第 19 章

MIDP 数据库程序设计进阶

19.1 本章目的

本章将接续前一个章节，讨论更深入的 MIDP 数据库程序设计。包括监控记录仓储的变化、走访记录仓储的方法以及过滤器（Filter）与比较器（Comparator）对走访记录仓储时所带来的影响。

19.2 监控记录仓储的变化

记录管理系统提供了一个监视机制，让我们的程序可以随时得知数据库目前的情况，比方说是否有新的一笔记录加入记录仓储中、记录被删除或是记录被修改。这个监视机制使用 RecordListener 接口来达成。

RecordListener 采用 Multicast 机制，也就是说，同一时间可以有许多人注册监听记录仓储的状况，如图 19-1 所示：

虽然 RecordListener 属于 Multicast 的设计，但是同一个人只能注册一次。所以同一个参考传入 addRecordListener() 是没有意义的。当记录仓储被关闭后，所有注册的 RecordListener 都会被移除。我们随时都可以利用 RecordStore 类的 removeRecordListener() 方法来移除之前注册的 RecordListener。

RMSMonitorTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.* ;
public class RMSMonitorTest extends MIDlet
```

```
implements RecordListener
{
   private Display display;
   public RMSMonitorTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
      String dbname = "testdb" ;
      Form f = new Form("RS Test") ;
      RecordStore rs = RMSUtil.openRSAnyway(dbname) ;
      rs.addRecordListener(this) ;
      if( rs == null )
      {
         //开启失败停止 MIDlet
         f.append("Table open fail") ;
      }else
      {
         try
         {
            byte []data = new byte[2] ;
            data[0] = 15 ;
            data[1] = 16 ;
            int id = rs.addRecord(data,0,data.length) ;
            data[0] = 25 ;
            data[1] = 26 ;
            rs.setRecord(id,data,0,data.length) ;
            rs.deleteRecord(id) ;
            rs.addRecord(data,0,data.length) ;
            rs.closeRecordStore() ;
            RMSUtil.deleteRS(dbname) ;
         }catch(Exception e)
         {
         }
      }
      display.setCurrent(f) ;
   }
   public void recordAdded(RecordStore rs,int recordid)
   {
      try
      {
         byte []tmp = rs.getRecord(recordid) ;
         System.out.println("Record " + recordid + "is added.") ;
         System.out.println("Content=" + tmp[0] + tmp[1]) ;
      }catch(Exception e)
      {
         System.out.println(e.getMessage()) ;
      }
   }
   public void recordChanged(RecordStore rs,int recordid)
   {
      try
```

```
        {
            byte []tmp = rs.getRecord(recordid) ;
            System.out.println(
                "Record " + recordid + "is changed.") ;
            System.out.println("Content=" + tmp[0] + tmp[1]) ;
        }catch(Exception e)
        {
            System.out.println(e.getMessage()) ;
        }
    }
    public void recordDeleted(RecordStore rs,int recordid)
    {
        try
        {
            //请勿在此使用 byte []tmp = rs.getRecord(recordid) ;
            //因为此时该笔记录已被删除
            System.out.println(
                "Record " + recordid + "is deleted.") ;
        }catch(Exception e)
        {
            System.out.println(e.getMessage()) ;
        }
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

执行结果:

```
Record lis added.
Content=1516
Record lis changed.
Content=2526
Record lis deleted.
Record 2is added.
Content=2526
```

RecordListener 之中的回调函数**都是在记录仓储完成动作之后才被调用，而非动作完成之前**，因此最需要留意的地方在于 recordDeleted()函数。因为加入数据或修改数据之后，该笔数据依旧存在，所以我们在 recordAdded()或 recordChanged()里头可以利用 getRecord()取出该笔记录。但是在 recordDeleted()函数之中去存取属于传入 ID 的数据的话，因为回调函数是在记录被删除之后才被调用，因此当时该笔记录早已不复在，因此，请勿在 recordDeleted()函数里头使用 getRecord()取得刚被删除的数据，这将引发 InvalidRecordIDException。

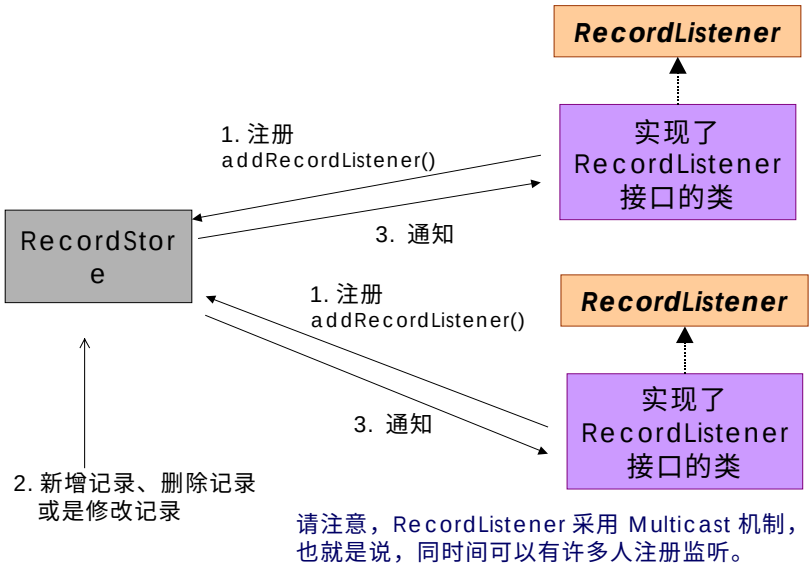

图 19-1　用 RecordListener 实现监视机制

19.3　走访记录仓储

到目前为止我们可以大致上了解，不管新增、修改或删除记录仓储的数据，都需要指定 Record ID，没有 Record ID，我们根本无法存取记录仓储之中的数据。假设我们的程序是在关闭后重新激活，如何得知之前所存入数据的 Record ID 呢? 因此，我们需要一些机制，让我们可以在没有 Record ID 的情况下就可以存取数据库仓储，这就是所谓的“走访”。

大部分的人听到走访，在脑中第一个闪过的想法大概如下：

```
for(int i = 1 ; i < rs.getNextRecordID() ; i++)
{
    byte []data = rs.getRecord(i) ;
    ... ...
}
```

这种走访记录仓储的方式有潜在的问题。我们知道 Record ID 是在记录一写进记录仓储之后就固定的，虽然使用 getNextRecordID()可以保证存取到所有可能出现的 Record ID，但是却无法保证所有的 Record ID 目前都还在数据仓储之中(可能早已被 deleteRecord()函数所删除)，所以上述方法并不恰当。

MIDP 规格中提供的另外一种更方便、安全的记录仓储走访方式——RecordEnumeration 接口。RecordEnumeration 的内部结构如图 19-2 所示。

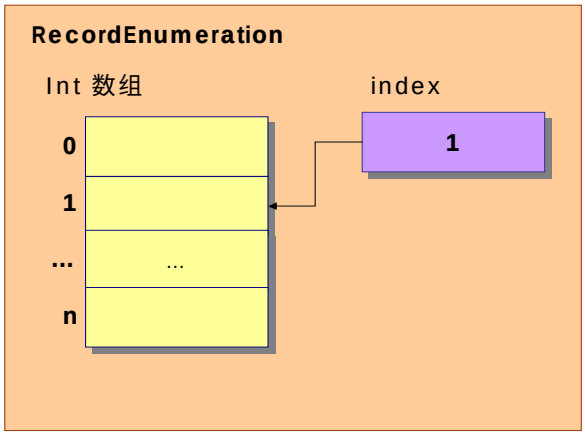

图 19-2 RecordEnumeration 的内部结构

其中，int 数组储存了某个时期记录仓储之中部分 Record ID 的内容，并且以特定顺序排序。int 数组会根据过滤器来决定哪些 Record ID 该放进来（如果没有过滤器，记录仓储的所有 Record ID 都会被放入），而 int 数组内容的排序方式则由比较器来决定（如果没有比较器，内容的顺序就会依不同的装置而定，规格里没有定义）。Index 变量则用来记录目前走访到 int 数组的哪个元素。RecordEnumeration 类提供的 numRecords()方法可以让我们知道 int 数组有多少元素。

RecordEnumeration 就像是一个记录仓储在某个时期可使用的 Record ID 的集合所构成的集合，而且会依我们指定的方式排列。只要使用 RecordStore 类的 enumerateRecords()方法，就可以产生调用 enumerateRecords()时某个记录仓储的 **Record ID 字段快照（snapshot）。**

由于 RecordEnumeration 内部并没有存放任何记录仓储数据的副本，因此往后我们用 RecordEnumeration 存取数据时，RecordEnumeration 底层还是去抓取 RecordStore 之中的数据，也就是真正记录仓储之中所存放的数据（如图 19-3 所示）。

enumerateRecords()方法需要三个参数。第一个参数需要 RecordFilter 对象（过滤器），RecordFilter 可以用来过滤记录仓储之中的内容。第二个参数为一个 RecordComparator 对象（比较器），比较器可以用来决定所传回之数据的顺序。如果第一个参数和第二个参数都是 null，那么规格中没有定义传回结果的顺序如何，但是保证用户可以走访所有的数据，而且这是最有效率的走访方式（因为没有过滤器与比较器运算时所带来的额外负担）。如果只有给定第一个参数，第二个参数却给 null，那么规格中没有定义传回结果的顺序如何。

enumerateRecords()方法的第三个参数，决定产生 RecordEnumeration 之后，这个 RecordEnumeration 与 RecordStore 之间的关系，**如果传入 false，那么不管稍后记录仓储的内容如何变动（有新增的数据，或是某些 ID 的记录被删除了），都与此 RecordEnumeration 无关**，使用 false 是最有效率的用法，但是很可能会走访到早已不在记录仓储之中的数据，也无法走访到之后才更新的数据。**如果传入 true，那么 RecordEnumeration 会自动加入一个**

RecordListener 到 RecordStore 之中，一旦记录仓储的内容变动了，RecordEnumeration 的内容也会随着变动，使用 true 是最没效率的方式，但是可以和记录仓储之中的数据保持同步。

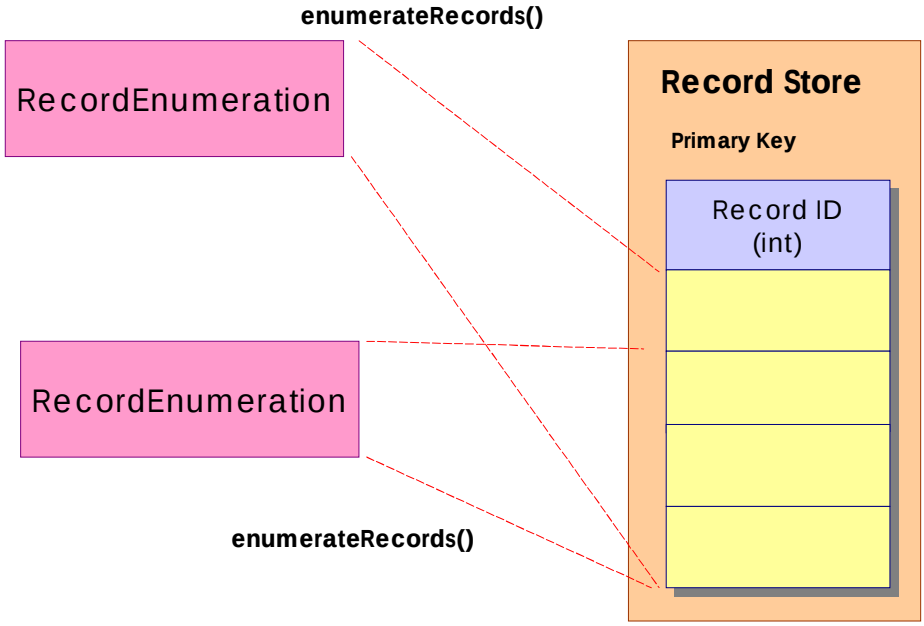

图 19-3　RecordEnumeration 存取数据

我们可以随时利用 RecordEnumeration 的 isKeepUpdated()观察 RecordEnumeration 是否会与后端记录仓储保持同步。也可以使用 keepUpdated()来设定 RecordEnumeration 是否要与后端记录仓储保持同步。与后端记录仓储保持同步会造成执行效能的下降，如果我们不希望效能下降，又希望能够在使用 RecordEnumeration 时，其内部的 int 数组存放的是后端记录仓储的最新内容，那么我们可以调用 RecordEnumeration 的 rebuild()方法来重建 RecordEnumeration 内部的 int 数组。**使用 rebuild()方法是一种比较折衷的方式，但是效能好多了**。

如果 RecordEnumeration 一开始处于不跟记录仓储同步的状态，后来我们又调用 keepUpdated(true)将它改成与记录仓储同步的话，系统会自动帮我们加入记录仓储的监视器（RecordListener），同时自动帮我们调用 rebuild()。

一旦 RecordStore 被关闭，RecordEnumeration 也会成为无效状态，调用 RecordEnumeration 任何方法都会产生 RecordStoreNotOpenException 或者传回 false。即使稍后 RecordStore 重新被开启，RecordEnumeration 一样无效，除非重新建立 RecordEnumeration。

使用完 RecordEnumeration 之后，请记得调用其 destroy()方法以释放 RecordEnumeration 内部所占用的系统资源。

19.4 RecordEnumeration 的使用方式

一旦我们取得 RecordEnumeration 之后，就可以通过 RecordEnumeration 内部的 int 数组与 index 变量帮我们走访记录仓储，并取得储存于记录仓储中的数据。

RecordEnumeration 的使用概念是一个双向环状数组（如图 19-4 所示）。

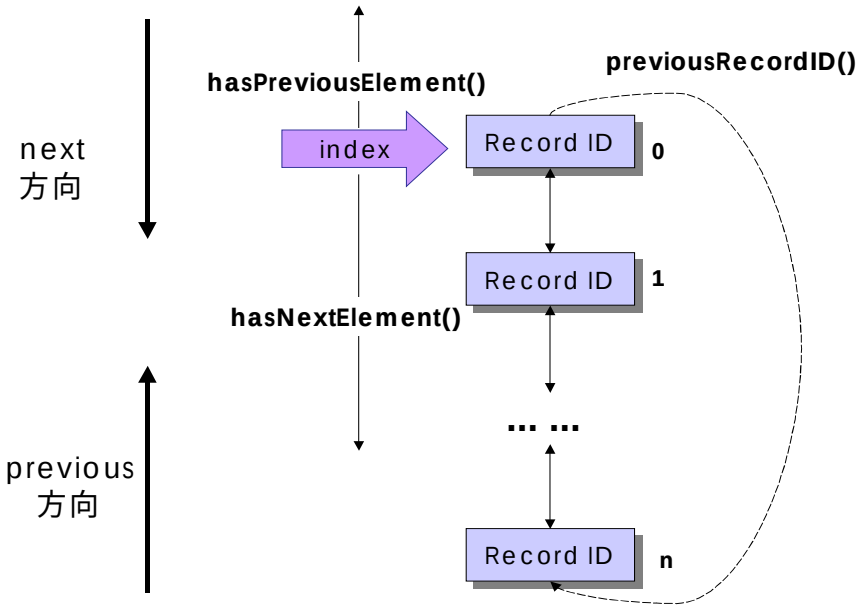

图 19-4 RecordEnumeration 的使用原理

RecordEnumeration 建立之初，index 变量内容为－1。任何时后都可以使用 RecordEnumeration 类的 reset()方法，将 index 变量的内容设定成 RecordEnumeration 建立之初的状态。

hashPreviousElement()可以探知 previous 方向是否有元素，如果目前 index 变量指向第一个元素（索引值 0），hashPreviousElement()就会传回 false。hashNextElement()可以探知 next 方向是否有元素，如果是目前 index 变量指向数组最后一个元素（索引值＝int 数组长度－1），hashNextElement()就会传回 false。

要改变 index 所指向的索引值，使它往 next 方向移动，请调用 nextRecordID()方法。如果 RecordEnumeration 建立之后，第一个调用的是 nextRecordID()，那么 index 变量的值就是 0，也就是 int 数组的第一个元素。接着每调用一次 nextRecordID()，index 变量的值就会加 1，直到 index 指向 int 数组的最后一个元素为止，如果继续调用 nextRecordID()，就会引发 InvalidRecordIDException。nextRecordID()会传回 index 变量改变之后，所指向的 int 数组元素之内容。所以传回值是记录仓储之中某一笔记录的 Record ID。

要改变 index 所指向的索引值，使它往 previous 方向移动，请调用 previousRecordID()方法。如果 RecordEnumeration 建立之后，第一个调用的是 previousRecordID()，那么 index 变量的值就是 int 数组的最后一个元素之索引值。接着每调用一次 previousRecordID()，index 变量的值就会减 1，直到 index 指向 int 数组的第一个元素为止，如果继续调用 previousRecordID()，就会引发 InvalidRecordIDException 。previousRecordID()会传回 index 变量改变之后，所指向的 int 数组元素之内容。所以传回值是记录仓储之中某一笔记录的 Record ID。

至于当我们调用 nextRecord()时，其内部会先调用 nextRecordID()传回某个 Record ID，然后再利用 RecordStore 的 getRecord()取得该笔数据的内容，并传回给调用者；当我们调用 previousRecord()时，其内部会先调用 previousRecordID()传回某个 Record ID，然后再利用 RecordStore 的 getRecord()取得该笔数据的内容，并传回给调用者。

RecordEnumeration 的用法如以下范例：

```
public void travelRS(RecordStore rs)
{
   try
   {
      RecordEnumeration re =
         rs.enumerateRecords(null,null,false) ;
      System.out.println("There are " +
         re.numRecords() + " in RecordStore") ;
      for(int i = 0 ; i < re.numRecords() ; i++)
      {
         int id = re.nextRecordId() ;
         byte tmp[] = rs.getRecord(id) ;
         //处理数据
      }
   }
   catch(Exception e)
   {
   }
}
```

或者

```
public void travelRS(RecordStore rs)
{
   try
   {
      RecordEnumeration re = rs.enumerateRecords(null,null,false) ;
```

```
        System.out.println("There are " + re.numRecords() +
            " in RecordStore") ;
        while(re.hasNextElement())
        {
            byte tmp[] = re.nextRecord() ;
            //处理数据
        }
    }
    catch(Exception e)
    {
    }
}
```

完整的使用范例如下：

RecordEnumerationTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.rms.* ;
public class RecordEnumerationTest extends MIDlet
{
    private Display display;
    public RecordEnumerationTest()
    {
        display = Display.getDisplay(this);
    }
    public void startApp()
    {
        String dbname = "testdb" ;
        Form f = new Form("RS Test") ;
        RecordStore rs = RMSUtil.openRSAnyway(dbname) ;
        if( rs == null )
        {
            //开启失败停止 MIDlet
            f.append("Table open fail") ;
        }else
        {
            try
            {
                byte []data = new byte[2] ;
                data[0] = 15 ;
                data[1] = 16 ;
                rs.addRecord(data,0,data.length) ;
                data[0] = 25 ;
                data[1] = 26 ;
                rs.addRecord(data,0,data.length) ;
                data[0] = 35 ;
                data[1] = 36 ;
                rs.addRecord(data,0,data.length) ;
                travelRS(rs) ;
```

```
            rs.closeRecordStore() ;
            RMSUtil.deleteRS(dbname) ;
         }catch(Exception e)
         {
         }
      }
      display.setCurrent(f) ;
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }
   public void travelRS(RecordStore rs)
   {
      try
      {
         RecordEnumeration re =
            rs.enumerateRecords(null,null,false) ;
         System.out.println("There are " +
            re.numRecords() + " in RecordStore") ;
         while(re.hasNextElement())
         {
            byte tmp[] = re.nextRecord() ;
            System.out.println(tmp[0] + " " + tmp[1]) ;
         }
      }
      catch(Exception e)
      {
      }
   }
}
```

执行结果：

```
There are 3 in RecordStore
35 36
25 26
15 16
```

19.5　结束语

本章介绍如何监控记录仓储的变化，以及利用 RecordEnumeration 来走访记录仓储的方法，RecordEnumeration 会受到过滤器（Filter）与比较器（Comparator）的影响；利用过滤器，我们可以筛选来自于底层记录仓储的内容；而利用比较器，我们可以对 RecordEnumeration 的内容排序。