Сократите сетевой траффик EJB при помощи астральных клонов

Для предотвращения удаленных вызовов используйте реализацию компонента вне контейнера.

Enterprise JavaBeans

Мартин Рес

При работе с EJB возникает необходимость вызывать набор удаленных методов, что даже при при простом отображении данных на экране влечет снижение эффективности. Использование *астральных клонов* на клиентской стороне предоставляет элегантный и практически прозрачный способ разрешить эти проблемы. В этой статье Мартин Рес описывает простую, но мощную технику использования астральных клонов. *(1,800 слов в английском тексте)*

Платформа Java 2 Enterprise Edition позиционирует Java в качестве доступного стандарта для распределенных вычислений. В этот стандарт включена спецификация Enterprise JavaBeans (EJB), которая предоставляет набор правил программирования и стандартные интерфейсы для построения бизнес объектов. Вы можете разместить эти объекты на сервере приложений, который вам нет необходимости разрабатывать самим. Сервер приложений предоставляет среду, в которой объекты будут надежно и эффективно исполнены, участвуя в распределенных транзакциях и сохраняться в реляционной базе данных.

Несмотря на то что эта модель концентрирует разработчика на бизнес логике, вместо изобретения механизмов транзакций и отображения на базу данных, она оказывается слишком жесткой для распределенных вычислений в реальном мире. Медленные сети реального мира снижают быстродействие взаимодействующих бизнес объектов исполняемых на разных машинах. Таким образом, простой ввод данных

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

пользователем, создание и редактирование бизнес объектов могут выполняться слишком медленно если, к примеру, приложение на Java Server Pages работает на другой машине по отношению к бизнес объекту, или человек использует клиентское приложение написанное на Java на своей рабочей станции.

В общем случае, если устойчивый компонент (entity bean) исполняется в ином месте чем процесс исполняющий конкретные действия, и сеть является узким местом, любой из двух процессов должен быть перенесен, либо тот что работает с данными должен быть перенесен к данным, либо данные должны быть перенесены к процессу. примером первого подхода может служить сессионный компонент (session bean), который выполняет определенные действия от имени клиента. Классическая клиент-серверная модель, является реализацией второго подхода, но я использую другой способ, который я называю *астральным клонированием.*

В этой статье, я покажу как использовать реализацию устойчивых компонентов на стороне клиента в качестве кешированной версии компонента. Реализация исполняется за пределами сервера приложений, в так называемом "вне-контейнерном режиме" (outer-container), исходя из чего и назван *астральным клоном*. Многие разработчики используют различные варианты кеширования для повышения производительности распределенной системы. Астральное клонирование представляет улучшеннцю версию решения предложенного Томасом Дэвисом (Thomas Davis) в статье "Direct Network Traffic of EJBs" (*JavaWorld,* Ноябрь 1999; см.Ресурсы) относительно композиции бизнес логики в объект и прозрачности ее для клиентского кода.

Примечание: Эта статья предполагает, что вы хорошо знакомы с программированием EJB. Также я опишу астральное копирование и его важность в сравнении с устойчивыми объектами управляемыми только контейнером.

1. Астральное копирование

Для того, чтобы поясненить написание астральных клонов, сначала я опишу обычную работу устойчивого компонента.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

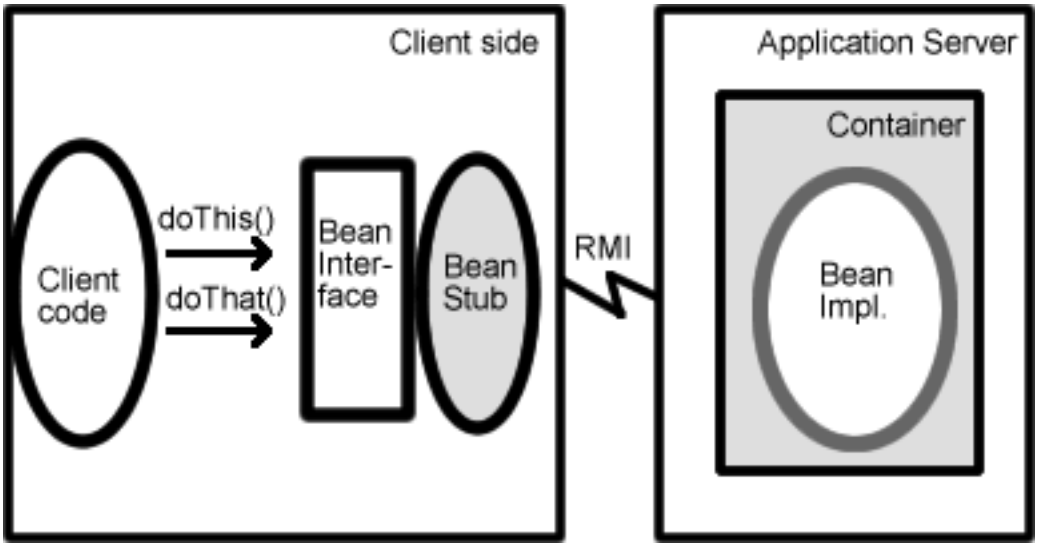

Рисунок 1. Обычный вызов через локальную заглушку (stub).

На рисунке 1 показано как клиент вызывает несколько методов сгенерированной программной заглушки (stub), которая реализует интерфейс компонента. При помощи механизма вызова удаленных методов (RMI), заглушка перезапрашивает эти вызовы у контейнера который создается для исполнения экземпляров реализаций компонента. наконец, контейнер вызывает нужную копию реализации компонента.

Вместо вызова удаленного компонента мы можем создать локальную копию этого компонента — астральную копию — следующим образом:

1. Объявить метод `getAstralClone()` в интерфейсе устойчивого компонента
2. Позволить самому устойчивому компоненту реализовать этот метод возвратив ссылку на него самого. (`this`)
3. Когда клиент вызывает `getAstralClone()` удаленного устойчивого компонента, клон компонента возвращается при помощи механизма сериализации/десериализации (стандартный механизм RMI)
4. Теперь клиент имеет доступ к тому что ранее было удаленными методами.

Рисунок 2 показывает как работает астральный клон.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

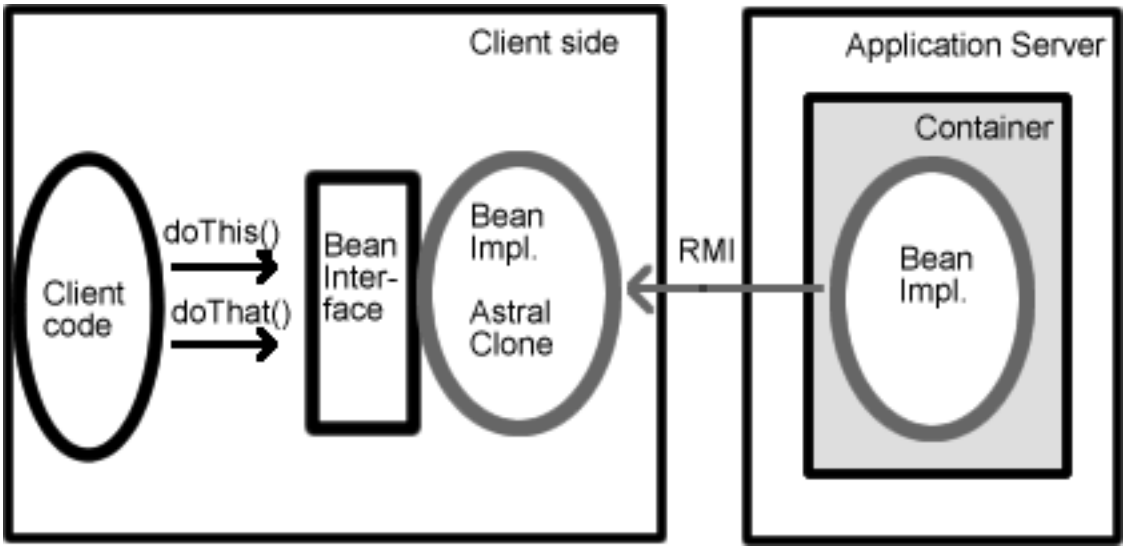

Рисунок 2. Работа с астральным клоном.

Применение астральных клонов дает следующие преимущества:

- Улучшение быстродействие при многочисленных вызовах `get()`, `set()`, так же как и других методов компонента.
- Один класс следит за согласованием состояния объектов. Это немаловажно потому что вы можете хранить *всю* бизнес логику объекта в одном месте, включая проверку ввода. Проверка ввода обычно разбрасывается по всей системе при попытках разработчика оптимизировать систему.
- Код клиентской части немного отличается от того что необходим для классического вызова через заглушку.

Давайте посмотрим на простой пример компонента `PersonBean` управляемого контейнером:

```
public interface Person {
    public String getName() throws RemoteException;
    public void setName(String aName) throws RemoteException;
    public int getAge() throws RemoteException;
    public setBirthDay(Calendar aDate)
             throws SomeValidationException, RemoteException;
}
```

Обратите внимание на то, что интерфейс `Person` не может быть использован в качестве интерфейса EJB, так как он не расширяет `EJBObject`. Я отделил

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

функциональную часть компонента Person от технических деталей, как и при использовании EJB. Следующий фрагмент кода объявляет интерфейс компонента:

```
public interface EJBPerson extends Person, EJBObject {
   // вернуть астральный клон
   public Person getAstralClone() throws RemoteException;
}
```

Реализация компонента `PersonBean` объявляет метод `getAstralClone()` в дополнение к остальным методам компонента `Person` примерно так:

```
// вернуть астральный клон
public Person getAstralClone() throws RemoteException {
   return this;
}
```

Чтобы заставить приведенный фрагмент работать, `PersonBean` должен реализовать следующие два интерфейса:

- `java.io.Serializable`, для того, чтобы он мог вернуть себя в качестве возвращаемого значения при удаленном вызове
- `Person`, для (1) компиляции приведенного выше фрагмента кода, и (2) позволить клиенту взаимодействовать с `PersonBean` как с астральным клоном так же, как он взаимодействует с заглушкой `EJBPerson`

Рисунок 3 показывает связь между различными интерфейсами и классами.

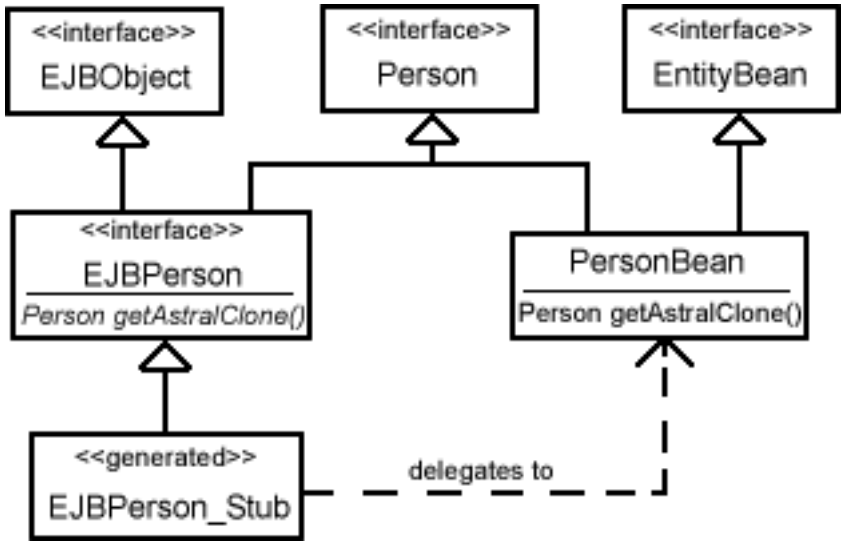

Рисунок 3. Диаграмма классов примера PersonBean.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Моя компания использовала технику астрального клонирования в наборе компонентов разработанных для казначейства Dutch bank-a. Устойчивые компоненты примененные для доступа к редко изменяющимся статическим данным, таким например как валюты или календарь выходных дней позволило черезвычайно повысить производительность торговой инфраструктуры и ядра рассчета курсов валют в реальном времени; т.е. тех модулей которые сильно зависят от данных. Основываясь на практическом опыте, в следующих разделах я буду тщательно анализировать и решать трудные моменты астрального клонирования такие как:

- Как поступать с `EntityContext`
- Как убедиться в том, что устойчивый компонент который мы используем в качестве использовании астрального клона инициализируется и работает аналогично тому как если бы он исполнялся в контейнере
- Как использовать переменные экземпляра (instance variables) которые не являются частью устойчивого состояния объекта
- Как сделать астральный клон безопасным для использования в потоках (thread)
- Как использовать астральный клон для изменения оригинального компонента

1.1. EntityContext

При работе с астральными клонами вы должны сначала решить, как поступать с `EntityContext`, как описано в спецификации пакета `javax.ejb`:

Интерфейс `EntityContext` предоставляет экземпляр корпоративного компонента с доступом его к контексту исполнения, предоставленным контейнером. Контейнер передает интерфейс `EntityContext` после создания экземпляра объекта.

(Вы можете найти ссылку на полную документацию по `EntityContext` в формате javadoc в приложенных Ресурсах

Каждый устойчивый компонент обязан иметь экземплярную переменную `javax.ejb.EntityContext`. Однако значение предоставленное контейнером содержащееся в ней не предназначено для функионирования за пределами контейнера (или значение может быть несериализуемым, и ,следовательно, не может быть доставлено через механизм RMI). Поэтому вы должны объявлять эту переменную следующим способом:

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
private transient EntityContext context;
```

Каждый раз, когда вы хотите сослаться на `context`, вы должны сначала проверить его на `null`. Если `context` возвращает `null`, то вы имеете дело с астральным клоном.

1.2. Инициализация клона

После инициализации астральный клон должен иметь то же состояние что и его оригинальный компонент после его загрузки и активации; и что оба готовы к использованию клиентом. В случае если компонент управляется контейнером, последний вызывает метод `ejbLoad()` сразу после заполнения переменных экземпляра значениями из базы данных. Вы можете свой поместить код в этот метод для подготовки объекта к использованию.

Но ничто автоматически не вызывает `ejbLoad()` для астрального клона после его создания и десериализации. Следовательно, после вызова `getAstralClone()` вы должны вызывать этот метод для клона. Однако, я предпочитаю оставлять код метода `ejbLoad()` пустым, и использовать идиому программирования ленивой инициализации для выполнения пропущенного кода.

Например, если таблица `person` в базе данных содержит поле `partner_id`, то вы можете инициализировать переменную экземляра `partner` EJB ссылкой на компонент партнера в методе `ejbLoad()`, и использовать переменную `partner` до конца жизни компонента. Однако лучше оставлять `ejbLoad()` пустым, и использовать метод `getPartner()` в жизненном цикле компонента примерно так:

```
private Person getPartner() throws FinderException, RemoteException {
  if (partner_id == null) {
     return null;
  }

  // просто получить объект партнера, при необходимости
  if (partner == null) {
     partner = getPersonHome().findByPK(new PersonPK(partner_id));
  }

  return partner;
}
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

1.3. Несохраняемые поля

Как было упомянуто ранее, переменные экземпляра вашего устойчивого компонента подобные переменной `partner` не являются частью устойчивого состояния, а содержат вычисленные результаты. Вы должны объявлять такие переменные как несохраняемые с сериализуемыми вычисленными данными, или как несохраняемые с результатами которые будут пересчитаны "ленивым" способом — при обращении.

Обращайтесь осторожно с переменными содержащими заглушки (stubs) или другие компоненты EJB или домашние интерфейсы EJB: сгенерированные заглушки не предназначены быть мобильными и не расчитаны на работу по сети. Они должны быть заменены другими, устойчивыми к работе по сети ссылками на объекты EJB или домашние интерфейсы `javax.ejb.Handle` и `javax.ejb.HomeHandle` соответственно.Но повторим: вы можете сделать эти переменные несохраняемыми и позволить астральному клону при необходимости получать заглушки.

1.4. Безопасность использования потоков (Thread safety)

Непосредственный вызов реализации компонента вместо вызова через заглушку и контейнер может породить проблемы с использованием потоков. Я предлагаю для решения этих проблем использовать следующий подход:

- Не используйте компонент из множества потоков, и документируйте что использование этого астрального компонента небезопасно для использования в потоках.
- Синхронизируйте критические методы или участки кода. Это не должно отразиться на исполнении компонента в контейнере, за исключением небольшого снижения производительности (по сравнению с накладными расходами добавляемыми контейнером).
- Позвольте `getAstralClone()` возвращать синхронизированную заглушку.
- Неизменный (immutable) объект также безопасен, так как иногда клиент хочет использовать его только для чтения (или неизменный) астральный клон. Простейший способ получить астральный клон только для чтения, это возвращать "обертку" (wrapper) вокруг астрального клона которая блокирует все методы, модифицирующие объект.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

1.5. Синхронизация состояния (State synchronization)

Теперь, когда мы разобрали общие проблемы, мы можем добавить немного функциональности: модификацию оригинального компонента при помощи астрального клона. Вы можете описать метод синхронизации состояния в интерфейсе `EJBPerson`:

```
// синхронизировать состояние с астральным клоном
public void syncAstralClone(PersonBean aClone) throws RemoteException;
```

и реализовать его в `PersonBean` примерно так:

```
public void syncAstralClone(PersonBean aClone) throws RemoteException {

  // проверить содержится ли 'this' ...
  if (this.context == null) {
     throw new RemoteException("должно содержаться 'this'");
  }

  // ... и клон действительно астральный
  if (aClone.context != null) {
     throw new RemoteException("'aClone' должен быть астральным");
  }

  // синхронизировать постоянное состояние
  setName(aClone.name);
  setBirthDay(aClone.birthDay);
}
```

В заключение рассмотрим метод `syncAstralClone(...)`:

- Убедимся что первичный ключ оригинального компонента и астрального клона идентичны
- Вызвать метод `ejbLoad()` если в нем проводится дальнейшая инициализация экземпляра
- Реализовать некоторую степень защиты от таких случаев разноса системы как описывал Томас Дэвис в "Полный сетевой траффик EJB"

2. Заключение

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Астральное копирование элегантно оптимизирует сетевой траффик, оставляя кодирование клиентской части почти неизменным. Клиент взаимодействует с астральным клоном через тот же интерфейс, что и с заглушкой. Далее, одно описание класса содержит всю бизнес логику объекта, включая проверку ввода, в результате чего повышается удобство сопровождения вашей системы.

При работе с астральными клонами у вас могут возникнуть дополнительные пожелания: клиентская реализация домашнего интерфейса могла бы сделать работу с астральными клонами более чем прозрачной. Для этого домашний интерфейс астрального клона должен реализовать определитель путем делегирования удаленному домашнему интерфейсу итератора завернутым в другой итератор, возращающего астральный клон при каждом вызове `next()`. Компонент содержащийся в итераторе может уведомлять домашние интерфейсы астральных компонентов об изменении его состояния так, чтобы домашний интерфейс астрального клона мог синхронизировать состояние астрального клона. Домашний интерфейс астрального клона принимает участие в распределенных транзакциях вписывая себя в эти транзакции в качестве `javax.transaction.xa.XAResource`, после чего синхронизирует состояние исходного компонента с астральными клонами по завершению транзакции.

Эти возможности, конечно, хороши, но последние два подхода довольно трудны для реализации. Я буду рад получить ваши предложения и комментарии по поводу их реализации.

3. Об авторе

Мартин Рес работает системным архитектором в маленькой инновационной компании называемой Virgil в Нидерландах. Он участвует в разработке больших онлайновых систем для казначейства с использованием EJB и JSP. Мартин живет в Амстердаме где он проводит свое свободное время за компьютером пытаясь создать отличные методы оптимизации.

4. Ресурсы

Ресурсы JavaWorld

- EJB 1.1 specifications from Sun:

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

http://java.sun.com/products/ejb/docs.html

- `EntityContext` javadoc: http://java.sun.com/j2ee/j2sdkee/ techdocs/api/javax/ ejb/EntityContext.html
- For a J2EE overview: http://www.javasoft.com/j2ee
- Read Thomas Davis's suggestion for directing network traffic of EJBs in "Direct Network Traffic of EJBs" (*JavaWorld,* November 1999): http://www.javaworld.com/javaworld/ jw-11-1999/jw-11-ejb.html
- For more *JavaWorld* stories on EJBs, go to: http://www.javaworld.com/javaworld/ topicalindex/jw-ti-ejb.html
- Sign up for the *Java in the Enterprise* newsletter for the latest on JavaBeans: http://www.itworld.com/ cgi-bin/subcontent12.cgi

Reprinted with permission from the December 2000 edition of JavaWorld magazine. Copyright © ITworld.com, Inc., an IDG Communications company.

View the original article at: http://www.javaworld.com/javaworld/jw-12-2000/jw-1208-clones.html

Перевод на русский © Сакен Сабден, 2001

Copyright © 1999 - 2007, IT-archiv. All rights reserved.