EJB系统开发实战录(4)

作者：庄永璋，李维

在前三期的文章中，我们已经大致完成了我们有关数据层组件 (Entity Bean，Value Object 等)的开发。 同时也说明了如何使用 JBuilder 开发 EJB 2.x 的 CMP 组件，以及建立所谓的 EJB Relationships，并且使用 JBuilder 的 EJB Test Client 精灵来快速开发一个测试客户端应用程序呼叫范例 EJB 组件以验证我们开发的 EJB 组件是否能够正确的工作。到目前为止，我们大致完成了我们的在线 Seminar 注册系统有关 CMP Entity Beans 的部份。

在本期的文章中我们将继续加入首期文中提到的 Session Bean 及另一 J2EE 设计模式 Session Façade Design Pattern，我们将会大致介绍有关 Session Bean 的开发及注意事项，同时如何运用 Session Façade Design Pattern，使我们的范例系统能在效能及维护性上达到最佳化的地步。当然，在本文的最后我们仍然会使用 EJB Test Client 精灵来确定实作的组件是能够正确工作的。

现在就让我们继续开发我们的 Seminar 注册范例系统，让我们就从前期文章中尚未讨论的 Session Bean 开始着手，同时在本期文章中，我们也会介绍有关 Session Façade Design Pattern 的部份。

Session Bean

在 EJB 中，我们用 Session Bean(注 1)来代表工作流程的概念，你可以把 Session Bean 想象成客户端在服务器上的一个代理人，它可替客户完成所要求的工作如取得 Entity Beans 的数据或整合各个 Beans 作数学运算等。事实上 Session Bean 是实作 MVC 模型中 Controller 最好的内定目标。Session Bean 之所以称为 Session Bean，是因一般而言它的生存期是跟随着客户的 Session 而定。

对于程序开发人员来说 Session Bean 是继承自 javax.ejb.SessionBean 这个 package。而 Session Bean 又可细分为 Stateless Session Bean 及 Stateful Session Bean 两种 Bean，顾名思义即一无状态而一为有状态，这里所指的状态是指客户端与 Session Bean 之间的对话状态，或许有的读者会有疑问，对于此一对话状态的维持方式，可能有读者要问到：是不是在 Stateless Session Bean 的宣告中，我就不能够宣告任何数据成员(Data Member)了呢？事实上，你仍就可以宣告数据成员在 Stateless Session Bean 的定义中，但要记得此数据成员，不可作为记录客户端与 Session Bean 的对话状态之用，同时在 Stateful Session Bean 中，你也还是可以不宣告任何数据成员。由于此一对话状态是由 EJB Container 来维护的，而 EJB Container 会根据 Session Bean 的型态，而有不同的实例管理方式。接下来，让我们了解一下这两类 Session Bean 在 EJB Container 的状态情形：

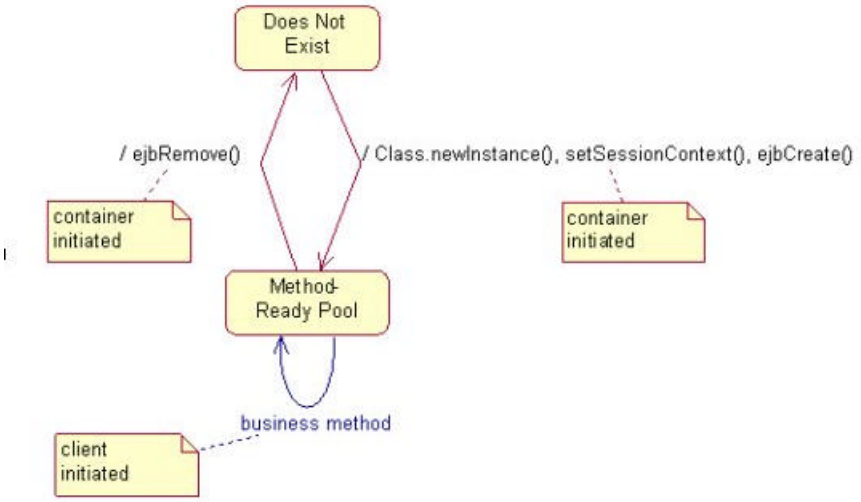

图 1，Stateless Session Bean 的状态图(State Diagram)

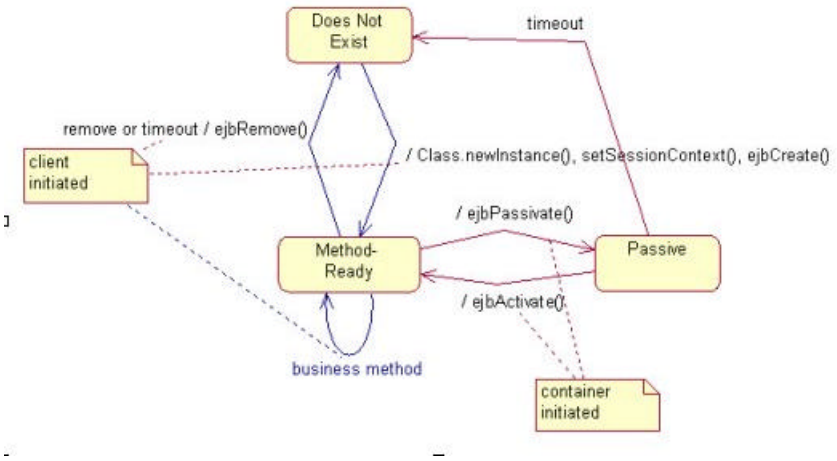

图 2，Stateful Session Bean 的状态图(State Diagram)

一般而言，Stateless Session Bean 在 EJB Container 中，会运用 Instance Swapping 的技术，让实例在内存内的运用能够达到最佳化，由于没有状态需要维护，EJB Container 可以迅速的将实例转换给新的客户使用。而 Stateful Session Bean 则因有对话状态的关系，当客户端长期未使用实例时，EJB Container 可以选择将实例转换给其它客户组件使用，惟 EJB Container 必须先将 Bean 的状态内容加以储存后，才能转换实例，这样的一个管理动作，对于 J2EE Server 的效能是有其影响面的。事实上，各家 J2EE Application Server 的厂商各有其作法，而这正也是我们用来评估各厂商产品的一个依据。

而我们要如何让 EJB Container 能够区分是何种 Session Bean 呢？事实上是透过 Deployment Descriptor 中的宣告，让我们的 EJB Container 了解应如何管理你所部署上来的 Session Bean，如下面这个 Deployment Descriptor 例子：

```
<session>
          <display-name>Scheduler</display-name>
          <ejb-name>Scheduler</ejb-name>
          <home>praticalejbprogramming.SchedulerHome</home>
          <remote>praticalejbprogramming.Scheduler</remote>
          <ejb-class>praticalejbprogramming.SchedulerBean</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
</session>
```

经由这样一个宣告，我们让 EJB Container 了解如何管理 Session Bean 的相关资源，如其生命周期及应如何管理 Session Bean 实例(Instance)等。

说了这么多，希望能够帮读者了解 Session Bean 的基本观念，那么在现实的世界中有那些例子可套用在 Session Bean 的开发呢？对于 Stateless SB 而言，如信用卡号的验证组件。对 Stateful SB 而言，如在线购物车（Shopping cart）组件等。接下来，让我们来了解一下一个常跟 Session Bean 相提并论的 Design Pattern-Session Façade Design Pattern。

Session Facade Design Pattern

一般提到 Session Bean，很难不提到 Session Façade Design Pattern。由于在 J2EE 架构中，通常在 EJB Container 中会有许多的企业对象(Business Objects)，如 Value Object，Entity Bean，甚至一般的 Java Class，而这些企业对象若直接与我们的客户端沟通的话，可能会产生下列几种情形：第一种情形是客户端的开发会与我们的企业对象过于紧密结合，造成日后系统维护及修改的不易；第二种情形是假如客户端直接存取企业对象的话，往往会产生太多网络

远程呼叫，造成系统效能的负担；第三种情形是透过此类作法，事实上并无一个一致性的对象存取策略，可能会造成客户端存取错误企业对象。

如何解决上述的情形呢？答案正是透过我们现在所要介绍的 Session Façade Design Pattern。何谓 Session Façade？Session Façade 即是我们利用 EJB 的 Session Bean，设计 Session Bean 为一个客户层与企业对象层的中介层，如此我们可以透过此一中介层来管理我们的企业对象，同时籍此提供一个一致性的窗口供客户层存取企业对象之用。读者应该还记得上期文章中，所介绍到的 Value Object，或许有的人会想到这两者有何差别呢？Value Object 主要负责的是将 Entity Bean 的数据成员加以封包为组件，以便我们在网络上传输；而 Session Bean 则是提供一个统一的窗口，以便客户端存取。所以事实上这两者所扮演的角色并无任何冲突，相反的，二者是一个相辅相成的情况。

运用 Façade Design Pattern 的设计，他的客户端可以是我们的 JSP 组件，Servlet 组件，甚至是一般 Java Application 都可以透过 Session Bean 来跟后端的企业对象作沟通。

实作 Session Bean 组件

在大致介绍过了 Session Façade 的观念，让我们开始实作我们范例系统所要设计的两个 Session Bean。首先让我们来看一下我们的系统 Class Diagram：

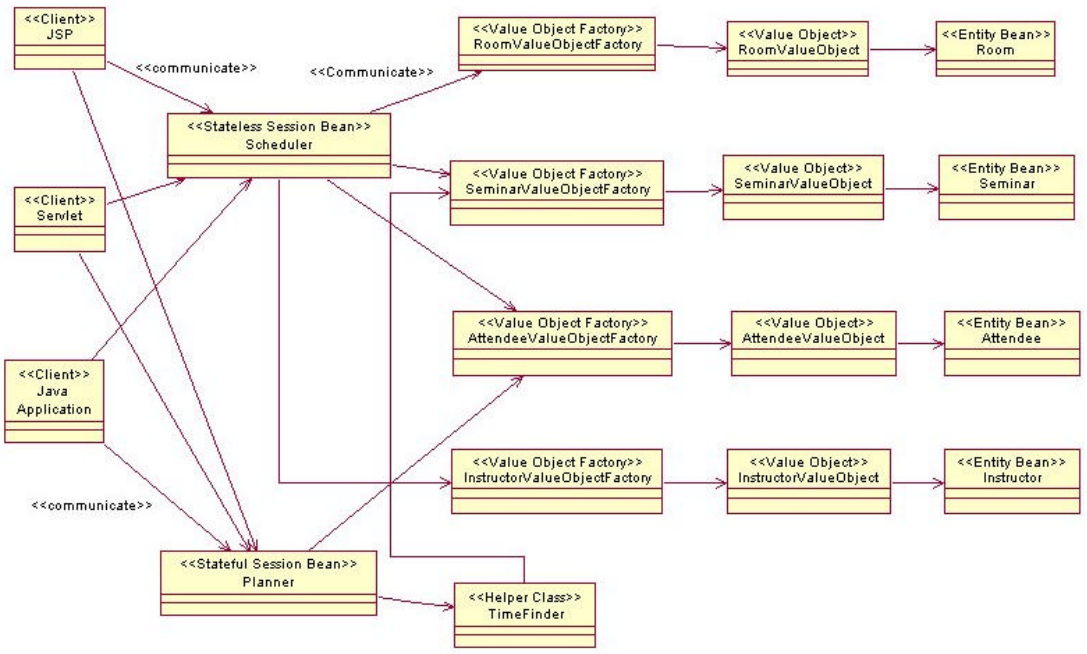

图 3，范例系统中运用 Session Façade 所设计的 Class Diagram

对于图中的两个 Session Bean，我们都采取了 Session Façade Design Pattern 来封装后端的企业对象，所以客户端都是透过这两个 Bean 和我们系统的企业对象作沟通。首先介绍的 Scheduler Session Bean 是一个 Stateless Session Bean，他的功能主要如下，如查询，新增，删除，修改学员名单及查询，新增，删除，修改教室或查询，新增，删除，修改课程等。

我们范例系统中的另一个 Session Bean 为 Planner Stateful Session Bean。他的主要功能是找到合适的时段及教室以方便 Seminar 的排定，之所以要用 Stateful SB 的方式来实作，是因我们希望每次呼叫 Planner SB 时，它能记得上次呼叫所传回的值，而不会传回重复的时段。通常我们会将这样的对话状态记录在 Stateful SB 的数据成员中。

实作 Scheduler Session Bean 组件

在 JBuilder 中，要实作 Session Bean 可透过点选 EJB Designer 中的 Session Bean 选项，如下图，

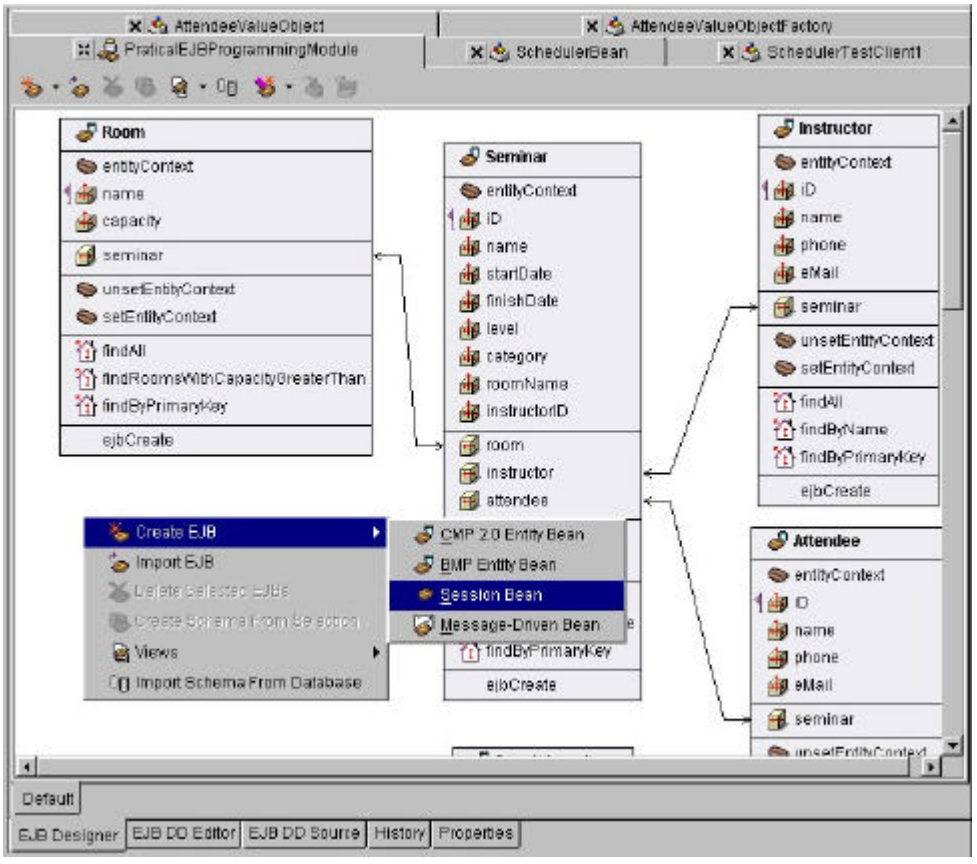

图 4，透过 JBuilder EJB Designer，产生 Session Bean 画面

在点选 Session Bean 之后，我们可以发现 Bean Properties 的窗口出现在 JBuilder EJB 设计家中，我们可以籍由此窗口，设定 Bean 的名称，及其

Session Type 等项目。透过 Session Type 的选项，我们可以选择所要开发的 Session Bean 为 Stateful 或是 Stateless。

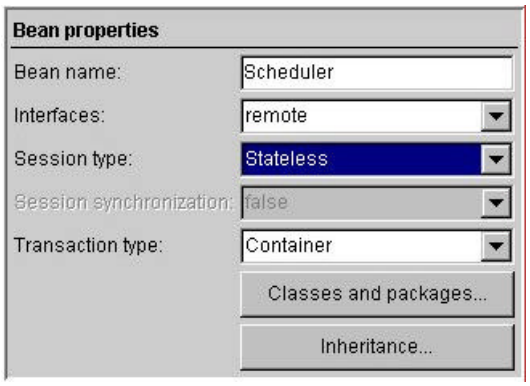

图 5，JBuilder 中 EJB 设计家 Session Bean Properties 之显示窗口

在产生了初步的 Bean 框架后，接下来我们必须提供 Scheduler Bean 的实作内容，首先，我们在 Scheduler Bean 内加入八个 private 的数据成员，并在 ejbCreate()中，将这八个资料成员的值初始化。

```
private RoomManager roomManager;
private SeminarManager seminarManager;
private AttendeeManager attendeeManager;
private InstructorManager instructorManager;

private RoomValueObjectFactory roomValueFactory;
private SeminarValueObjectFactory seminarValueFactory;
private AttendeeValueObjectFactory attendeeValueFactory;
private InstructorValueObjectFactory instructorValueFactory;

public void ejbCreate() throws CreateException {
  try {
    roomManager = RoomManager.getInstance();
    seminarManager = SeminarManager.getInstance();
    attendeeManager = AttendeeManager.getInstance();
    instructorManager = InstructorManager.getInstance();

    roomValueFactory = RoomValueObjectFactory.getInstance();
    seminarValueFactory = SeminarValueObjectFactory.getInstance();
    attendeeValueFactory = AttendeeValueObjectFactory.getInstance();
    instructorValueFactory = InstructorValueObjectFactory.getInstance();
  }
```

```
    catch(Exception e) {
      throw new EJBException(e);
    }
  }
```

由于 Scheduler Session Bean **的功能主要如下，如查询，新增，删除，修改学员，授课人员名单及查询，新增，删除，修改教室或查询，新增，删除，修改课程等。所以我们必须将这部份的** business method **也加入到** Scheduler Bean **的实作内容中。读者可以在** JBuilder EJB **设计家中，按右键点选** Scheduler Bean**，透过** Add **选项，选择加入新的字段或新的** method**。以下列出的是** Scheduler Bean **的** Remote Interface **中** business methods **宣告的部份，至于实作的部份，由于篇幅有限，我们会放在网站上供读者下载参考：**

```
  public interface Scheduler extends javax.ejb.EJBObject {
  public String createRoom(RoomValueObject roomValue) throws RemoteException;
  public RoomValueObject findRoom(String name) throws RemoteException;
  public Collection findAllRooms() throws RemoteException;
  public void updateRoom(RoomValueObject roomValue) throws RemoteException;
  public void removeRoom(String name) throws RemoteException;
  public void removeAllRooms() throws RemoteException;
  public long createSeminar(SeminarValueObject seminarValue) throws RemoteException;
  public SeminarValueObject findSeminar(long id) throws RemoteException;
  public Collection findSeminarsOfAttendee(long id) throws RemoteException;
  public Collection findAllSeminars() throws RemoteException;
  public void updateSeminar(SeminarValueObject seminarValue) throws RemoteException;
  public void removeSeminar(Long id) throws RemoteException;
  public void removeAllSeminars() throws RemoteException;
  public long createAttendee(AttendeeValueObject attendeeValue) throws RemoteException;
  public AttendeeValueObject findAttendee(long id) throws RemoteException;
  public AttendeeValueObject findAttendeesByName(String name) throws RemoteException;
  public Collection findAllAttendees() throws RemoteException;
  public void updateAttendee(AttendeeValueObject attendeeValue) throws RemoteException;
  public void removeAttendee(long id) throws RemoteException;
  public void removeAllAttendees() throws RemoteException;
  public InstructorValueObject findInstructor(String name) throws RemoteException;
  public Collection findAllInstructors() throws RemoteException;
  public void updateInstructor(InstructorValueObject instructorValue) throws
RemoteException;
  public void removeInstructor(String name) throws RemoteException;
  public void removeAllInstructors() throws RemoteException;
```

```
  public long createInstructor(InstructorValueObject instructorValue) throws
RemoteException;
}
```

Session Bean EJB Local References

接下来，我们要介绍一个在 BES **比较特别的设定项目，大家都知道从** EJB 2.0 **之后，增加了对** Local Interface **的支持，目的就是为了避免在同一** EJB Container **内的** EJB **组件，还须透过** JNDI **这样一个远程呼叫去找到另一** EJB **组件。事实上，**Borland Enterprise Server **在还是** Borland AppServer 4.5 **版的时候，已经看到了这样一个情形，所以当时即提出了**EJB Local References **这样一个解决方案，目的即是达到和** Local Interface **相同的一个功效。所以在我们开发完了我们的** Scheduler Session Bean **之后，除了有关** Transaction **的设定以外，我们还可以透过**EJB Local References **的设定，来方便**Scheduler Bean **找到它所需要用到的其它** EJB **组件。**

如何设定呢？首先请将鼠标指到 JBuilder **左上角的** Project Pane **内的** PracticalEJBProgramming module**，将其展开后，接着双点击** Scheduler Bean**。你会看到在** JBuilder **右边的** Content Pane **中会出现一个可视化的接口，如下图：**

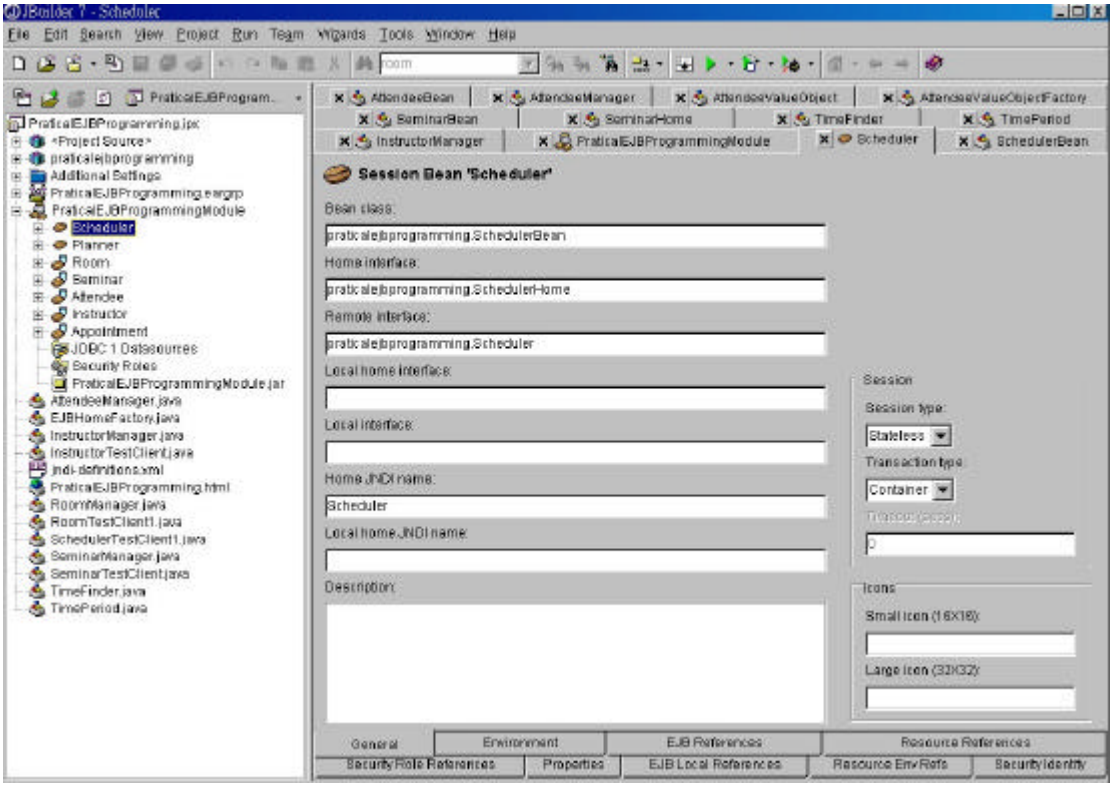

图 6**，**JBuilder Enterprise Java Bean Properties **之显示窗口**

接着，请看到右下方的各个不同的页次(Tab)，请切换到 EJB Local References 页面。读者可透过 Add 按键来加入所要关联的 EJB 组件，这个组件可以是 Entity Bean，也可以是 Session Bean。记得要将 Link 选项，设定为你所要关联的 EJB 组件，同时将 IsLink 的选项勾选起来。下图即是我们 Scheduler Bean 的 EJB Local References 设定情形：

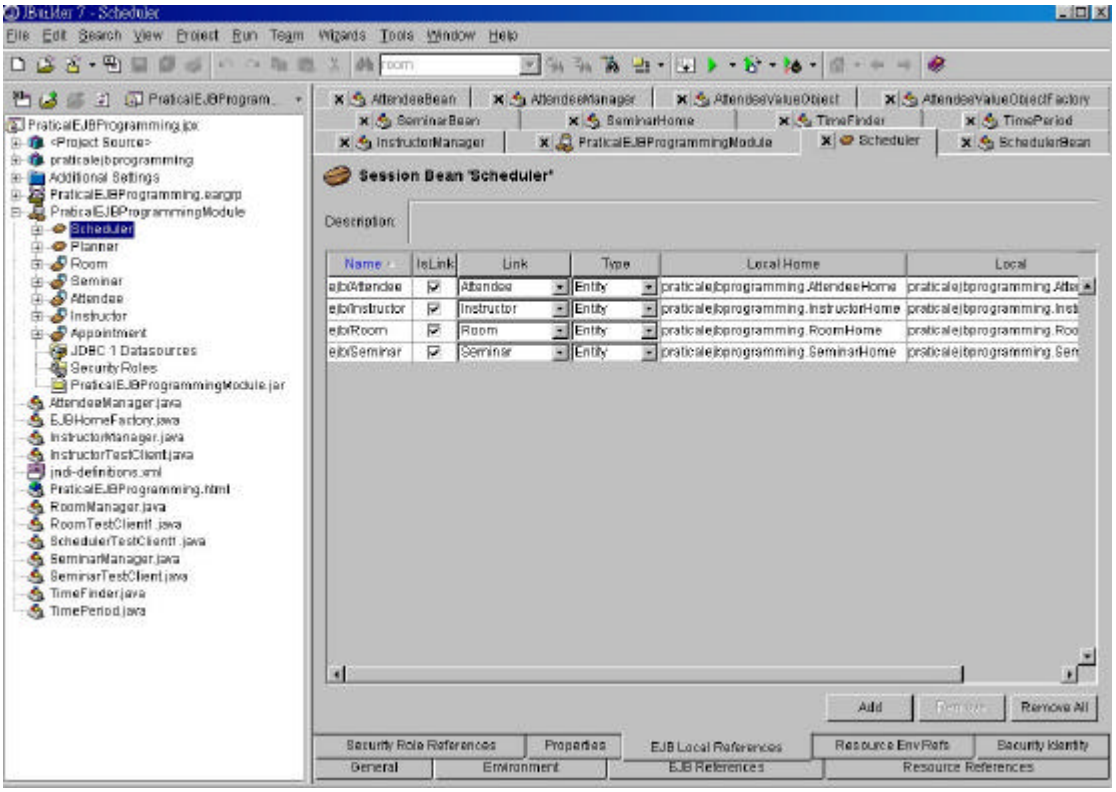

图 7，Scheduler Session Bean EJB Local References Properties 之显示窗口

透过 EJB Local References 的设定，让我们在同一 Container 中的 EJB 组件，可以经由 local reference 直接去找到其它位于本机的 EJB 组件，对于系统效率的提升有直接的帮助。所以请读者注意，如果你的 EJB 组件是布署在同一 EJB Container 的话，要记得为你开发的 EJB 组件加上 EJB Local References 的相关设定。

实作 Planner Session Bean 组件

在设计完了我们的 Scheduler Stateless Session Bean 之后，紧接着我们要实作我们的 Planner Stateful Session Bean。首先我们要介绍两个 Planner Bean 要用到的 Java 辅助类别。一为 TimeFinder class，另一为 TimePeriod class；这两个 Java 类别都是用来帮助 Planner Bean 寻找一个合适的时段，以方便 Seminar 的排程。我们将与会人员名单传入 TimeFinder 类别，而后由 TimeFinder 类别去找到适合的时段来排程。TimePeriod 类别则是一个用来暂

存时段结果的类别。在这里，我们先不列出他们的内容，有兴趣的读者可日后至网站上下载本范例系统的完整原始码。

接下来让我们实作我们的 Planner Session Bean，如同产生 Scheduler Session Bean 的作法，一样的我们可以透过 EJB 设计家产生我们所需要的程序框架，不同于 Scheduler Bean 的是我们将 Planner Bean 设为 Stateful。Planner Bean 的内部，只有一个 method 叫做 nextAvailableTimePeriod()，当呼叫这个 method 时，我们希望每次呼叫的结果都不一样，我们是怎么做到的呢？在 Planner Bean 里面，我们定义了一个 nextStartTime 的数据成员，在每次呼叫 nextAvailableTimePeriod()，我们都会将 nextStartTime 的现值一并传入，当我们收到返回值后，我们同时将 nextStartTime 的值加以更新，以便下次呼叫时传入，以下是 Planner Bean 的部份实作内容：

```
public class PlannerBean implements SessionBean {
 SessionContext sessionContext;
 private Date nextStartTime;
 private Collection attendees;
 transient TimeFinder timeFinder;
... //以下省略
   public TimePeriod nextAvailableTimePeriod() {
   try {
     TimePeriod availablePeriod =
        timeFinder.firstAvailableTime(attendees, nextStartTime);
     if(availablePeriod != null) {
       // 将传回结果的结束时间，设为新的 nextStartTime 内容
       nextStartTime = availablePeriod.getFinishTime();
     }
     return availablePeriod;
   } catch (Exception e) {
     throw new javax.ejb.EJBException(e);
   }
 }
```

在撰写完了 Planner Bean 的实作部分后，同样的我们要将它的 EJB Local References 设上，因为在 Planner Bean 中有使用到 Attendee Bean 及 TimeFinder 类别，而在 TimeFinder 类别中，我们又调用了 Seminar Bean，所以我们在 Planner Bean 的 EJB Local References 中设上 Attendee 及 Seminar 两个 Entity Bean。下图是我们 Planner Bean 的设定页面：

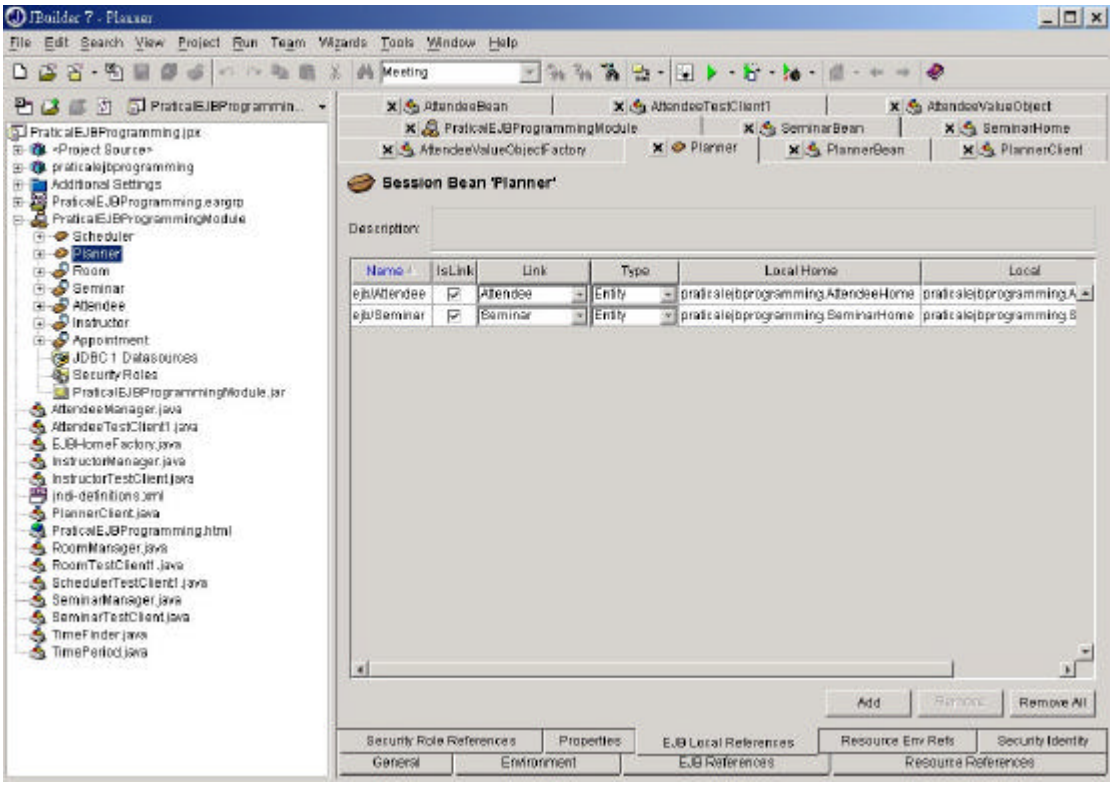

图 8，Planner Session Bean EJB Local References Properties 之显示窗口

实作 Test Client

以上我们介绍完了我们范例系统所要用到的两个 Session Bean，同样的我们将所需的类别档案加以重新打包，重新产生我们的 PracticalEJBProgramming.EAR 档案。同时将其布署至我们的 EJB Container。现在让我们来撰写一个测试程序 PlannerClient 来看 Planner Session Bean 是否能如我们所预期的正常运作，有关 Scheduler Bean 的测试，由于上期已做过，在此先省略不谈。下面是 PlannerClient 程序的主要内容：

```
public static void main(String[] args) {
  try {
    //get naming context
    Context ctx = new InitialContext();

    System.out.println("Looking for PlannerHome");
    //look up jndi name
    java.lang.Object ref = ctx.lookup("Planner");

    //cast to Home interface
    PlannerHome plannerHome = (PlannerHome)
```

```
      javax.rmi.PortableRemoteObject.narrow(ref, PlannerHome.class);

    System.out.println("Initializing meeting data");
    java.util.Collection attendeeIds = new java.util.Vector(2);
    attendeeIds.add(new Long(1));
    attendeeIds.add(new Long(2));

    Calendar cal = Calendar.getInstance();
    cal.set(2002, Calendar.OCTOBER, 8, 0, 0, 0);
    Date startTime = cal.getTime();

    System.out.println("Start time: " + startTime);
    System.out.println("Looking for periods of availability");
    Planner planner = plannerHome.create(attendeeIds, startTime);
    for(int i=0; i < 5; i++) {
      TimePeriod tp = planner.nextAvailableTimePeriod();
      if(tp == null) {
       break;
      }
      System.out.println(tp.getStartTime() + ", " +
       tp.getFinishTime());
    }
    System.out.println("Done");
    planner.remove();
  }
  catch(Exception e) {
    System.err.println("Exception caught");
    e.printStackTrace();
  }
}
```

首先我们一样透过 JNDI **取得** Planner Bean **的** Home Interface reference**，然后我们呼叫他的** create() method**，同时传入两个值作起始化动作。一个是参加** Seminar **人员名单的** Vector**，另一是参考的起始时间。为了要看** Planner Bean **是否能正常运作，我们特地用了** for loop **循环来呼叫** nextAvailableTimePeriod()**五次，如果运作正常的话，我们应该会收到五个不同的时段返回值。下图是我们的测试结果：**

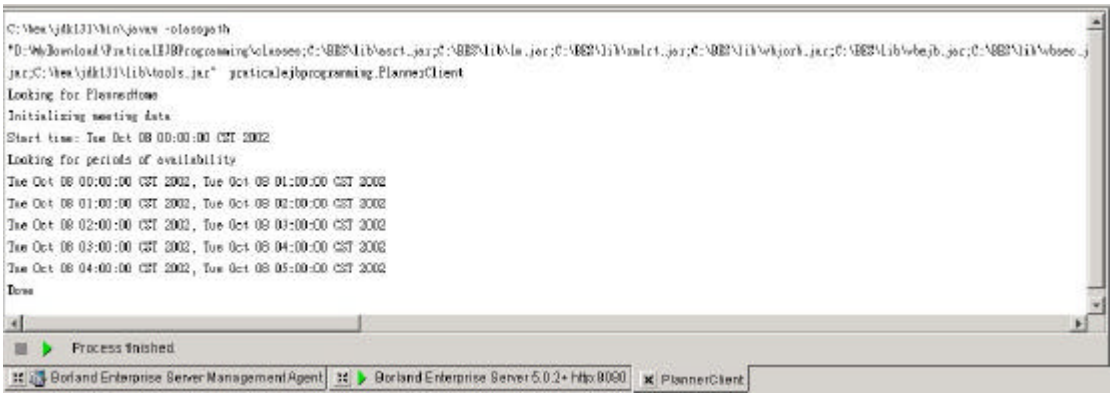

图 9，PlannerClient 测试结果

我们可以发现结果如我们预期一般，Planner Bean 返回了五笔不同的时段给我们，接下来让我们看一下在 BES 中，所产生的实例数目及其调用情形，经过五次的呼叫，Planner 实例仍是只有一个，正如我们所预期，如下图所示：

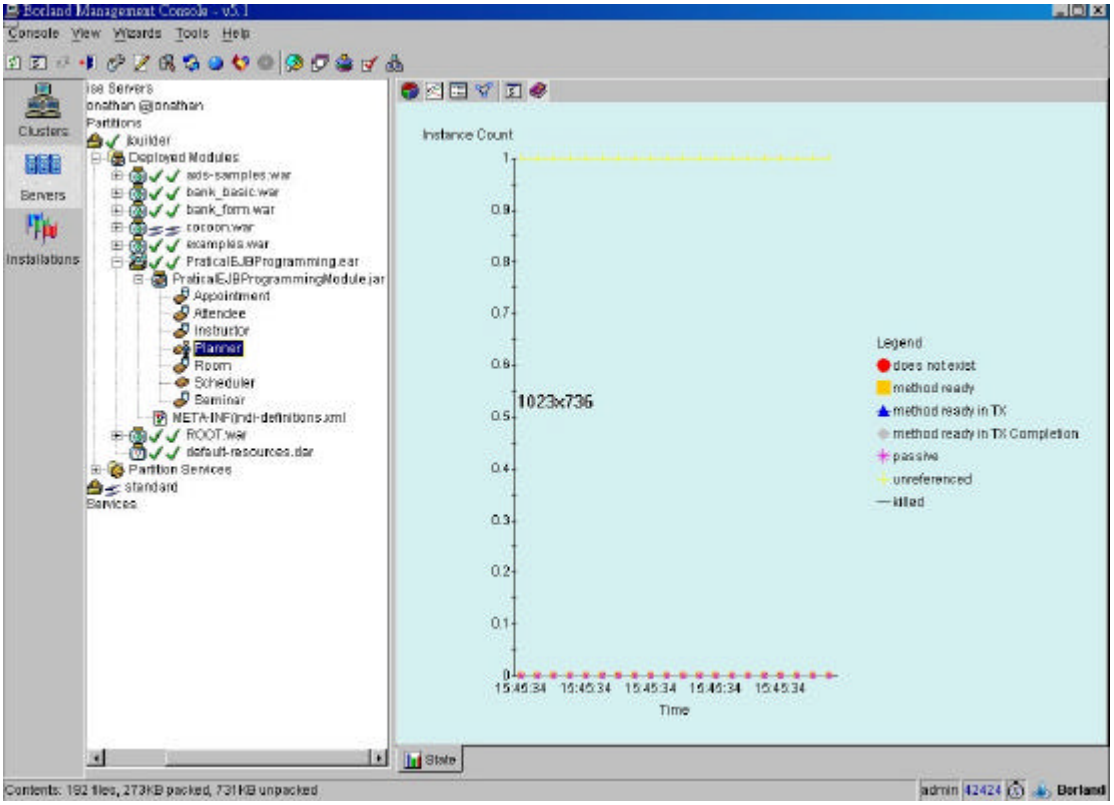

图 10，Borland Enterprise Server Console 中 Planner Session Bean 之显示情况

结论

在这期的文章中，笔者试着说明何谓 Session Bean 组件及 Session Façade Design Pattern，同时希望藉由我们范例系统的设计，尝试让读者了解如何透过 J2EE Design Patterns 的设计考量，来设计我们的系统及运用所谓的 Session Façade Design Pattern。一般说来，Session Bean 的运用往往和项目所设计的 use cases 相结合，但这并不代表各位要将每一个 use case，都以一个 Session Bean 来代表，在实作上，通常我们会将相关的 use case 加以结合，而以一个Session Bean 来作代表，同时在设计之时，可根据我们use case 的需求，而将 Session Bean 设计为 Stateless 或 Stateful。

另外透过 Session Façade Design Pattern 的设计，我们可提供一个系统化且一致性的窗口，同时在存取 Bean 实例上及 Transaction 的控管上，也提供一个较好的接口。在这里希望读者也能够加以尝试。

由于笔者所知有限，如果读者有任何的建议和问题都欢迎和笔者们讨论。我想读者在经由本文的介绍之后，应该能有初步的了解，希望读者也能在实际的项目开发上加以运用，毕竟凡事坐而言，不如起而行。在这里，我们先说再见了。

注 1 读者可参考下列有关 EJB 的书籍，了解有关 Session Bean 的介绍，

- **Mastering Enterprise JavaBeans, 2nd Edition** by Ed Roman, Scott W. Ambler, Tyler Jewell
- **Enterprise JavaBeans, 3rd Edition** by Richard Monson-Haefel