北京郵電大學

分布计算环境

北京邮电大学计算机学院

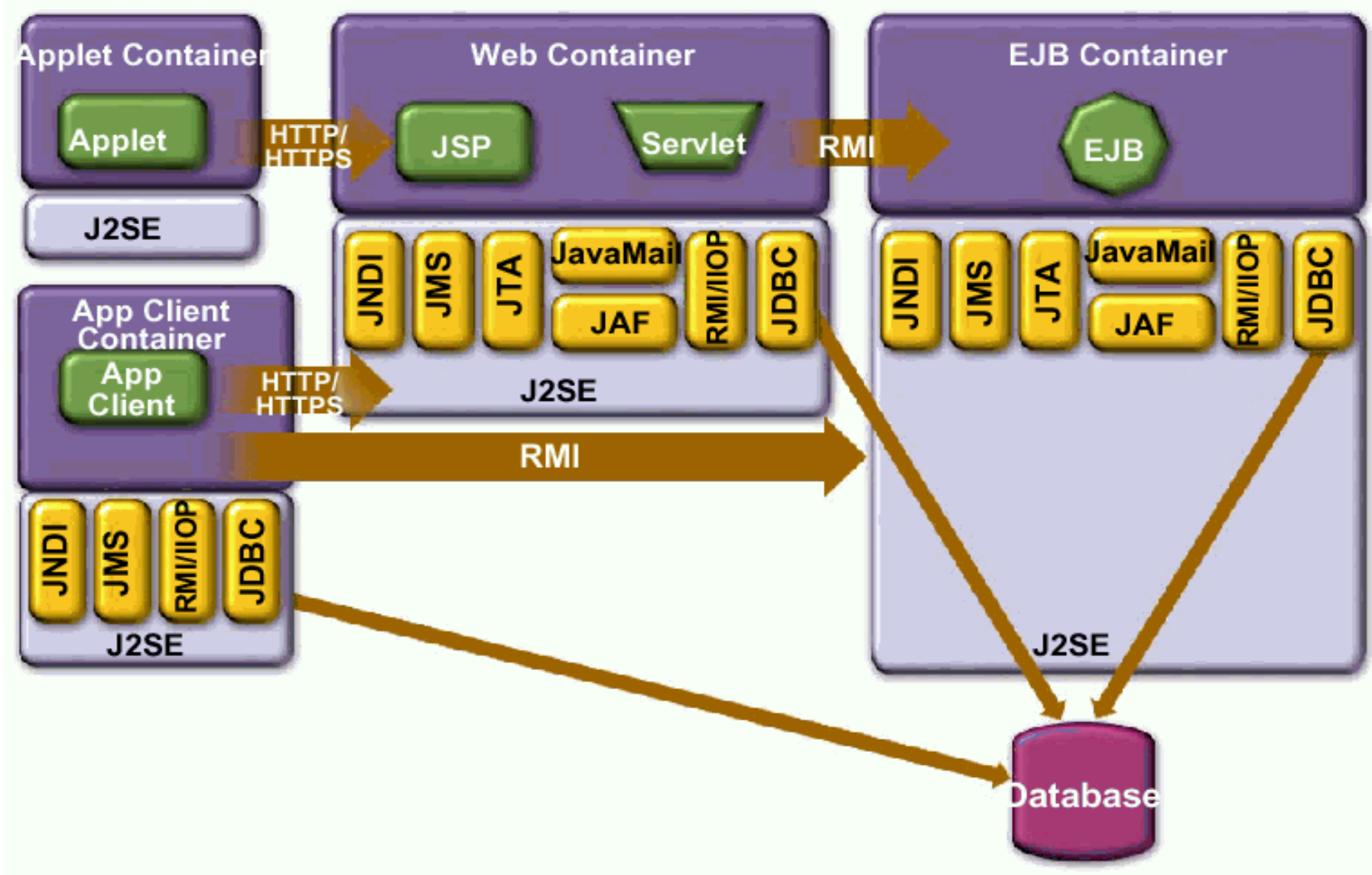
主要内容

- 基于构件的软件体系结构
- J2EE/Java EE
- EJB
- 轻量级框架和EJB3.0

主要内容

- 概述
- EJB 2.0的开发
- EJB分类
- EJB容器
- EJB小结

EJB 在Java EE中的位置



EJB 定义

- **Enterprise JavaBeans(EJB) 是：**
  - Java服务器端服务框架的规范，软件厂商根据它来实现EJB服务器。应用程序开发者可以专注于支持应用所需的商业逻辑，而不用担心周围框架的实现问题
- **SUN公司对EJB的定义：**
  - EJB是用于开发和部署多层结构的、分布式的、面向对象的Java 应用系统的跨平台的构件体系结构
  - 采用EJB可以使开发商业应用系统变得容易，应用系统可以在一个支持EJB的环境中开发，开发完之后部署在其他的环境中，随着需求的改变，应用系统可以不加修改地迁移到其他功能更强、更复杂的服务器上
- EJB1.0(1998)，EJB2.0 (2001)，EJB3.0(2006)，EJB3.1(2009)，EJB3.2(2013)

EJB 的主要特点(1)

- EJB是用Java语言开发分布式的、面向对象的企业应用系统的标准构件体系结构，EJB使得通过组合构件得到分布式应用成为可能
- EJB定义了一个协议，使得不同供应商提供的构件能在运行时互操作
- EJB遵循Java的“write once, run anywhere”的原则，一个EJB可以部署在任何EJB平台上
- EJB通过定义一系列标准的服务API来封装现有的基础性服务，EJB应用通过这些标准的API来使用服务
- EJB不需要应用开发人员了解底层的事务处理细节,状态管理,多线程,资源共享管理,以及其它底层API细节

EJB 的主要特点(2)

- EJB可以在不修改源代码的基础上进行定制：
  - EJB应用可以通过部署描述文件（Deploy Descriptor）进行定制化
  - 厂商可以进行自己的扩充，以支持更强的可定制特性：很多厂商除了J2EE标准的DD（ejb-jar.xml）外，还会有一个厂商特定的辅助部署描述符，其中描述了厂商自己扩充的运行时可定制特性
- EJB体系结构和已有的服务器平台，其它的Java APIs, CORBA兼容

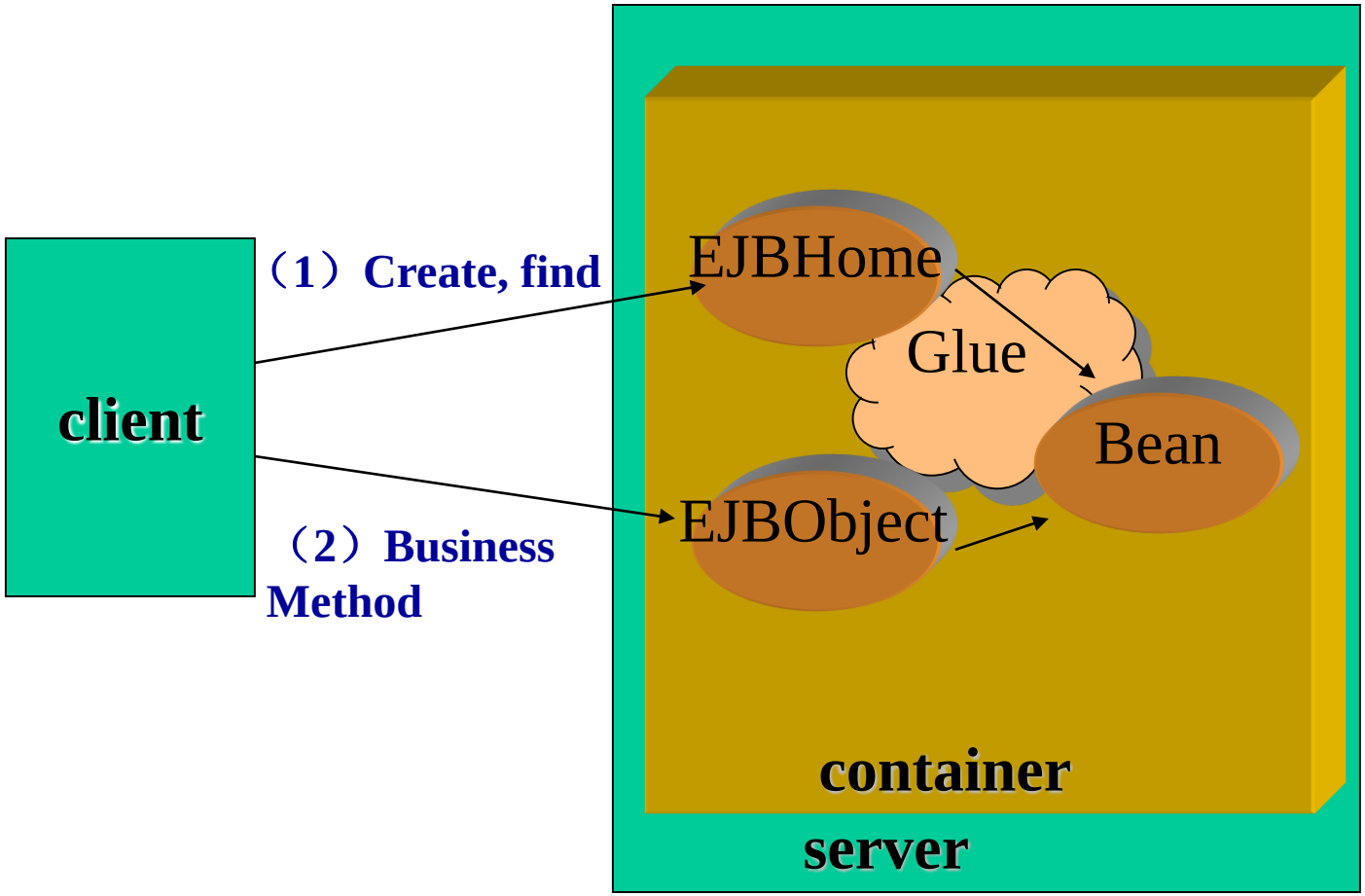
主要内容

- 概述
- EJB 2.0的开发
- EJB分类
- EJB容器
- EJB小结

EJB2.0的三个关键构件

- **EJBHome 接口**（扩展javax.ejb.EJBHome接口）：
  - 使用了factory设计模式，定义了创建、查找EJB的方法。
  - EJB Home接口由EJB容器创建，用户可通过JNDI来得到每个构件相应的Home接口，通过此Home接口来创建、删除EJB对象、构件以及相应的信息
- **EJBObject接口**（扩展javax.ejb.EJBObject接口）：
  - 使用了proxy设计模式，定义了在bean中实现的业务逻辑方法
  - EJB对象类似于EJB构件的“代理”，由EJB Home接口创建，客户端程序通过EJB对象来调用服务器端构件的方法
- **Bean实现类**（实现javax.ejb.EntityBean/SessionBean）：
  - 实现业务逻辑

三个构件的关系

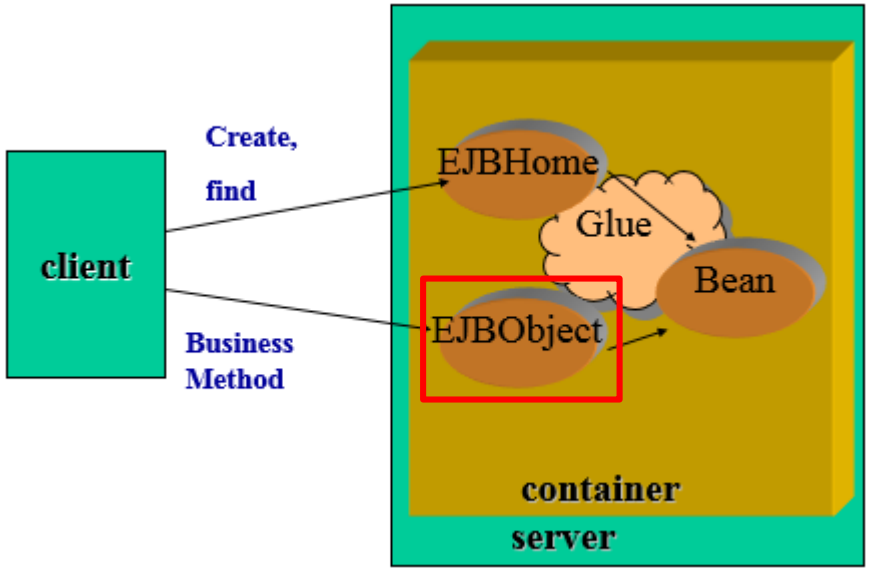

client
（1）Create, find
（2）Business Method
EJBHome
Glue
Bean
EJBObject
container
server

Server侧：EJB构件的实现步骤

- 定义远程接口（EJBObject接口）
  - 业务方法
- 定义Home接口
  - 允许用户创建、查找、删除远端对象
- 实现Bean的实现类，实现远程接口
  - 编写业务方法实现代码
- 编译远程接口、Home接口、bean实现类
  - 远程接口、Home接口中的方法实现代码是自动生成的
- 创建部署描述符（Deploy Descriptor：说明构件的配置要求，如事务、持久化和安全等方面）
- 将以上三个文件与部署描述符文件打包为一个ejb-jar文件
- 部署EJB构件：将Jar文件发布到EJB应用服务器环境中，测试各层的连接情况

EJB的实现(举例）—定义远程接口

- 远程接口包含enterprise bean实现的远程方法的定义
- 客户程序只能通过远程接口访问EJB实现的远程方法，不能直接调用
- Remote接口由Enterprise Bean Provider编写
- EJB容器生成Remote接口的实现



HelloWorld远程接口

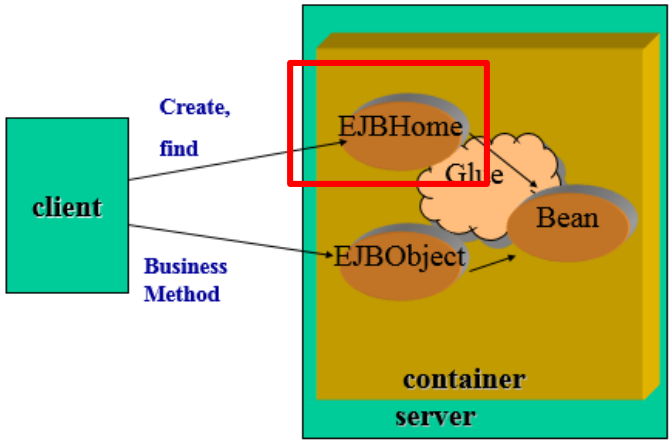
```
package testejb;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface HelloWorld extends javax.ejb.EJBObject
{
        public   String sayHello()  throws RemoteException;
}
```

远程接口必须继承EJBObject接口类

远程方法必须抛出RemoteException异常

EJB的实现—定义Home接口

- Home Interface包含enterprise bean生命周期管理的相关方法
- 客户程序使用Home Interface创建、查找或删除EJB的实例
- Home接口由Enterprise Bean Provider编写
- EJB容器生成home接口的实现
- Home接口中主要定义Create方法：
  - 在EJB中，create方法取代传统OO中的constructor来初始化一个对象
  - EJB容器可以维护一定数量的对象实例，并且通过create方法来初始化对象的内部状态
    - ➔这样要比实例化一个对象实例快
  - 可有多个Create方法，但每个方法的参数定义不同



HelloWorldHome接口

- package testejb;
- import javax.ejb.*;
- import java.util.*;
- import java.rmi.*;

- public interface HelloWorldHome ext[illegible]s
  javax.ejb.EJBHome {
- public HelloWorld create() throws CreateException, RemoteException;
- }

Home接口必须继承EJBHome接口

Create方法返回的是前面定义的远程接口的引用

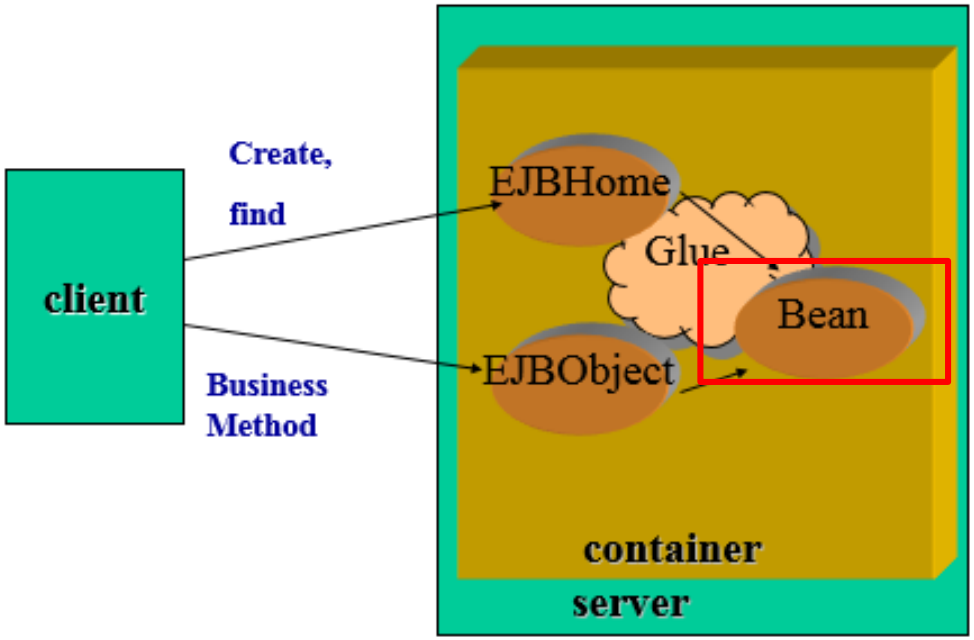
Create方法必须抛出CreateException和RemoteException异常

Home接口中可定义的其它类方法

- 删除（remove）方法
  - 删除方法允许客户端应用删除EJB的实例
  - 一个enterprise bean的home接口可以包含多个remove方法，但是每个remove方法的定义必须不同
- getEJBMetaData方法
  - 该方法用于获得EJB的EJBMetaData接口
  - EJBMetaData接口用来获取EJB的相关信息

EJB的实现—定义Bean的实现类

- 定义好remote接口和home接口后，就可以定义相关的EJB的实现类
- 在EJB中，remote接口中所定义的远程方法由EJB的实现类实现
- 该类由Enterprise Bean Provider定义实现



HelloWorldBean类

```
package testejb;
import javax.ejb.*;
public class HelloWorldBean implements SessionBean, HelloWorld {
    SessionContext sessionContext;
     public void ejbCreate() throws CreateException {}
    public void ejbRemove() {}
     public void ejbActivate() {}
     public void ejbPassivate() {}
     public void setSessionContext(SessionContext sessionContext) {
          this.sessionContext = sessionContext;
    }
     public String sayHello() {
          return "Hello World";
    }
}
```

根据EJB的种类决定要实现什么接口

EJB开发人员访问容器上下文的入口

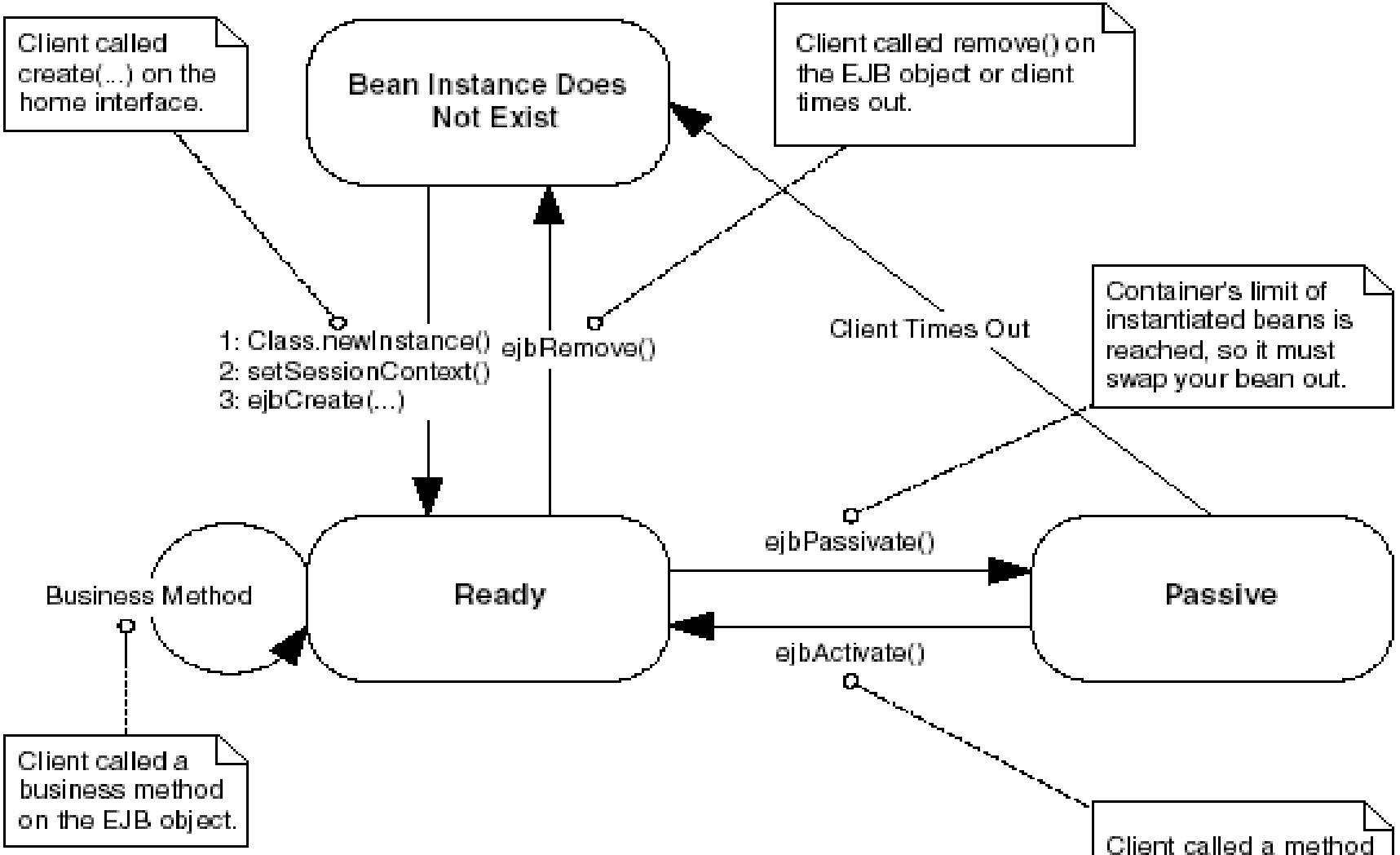
用来初始化sessionContext变量；每次创建一个session bean时，容器会调用

本例EJB的远程方法

EJBContexts：通向容器之门

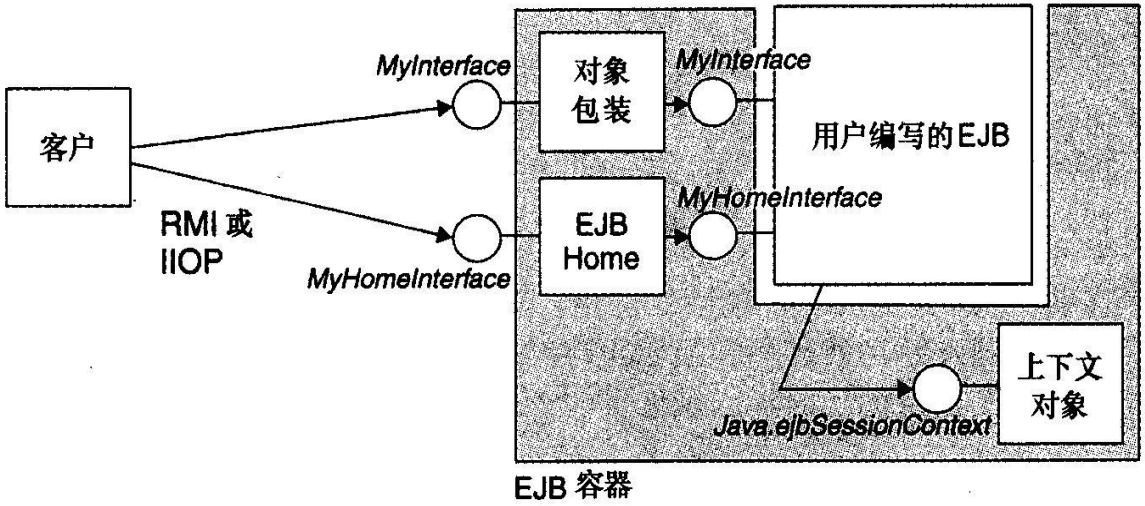
- EJB运行于容器之内，容器将EJB当前的状态信息以及运行时环境信息封装在EJBContext中，通过调用**setSessionContext**方法传给EJB
  - **setSessionContext(SessionContext sessionContext)**
- EJBContext接口定义了多种方法供EJB访问这些信息，如
  - 调用EJB的用户的安全角色
    - ➔sessionContext .getCallerPricipal();
  - 如事务相关信息；
    - ➔sessionContext .getUserTransaction();

用于生命周期管理



称为生命周期方法（函数）或回调方法（函数）

再看构件与容器



- 容器在客户与构件之间提供一个屏蔽，可截获每个操作调用，进行安全验证、生命周期管理等
- 客户引用不直接指向构件，而是指向对象包装(Wrapper)，容器可使用它进行构件的激活\去激活管理，提高性能

部署描述文件

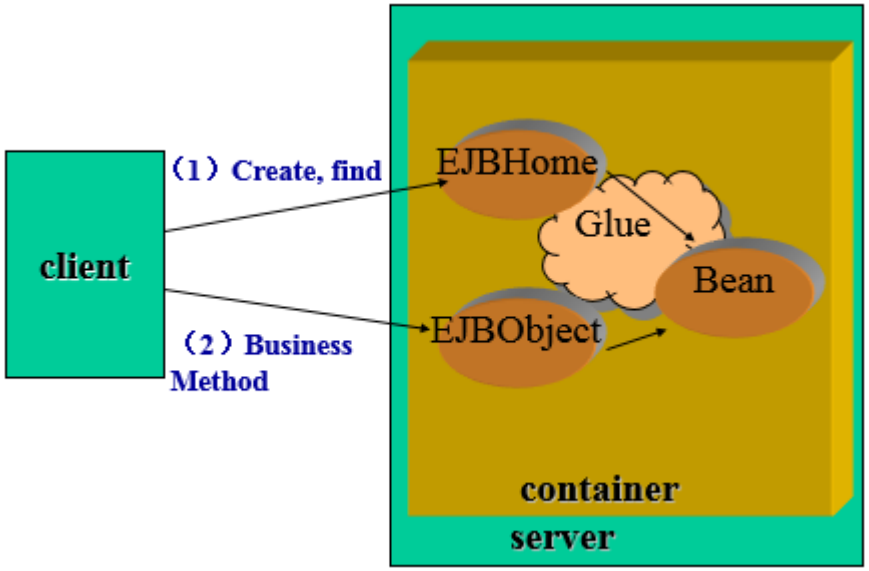
- 部署描述文件描述了ejb-jar的相关信息，例：
  - EJB的名字
  - EJB的实现类名
  - EJB的远程HOME接口
  - EJB的远程接口
  - EJB的类型
  - 会话Bean的状态管理类型
  - 实体Bean的持久化管理类型
  - 实体Bean的主键类

HelloWorld部署描绘符

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
    "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>HelloWorld</display-name>
      <ejb-name>HelloWorld</ejb-name>
      <home>testejb.HelloWorldHome</home>
      <remote>testejb.HelloWorld</remote>
      <ejb-class>testejb.HelloWorldBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <description />
        <ejb-name>HelloWorld</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

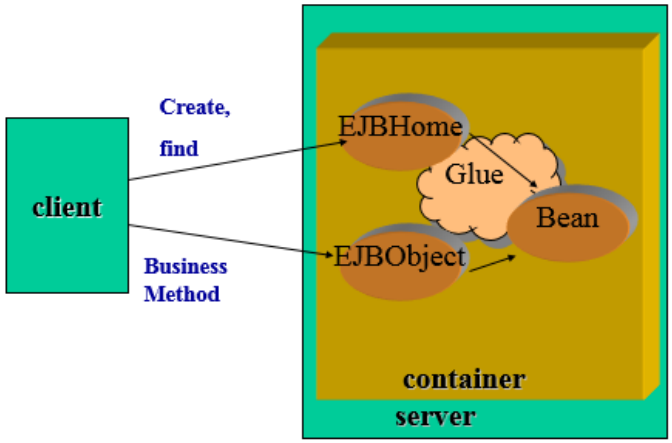
Client侧：使用EJB

- 查找Home对象；
  - 使用JNDI定位本地Home对象
- 使用Home对象创建EJB对象
  - 相应的create方法
- 调用EJB对象的远程方法
- 删除EJB对象



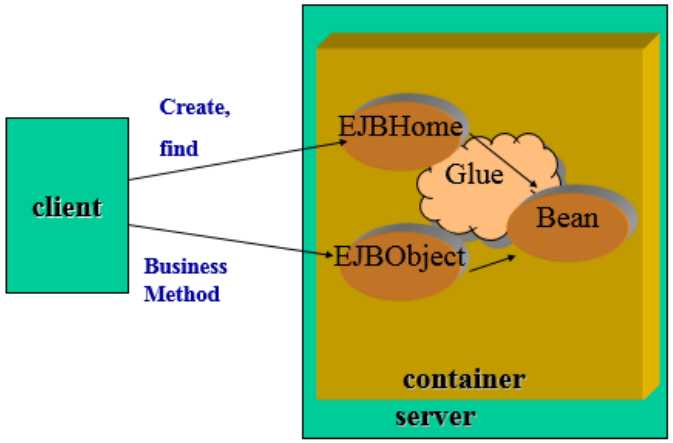
HelloWorldTestClient1

```
……
public class HelloWorldTestClient1  {
    ……
    private HelloWorldHome helloWorldHome = null;
    private HelloWorld helloWorld = null;

    //Construct the EJB test client
    public HelloWorldTestClient1() {
        ……..
        //get naming context
         Context context = getInitialContext();
         //look up JNDI name
         Object ref = context.lookup("HelloWorld");
         //look up jndi name and cast to Home interface
         helloWorldHome = (HelloWorldHome)
                  PortableRemoteObject.narrow(ref, HelloWorldHome.class);
         ……
}
```



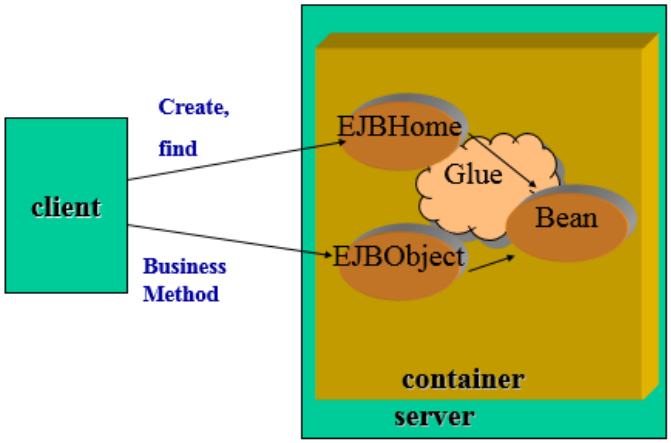
HelloWorldTestClient1(续）

```
//--------------------------------------------------------------------
    -
  // Methods that use Home interface methods to generate a
     Remote interface reference
  //--------------------------------------------------------------------
     ---
public HelloWorld create() {
     …….
     try {
          helloWorld = helloWorldHome.create();
     }catch(Exception e) {
          e.printStackTrace();
     }
     return helloWorld;
}
```



HelloWorldTestClient1(续）

```
public static void main(String[] args) {
   HelloWorldTestClient1 client = new HelloWorldTestClient1();
   HelloWorld aBean = client.create();
   try{
            System.out.println(aBean.sayHello());
    }
    catch(java.io.IOException ex){
         System.out.println("Can't read from `" + ex.getMessage() + "'");
    }
  }
};
```



EJB编程限制

- EJB的开发者并不需要在EJB组件代码中编写系统级的服务，EJB应该仅仅聚焦于商务逻辑，它依赖于EJB容器所提供的服务而不是自己来直接解决底层的系统层问题
- EJB容器完全负责EJB组件的生命期、并发处理、资源访问、安全等，所以与容器本身的锁定和并发管理等方面相冲突的可能性就需要消除；如：
  - 使用任何方法启动、停止和管理线程
  - 传递this引用指针作为一个参数或者作为返回值返回this引用指针
- EJB组件开发者应该明白EJB编程限制的重要性，并从组件的稳定性和可移植性方面来遵循它们。实际上，这些限制已经在组件的合同中成为了一个条款

主要内容

- 概述
- EJB 2.0的开发
- EJB分类
- EJB容器
- EJB小结

EJB的分类

- 会话Bean
- 实体Bean
- 消息驱动Bean（EJB2.0新增的类型）

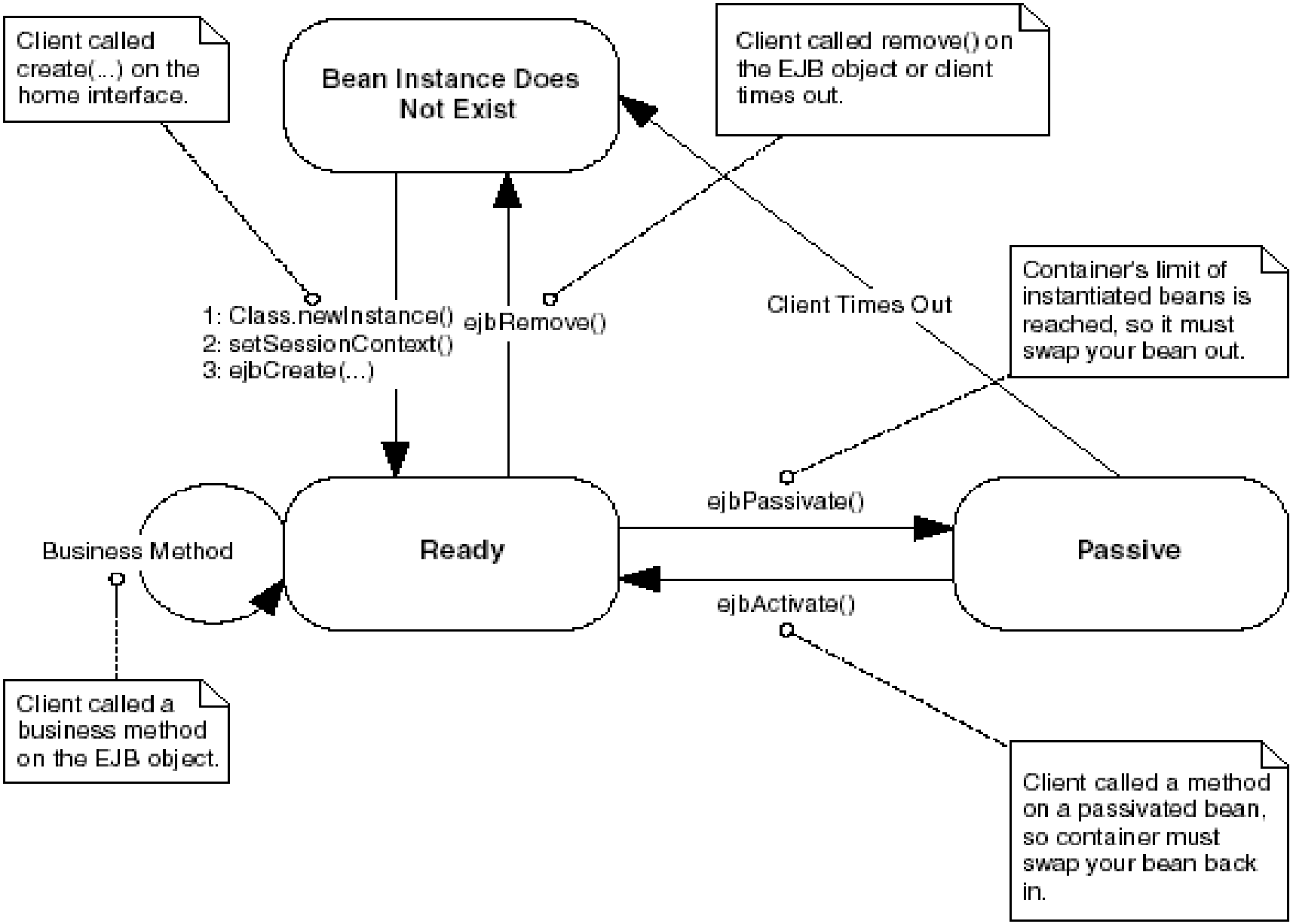
会话bean(Session Bean)

- 会话：指从客户端获得 EJB对象开始，然后调用EJB的方法（可以多次），直到客户端生命周期结束，或客户端释放了EJB对象为止，称为一次会话。随着会话的终止，EJB对象也有可能被EJB容器销毁
- 用于编写商业逻辑，比如网上购物
- 是一种非持久性的Bean，相对生命较短，一般与客户同步，存在于客户应用与应用服务器交互的时间段内；
- 其中的数据不保存在数据库中；在EJB服务器崩溃时被删除
- 不表示数据库中的数据，但可以访问数据
- 有两类：
  - 无状态会话Bean
  - 有状态会话Bean

有状态（Stateful）bean

- 同时只处理一个客户应用的请求，一对一的维持相应调用客户的状态，并且在不同的方法调用中维持这个状态
- 每个会话中的请求始终委托至Bean类的同一个实例(客户端1-- 实例1)
- 适用于某业务进程中会话时间长，且有多个请求的情况（此时必须记录两个请求间用户的状态）
- 由于对于每一个并发用户，必须有一个对应的Stateful Session Bean，为了提高系统的效率，Stateful Session Bean可以在一定的客户空闲时间后被容器写入二级存储设备(如硬盘)，在客户发出新的调用请求后，再从二级存储 设备恢复到内存中

Stateful bean的生命周期

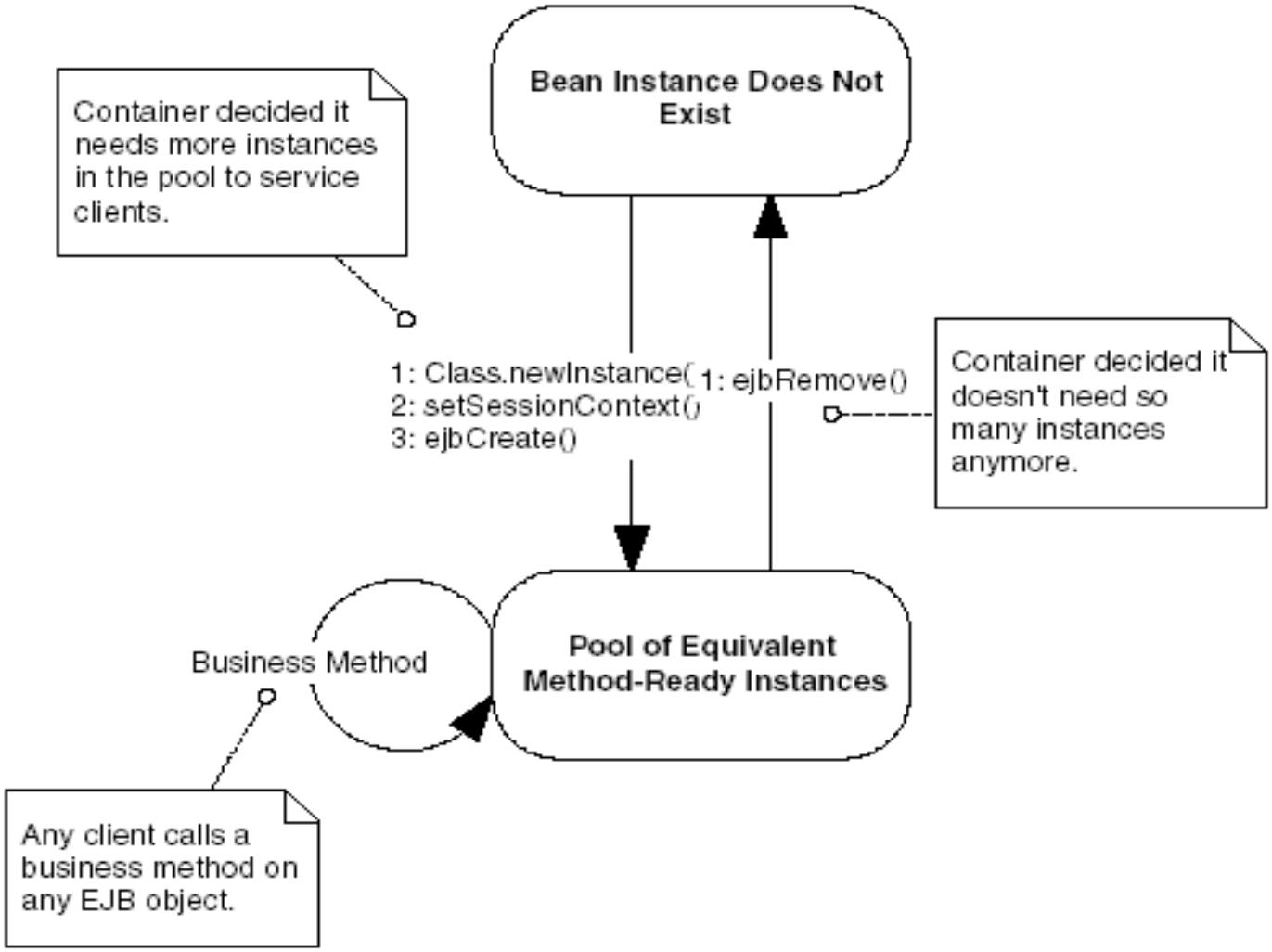


无状态（Stateless）bean

- 表达一个无状态的服务，在方法调用中间不维护任何状态，不存储用户相关信息；同时可以处理多个客户应用的请求
- 所有实例都是完全相同的，可随意分发给某一客户端：可用来构造响应频繁而简单的访问的bean池
- 适合于一个应用有非常多的客户端的情况

- 注意：由于这种Session Bean没有任何状态，所以对于客户端来讲，所有的Bean实例都是相同的，Container在创建这个Bean实例的时候也就不能带任何参数

Stateless bean的生命周期



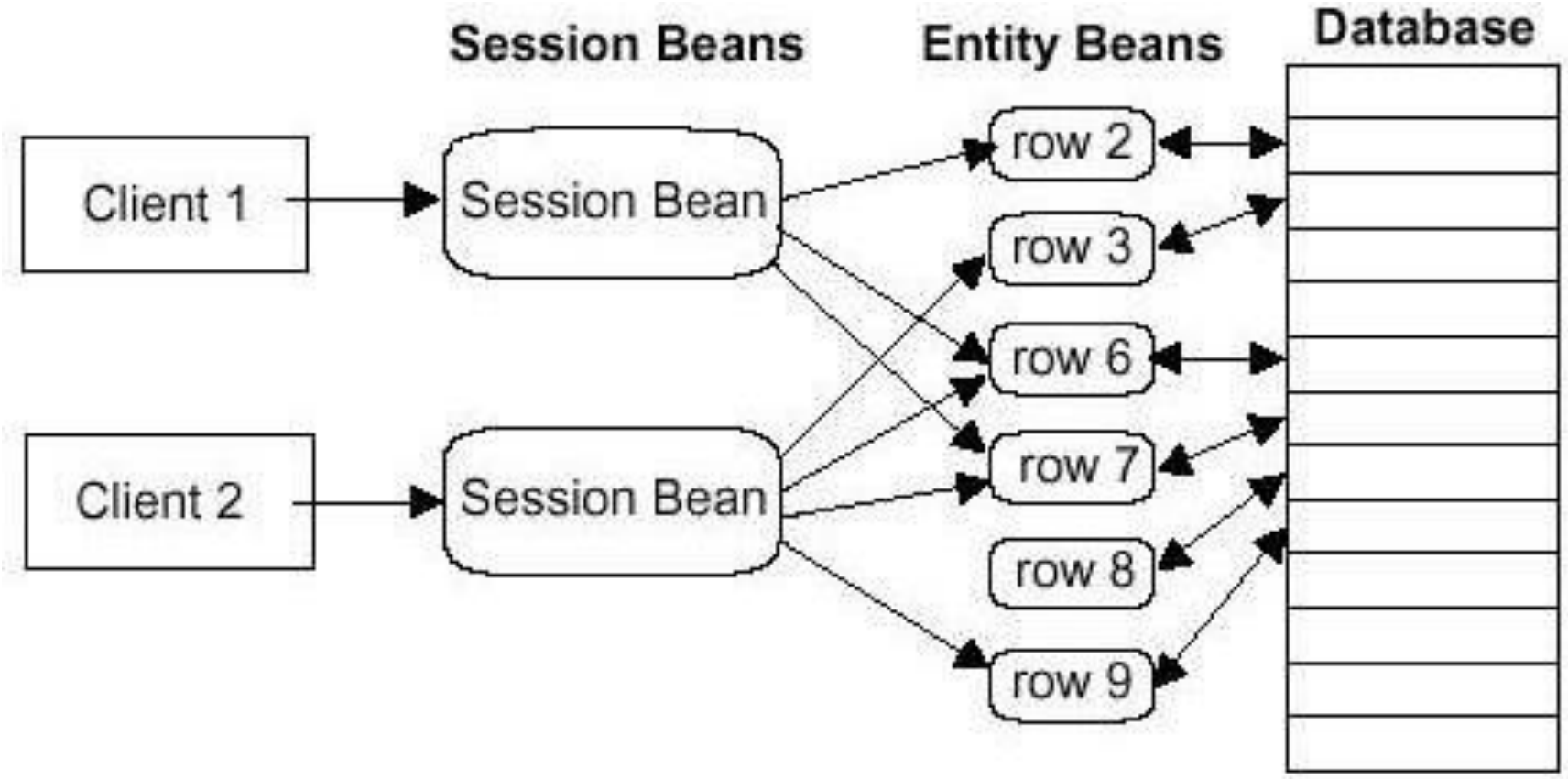
EJB的分类

- 会话Bean
- 实体Bean
- 消息驱动Bean（EJB2.0新增的类型）

实体bean（Entity Bean)

- 用对象表示数据库中的实体，封装相应的操作，客户端可以使用这些对象进行对数据库的操作
  - 如一个**Entity Bean**代表一张数据库表，其实例代表的就是一个数据库记录
  - 数据库表中的每个**Field**在**Entity Bean**中定义相应的属性进行对应
- 可以支持多个并发用户，而容器则使访问和事务同步化
- 具有持久性：
  - 与数据库中的数据有一样长的生命
  - 知道如何把自己映射成永久性数据
  - 可以从数据库中存储的数据重新实例化
  - **EJB** 服务器崩溃后仍可重构

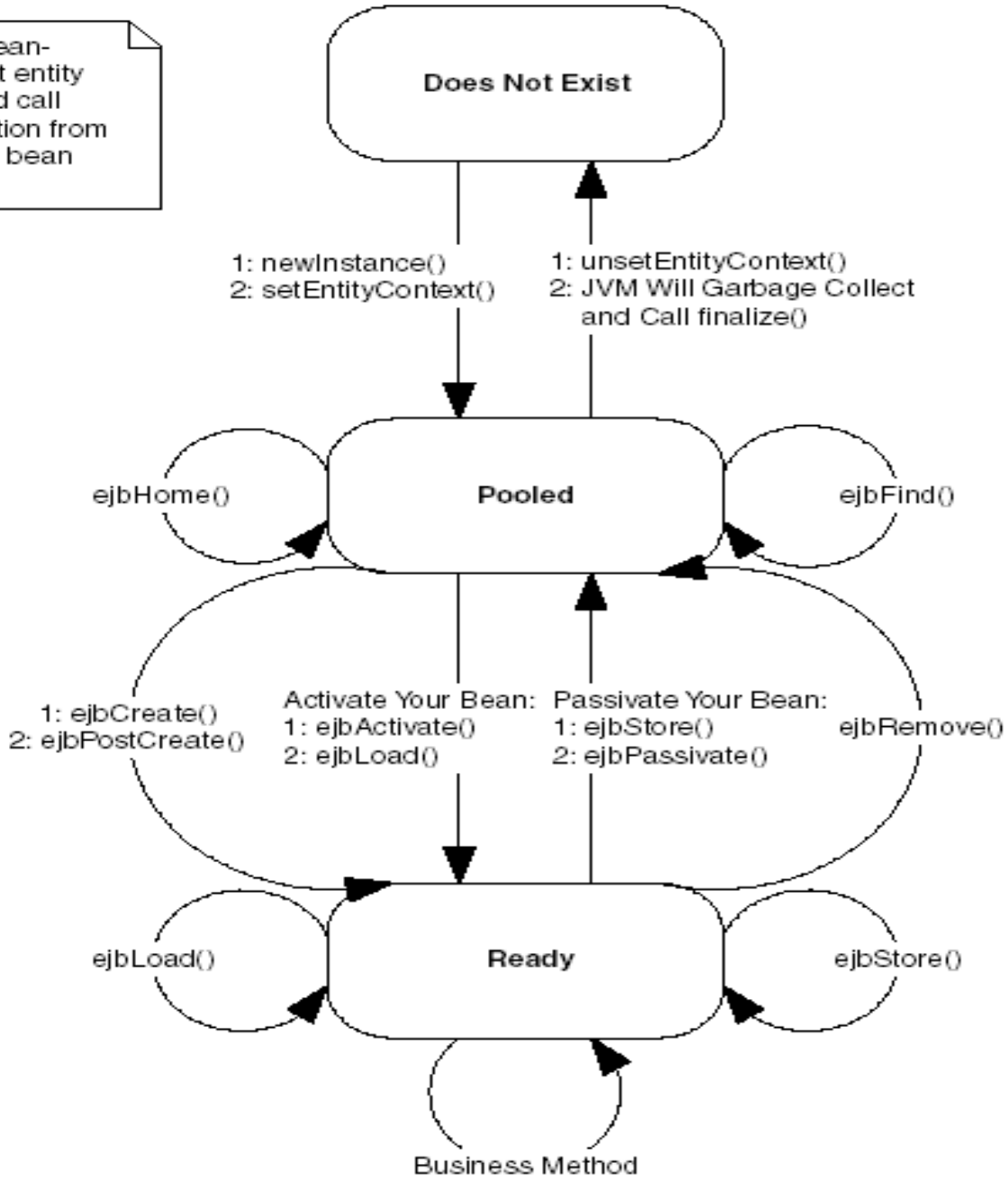
会话Bean vs. 实体Bean



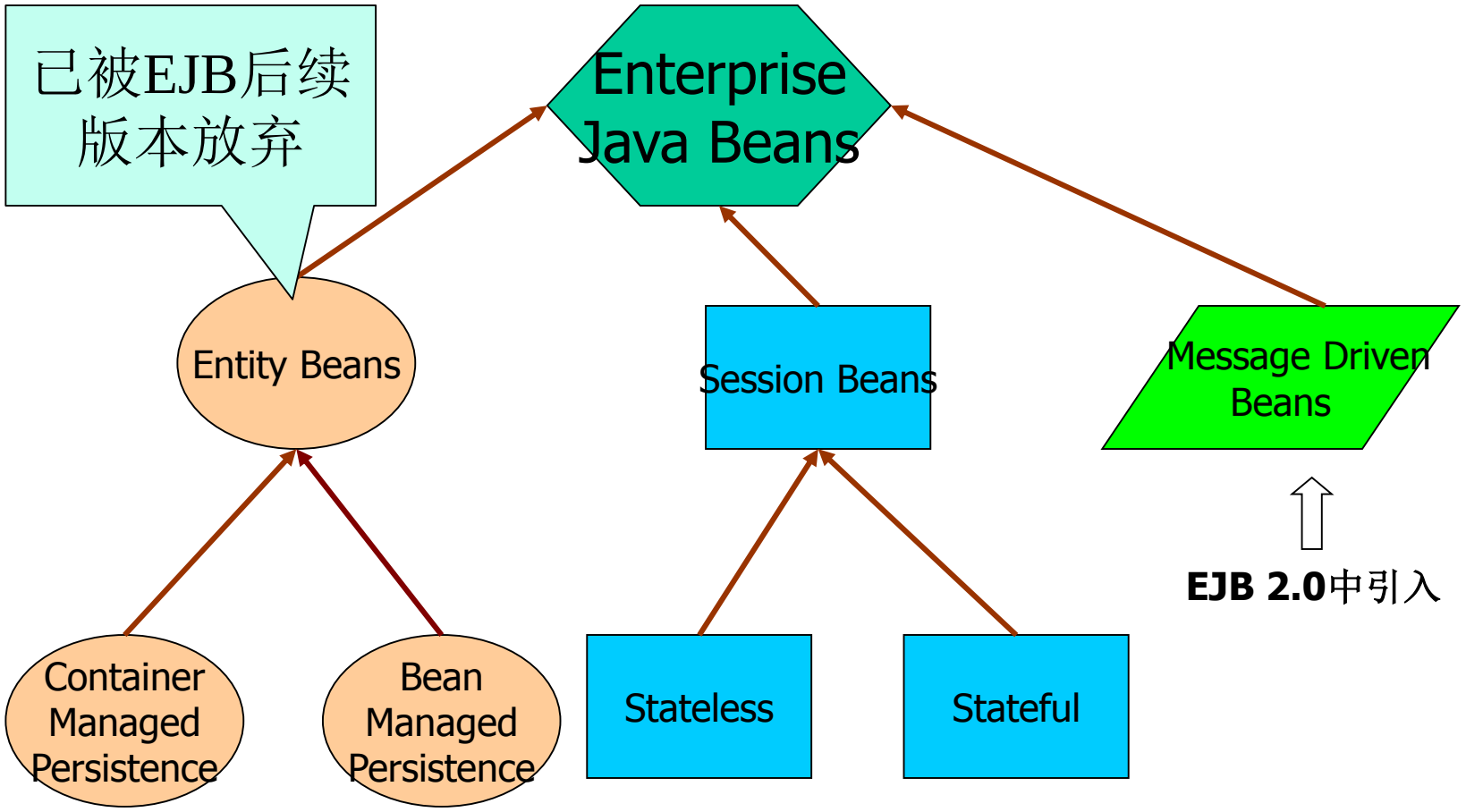
一种实体Bean的 ejbCreat()例

```
Public String ejbCreate(String productID,String name,double  basePrice) {
    this.productID = productID;
    this.name = name;
    this.basePrice = basePrice;
    try{
        Connection conn = getConnection();
        PreparedStatement  pstmt = conn.prepareStatement(
          "insert into products (id, name, price) values(?,?,?)");
        pstmt.setString(1,productID);
        pstmt.setString(2,name);
        pstmt.setString(3,basePrice);
        pstmt.executeUpdate();
        return productID;
    }catch(Exception e){}
    finally{……//释放数据库连接资源；}
}
```

The lifecycle of a bean-managed persistent entity bean. Each method call shown is an invocation from the container to the bean instance.



EJB分类小结

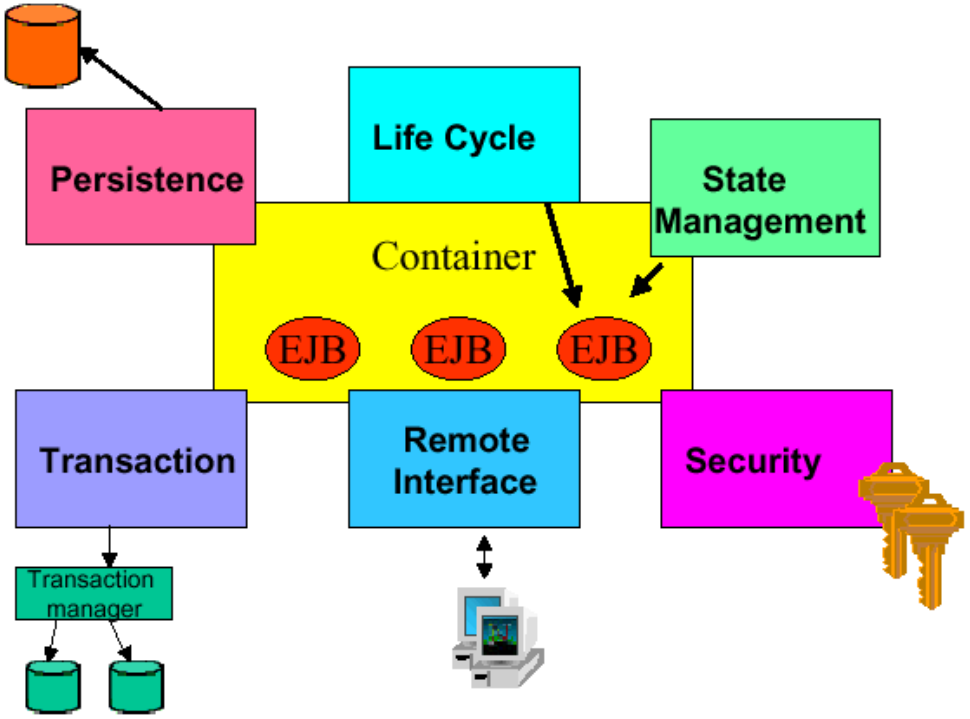

已被EJB后续版本放弃
Enterprise Java Beans
Entity Beans
Session Beans
Message Driven Beans
EJB 2.0中引入
Container Managed Persistence
Bean Managed Persistence
Stateless
Stateful

主要内容

- 概述
- EJB 2.0的开发
- EJB分类
- EJB容器
- EJB小结

EJB容器

- 一个管理一个或多个EJB类/实例的组件。它通过规范中定义的接口来提供EJB类所需的服务。容器厂商也可以在容器或服务器中提供额外服务的接口
- 理论上讲，一个EJB容器可以包含任何数量的enterprise bean



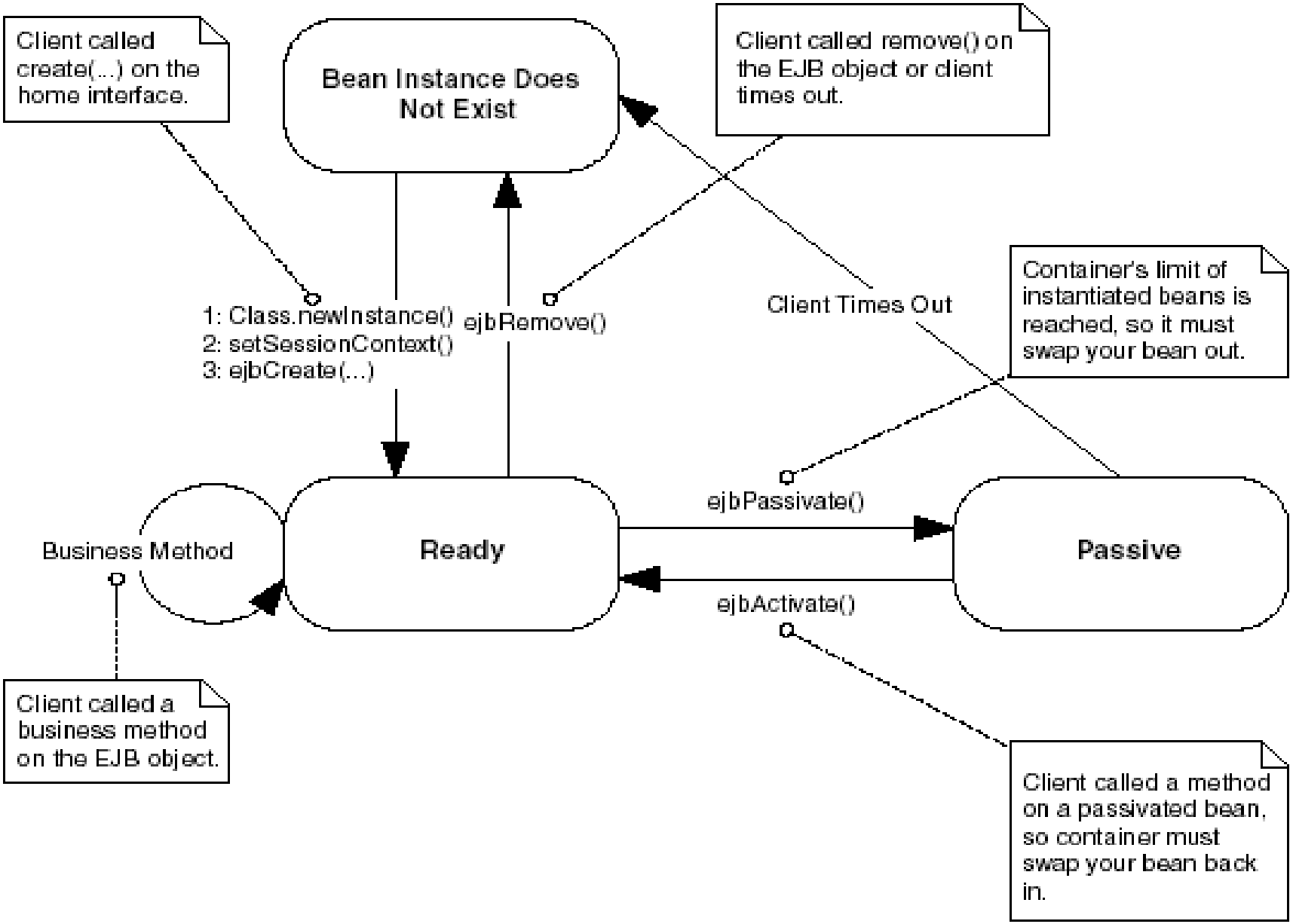
Remote Client Connectivity

- 是EJB容器提供的最基本的服务，该服务支持远端的客户应用访问enterprise bean
  - EJB容器使用JNDI服务将enterprise bean的home interface注册到一个目录服务中
  - 客户应用通过JNDI服务接口获取特定enterprise bean的home接口的引用

Life Cycle Management

- 生命周期管理服务支持管理enterprise bean的生命周期，基于生命周期管理服务，enterprise bean可以为来自多个客户应用的请求服务
- 在处理来自多个客户应用的请求时，EJB容器会改变enterprise bean的状态。在此上下文下，Enterprise bean的状态表明EJB容器中是否存在enterprise bean的实例。例如
  - 当客户应用请求某个enterprise bean的远程方法时，如果EJB容器中不存在该enterprise bean的实例，EBJ容器就会为该enterprise bean创建一个实例
  - 当一个enterprise bean的实例不再被客户程序使用时，EJB容器会删除其实例
- 不同种类的Bean 有不同的生命周期模型（即状态模型）

Stateful bean的生命周期



Security Management

- 安全性管理服务保证只有授权用户才能够访问EJB应用中的enterprise bean
- 具有安全要求的EJB的客户端需要进行：
  - 认证：校验客户端所声明的身份
  - 授权：一旦客户端获得认证，它必须拥有权限才能执行想要的操作
- 认证在EJB方法调用之前的某个时间执行，授权发生在EJB的方法调用期间
- EJB规范不涉及认证，跟应用服务器的具体实现以及客户端代码相关

Persistence Management

- 持久性管理服务管理entity bean数据的存储与获取
- EJB容器提供持久性管理所采用的机制是与具体的厂商相关的

Resource Management

- 资源管理服务管理大量的enterprise bean，这些bean在处理客户应用的请求时需要用到资源
- 资源管理服务将这些资源在大量的enterprise bean之间共享
- 例如：数据库连接资源：
  - 建立数据库连接是很耗时的，也是有限的。所以资源管理服务维护一组活跃的数据库连接，Enterprise bean可以快速的获取可用的数据库连接，一个enterprise bean释放的数据库连接可以被其它bean重用

Transaction Management

- 事务提供一个“all-or-nothing”的简单模型，如所有对象成功更新，所有工作成功提交，或者一个对象失败而整个工作回滚
  - 事务：一组或者全部发生或者一步也不执行的处理步骤
  - 提交：所有的事务步骤当作一个操作被完整地执行，称为该事务被提交
  - 回滚：事务的一部分或多步执行失败，导致没有步骤被提交，则事务必须回到最初的状态

事务的启动、提交与中止

- Begin:
  - 启动一个新事务
- Commit:
  - 结束一个事务
  - 存储事务过程所做的修改
  - 使得修改可以被其它事务访问
- Abort:
  - 结束一个事务
  - 取消事务过程所做的所有修改

EJB对事务的支持

- bean的客户可以以两种方式驱动事务：
  - 使用容器管理事务
  - 使用Bean管理的事务

容器管理事务

- EJB容器设置事务的边界，自动管理客户程序与enterprise bean交互的开始与结束
- 一般，容器在EJB方法开始之前开始一个事务，而方法就在事务提交之后立即退出
- 在容器管理的事务中，可以同时使用会话Bean和实体Bean

容器管理事务的事务属性

- 事务属性控制事务的范围
- 事务属性值举例：
  - Required：如果客户端运行在一个事务中同时调用了另一个EJB方法，那么该方法就在客户端的事务范围内执行，否则容器在运行该方法之前就会开始一个事务。
  - RequiresNew：如果客户端运行在一个事务中同时调用了另一个EJB方法，容器会首先挂起客户端事务，启动一个新事务执行方法，方法完成后恢复客户端的事务；
  - Mandatory：客户端必须已经运行在一个事务中，才能调用此EJB方法；
  - ……

事务属性和范围

事务属性	客户端事务	业务方法的事务
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	Error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

设置事务属性

- 不要求所有方法与事务相联系，当部署一个EJB时，在“部署描述符”中声明事务属性
- 可以为整个EJB或者某个方法指定事务属性
- 如果为一个方法指定一个属性，同时为Bean指定另一个属性，则方法属性优先

HelloWorld部署描绘符

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
    "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>HelloWorld</display-name>
      <ejb-name>HelloWorld</ejb-name>
      <home>testejb.HelloWorldHome</home>
      <remote>testejb.HelloWorld</remote>
      <ejb-class>testejb.HelloWorldBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <description />
        <ejb-name>HelloWorld</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

回滚一个容器管理的事务

- 两种方法：
  - 如果一个系统异常被抛出，容器将自动回滚异常；
  - 通过调用EJBContext接口的setRollbackOnly方法，EJB方法通知容器回滚这个事务；
- 如果在方法执行过程中，发现一个应用异常可以在该方法中捕获这个异常
  - 重新抛出系统异常，从而导致容器自动回滚事务
  - 或调用EJBContext接口的setRollbackOnly方法

setRollbackOnly（）方法举例

```
….
If (checkingBalance<0.00){
        context.setRollbackOnly();
        throw new InsufficientBalanceException();
}
updateSaving(savingBalance);
…..
```

容器回滚事务的基本方法

- 当容器回滚事务时，总是撤销事务内SQL调用带来的数据改变
- 只有在实体Bean内，容器才可以撤销对实例变量所作的修改
  - 容器自动调用ejbLoad()方法
- 当回滚发生时，会话Bean必须明确地重置事务范围内任何有变化的实例变量
- 重置会话Bean实例变量的最简单方法就是实现SessionSynchronization接口：
  - AfterBegin:：在事务开始之后，业务方法开始之前调用；
    - ➔可以在这里保护现场
  - afterCompletion(boolean):事务完成之后调用；其参数表示事务是否成功
    - ➔撤销对实例变量所作的修改可在这里完成

Bean管理的事务

- 容器管理事务有一个限制：当方法正在执行时，要么关联一个独立的事务，要么什么都不关联。使用Bean管理事务能获得对事务更精确的控制。如在Bean的某个方法实现代码中可以(使用伪代码）：

```
op1(){
        ……..
        Begin transaction
            If condition1 then commit transaction
            Else if condition2 then do something, commit transaction
        End transaction
        ……..
 }
```

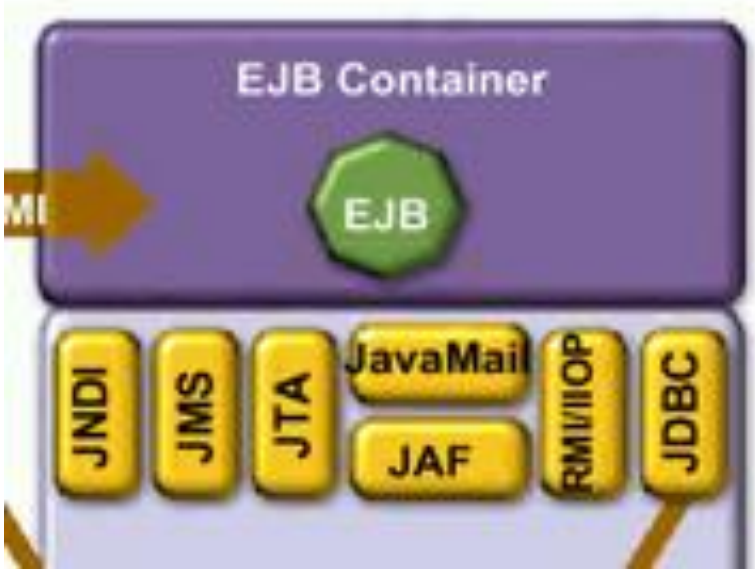
- 只适用于会话Bean

Bean管理的事务需要的API

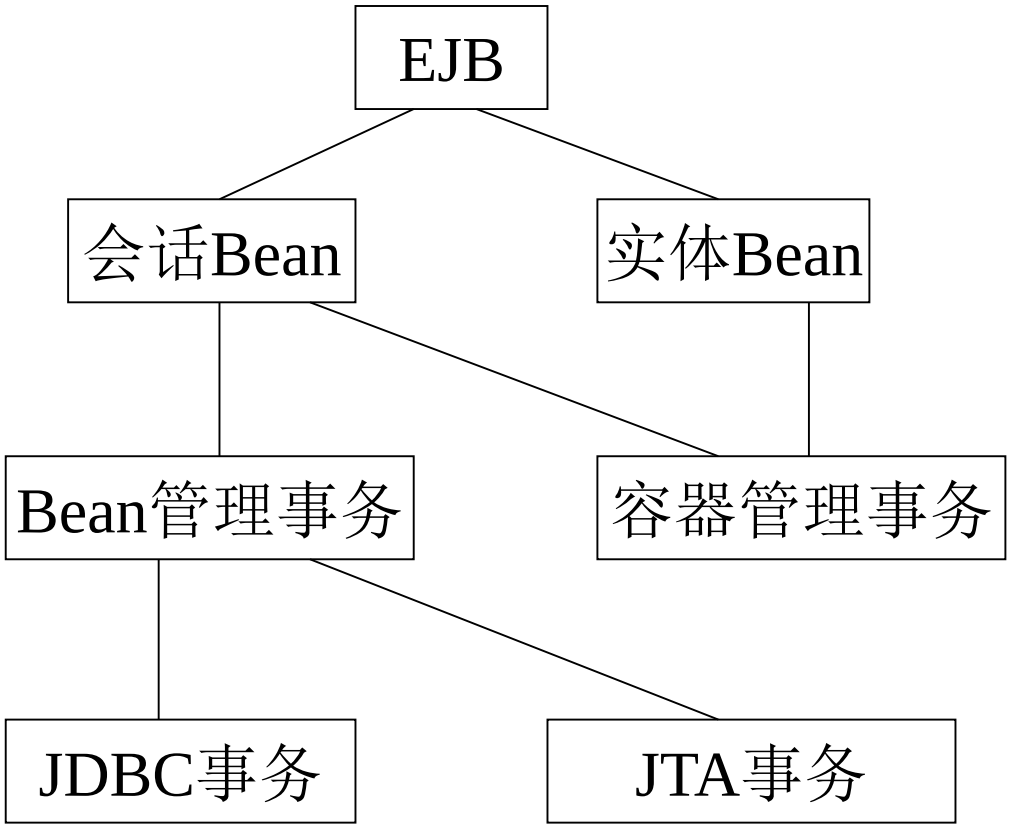
- 编码Bean管理的事务时，只可使用JDBC和JTA事务
- JDBC事务被数据库管理系统的事务管理器所控制
  - 使用Connection接口：setAutoCommit()、commit()、rollback();
- JTA事务受J2EE事务管理控制器控制，能够跨越多个来自不同厂商的数据库进行更新。
  - 使用UserTransaction接口：begin()、commit()、rollback()

JTA使用实例

```
UserTransaction ut = context.getUserTransaction();
try{
    ut.begin();
    updateChecking(amount);
    machineBalance -= amount;
    inserMachine(machineBalance);
    ut.commit();
}
catch(Exception ex){
     try{
          ut.rollback();
    }
    catch…..
}
```



EJB事务可选方式总结



主要内容

- 概述
- EJB 2.0的开发
- EJB分类
- EJB容器
- EJB小结

EJB小结

- EJB技术的主要思想就是让“恰当的专家做恰当的事情”
  - 应用领域的开发人员将开发精力放在应用逻辑方面，而不用考虑底层的计算技术；
  - 计算机专业开发人员去处理底层的计算技术细节，而不用考虑应用领域的专业知识
- 在开发EJB的时候
  - 根据应用的特点，选择合适的EJB类型
  - 根据应用的特点，使用容器提供的事务、安全等服务