# Enterprise JavaBeans

Third Edition

*Richard Monson-Haefel*

O'REILLY®

# Enterprise JavaBeans

Третье издание

*Ричард Монсон-Хейфел*

*Санкт-Петербург*
*2002*

Ричард Монсон-Хейфел

# Enterprise JavaBeans, 3-е издание

Перевод И. Васильева


| | |
|---|---|
| Главный редактор | *А. Галунов* |
| Зав. редакцией | *Н. Макарова* |
| Научный редактор | *В. Шальнев* |
| Редактор | *В. Овчинников* |
| Корректура | *С. Беляева* |
| Верстка | *А. Дорошенко* |

*Монсон-Хейфел Р.*
Enterprise JavaBeans, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 672 с., ил.
ISBN 5-93286-041-3


Третье издание «Enterprise JavaBeans» описывает технологию построения сложных ответственных систем, основанную на объединении компонентов, моделирующих прикладные объекты и процессы. EJB берет на себя заботу об объектном постоянстве, безопасности и управлении транзакциями. Еще недавно казалось невероятным, что компоненты EJB смогут не только выполняться в любой операционной системе без какой-либо модификации, но и работать на любом корпоративном сервере EJB. В EJB 2.0 введены компоненты, управляемые сообщениями, способные с помощью службы сообщений JMS взаимодействовать с системами промежуточного уровня; предложена более развитая модель управляемого контейнером постоянства и поддерживаются сложные отношения между объектными компонентами.

В книге рассматриваются: EJB 2.0 и 1.1, сеансовые и объектные компоненты (включая новую модель CMP и язык запросов EJB QL), компоненты, управляемые сообщениями, и служба JMS, XML-дескрипторы развертывания, управление транзакциями и безопасность, взаимосвязь EJB и Java 2, Enterprise Edition. Опытные разработчики корпоративных программных продуктов знают, как сильно технология EJB изменила эту область. Данное третье издание поможет им освоить последние достижения. А для тех, кто не имеет опыта работы с EJB, автор подготовил стартовую площадку, с которой они могут начать изучение этой захватывающей технологии построения прикладных систем.


**ISBN 5-93286-041-3**
**ISBN 0-596-00226-2 (англ)**





Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 15.07.2002. Формат 70×100 1/16 . Печать офсетная.
Объем 42 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

# Оглавление

# Предисловие

## Заметки автора

Зимой 1997 года я консультировал один проект по электронной коммерции, использовавший Java RMI. Не удивительно, что проект провалился, потому что Java RMI не ориентирован на производительность, масштабируемость, надежность, безопасность и транзакции – на все то, что жизненно необходимо производственному окружению. Хотя исход этого проекта не является уникальным для Java RMI, я вижу, что то же самое происходит и с CORBA – время, в которое проводилась реализации этого проекта, было особенно интересным. Примерно в это же время Sun Microsystems впервые представила Enterprise JavaBeans™, и если бы Enterprise JavaBeans стал доступным немного раньше, этот проект, возможно, завершился бы успешно.

Во время работы над тем злополучным проектом я также вел колонку в «JavaReport Online» под названием «The Cutting Edge» (Срезанный угол). Колонка охватывала новые на тот момент технологии Java, такие как Java Naming and Directory Interface™ (JNDI) и JavaMail™ API. Я искал новую тему для третьего выпуска «Срезанного угла», когда обнаружил первый опубликованный черновой проект Enterprise JavaBeans версии 0.8. Первый раз я услышал об этой технологии в 1996 году, но открытая документация стала доступной только в начале 1997. Работая с CORBA, Java RMI и другими технологиями распределенных объектов, я понял, что это хорошая вещь, и сразу же начал свою статью об этой новой технологии.

Кажется, что прошла уже вечность. С тех пор как я опубликовал статью в марте 1998 года, о EJB были написаны буквально сотни статей и уже несколько книг пришло и ушло. Эта книга (уже третье издание) идет в ногу с тремя версиями спецификации EJB много лет. Сейчас, когда выходит очередная новая версия спецификации и появляется множество новых книг на эту тему, я не могу удержаться, чтобы не вспомнить те дни, когда слова «Enterprise JavaBeans» почти у всех вызывали непонимающие взгляды. И я рад, что эти времена прошли навсегда.

# Что такое Enterprise JavaBeans?

Летом 1995 года, после первого представления Java, почти вся индустрия информационных технологий обратила внимание на возможности графического пользовательского интерфейса и конкурентоспособные преимущества, предлагаемые этой технологией с точки зрения переносимости и платформенной независимости. Это было интересное время. Лишь немногие из нас пробовали применять апплеты на стороне сервера. В действительности мы тратили половину нашего времени на программирование, а другую – на то, чтобы убедить руководство, что Java – это не прихоть моды.

Сегодня фокус значительно расширился: язык Java получил признание как отличная платформа для создания корпоративных решений, особенно при разработке распределенных серверных приложений. Такое смещение акцентов произошло благодаря становлению Java как универсального языка для создания независимых от реализации абстракций для всех распространенных корпоративных технологий. Программный интерфейс JDBC – первый и наиболее известный пример. JDBC (Java Database Connectivity) предлагает независимый от производителя Java-интерфейс для доступа к базам данных. Эта абстракция была такой успешной, что сейчас трудно найти производителя реляционных баз данных, который не поддерживал бы JDBC. Абстракции Java значительно расширились и включают JNDI для абстрактных служб каталогов, JTA (Java Transaction API – программный интерфейс транзакций) для абстрактного доступа к менеджерам транзакций, JMS (Java Message Service – служба сообщений) для абстрактного доступа к различным продуктам, ориентированным на сообщения.

Технология Enterprise JavaBeans впервые была представлена в виде рабочей спецификации в конце 1997 года и заявила о себе как о наиболее значительной из всех корпоративных технологий Java, предлагаемых компанией Sun Microsystems. EJB обеспечивает абстракцию для мониторов компонентных транзакций (CTM), представляющих собой сплав двух технологий: традиционных мониторов обработки транзакций (TP), таких как CLCS, TUXEDO и Encina, и служб распределенных объектов, таких как CORBA, DCOM и «родной» для Java RMI. Соединив в себе все самое лучшее из обеих технологий, мониторы компонентных транзакций предлагают мощное объектно-ориентированное окружение, которое упрощает распределенную разработку и при этом автоматически управляет наиболее сложными аспектами корпоративных вычислений, такими как взаимодействие между объектами, управление транзакциями, безопасность, постоянство и параллелизм.

Enterprise JavaBeans определяет модель серверных компонентов, позволяющую разрабатывать прикладные объекты и перемещать их между контейнерами EJB, созданными разными производителями. Ком-

понент (enterprise bean) представляет собой простую программную модель, которая позволяет разработчику сконцентрироваться на стоящей перед ним конкретной проблеме. А сервер EJB отвечает за то, чтобы сделать этот компонент распределенным объектом, и за управляющие службы, такие как транзакции, постоянство, параллелизм и безопасность. Кроме создания бизнес-логики компонента разработчик задает атрибуты поведения компонента во время выполнения, используя подход, похожий на выбор свойств отображения визуальных элементов. Поведение компонента по отношению к транзакциям, постоянству и безопасности может быть задано путем выбора из списка свойств. Конечный результат состоит в том, что EJB значительно облегчает создание систем распределенных компонентов, управляемых мощными транзакционными средами. Для разработчиков и корпоративных информационно-технических отделов, которые борются с проблемами развертывания надежных, производительных распределенных систем на базе CORBA, DCOM и Java RMI, EJB предоставляет в качестве основы для программирования более простую и более производительную платформу.

В 1998 году была завершена спецификация Enterprise JavaBeans 1.0, которая фактически сразу же стала промышленным стандартом. Многие производители объявили о поддержке спецификации еще до ее завершения. С того времени EJB дважды улучшалась. Первый раз спецификация была обновлена в 1999 году до версии 1.1, которая была рассмотрена во втором издании этой книги. Самый последний вариант спецификации – версия 2.0 – описывается в этом, третьем издании «Enterprise JavaBeans», охватывающем также версию EJB 1.1, которая, по большей части, является подмножеством функциональных возможностей, предлагаемых EJB 2.0.

Продукты, соответствующие стандартам EJB, приходят из всех секторов информационной индустрии, в числе которых мониторы обработки транзакций (TPM), CORBA ORB, серверы приложений, реляционные и объектные базы данных, веб-серверы. Некоторые из этих продуктов базируются на собственных моделях, адаптированных под EJB.

Вкратце, обе версии – Enterprise JavaBeans 2.0 и 1.1 – представляют стандартную модель распределенных компонентов, которая значительно упрощает процесс разработки и позволяет компонентам, созданным и развернутым на сервере EJB одного производителя, с легкостью быть развернутыми на сервере другого производителя. Эта книга дает основу, необходимую для создания независимых от производителя EJB-проектов.

# Для кого предназначена эта книга?

Эта книга объясняет и демонстрирует основные принципы архитектур Enterprise JavaBeans 2.0 и 1.1. Хотя EJB сильно упрощает распреде-

ленные вычисления, она достаточно сложна и требует много времени для овладения ею на уровне мастера. Книга понятно и последовательно разъясняет лежащую в основе технологию, классы и интерфейсы Java, компонентную модель и поведение компонентов во время выполнения программы. Она содержит материал, обратно совместимый с EJB 1.1, и включает отдельные замечания и главы, охватывающие наиболее значимые различия между версиями 1.1 и 2.0.

Хотя эта книга сфокусирована на основах, она не предназначена для начинающих. Enterprise JavaBeans – крайне сложная и амбициозная корпоративная технология. Применение EJB может быть достаточно простым, но для ее полного освоения и правильного понимания требуется очень много времени. Перед тем как читать эту книгу вы должны свободно овладеть языком Java и получить практический опыт создания бизнес-проектов. Опыт работы с системами распределенных объектов не требуется, но для того чтобы разобраться с примерами, приведенными в книге, необходимо определенное знание (или хотя бы понимание основ) технологии JDBC. Если вы не знакомы с языком Java, я рекомендую книгу «Learning Java» (Изучаем Java) Патрика Нимайера (Patrick Niemeyer) и Джонатана Кнудсена (Jonathan Knudsen), которая первоначально называлась «Exploring Java». Тем же, кто не знаком с JDBC, рекомендую книгу «Database Programming with JDBC and Java» (Программирование баз данных с помощью JDBC и Java) Джорджа Риза (George Reese), а тем, кому нужны прочные знания распределенных вычислений – «Java Distributed Computing» (Распределенные вычисления на Java) Джима Фарли (Jim Farley).

## Структура книги

Первые три главы содержат общий ознакомительный материал, рассматривающий Enterprise JavaBeans 2.0 и 1.1 в контексте других родственных технологий и объясняющий на максимально отвлеченном уровне, как работает технология EJB и из чего состоят компоненты EJB. В главах с 4 по 13 подробно объяснено, как разрабатываются компоненты различных типов. Главы 14 и 15 могли бы считаться материалом для дополнительного изучения, если бы не транзакции (глава 14), которые существенны для всего, что происходит в корпоративных вычислениях, и стратегии дизайна (глава 15), помогающие справляться с реальными проблемами, влияющими на разработку компонентов. В главе 16 подробно описан дескриптор развертывания на базе XML, используемый в EJB 2.0 и 1.1. Глава 17 дает общий обзор пакета Java 2, Enterprise Edition (J2EE), включающего сервлеты, Java Server Pages (JSP) и EJB. Наконец, три приложения предоставляют справочную информацию, которая может быть вам полезна.

*Глава 1. «Введение»*

Здесь вводится определение мониторов компонентных транзакций и объясняется, каким образом они формируют технологию, лежащую в основе модели компонентов Enterprise JavaBeans.

*Глава 2. «Обзор архитектуры»*

Вводится понятие архитектуры компонентной модели Enterprise JavaBeans и рассматриваются различия между тремя основными типами компонентов: объектными компонентами, сеансовыми компонентами и компонентами, управляемыми сообщениями.

*Глава 3. «Управление ресурсами и основные службы»*

Объясняется, как EJB-совместимый сервер управляет компонентами во время выполнения.

*Глава 4. «Создание вашего первого компонента»*

В этой главе описываются все этапы разработки простого компонента.

*Глава 5. «Клиентское представление»*

Содержит подробный рассказ о том, как удаленное клиентское приложение получает доступ и использует компоненты.

*Глава 6. «EJB 2.0 CMP: Основы постоянства»*

Здесь объясняется, как в EJB 2.0 создать простой объектный компонент, управляемый контейнером.

*Глава 7. «EJB 2.0 CMP: Отношения между объектами»*

В этой главе продолжается тема, начатая в главе 6, расширяя границы вашего понимания от постоянства, управляемого контейнером, до сложных взаимоотношений между компонентами.

*Глава 8. «EJB 2.0 CMP: EJB QL»*

Эта глава обращается к языку запросов Enterprise JavaBeans (EJB Query Language, EJB QL), который применяется для обращения к компонентам и для поиска определенных компонентов при использовании постоянства, поддерживаемого контейнером EJB.

*Глава 9. «EJB 1.1 CMP»*

Описывается постоянство, управляемое контейнером в EJB 1.1, которое для обратной совместимости поддерживается и в EJB 2.0. Прочитайте эту главу, если только вам нужно поддерживать существующие EJB-приложения.

*Глава 10. «Постоянство, реализуемое компонентом»*

Здесь рассказывается о создании компонентов, самостоятельно реализующих постоянство (bean-managed persistence), в том числе о том, когда сохранять, загружать и удалять данные из базы данных.

*Глава 11. «Соглашения между объектом и контейнером»*

Эта глава описывает общий протокол взаимодействия между объектным компонентом и его контейнером во время выполнения, применяемый для реализации постоянства, поддерживаемого контейнером как в версии 2.0, так и в 1.1.

*Глава 12. «Сеансовые компоненты»*

В этой главе показано, как создавать сеансовые компоненты с состоянием и без состояния.

*Глава 13. «Компоненты, управляемые сообщениями»*

Показано, как в EJB 2.0 создавать компоненты, управляемые сообщениями.

*Глава 14. «Транзакции»*

Эта глава дает глубокое объяснение транзакций и описывает модель транзакций, введенную в Enterprise JavaBeans.

*Глава 15. «Стратегии дизайна»*

Даются некоторые основные стратегии дизайна, уменьшающие усилия, требуемые для разработки компонентов и делающие вашу систему EJB более эффективной.

*Глава 16. «XML-дескрипторы развертывания»*

В этой главе представлено подробное описание XML-дескрипторов развертывания, используемых в EJB 1.1 и 2.0.

*Глава 17. «Java 2, Enterprise Edition»*

Приводится обзор J2EE 1.3 и объясняется, как EJB 2.0 встраивается в эту новую платформу.

*Приложение A. «Программный интерфейс Enterprise JavaBeans»*

Данное приложение представляет собой краткий справочник по классам и интерфейсам, определенным в пакетах EJB.

*Приложение B. «Диаграммы состояний и последовательностей»*

Представлены диаграммы, поясняющие жизненный цикл компонентов во время выполнения.

*Приложение C. «Производители EJB»*

В этом приложении приводится информация о производителях серверов EJB.

# Программы и версии

Эта книга описывает Enterprise JavaBeans версий 2.0 и 1.1, включая все дополнительные возможности. В ней используются особенности языка Java из платформы Java 1.2 и JDBC. Ввиду того что данная книга нацелена на создание компонентов Enterprise JavaBeans и проек-

тов, не зависящих от конкретного производителя, я старался избегать нестандартных расширений и диалектов, специфичных для определенного производителя. Для чтения этой книги достаточно любого совместимого с EJB сервера, но работа с примерами требует знания процедур установки, развертывания и управления сервером во время выполнения.

EJB 2.0 и 1.1 имеют много общего, но там, где они отличаются, главы или разделы глав, специфичные для каждой версии, выделены особо. Можете пропускать те разделы, которые вас не касаются. Если не указано обратное, исходные коды из этой книги будут работать как для EJB 2.0, так и для EJB 1.1.

Примеры, приведенные в этой книге, доступны на сайте *ftp://ftp.oreilly.com/pub/examples/java/ejb*. Примеры сгруппированы по главам.

# Рабочие книги с примерами

Хотя EJB-приложения сами по себе переносимы, способы установки и запуска EJB-продуктов варьируются от производителя к производителю. По этой причине не представлялось возможным охватить все доступные EJB-продукты и был выбран радикальный, но в то же время эффективный способ, позволяющий рассмотреть эти различия, – рабочие упражнения (workbooks).

Для того чтобы помочь читателям установить примеры из книги на различные EJB-продукты, я опубликовал несколько бесплатных рабочих упражнений, которые можно использовать вместе с этой книгой для запуска примеров на конкретных коммерческих и некоммерческих серверах. Рабочие книги для конкретного продукта будут относиться к самой последней версии сервера. Так, если производитель поддерживает EJB 2.0, пример из рабочей книги будет использовать особенности EJB 2.0. С другой стороны, если производитель поддерживает только EJB 1.1, примеры будут специфичными для версии 1.1.

Я планирую опубликовать рабочие книги для нескольких различных серверов EJB, и по крайней мере две из них будут доступны в самое ближайшее время. Эти книги будут бесплатно доступны (в формате PDF) на *http://www.oreily.com/catalog/entjbeans3/* и *http://www.titanbooks.com*.

# Соглашения

В книге приняты следующие типографские соглашения:

*Курсив*

Предназначен для имен файлов и путей, имен хостов, доменов, URL и адресов электронной почты. Курсивом также выделяются вновь вводимые термины.

`Моноширинный`

Используется в примерах программ и их фрагментов, для элементов XML, тегов, команд SQL, имен таблиц и колонок. Моноширинный шрифт применяется также для имен классов, переменных методов и для ключевых слов Java, используемых в тексте.

**`Моноширинный полужирный`**

Служит для выделения в некоторых примерах программ.

*`Моноширинный курсив`*

Указывает на то, что текст должен быть заменен. Например, в слове *`BeanName`*`PK` следует заменить *`BeanName`* конкретным именем.

Указывает на подсказки, советы, общие замечания.

Означает предупреждение или предостережение.

Компонент Enterprise JavaBeans состоит из нескольких частей. Это не один объект, а набор объектов и интерфейсов. Для ссылки на компонент как на единое целое мы будем использовать его прикладное имя, набранное основным шрифтом. Например, будем ссылаться на компонент Customer тогда, когда нам нужно обозначить компонент вообще. Если мы набираем имя моноширинным шрифтом, значит, мы явно ссылаемся на удаленный интерфейс. Таким образом, `CustomerRemote` представляет собой удаленный интерфейс, определяющий прикладные методы компонента Customer.

# Комментарии и вопросы

Пожалуйста, направляйте комментарии и вопросы, касающиеся этой книги, в издательство:

O'Reilly & Associaties, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (для США и Канады)
(707) 829-0515 (международный и местный)
(707) 829-0104 (факс)

У этой книги есть веб-страница, содержащая найденные опечатки, примеры и некоторую дополнительную информацию. Ее можно найти по адресу:

*http://www.orielly.com/catalog/entjbeans3/*

Комментарии и технические вопросы направляйте по адресу:

*bookquestions@orielly.com*

За дополнительной информацией о книгах, конференциях, программных продуктах, центрах ресурсов и сети O'Reilly обращайтесь на вебсайт O'Reilly:

*http://www.orielly.com*

Автор поддерживает сайт для дискуссий по EJB и связанным с ней распределенным компьютерным технологиям на *http://www.jmiddleware.com*. *jMiddleware.com* предлагает последние новости по этой книге, а также подсказки по программированию, статьи и большой список ссылок на ресурсы EJB.

# Благодарности

Хотя на обложку этой книги вынесено только одно имя, к ее написанию и распространению приложили усилия многие люди. Успех каждого издания этой книги зависел в основном от Майкла Лукидеса (Michael Loukides). Без его опыта, ловкости и руководства этой книги просто не было бы. Другие люди из O'Reilly & Associates, включая Рэчел Уилер (Rachel Wheeler), Роба Романо (Rob Romano), Кайла Харта (Kyle Hart) и многих других, также много сделали для успеха этой книги.

Многие технические редакторы помогли обеспечить то, что материал о сущности Enterprise JavaBeans был технически точным и правильным. Отдельно нужно упомянуть Грега Найберга (Greg Nyberg) из Object Partners, Хеманта Ханделвала (Hemant Khandelwal) из Pramati, Кайла Брауна (Kyle Brown) из IBM, Роберта Кастанеда (Robert Castaneda) из Custom Ware, Джоя Фиалли (Joy Fially) из Sun Microsystems, Анила Шарму (Anila Sharma) и Прайенка Растджи (Priyank Rastogi) из Pramanti, Сева Уйта (Seth White) из BEA, Эвана Ирланда (Evan Ireland) и Майера Тануана (Meyer Tanuan) из Subase, Дэвида Чаппела (David Chappell) из David Chappell & Associates, Джима Фарли (Jim Farley) – автора «Java Distributed Computing» (O'Reilly) и Прасада Муппиралу (Prasad Muppirala) из BORN Information Services. Они во многом способствовали обеспечению технической точности книги и вложили свой профессиональный и жизненный опыт, помогая сделать ее лучшей книгой по Enterprise JavaBeans из выпускаемых сегодня.

Я хотел бы поблагодарить всех тех, кто присылал ценные замечания по книге во время ее написания по адресу *http://orielly.techrev.org*, включая (в алфавитном порядке):

Диона Алмаера (Dion Almaer), Джима Арчера (Jim Archer), Стефана Дейвиса (Stephen Davis), Джона Де Ла Круза (John De La Cruz), Тома Гулло (Tom Gullo), Марти Харви (Marty Harvey), Хая Хонга (Hai Hoang), Мирея Куннумпурата (Meeraj Kunnumpurath), Тома Ларсона (Tom Larson), Бьерна Расмуссена (Bjarne Rassmussen), rgparker (имя неизвестно), Лари Сельцера (Larry Seltzer) и Курта Смита (Curt Smith).

Отдельные благодарности Шрираму Шринивасону (Sriram Srinivason) из BEA, Анне Томас (Anne Tomas) из Sun Microsystems, Яну Маккалиону (Ian McCallion) из IBM Hursley, Тиму Рохайли (Tim Rohaly) из jGuru.com, Джеймсу Френтрессу (James D Frentress) из корпорации ITM, Анджее Джан Тарамину (Andrzej Jan Taramina) из Accredo Systems, Марку Лою (Marc Loy), соавтору книги «Java Swing», Дону Уэйсу (Don Weiss) из Step 1, Майку Слину (Mike Slinn) из корпорации The Dialog и Кэвину Дику (Kevin Dick) из Kevin Dick & Associates. Помощь этих технических экспертов была очень важна для технической и принципиальной точности первых изданий этой книги. Кроме этого, я хотел бы поблагодарить Мегги Мескита (Maggie Mezquita), Грэга Хартсела (Greg Hartzel), Джона Клуга (John Klug) и Яна Джамсу (Jon Jamsa) из BORN Information, которые просмотрели первый вариант первого издания и составили ценные замечания.

Также хочу поблагодарить Влада Матену (Vlad Matena) и Марка Хапнера (Mark Hapner) из Sun Microsystem, главных архитекторов Enterprise JavaBeans, Линду Де Мишель (Linda DeMichiel), руководителя спецификации EJB 2.0, и Бони Кельта (Bonnie Kellett), менеджера программы J2EE, за их желание ответить на мои самые сложные вопросы. Благодарю всех участников списка рассылки EJB-INTEREST, поддерживаемого компанией Sun Microsystems за их интересные и иногда противоречивые, но всегда информативные послания за последние четыре года.

И наконец, выражаю самую искреннюю благодарность моей жене Холли за помощь и поддержку в течение трех лет мучительных исследований и написания всего того, что потребовалось для создания трех изданий этой книги. Без ее неизменной поддержки и любви эта книга не была бы завершена.

# 9

# EJB 1.1 CMP

## Замечание для тех, кто работает с EJB 2.0

Постоянство, управляемое контейнером, подверглось в EJB 2.0 кардинальному изменению, которое не является обратно совместимым с EJB 1.1. По этой причине производитель EJB 2.0 вынужден поддерживать обе модели постоянства, управляемого контейнером, – и EJB 2.0 и EJB 1.1. Модель EJB 1.1 поддерживается только для обратной совместимости, для того чтобы разработчики могли безболезненно переносить существующие приложения на новую платформу EJB 2.0. Ожидается, что все новые объектные компоненты и новые прикладные программы будут использовать постоянство, управляемое контейнером, версии EJB 2.0, а не EJB 1.1. Хотя постоянство EJB 1.1 рассмотрено в этой книге, следует избегать его применения, если только вам не приходится поддерживать существующие системы EJB 1.1. Постоянство EJB 2.0 рассмотрено в главах 6–8.

Постоянство, управляемое контейнером, в EJB 1.1 ограничено в нескольких отношениях. Например, у компонентов EJB 1.1 CMP могут быть только удаленные интерфейсы, и не может быть ни локальных, ни локальных внутренних интерфейсов. Важнее всего то, что в EJB 1.1 управляемые контейнером объектные компоненты не поддерживают отношения. Существуют также и другие мелкие отличия, делающие EJB 1.1 CMP ограниченным по сравнению с EJB 2.0 CMP. Например, методы `ejbCreate()` и `ejbPostCreate()` в EJB 1.1 не поддерживают суффикс *<ИМЯ-МЕТОДА>*, разрешенный в EJB 2.0, что делает перегрузку ме-

тодов более сложной. EJB 2.0 CMP представляет собою большой шаг вперед по сравнению с CMP 1.1 и должно использоваться везде, где это возможно.

# Обзор для тех, кто работает с EJB 1.1

Следующий обзор постоянства, управляемого контейнером, версии EJB 1.1 в значительной степени повторяет сказанное в главе 6, но для тех, кто работает с EJB 1.1 и не читал главу 6, данный обзор важен, т. к. поможет им понять принципы, связанные с объектными компонентами и с постоянством, управляемым контейнером.

В главе 4 мы начали разрабатывать несколько простых компонентов, пропуская массу деталей, относящихся к их созданию. Здесь же мы приведем полный обзор процесса создания объектных компонентов.

Объектные компоненты моделируют прикладные понятия, которые могут быть выражены существительными. Это скорее практическое правило, чем требование, но оно помогает определить, когда прикладное понятие следует реализовывать в виде объектного компонента. Со средней школы нам известно, что существительные – это слова, которые описывают человека, место или предмет. Понятия «человек» и «место» достаточно очевидны: компонент «Человек» может представлять клиента или пассажира, а компонент «Место» – город или пункт назначения. Точно так же объектные компоненты обычно представляют «предметы»: реальные объекты, такие как суда, кредитные карточки и т. д. Объектный компонент может даже представлять нечто довольно абстрактное, вроде заказа билетов. Объектные компоненты описывают и состояние, и поведение реальных объектов и позволяют разработчикам включать в них данные и прикладные правила, связанные с отдельными понятиями. К примеру, компонент Customer содержит данные и прикладные правила, связанные с клиентом. Это делает возможным логичное и безопасное связывание данных с понятием, которым необходимо манипулировать.

В примере с туристической фирмой «Титан» мы можем выделить сотни прикладных понятий, которые обозначаются существительными и поэтому могут быть естественно смоделированы объектными компонентами. Мы уже видели простой компонент Cabin в главе 4, а здесь мы создадим компонент Ship (Корабль). Очевидно, что в примере могли бы использоваться компоненты Customer, Cruise, Reservation и многие другие. Каждое из этих прикладных понятий представляет данные, которые требуется отслеживать и которыми, возможно, необходимо управлять.

Элементы представляют данные в базе данных, поэтому изменения объектного компонента приводят к изменениям в ней. Имеется много причин для того, чтобы предпочесть объектные компоненты непо-

средственному доступу к базе данных. Применение объектных компонентов для хранения данных дает программистам более простой механизм для доступа и изменения данных. Значительно проще, например, поменять имя клиента, вызвав метод `Customer.setName()`, чем выполнять для базы данных команду SQL. Кроме того, хранение данных с помощью объектных компонентов упрощает повторное использование программы. Как только объектный компонент определен, он логичным образом может использоваться во всей системе «Титан». Понятием судна, например, можно оперировать во многих областях системы «Титан», включая регистрацию, планирование расписаний и маркетинг. Компонент Ship предоставляет «Титану» законченный способ доступа к информации о судне. Таким образом гарантируется логичность и простота доступа к информации. Представление данных в виде объектных компонентов может сделать разработку более легкой и более рентабельной.

Во время создания нового компонента в базу данных должна быть вставлена новая запись, а экземпляр компонента необходимо связать с этими данными. В процессе работы с компонентом его состояние меняется, а изменения необходимо синхронизировать с данными в базе данных: записи должны вставляться, модифицироваться и удаляться. Этот процесс координации данных, представляющих экземпляр, и базы данных называется *постоянством* (*persistence*).

Существуют два основных типа объектных компонентов, различающихся тем, как они управляют постоянством. Компоненты с *постоянством, управляемым контейнером*, предполагают, что их постоянство будет автоматически поддерживаться контейнером. Контейнер знает, как поля постоянства экземпляра и его отношения проецируются на базу данных, и автоматически вставляет, модифицирует и удаляет данные, связанные с объектами, из базы данных. Объектные компоненты, *самостоятельно реализующие постоянство*, выполняют всю эту работу вручную: разработчик компонента должен написать код для манипулирования базой данных. Контейнер EJB сообщает экземпляру компонента, когда безопасно вставлять, модифицировать и удалять его данные из базы данных, но не предоставляет никакой другой помощи. Экземпляр компонента делает всю работу по обеспечению своего постоянства самостоятельно. Постоянство, управляемое компонентом, рассматривается в главе 10.

# Постоянство, управляемое контейнером

Развертывая объектный компонент EJB 1.1 CMP, необходимо указать поля компонента, которые будут управляться контейнером, и то, как они должны сопоставляться с базой данных. Затем контейнер автоматически генерирует код, необходимый для сохранения состояния экземпляра компонента.

Поля, связываемые с базой данных, называются полями, управляемыми контейнером. EJB 1.1 не поддерживает поля отношений, как это сделано в EJB 2.0. Управляемые контейнером поля могут иметь любой примитивный тип Java или сериализуемого объекта. Большинство компонентов при реализации постоянства с помощью реляционной базы данных используют примитивные типы Java, поскольку связать примитивы Java с типами данных реляционной базы данных значительно проще.

EJB 1.1 позволяет также реализовывать в виде управляемых контейнером полей ссылки на другие компоненты. Для того чтобы это работало, производитель EJB должен обеспечить автоматическое преобразование ссылки (на тип удаленного или внутреннего интерфейса) в нечто, что может быть сохранено в базе данных, а затем преобразовано обратно в удаленную ссылку. Производителям обычно приходится конвертировать удаленные ссылки в первичные ключи, объекты `Handle` и `HomeHandle` или некоторые другие собственные типы указателей, которые могут применяться для хранения ссылки на компонент в базе данных. Контейнер автоматически будет управлять этим преобразованием удаленной ссылки в постоянный указатель и обратно. В EJB 2.0 CMP эта особенность была заменена управляемыми контейнером полями отношений.

Преимущество постоянства, управляемого контейнером, состоит в том, что компонент может быть разработан независимо от того, в какой базе данных будет храниться его состояние. Управляемые контейнером компоненты могут получать доступ к услугам и реляционных и объектно-ориентированных баз данных. Состояние компонента описывается независимо от них, что делает компонент более гибким и пригодным для многократного использования.

Недостаток управляемых контейнером компонентов состоит в том, что они требуют сложных инструментальных средств для описания связи полей компонентов с базой данных. В некоторых случаях потребуется просто сопоставить каждое поле в экземпляре компонента со столбцом базы данных или – в случае сериализации компонента – с файлом. В других случаях это может быть более сложно. Например, состояние некоторых компонентов может быть задано с помощью сложного объединения в реляционной базе данных или отображения на некоторую существующую систему, такую как CICS или IMS.

Здесь мы создадим новый управляемый контейнером объектный компонент Ship (Корабль), который мы изучим подробно. Компонент Ship также фигурирует в главе 7 при обсуждении сложных отношений в EJB 2.0 и в главе 10 при обсуждении постоянства, управляемого компонентом. Одолев эту главу, вы можете сравнить компонент Ship, разработанный здесь, с компонентами, созданными в главах 7 и 10.

Давайте начнем думать о том, что же мы хотим сделать. Огромное количество данных вошло бы в полное описание судна, но для наших

целей мы ограничимся небольшим набором информации. Сейчас мы можем сказать, что судно имеет следующие характеристики (или атрибуты): название, вместимость и тоннаж (т. е. размер). Компонент Ship будет инкапсулировать эти данные, поэтому нам придется создать в нашей базе данных таблицу SHIP для их хранения.

Приведем определение таблицы SHIP с помощью стандартного SQL:

```
CREATE TABLE SHIP (ID INT PRIMARY KEY NOT NULL, NAME CHAR(30), CAPACITY INT,
TONNAGE DECIMAL(8,2))
```

Создание любого компонента мы начинаем с программирования удаленных интерфейсов. Это акцентирует наше внимание на наиболее важном аспекте компонента – его прикладной цели. Определив интерфейсы, мы можем начать работу над действительным определением компонента.

## Удаленный интерфейс

Нам понадобится удаленный интерфейс для компонента Ship. В этом интерфейсе определяются прикладные методы, используемые клиентами для взаимодействия с компонентом. Описывая удаленный интерфейс, примем в расчет все отдельные части системы «Титан», которым может потребоваться понятие судна. Здесь приведен удаленный интерфейс ShipRemote компонента Ship:

```
package com.titan.ship;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface ShipRemote extends javax.ejb.EJBObject {
    public String getName() throws RemoteException;
    public void setName(String name) throws RemoteException;
    public void setCapacity(int cap) throws RemoteException;
    public int getCapacity() throws RemoteException;
    public double getTonnage() throws RemoteException;
    public void setTonnage(double tons) throws RemoteException;
}
```

## Удаленный внутренний интерфейс

Удаленный внутренний интерфейс любого объектного компонента применяется для создания, поиска и удаления объектов из системы EJB. У каждого типа объектных компонентов есть свой собственный внутренний интерфейс. Во внутреннем интерфейсе определяются два основных типа методов: несколько необязательных конструирующих методов и один или несколько поисковых.[1] Конструирующие методы

[1] В главе 15 объясняется, когда следует определять конструирующие методы во внутреннем интерфейсе.

действуют в качестве удаленных конструкторов и определяют, как будут создаваться новые компоненты Ship. (В нашем внутреннем интерфейсе мы ввели только один конструирующий метод.) Поисковый метод предназначен для поиска одного или нескольких компонентов Ship.

Следующий код содержит полное определение интерфейса `ShipHomeRemote`.

```
package com.titan.ship;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import java.util.Enumeration;

public interface ShipHomeRemote extends javax.ejb.EJBHome {

    public ShipRemote create(Integer id, String name,
        int capacity, double tonnage)
        throws RemoteException,CreateException;
    public ShipRemote create(Integer id, String name)
        throws RemoteException,CreateException;
    public ShipRemote findByPrimaryKey(Integer primaryKey)
        throws FinderException, RemoteException;
    public Enumeration findByCapacity(int capacity)
        throws FinderException, RemoteException;
}
```

Enterprise JavaBeans определяет, что конструирующие методы внутреннего интерфейса должны генерировать исключение `javax.ejb.CreateException`. В случае постоянства, управляемого контейнером, контейнеру требуется одно общее исключение для индикации проблем со связью, возникающих во время создания компонента.

## Поисковые методы

В EJB 1.1 CMP поддерживаются только поисковые методы, поддержка методов выборки EJB 2.0 не реализована. Кроме того, поисковые методы поддерживаются только в удаленном внутреннем интерфейсе: объектные компоненты EJB 1.1 не поддерживают локальные компонентные интерфейсы.

При использовании постоянства EJB 1.1, управляемого контейнером, реализация поисковых методов автоматически генерируется во время развертывания. Разные производители контейнеров EJB руководствуются различными подходами, определяя, как будут работать поисковые методы. Независимо от реализации во время развертывания компонента нам потребуется выполнить некоторую работу по определению правил для поискового метода. `findByPrimaryKey()` представляет собою стандартный метод, который должны поддерживать все внутренние интерфейсы объектных компонентов. Этот метод ищет компо-

ненты, основываясь на атрибутах первичного ключа. В случае с компонентом Ship первичным ключом является класс `Integer`, соответствующий полю `id` класса `ShipBean`. Для реляционных баз данных атрибуты первичного ключа обычно соответствуют первичному ключу таблицы. К примеру, в классе `ShipBean` атрибут `id` соответствует столбцу первичного ключа `ID` таблицы `SHIP`. В объектно-ориентированной базе данных атрибуты первичного ключа могли бы указывать на какой-то другой уникальный идентификатор.

EJB 1.1, в дополнение к методу `findByPrimaryKey()`, позволяет определять во внутреннем интерфейсе другие поисковые методы. Всем поисковым методам следует давать имена, соответствующие образцу `find<`*`СУФФИКС`*`>()`. Так, если бы мы захотели включить поисковый метод, ищущий суда заданной вместимости, то могли бы назвать его `findByCapacity(int capacity)`. В постоянстве, управляемом контейнером, контейнер должен быть «познакомлен» с каждым поисковым методом, входящим во внутренний интерфейс. Другими словами, управляющий развертыванием должен способом, понятным контейнеру, указать, как этот поисковый метод будет работать. Это делается во время развертывания с помощью инструментальных средств, с использованием уникального для конкретного производителя синтаксиса.

Поисковые методы возвращают либо подходящий для этого компонента тип удаленного интерфейса, либо экземпляр с типом `java.util.Enumeration`, либо `java.util.Collection`. В отличие от EJB 2.0 CMP, EJB 1.1 CMP не поддерживает в качестве типа возвращаемого значения для поисковых методов тип `java.util.Set`.

Указание типа удаленного интерфейса говорит о том, что метод ищет только один компонент. Очевидно, что метод `findByPrimaryKey()` возвращает одиночную удаленную ссылку, поскольку между значением первичного ключа и объектом устанавливается отношение «один-к-одному». Однако метод `findByCapacity(int capacity)` мог бы возвращать несколько удаленных ссылок, по одной для каждого судна, имеющего вместимость, указанную параметром capacity. Для того чтобы иметь возможность возвращать несколько удаленных ссылок, поисковый метод должен возвращать либо тип `Enumeration`, либо `Collection`. Enterprise JavaBeans определяет, что все поисковые методы, применяемые во внутреннем интерфейсе, должны генерировать исключение `javax.ejb.FinderException`. Поисковые методы, возвращающие одиночную удаленную ссылку, генерируют `FinderException`, если происходит прикладная ошибка, и исключение `javax.ejb.ObjectNotFound`, если соответствующий компонент не найден. `ObjectNotFoundException` представляет собой подтип `FinderException` и генерируется *только* поисковыми методами, возвращающими одиночные удаленные ссылки. Поисковые методы, возвращающие типы `Enumeration` или `Collection` (многообъектные поисковые методы), если не найден ни один соответствующий компонент, возвращают пустую коллекцию (а не нулевую

ссылку), а в случае возникновения прикладной ошибки возбуждают исключение `FinderException`.

Способ, которым поисковые методы в постоянстве, управляемом контейнером, будут обращаться к базе данных, не определен в спецификации EJB 1.1 – он специфичен для каждого производителя. Для того чтобы определить, как во время развертывания описываются поисковые методы, обратитесь к документации по вашему серверу EJB. В отличие от CMP в EJB 2.0, здесь нет никакого стандартного языка запросов для задания поведения поисковых методов во время выполнения.

## Первичный ключ

Первичный ключ – это объект, уникально идентифицирующий объектный компонент в соответствии с типом компонента, внутренним интерфейсом и контекстом контейнера, в котором он используется. В постоянстве, управляемом контейнером, первичным ключом может являться сериализуемый объект, определенный разработчиком специально для этого компонента, или его определение может быть отложено до начала процесса развертывания. Первичный ключ определяет атрибуты, используемые для поиска отдельного компонента в базе данных. В данном случае нам нужен только один атрибут (`id`), но для первичного ключа возможно наличие нескольких атрибутов, каждый из которых уникально идентифицирует данные компонента. Мы рассмотрим первичные ключи подробно в главе 11, а пока зададим, что компонент Ship использует простой первичный ключ с единственным значением с типом `java.lang.Integer`.

## Класс ShipBean

Компонент не считается законченным без реализации его класса. Итак, мы определили удаленные интерфейсы компонента Ship и первичный ключ и готовы определить непосредственно `ShipBean`. `ShipBean` постоянно находится на сервере EJB. Когда клиентское приложение или компонент вызывает прикладной метод удаленного интерфейса компонента Ship, то этот вызов попадет к компонентному объекту, который затем перенаправит его экземпляру `ShipBean`.

При разработке любого компонента мы должны использовать удаленные интерфейсы компонента в качестве руководства. Прикладные методы, определенные в удаленном интерфейсе, должны быть продублированы в классе компонента. Согласно спецификации EJB 1.1, в управляемых контейнером компонентах конструирующие методы внутреннего интерфейса также должны иметь в классе компонента соответствующие им методы. Наконец, методы обратного вызова, содержащиеся в интерфейсе `javax.ejb.EntityBean`, обязательно должны быть реализованы. Далее приведен код класса `ShipBean`:

```
package com.titan.ship;

import javax.ejb.EntityContext;

public class ShipBean implements javax.ejb.EntityBean {
    public Integer id;
    public String name;
    public int capacity;
    public double tonnage;

    public EntityContext context;

    public Integer ejbCreate(Integer id, String name, int capacity, double
tonnage) {
        this.id = id;
        this.name = name;
        this.capacity = capacity;
        this.tonnage = tonnage;
        return null;
    }

    public Integer ejbCreate(Integer id, String name) {
        this.id = id;
        this.name = name;
        capacity = 0;
        tonnage = 0;
        return null;
    }

    public void ejbPostCreate(Integer id, String name, int capacity,
        double tonnage) {
        Integer pk = (Integer)context.getPrimaryKey();
          // Используем первичный ключ
    }

    public void ejbPostCreate(int id, String name) {
        ShipRemote myself = (ShipRemote)context.getEJBObject();
          // Используем ссылку на компонентный объект
    }
    public void setEntityContext(EntityContext ctx) {
        context = ctx;
    }
    public void unsetEntityContext() {
        context = null;
    }
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbRemove() {}

    public String getName() {
        return name;
    }
    public void setName(String n) {
```

```
        name = n;
    }
    public void setCapacity(int cap) {
        capacity = cap;
    }
    public int getCapacity() {
        return capacity;
    }
    public double getTonnage() {
        return tonnage;
    }
    public void setTonnage(double tons) {
        tonnage = tons;
    }
}
```

В компоненте Ship определено четыре поля постоянства: `id`, `name`, `capacity` и `tonnage`. Эти поля представляют собой состояние постоянства компонента Ship; они определяют уникальный объект Ship в базе данных. Кроме этого, компонент Ship определяет еще одно поле, `context`, содержащее `EntityContext` компонента. Позднее мы поговорим о нем подробно.

Методы set и get – это прикладные методы, определенные нами для компонента Ship; и удаленный интерфейс, и класс компонента должны их поддерживать. Это означает, что сигнатуры этих методов должны быть точно такие же, за исключением `javax.ejb.RemoteException`. Прикладным методам класса компонента не требуется генерировать `RemoteException`. Это имеет смысл, поскольку данные методы фактически не вызываются удаленно – они вызываются компонентным объектом. В случае проблем со связью исключение `RemoteException` будет автоматически сгенерировано контейнером для компонента.

## Реализация интерфейса javax.ejb.EntityBean

Будучи объектным компонентом, компонент Ship должен реализовать интерфейс `javax.ejb.EntityBean`. Интерфейс `EntityBean` содержит ряд методов обратного вызова, используемых контейнером для того, чтобы во время выполнения информировать экземпляр компонента о различных событиях:

```
public interface javax.ejb.EntityBean extends javax.ejb.EnterpriseBean {
    public abstract void ejbActivate() throws RemoteException;
    public abstract void ejbPassivate() throws RemoteException;
    public abstract void ejbLoad() throws RemoteException;
    public abstract void ejbStore() throws RemoteException;
    public abstract void ejbRemove() throws RemoteException;
    public abstract void setEntityContext(EntityContext ctx)
        throws RemoteException;
    public abstract void unsetEntityContext() throws RemoteException;
}
```

Каждый метод обратного вызова вызывается в определенное время в течение жизненного цикла экземпляра `ShipBean`. В большинстве случаев управляемые контейнером компоненты (такие как компонент Ship) не должны ничего делать, когда происходит вызов метода обратного вызова. Постоянство управляемых контейнером компонентов реализуется автоматически; большинство ресурсов и логики, которые могли бы обрабатываться в этих методах, уже обработаны контейнером.

Текущая версия компонента Ship содержит пустые реализации его методов обратного вызова. Однако следует заметить, что даже управляемый контейнером компонент может при необходимости пользоваться услугами методов обратного вызова; нам они в классе `ShipBean` в настоящее время просто не нужны. Методы обратного вызова подробно рассмотрены в главе 11. Прочитайте эту главу, чтобы больше узнать о методах обратного вызова и о том, когда они вызываются.

## Конструирующие методы

Когда вызывается метод `create()` внутреннего интерфейса, внутренний объект перенаправляет его экземпляру компонента таким же образом, как это происходит с прикладными методами удаленного интерфейса. Это значит, что в классе компонента нам нужен метод `ejbCreate()`, который соответствует каждому методу `create()` внутреннего интерфейса.

Метод `ejbCreate()` возвращает нулевое значение типа `Integer` первичного ключа компонента. Значение, возвращаемое методом `ejbCreate()` управляемого контейнером компонента, фактически игнорируется контейнером.

В EJB 1.1 возвращаемое значение метода `ejbCreate()` было изменено с `void`, использованного в EJB 1.0, на тип первичного ключа для того, чтобы облегчить наследование, – это помогает компоненту, самостоятельно реализующему постоянство, расширять компонент, управляемый контейнером. В EJB 1.0 из-за того, что возвращаемое значение имело тип `void`, это не было возможно: Java не даст вам перегрузить методы с разными возвращаемыми значениями. Изменив это определение так, чтобы компонент, управляющий собой самостоятельно, мог расширять управляемый контейнером компонент, спецификация EJB 1.1 дала возможность производителям поддерживать постоянство, управляемое контейнером, путем расширения управляемого контейнером компонента с помощью сгенерированного компонента, реализующего постоянство самостоятельно, – довольно простое решение трудной задачи. Разработчики компонента также могут воспользоваться преимуществами наследования, чтобы изменить существующий CMP-компонент на BMP-компонент, который может потребоваться для решения сложных задач, связанных с постоянством.

Для каждого метода create(), определенного во внутреннем интерфейсе компонента, в классе экземпляра компонента должен существовать соответствующий метод ejbPostCreate(). Другими словами, методы ejbCreate() и ejbPostCreate() встречаются парами с совпадающими сигнатурами. Для каждого метода create(), определенного во внутреннем интерфейсе, должна существовать одна пара.

## ejbCreate () и ejbPostCreate ()

В управляемом контейнером компоненте метод ejbCreate() вызывается непосредственно перед записью управляемых контейнером полей компонента в базу данных. Значения, переданные в метод ejbCreate(), должны использоваться для инициализации полей экземпляра компонента. После завершения метода ejbCreate() к базе данных добавляется новая запись, основанная на управляемых контейнером полях.

Разработчик компонента должен обеспечить, чтобы метод ejbCreate() устанавливал поля постоянства, соответствующие полям первичного ключа. Если для управляемого контейнером компонента определен составной первичный ключ, в нем должны быть определены поля, соответствующие одному или нескольким управляемым контейнером полям (постоянства) в классе компонента. Эти поля должны точно соответствовать и по типу, и по имени. Во время выполнения контейнер предполагает, что поля в первичном ключе соответствуют нескольким или всем полям в классе компонента. При создании нового компонента контейнер будет использовать управляемые контейнером поля в классе компонента для автоматического создания и заполнения первичного ключа компонента.

После того как состояние компонента заполнено и установлен его EntityContext, вызывается метод ejbPostCreate(). Этот метод дает компоненту возможность выполнить любую постобработку перед обслуживанием клиентских запросов. Уникальность компонента в течение запроса ejbCreate() еще не реализована, но она становится доступной в методе ejbPostCreate(). Это означает, что компонент может обращаться к его собственному первичному ключу и компонентному объекту, который может быть полезен для инициализации экземпляра компонента перед обслуживанием вызовов прикладных методов. Метод ejbPostCreate() может использоваться для выполнения любой дополнительной инициализации. У каждого метода ejbPostCreate() должны быть такие же параметры, как и у соответствующего ему метода ejbCreate(). Метод ejbPostCreate() возвращает void.

Дополнительную информацию о методах ejbCreate() и ejbPostCreate() и о том, как они связаны с жизненным циклом объектных компонентов, можно найти в главе 11.

### Использование ejbLoad() и ejbStore() в управляемых контейнером компонентах

Процесс обеспечения эквивалентности записи базы данных и экземпляра объектного компонента называется *синхронизацией* (*synchronization*). В постоянстве, управляемом контейнером, управляемые контейнером поля компонента синхронизируются с базой данных автоматически. В большинстве случаев нам не потребуются методы `ejbLoad()` и `ejbStore()`, поскольку постоянство в управляемых контейнером компонентах достаточно простое. Эти методы более подробно рассмотрены в главе 11.

## Дескриптор развертывания

Закончив с определением компонента Ship, включающим удаленный и внутренний интерфейсы, мы готовы создать дескриптор развертывания. В следующем коде показан XML-дескриптор развертывания компонента. Особенно в нем важен элемент `<cmp-field>`. Эти элементы перечисляют поля, управляемые контейнером. Они имеют то же значение, что и в постоянстве, управляемом контейнером, в EJB 2.0:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
    <enterprise-beans>
        <entity>
            <description>
                Этот компонент представляет корабль.
            </description>
            <ejb-name>ShipEJB</ejb-name>
            <home>com.titan.ship.ShipHomeRemote</home>
            <remote>com.titan.ship.ShipRemote</remote>
            <ejb-class>com.titan.ship.ShipBean</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.Integer</prim-key-class>
            <reentrant>False</reentrant>
            <cmp-field><field-name>id</field-name></cmp-field>
            <cmp-field><field-name>name</field-name></cmp-field>
            <cmp-field><field-name>capacity</field-name></cmp-field>
            <cmp-field><field-name>tonnage</field-name></cmp-field>
            <primkey-field>id</primkey-field>
        </entity>
    </enterprise-beans>

    <assembly-descriptor>
        <security-role>
            <description>
            Эта роль представляет всех, кому разрешен полный доступ к Ship EJB.
            </description>
```

```
            <role-name>everyone</role-name>
        </security-role>

        <method-permission>
            <role-name>everyone</role-name>
            <method>
                <ejb-name>ShipEJB</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>

        <container-transaction>
            <method>
                <ejb-name>ShipEJB</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Элементы `<cmp-field>` перечисляют все управляемые контейнером поля класса объектного компонента. Это те поля, которые будут сохранены в базе данных и которые управляются контейнером во время выполнения.

📖 Рабочее упражнение 9.1. Объектный компонент в CMP 1.1