Основные J2EE-паттерны
Transfer Object Assembler (сборщик объекта перемещения)

Контекст

В приложениях Java 2 Platform, Enterprise Edition (J2EE), серверные бизнес-компоненты реализуются как сессионные компоненты, компоненты управления данными, объекты DAO, и т.д. Клиентам приложения часто необходимо получить доступ к данным, составленным из нескольких объектов.

Проблема

Клиентам приложения обычно нужны данные для модели, или частей модели, чтобы предоставить их пользователю или использовать для промежуточной обработки перед предоставлением какой-либо службы. Прикладная модель является абстракцией бизнес-данных и бизнес-логики, реализованной на сервере в виде бизнес-компонентов. Модель может быть выражена как коллекция объектов в структурированном виде (дерево или граф). В J2EE-приложении модель является распределенной коллекцией объектов, таких как сессионные компоненты, компоненты управления данными или DAO и другие объекты. Чтобы клиент получил данные для модели, например, для отображения пользователю или для выполнения какой-либо обработки, он должен получить доступ отдельно к каждому распределенному объекту, определяющему модель. Такой подход имеет несколько недостатков:

- Поскольку клиент должен получить доступ к каждому распределенному компоненту по-отдельности, существует тесная связь между клиентом и распределенными компонентами модели по сети.
- Клиент обращается к распределенным компонентам через сетевой уровень, и это может привести к уменьшению производительности в тех случаях, когда модель является сложной, с многочисленными распределенными компонентами. Понижение сетевой и клиентской производительности происходит тогда, когда прикладную модель реализует некоторое количество распределенных компонентов и клиент непосредственно взаимодействует с этими компонентами для получения данных модели от этих компонентов. Каждое такое обращение приводит к удаленному вызову метода, что забирает сетевые ресурсы и увеличивает трафик между клиентом и бизнес-уровнем.
- После получения частей модели от распределенных компонентов клиент должен воссоздать модель. То есть, клиент должен обладать необходимой бизнес-логикой для создания модели. Если построение модели является сложным процессом и в ее описание включено большое количество объектов, то возможно появление дополнительных сетевых накладных расходов на клиенте из-за процесса построения. Кроме того, клиент должен содержать бизнес-логику для управления взаимоотношениями между компонентами, что приводит к появлению более сложных и больших клиентов. Построение прикладной модели

происходит на стороне клиента. Построение сложной модели может привести к значительному снижению производительности на стороне клиента, имеющего ограниченные ресурсы.
- Поскольку клиент тесно привязан к модели, при изменениях в модели необходимо изменить и клиентский код. Более того, если существуют несколько типов клиентов, управлять изменениями всех этих типов клиентов становится еще более трудно. При существовании тесной связи клиента и реализации модели, возникающей в тех случаях, когда клиент обладает информацией о модели и может непосредственно управлять взаимоотношениями бизнес-компонентов, изменения в модели неизбежно приводят к необходимости изменений в клиентском коде. Существует также проблема дублирования кода доступа к модели, возникающая при наличии нескольких типов клиентов в приложении. Такое дублирование усложняет управление клиентским кодом при изменениях модели.

Ограничения

- Необходимо разделение бизнес-логики между клиентом и серверными компонентами.
- Поскольку модель состоит из распределеных компонентов, доступ к каждому компоненту связан с расходом сетевых ресурсов. Желательно минимизировать количество удаленных вызовов методов по сети.
- Клиентскому приложению обычно необходимо только получить модель для предоставления ее пользователю. Если клиент должен для построения модели взаимодействовать с несколькими компонентами в реальном времени, трафик между ним и приложением увеличивается. Этот трафик может снизить сетевую производительность.
- Даже тогда, когда клиент хочет выполнить обновление, он обычно изменяет только определенную часть модели, а не модель полностью.
- Клиенты не должны ничего знать о сложностях и зависимостях в реализации модели. Желательно наличие слабой связи между клиентами и бизнес-компонентами, реализующими прикладную модель.
- Другими словами, клиенты не должны реализовывать дополнительную бизнес-логику, требуемую для построения модели из информации, полученной от различных бизнес-компонент.

Решение

Используйте Transfer Object Assembler для построения требуемой модели или субмодели. Transfer Object Assembler использует паттерн Transfer Objects для извлечения данных из различных бизнес-объектов, определяющих модель, или часть модели.

Transfer Object Assember создает составной Transfer Object, представляющий данные от различных бизнес-компонентов. Transfer Object передает клиенту данные о модели за один вызов метода. Поскольку данные о модели могут быть сложными, рекомендуется, чтобы этот Transfer Object был неизменяемым. То есть, клиент получает такие Transfer Object с единственной целью - использовать их для

презентации и обработки, получая доступ к ним только для чтения. Клиентам запрещено производить изменения в Transfer Object.

Если клиенту нужны данные о модели, а модель представлена единственным компонентом общего назначения (таким как Composite Entity), то процесс получения этих данных является простым. Клиент просто передает запрос этому компоненту на получение его составного Transfer Object. Однако, большинство реальных приложений используют модель, составленную из комбинации нескольких компонентов общего назначения и специализированных компонентов. В этом случае клиент для получения всех необходимых данных о модели должен взаимодействовать с несколькими бизнес-компонентами. Сразу же можно увидеть недостатки этого подхода - клиенты становятся тесно связанными с реализацией модели (элементами модели) и должны выполнить несколько удаленных вызовов методов для получения данных от каждого конкретного компонента.

В некоторых случаях единственный компонент общего назначения предоставляет модель или части модели в виде одного Transfer Object (простого или составного). Однако, когда модель представляется несколькими компонентами, один Transfer Object (простой или составной) может не предоставить модель полностью. Для этого необходимо получить паттерны Transfer Object из различных компонентов и собрать их вместе в один новый составной Transfer Object. Такое построение модели «на лету» должен выполнять сервер, а не клиент.

Структура

На рисунке 8.27 приведена диаграмма классов, представляющая взаимосвязи паттерна Transfer Object Assembler.

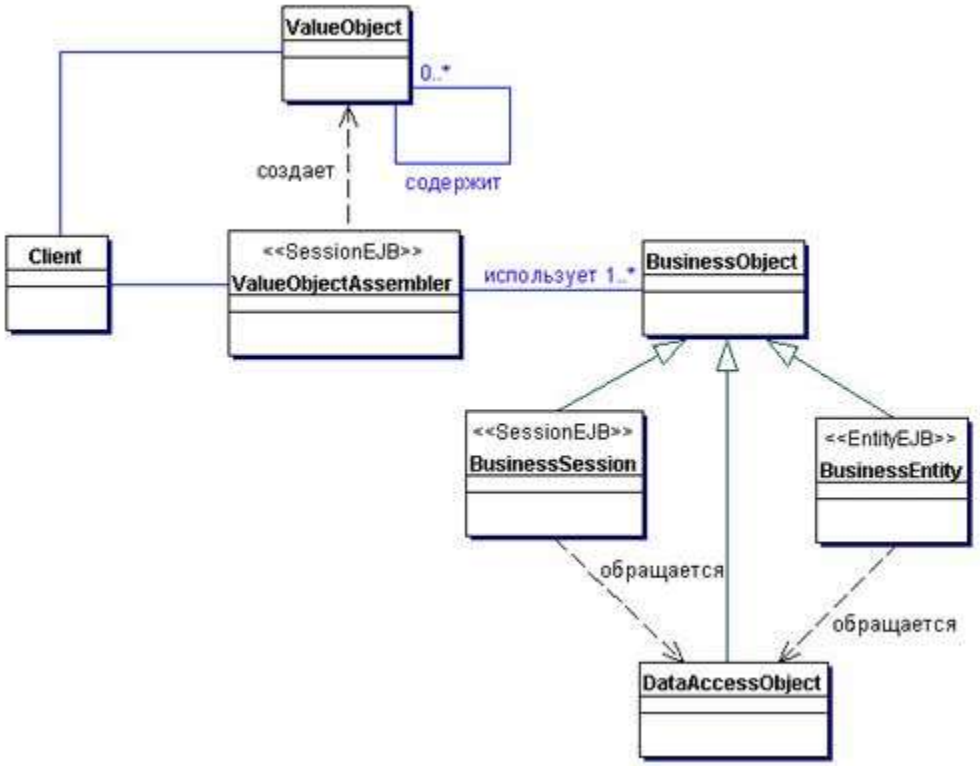

Рисунок 8.27 Диаграмма классов Transfer Object Assembler

Участники и ответственности

Диаграмма последовательности действий, изображенная на рисунке 8.28, показывает взаимодействие между различными участниками в паттерне Transfer Object Assembler.

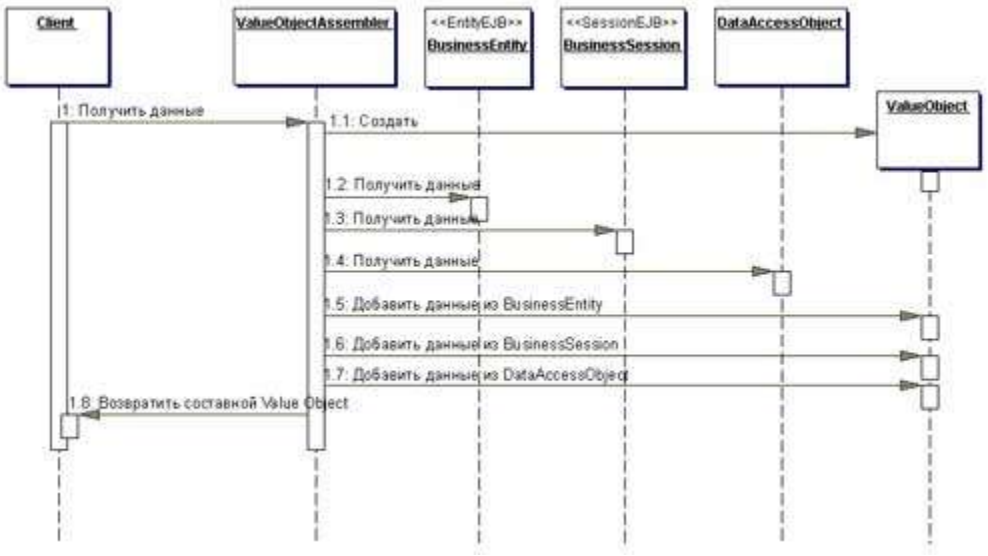

Рисунок 8.28 Диаграмма последовательности действий Transfer Object

Assembler

TransferObjectAssembler

TransferObjectAssembler представляет собой основной класс данного паттерна. TransferObjectAssembler создает новый Transfer Object, основываясь на требованиях приложения, при выполнении клиентом запроса на получение составного Transfer Object. Затем TransferObjectAssembler находит требуемые экземпляры BusinessObject для извлечения данных и построения составного Transfer Object. Объекты BusinessObject являются компонентами бизнес-уровня, например, компонентами управления даными или сессионными компонентами, объектами DAO и др.

Client

Если TransferObjectAssembler реализуется как произвольный Java-объект, клиентом обычно является Session Facade, представляющий уровень контроллера для бизнес-уровня. Если TransferObjectAssembler реализуется как сессионный компонент, клиентом может быть Session Facade или Business Delegate.

BusinessObject

BusinessObject участвует в построении нового Transfer Object, предоставляя требуемые данные в TransferObjectAssembler. Следовательно, BusinessObject является ролью, которая может быть выполнена сессионным компонентом, компонентом управления данными, объектом DAO или произвольным Java-объектом.

TransferObject

TransferObject представляет собой составной Transfer Object, созданный объектом TransferObjectAssembler и возвращаемый клиенту. Он представляет сложные данные от различных компонентов, определяющих прикладную модель.

BusinessObject

BusinessObject представляет собой роль, которая может быть выполнена сессионным компонентом, компонентом управления данными или объектом DAO. Когда паттерну необходимо получить данные непосредственно из персистентного хранилища для построения Transfer Object, он может использовать DAO. Он показан на диаграммах как объект DataAccessObject.

Стратегии

В данном разделе рассматриваются различные стратегии реализации паттерна Transfer Object Assembler.

Стратегия Java Object

TransferObjectAssembler может быть произвольным Java-объектом и не обязательно должен быть корпоративным компонентом. В таких реализациях доступ к

TransferObjectAssembler осуществляется обычно через сессионный компонент. Сессионным компонентом обычно является Session Facade, выполняющий и другие функции, связанные с предоставлением бизнес-служб. TransferObjectAssembler выполняется на бизнес-уровне независимо от стратегий реализации. Это вызвано стремлением защитить от перехода через уровень удаленные вызовы из TransferObjectAssembler к объектам-источникам.

Стратегия Session Bean

В этой стратегии TransferObjectAssembler реализуется в виде сессионного компонента (как показано на диаграмме классов). Если для предоставления TransferObjectAssembler как бизнес-службы предпочтительнее выбрать сессионный компонент, его обычно реализуют как сессионный компонент, не имеющий состояния. Составляющие прикладную модель бизнес-компоненты постоянно вовлечены в транзакции с различными клиентами. В результате этого, при построении объектом TransferObjectAssembler нового составного Transfer Object из различных бизнес-компонентов, он создает снимок модели на момент ее построения. Модель может сразу же измениться, если другой клиент меняет один или несколько бизнес-компонентов, меняя, соответственно, прикладную бизнес-модель.

Следовательно, реализация TransferObjectAssembler как сессионного компонента, сохраняющего состояние, не предоставляет никаких преимуществ в сравнении с реализацией его в виде сессионного компонента, не имеющего состояния, поскольку сохранение состояния значений составных данных модели, которые могут изменяться, бесполезно. Если модель меняется, хранимый Transfer Object становится устаревшим. TransferObjectAssembler при следующем запросе объекта Transfer Object может возвратить либо устаревший экземпляр, либо заново построить его для возврата самого последнего состояния. Таким образом, рекомендуется реализовывать Transfer Object Assembler в виде сессионного компонента, не имеющего состояния, для использования его преимуществ перед компонентами, сохраняющими состояние.

Однако, если прикладная модель изменяется редко, Transfer Object Assembler может быть сессионным компонентом, сохраняющим состояние и хранящим построенный Transfer Object. В данном случае TransferObjectAssembler должен иметь механизмы распознавания изменений модели для того чтобы заново построить модель при следующем клиентском запросе.

Стратегия Business Object

Роль BusinessObject в данном паттерне может выполняться различными типами объектов, что и рассматривается ниже.

- BusinessObject может быть сессионным компонентом. Transfer Object Assembler может использовать Service Locator (см. раздел "Service Locator" на стр. 368) для поиска требуемого сессионного компонента. Transfer Object Assembler выполняет запрос к этому сессионному компоненту на получение данных для

построения составного Transfer Object.

- BusinessObject может быть компонентом управления данными. Transfer Object Assembler может использовать Service Locator для поиска требуемого компонента управления данными. Transfer Object Assembler выполняет запрос к этому компоненту на получение данных для построения составного Transfer Object.
- BusinessObject может быть объектом DAO. Transfer Object Assembler выполняет запрос к этому DAO на получение данных для построения составного Transfer Object.
- BusinessObject может быть произвольным Java-объектом. The Transfer Object Assembler выполняет запрос к этому Java-объекту на получение данных для построения составного Transfer Object.
- BusinessObject может быть другим объектом Transfer Object Assembler. Первый Transfer Object Assembler выполняет запрос ко второму Transfer Object Assembler на получение данных для построения составного Transfer Object.

Выводы

- **Разделяет бизнес-логику**
  Когда клиент содержит логику для управления взаимодействиями с распределенными компонентами, становится трудно четко отделить бизнес-логику от клиентского уровня. Transfer Object Assembler содержит бизнес-логику для управления взаимодействием между объектами и для построения составного Transfer Object, представляющего модель. Клиенту не нужна информация о том, как построить модель, и информация о различных компонентах, предоставляющих данные для построения этой модели.
- **Ослабляет связь между клиентами и прикладной моделью**
  Transfer Object Assembler скрывает сложность построения данных о модели от клиентов и устанавливает слабую связь между клиентами и моделью. При слабой связи, если модель изменяется, необходимо также соответствующее изменение и Transfer Object Assembler. Однако, клиент не зависит от процесса построения модели и от взаимозависимостей между бизнес-компонентами модели, так что изменение модели прямо не влияет на клиента. В общем случае слабая связь предпочтительнее тесной связи.
- **Улучшает сетевую производительность**
  Transfer Object Assembler значительно уменьшает накладные расходы на удаленные вызовы методов и на трафик. Клиент может запросить данные о прикладной модели у Transfer Object Assembler за один удаленный вызов метода. Transfer Object Assembler строит составной Transfer Object для модели и возвращает клиенту. Однако, составной Transfer Object может содержать большое количество данных. Таким образом, хотя использование Transfer Object Assembler уменьшает количество сетевых вызовов, увеличивается объем данных, передаваемых в одном запросе. Этот компромисс следует учитывать при использовании данного паттерна.
- **Улучшает клиентскую производительность**
  Серверный Transfer Object Assembler строит модель в виде составного Transfer Object без использования каких-либо клиентских ресурсов. Клиент не тратит

время на построение модели.

- **Улучшает производительность транзакций**
  Обычно обновления изолированы для очень маленькой части модели и могут быть выполнены узкоспециализированными транзакциями. Эти транзакции концентрируются на изолированных частях модели, а не блокируют объект (модель) полностью. После получения клиентом модели и отображения ее, либо локальной обработки, пользователь (или клиент) может пожелать обновить, либо изменить модель. Для выполнения этой задачи клиент может взаимодействовать с Session Facade напрямую на соответствующем уровне модульности. Transfer Object Assembler не вовлечен в транзакцию при обновлении или изменении модели. Это повышает производительность, поскольку работа транзакций с моделью происходит на соответствующем уровне модульности.
- **Может предоставлять устаревшие объекты Transfer Object**
  Transfer Object Assembler строит объекты Transfer Object по требованию. Эти объекты являются снимками текущего состояния модели, представленной различными бизнес-компонентами. После получения клиентом Transfer Object этот Transfer Object является полностью локальным для клиента. Поскольку объекты Transfer Object не уведомляются по сети об изменениях, изменения, выполненные в бизнес-компонентах и используемые при построении Transfer Object, не отражаются в объектах Transfer Object. Следовательно, после получения Transfer Object он может быстро устареть при выполнении транзакций с бизнес-компонентами.

Пример

Реализация Transfer Object Assembler

Рассмотрим приложение «Управление проектами», в котором несколько компонентов бизнес-уровня определяют сложную модель. Предположим, что клиент хочет получить данные о модели, составленные из данных от различных бизнес-объектов, а именно:

- Информацию о проекте из компонента Project
- Информацию о менеджере проектов из компонента ProjectManager
- Список проектов из компонента Project
- Информацию о ресурсах из компонента Resource

Составной объект Transfer Object, содержащий эти данные, может быть определен так, как показано на рисунке 8.24. Для сборки этого составного Transfer Object может быть реализован паттерн Transfer Object Assembler. Код Transfer Object Assembler приведен в примере 8.28.

Пример 8.24 Класс составного Transfer Object

```
public class ProjectDetailsData {
  public ProjectTO projectData;
  public ProjectManagerTO projectManagerData;
  public Collection listOfTasks;
```

```
  ...
}
```

Список задач в ProjectDetailsData представляет собой коллекцию объектов TaskResourceTO. Объект TaskResourceTO является комбинацией TaskTO и ResourceTO. Код этих классов приведен в примерах 8.25, 8.26 и 8.27.

Пример 8.25 Класс TaskResourceTO

```
public class TaskResourceTO {
  public String projectId;
  public String taskId;
  public String name;
  public String description;
  public Date startDate;
  public Date endDate;
  public ResourceTO assignedResource;
  ...

  public TaskResourceTO(String projectId,
    String taskId, String name, String description,
    Date startDate, Date endDate, ResourceTO
    assignedResource) {
        this.projectId = projectId;
        this.taskId = taskId;
        ...
        this.assignedResource = assignedResource;
    }
    ...
}
```

Пример 8.26 Класс TaskTO

```
public class TaskTO {
  public String projectId;
  public String taskId;
  public String name;
  public String description;
  public Date startDate;
  public Date endDate;
  public assignedResourceId;

  public TaskTO(String projectId, String taskId,
      String name, String description, Date startDate,
      Date endDate, String assignedResourceId) {
        this.projectId = projectId;
        this.taskId = taskId;
        ...
        this.assignedResource = assignedResource;
    }
    ...
}
```

Пример 8.27 Класс ResourceTO

```
public class ResourceTO {
  public String resourceId;
  public String resourceName;
  public String resourceEmail;
```

```
  ...

  public ResourceTO (String resourceId, String
    resourceName, String resourceEmail, ...) {
      this.resourceId = resourceId;
      this.resourceName = resourceName;
      this.resourceEmail = resourceEmail;
      ...
  }
}
```

Код класса ProjectDetailsAssembler, собирающего объект ProjectDetailsData приведен в примере 8.28.

Пример 8.28 Реализация Transfer Object Assembler

```
public class ProjectDetailsAssembler
  implements javax.ejb.SessionBean {

  ...

  public ProjectDetailsData getData(String projectId){

  // Построить составной Transfer Object
    ProjectDetailsData pData = new
                      ProjectDetailsData();

    //получить детальную информацию о проекте;
  ProjectHome projectHome =
        ServiceLocator.getInstance().getHome(
          "Project", ProjectEntityHome.class);
    ProjectEntity project =
      projectHome.findByPrimaryKey(projectId);
    ProjectTO projTO = project.getData();

  // Добавить информацию о проекте в ProjectDetailsData
  pData.projectData = projTO;

    // получить детальную информацию о менеджере проекта;
  ProjectManagerHome projectManagerHome =
      ServiceLocator.getInstance().getHome(
        "ProjectManager", ProjectEntityHome.class);

    ProjectManagerEntity projectManager =
      projectManagerHome.findByPrimaryKey(
        projTO.managerId);

    ProjectManagerTO projMgrTO =
      projectManager.getData();

  // Добавить информацию о ProjectManager в ProjectDetailsData
  pData.projectManagerData = projMgrTO;

  // Получить список TaskTO из Project
    Collection projTaskList = project.getTasksList();

  // построить список TaskResourceTO
```

```
    ArrayList listOfTasks = new ArrayList();

    Iterator taskIter = projTaskList.iterator();
    while (taskIter.hasNext()) {
      TaskTO task = (TaskTO) taskIter.next();

      // получить детальную информацию о ресурсе;
    ResourceHome resourceHome =
      ServiceLocator.getInstance().getHome(
        "Resource",         ResourceEntityHome.class);

      ResourceEntity resource =
        resourceHome.findByPrimaryKey(
          task.assignedResourceId);

      ResourceTO resTO = resource.getResourceData();

    // построить новый объект TaskResourceTO, используя данные из Task
    // и Resource
      TaskResourceTO trTO = new TaskResourceTO(
              task.projectId, task.taskId,
              task.name, task.description,
              task.startDate, task.endDate,
              resTO);

    // добавить TaskResourceTO в список
      listOfTasks.add(trTO);
    }

  // добавить список задач в ProjectDetailsData
    pData.listOfTasks = listOfTasks;

  // добавить любые другие данные в Transfer Object
  ...

  // возвратить составной Transfer Object
    return pData;

  }

  ...
}
```

Связанные паттерны

- **Transfer Object**
  Transfer Object Assembler использует паттерн Transfer Object для создания и передачи объектов Transfer Objects клиенту. Созданные объекты Transfer Object переносят данные, представляющие прикладную модель, из бизнес-уровня к клиентам, запрашивающим данные.
- **Composite Entity**
  Паттерн Composite Entity поддерживает разработку компонента управления данными общего назначения, который может создавать составные объекты Transfer Object, аналогичные создаваемым паттерном Transfer Object Assembler. Однако, Transfer Object Assembler больше подходит для построения составного

Transfer Object, порождаемого из нескольких компонентов (сессионных компонентов, компонентов управления данными, DAO и др.), в то время как паттерн Composite Entity создает Transfer Object из своих собственных данных (то есть, из одного компонента управления данными).

- **Session Facade**
  Transfer Object Assembler обычно реализуется в виде сессионного компонента, не имеющего состояний. По существу, его можно рассматривать как ограниченное специальное приложение паттерна Session Facade. Что более важно, Transfer Object Assembler создает составные объекты Transfer Object, являющиеся неизменяемыми. Следовательно, клиент, получая этот составной Transfer Object, может использовать его только для целей презентации и обработки. Клиент не может обновлять Transfer Object. Если клиенту понадобится обновить бизнес-объекты, составляющие составной Transfer Object, он должен обратиться к Session Facade (сессионному компоненту), обеспечивающему такую возможность.
- **Data Access Object**
  Возможной стратегией для Transfer Object Assembler является получение данных для составного Transfer Object из базы данных без использования корпоративного компонента. В этом случае может быть использован паттерн Data Access Object, что позволит в паттерне Transfer Object Assembler использовать его преимущества для обеспечения доступа к базе данных.
- **Service Locator**
  Transfer Object Assembler должен искать и использовать различные бизнес-объекты. Для поиска бизнес-объекта или службы совместно с паттерном Transfer Object Assembler может быть использован паттерн Service Locator.