Основные J2EE-паттерны
Value List Handler (обработчик списка значений)

Контекст

Клиенту необходим от службы список элементов для презентации. Количество элементов в списке неизвестно и во многих ситуациях может быть довольно большим.

Проблема

Большинство приложений Java 2 Platform, Enterprise Edition (J2EE) должны выполнять операции поиска и запроса списка определенных данных. В некоторых случаях результатом этих операций может быть довольно большой объем информации. Не целесообразно возвращать данные в полном объеме в ситуациях, когда клиенту нужно только последовательно просматривать результаты, а не обрабатывать их полностью. Обычно клиент использует результаты запроса только для чтения, например, для отображения полученного списка. Часто клиент просматривает только первые несколько совпадающих записей и может затем игнорировать оставшиеся записи и выполнить новый запрос. Операции поиска часто не заканчиваются непосредственными транзакциями над найденными объектами. Плохим стилем работы является получение списка значений, представленных в компоненте управления данными, при помощи вызова метода ejbFind(), который возвращает коллекцию удаленных объектов, а затем вызывать каждый компонент для получения значения, расходуя при этом значительные сетевые ресурсы.

Имеются некоторые негативные последствия использования методов finder корпоративных компонентов JavaBeans (EJB), которые могут возвращать большие наборы данных. При реализации каждого контейнера существуют определенные накладные расходы на создание коллекции ссылок EJBObject. Производительность работы метода finder меняется в зависимости от реализации контейнера поставщиком. Согласно EJB-спецификации контейнер может вызывать методы ejbActivate() объектов, найденных методом finder. Как минимум, метод finder возвращает первичные ключи соответствующих сущностей, которые контейнер возвращает клиенту в виде коллекции ссылок EJBObject. Такое поведение присуще всем реализациям контейнеров. Некоторые реализации могут иметь дополнительные накладные расходы, связанные с методом finder, при сопоставлении экземпляров компонентов управления данными с этими EJBObject-ссылками, для того чтобы предоставить клиенту доступ к этим компонентам. Однако, если клиенту не нужно обращаться к компоненту или вызывать его методы, эти ресурсы используются непроизводительно. В этом случае производительность приложения может значительно понизиться, если приложение использует запросы, генерирующие большое число совпадающих записей.

Ограничения

- Клиенту приложения необходим эффективный метод работы с запросами, для того чтобы избежать вызова метода ejbFind() компонента управления данными и обращения к каждому возвращенному удаленному объекту.
- Необходим механизм кэширования на сервере для обслуживания клиентов, которые не могут принимать и обрабатывать весь возвращаемый набор данных.
- Запрос, который периодически выполняется с достаточно статичными данными, может быть оптимизирован для более быстрого получения результатов. Это зависит от приложения и от реализации паттерна.
- Методы EJB finder не подходят для просмотра таблиц в базе данных или для поиска больших наборов данных в таблице.
- Методы finder могут потребовать значительных ресурсов при использовании их для поиска большого количества объектов. Контейнер может создать большое число объектов инфраструктуры для помощи методам finder.
- Методы EJB finder не подходят для кэширования результатов. Клиент может быть не способен справиться со всем набором возвращаемых данных за один запрос. Если это так, клиенту может потребоваться кэширование на стороне сервера и функции навигации для перебора возвращаемых данных.
- Методы EJB finder используют предопределенную конструкцию запросов и предлагают минимальную гибкость. EJB-спецификация версии 2.0 разрешает использование языка запросов, EJB QL, для компонентов управления данными управляемых контейнером. EJB QL облегчает запись переносимых методов finder и предлагает большую гибкость при запросах.
- Клиент хочет прокручивать возвращаемый набор данных вперед и назад.

Решение

Используйте Value List Handler для управления поиском, кэшированием результатов и предоставлением результатов клиенту в наборе, чей размер и метод обхода отвечает клиентским требованиям.

Данный паттерн создает ValueListHandler для управления функциями выполнения запросов и кэшированием результатов. ValueListHandler непосредственно обращается к объекту DAO, который может выполнить требуемый запрос. ValueListHandler сохраняет полученные от DAO результаты в виде коллекции объектов Transfer Object. Клиент при необходимости выполняет запросы к ValueListHandler на получение результатов запроса. ValueListHandler реализует паттерн Iterator [GoF] для предоставления решения.

Структура

Диаграмма классов паттерна Value List Handler приведена на рисунке 8.29.

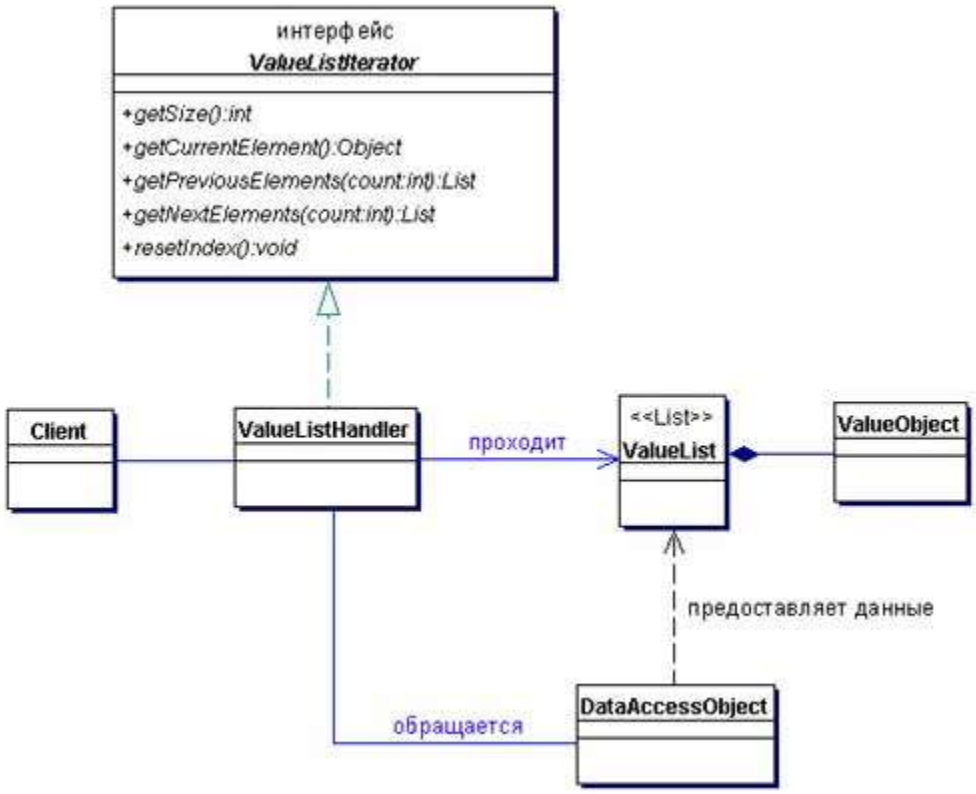

Рисунок 8.29 Диаграмма классов Value List Handler

Участники и ответственности

Диаграмма последовательности действий, изображенная на рисунке 8.30, показывает взаимодействия паттерна Value List Handler.

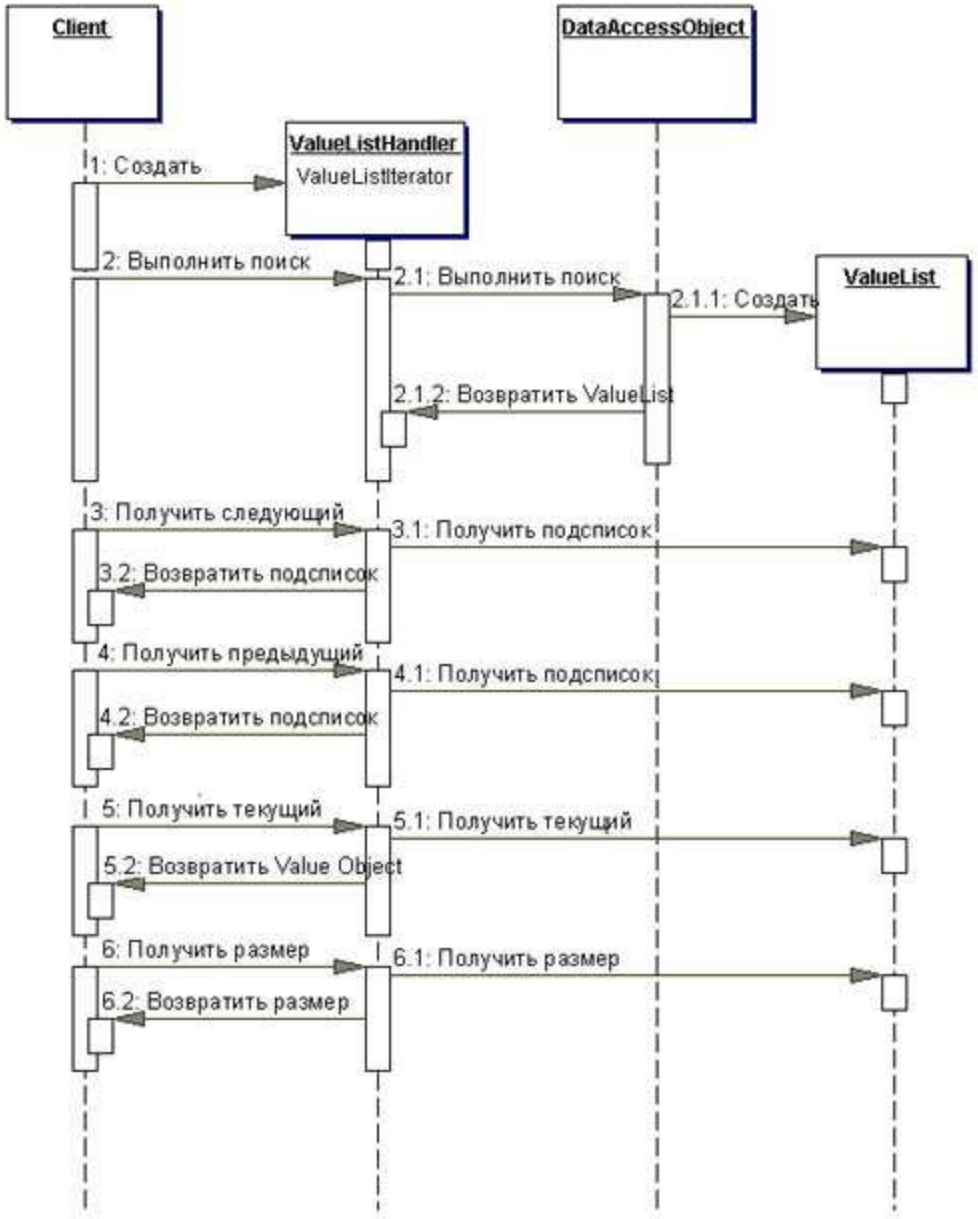

Рисунок 8.30 Диаграмма последовательности действий Value List Handler

ValueListIterator

Данный интерфейс может обеспечить функции итерации, например, следующими методами:

- getSize() - получить размер возвращаемого набора данных.
- getCurrentElement() - получить текущий Transfer Object из списка.
- getPreviousElements(int howMany) - получить коллекцию объектов Transfer Object, находящихся в списке перед текущим элементом.
- getNextElements(int howMany) - получить коллекцию объектов Transfer Object, находящихся в списке после текущего элемента.
- resetIndex() - сбросить индекс на начало списка.

В зависимости от требований и другие удобные методы могут быть включены как часть интерфейса ValueListIterator.

ValueListHandler

Это объект идентификатора списка, который реализует интерфейс ValueListIterator. При получении запроса от клиента ValueListHandler выполняет этот запрос. ValueListHandler получает результаты запроса, которые он сохраняет в собственной коллекции, представленной объектом ValueList. ValueListHandler создает и управляет коллекцией ValueList. Когда клиент запрашивает результаты, ValueListHandler получает объекты Transfer Object из кэшированного ValueList, создает новую коллекцию объектов Transfer Object, сериализует ее и передает клиенту. ValueListHandler отслеживает также текущий индекс и размер списка.

DataAccessObject

ValueListHandler может использовать DataAccessObject, реализующий доступ к базе данных. DataAccessObject предоставляет простой API для доступа к базе данных (или любому другому персистентному хранилищу данных), выполняет запрос и извлекает результаты.

ValueList

ValueList представляет собой коллекцию (список), содержащий результаты запроса. Результаты хранятся в виде объектов Transfer Object. Если запросу не соответствуют никакие результаты, то этот список является пустым. Сессионный компонент ValueListHandler кэширует ValueList для того, чтобы избежать повторных необязательных выполнений запроса.

TransferObject

TransferObject предоставляет объектный взгляд на индивидуальную запись из результатов запроса. Он является неизменяемым сериализуемым объектом, предоставляющим место для атрибутов данных каждой записи.

Стратегии

Стратегия Java Object

ValueListHandler может быть реализован в виде произвольного Java-объекта. В этом случае ValueListHandler может использоваться любым клиентом, которому необходимы функции отображения списка. Эта стратегия полезна для приложений, не применяющих корпоративные компоненты. Например, простые приложения могут быть построены с использованием сервлетов, JavaServer Pages (JSP), паттернов Business Delegate и объектов DAO. В данном сценарии объекты Business Delegate могут использовать ValueListHandler, реализованный как Java-объект, для получения списка значений.

Стратегия Stateful Session Bean

Если приложение на бизнес-уровне использует корпоративные компоненты, вероятно предпочтительнее реализовать сессионный компонент, использующий ValueListHandler. В этом случае сессионный компонент просто служит фасадом экземпляра ValueListHandler. Следовательно, сессионный компонент может быть реализован как сессионный компонент, сохраняющий состояние, для хранения идентификатора списка и может выступать в качестве фасада (см. раздел "Session Facade" на стр. 291) или прокси.

Выводы

- **Предоставляет альтернативу методам EJB finder для больших запросов**
  Обычно метод EJB finder требует много ресурсов и является дорогим способом получения списка элементов, поскольку включает в себя ряд EJBObject-ссылок. Value List Handler реализует сессионный компонент, использующий объект DAO для выполнения запроса и создания коллекции объектов Transfer Object, соответствующих критериям запроса. Поскольку объекты Transfer Object требуют относительно низких накладных расходов по сравнению с EJBObject-ссылками и связанной с ними инфраструктурой, данный паттерн предоставляет преимущества в тех случаях, когда клиентам приложения необходимо посылать запросы, возвращающие большой набор данных.
- **Кэширует результаты запросов на сервере**
  Если клиент должен отображать результат небольшими подмножествами, а не большим списком, набор возвращаемых данных должен быть кэширован. Однако не все клиентские приложения, основанные на броузерах, могут выполнять такое кэширование. В таком случае эти функции должен предоставлять сервер. Паттерн Value List Handler предоставляет функции кэширования в сессионном компоненте, хранящем полученный после выполнения запроса результат. Этот результат представляет собой коллекцию объектов Transfer Object, которые могут быть, если понадобится, сериализованы. Когда клиент запрашивает коллекцию или подмножество коллекции, обрабатывающий компонент возвращает запрошенные результаты в виде сериализованной коллекции объектов Transfer Object. Клиент получает коллекцию и имеет локальную копию запрошенной информации, которую может отобразить или обработать. Когда клиенту понадобится дополнительное подмножество результатов, он передает запрос компоненту на получение другой сериализованной коллекции, содержащей необходимые данные. Клиент может обрабатывать результаты запроса небольшими управляемыми порциями. Компонент также предоставляет клиенту функции навигации (предыдущий и следующий), то есть результат может быть просмотрен при необходимости как в прямом, так и в обратном направлениях.
- **Обеспечивает большую гибкость запросов**
  Добавление нового запроса может потребовать создания нового метода finder, либо изменения существующего метода, особенно при использовании управляемых компонентом компонентов управления данными. (В таких компонентах разработчик реализует методы finder в самом компоненте.) В компонентах управления данными, управляемых контейнером, методы finder

указываются в дескрипторе размещения компонента. Изменения запроса для управляемого контейнером компонента требует изменения спецификации метода finder в дескрипторе размещения. Следовательно, методы finder не пригодны для обработки динамически изменяющихся запросов. Вы можете реализовать Value List Handler более гибким, чем EJB-методы finder, используя специальные функции запросов, создавая аргументы запросов во время исполнения при помощи шаблонных методов и т.д. Другими словами, разработчик Value List Handler может реализовать разумные алгоритмы поиска и кэширования, не ограничиваясь методами finder.

- **Улучшает сетевую производительность**
  Сетевая производительность может улучшиться, поскольку только запрашиваемые данные передаются (сериализуются) клиенту при необходимости. Если клиент отображает первые несколько записей и затем останавливает запрос, сетевой трафик не расходуется, поскольку данные кэшированы на сервере и не передаются клиенту. Однако, если клиент обрабатывает все результаты запроса, он выполняет несколько удаленных вызовов сервера для получения этих результатов. Если клиент заранее знает, что ему необходимы все данные запроса, компонент может предоставить метод, передающий клиенту все результаты за один вызов метода, и функции кэширования не будут использованы.
- **Разрешает отсроченные транзакции компонента управления данным**и
  Кэширование результатов на сервере и минимизация накладных расходов метода finder могут улучшить управление транзакциями. Когда клиент готов для дальнейшей работы с компонентом управления данными, он обращается к компоненту в контексте транзакции, определенном функцией. Например, запрос на отображение списка книг использует Value List Handler для получения списка. Если пользователь захочет просмотреть книгу детально, он включает компонент управления данными этой книги в транзакцию.

Примеры

Реализация Value List Handler в виде объекта Java

Рассмотрим пример, в котором должен быть извлечен и отображен список объектов Project. В данном случае может быть применен паттерн Value List Handler. Исходный код этой реализации приведен в примере 8.29 ProjectListHandler, который отвечает за предоставление списка проектов. Этот класс расширяет основной класс ValueListHandler, обеспечивающий общие функции итерации для всех реализаций Value List Handler в этом приложении. Код ValueListHandler приведен в примере 8.30. ValueListHandler реализует общий интерфейс iterator ValueListIterator, показанный на рисунке 8.32. Соответствующий код из объекта доступа к данным ProjectDAO, используемый объектом ValueListHandler для выполнения запроса и получения соответствующих результатов, приведен в примере 8.31.

Пример 8.29 Реализация паттерна Value List Handler

```
package corepatterns.apps.psa.handlers;
```

```
import java.util.*;
import corepatterns.apps.psa.dao.*;
import corepatterns.apps.psa.util.*;
import corepatterns.apps.psa.core.*;

public class ProjectListHandler
extends ValueListHandler {

  private ProjectDAO dao = null;
 // использовать ProjectTO как шаблон для определения
 // критерия поиска
  private ProjectTO projectCriteria = null;

 // Клиент создает экземпляр ProjectTO, устанавливает
  // значения критерия поиска и передает
 // экземпляр ProjectTO как projectCriteria
 // в конструктор и метод setCriteria()
 public ProjectListHandler(ProjectTO projectCriteria)
  throws ProjectException, ListHandlerException {
    try {
      this.projectCriteria = projectCriteria;
      this.dao = PSADAOFactory.getProjectDAO();
      executeSearch();
    } catch (Exception e) {
   // Обработать исключительную ситуацию, сгенерировать ListHandlerException
    }
  }

  public void setCriteria(ProjectTO projectCriteria) {
    this.projectCriteria = projectCriteria;
  }

  // выполнить поиск. Клиент может инициировать операцию поиска,
  // правильно установив критерий поиска.
  // Используется для обновления
 // списка с последними данными.
  public void executeSearch()
  throws ListHandlerException {
    try {
      if (projectCriteria == null) {
        throw new ListHandlerException(
          "Project Criteria required...");
      }
      List resultsList =
        dao.executeSelect(projectCriteria);
      setList(resultsList);
    } catch (Exception e) {
   // Обработать исключительную ситуацию, сгенерировать ListHandlerException
    }
 }
}
```

Value List Handler является общим классом-итератором, обеспечивающим функции итерации.

Пример 8.30 Реализация общего класса ValueListHandler

```
package corepatterns.apps.psa.util;

import java.util.*;

public class ValueListHandler
implements ValueListIterator {

  protected List list;
  protected ListIterator listIterator;

  public ValueListHandler() {
  }

  protected void setList(List list)
  throws IteratorException {
    this.list = list;
    if(list != null)
      listIterator =  list.listIterator();
    else
      throw new IteratorException("List empty");
  }

  public Collection getList(){
    return list;
  }

  public int getSize() throws IteratorException{
    int size = 0;

    if (list != null)
      size = list.size();
    else
      throw new IteratorException(...); //No Data

    return size;
  }

  public Object getCurrentElement()
  throws IteratorException {

    Object obj = null;
  // Не будет продвигать итератор
    if (list != null)
    {
      int currIndex = listIterator.nextIndex();
      obj = list.get(currIndex);
    }
    else
      throw new IteratorException(...);
    return obj;

  }

  public List getPreviousElements(int count)
  throws IteratorException {
    int i = 0;
    Object object = null;
    LinkedList list = new LinkedList();
    if (listIterator != null) {
```

```
      while (listIterator.hasPrevious() && (i < count)){
        object = listIterator.previous();
        list.add(object);
        i++;
      }
  }// конец if
    else
   throw new IteratorException(...); // Нет данных

    return list;
  }

  public List getNextElements(int count)
  throws IteratorException {
    int i = 0;
    Object object = null;
    LinkedList list = new LinkedList();
    if(listIterator != null){
      while(  listIterator.hasNext() && (i < count) ){
        object = listIterator.next();
        list.add(object);
        i++;
      }
  } // конец if
    else
   throw new IteratorException(...); // Нет данных

    return list;
  }

  public void resetIndex() throws IteratorException{
    if(listIterator != null){
      listIterator = list.ListIterator();
    }
    else
      throw new IteratorException(...); // No data
  }
  ...
}
```

Пример 8.31 Класс ProjectDAO

```
package corepatterns.apps.psa.dao;

public class ProjectDAO {
  final private String tableName = "PROJECT";

 // команда select использует поля
  final private String fields = "project_id, name," +
      "project_manager_id, start_date, end_date, " +
      " started, completed, accepted, acceptedDate," +
      " customer_id, description, status";

 // метод, соответствующий ValueListHandler,
 // приведен здесь.
 // См. паттерн Data Access Object для дополнительной информации.
 ...
  private List executeSelect(ProjectTO projCriteria)
```

```
  throws SQLException {

    Statement stmt= null;
    List list = null;
    Connection con = getConnection();
    StringBuffer selectStatement =   new StringBuffer();
    selectStatement.append("SELECT "+ fields +
          " FROM " + tableName + "where 1=1");

  // добавить дополнительные условия в выражение where
    // в зависимости от значений, указанных в
  // projCriteria
    if (projCriteria.projectId != null) {
      selectStatement.append (" AND PROJECT_ID = '" +
        projCriteria.projectId + "'");
    }
  // проверить и добавить другие поля в выражение where
  ...

    try {
      stmt = con.prepareStatement(selectStatement);
      stmt.setString(1, resourceID);
      ResultSet rs = stmt.executeQuery();
      list = prepareResult(rs);
      stmt.close();
    }
    finally {
      con.close();
    }
    return list;
  }

  private List prepareResult(ResultSet rs)
  throws SQLException {
    ArrayList list = new ArrayList();
    while(rs.next()) {
      int i = 1;
      ProjectTO proj = new
        ProjectTO(rs.getString(i++));
      proj.projectName = rs.getString(i++);
      proj.managerId = rs.getString(i++);
      proj.startDate = rs.getDate(i++);
      proj.endDate = rs.getDate(i++);
      proj.started = rs.getBoolean(i++);
      proj.completed = rs.getBoolean(i++);
      proj.accepted = rs.getBoolean(i++);
      proj.acceptedDate = rs.getDate(i++);
      proj.customerId = rs.getString(i++);
      proj.projectDescription = rs.getString(i++);
      proj.projectStatus = rs.getString(i++);
      list.add(proj);

    }
    return list;
  }
  ...
}
```

Пример 8.32 Класс ValueListIterator

```
package corepatterns.apps.psa.util;

import java.util.List;

public interface ValueListIterator {

  public int getSize()
    throws IteratorException;

  public Object getCurrentElement()
    throws IteratorException;

  public List getPreviousElements(int count)
    throws IteratorException;

  public List getNextElements(int count)
    throws IteratorException;

  public void resetIndex()
    throws IteratorException;

 // другие необходимые общие методы
  ...
}
```

Связанные паттерны

- **Iterator [GoF]**
  Паттерн Value List Handler основывается на паттерне Iterator, описанном в книге GoF, *«Паттерны проектирования: Элементы повторно используемого объектно-ориентированного программного обеспечения»*.
- **Session Facade**
  Поскольку Value List Handler является сессионным компонентом, его можно было бы рассматривать как специализированный паттерн Session Facade. Однако, он скорее является специализированным сессионным компонентом, а не специализированным Session Facade. Паттерн Session Facade имеет другие назначения и характеристики (рассмотренные в разделе «Session Facade»), и он выполняет более общие функции.