参考：<http://blog.csdn.net/houjiyu243042162/article/details/41205979>

[EJB详解](http://blog.csdn.net/houjiyu243042162/article/details/41205979)

**Chapter01 企业级开发背景知识**

一、什么是企业级程序（Enterprise Application）？

  具有以下特点的程序：

  (1) 围绕商业目的；

  (2) 分布分层的程序[**架构**](http://lib.csdn.net/base/architecture)（业务逻辑分层，服务部署分布）。

二、企业级应用的架构发展历史

Host/Terminal（主机/终端，终端不具备处理能力，数据由主机处理）

终端为哑终端，可接受命令，不处理命令

优点：安全（不能直接访问）

    性能优越（目前关键应用仍在使用）

缺点：使用代价过高，设备昂贵1·

    升级维护困难

C/S（Client/Server客户端/服务器端）

  终端拥有一定的计算能力（而ejb正式将复杂的加速从客户端放在服务端），应用架构演变为C/S架构

Multi-layered（多层架构）

  Sample：Browser——Web Server——EJB Server——Database

三、为什么要使用EJB，即使用EJB的动机是什么？

  (1) 大型的企业级程序分解为分层分布的应用，必须使用EJB;

  (2) 构建多层程序架构；

  (3) 各层独立，专注自己擅长的领域；

  (4) 采用EJB可以不用考虑中间件服务（Middleware Service）

Tips：中间件服务

1，RMI（Remote Method Invocations, 远程调用）

2，Load Balancing(负载均衡，将访问负荷分散到各个服务器中)

3，Transparent Fail-over(透明的故障切换)

4，Clustering(集群,用多个小的服务器代替大型机）

5，Back-end-Integration(后端集成，用现有的、新开发的系统如何去集成遗留的系统)

6，Transaction事务（全局/局部）全局事务（分布式事务）局部事务（在同一[**数据库**](http://lib.csdn.net/base/mysql)联接内的事务）

7，Dynamic Redeployment(动态重新部署,在不停止原系统的情况下，部署新的系统）

8，System Management(系统管理)

9，Threading(多线程处理)

10，Message-oriented Middleware面向消息的中间件（异步的调用编程）

11，Component Life Cycle(组件的生命周期管理)

12，Resource pooling（资源池）

13，Security（安全）

14，Caching（缓存）

四、什么是EJB？

EJB是采用[**Java**](http://lib.csdn.net/base/java)语言开发、部署服务器端的分布式组件的标准和框架。

分布式组件：

程序=业务+底层中间件服务

组件：按照一定的规范开发，并部署到相应的容器中适当的对象和相关部署文件。

分布式组件：

多个容器之间的组件需要进行跨容器调用。

EJB：动态可伸缩的、可靠、安全的组件框架。

   与中间件服务提供商无关，采用EJB开发，不用考虑容器。

五、EJB可以用在哪些地方？

  业务运算。

六、EJB的价值

  (1) 行业标准(Industry standard)

  (2) 便携性

  (3) 快速开发

七、EJB的运行机制

  (1) 代理/委托模式

代理模式：存根（Stub）和拦截器（Request interceptor）都使用了同一个远程接口（Remote Interface）。

委托模式：拦截器将业务请求委托EJB来负责。

  (2) 显示中间件服务：

  缺点：生产率底下

        代码维护困难（中间件服务与业务逻辑纠缠）

  优点：细粒度地控制中间件服务

    什么叫粒度：对象的相对大小，对客户端暴露的细节的程度。

  (3) 隐式中间件服务：不直接调用中间件，采用配置文件来声明中间件服务，容器自动调用。

  优点：相对与显示中间件服务，开发进度加快，容易维护。

  缺点：粗粒度，只能在方法级来声明中间件服务，稍微影响性能。

API:

Request interceptor：拦截器，拦截中间件服务，负责调用相应的中间件API，业务调用交给相应的API。

Stub/Skeleton：屏蔽了网络调用的细节，拦截器屏蔽了中间件调用的细节，EJB只需考虑对象的调用，

Stub、Skeleton、Request Interceptor由容器生成。

编程需要写的类：业务接口（Remote Interface），EJB，客户端（Client，用于访问EJB）。

  EJB是怎么运行的：

  (1) 开发出来的EJB对应的Stub被应用服务器或容器（如 Sun Application Server）绑定到JNDI服务器，名称为全限定名。

  Stub相当于远程对象在本地的代理，屏蔽了网络调用，编码，解码，协议转换等细节。

  (2) 客户端通过上下文（Context），找到Stub。

八、EJB生态系统的八个角色

九、SOA（面向服务架构）

  (1) 程序由多个服务构成；

  (2) 服务是由一组相关的组件构成，完成某个特定的业务功能，服务可由不同的编程语言实现；

  服务通过HTTP协议，SOAP、WSDL等相关协议实现的一种分布式应用架构。

SOA（面向服务的架构）和EJB的关系

SOAP（简单对象访问协议-类似http+xml+rpc）

WSDL（Web服务描述语言）（Web Services）

EJB发布为Web服务，从而实现SOA。

十、[**Java**](http://lib.csdn.net/base/java)EE相关知识

  (1) [**java**](http://lib.csdn.net/base/java)EE是一个标准；

  (2) [**Java EE**](http://lib.csdn.net/base/javaee)是用来开发多层、分布式企业应用的平台。

**Chapter02 EJB开发流程**

一、开发EJB的步骤

第一大步：开发编译业务接口和Bean类

(1) 编写业务接口和Bean类

(2) 编译

第二大步：打包部署到JNDI服务器

(3) 提供部署描述文件

(4) 打包（编译后的字节码文件和部署描述文件）成ejb-jar

(5) 部署ejb-jar到容器

第三大步：[**测试**](http://lib.csdn.net/base/softwaretest)

(6) 检测部署是否成功（看容器是否发现）

(7) 识别客户端程序来调用EJB

业务接口（Business Interface）：

(1) 对客户端暴露可以调用的方法，它是一个普通的Java接口，POJO类，或POJI。

(2) 业务接口可以划分为两类：

A、远程业务接口

              B、本地业务接口

如果客户端和EJB运行在同一个JVM中，我们应该用本地接口，否则只能用远程业务接口。

二、EJB的编程模型和编程涉及的标注

Bean类（The Bean Class）：

(1) Bean类也是一个POJO，实现了业务接口中的方法

(2) 在Bean类中一般带有标注（Annotation），用于代替传统的部署描述文件

 @stateless 无状态（标注该类是一个无状态的会话Bean）

 @Remote（XXX.class）（标注该Bean类的业务接口是XXX.class）

(3) Bean类要有缺省的构造方法

(4) Bean类可以不实现业务接口（因为标注已经说明），不过建议实现

(5) ejb-jar.xml（Optional）标准的部署描述文件

  我们可以用EJB3.0中的标注来代替对描述文件的编写。

EJB3.0使用了[**Java SE**](http://lib.csdn.net/base/javase) 5.0 中的注释新特性，只要在方法上写EJB规范的注释，就可以不用来写标准的部署表述，ejb-jar.xml这个要放到jar文件的mate-inf文件夹里。

<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar   xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   metadata-complete="true" version="3.0"   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar\_3\_0.xsd">

 <enterprise-beans>

  <session>

    <display-name>HelloBean</display-name>

    <ejb-name>HelloBean</ejb-name>

    <business-remote>test.HelloRemote</business-remote>

    <ejb-class>test.HelloBean</ejb-class>

    <session-type>Stateless</session-type>

    <transaction-type>[**Container**](http://lib.csdn.net/base/docker)</transaction-type>

    <security-identity>

        <use-caller-identity/>

    </security-identity>

  </session>

 </enterprise-beans>

</ejb-jar>

EJB Client（客户端）：

  (1) EJB不能够被客户端直接访问，客户端必须通过业务接口来调用EJB

  (2) EJB客户端分为两类：

    A、EJB客户端与EJB运行在同一个容器中→本地业务接口

    B、EJB客户端与EJB运行在不同的JVM中→远程业务接口

EJB 容器（EJB Container）：

  (1) EJB容器是一个运行在JVM上的程序

  (2) EJB容器为EJB提供中间件服务。

EJB客户端的编程步骤：

1、运行客户端前要加上在SUN的服务器文件夹中的lib下的javaee.jar和appserv-rt.jar

Tips：Java Runtime相关的包：javaee.jar([**java ee**](http://lib.csdn.net/base/javaee))，appserw-rt.jar(Java 运行时) ，appserv-ws.jar appserv-jwsacc.jar（Web Service）

2、配置系统的JNDI的环境（客户端所依赖的Object和Sun运行环境）

3、写测试类，注意在测试类中也要有业务接口

  Step1：设置JNDI服务器入口信息（ system.setProperty）

  Step2：初始化上下文

  Step3：查找JNDI服务器的Stub

  Step4：调用对象

EJB调用过程：

  (1) EJB容器将Stub绑定到JNDI服务器上，JNDI上使用的绑定（默认情况下）是远程业务接口的全限定名。

  (2) 客户端通过上下文（Context）的方法来查找JNDI服务器上绑定的Stub。

在厂商的部署描述中可以配置EJB的实例池，可以一次创建多个EJB的实例。

注意：EJB一定要先部署到服务器，EJB需要部署环境才能运行

EJB客户端

public class Cilent {

    public static void main(String[] args) throws NamingException {

        System.setProperty("jndi.factory","com.sun.jndi.cosnaming.CNCtxFactory");

        System.setProperty("jndi.provider.url","corbaloc::localhost:3007/NameSpace");

        Context ctx=new InitialContext();

        HelloRemote hello=(HelloRemote)ctx.lookup("test.HelloRemote");

        System.out.println(hello.hello());

        System.out.println(hello.helloword("lichen"));

    }

}

注意：在写客户端时，要引入javaee.jar和appser-rt.jar还要引入已经编译好的EJB的jar文件。

**Chapter03 会话Bean**

一、什么是会话Bean（Session）？

  会话Bean：

  (1) 会话Bean是EJB中一种用来建模、业务逻辑/业务流程的EJB类型。

  (2) 会话Bean实例的生成是客户端调用的结果。

  (3) 会话Bean接受客户端的请求，并作出响应，是一个业务处理对象（控制类，OO中类的一种（边界类，实体类，控制类））

  (4) 会话Bean的存活时间很短。

常用的EJB类型：SessionBean，EntityBean，MessageDrivenBean，Entity Bean，JPA。

1、SessionBean（Statless/Statfull）

    无状态的SessionBean是不会保存的信息的和状态，无状态的SessionBean是可以重用的，而不是被立刻销毁

    有状态的SessionBean只为一个用户来服务。

2、MDB（Message Driver Bean）消息驱动Bean，可以进行异步的编程

3、Entity Bean 访问数据库，已经被JPA（JAVA持久化API）替代。

4、JPA基于ORM方式来实现对数据库的访问

1.SessionBean：

    用于对业务建模，表示过程。流程（动词）

    作为门面（Facade）。

    负责接受远程访问，并操作EntityBean。同步的调用。

2.EntityBean：

    生命周期：持久化。过程结束后，数据仍存在。

    用于对数据建模，表示名词。

    SessionBean与EntityBean组成动宾关系。

    争论焦点：O/R较复杂。

    一般不直接接受远程访问。

3.MessageDrivenBean：

    用于封装业务过程，动词。

    为了完成异步调用：客户调用，消息服务器缓存指令，发送消息给客户端，消息服务器分发消息，Bean执行。

    责任同SessionBean，完成异步调用。

二、SessionBean的类型：

  (1) stateless Session Bean

(2) stateful Session Bean

（一）无状态SessionBean（Statless）

  (1) 无状态Session Bean单个方法调用就能完成一个完整的业务流程

(2) 无状态Session是可以被重用，不能被客户端并发共享，只可以串行共享，并不保留客户端方法调用后的的状态，而是直接返回。

(3) 无状态的SessionBean是可以池化的（pooling）（类似apache服务器启动时初始化几个连接池以便于在访问量大的时候直接使用避免初始化提高性能），以优化性能，用以被多个客户共享。

无状态SessionBean的生命周期

如果实例不存在，就会调用构造方法，然后调用资源注入方法，接着会调用有@PostConstruct标注的方法，在销毁时会调用有@PerDestroy标注的方法，

然后销毁对象，如果实例存在就会从Bean实例池中取出实例并调用方法。

回调方法是基于事件机制的。

生命周期回调方法的规则

1、对于直接定义在Bean中的回调，其格式应该是public void <method()>，也就是直接写在SessionBean中。

2、对于为Bean类单独提供(单个或多个)回调监听而言。其格式是public void <method(InvocationContext context)>

java.interceptor.InvocationContext，提供了调用的上下文信息。只在SessionBean类上加上@Interceptors(Xxxx.class)制定定义了回调方法的类。

3、回调方法不能够抛出已检查异常，但可以抛运行时异常。

在创建实例并注入资源之后，会回调这个方法。

@PostConstruct

public void constructed(){}

在销毁对象时回调这个方法

@PerDestroy

public void destroy(){}

（二）有状态的SessionBean（Statful）

有状态的SessionBean只为一个客户端服务，不能共享，并且会保留方法调用后的状态。

  (1) 多个方法调用才能完成一个业务处理流程；

  (2) 需要保留客户端的状态

  (3) 不被多个客户共享。

当有状态的SessionBean暂时不被使用时，就会被存储到缓存当中，也就是被存到虚拟内存或者是将信息同步到Session数据库中（Session数据库是应用服务器所提共的小型数据库，用来保存Session的信息，多应用服务器共享Session数据库，同步Bean的信息，达到集群处理）。

swap out passivation钝化过程，保存SessionBean的状态，在SessionBean处在事务中时是不会进行钝化的。

swap in activation激活过程，提取保存的SessionBean到内存中，可以继续被客户端使用。

有状态的Session，会保存成员变量（没有加transient关键字，以及串行化的类型和基本类型），也会保存其他的SessionBean的引用。

对有状态的SessionBean中有transient属性时，就需要在Bean中提供激活的方法也就是

@Statful

Test implements TestRemote{

   transient private int a;

   @PostActivate

   public void activate(){

  a=5;

   }

}

在SessionBean中实现事务处理时要实现SessionSynchronization接口。

  关于有状态会话Bean与无状态会话Bean的比较：

  Stateless Session Bean

  (1) EJB容器可以预先实例化多个无状态会话Bean的实例，组成一个pool；

  (2) 当客户请求调用时，容器可以任选一个实例处理请求；

  (3) EJB容器可以依照请求量的大小来调整pooling中实例的个数；

     对于客户端来讲，这些stateless Session Bean毫无区别。

  (4) EJB容器可以通过pooling来达到使用少量实例来应对大量的请求。

  Stateful Session Bean

  对于有状态的会话Bean，不能够同时被多个客户所共享。

  如果要让stateful Session Bean能够同时服务多个客户，就要使用Swapping。

  钝化：将会话Bean中客户状态钝化到存储设备（swap out）。

  激活：将存储设备中的客户状态读回到会话Bean中（swap in）。

使用有状态会话Bean时如何维护客户状态？

  (1) 在默认状况下，可以不用考虑，容器会自动维护客户端的状态

  Tips：容器自动维护的状态

  A、非transient修饰的属性

  B、EJB中的其他业务接口

  C、EJB中的home接口

  D、EJB中的会话上下文，事务对象

  E、JNDI上下文（Context）

  (2) 对于一些无法采用默认的swap机制钝化、激活的状态，应该采用回调方法手动实现激活和钝化。

  1、何时需要使用有状态的会话Bean？

  需要跨多个方法完成某个业务处理流程，并且需要保持客户端状态。

  2、有状态会话Bean能否实现pooling？

  可以，但需要swping，不停地激活、钝化。

  3、无状态会话Bean的pooling性能优于有状态会话Bean的pooling（swap有I/O瓶颈）。

  4、无状态会话Bean可以有属性，但是属性的值被所有的客户端共享。

在厂商的部署描述中可以配置EJB的实例池，可以一次创建多个EJB的实例。

SessionBean

SessionBean接受客户端的请求，并作出响应，是商业过程处理对象。

无状态SessionBean（Statless）

无状态Session是可以被重用，不能被客户端并发共享，只可以串行共享，并不保留客户端方法调用后的的状态，而是直接返回。

无状态的SessionBean是可以池化的，以优化性能。

无状态SessionBean的生命周期

如果实例不存在，就会调用构造方法，然后调用资源注入方法，接着会调用有@PostConstruct标注的方法，在销毁时会调用有@PerDestroy标注的方法，然后销毁对象，如果实例存在就会从Bean实例池中取出实例并调用方法。

回调方法是基于事件机制的。

生命周期回调方法的规则

1，对于直接定义在Bean中的回调，其格式应该是public void <method()>，也就是直接写在SessionBean中。

2，对于为Bean类单独提供(单个或多个)回调监听而言。其格式是public void <method(InvocationContext context)>

java.interceptor.InvocationContext，提供了调用的上下文信息。只在SessionBean类上加上@Interceptors(Xxxx.class)制定定义了回调方法的类。

3，回调方法不能够抛出已检查异常，但可以抛运行时异常。

在创建实例并注入资源之后，会回调这个方法。

@PostConstruct

public void constructed(){}

在销毁对象时回调这个方法

@PerDestroy

public void destroy(){}

有状态的SessionBean（Statful）

有状态的SessionBean只为一个客户端服务，不能共享，并且会保留方法调用后的状态。

当有状态的SessionBean暂时不被使用时，就会被存储到缓存当中，也就是被存到虚拟内存或者是将信息同步到Session数据库中（Session数据库是应用服务器所提共的小型数据库，用来保存Session的信息，多应用服务器共享Session数据库，同步Bean的信息，达到集群处理）。

swap out passivation钝化过程，保存SessionBean的状态，在SessionBean处在事务中时是不会进行钝化的。

swap in activation激活过程，提取保存的SessionBean到内存中，可以继续被客户端使用。

有状态的Session，会保存成员变量（没有加transient关键字，以及串行化的类型和基本类型），也会保存其他的SessionBean的引用。

对有状态的SessionBean中有transient属性时，就需要在Bean中提供激活的方法也就是

@Statful

Test implements TestRemote{

   transient private int a;

   @PostActivate

   public void activate(){

  a=5;

   }

}

在SessionBean中实现事务处理时要实现SessionSynchronization接口。

java -Dorg.omg.CORBA.ORBInitialHost=192.168.12.41

**Chapter04 Web Service**

一、Web Service的概念和原理

1、什么是Web Service？

WebService是一个SOA（面向服务的编程）的架构，它是不依赖于语言，不依赖于平台，可以实现不同的语言间的相互调用，通过Internet进行基于Http协议的网络应用间的交互。

WebService实现不同语言间的调用，是依托于一个标准，webservice是需要遵守WSDL（web服务定义语言）/SOAP（简单请求协议）规范的。

WebService=WSDL+SOAP+UDDI（webservice的注册）

  Soap是由Soap的part和0个或多个附件组成，一般只有part，在part中有Envelope和Body。

  Web Service是通过提供标准的协议和接口，可以让不同的程序集成的一种SOA架构。

2、Web Service的优点

  (1) 可以让异构的程序相互访问（跨平台）

  (2) 松耦合

  (3) 基于标准协议（通用语言，允许其他程序访问）

3、Web Service的基本原理

  (1) Service Provider采用WSDL描述服务

  (2) Service Provider 采用UDDI将服务的描述文件发布到UDDI服务器（Register server）

  (3) Service Requestor在UDDI服务器上查询并 获取WSDL文件

  (4) Service requestor将请求绑定到SOAP，并访问相应的服务。

Tips：WSDL与SOAP

  WSDL：Web Service Description Language（Web服务描述语言）。

  作用：

  (1) 描述服务是什么，服务提供的操作（方法名，方法参数，返回值，数据类型等）；

  (2) 如何访问该服务→binding

  (3) 服务访问的地址→service

  Banding：定义Web服务访问采用的协议（SOAP）

  SOAP：Simple Object Access Protocol（简单对象访问协议）

4、如何编写Web Service？

  (1) 先编写Java类，然后利用工具生成WSDL文档以及相关文档

    或先编写WSDL文档，后利用工具生成Java类

  (2) @Web Method标注的方法要发布到WSDL描述中，如果没有任何方法加注该标注，则所有方法均发布到WSDL上。

  @WebService标注该Session Bean要发布Web Service。

二、如何访问Web Service？

  Dynamic Proxy：动态代理

  Dynamic invocation intenface：动态调用接口

  Dynamic Proxy：

  URL：Service的WSDL地址

  QName：全限定名

  Namespace：目标名字空间，Web Service的名字

  Service类：javax.xml.ws.Service

  实例：

Service greeterService=Service.create(wsdlLoction,serviceQName);

  helloWSBean hws=greeterService.getPort(helloWSBean.class)

  System.out.println(hws.sayHello());

EJB中使用WebService

@WebService(serviceName="",portName="")，使用这个标注可以将SessionBean中用@WebMethod标注来表示的方法发布成WebService

@Stateless

@WebService(serviceName="Greeter",portName="GreeterPost")

public class HelloSessionBean implements HelloSessionRemote {

    @WebMethod

    public String hello(String name) {

        return "Hello world "+name+"!";

    }

}

三、Web Service总结

1、Web Service的优缺点

优点：① 跨平台，可以穿透防火墙，构建一个分布式系统

    ② 基于标准协议（HTTP，SOAP，WSDL，XML）

    ③ 松耦合

缺点：性能低下（SOAP协议XML解析，采用HTTP协议等原因）

分布式系统（EJB，DCOM，CORBA，Web Service）缺点：

  ① EJB不能穿透防火墙，而且限制编程语言只能是Java

  ② 技术不完全成熟

2、Web Service应用场合

  (1) 用于企业内部的应用集成

  (2) 用于企业间的应用集成

**Chapter05  Entity 基础**

一、Entity的概念

  1、什么是Entity？

  Entity是一个POJO，JPA（Java Persistence API）可以使用实体来持久化。

  2、JPA的特点

  ① JPA是一种标准的持久化机制

  ② 在容器内外均可以运行

  ③ 持久化提供者（[**hibernate**](http://lib.csdn.net/base/javaee)等）不影响JPA的使用

  3、Entity与Session Bean的区别

  (1) Session Bean用来建模业务流程，是一个动作；

  (2) Entity表示被持久化的数据，也可以认为是在内存中的Java对象的表示；

  (3) Entity生命周期很长，会话Bean表示业务逻辑，它的生命周期很短；

  (4) Session Bean是可被远程客户调用的，Entity不能被远程客户调用；

  (5) 实体不是EJB，Session Bean是EJB。

EJB3.0的JPA（Java持久化API）

O/R Mapping（对象关系映射）

TopLink，JDO，Hibernate

类型对应表，属性对应字段，关系对应引用

BO（商业对象，操作数据对象）

DO（数据对象）

持久化的数据对象，也就是已将对象信息同步到数据库中的对象，持久化对象也叫实体。

操作实体也就使操作实体在数据库中所对应的数据。

实体和SessionBean的区别

实体本身不支持远程访问，他的生命周期是比较长的。

实体有唯一性标识，也就对应数据库表中的主键。

注意：在实体中不要写商业方法

实体的唯一标识，可以使用标签@Id（标识属性可以使用public描述，也可以完全封装为其提供set，get方法），也可以使用XML文件来进行配置。

二、实体的编程

@Entity(name="Account")，实体类标注，其属性name是指定实体名，在EJB-QL中使用，默认是类的全名

@Id，指定实体的唯一标识属性，默认这个属性会合数据库中对应表的主键对应。

@GeneratedValue(strategy = GenerationType.AUTO)指定主键的生成策略。

@Colum(name="...",unique="true|false",nullable="true|false",insertable="true|false",

updateable="true|false",table="...")，指定类中属性对应的列名以及约束，

name属性指定类中属性对应的列名，默认为属性名

unique属性指定类中属性对应的列是否唯一，默认为false

nullable属性指定类中属性对应的列是否可空，默认为true

insertable="true|false"属性指定类中该属性是否会出现在insert语句中，也就是会不会被同步到数据库，默认为true，也就数会同步到数据库

updateable="true|false"属性指定类中该属性是否会出现在update语句中，也就是会不会被修改，默认为true可以被修改。

table属性指定类中属性的列所对应的表，默认为实体类所对应的表。

在使用实体同步到数据库时，SessionBean中要写EntityManager类型的属性，这个属性在Bean部署在容器中后，在运行时会容器依赖注入，如果没有容器也可以使用，但需要为其赋值。

EntityManager是一个接口，也就是规则，可以有不同的实现，Hibernate3.2就实现了这些JPA的接口。

  实体必须提供默认的构造方法，getter，setter方法。

  实体可以有业务方法，用于属性操作。

实体的状态

new新建，也就是新建的实体实例，其信息还没有持久到数据库中。

managed受管状态，也就是实体已经持久化到数据库中，并且已经和持久化上下文进行了关联。

detached分离状态，也就是与持久化上下文解除关联的实体的状态

removed删除，此时实体和持久化上下文进行了关联，但是要从数据库中删除这个实体。

  new persist()  中止PersistenceContext >

  ——>新建————>受管========================分离

                     ||\   <  merge()

            remove()\|| persist()

                    删除

@PersistenceContext，持久化上下文是内存中的实例和数据库间的连接枢纽，就像是一快缓冲区，但这个缓冲区是由容器来进行管理的，在这个缓冲区中的实体是处在受管理状态。

@PersistenceContext(type=PersistenceContextType.EXTENDED,unitName="PetPU")

type属性使用来标识持久化上下文的类型的，持久化上下文有两种类型事务范围和扩展的。这两种类型的持久化上下文的生命周期不同。unitName属性是指定持久化单元的名字，其值是在持久化单元定义文件中persistence-unit标签中的name属性的值。

  Psersistence Context（持久化上下文）：

  持久化上下文表示一组实体，这些实体被Entity Manager所管理。

  两种Persistence Context：

  (1) 事务范围的持久化上下文，一般用于五状态的会话Bean。在默认情况下，EJB容器对每一个业务方法都会提供事务支持。

  (2) 可扩展的持久化上下文：用于有状态的Session Bean，当方法结束，持久化对象还会存在。

  可扩展的持久化上下文会在各个方法中共享。

  Extended Persistence Context：

  当方法调用结束是，persistence Context会继续存在。只有当EJB实例销毁是，才会删除。

  Extended Persistence Context可用在有状态的会话Bean中，用来缓存客户端的实体状态。

  @persistence Context→持久化单元信息注入

  type=PersistenceContextType.EXTENDED 可扩展的持久化上下文

  unitName=“firstejb”→持久化单元的名字

  Tips：持久化单元（配置文件persistence.xml）的作用：

  (1) 持久化提供者

  (2) 持久化单元的名字

  (3) 设置表的生成机制

关于Extended Persistence Context：

  (1) 在容器管理下，由实体管理器（EntityManager）创建

  (2) @persistence Context注入

    可以脱离容器运行，有EntityManagerFactory来创建。

事务范围的持久化上下文应用到无状态SessionBean，一旦事务终结，就销毁，其中的实体也会变成分离状态。

扩展的持久化上下文应用在有状态的SessionBean，只有在有状态的SessionBean被容器销毁是才会销毁持久化上下文，也就使实体一直是被管理的。

持久化单元的定义文件，这个文件是描述数据库连接和事务管理的文件

<persistence   version="1.0"   xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">

  <persistence-unit name="PetPU" transaction-type="JTA">

    <!--transaction-type是事务管理的类型-->

    <provider>[**Oracle**](http://lib.csdn.net/base/oracle).toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider</provider>

    <!--SPI，持久化API的实现类的提供者-->

    <jta-data-source>my.jdbc</jta-data-source>

     <!--数据源的配置-->

    <properties>

      <property name="toplink.ddl-generation" value="create-tables"/>

    </properties>

  </persistence-unit>

</persistence>

三、EntityManager<interface>

EntityManager实体管理器，它是用来管理实体的，如果使用容器管理实体，容器就会注入EntityManager，EntityManager可以提供实体类生命周期管理，实现数据的同步，和实体的数据的查询。

  EntityManager，一个用来操作实体，包括增删查改以及实体的生命周期的管理的接口。

实体生命周期回调方法

实体的生命周期回调方法不是通用的。

在实体类中定义时，只在方法前加标注，并且方法的返回值为void，方法为public

也就是：

@PrePersist

public void save(){}

在另外的类中写生命周期回调方法时，就需要以这个实体类为参数了

class AcountListenter{

@PrePersist

public void save(Acount a){}

}

@Entity

@EntityListenters(AcountListenter.class)

class Acount...

PostPersist

PreRemove

PostRemove

PreUpdate

PostUpdate

PostLoad

怎样操作EntityManager？

  由容器依照持久化的单元信息来创建，以及实体生命周期管理。并且，可以用@Persistence Context来注入到程序中。

EntityManager API：

  (1) Entity life-cycle management 实体生命周期管理

  (2) Database syschronization operations 数据库同步操作

  (3) Entity lookup and queries  实体查询

Entity life-cycle：

  ① new（暂态）：实体没有纳入容器的管理，没有主键，数据库无记录

  ② managed（受管态/持久化状态）：处于容器管理之下，有主键，数据库有记录

  ③ removed（删除状态）：处于容器管理之下，此时可以通过persist（）方法返回managed状态

  ④ detached（游离态）：不处于容器管理下，数据库不确定是否有对应记录。

四、实体的回调

  回调是由持久化管理器来调用的，不是由EJB容器调用的。

  回调的作用是什么？

  在实体被JPA操作时，可以作预处理和后处理。

  回调方法可以定义在Entity内部，也可以定义在单独的监听器类中。如果定义在监听器里，回调方法必须带一个参数，即要监听的实体。

  @EntityListeners（XXXLister.class）监听器可以有多个，用{ ，，}表示。

五、实体的[**版本控制**](http://lib.csdn.net/base/git)（锁机制）

(1) 在JPA中，我们可以使用乐观锁（Optimistic Locking），但必须是以下几种类型：

  int,Integer,short,Short,long,Long,Timestamp

(2) 采用@version来标注version字段，程序不用是对version字段进行任何操作，version字段值由容器来维护。

@NamedQuery（name=”findAll”,query=”select……”）静态查询语句，可以用标注。

多条语句

@NamedQueries（{@NamedQuery（），@NamedQuery（）} ）

**Chapter06  消息驱动Bean**

一、JMS（Java Message Service）

  MOM：Message Oriented Middleware（面向消息中间件）

  消息中间件的作用：

  (1) 接收消息发送方发送的消息

  (2) 将接收的消息可靠地发送给消息接收方

MOM系统中的角色：

  MOM消息中间件：Server程序

  消息客户端：消息收发方，Client，消息发送方（消息生产者），消息接收方（消息消费者）

常见的MOM：

  IBM WebSphere MQ

  BEA tuxedo/Q

  Tibco Rendezvous

  Microsoft MSMQ

JMS API

  作用：消息程序不再依赖特定的MOM。

JMS支持的两种编程模型：

  (1) Publish/Subscribe(pub/sub) 发布/订阅模型

  (2) Point-to-Point（PTP）点对点模型

  pub/sub：

    ① 用Topic存放消息

    ② 允许有多个生产者和消费者，同一个消息可被多个消费者消费，且在Topic中不会因消费而删除。

  PTP:

    ① 消息服务器上用Queue队列来存放消息

    ② 允许多个消息的生产者发送消息到Queue，但是消息只允许一个消息消费者消费。一旦消息被消费，MOM会把消息从Queue中删除。

JMS编程模型的共同点：

  消息生产者发送消息到MOM，并不关心消息消费者是谁，何时消费该消息，JMS也不会影响正常的业务逻辑代码的运行。

JMS编程步骤：

Step1：配置管理对象（在Application server上）

  ConnectionFactory：消息连接工厂，负责创建连接

  Queue/Topic：消息目的地，负责存放消息

Step2：分别编写消息发送/接收程序

TopicSession session=connection.createTopicSession(false,TopicSession.AUTO\_ACKNOWLEDGE);

  AUTO\_ACKNOWLEDGE：不使用事务，使用自动应答的方式来确认消息

  默认情况下（使用事务），消息的确认：当事务结束，消息就会得到确认。

  对于pub/sub模型：

  订阅者（消息接收者）必须先运行（原因是消息消费者要负责启动消息连接）

  对于PTP模型：消息发送者和消息接收者无运行顺序限制。

二、MDB（Message-Driven Bean）

Message-Driven Bean（消息驱动Bean）

  MDB是一种EJB，用来监听消息目的地的消息，并且处理该消息（相当于消息消费者）。

消息驱动Bean，可以实现异步编程，也就是发送消息，而不必等待返回确认，也就像发邮件一样，只是发出，而阻塞不必等待回应。

消息驱动Bean是在接受到消息之后再通过onMessage方法进行消息的处理，也就是消息的消费者，消息驱动Bean也是无状态的。

  MDB的特点：

  (1) MDB不能被客户端直接调用，因为MDB没有本地/远程接口；

(2) MDB是一个无状态的EJB，不能保存客户端状态，可以方便地构造一个池（pooling）

(3) MDB不能返回异常给客户端；

(4) MDB通过实现onmessage（）方法来处理消息。

    public void onMessage(Message message)  {

        if(message instanceof TextMessage){

            TextMessage tm=(TextMessage)message;

            String s = null;

            try {

                s = tm.getText();

            } catch (JMSException ex) {

                ex.printStackTrace();

            }

            try {

                tm.setText("reserve"+s);

            } catch (JMSException ex) {

                ex.printStackTrace();

            }

        }

}

MDB编程：

  (1) javax.jms.ManagerListener必须实现这个接口

     javax.ejb.MessageDrivenBean(Opitional)

  (2) 必须提供缺省的构造方法（容器随时会通过构造方法构造MDB）

  (3) 可以通过@MessageDriven标注该类为一个消息驱动Bean

    @MessageDriven（mappedName=“jms/Topic”）告诉容器，该MDB监听消息的目的地。

  观察者模式：只有实现了MessageListener，有消息就会处理消息。

  不要在onmessage（）方法中抛出异常，应该在catch语句块中处理异常。

三、MDB高级特性：

Transaction（事务）

  消息的发送与消息的接收不能处在同一个事务中（消息收发可能阻塞）。

Security（安全）

  MDB（消息消费者）不能组织非安全消息的发送。

Load balancing（负载均衡）

  MDB采用拉模型去构造负载均衡（监听到有消息，就会进行“拉过去”处理），Session Bean采用推模型（在会话Bean的实例化池中，来一个客户端请求就“推出去”一个Session Bean去处理请求），因此MDB是一种比Session Bean更好完成负载均衡功能的EJB。

Clustering（集群）

  在集群技术中，MDB常发生重复消费(Duplicate consumption in a cluster)，用于保证业务(Stateful Session Bean)执行的连续性。

**Chapter07  资源注入**

一、EJB之间的相互调用

  1、EJB之间的相互调用可以采用以下四种方式：

  (1) 使用默认的JNDI上下文

  (2) 使用@EJB标注（推荐）

  (3) 使用EJB Context查找

  (4) 使用本地接口

EJB扩展组件

EJB的依赖注入，都是依赖于JNDI的。

在EJB组建中调用其他的EJB组件

2、EJB调用方式：标注

@EJB：

可以借助于JNDI，查找到其他EJB组件，也可以借助于业务接口，调用EJB实例中的业务方法，也可以直接引用其他的需要的EJB组件的业务接口，但是需要使用标注。

标注类型一：

@EJB

private TaxRateRemote taxRate;

需要配置ejb-ref值，配置文件是容器特定的，并且ejb-ref-name的值是 lab.morefunnction.PriceBean/taxRate，lab.morefunnction.PriceBean是要访问的EJB的全限定名，taxRate是被访问的EJB引用。

标注类型二：

@EJB（name=“taxRate”）

private TaxRateRemote taxRate;

同样需要配置（sun-ejb-jar.xml）中的ejb-ref-name的值：“taxRate”，EJB引用名与JNDI名字之间的映射。

3、EJB调用方式：EJB Context查找

EJB Context<interface>

  EJB容器提供给EJB的一个对象，用来给EJB访问EJB容器调用的一个接口。

二、EJB的固有标注

@Statless @Statful @MessageDriven

name=" "，mappedName=" " description=" "

name属性是标识EJB的名字（可以通过JNDI来查找）。默认是EJB的名字。

mappedName属性也是标识EJB的名字的是容器相关的，所以不要重置，降低可移植性。

description属性是EJB的描述内容。

以上3个属性是3个标签共有的属性。

@Remote @Local

这两个标注是标注业务接口的，分别是标注远程的和本地的商业接口的，如果不写那就默认为本地的。

① 不能直接调用EJB，必须通过业务接口去访问；

② 如果客户端与EJB运行在同一个容器当中，用本地业务接口访问，则性能更高。因为使用本地业务接口，就避免了RMI调用，即避免了网络调用的性能损耗。通过Reference（引用），而不是Stub，Skeleton编码解码，网络传输。

③ 一个EJB可以同时具有本地业务接口和远程业务接口。

④ 如果有多个业务接口，可以采用@Remote（{业务接口1，业务接口2}）

  此时EJB在JNDI上绑定的JNDI名字为：“#”+业务接口的全限定名。

@Statful用于有状态SessionBean的标注

有状态会话Bean的标注：@Remove @PreDestory @Init

@Remove(retainIfExctption="true|false")

是否在Bean的实例抛出异常之后是否保留着个实例，@Remove标注能够应用的有状态SessionBean中的某个业务方法上。当调用到应用了这一注释的方法时，EJB容器会将这个Bean的实例销毁或是保留，

retainIfExctption属性置为true，则会在调用Bean的业务方法中出现异常则会保留这个Bean的实例，置为false则会销毁该实例。

EJB的依赖注入

@Resource标注能够引用到EJB组件环境中的资源。

@Resource(name="" ,type="")

注意在使用资源引用的时候，也需要在厂商的部署描述文件中对资源进行配置，也就是配置资源的类型和名字，以及JNDIname

@EJB(name="",beanInterface=""，beanName="",description="")

name属性指定引用EJB组件的名字

beanInterface属性用于指定被引用的EJB组件的接口类型

beanName属性如果在要引入的EJB组件在相同模块中也就是在同一个jar文件中，就需要自定beanName了

EJB拦截器（Intercepters）

EJB拦截器，使用来截取客户的调用方法的，就像是过滤器，他只适用于SessionBean MDB。

对目标对象方法进行调用之前，拦截器先执行调用。

@Interceptors和@AroundInvoke

使用@AroundInvoke标注注释方法，并且方法的返回值为Object

(1) 拦截器方法要用InvocationContext参数（Interface）

  InvocationContext的实现可以提供目标对象的相关信息。

(2) 拦截器方法需要用@AroundInvoke标注，表明为拦截器方法。

(3) 在拦截方法中用InvocationContext.proceed（）来执行目标对象的方法。

(4) 拦截器方法可以定义在单独的拦截器里，也可以定义在Bean类里。外部拦截方法优先级高。

  单独的拦截器用@ Intercepters来标注属于哪个Bean里。

(5) 假设目标对象有多个拦截器，按照先后顺序执行。

例

public class Test{

  @AroundInvoke

        public Object test(InvocationContext inv){

               System.out.println("Intercepted call via external class to:"+

                inv.getMethod().getName());//打印客户调用的方法的名字

                Object[] params=inv.getParameters();//获得客户调用的方法的参数表

                for(int i=0;i<param.length;i++){

                    System.out.println("\tparam: "+params[i]);//打印所有参数

                }

        }

}

三、依赖注入（DI，Dependency Injection）

  依赖注入，由容器将资源赋值给EJB。EJB的资源依赖容器来注入，因此称为依赖注入。

  @Resource（name=“jdbc/sample”）

    private……

  资源的引用

  该引用名需要在容器特定的配置文件里配置，并指向真正的资源在JNDI的名字。

  @Resource(……)标注可以用在属性，setter方法，类之前。

**Chapter08  Entity高级特性**

继承映射策略（Inheritance Mapping Stategies）

(1) 单表映射

(2) 关联关系映射

(3) 每个实体类一张表（不支持多态）

1、单表映射

优点：(1) 支持多态查询/多态更新

    (2) 简洁

    (3) 效率高（单表操作）

缺点：(1) 不适合深层次的继承结构，会形成庞大的表

    (2) 要求列允许为空

@Inheritance

@DiscriminatorColumn(name="DISC", discri:

minatorType=STRING,length=20)//指定区别父子类的表述符的列名、类型和长度

@DiscriminatorValue("CUSTOMER")//指定本类描述符的值

以上两个标注只适用于所有类建一个表的情况。

@Inheritance(strategy=JOINED|SINGLE\_TABLE|TABLE\_PER\_CLASS)

SINGLE\_TABLE也就是所有类建一张表

JOINED也就是每个类建一个表。

TABLE\_PER\_CLASS只针对对具体类建表。

@MappedSuperclass//映射子类型，这个标注没有属性

@AttributeOverride

@AttributeOverrides

2、关联关系映射

一对一关联

@OnetoOne(targetEntity="",cascade="CascadeType.ALL|MERGE|PERSIST|REFRESH|REMOVE",fetch="FetchType.EAGER|LAZY", mappedBy="",optional="true|false")

cascade级联设置，

ALL对主对象的增删该操作，都会级联到子对象

PERSIST只有对主对象进行增加操作时，才级联到子对象

REMOVE只有对主对象进行删除操作时，才级联到子对象

MERGE只有对主对象进行修改操作时，才级联到子对象

REFRESH只有对主对象进更新操作时，才级联到子对象

fetch加载策略，FetchType.EAGER是采取立即加载策略，FetchType.LAZY是采用延迟加载。一对一关系，在默认情况下，会采用Fetch操作立即加载。

一对多关联

一对多单向关系需要中间表。

双向一对多关系，也要注意（mappedBy=“”）关系维护方的问题。

mappedBy，是指定拥有关系的属性，只需要在关联的反方向(非主对象)一端指定使用mappedBy的属性。

@JoinColumn(name="",referencedColumnName="",unique="true|false",nullable="true|false",updateable="true|false",table="...")// 用来指定根据类关系映射到表后和其他表关联的列名，以及关联的列名，以及本列的属性

name属性指定类中属性对应的列名，默认为属性名

referencedColumnName属性指定类对应的表中的关联引用的列的列名。

nullable属性指定类中属性对应的列是否可空，默认为true

updateable="true|false"属性指定类中该属性是否会出现在update语句中，也就是会不会被修改，默认为true可以被修改。

table属性指定类中有关联属性的列所对应的表，默认为实体类所对应的表。

@PrimaryKeyJoinColumn主键关联

例：

    引用外键

    @OneToOne(optional=false)

    @JoinColumn(name="CUSTREC\_ID", unique=true, nullable=false, updatable=false)

    public CustomerRecord getCustomerRecord() { return customerRecord; }

    @OneToOne(optional=false, mappedBy="customerRecord")

    public Customer getCustomer() { return customer; }

    共享主键

    @Entity

    public class Employee {

        @Id

  @OneToOne @PrimaryKeyJoinColumn

        Integer id;

        EmployeeInfo info;

        ...

    }

    On EmployeeInfo class:

    @Entity

    public class EmployeeInfo {

        @Id

        Integer id;

        ...

    }

@OneToMany(targetEntity="",cascade="CascadeType.ALL|MERGE|PERSIST|REFRESH|REMOVE",fetch="FetchType.EAGER|LAZY", mappedBy="")

    @OneToMany(cascade=ALL, mappedBy="customer")

    public Set getOrders() { return orders; }

    In Order class:

    @ManyToOne

    @JoinColumn(name="CUST\_ID", nullable=false)

    public Customer getCustomer() { return customer; }

   @OneToMany(targetEntity=com.acme.Order.class, cascade=ALL,

            mappedBy="customer")

    public Set getOrders() { return orders; }

    In Order class:

    @ManyToOne

    @JoinColumn(name="CUST\_ID", nullable=false)

    public Customer getCustomer() { return customer; }

多对多关联

@ManyToMany(targetEntity="",cascade="CascadeType.ALL|MERGE|PERSIST|REFRESH|REMOVE",fetch="FetchType.EAGER|LAZY", mappedBy="",optional="true|false")

@JoinTable标注是标识多对多中间表的中对应的信息的。

对于双向多对多关系，mappedBy=“”可以放在任意一方，@mappedBy与@JoinTable不能在同一个实体。

@JoinTable所在的实体称为主控方。

    例:

    @JoinTable(

    name="CUST\_PHONE"//中间表表名,

    joinColumns=

        @JoinColumn(name="CUST\_ID", referencedColumnName="ID"),

       //指明本方id关联的列名及引用自哪一列

    inverseJoinColumns=

        @JoinColumn(name="PHONE\_ID", referencedColumnName="ID")

       //指明关联方id所对应的列名及引用自哪一列

    )

    关联的两个类Customer用户，和用户的电话号码PhoneNumber

    例 1:

    In Customer class:

    @ManyToMany

    @JoinTable(name="CUST\_PHONES")

    public Set getPhones() { return phones; }

    In PhoneNumber class:

    @ManyToMany(mappedBy="phones")

    public Set getCustomers() { return customers; }

    例 2:

    In Customer class:

    @ManyToMany(targetEntity=com.acme.PhoneNumber.class)

    public Set getPhones() { return phones; }

    In PhoneNumber class:

    @ManyToMany(targetEntity=com.acme.Customer.class, mappedBy="phones")

    public Set getCustomers() { return customers; }

    例 3:

    In Customer class:

    @ManyToMany

    @JoinTable(name="CUST\_PHONE",

        joinColumns=

            @JoinColumn(name="CUST\_ID", referencedColumnName="ID"),

        inverseJoinColumns=

            @JoinColumn(name="PHONE\_ID", referencedColumnName="ID")

        )

    public Set getPhones() { return phones; }

    In PhoneNumberClass:

    @ManyToMany(mappedBy="phones")

    public Set getCustomers() { return customers; }

在实体中，两种类型的属性需要进行特殊标注，也就是Date和Calendar

需要使用@Temporal(Date),@Temporal(Calendar)

EJB QL语句

EJB3.0中的QL，可以使用order by和having字句，对得出的结果进行分组处理。

EJB3.0中的QL关联查询和hibernate的HQL的写法基本相同。

Bulk updates or deletes：

优点：批处理操作直接在数据库进行，避免了内存的大量占用，提高了性能。

缺点：没有缓存操作。

Group by

  Group by中分组的字段必须在select中出现

例：

Query query = em.createQuery("select o from Order o left join o.orderItems where                               o.ower.age=26 order by o.orderid");

EJB QL也可以使用构造器查找。

把查询结果构造成一个对象，类必须使用全限定名。

例：

Query   query=em.createQuery("select    new com.foshanshop.ejb3.bean.SimplePerson(p.name,p.sex) from Person p order by p.personid desc");

**chapter09 事务**

一、事务的基本知识

1、什么是事务？

  通常，我们把一组连续不可分的数据库操作称为一个事务。

事务解决了两大问题，原子操作，数据共享

原子操作，是由不可分操作组成的，使这些不可分的操作，一同操作成功或者一同失败。

2、事务的ACID特性

  A（Atomicity）原子性：操作不可分割，一连串操作当作一个整体执行；

  C（Consistency）一致性：数据库的数据在事务完成前后保持一致；

  I（insulation）隔离性：多个事务独立地执行，一个事务的执行不影响另一个事务的执行结果，隔离性即一个事务对另一个事务的可见性；

  D（durability）持久性：事务操作的结果应该保留下来。

3、事务的类型

  (1) 扁平式事务：事务要么成功，要么失败，不允许包含子事务；

  (2) 嵌套式事务：一个大事务里潜逃多个小事务，允许子事务回滚。各个子事务必须都成功，整个事务才能成功提交；

  Tips：EJB不支持嵌套式事务。

4、分布式事务（Distributed Transaction）

  (1) 什么是分布式事务？

  跨越多个数据资源的事务叫做分布式事务。

数据资源：对数据进行存储、管理的空间（如Database，JMS中的主题队列，File System，后端系统等）。

JDBC只支持本地事务。

本地事务：只负责处理单个数据资源（Data Resource）的事务。

(2) 为什么要使用分布式事务？

企业级应用中，业务需要跨越多个数据资源，并且要作为一个整体来执行，就需要使用分布式事务。

(3) 分布式事务的原理

两阶段提交协议（Two-phrase commit protocol）

① 四个角色：A、Data Resource（数据存取的目的地，通常为Database）

       B、Resource Manager（负责数据资源管理，通常为Database device）

       C、Transaction Manager（负责对资源管理器施加事务的管理）

       D、Transaction Coordinator（事务协调器，负责对多个事务管理器进行协调，保证分布式事务能够执行）

②Two-phrase commit protocol执行过程（类似于投票机制中的一票否决制）

第一阶段：事务准备阶段

a、事务协调器（Transaction Coordinator）向各个事务管理器（Transaction Manager）发送事务准备提交的信息；

b、Transaction Manager向Transaction Coordinator回应是否提交事务；

c、Transaction Coordinator记录日志（各个Transaction Manager的回应）；

第二阶段：事务提交阶段

d、Transaction Coordinator向各个Transaction Manager发送事务提交信息；

e、Transaction Manager将执行结果返回给Transaction Coordinator。

二、EJB中的事务机制

  1、JTS（Java Transaction Service）

  JTS协议基于OTS协议，JTS协议即Java中分布式事务服务的协议。

  JTA（Java Transaction API）：

  ① UserTransaction（EJB或其他组件用该接口使用分布式事务）

  ② TrasactionManager：应用服务器与分布式事务服务器的接口

  ③ ResourceManager

EJB事务调用流程：EJB→UserTransaction→JTA→JTS→OTS

2、EJB的事务

  ① EJB事务的特点

  提供声明式事务与编程式事务

  声明式事务：应用程序只需要关心业务逻辑，由容器来负责事务的管理。

  编程时事务：应用程序编码人员自己写事务代码。

  ② EJB事务编程的类型

  A、CMT    容器管理事务

  B、BMT    Bean管理事务

  C、Client-MT    Client-Controlled Transaction客户端管理事务

  实体Bean只能用CMT。

  CMT：

  由容器实现的远程对象/拦截器，负责调用中间件服务。

  优点：在应用程序代码中，不用编写事务服务代码；

  缺点：粗粒度，只能在方法级别控制事务。

  EJB Bean类中编程方式来使用事务（BMT）：

  优点：细粒度地控制事务

  缺点：事务代理与业务代码纠缠

  Client-controlled Transaction：

  优点：客户端可以精确控制事务

  缺点：可能会因为网络问题引起是事务的回滚。

3、EJB事务边界的划分

  事务边界：事务边界是指事务从哪里开始。

  CMT的事务特性：

  Required：Bean类的方法必须要在事务环境下运行，这是容器默认的事务机制。

  事务特性只能使用在CMT。

  RequiredNew：Bean类中的方法必须在一个新的事务环境下运行。

  Supports：Bean类的方法不需要支持事务。如果客户端有事务，则继续沿用原事务环境。

  Mandatory：Bean类中方法必须要在事务环境下运行。客户端不启动事务则报错。

  NoSupported：Bean类中方法不支持事务。如果客户端启动了事务，则挂起该事务。

  Never：Bean类中的方法不支持事务。如果客户端启动了事务，则报错。

三、EJB事务的编程

  1、CMT

    @TransationManagement  用在类前，标注该EJB事务管理方式为Bean | Container（默认）

    @TrasactionAttribute    用在方法前，标注事务特性（事务的边界）

    @SessionContext.setRollbackOnly()  回滚标识，setRollbackOnly()方法必须在事务环境下运行。

    EJB容器对于非受查异常（主要指RuntimeException）会回滚，事务对于受查异常则会提交事务。

  2、BMT

  UserTransaction：

  ① interface

  ② 提供控制事务的方法

  ③ 由容器实现，可以使用@Resource注入

    UserTransaction.begin（）| commit（）| .rollback（）

  3、客户端控制事务

  调用EJB的方法，要求EJB必须采用CMT形式。

  4、事务的隔离性

  事务的隔离级别：

  A、Read uncommitted：性能最高

  B、Read committed：解决脏读问题

  C、Repeatable read：解决重复读取问题

  D、Serializable：解决幻读问题

  EJB本身不提供隔离级别的设置，可以通过直接设置数据库（连接池）的隔离级别。

  SessionSynchronization接口：

  在有状态会话Bean中，如果事务失败，可以恢复其状态。

afterbegin（）：在事务刚启动是，容器调用该方法，一般做状态初始值的保存。

beforeCompletion（）：在事务完成之前，容器回调。

afterCompletion（）：在事务提交之后调用，boolean值由容器提供。true表示事务提交成功，false则表示事务失败。在此方法中，做状态的恢复。

**chapter10 Security**

一、Security（安全）的概念

  1、Violations（侵害）：

  Violations，系统运行中，需要避免发生的事件（事件一旦发生，会侵害系统的安全）。

  2、Vulnerabilities（弱点）：

  Vulnerabilities，系统中存在的漏洞或缺陷（如默认数据库管理员密码忘记修改）

  3、Risk（风险）

  Risk，是指发生侵害型事件的概率与发生后的危害的乘积。

  4、Controls（控制）

  Controls，发生侵害事件、弱点、风险采取何种方式去避免侵害的发生。

  ① Authentication（认证/验证）

    系统要求访问这提供身份信息（通常为用户名和密码），系统采取某种方式去检验该身份信息（常利用Database），如IDAP，验证完后，将身份信息标识为一个主体/实体。

  ② Authorization（授权）

    授权是系统根据认证后形成的主体找到主体对应的角色。角色与系统资源是一一对应的。

  授权流程：

  （认证）：访问者→主体| |（授权）：→角色→资源

  系统依据角色赋予访问者相应的系统资源访问权限。

  ③ Data integrity protection（数据集成性保护）

  系统对资源写操作进行相应的控制。

  ④ Data confidentiality protection（数据机密性保护）

  系统对资源读操作进行相应的控制。

二、编程式与声明式安全

Web Application Security

1、Web中认证的方式

  ① 基本认证/摘要认证

  a、通过配置web.xml来配置一个基本认证的方案；

  b、系统提供一个默认的表单接受用户输入的认证信息；

  c、系统对用户输入的数据（用户名/密码等）不作输入加密操作，传送到后台处理；

  d、认证信息存放在HTTP请求头当中。

  基本认证常用在企业内部环境，少有恶意破坏的应用环境，性能较高。

  ② 基于表单的认证

  a、系统需要编程人员提供表单程序（表单定制）；

  b、系统将用户认证信息存放在HTTP请求体中；

  c、不加密传送到后台处理。

  ③ 基于HTTPS协议的认证

  a、系统采用HTTPS协议来传递信息（信息会加密）；

  b、系统采用公钥、私钥来对信息加密，采用证书来验证。

  这种认证很常用，认证的方式也很可靠。

2、Web当中的授权

  Declarative security（声明式授权）

  ① 只需要在web.xml中配置相关元素

  ② 需要在容器特定配置文件中作主体与角色的映射。

配置步骤：

  ① 在容器中配置主体（容器中的用户）以及角色和主体对应关系；

  ② 在web.xml中配置角色，资源，角色与资源的对应关系

  ③ 在sun-web.xml中配置主体与角色的对应关系。

关于声明式授权：

  缺点：① 粗粒度，只能控制到页面级；

      ② 只适用与简单的应用情况。

EJB Security

EJB中的认证与授权：

  ① Web容器对客户端进行认证

  ② Web容器将认证后的主体传递给EJB容器

  ③ EJB容器按照主体来控制EJB的调用

Web主要作认证，EJB主要作授权。

EJB认证

  JAAS（Java Authentication Authorization Service，Java授权认证服务）

  采用了策略模式

  作用：在具体的认证与授权方案之前，提供一个抽象层（JAAS），可以使具体的认证授权方案与应用层解耦。

EJB授权

1、EJB中的编程式授权

  ① 在EJB Bean类中编写安全的逻辑

  ② 在EJB Bean类中，申明安全角色

  ③ 映射安全角色与主体的关系

@DeclareRoles（’manager’）→申明安全角色（可以是多个）

  SessionContext.isCallerInRole（）→逻辑安全角色

isCallerRole（）方法可以用来判断调用这（客户端）的角色是否为安全角色

① ejb-jar.xml中配置逻辑安全角色与实际安全角色的对应关系；

② 配置实际安全角色；

    ③ 配置实际安全角色与主体（Principal）的对应关系（sun-ejb-jar.xml中）

       主体是容器中配置的用户。

  2、EJB中的声明式授权

    ① 声明安全角色

    ② 使用标注来声明方法权限

    例如，@RuleAllowed（“employee”），即哪些安全角色可以访问该方法

    ③ 在配置文件中，映射安全角色与主体的对应关系

    @Declare Roles 声明角色

    @DenyAll 标注任何角色都不能访问该方法

    @PermitAll 标注任何角色都可以调用该方法

    @RuleAllowed（）指定哪些角色可访问该方法

    @RunAs 将角色暂时转换为……调用

**chapter11 Time Service**

Time Service（定时服务）

EJB如何调用EJB容器通过的定时服务？

在指定的某个时间点，执行相应的人任务。

常用的定时服务框架：

  EJB（Since EJB2.1）

  QuantZ（Open Source）

  脚本（如UINX中的cron at）

1、EJB定时服务的原理

  采用了模板方法

  ① 容器提供定时服务功能

  ② EJB要注册到定时服务器上

  ③ 到达指定时刻，容器会回调EJB中的方法

2、EJB定时服务的编程

TimerService API

  javax.ejb.TimerService<intenface>

  ① TimerService生成并返回一个Timer对象

  ② 构造在指定的时间点执行的Timer 或

     构造演示一段事件后重复执行的Timer

  ③ 定义回调方法 TimeObject<interface> 或

    a、EJB实现该接口

    b、容器回调TimeObject也可以采用标注@Timeout来定义回调方法

    通过EJBContext来获得定时服务

  @Resource private TimerService ts；

  ts.createTimer（Date date，long interval，String meetingNote）创建定时服务逻辑并注册

  @Timeout 标注该方法为定时服务方法，导师EJB容器会回调这个方法。

EJB定时服务的优缺点：

  优点：与平台无关，底层定时服务与EJB无关

  缺点：粗粒度，不能在配置文件中配置时间

**chapter12 EJB补充知识**

一、将.war（Web模块）与.jar（EJB模块）打包成.ear（企业级模块）

  方式1、先定义企业级程序框架，然后再将已有的.war，.jar加进来，最后打包成.ear

  方式2、从企业级框架建立开始，新建.war，.jar，最后打包成.ear文件

二、在容器外（脱离容器），使用JPA

1、配置库文件：A、EntityManager 持久化提供者的库文件

         B、持久化的包

         C、数据库的包（数据库驱动程序）

         D、连接池的包

2、编程一个单独的Java程序

  使用到的API：

  (1) persistence

  (2) EntityManagerFactory

  (3) EntityManager

  (4) EntityTransaction

(5) Entity  Query

hibernate-entitymanager\_3.2.1GA.zip  用于支持JPA的包

hibernate-annotation\_3.2.1GA.zip    用于标注的包

hibernate\_3.2.1.zip            Hibernate核心包

顶

0

踩