Processing Large Search Result Sets in Java Internet Applications

NYOUG Meeting
January, 2002

Claudio Fratarcangeli
Adept Technology Inc.
claudiof@computer.org
Internet Search Requirements

- There can be a large number of hits (1000's) matching criteria
- Results are displayed 1 page at a time
- PREVIOUS and NEXT Buttons to scroll through pages
- A large number of users may be doing queries at same time
Functional Design Goals

- Return first page as quickly as possible
- PREVIOUS page and NEXT page must also be fast

Performance Goals

- Keep resource load on system to a minimum
- Only retrieve as much data as necessary to satisfy user request

Architecture Goals

- Keep presentation layer separate from data access layer
- Provide Search/Retrieval Interface that encapsulates implementation of Search/Retrieval Layer
Current Alternatives

EJB Finder Method
- Inefficient
- Resource Consumptive

Fetch All Hits Into a Collection Before Displaying First Page
- Slow to show First Page
- Very Resource Intensive (Processing and Memory)
- Users typically only browse through a small number of pages

Re-execute Query for Each Page of Display
- Too slow for expensive queries
- Very resource intensive

Fetch several pages of Hits/ Re-execute Query When More Needed
- Efficient if user only browses first few pages
- Can be expensive if user chooses to scroll through many pages
Internet Search Solution: Architecture

Provide List Handler Interface that Encapsulates:

- Search/Retrieval Implementation
- Data Access Implementation

Interface Must Support a Large Variety of Underlying Implementations

- Retrieval of Data from a Database using JDBC
- Retrieval of Data from non-database datastores
Internet Search Solution: Performance

Provide Efficient Implementation of Generic List Handler Interface

- Handles efficient retrieval of data from a database using JDBC
List Handler Interfaces

DataListHandler

- Handles interaction with client
- Execute search
- Return list of search results

DataList

- Represents list of objects retrieved by search
- Does not extend Java List interface to allow greater flexibility in underlying implementations

DataListIterator

- Traverse and access items in list
**List Handler Classes**

DataListHandlerImpl implements DataListHandler

DataListImpl implements DataList

- Represents list of all objects retrieved by search

DataListChunk implements DataList

- Represents subset of objects in DataListImpl
  - Returned to client to satisfy request to display a page worth of hits

DataListIteratorImpl implements DataListIterator
public interface DataList {

    public DataListIterator iterator() throws DataListException;

    public Object get(int index, Object item) throws DataListException, IndexOutOfBoundsException;

    public boolean isEmpty() throws DataListException;

    /* Release resources */
    public void close() throws DataListException;
}
Interfaces: DataListIterator

```java
public interface DataListIterator {

    public boolean hasNext() throws DataListException;

    public Object next(Object obj) throws NoSuchElementException, DataListException;

}
```
public interface DataListHandler {

    public DataList getListChunk(int startIndex, int count) throws DataListException;

    public boolean elementExists(int index) throws DataListException;

    /* Release resources */
    public void close() throws DataListException;

}
Class: **DataListImpl** implements DataList

- Implements a DataList that represents the entire collection of hits that satisfy a user search request

Class: **DataListChunk** implements DataList

- Implements a DataList that represents a subset of DataListImpl that is returned to display a page worth of hits

- Returned by DataListHandler.getListChunk method
Class: DataListHandlerImpl implements DataListHandler

• Executes user search request

• Creates a DataListImpl instance

• Stores search results in DataListImpl instance

• Satisfies getListChunk method request by creating a DataListChunk instance that represents a subset of DataListImpl and returning it
Class: DataListIteratorImpl
implments DataListIterator

- Interacts with DataList instance to traverse and access the items in the DataList Collection
Data List Handler Sequence Diagram

Client → aDataListHandler:
- executeSearch()

aDataListHandler → aDataListImpl:
- getListChunk()

aDataListImpl → aDataListChunk:
- iterator()

aDataListChunk → aDataListIteratorImpl:
- create()
  - hasNext()
  - next()

aDataListIteratorImpl → aDataListImpl:
- create()
  - create()
  - get()
  - get()
Strategy to Achieve Efficient Retrieval of Large Search Sets In RDBMS

Keep database connection open across requests

- Other strategies open and close a connection for each request

Fetch only as many rows as necessary to return a page of data to client

- Make use of Scrollable ResultSets in JDBC 2.0 Spec
JDBC 2.0: Scrollable ResultSet

New Option When Creating Statement Objects

• ResultSet Type
JDBC 2.0: Result Set Type

TYPE_FORWARD_ONLY (Default, JDBC 1.0 behavior)
- ResultSet is not scrollable
- Fetch forward only

TYPE_SCROLL_INSENSITIVE
- ResultSet is scrollable
- Fetch forward and backward
- Position to absolute or relative row in ResultSet

TYPE_SCROLL_SENSITIVE
- ResultSet is scrollable like TYPE_SCROLL_INSENSITIVE
- Rows changed and committed by other users visible as you scroll
JDBC 2.0 New ResultSet Cursor Positioning Methods

- public void previous()
- public boolean absolute(int index) – Positions to an absolute row number
- public void beforeFirst()
- public void afterLast
- public void first()
- public void last()
JDBC 2.0: New ResultSet Cursor
Informational Methods

- public int getRow() – Returns current row number
- public boolean isBeforeFirst()
- public boolean isAfterLast()
- public boolean isFirst()
- public boolean isLast()
Statement stmt = conn.prepareStatement(
    "select id, descr from product " +
    "where descr like '%'||upper(?)||'%' order by id",
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);

stmt.setString(1,keywords);
ResultSet rs = stmt.executeQuery();

boolean success = rs.absolute(10); // position to 10th row
String id = rs.getInt(1);

rs.previous(); // position to previous row (9th row)
id = rs.getInt(1);
rs.last(); // position to last row
Oracle Implementation of a Scrollable ResultSet

- Rows are fetched from the database in the forward direction only

- As rows are fetched they are stored in a client-side memory cache maintained by the JDBC driver

- Requests to scroll to a row that has already been fetched from database is satisfied by accessing row in local memory cache.

- Positioning to rows already fetched is very efficient
Strategy: Efficient Implementation of DataList Interfaces

- Keep Database Connections open across Http Requests
- Use JDBC 2.0: Scrollable ResultSets
- Fetch rows only one page at a time
DataList Interface Implementation

• Do not provide direct implementation of DataList Interfaces

• Provide another set of Interfaces that extend DataList Interfaces

• These interfaces are used for DataList implementations that generate search result sets from database queries using JDBC

• Other implementations of DataList might access non-database data stores
JDBC Database Access DataList Interfaces

• ResultSetDataList extends DataList

• ResultSetRowMapper
  – No counterpart in DataList interfaces

JDBC Database Access DataList Classes

• ResultSetDataListImpl (DataListImpl) implements ResultSetDataList

• ResultSetDataListChunk (DataListChunk) implements ResultSetDataList

• ResultSetIterator (DataListIteratorImpl) implements DataListIterator
public interface ResultSetDataList extends DataList {

    public boolean hasNext() throws SQLException;

    public void beforeFirst() throws SQLException;

    public boolean absolute(int index) throws SQLException;

    public boolean elementExists(int index) throws SQLException;

}
Class: ResultSetDataListImpl implements ResultSetDataList (extends DataList)

- Implements a DataList as a virtual collection that represents the entire result set of a client query
- Contains a scrollable JDBC ResultSet object
- The underlying scrollable JDBC ResultSet is the real collection (also implemented as virtual)
- Implementation of scrollable ResultSet defers fetching of rows until cursor is positioned on the row
- Overhead of fetching all hits up front is avoided
Class: `ResultSetDataListChunk` implements `ResultSetDataList` (extends `DataList`)

- Implements a `DataList` as a virtual collection that represents a subset of the elements in another `ResultSetDataList(DataList)`

- Represents a subset of a `ResultSetDataList(DataList)` that would be retrieved by `DataListHandler.getListChunk()` to display a page worth of results

- It is a virtual collection that is implemented as a window on another `ResultSetDataList`

- Because it is simply a window on another collection it is very efficient with low creation overhead
Class: ResultSetIterator implements DataListIterator

- Implementation of DataListIterator that is specialized to iterate over ResultSetDataList's
Clients of ResultSetDataListImpl and ResultSetDataListChunk

- Clients have knowledge only of DataList's
- Clients have no knowledge of ResultSetDataList's
- From client perspective they behave as if they were real collections
Standard ListIterator.next Method

• Method in the standard Java ListIterator interface is:
  – public Object next()

• Returns next Object in collection

• Implies that there is a real underlying collection of objects
DataListIterator.next Method

• Method Signature:
  - public Object next(Object obj)

• Copies state of next logical object in collection to passed in object

• Returns same object instance to caller

• Supports underlying implementations of DataList in which there does not exist a physical collection of objects

• A virtual collection of objects is possible

• Object is materialized at the time of next method call

• Empty object instance is passed in as input to avoid overhead of creating a new Object instance with every next call

• Input argument is an object of the same Class as the items in the collection
DataListIterator.next Method

- Requires a way to map data values of an item instance to the instance variables of the passed in object argument

- Real class of object is not known

- ResultSetIterator uses a ResultSetRowMapper object

- ResultSetRowMapper object maps item data values to instance variables of passed in object
Interface: ResultSetRowMapper

- Provides a way to populate the contents of an Object passed to ResultSetIterator.next method

  ```java
  public interface ResultSetRowMapper {
      public Object mapRow(ResultSet rs, Object item)
      throws SQLException;
  }
  ```

- mapRow method accesses a JDBC ResultSet object

- It gets the field values of the current row of the JDBC ResultSet and puts them in the passed in Object
Example: ResultSetRowMapper Class

```java
public class ProductRowMapper implements ResultSetRowMapper {

    public ProductRowMapper() {
    }

    public Object mapRow(ResultSet rs, Object itemObj) throws SQLException {
        ProductDataItem item = (ProductDataItem)itemObj;
        item.id = rs.getInt("ID");
        item.descr = rs.getString("DESCR");
        return item;
    }
}
```
ResultSetRowMapper Interface

• Items of collection are not physically stored in a collection

• Items of collection are actually rows of JDBC Result Set

• ResultSet row field values are retrieved and placed into instance of Object at the time of DataListIterator.next call

• Avoids overhead of creating a new Object instance for each item in collection
**DataList: Implementation Tier**

**Servlet Tier**

- ResultSetDataListImpl class resides in Servlet Tier
- Client directly accesses ResultSetDataListImpl through DataList interface
- Data Access Object resides in Servlet Tier

**EJB Tier**

- ResultSetDataListImpl class resides in EJB Tier
- DataAccess Object resides in EJB Tier
- EJB directly access ResultSetDataListImpl through DataList interface
Servlet Tier: Implementation Issues

- JDBC Connection and ResultSet is stored in ResultSetDataListImpl Class object
- ResultSetDataListImpl object must be preserved across Http requests
- Implies that JDBC Connection is maintained across requests
- If a large amount of time lapses between requests, JDBC Connection may remain open for an extended period of time
- Resources wasted with keeping JDBC Connections open across requests
**Servlet Tier: Where do we store ResultSetDataListImpl or JDBC Connection across requests**

**HttpServletRequest object**

- JDBC Connection closed when session expires
- Can be costly if session expiration time is long

**ServletContext object**

- JDBC Connections can be stored in a hash table
- Batch process required to periodically inspect cache of JDBC Connections
- Close connections that have not been accessed in a while
- Cleanup of open JDBC Connection not tied to session expiration.
- More timely cleanup of JDBC Connections
Storing JDBC Connection in ServletContext object

What happens when a user attempts to go to next page after JDBC Connection has expired?

• Option 1:
  – Generate error telling user to requery

• Option 2:
  – Automatically reestablish connection and reissue query
  – Position cursor to page that was active when user made last request
  – More user friendly option
EJB Tier: How do we preserve ResultSetDataListImpl or JDBC Connection object across requests

- Stateful Session EJB preserves all state across requests

- Options for storing reference to stateful EJB in Servlet tier
  - HttpSession Object
    - EJB closed when session expires
  - ServletContext Object
    - EJB stored in a hash table
    - Batch process inspects hash table periodically to close idle EJB's
    - More timely cleanup of idle EJB's
Minimize Impact of Long Lived Open Connections

• Utilize
  – Shared Server
  – Connection Pooling
  – Session Multiplexing

• Oracle 9i implementation of these features is very efficient

• Overhead of a connection is minimal

• Capable of handling 1000's of open connections
public class Client1 {

    ProductDataItem item = new ProductDataItem();
    ProductHandler handler; // implements DataListHandler

    public boolean doSearch(String keywords)
        throws ProductHandlerException, DataListException {
        handler = new ProductHandler();
        handler.executeQuery(keywords);
        return handler.elementExists(0);
    }

    ...
}
public DisplayStatus displayPage(int startIndex, int count) throws DataListException {
    DataList dl = handler.getListChunk(startIndex, count);
    DataListIterator iterator = dl.iterator();

    int i = startIndex;
    while (iterator.hasNext()) {
        item = (ProductDataItem)iterator.next(item);
        i++;
    }

    boolean moreBefore;
    if (startIndex == 0)
        moreBefore = false;
    else
        moreBefore = handler.elementExists(startIndex-1);
    boolean moreAfter = handler.elementExists(i);

    // Return object which indicates whether more before or more after
    return new DisplayStatus(moreBefore, moreAfter);
}
Methods of Client Class: Continued

... 

/**
 * Closes DataListHandler class to release resources which includes Database Connection
 */

public void cleanup() throws DataListException {
    handler.close();
}
}
**ProductHandler Class**

```java
public class ProductHandler implements DataListHandler {

    private ProductCatalogDAO productCatalog;
    ResultSetDataList productDataList;

    public ProductHandler() throws ... {
        try {
            productCatalog = ProductCatalogDAOFactory.createInstance();
        } catch ... // handle exception
    }

    // Execute query.
    // DAO returns a ResultSetDataList representing search results
    public void executeQuery(String keywords) throws ..{
        try {
            productDataList = productCatalog.executeQuery(keywords);
        } catch ... // handle exception
    }

    ...
```
// Get subset of elements in result set
public DataList getListChunk(int startIndex, int count) throws DataListException {
    return new ResultSetDataListChunk(
        productDataList, startIndex, count);
}

public boolean elementExists(int index) ... {
    try {
        return productDataList.elementExists(index);
    } catch ... // handle exception
}

public void close() throws ... {
    productDataList.close();
}
```java
public class ResultSetDataListImpl implements ResultSetDataList {

    private ResultSetRowMapper rowMapper;
    private Connection conn;
    private Statement stmt;
    private ResultSet rs;

    public ResultSetDataListImpl(ResultSetRowMapper rowMapper,
                                 Connection conn, Statement stmt, ResultSet rs) {
        ...
        // Set instance variables
    }

    public DataListIterator iterator() throws DataListException {
        try {
            return new ResultSetDataListIterator(this);
        } catch ... // Handle exception
    }

    public void close() throws DataListException {
        // close ResultSet, Statement, and Connection
    }
```
ResultSetDataListImpl Class Methods (Continued)

// param index Range is 0 .. num_elements.
// Result set index ranges from 1 ..
public Object get(int index, Object item) throws DataListException, ...
{
    try {
        if (rs.absolute(index+1)) {
            if (rs.getRow() != index + 1)
                throw new IndexOutOfBoundsException();
            return rowMapper.mapRow(rs, item);
        } else
            throw new IndexOutOfBoundsException();
    } catch ... // Handle exception
}

public boolean hasNext() throws SQLException {
    return !rs.isLast() && (rs.getRow() != 0 || rs.isBeforeFirst());
}

public void beforeFirst() throws SQLException {
    rs.beforeFirst();
}
**ResultSetDataListImpl Class Methods (Continued)**

```java
/**
 * @param index Range is 0 .. num_elements.
 * Result set index ranges from 1 ..
 */
public boolean absolute(int index) throws SQLException {
    return rs.absolute(index + 1);
}

public boolean isEmpty() throws DataListException {
    try {
        return !rs.isBeforeFirst() &&
               !rs.isAfterLast() &&
               rs.getRow() == 0;
    } catch ... // Handle exception
    // Handle exception
}
```
public boolean elementExists(int index) throws SQLException {

    boolean beforeFirst = rs.isBeforeFirst();
    boolean afterLast = rs.isAfterLast();
    int currIndex = rs.getRow();
    boolean exists = rs.absolute(index + 1);
    if (beforeFirst)
        rs.beforeFirst();
    else if (afterLast)
        rs.afterLast();
    else if (currIndex != 0)
        rs.absolute(currIndex);
    return exists;
}