第一章、 简介

1. JDBC 需要很多连接，EJB 是连接池
2. 客户端不直接与 EJB 对话，而是与容器生成的骨架对话(RMI 或 CORBA)
3. Session Beans 的三种类型 Stateful，Stateless 和 Singleton，可能实现 SessionBean 接口
4. 不受管的 Entity Bean 被称为 Value Objects 或 Data Transfer Objects

第二章、 EJB 的服务

1. 单个 Session Bean 不支持同时处理多个连接，即不支持并发，MDB 也是
2. Entity Bean 支持并发，由事务 Transaction 负责保证完整性
3. 几个不同的标注：@Stateful，@Stateless，@MessageDriven，@Local，@Remote，@LocalBean，@RemoteBean
4. 老式的用@EJBObject 和@EJBLocalObject，@EJBHome 和@EJBLOcalHome
5. 打包到 Jar 文件里
6. 查找接口三步走
   a. InitialContext()
   b. context.Lookup()
   c. PortableRemoteObject.narrow(实际类名.class)

第三章、 Session Bean

1. 无状态 Bean 每个服务都是一个 method，每个 method 互相没什么关系
2. 有状态 Bean 可以在 Method 之间共享客户端信息
3. 单例 SessionBean 用一个 Bean 处理所有请求
4. EJB 的配置放在 META-INF/ejb-jar.xml 文件里
5. @Resource SessionContext ctx
6. EJBContext 可以获取时间服务，和登录者名字 callerPrincipal 和 isCallerInRole
7. 无状态 SessionBean 就两个状态: 不存在和 Method-Ready Pool
8. 两个生命周期回调: @PostConstruct 和@PreDestroy
9. ENC 全称: Enterprise Naming Context，用@Resource 和@EJB 获取
10. 有状态 Bean 用 EJBObject 做 Proxy，Method 之间可以不独立，前序影响后序
11. 有状态 Bean 可以存储状态，给客户端瘦身，但服务端重启会丢失状态
12. 使用 Session Bean:应用服务器@EJB 来注入，容器外用 JNDI 查找，CDI 用@Inject 注入
13. 有状态 Bean 没有池，它们会被钝化(Passivated)直到被再次激活(Activated)，或者删除
14. 有状态 Bean 多了两个生命周期回调：@PrePassivate 和@PostActivate
15. 客户端调用了@Remove 标记的函数也能删除有状态 Bean
16. 嵌套的有状态 bean 当外层有状态 bean 被删除时，内层也被删除

第四章、 单例子 Bean

1. 处理任意多个并发链接，所以必须是线程安全的
2. 必须使用线程安全的数据结构，比如 java.util.concurrent.atomic 下的 AtomicInteger
3. 可以用锁和超时
   @javax.ejb.Lock(locktype) type 可以是 READ 或 WRITE
   @javax.ejb.AccessTimeout()
   如果要自己管理同步，用@javax.ejb.ConcurrencyManagement(BEAN)
4. 多了一个生命周期函数@javax.ejb.Startup，在部署时运行

第五章、 Entity Manager

1. 受管 Bean 被成为 attached，自动与数据库同步，也可以调用 flush
2. 不受管的称为 detached，不被 entity manager 管理，与数据库无关
3. 两种 Persistent Context
   Scoped：在函数内有效，被事务管理 type=TRANSACTION，只能用于 SLSB
   Extended：可游离在事务之外，被有状态 Bean 使用：
   @PersistenceContext(type=EXTENDED)
   EntityManager em
4. 解除附加：em.detach(bean)，之后 bean 就是普通的 POJO
5. Persistence Unit 在 META-INF/persistence.xml 定义，name 属性是必须的
6. 注入用@PersistenceContext(unitName="titan") private EntityManager entityManager
7. EntityManagerFactory 的 createEntityManager()可在非 Java EE 的客户端使用
8. 增删改查函数：persist, remove, merge, find(返回 null)/getReference(抛异常)
9. 示例 Query
   Query query = entityManager.createQuery("from Employee e where id=2");
   Employee emp = (Employee)query.getSingleResult();
10. clear()会剥离所有 Entity Bean，使其变成 POJO

第六章、 Entity Bean

1. @Table 和@Column 默认映射类名和变量名
2. 表自动生成主键用@GeneratedValue
3. @Id 标注主键，可以是一个字段也可以是多个字段
4. 组合主键的类必须实现 hashCode()和 equals()
5. 组合主键可用@IdClass(放 Entity Bean 上，Entity Bean 内部多个@Id)
   也可@EmbeddedId(主键类字段直接在 Entity Bean 里面，字段就是类)配合@Embeddable
6. 用组合主键时，要用@AttributeOverrides 和@AttributeOverride 指明多个列名和变量映射
7. 其他标注
   - @Transient 标记不持久化的成员
   - @Basic and FetchType 是否延时加载，但没人用，因为不可靠
   - @Temporal 时间(DATE, TIME, or TIMESTAMP) 如@Temporal(TemporalType.DATE)
   - @Lob 映射二进制数据
   - @Enumerated 映射可枚举的类型
8. 如果类的成员分散在多个数据表，可以用@SecondaryTable，如

```
@Entity
@Table(name="CUSTOMER")
@SecondaryTable(name="CUST_DETAIL",
   pkJoinColumns={
      @PrimaryKeyJoinColumn(name="CUST_ID"),
      @PrimaryKeyJoinColumn(name="CUST_TYPE")})
public class Customer { ... }
```

第七章、 Entity 的继承

1. 用@Inheritance 来标记：strategy = InheritanceType.XXXXX

2. 三种继承关系
   SINGLE_TABLE：一张表中包括类和子类中全部成员的属性
   TABLE_PER_CLASS：每个层次一个表，表中有当前类全部成员的属性
   JOINED：每个子类一张表，只有自己的属性，没有父类的属性，最优解
3. @DiscriminatorColumn 和@DiscriminatorValue 表示这一个数据属于哪个类

第八章、 ORM

1. 一对一
   a. 单向(JoinColumn 指向下级的主键列名)

```
@OneToOne(cascade={CascadeType.ALL})
@JoinColumn(name="ADDRESS_ID")
public Address getAddress( ) {
   return homeAddress;
}
```

   b. 双向(mappedBy 指向控制端中自己所在的变量名)

```
@OneToOne(mappedBy="creditCard")
public Customer getCustomer( ) {
   return this.customer;
}
```

2. 单向一对多(控制端加 OneToMany 和 JoinColumn)

```
@OneToMany(cascade={CascadeType.ALL})
@JoinColumn(name="CUSTOMER_ID")
public Collection<Phone> getPhoneNumbers( ) {
   return phoneNumbers;
}
```

3. 单向多对一 (多的那一方发出，一的哪一方不知道被关联了)

```
@ManyToOne
@JoinColumn(name="SHIP_ID")
public Ship getShip( ) { return ship; }
```

4. 双向的一对多和多对一是一样的，**ManyToOne 的那一方必须是支配方**
5. 双向多对多

```
@ManyToMany
@JoinTable(name="RESERVATION_CUSTOMER",
        joinColumns={@JoinColumn(name="RESERVATION_ID")},
        inverseJoinColumns={@JoinColumn(name="CUSTOMER_ID")})
public Set<Customer> getCustomers( ) { return customers; }
```

```
@ManyToMany(mappedBy="customers")
public Collection<Reservation> getReservations( ) {
   return reservations;
}
```

6. 单向多对多

```
@ManyToMany
@JoinTable(name="CABIN_RESERVATION",
        joinColumns={@JoinColumn(name="RESERVATION_ID")},
        inverseJoinColumns={@JoinColumn(name="CABIN_ID")})
public Set<Cabin> getCabins( ) { return cabins; }
```

7. cascade 值：ALL, PERSIST, MERGE, REMOVE, REFRESH

第九章、 Entity Bean 生命周期回调

1. 各种回调
   @PrePersist
   @PostPersist
   @PostLoad
   @PreUpdate
   @PostUpdate
   @PreRemove
   @PostRemove
2. @EntityListeners 可以为 Entity 指定一个到多个监听类，例如
   @EntityListeners ({Auditor.class})
3. 如果要排除 xml 配置的默认的监听器，用@ExcludeDefaultListeners
4. 如果要排除父类的监听器，用@ExcludeSuperClassListeners

第十章、 验证与授权

1. 定义：Authentication 验证，Authorization 授权
2. 在 standalone.xml 的<security-realm name=...>定义验证，支持多种类型，包括数据库
3. 支持 JNDI 的验证，如下
   properties.put(Context.SECURITY_PRINCIPAL, "username");
   properties.put(Context.SECURITY_CREDENTIALS, "password");
   Context ctx = new InitialContext(properties);
   object ref ctx.lookup("SecureBean/remote");
   SecureRemoteBusiness remote = (SecureRemoteBusiness) ref;
4. 对方法的授权可以用@RolesAllowed，如果允许所有可以用@PermitAll
5. 声明角色用@DeclareRoles({Roles.ADMIN, Roles.STUDENT, Roles.JANITOR})
6. @RunAs 可以让方法运行于某角色之下
7. @Resource SessionContect ctx 下面的 getCallerPrincipal()可以知道当前用户，还能用 isCallerInRole 得知角色是否在某个 Role 内

第十一章、CDI 注入

1. 写一个 Qualifier
   @Qualifier
   @Target({TYPE, METHOD, PARAMETER, FIELD})
   @Retention(RUNTIME)
   public @interface Secure{}
2. 默认@Default，要任意一个就@Any
3. @New 注入的必须是受管 bean 或 session bean

第十二章、Message Driven Bean

1. MDB 是异步的监听者，客户端给 Broker 发完消息就走，服务端在新线程处理消息
2. JMS 就是 Java Message Service
3. JMS 与 JCA 兼容，JCA 是 Java Connector Architecture
4. JMS 像 JDBC 一样，由厂商无关的 API 组成，消息队列系统 MQ 由第三方提供
5. 使用 JMS 叫做 JMS 客户端，JMS 系统叫做 JMS 提供者

6. 发送消息的叫生产者，处理消息的叫消费者
7. EJB 可以是生产者，Java 程序或 MDB 是消费者
8. 比 RMI 远程调用函数更好，因为它是平台无关的
9. 发送消息需要连接到 Provider 以及消息的目标地址
10. JMS 客户端可以订阅主题(一对多的发布和订阅)或 Queue(点对点)
11. 无法把事务或安全的上下文传播出去
12. 可以在事务里使用 JMS
13. 消息主题无所谓谁来处理消息，客户端断开后重连还能拿到之前丢掉的消息
14. Queue 的消息只能被一个消费者处理，发消息的与特定的接受者联系
15. MDB 是服务端的、无状态的、对交易敏感的、用于处理 JMS 消息的
16. Java 容易管理事务、安全、资源、并发、消息确认，还可以创建多实例处理更多消息
17. MDB 用@MessageDriven 来标注，可以有 activationConfig 参数，参数是单个到多个 @ActivationConfigProperty 标注

```
@MessageDriven(activationConfig={
  @ActivationConfigProperty(
     propertyName="destinationType",
     propertyValue="javax.jms.Queue"),
  @ActivationConfigProperty(
     propertyName="messageSelector",
     propertyValue="MessageFormat='Version 3.4'"),
  @ActivationConfigProperty(
     propertyName="destination",
     propertyValue="queue/reservationQueue")})
```

18. 当 PropertyName 为 messageSelector，是过滤器，只接收满足条件的消息
19. 当 PropertyName 为 acknowledgeMode，只能取 Auto-acknowledge 或 Dups-ok-acknowledge
20. 当 PropertyName 为 subscriptionDurablilty,只能取 Durable 重连保留/NonDurable 重连丢失，但对 Queue 无所谓
21. 当 PropertyName 为 destinationType,只能取 javax.jms.Topic 或 javax.jms.Queue
22. MessageDrivenContext 可以用@Resource MessageDrivenContext context;注入，有三个函数：getUserTransaction, getRollbackOnly, setRollbackOnly
23. MDB 有自己的 ENC，可以访问资源、环境和 EJB
24. MDB 必须实现接口 MessageListener 的 onMessage(Message message)函数
25. 发送消息的例子

```
Connection connect = connectionFactory.createConnection();
Session session = connect.createSession(true,0);
MessageProducer sender = session.createProducer(queue);
ObjectMessage message = session.createObjectMessage();
message.setObject(ticket);
sender.send(message);
connect.close();
```

26. 支持@PostConstruct 和@PreDestroy
27. 任何支持 JCA 的消息系统都可以用
28. 支持的类型：TextMessage, MapMessage,ObjectMessage, StreamMessage,BytesMessage
29. MOM：Message Oriented Midware

第十三章、 Enterprise Naming Context (ENC)

1. 用@EJB 或@Resource 注入
2. 用 JNDI 注册表实现
3. 可以引用例如 EJB、JMS Provider、数据源等资源
4. 全局 JNDI 支持 java:global[/app-name]/module-name/bean-name [!FQN]以及
   java:app/module-name/bean-name [!FQN]和
   java:module/bean-name [!FQN]
5. 用 javax.naming.InitialContext 初始化 Context 后 Lookup，一般用 jndi.properties 定义环境
6. JNDI Scope 支持 java:comp/(组件内)和 java:/(虚拟机内)以及其他名字(远程)
7. 向 ENC 放东西有几种方式
   a. 标注
      @EJB(name="ejb/ProcessPayment", //这里如果没指定，要用 FQN
      beanInterface=ProcessPaymentLocal.class,
      beanName="ProcessPaymentBean")
   b. @Resource 注入
      @Resource SessionContext sessionContext
   c. Setter 注入
      @EJB
      public void setProcessPayment(ProcessPaymentLocal payment) {…}
8. EntityManagerFactory 用@PersistenceUnit 注入
9. EntityManager 用 @ PersistenceContext 注入， PersistenceContextType 默认是 TRANSACTION，可以是 EXTENDED
10. @Resource 可以注入 Datasource、JMS、Session、URL 等
11. Resource 环境和管理对象，用 @Resource 标注，可以引用 UserTransaction,TransactionSynchronizationRegistry，不知道干什么的
12. @Resource 缺少定义 Topic 和 Queue 的全部属性
13. @ WebServiceRef 引用 WebService

第十四章、事务

1. ACID 分别是什么
   Atomic，Consist, Isolated, Durable
2. JTS：Java Transaction Service
3. @TransactionAttribute 标记 TransactionAttributeType
   a. NotSupported: 不论调用方有没有事务，我都没有
   b. Supports:你有我就有，你没有我也没有
   c. Required: 你有我跟你一样，你没有我自己搞一个
   d. RequiresNew: 不管你有没有，我自己都搞一个
   e. Mandatory: 你不支持，我就崩溃
   f. Never: 你支持，我就崩溃
   g. MDB 只能用 NotSupported or Required
4. 容器管理的事务，在函数被调用开始，在函数运行时结束，不允许嵌套不同事务
5. Context Propagation：难记，略
6. 脏读：A 读到了 B 没提交的数据，结果 B 他妈的回滚了
7. 重复读：A 刚刚读的数据，被 B 改了，A 再读，不一样了

8. 幻读：A 刚刚读还没这个数据，B 插入进来了，A 再读就有了，A 感到了幻觉
9. 数据锁：
   读锁：当前事务没完成，其他修改数据的事务不许进来，可阻止重复读
   写锁：当前事务没完成，其他修改数据的事务不许进来，允许脏读
   独占锁：当前事务没完成，其他读写数据的事务不许进来
10. 事务隔离等级：去百度，不好记。等级越高，性能越低。Serializable 最高
11. ApplicationException 不能回滚事务（除非指定 rollback=true），扔给客户端处理
12. RuntimeException 和 EJBException 可以回滚事务
13. Conversation State： Stateless bean 没有，Stateful Bean 有
14. SessionSynchronizatrion 接口监听事务：afterBegin(), beforeCompletion(),afterCompletion(boolean committed)