Almacenamiento Persistente

Juan Manuel Fernández Luna

Departamento de Ciencias de la Computación e Inteligencia Artificial

Programación de Dispositivos Móviles con J2ME

Septiembre/Octubre de 2006

Introducción

- MIDP ofrece un mecanismo para que los MIDlets almacenen datos de manera persistente y los recuperen posteriormente.

- Este mecanismo de almacenamiento persistente, llamado **Record Management System (RMS)**, se modela mediante una "base de datos" simple basada en registros.

Almacén de registros

- Un almacén de registros (record store) consiste en una colección de registros que persistirán a lo largo de múltiples invocaciones de un MIDlet.
- Los almacenes de registros se crean en zonas dependientes de la plataforma, que no están expuestos directamente a los MIDlets.
- El espacio de nombres del almacén de registro se controla en el ámbito del MIDlet Suite.
  - Los MIDlets dentro de una MIDlet suite tienen permitido crear múltiples almacenes y acceder a ellos. El resto no.

Almacén de registros

- Sólo se permite la manipulación de los almacenes de registros a las suites de MIDlets que los poseen.
- No se ofrece ningún mecanismo para compartir registros entre MIDlets que pertenezcan a suites diferentes.

Almacén de registros

- Los nombres de los almacenes de registros es sensible a las mayúsculas y puede consistir en cualquier combinación de hasta 32 caracteres Unicode.
- No se suministran operaciones de bloqueo.
- La implementación de los almacenes de registros asegura que todas las operaciones sobre los mismos son atómicas, síncronas y serializadas, por lo que no puede llegar a ocurrir que se corrompan con accesos múltiples.

Almacén de registros

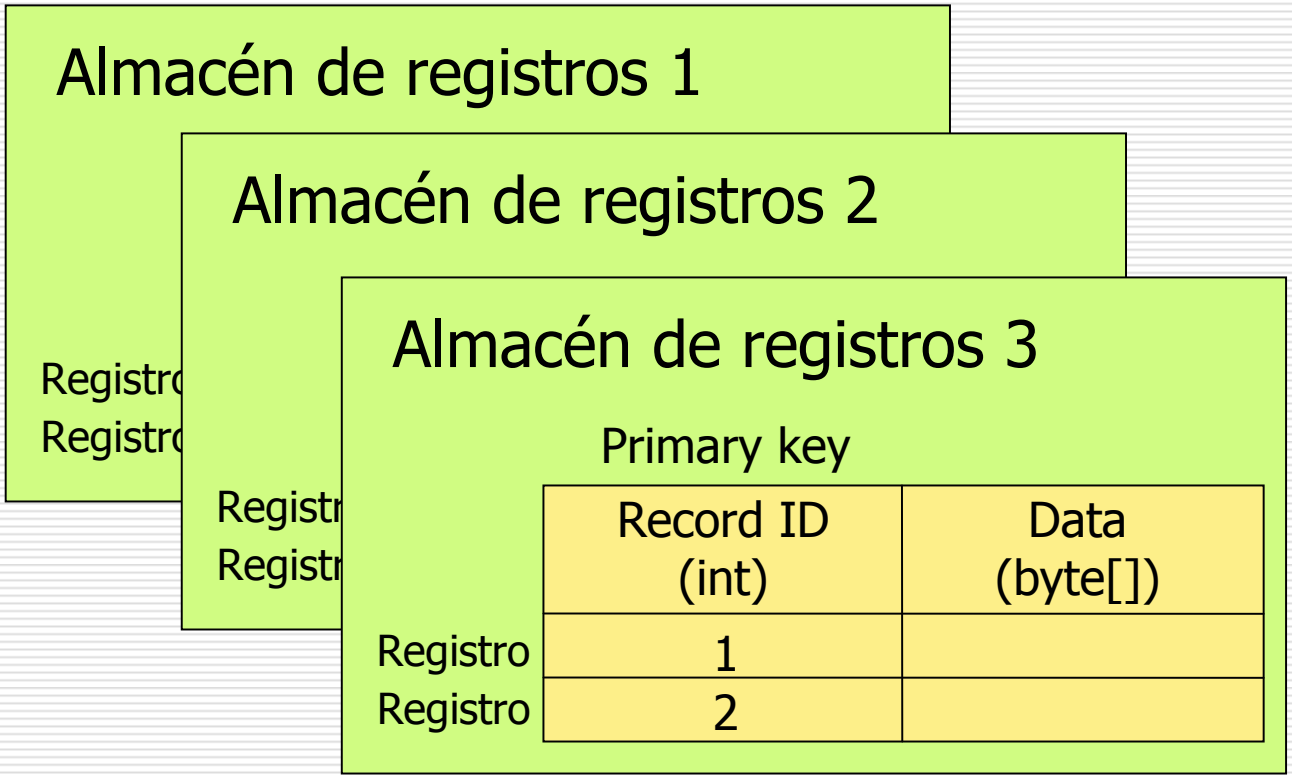
- El almacén guarda el día y hora de última modificación.
- Mantiene una versión (cada vez que hay un cambio se incrementa ésta).
- Esto es útil para sincronización de aplicaciones.
- Cuando una aplicación se elimina del dispositivo también se elimina los almacenes que tuviera.

Registros

- Registros: vectores de bytes.
- Se puede usar: *DataInputStream* y *DataOutputStream*, así como *ByteArrayInputStream* y *ByteArrayOutputStream* para empaquetar y desempaquetar diferentes tipos de datos en y desde un vector de bytes.

Registros

- Los registros de identifican unívocamente dentro del registro al que pertenecen por medio de su `recordId`, que es un valor entero.
- `RecordId` se usa como clave primaria del registro.
- El primer registro que se crea en un almacén tendrán un `recordId` igual a 1, y cada nuevo registro se incrementará en una unidad.
- Los MIDlets pueden crear sus índices utilizando la clase `RecordEnumeration`.


Almacén de registros 1
Almacén de registros 2
Almacén de registros 3
Primary key
Record ID (int)
Data (byte[])
Registro
Registro
1
2

Clases para la persistencia

- Paquete:
  - javax.microedition.rms
- Clases:
  - RecordStore, RecordEnumeration
- *Interfaces:*
  - RecordComparator, RecordFilter, RecordListener
- Excepciones:
  - RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException, InvalidRecordException, RecordStoreNotOpenException

Gestión del almacén

- openRecordStore()
- closeRecordStore()
- listRecordStore()
- deleteRecordStore()
- getVersion()
- getLastModified()

Gestión del almacén

- Apertura de un almacén:

  static RecordStore **openRecordStore**

  (recordStoreName, createIfNecessary)

  - Abre (y en su caso crea) un almacén de registros asociado con la correspondiente suite MIDlet.
  - Excepciones:
    - RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException

Gestión del almacén

- Cierre de un almacén:

  void closeRecordStore()

  - Excepciones:
    - RecordStoreException, RecordStoreNotOpenException
- Listado de almacenes:

  static** String[] **listRecordStores()

  - Devuelve un vector de nombres de almacenes de registros poseídos por el MIDlet suite.

Gestión del almacén

- Un almacén de registros debería cerrarse tan pronto como se acabe de usar, ya que consume recursos.
- El método closeRecordStore() realmente no cierra el almacén, sino que informa que el proceso o hilo ha dejado de usarlo.
- El almacén se cierra sólo cuando todos los procesos / hilos que lo utilizaban han llamado al método de cierre.

Gestión del almacén

- Borrado de un almacén:

  static void deleteRecordStore (recordStoreName)

  - Excepciones:
    - RecordStoreException, RecordStoreNotFoundException

- Versión (se incrementa en uno en cada modificación del almacén)

  int getVersion()

- Fecha de última modificación:

  long getLastModified()

Manipulación de registros

- Manipulación básica:
  - addRecord(), deleteRecord(), and getRecord(), setRecord()
- Información sobre un almacén:
  - getNumRecords() , getRecordSize()
- Información sobre un registro:
  - getSize(), getSizeAvailable()
- Enumeración:
  - getNextRecordID()

Manipulación de registros

- Añadir un registro:

  *int **addRecord**(byte[] data, int offset, int numBytes)*

  - Añade un nuevo registro al almacén.
  - Se devuelve el recorId del nuevo registro.
- Borrar un registro:

  *void **deleteRecord**(int recordId)*

  - Se borra el registro indicado. No se libera espacio, sino que se añade a la lista de registros libres para su posterior uso. No se utilizan los antiguos recordIds.

Manipulación de registros:

- Obtener un registro:

  *int **getRecord**(int recordId, byte[] buffer, int offset)*

  - Devuelve el dato almacenado en el registro dado.
  - Devuelve el número de bytes copiados en el buffer.

  *byte[] **getRecord**(int recordId)*

Manipulación de registros:

- Asignar valor a un registro:

*void **setRecord**(int recordId, byte[] newData, int offset,*

I int nBytes)

  - Asigna nuevos datos al registro ya existente.

Monitorización de cambios

- RecordListener
  - *recordAdded(recordStore, recordID)*
  - *recordChanged(recordStore, recordID)*
  - *recordDeleted(recordStore, recordID)*
- RecordStore
  - *addRecordListener(listener)*
  - *removeRecordListener(listener)*

Información de tamaño

- Número de registros:

  int ***getNumRecords()***

  - Devuelve el número de registros en el almacén.
- Tamaño del registro:

  int ***getRecordSize****(int recordId)*

  - Devuelve el tamaño (en bytes) de los datos del MIDlet disponible en el registro.

Información de tamaño.

- getSize

  int **getSize**()

  - Devuelve la cantidad de espacio, en bytes, que ocupa el registro.
- getSizeAvailable

  int **getSizeAvailable**()

  - Devuelve la cantidad de espacio adicional (en bytes) disponible para que el almacén crezca.

Enumeración

- getNextRecordID

  int **getNextRecordID**()

  - Devuelve el recordId del siguiente registro que será añadido al almacén.

Emumeración

- Interfaz que mantiene una secuencia de los recordids de los registros almacenados.
- Métodos:
  - hasNextElement(), hasPreviousElement()
  - nextRecord(), nextRecordId()
  - previousRecord(), previousRecordId()
  - numRecords()
  - …

Emumeración

```
try {
RecordEnumeration re =
    rs.enumerateRecords(null,null,false) ;
System.out.println("Hay " + re.numRecords()
                                + " en RecordStore") ;
while(re.hasNextElement()) {
        byte tmp[] = re.nextRecord() ;
        System.out.println(tmp[0] + " " + tmp[1]) ;
    }
} catch (Exception e) { …}
```

Filtrado de registros

- RecordFilter es un interfaz que se utiliza para seleccionar registros (para actualizarlos o eliminarlos), mediante algún criterio.
- Se debe implementar el siguiente método:

  boolean matches(byte[] candidate)

- Devuelve verdadero si el candidato se corresponde con el criterio.

Comparación de registros

- Interfaz que define un comparador de dos registros para determinar un orden relativo entre registros.
- Define un método para comparar.

  ***int compare**(byte[] rec1, byte[] rec2)*
- Devuelve:
  - PROCEEDS, FOLLOWS, EQUIVALENT

Comparación de registros

```
RecordComparator c = new AddressRecordComparator();
        // clase que implementa RecordComparator

if (c.compare(recordStore.getRecord(rec1),
                recordStore.getRecord(rec2)) ==
                RecordComparator.PRECEDES)

return rec1;
```

Fuentes

- http://developers.sun.com/techtopics/mobility/articles/databaserms