Value List Handler

Context

The client requires a list of items from the service for presentation. The number of items in the list is unknown and can be quite large in many instances.

Problem

Most J2EE applications have a search and query requirement to search and list certain data. In some cases, such a search and query operation could yield results that can be quite large. It is impractical to return the full result set when the client's requirements are to traverse the results, rather than process the complete set. Typically, a client uses the results of a query for read-only purposes, such as displaying the result list. Often, the client views only the first few matching records, and then may discard the remaining records and attempt a new query. The search activity often does not involve an immediate transaction on the matching objects. The practice of getting a list of values represented in entity beans by calling an `ejbFind()` method, which returns a collection of remote objects, and then calling each entity bean to get the value, is very network expensive and is considered a bad practice.

There are consequences associated with using EJB finder methods that result in large results sets. Every container implementation has a certain amount of finder method overhead for creating a collection of EJBObject references. Finder method behavior performance varies, depending on a vendor's container implementation. According to the EJB specification, a container may invoke `ejbActivate()` methods on entities found by a finder method. At a minimum, a finder method returns the primary keys of the matching entities, which the container returns to the client as a collection of EJBObject references. This behavior applies for all container implementations. Some container implementations may introduce additional finder method overhead by associating the entity bean instances to these EJBObject instances to give the client access to those entity beans. However, this is a poor use of resources if the client is not interested in accessing the bean or invoking its methods. This overhead can significantly impede application performance if the application includes queries that produce many matching results.

Forces

- The application client needs an efficient query facility to avoid having to call the entity bean's `ejbFind()` method and invoking each remote object returned.
- A server-tier caching mechanism is needed to serve clients that cannot receive and process the entire results set.
- A query that is repeatedly executed on reasonably static data can be optimized to provide faster results. This depends on the application and on the implementation of this pattern.
- EJB finder methods are not suitable for browsing entire tables in the database or for searching large result sets from a table.
- Finder methods may have considerable overhead when used to find large numbers of result objects. The container may create a large number of infrastructure objects to facilitate the finders.
- EJB finder methods are not suitable for caching results. The client may not be able to handle the entire result set in a single call. If so, the client may need server-side caching and navigation functions to traverse the result set.

- EJB finder methods have predetermined query constructs and offer minimum flexibility. The EJB specification 2.0 allows a query language, EJB QL, for container-managed entity beans. EJB QL makes it easier to write portable finders and offers greater flexibility for querying.
- Client wants to scroll forward and backward within a result set.

Solution

Use a Value List Handler to control the search, cache the results, and provide the results to the client in a result set whose size and traversal meets the client's requirements.

This pattern creates a ValueListHandler to control query execution functionality and results caching. The ValueListHandler directly accesses a DAO that can execute the required query. The ValueListHandler stores the results obtained from the DAO as a collection of value objects. The client requests the ValueListHandler to provide the query results as needed. The ValueList-Handler implements an Iterator pattern [GoF] to provide the solution.

Structure

The class diagram in Figure 1.1 illustrates the Value List Handler pattern.

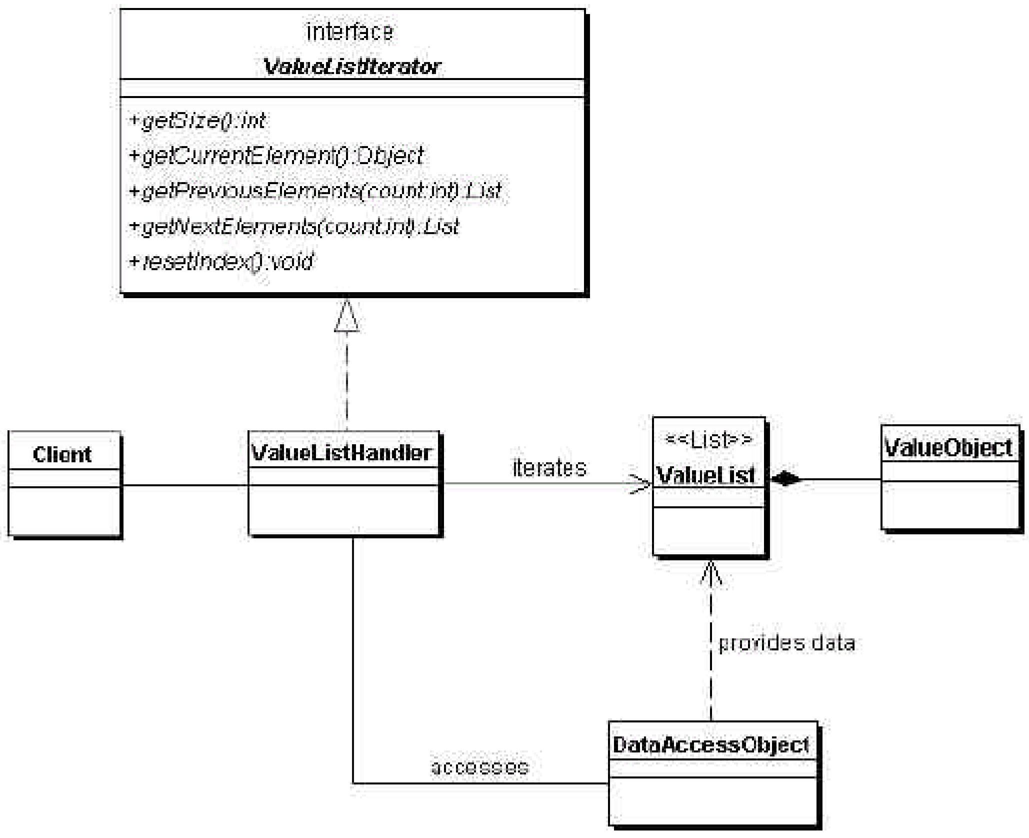

Figure 1.1 Value List Handler Class Diagram.

Participants and Collaborations

The sequence diagram in Figure 1.2 shows the interactions for the Value List Handler.

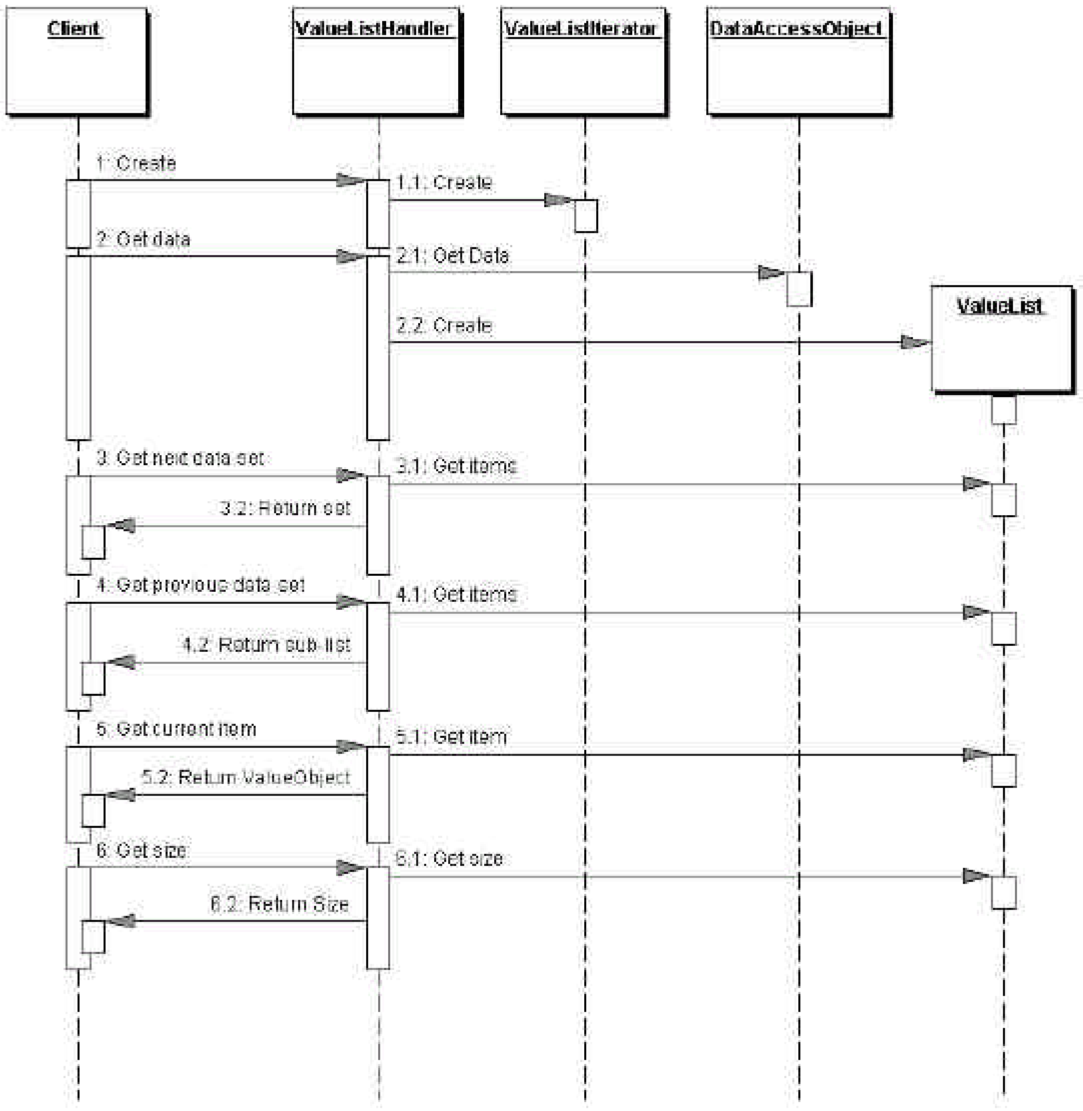

Figure 1.2 Value List Handler Sequence Diagram.

ValueListIterator

This interface may provide iteration facility with the following example methods:

- `getSize()` obtains the size of the result set.
- `getCurrentElement()`obtains the current value object from the list.
- `getPreviousElements(int howMany)` obtains a collection of value objects that are in the list prior to the current element.
- `getNextElements(int howMany)` obtains a collection of value objects that are in the list after the current element.
- `resetIndex()` resets the index to the start of the list.

Depending on the need, other convenience methods can be included to be part of the ValueListIterator interface.

ValueListHandler

This is a list handler object that implements the ValueListIterator interface. The ValueListHandler executes the required query when requested by the client. The ValueListHandler obtains the query results, which it manages in a privately held collection represented by the ValueList object. The ValueListHandler creates and manipulates the ValueList collection. When the client requests the results, the ValueListHandler obtains the value objects from the cached ValueList, creates a new collection of value objects, serializes the collection, and sends it back to the client. The ValueListHandler also tracks the current index and size of the list.

DataAccessObject

The ValueListHandler can make use of a DataAccessObject to keep separate the implementation of the database access. The DataAccessObject provides a simple API to access the database (or any other persistent store), execute the query, and retrieve the results.

ValueList

The ValueList is a collection (a list) that holds the results of the query. The results are stored as value objects. If the query fails to return any matching results, then this list is empty. The ValueListHandler session bean caches ValueList to avoid repeated, unnecessary execution of the query.

ValueObject

The ValueObject represents an object view of the individual record from the query's results. It is an immutable serializable object that provides a placeholder for the data attributes of each record.