Ю.Д.Заковряшин

Программирование на платформе Java EE. Разработка компонентов на основе технологии Enterprise JavaBeans
Часть I

Санкт-Петербургский государственный политехнический университет
2016

Цель курса

- Основная задача: *формирование компетенций, необходимых для разработки бизнес компонентов с использованием технологии Enterprise JavaBeans*
- Ограничения курса

Необходимые знания

- знание синтаксиса языка Java в версии не ниже JDK 1.5
- знание процесса разработки программ на платформе Java
- знание Java API, используемого для разработки сетевых приложений и приложений с доступом к базам данных
- общее понимание архитектуры распределённых приложений
- представление о требованиях, предъявляемых к распределённым приложениям масштаба предприятия

Используемый инструментарий

- Java EE 7 Software Development Kit
  http://www.oracle.com/technetwork/java/javaee/downloads/

- NetBeans IDE и необходимые plugin-ы

  http://www.netbeans.org/downloads/

- Настройка среды

Общий тематический план курса

- Основные принципы и понятия
- Платформа Java EE
- Сеансовые компонент
- Компоненты, управляемые сообщениями
- Использование служб таймера
- Реализация классов и методов перехватчиков
- Реализация транзакций
- Реализация безопасности
- Использование технологии EJB - рекомендации

Основные принципы и понятия

- Распределённое приложение
- Корпоративное приложение
- Основные требования к корпоративным приложениям
- Принципы разработки корпоративных приложений

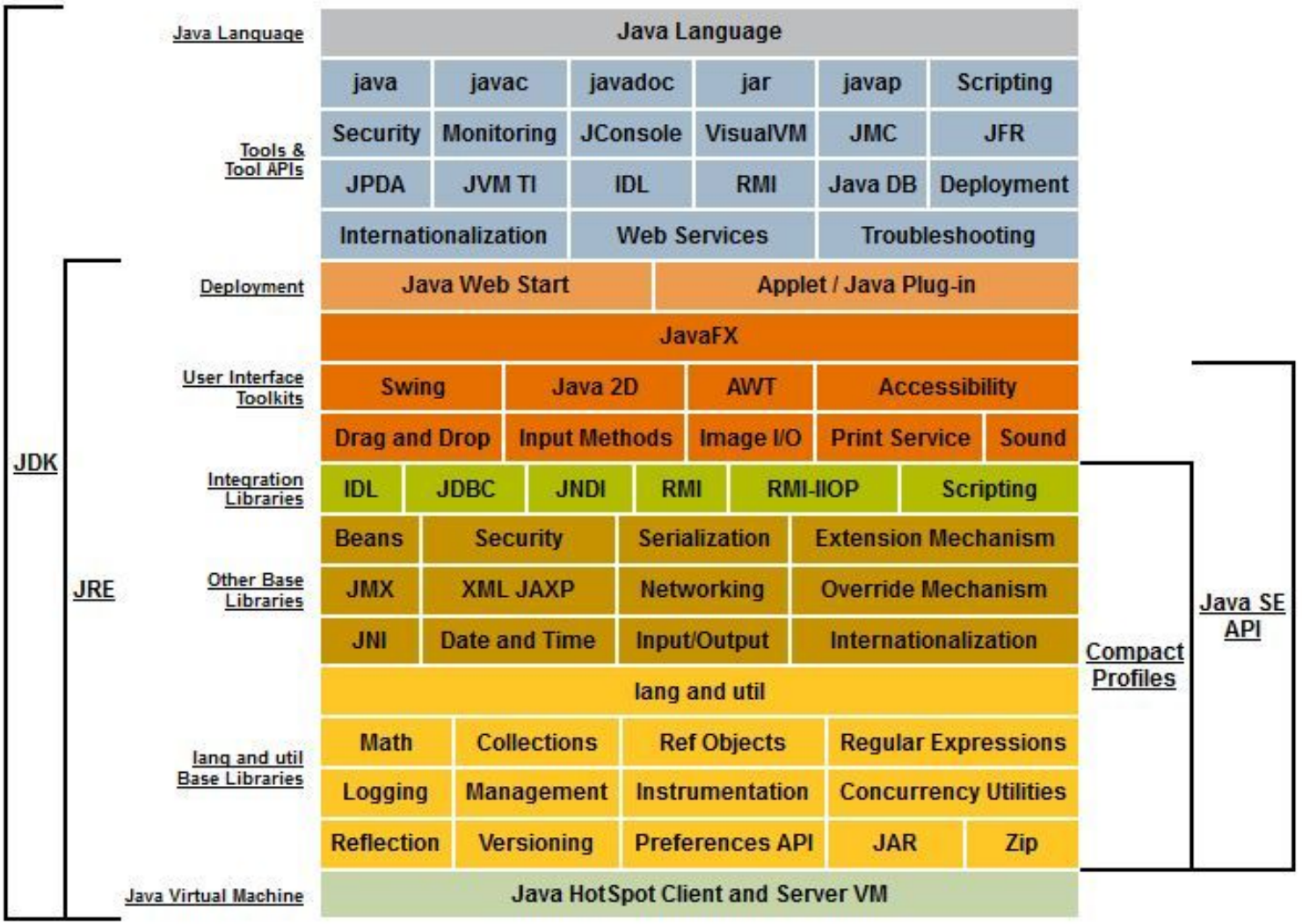
Тема 1. Введение в Java EE

- Платформа Java
- Платформа Java Enterprise Edition (Java EE)
- Архитектура Java EE приложения
- Службы контейнера Java EE
- Типы компонент EJB
- Легковесный контейнер EJB
- Преимущества разработки корпоративных приложений на основе Java EE

Платформа Java

- Платформа Java™ включает:
  - спецификацю языка Java
  - стандартные утилиты
  - стандартные пакеты (Java API)
  - виртуальную Java-машину (JVM)

Структура платформы Java SE



Редакции платформы Java

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)

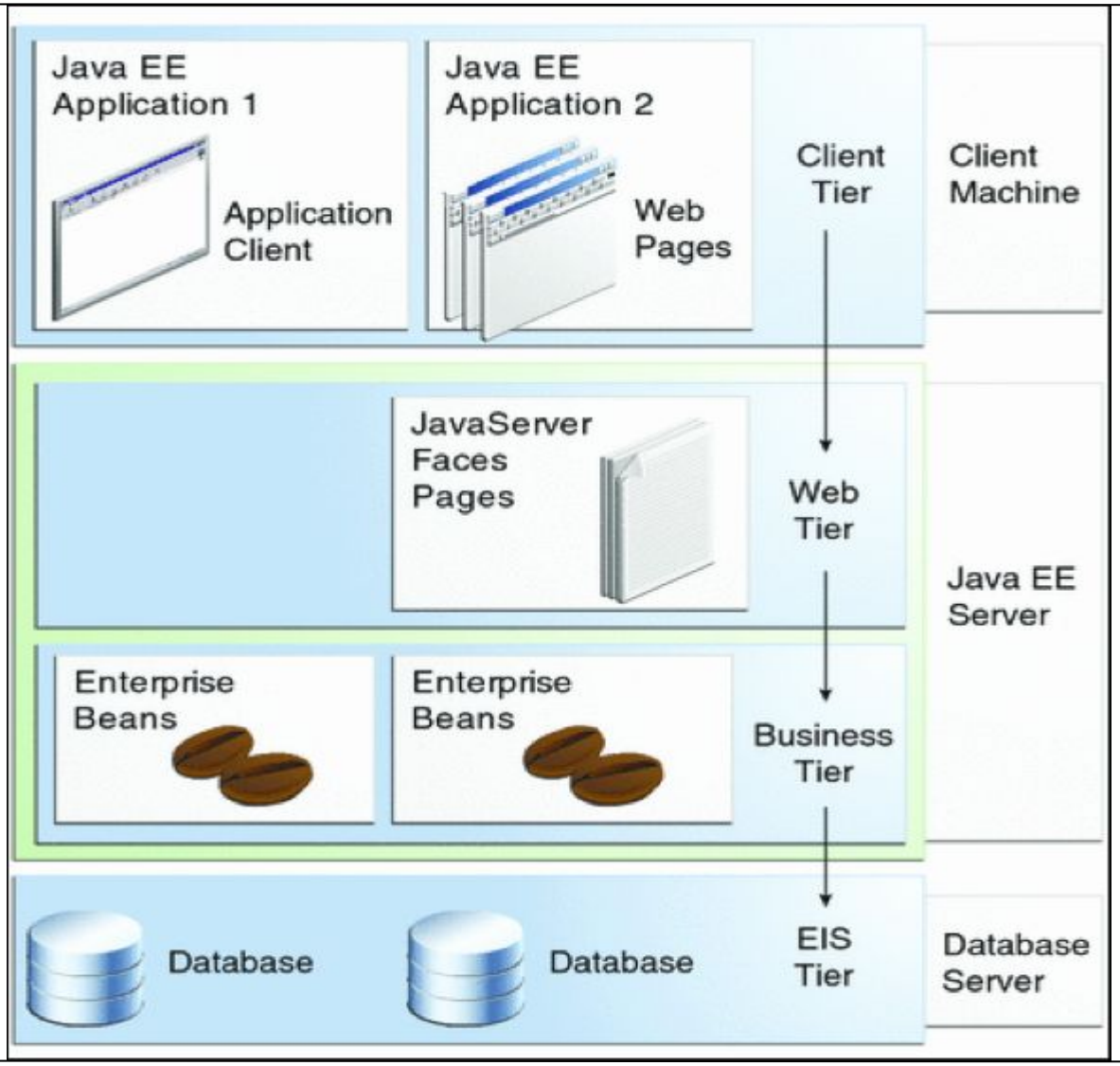
Платформа Java Enterprise Edition

- Является “надстройкой“ над платформой Java SE
- Облегчает разработку приложений, которые являются:
- распределёнными
- крупномасштабными
- многоуровневыми
- масштабируемыми
- надёжными
- безопасными

Архитектура Java EE приложения

- Типичное Java EE приложение содержит:
- клиентский уровень
- промежуточный уровень, который обычно реализует бизнес-логику приложения и обеспечивает интернет-доступ к службам приложения
- уровень данных (Enterprise Information System - EIS)

Архитектура Java EE приложения



Технология Enterprise JavaBeans (EJB)

- **Enterprise JavaBeans™ (EJB)** - стандартная архитектура для разработки на основе платформы Java™ объектно-ориентированных приложений масштаба предприятия (корпоративных приложений)
- EJB поддерживает полный жизненный цикл корпоративных приложений, написанных на языке Java
- EJB поддерживает полный жизненный цикл web-сервисов

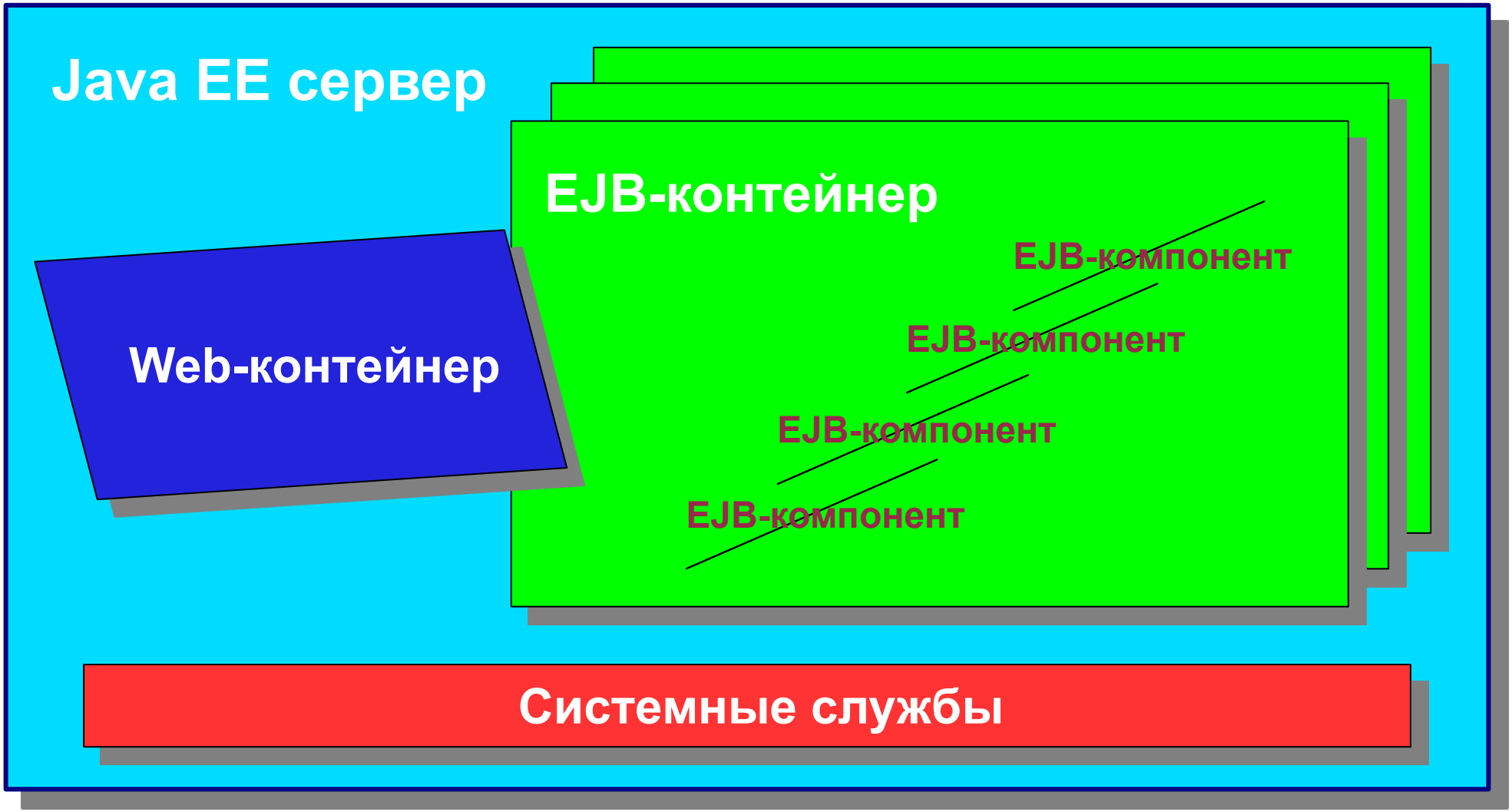
Основные термины

- **EJB-сервер** определяется как логическое устройство, которое обеспечивает инфраструктуру (среду выполнения), необходимых для функционирования EJB-компонентов
- **EJB-контейнер** является специализированным сервисом, который непосредственно обслуживает работу EJB-компонента
- **EJB-модуль** представляет собой набор EJB-компонент и дескриптор развёртывания

Основные термины

- EJB-компонент - многократно используемый программный компонент, обладающий возможностью взаимодействия с аналогичными компонентами
- EJB-компоненты обеспечивают функциональность на стороне сервера
- EJB-компонент может экспортировать свойства, методы и события
- EJB-компоненты можно относительно легко добавлять и удалять из системы

Структура Java EE сервера



Службы контейнера Java EE

- **Lifecycle Management:** обеспечивает управление жизненным циклом объекта
- **State Management**: управляет состоянием объекта
- **Security**: обеспечивает выполнение проверок безопасности
- **Transactions**: позволяет управлять транзакциями по требованию EJB-компонента
- **Persistence**: обеспечивает управление сохранением данных.

Типы EJB-компонентов

- **Сеансовые** (Session Beans) компоненты реализуют логику клиентского приложения, обычно не имеют устойчивого состояния
- **Управляемые сообщениями** (Message Driven Beans) — определяют асинхронную реакцию приложения на определённые события в системе
- **Объектные компоненты** (Entity Beans) (опционально) представляют собой данные и имеют устойчивое состояние

API для Java EE приложений

- Разработка EJB-компонентов основывается на специализированных пакетах:
  - javax.ejb
  - javax.annotation
- Также используются:
  - javax.naming (JNDI API)
  - javax.jms (Java Message Service API)
  - ...

Общая схема разработки Java EE приложений

- Определение интерфейсов
- Разработка собственно компонента, реализующего необходимые интерфейсы
- Разработка дополнительных классов
- Подготовка EJB JAR файла
- Развертывание приложения

Основные роли жизненного цикла EJB-приложения

- Роли обеспечения инфраструктуры:
- *EJB Server Provider;*
- *EJB Container Provider.*
- Роли разработки приложений:
- *Enterprise Bean Provider;*
- *Application Assembler.*
- Роли поставки и настройки:
- *Deployer;*
- *System administrator.*

Когда можно использовать EJB

При необходимости обеспечить:

- многопользовательский удалённый доступ
- масштабируемость
- безопасность и ограничение доступа к данным
- целостность и сохранность данных
- поддержку распределённых транзакций
- поддержку событий
- хорошую сопровождаемость

Преимущества разработки на основе EJB

- Enterprise JavaBeansTM (EJB) следует общему принципу Java: “написанное однажды – работает везде”. Разработанные EJB-компоненты могут разворачиваться на множестве платформ без изменения кода или перекомпиляции
- Технология EJB избавляет разработчика от необходимости использовать низкоуровневое API для выполнения рутинных операций

Преимущества разработки на основе EJB

Для **корпоративных приложений**:

- повышается производительность
- повышается надёжность функционирования
- повышается возможность масштабирования
- снижаются затраты на разработку
- снижаются затраты на сопровождение

2. Реализация сеансовых компонентов

- Сеансовые компоненты
- Три типа сеансовых компонент
- Выбор правильной сеансовой компоненты при заданном бизнес-ограничении
- Создание сеансовых компонент
- Пакетирование и развертывание сеансовых компонент

Сеансовый компонент

Типичный сеансовый компонент:

- в каждый конкретный момент времени представляет на стороне сервера одного и только одного клиента
- не сохраняет своего состояния
- может представлять web-сервис
- может участвовать в транзакции
- не представляет непосредственно разделяемые данные из БД, но может получать к ним доступ и обновлять их
- может иметь произвольный по продолжительности жизненный цикл

Клиенты сеансового компонента

- Локальный клиент должен работать в том же адресном пространстве (на той же JVM), может являться другим компонентом или web-компонентом. Для локального клиента расположение компонента не может быть прозрачным.
- Удаленный клиент – может работать на удаленной JVM и являться клиентским приложением, другим EJB-компонентом, web-компонентом. Для удаленного клиента действительное положение компонента должно быть прозрачным.

Разработка сеансового компонента

- Сеансовый компонент может реализовывать:
  - локальный интерфейс
  - удалённый интерфейс
- Сеансовый компонент может использовать:
  - вспомогательные интерфейсы
  - вспомогательные классы
  - внешние ресурсы

Разновидности сеансовых компонентов

- Сеансовые (Session Beans) компоненты подразделяются на:
  - **stateless** (без сохранения состояния)
  - **stateful** (с поддержкой сессии)
  - **singleton** («одиночка» начиная с версии 3.1)

Выбор вида сеансового компонента

- Необходимо учитывать различия сеансовых компонентов, т.к. они:
  - имеют разный жизненный цикл
  - имеют разные возможные состояния
  - по разному используются клиентами
  - могут реализовывать разную логику обработки запросов

Основные критерии выбора вида сеансового компонента

Критерий выбора	Stateless	Stateful	Singleton
Состояние компонента должно сохраняться между вызовами одного клиента (поддержка сессии)?	Нет	Да	Нет
Вызовы клиента могут использовать индивидуальные данные клиента?	Нет	Да	Нет
Запросы клиентов могут обрабатываться параллельно?	Да	Да	Нет

Определение сеансового компонента

- Определение интерфейса для последующей имплементации EJB-компонентами

```
@Remote
public interface name {
    // method declaration
}
```

```
@Local
public interface name {
    // method declaration
}
```

Определение сеансового компонента

- Определение класса сессионного компонента без наследования интерфейса
- Методы такого класса могут вызываться локальными клиентами
- Аннотация определяет конкретную разновидность компонента

```
@Annotation
public class className {
    // method realization
}
```

Определение сеансового компонента

- Класс сессионного компонента может наследовать любой интерфейс и аннотировать его как EJB-интерфейсы
- Для этого обычно используются аннотации:
- @Remote
- @Local

```
@Annotation (value={interfaceName.class})
public class className implements interfaceName{
    // method realization
}
```

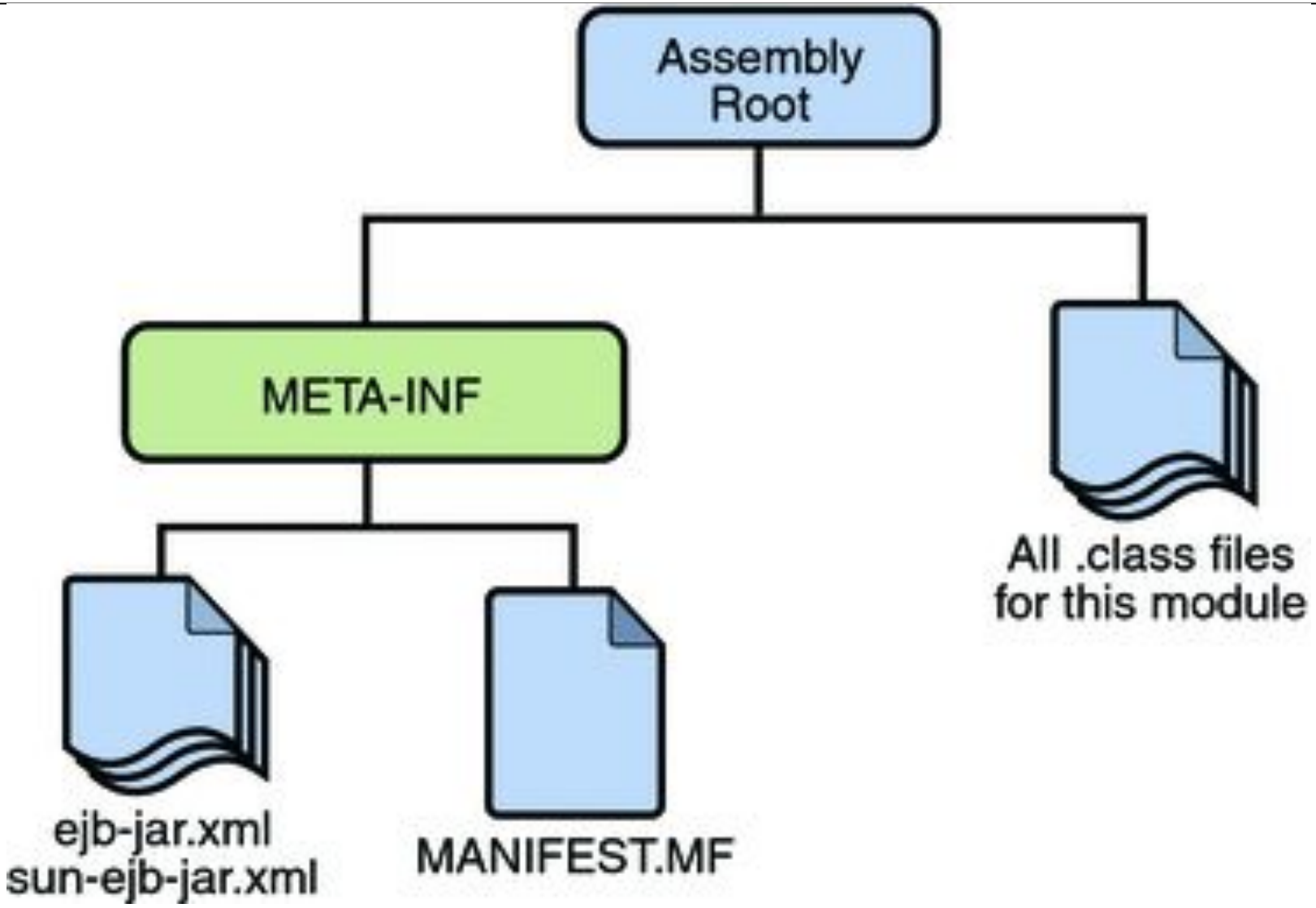
Соглашения по архивации компонентов:

- Java EE приложения хранятся в архивах с расширением .ear
- EJB-модули хранятся в архивах с разрешением .jar
- Web-приложения хранятся в архивах с разрешением .war
- Дополнительные файлы и файлы ресурсов хранятся в архивах с расширением .jar
- Разделяемые пакеты хранятся в архивах с разрешением .jar и в подкаталоге /lib
- Клиенты приложения с Main-Class хранятся в архивах с разрешением .jar

Пакетирование сеансовых компонентов

- Откомпилированные файлы компонентов заносятся в специально структурированный архив с расширением jar
- Данный архив компонентов:
- непосредственно в корне архива имеет специальную каталог META-INF для размещения конфигурационных файлов таких как MANIFEST.MF
- может содержать дескриптор поставки в каталоге META-INF
- в отдельном каталоге содержит откомпилированные компоненты
- может содержать дополнительные каталоги со вспомогательными классами и файлами

Пакетирование сеансовых компонентов



Дескриптор поставки

- Дескриптор поставки представляет собой файл в формате XML, содержащий определение атрибутов EJB-компонентов
- Дескриптор поставки размещается в подкаталоге META-INF.
- Дескриптор поставки может хранить информацию:
  - о структуре компонентов (имя, тип и класс компонента, его интерфейсы, параметры среды, ссылки на внешние ресурсы, тип транзакции)
  - о правилах и особенностях развертывания компонентов.

Пример дескриптора поставки

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>MessageBean</ejb-name>
      <mdb-connection-factory>
        <jndi-name>
jms/JupiterConnectionFactory
        </jndi-name>
      </mdb-connection-factory>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

Развёртывание сеансовых компонентов

- Архив с компонентами должен развёртываться на Java EE-сервере
- Развёртывание может осуществляться:
  - штатными средствами IDE
  - встроенными средствами Java EE сервера
  - на основе процедурного описания, например с помощью технологии ant

EJB: использование аннотаций

```
@Resource(name="myDB", type=javax.sql.DataSource)
@Stateful public class ShoppingCartBean
                       implements ShoppingCart {
    @Resource SessionContext ctx;

    public Collection startToShop (String productName) {
        ...
        DataSource productDB =
           (DataSource)ctx.lookup("myDB");
        Connection conn = myDB.getConnection():
        ...
    }
    ...
}
```

3. Доступ к сеансовым компонентам

- Роль JNDI в установлении связи с EJB компонентами
- Конфигурирование свойств JNDI окружения
- Использование JNDI для поиска ресурса
- Создание кода для получения ссылки на ресурс путём внедрения (injection)
- Создание клиента сеансовой компоненты
- Создание фасада сеанса (session fasade)
- Использование внедрения зависимости (dependency injection) для указания на EJB

JNDI пространства имён

- Роль JNDI
- Пакет javax.naming
- Глобальное пространство имён:

java:global[/app-name]/module-name/bean-name[!fqn]

- Пространство имён приложения:

java:app/module-name/bean-name[!fqn]

- Модульное пространство имён:

java:module/bean-name[!fqn]

Пример определения JNDI

```
package demo;

@Singleton(name="Demo")
@LocalBean
@Remote(demo.DemoRemote.class)
public class DemoBean { ... };
```

- Для модуля demo.jar определены следующие имена:
- java:global/demo/Demo!demo.DemoBean
- java:global/demo/Demo!demo.DemoRemote
- java:app/demo/Demo!demo.DemoBean
- java:module/demo!demo.DemoRemote

Настройка JNDI

- Настройка может осуществляться с помощью конструктора класса

```
InitialContext(Hashtable<?,?> environment)
```

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,

    "com.sun.jndi.fscontext.RefFSContextFactory");

env.put(Context.PROVIDER_URL, file:/tmp/tutorial");

Context ctx = new InitialContext(env);
```

Поиск ресурса

- Интерфейс Context и класс InitialContext
- Поиск осуществляется с помощью методов класса InitialContext:
- lookup (...)

```
Object obj = ctx.lookup(name);
```

- Регистрация объектов с помощью метода
- bind(...)

Ссылки на ресурсы и зависимости

- Интерфейс SessionContext
- Определение ресурсов

аннотация @Resource

```
public class DemoServlet extends HttpServlet {

@Resource(name="java:comp/DefaultDataSource")
   private javax.sql.DataSource dsc;
   ...
}
```

Ссылки на ресурсы и зависимости

- Определение зависимостей

```
@javax.enterprise.context.RequestScoped
public class Demo { ... }

public class DemoServlet extends HttpServlet {
   @Inject Demo d;
...
}
```

Разработка клиента компонента

- Разработка локального клиента
- Разработка удалённого клиента
- Создание фасада сессии

4. Дополнительные сведения о сеансовых компонентах

- Связь между EJB контейнером и EJB компонентой
- Жизненный цикл сеансовых компонент, обладающих и не обладающих состоянием
- Реализация методов жизненного цикла сеансовой компоненты
- Использование сеансовой компоненты для осуществления асинхронной коммуникации
- Тонкая настройка управления пакетированием и развертыванием

Особенности методов жизненного цикла

- Методы жизненного цикла должны возвращать тип void и не должны иметь параметров.
- Метод @PostConstruct вызывается контейнером после создания и настройки компонента, но до первого вызова бизнес-методов.
- Метод @PreDestroy вызывается контейнером после завершения метода, аннонсированного как @Remove, перед удалением компонента из контейнера.
- Метод @PostActivate вызывается контейнером после того, как контейнер перевел компонент из пассивного состояния в активное
- Метод @PrePassivate вызывается контейнером перед переводом компонента из активного состояния в пассивное.

Жизненный цикл сеансового компонента

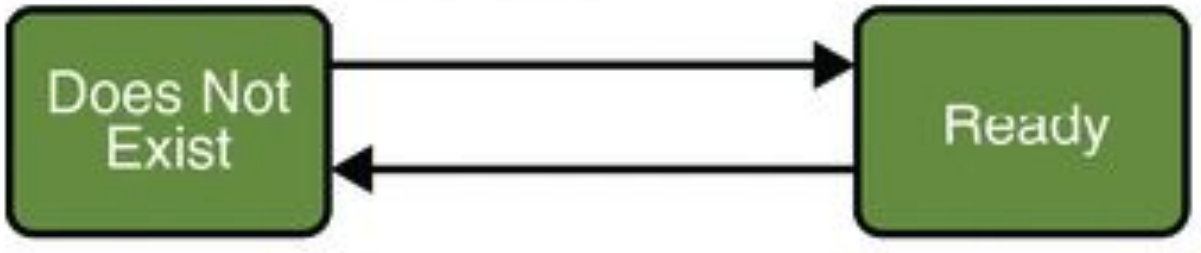
- Создание компонента
- Установление зависимостей
- Методы жизненного цикла:

-@PostConstruct, init() и ejbCreate<Method>

-@PrePassivate() и @PostActivate()

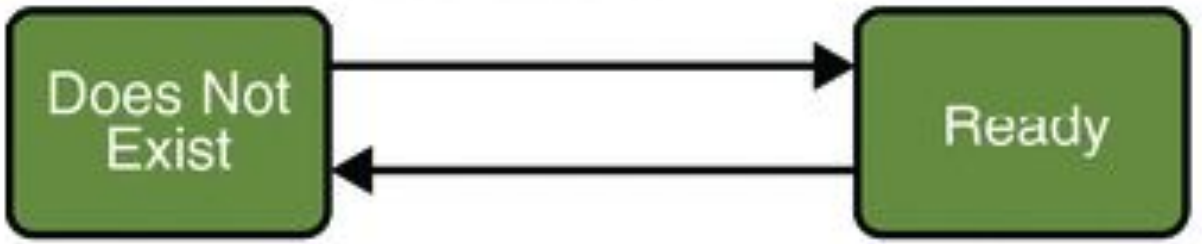
-@Remove() и @PreDestroy(

Жизненный цикл сеансового компонента



- Создание
- Установление зависимостей
- @PostConstruct()
- @PreDestroy()

EJB: жизненный цикл Stateless компонента



Создание
Установление зависимостей
@PostConstruct()
@PreDestroy()

Асинхронные вызовы

- Особенности асинхронных вызовов
- Область применения
- Реализация асинхронных методов
- Обработка исключений

Асинхронные вызовы

```
@Asynchronous
public Future<Integer> foo (...) {
// do something
   Integer result = ...;
   return new AsyncResult<Integer>(result);
}
```

5. Сеансовая компонента синглтон

- Преимущества и недостатки использования синглетонных сеансовых компонент
- Создание синглетонной сеансовой компоненты
- Жизненный цикл синглетонной сеансовой компоненты
- Реализация методов жизненного цикла синглетонной сеансовой компоненты
- Параллельный доступ к синглетону
- Управление параллелизмом

Заключение

- Обобщающий обзор рассмотренных вопросов
- Вопросы, требующие дополнительного рассмотрения
- Вопросы к лектору