID = ... // un id de un registro
ByteArrayInputStream arrayEntrada;
DataInputStream dEntrada;

try {
    byte[] datos = rs.getRecord( registroID );

    arrayEntrada = new ByteArrayInputStream( datos );
    dEntrada = new DataInputStream(arrayEntrada);
```

```
    int id = dEntrada.readInt();
    String nombre = dEntrada.readUTF();
    String apellido = dEntrada.readUTF();
    long timestamp = dEntrada.readLong();

    dEntrada.close();

    ... // process data here
}
catch( RecordStoreException e ){
    // error en RMS
}
catch( IOException e ){
    // error de E/S
}
...
```

Creo que ya hemos visto lo más básico en el manejo de record store para dispositivos móviles, por el momento no es más espero que te sea de utilidad, lo poco que se a mostrado en esta guía, de pronto en uno próxima entrega si se desarrolle una pequeña aplicación en la cual se maneje record store.

De nuevo cualquier duda sobre esta guía la puedes enviar a correo (drincast@hotmail.com) y si estoy en posibilidades de respondértela ten por seguro que así será, hasta la próxima.

Referencias

- Eric Giguere, Databases and MIDP, Part 1: Understanding the Record Management System, February 2004
- Eric Giguere, Databases and MIDP, Part 2: Data Mapping, May 2004.