G08_接口设计之美_通用性接口的组合应用

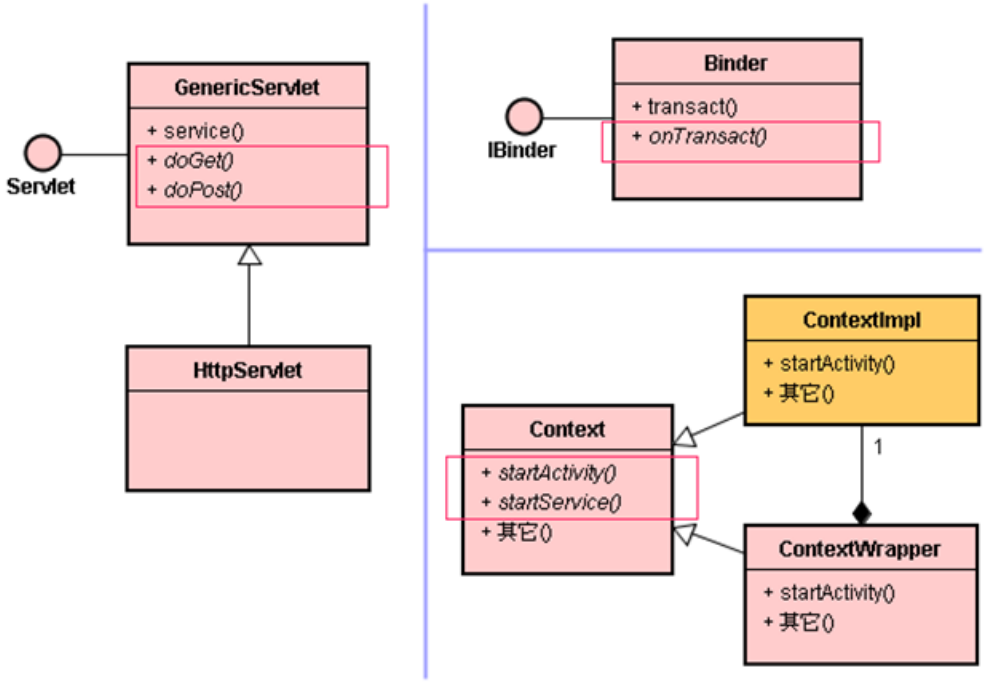
内容：

◇ 复习：通用性接口的基本结构

◇ 此结构是由 2 个 EIT 造形所组成

前言：

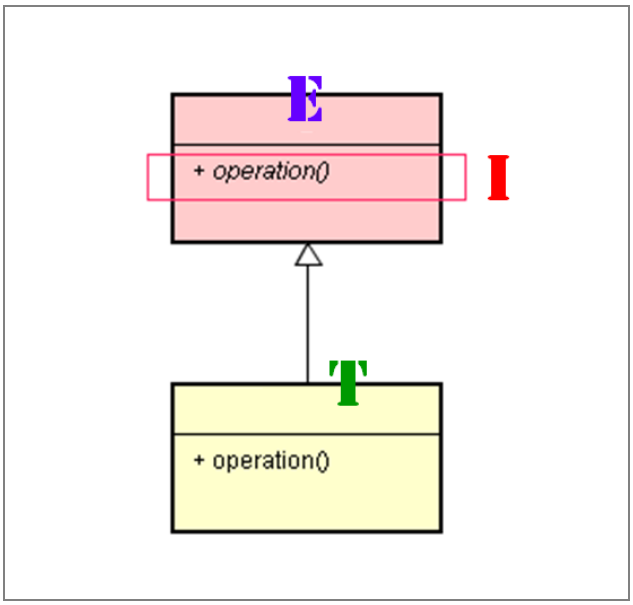
- 在上一单元里，我们自己设计了通用性接口，来整合 Agile-Use Case 的开发。
- 在这单元里，将运用多个既有的通用性接口，联合应用来整合 iPhone 手机与 Android TV。



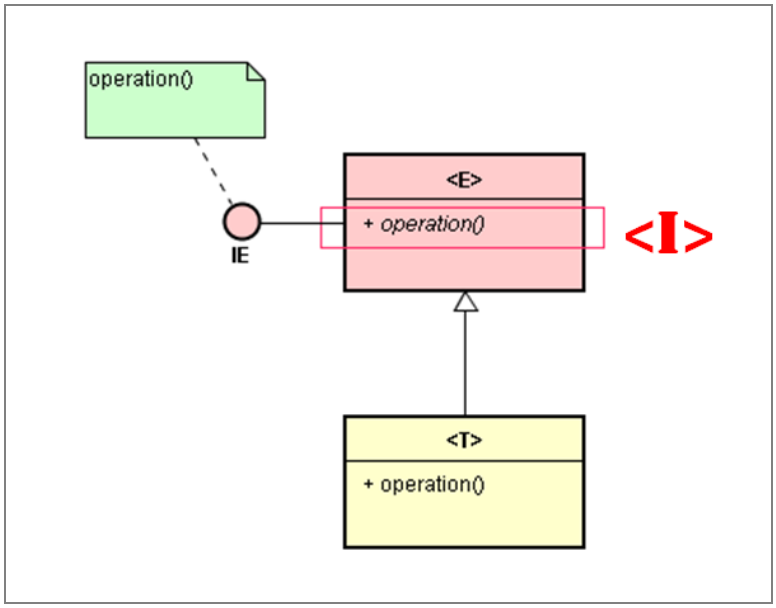
- 本单元的范例，以前也介绍过；然而，在这里将具焦于通用性接口的联合应用上。

1. 复习：通用性接口的基本结构

兹看看一个大家已经很熟悉的 EIT 造形：

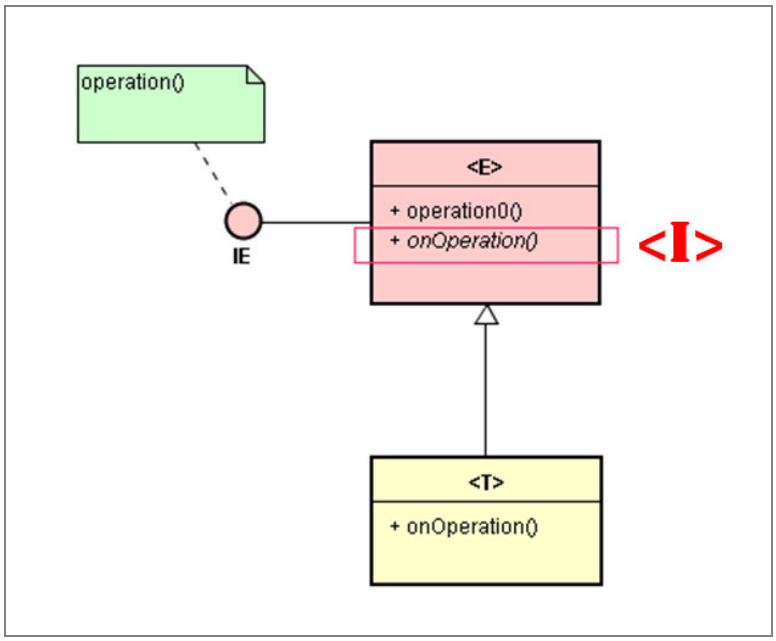


这个<E>可以提供一个接口，例如下图的 IE 接口：

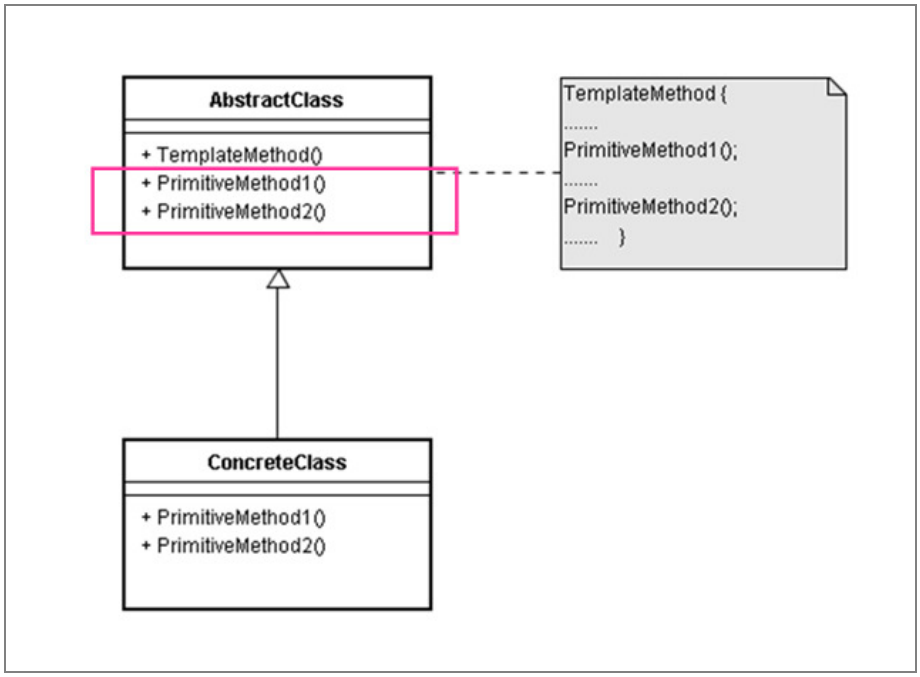


<E>提供了 IE 接口给 Client 使用；于是 Client 透过 IE 接口而调用<E>的 operation()函数，由于它是一个抽象函数，所以就直接执行了<T>的 operation()函数的实现代码。这里的 IE 接口与<I>接口是一致的。

这就是一个通用接口的设计模式(Pattern)了；它有一个更常见的变形：



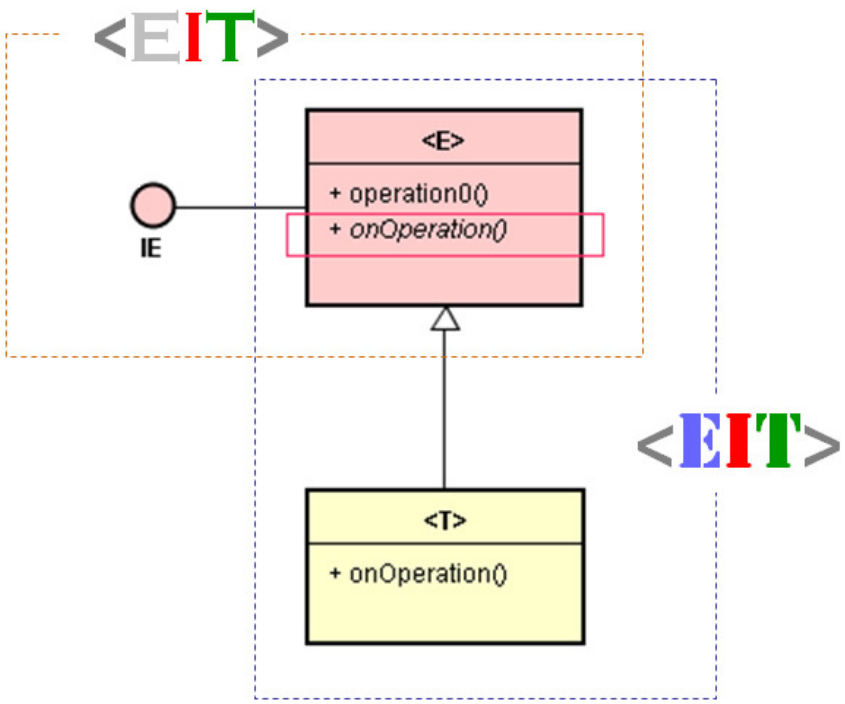
其中的 operation()函数将 IE 接口转换成为<I>接口；于是 Client 透过 IE 接口而调用<E>的 operation()具象函数，然后调用 onOperation()抽象函数，就执行了<T>的 onOperation()函数的实现代码。这里的 IE 接口与<I>接口是不一样的。这通用接口的设计模式，非常接近 GoF 的 TemplateMethod 设计模式，如下图：



2. 此结构是由 2 个 EIT 造形所组成

2.1 通用性接口模式(Pattern)

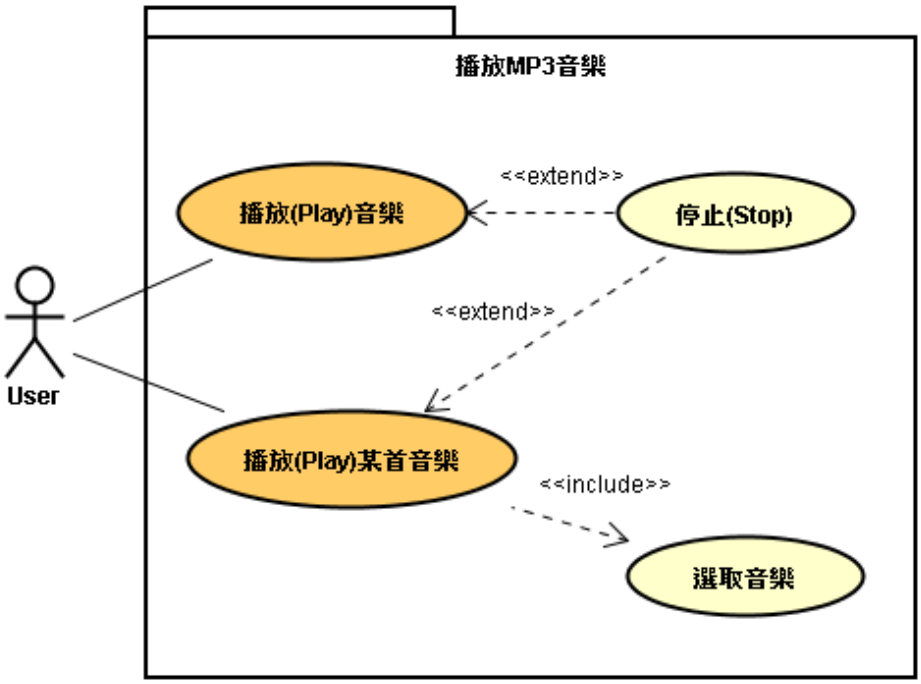
其实，上述这是接口设计模式，是由两个 EIT 造形所组成的，所以我们称之为”模式”(Pattern)，而不称为”造形”(Form)。



上面的 EIT 造形，是一个退化的 EIT 造形，没有限定它的<E>；也可以说，所有的 Client 都能扮演它的<E>角色，而是这个 IE 接口就成为”通用性接口”了。

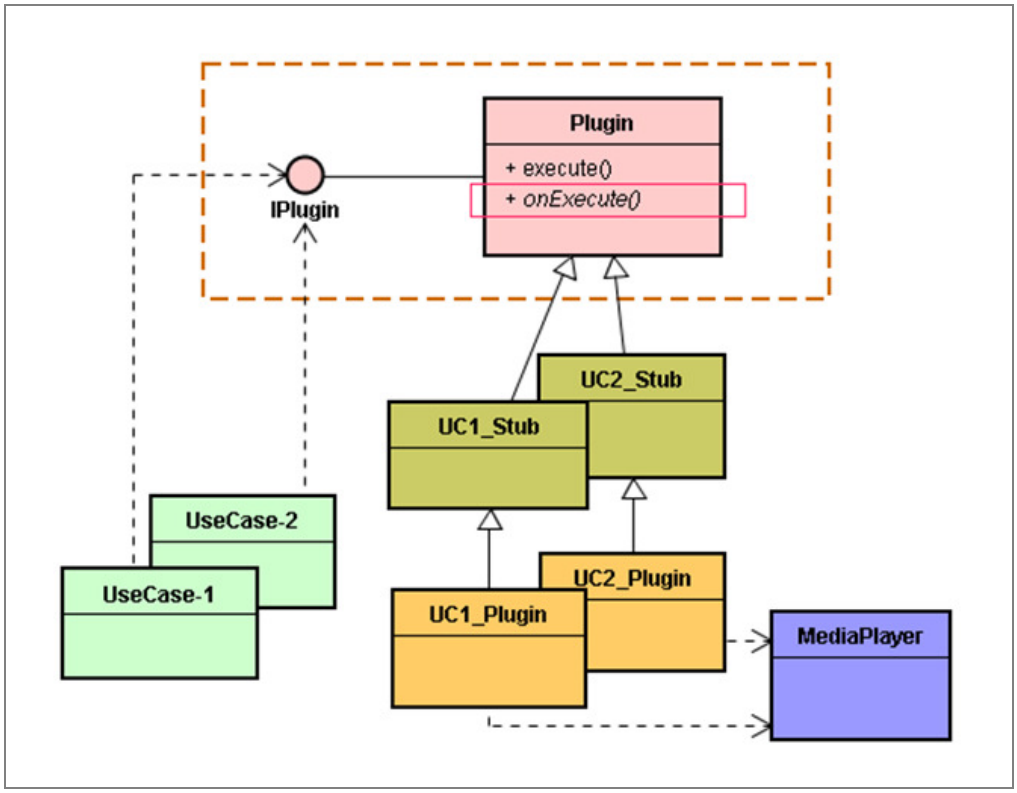
2.2 自己设计通用性接口：上一节的范例

在上一节里的”播放 MP3 音乐”的范例，就曾经亲自设计通用性接口，来支持敏捷开发，有效整合 Use Case 的代码。其 Use Case 图：



这用例图里，含有两个用例：1)播放(Play)音乐；2)播放(Play)某首音乐。

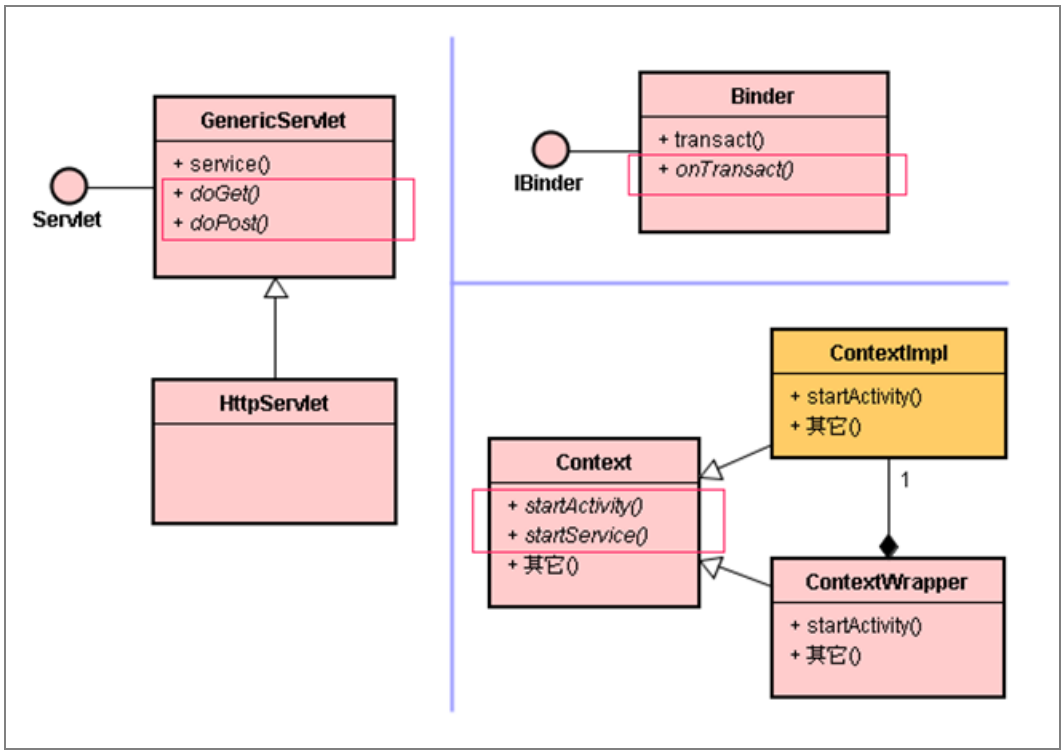
基于这项需求(Requirements)，而设计了一个通用性接口，成为一个简单的应用框架，如下图：



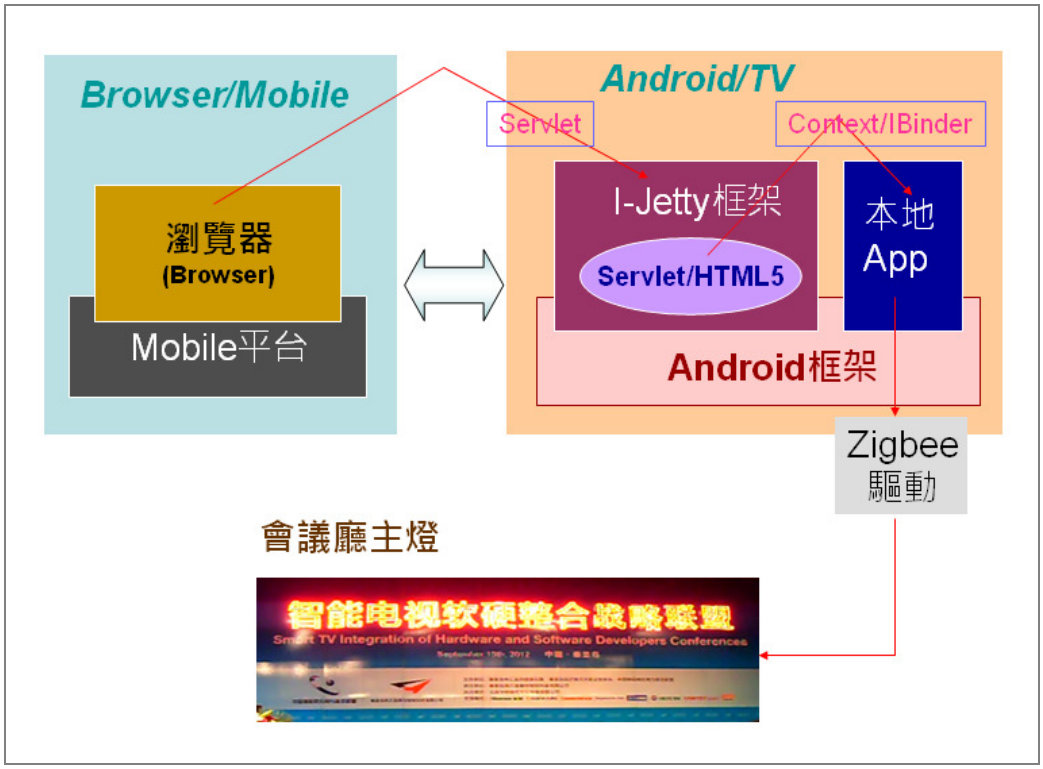
此范例展示如何建立一个框架(Framework)：包含一个 IPlugin 接口和一个 Plugin 基类。由它来提供统一的通信接口，来整合各敏捷跌代过程中所开发的各用例代码。

2.3 也善用(别人)既有的通用性接口：本节的范例

在本节(单元)里，将运用多个既有的通用性接口，联合应用来整合 iPhone 手机与 Android TV。其中，包括了 Android 的 IBinder 接口和 Context 接口；以及 i-Jetty 的 Servlet 接口。



透过这 3 个通用性接口的串接，就能顺利将 iPhone 手机与 Android TV 衔接起来，让用户能使用自己的手机，上网去开起大会议厅里的主灯和电视墙等。如下图：


Browser/Mobile
瀏覽器
(Browser)
Mobile平台
Android/TV
Servlet
Context/IBinder
I-Jetty框架
Servlet/HTML5
本地
App
Android框架
Zigbee
驅動
會議廳主燈
智能电视软硬整合战略联盟
Smart TV Integration of Hardware and Software Developers Conferences

☆

G08_接口设计之美_通用性接口的组合应用

内容：

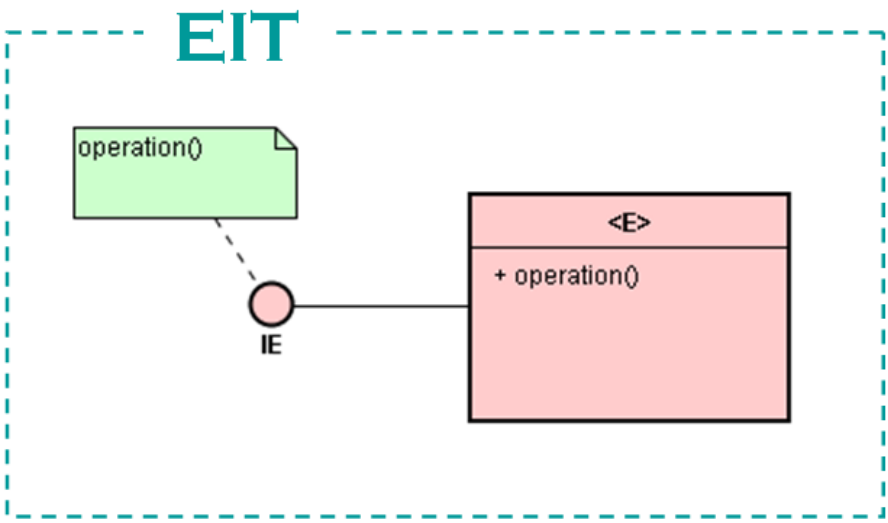
1. 复习：通用性接口的基本结构
2. 此结构是由 2 个 EIT 造形所组成
3. 谁来”实现”通用性接口呢?
4. 应用范例：手机与 Android TV 的多机整合
   - 4.1 应用情境
   - 4.2 介绍 3 个通用性接口：Servlet、Context 和 IBinder
5. 范例架构设计：联合应用 3 个通用性接口
   - 5.1 iPhone 手机端的规划
   - 5.2 i-Jetty 的 Servlet 接口与浏览器对接
   - 5.3 Android 的 Context 和 IBinder 接口与 myServlet 对接
   - 5.4 Android 本地 App 的设计
6. 结语

前言：

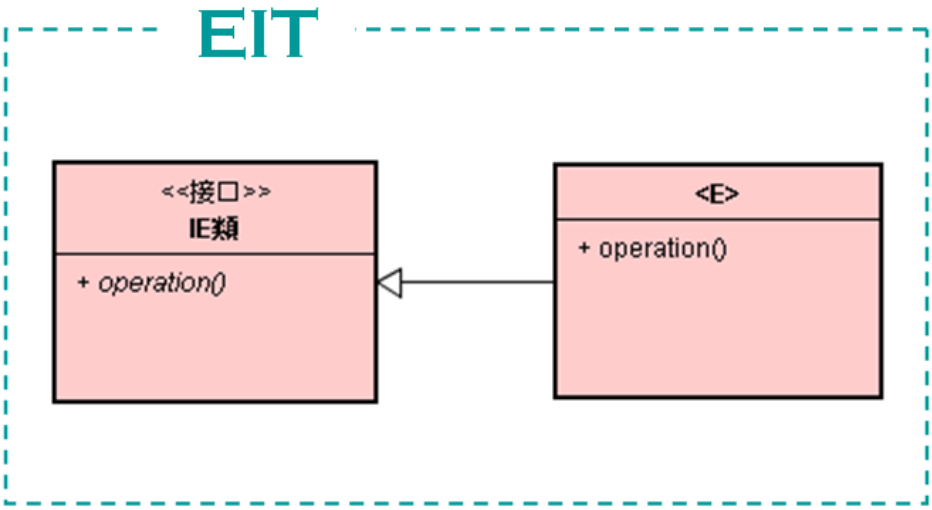
- 在上一节所介绍的通用形接口<I>，是一种标准的结构：由子类<T>来实现接口的函数。
- 在本节里，将介绍一种较为特殊的结构：子类<T>并不直接实现接口，而是委托别的类来实现接口<I>。

3. 谁来”实现”通用性接口呢?

谁来实现(Implement)通用性接口呢？例如下图：



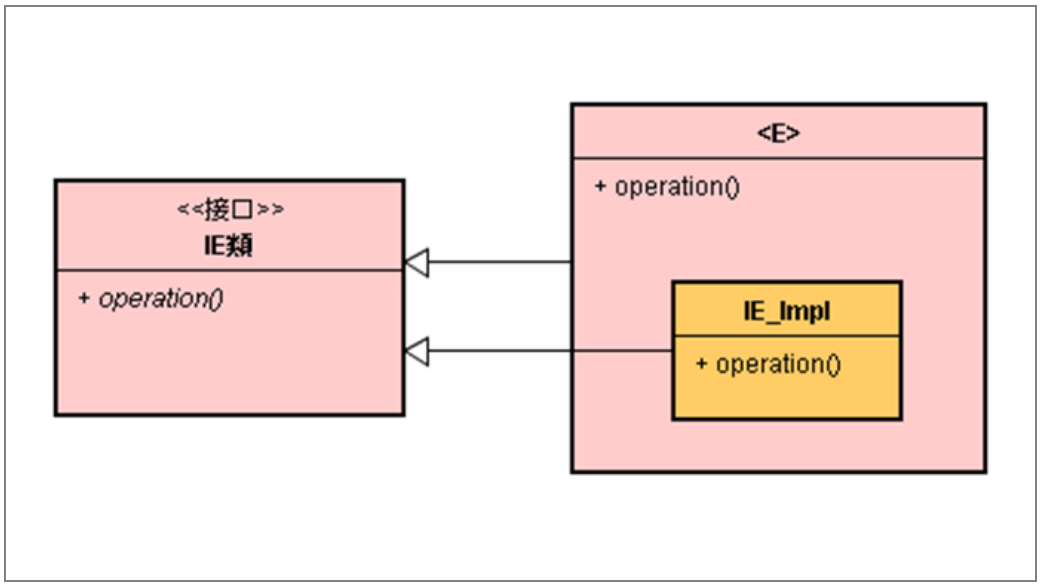
你会回答：当然由<E>来实现这个 IE 通用性接口。这项答案是对的。由于接口(Interface)就是一个纯粹抽象类(Pure Abstract Class)，上图就相当于下图：



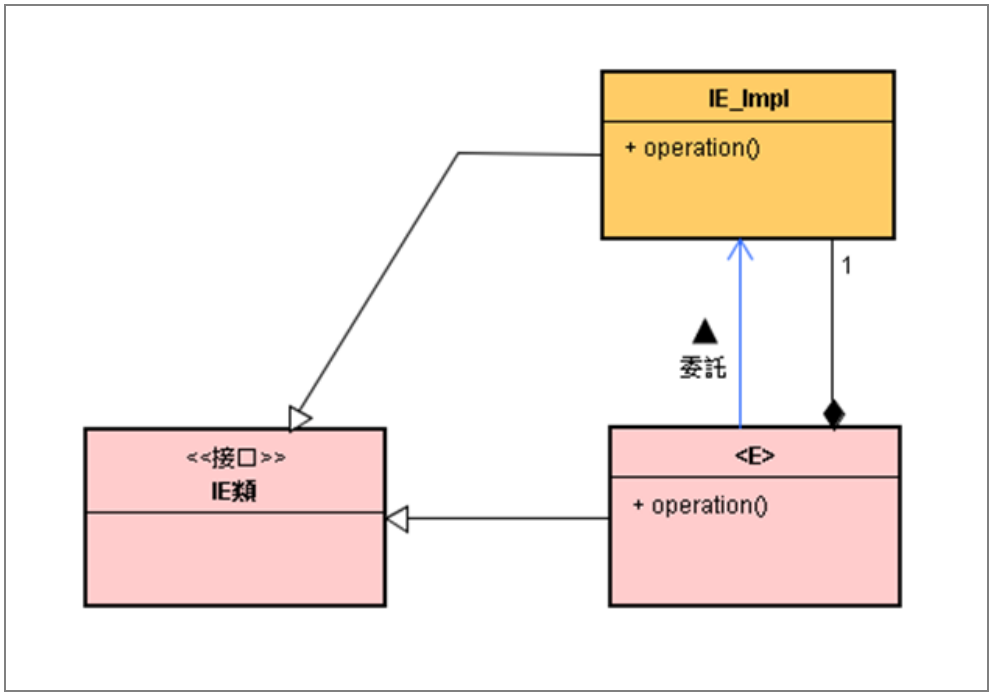
一样地，这也是由<E>来实现这个 IE 类所提供的通用性<operation>接口。由于 IE 类里的 operation()是抽象函数，<E>必须提供 operation()函数的实现代码。然而，有一种状况，就是当我们必须赚写很多个<E>类来实现同一个 IE 接口时，而且实现代码是相同的；此时将会重复赚写一样的实现代码，其开发和维护成本就上升了。此时，可以采取”委托实现”的架构。

委托(Delegation)实现

我们可以将上述的 IE 接口的实现代码集中于一个小类里，如下图里的 IE_Impl 类：

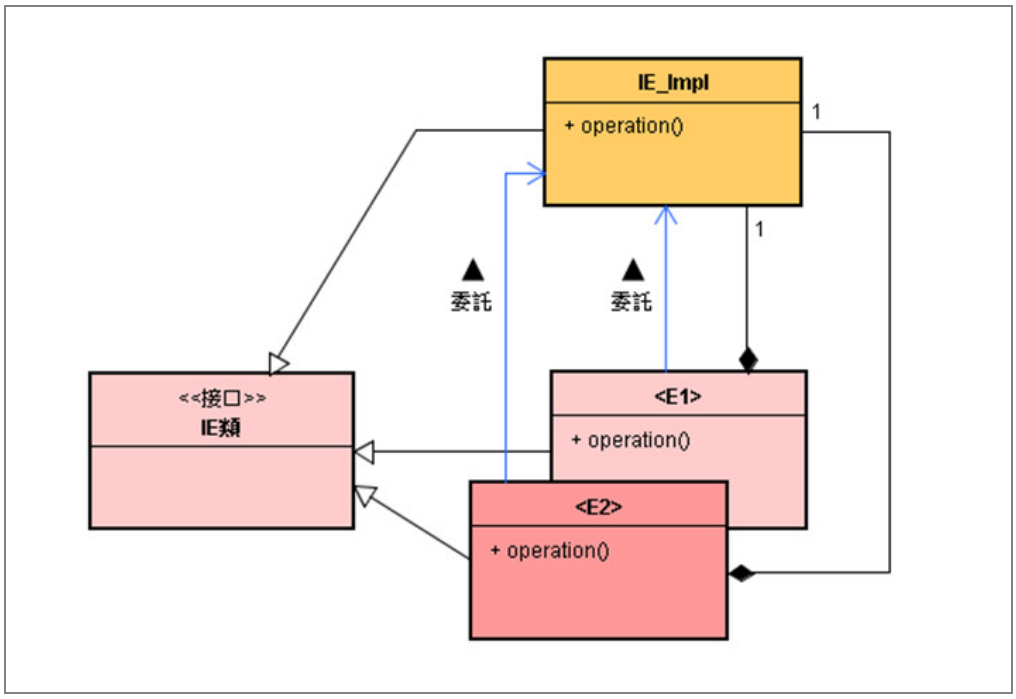


此时，<E>的 operation()函数可以去调用 IE_Impl 类里的 operation()函数，执行其实现代码了，它(即<E>)的 operation()函数就不必重复撰写 operation()函数的实现代码了。这项途径，在软件设计上称为”委托”(Delegation)。也可以将 IE_Impl 类独立出来，如下图所示：



本来是<E>类必须撰写 operation()函数的实现代码，为了避免重复撰写实现代码，<E>类就就转而”委托”IE_Impl 类，调用它的 operation()实现代码。

这种架构设计的效益是，<E>类不必再重复撰写 operation()函数的实现代码，可节省开发、维护实现代码的负担。当<E>类的个数愈多时，这种架构设计的效益就愈大。如下图所示：



图里的黑色菱形符号(♦)表示<E>类都会创建一个 IE_Impl 类的对象，然后<E>类的 operation()函数会调用 IE_Impl 类的 operation()的实现代码 也就是，<E>类将它自己的 operation()函数的实现工作委托给 IE_Impl 类去做了。

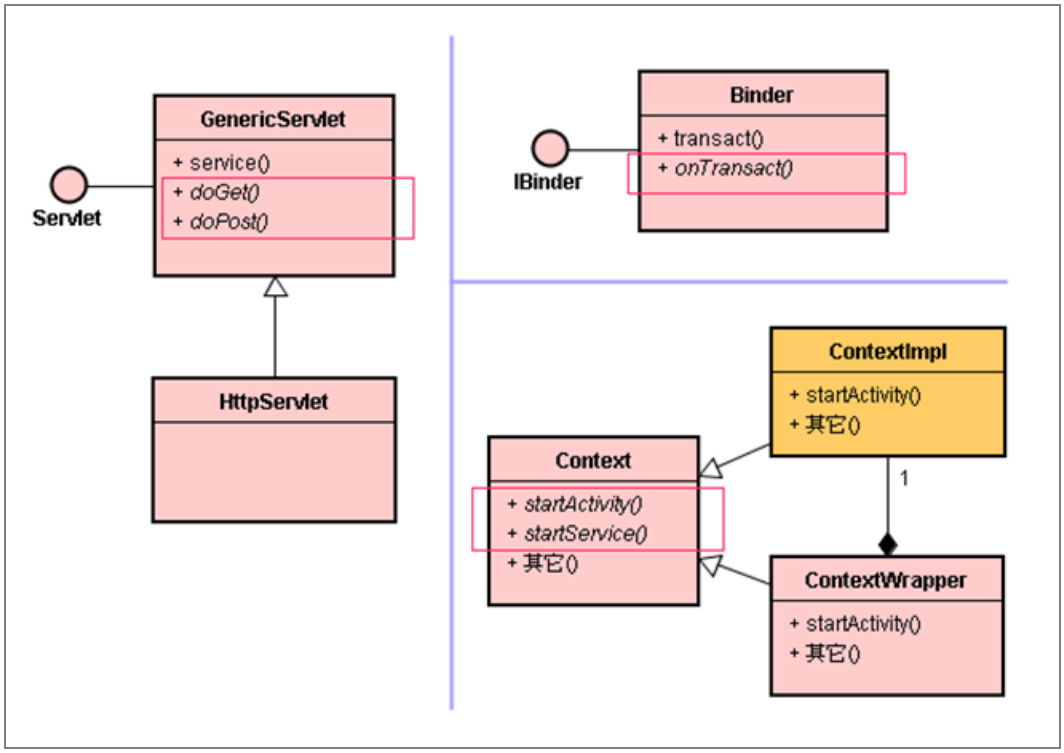
4. 应用范例：手机与 Android TV 的多机整合

4.1 应用情境

其实，大家已经看过这个范例了。这是 2012 年 9 月 21 日于秦皇岛市举行的<智能电视软硬整合产业联盟会议>>开幕时，副市长以他自己的手机上网访问会议厅中 Android TV 里的 i-Jetty 网页，再透过 Android App 发出 Zigbee 信号，打开会议厅的主灯和电视墙。

4.2 介紹 3 个通用性接口：Servlet、Context 和 IBinder

在本节里，你将会看到多个通用性接口组合应用的壮丽情境。其中，包括了三个主要的通用性接口，如下图：



其实，而所谓”通用性”又分为不同等级。例如，从上图里，就能看到接口的”通用性”级别了。其中的 Servlet、IBinder 和 Context 都是最通用的接口，来自各方的 Client 都能使用它。而 <doGet, doPost> 就是属于 GenericServlet 类的专属接口；相对上，Servlet 接口的通用等级，就比<doGet, doPost>接口来得高。同样地，IBinder 接口的通用等级，就比<onTransact>接口来得高。☆

G08_接口设计之美_通用性接口的组合应用

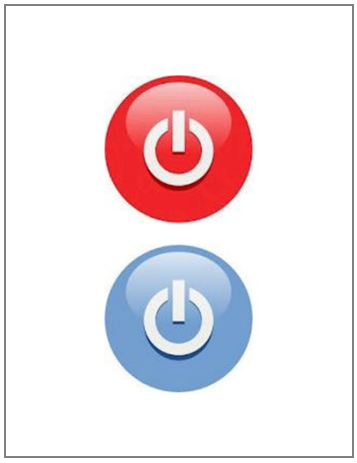
内容：

5. 范例架构设计：联合应用 3 个通用性接口

5.1 iPhone 手机端的规划

在这个范例里，为了让各种移动终端皆能来访问 Android TV，就不特别去开发 iPhone 客户端的 App 了。而是，直接采取通用的一般浏览器(Browser)做为这个案例的手机客户端软件了。如下图：

如上图所示，从手机上网联机之后，手机浏览器出现首页画面如下：



这首页的网页 HTML 代码如下：

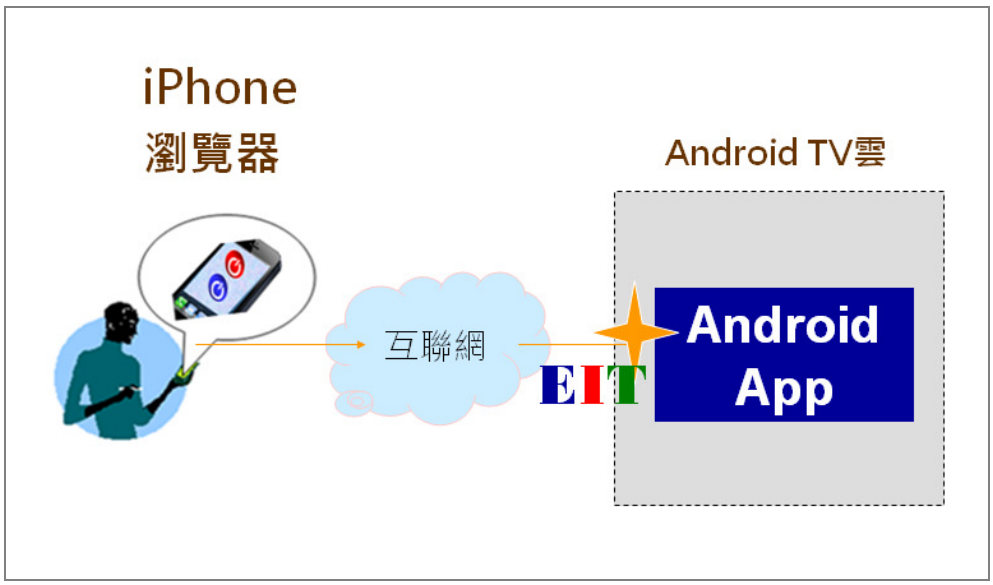
范例代码

//index.html

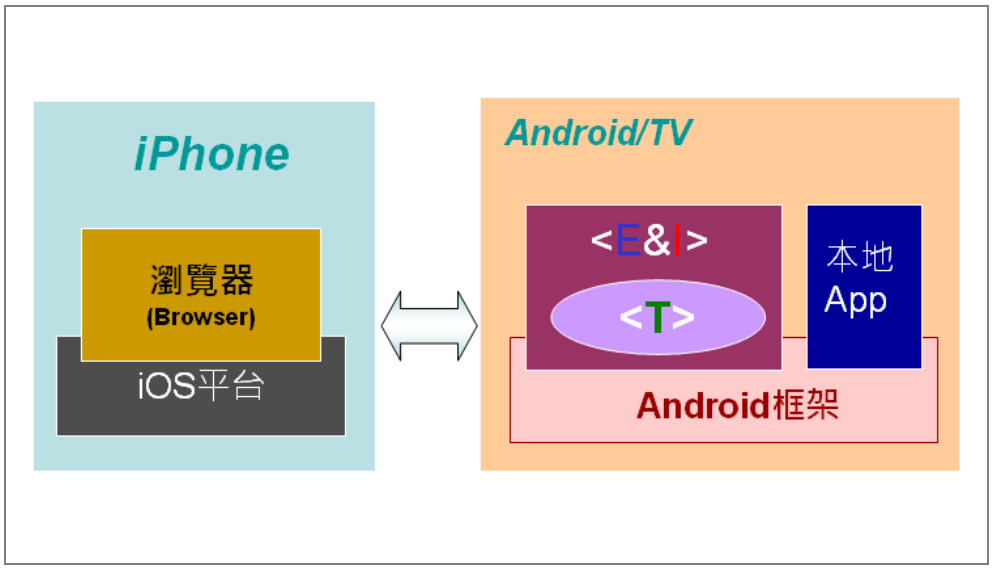
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gbk">
<title>Insert title here</title>
</head>
<body>
<h1>智能家庭 – 移动互联网</h1>
    <div >
        <li>控制操作:</li>
        <p>
        <li ><center><a href="ZigbeeDemo?zigbeecmd=1"><img src="RED.jpg"
border="0"></a></center></li>
        <p>

        <li><center><a href="ZigbeeDemo?zigbeecmd=0"><img src="BLUE.JPG"
border="0"></a></center></li>

    </div>
</body>
</html>
```

5.2 衔接 iPhone 手机端的(通用性)接口：Servlet

iPhone 手机与 Android TV 是两个相互竞争的平台，也是各自发展的。那么，又如何将两个平台整合起来呢？答案是：以 EIT 造形来实践<通用性接口>。如下图：

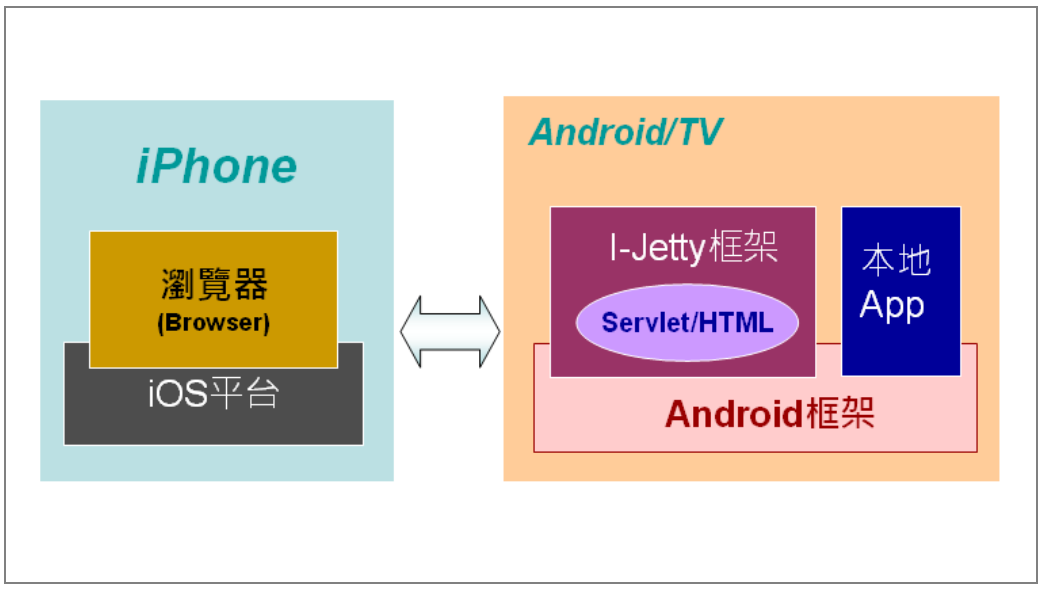


其中，<E>提供通用性接口来与手机端的浏览器对接；而<T>则与本地 App 来对接。然后<E>则透过<I>来与<T>衔接起来。

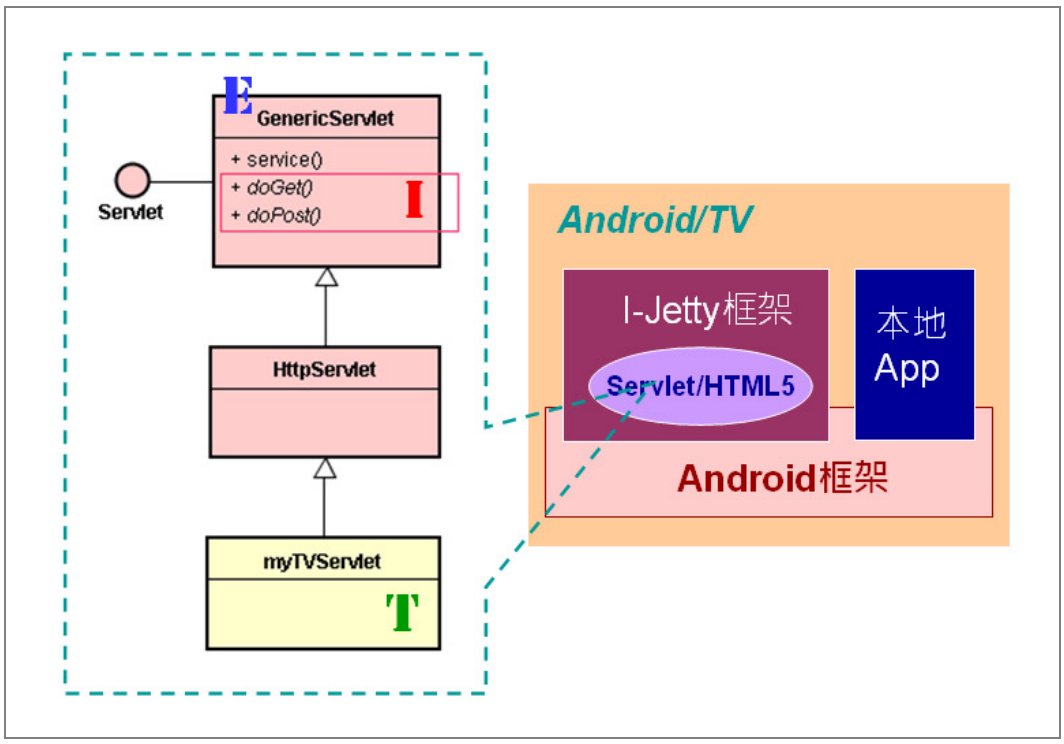


于是，我们找到现成的<E&I>平台，就是：i-Jetty 框架。

这 i-Jetty 本身是以 Android 应用程序形式嵌入(运行)于 Android 平台里，它可以透过 Android 框架的 API 与其它应用程序沟通。因而，在 i-Jetty 里执行的 Servlet 程序也能透过该 Android API 而与其它应用程序沟通、分享数据。例如，在家庭的 Android TV 里，加入一个 i-Jetty 插件，来容纳 HTML 网页和 Servlet 程序。数千公里外的家庭成员，透过手机 Browser 解析 HTML，与家里的 TV 沟通，形成大小屏互动、多机整合的架构设计了。



有了 i-Jetty 框架之后，我们只需要基于 GenericServlet 的<E&I>来撰写子类<T>就行了；如下图：



由 i-Jetty 提供 Servlet 通用性接口来与 iPhone 手机上的浏览器对接。在 i_Jetty 里，转换成为<doGet, doPost>接口，再经由<T>来使用 Context 通用性接口，调用它的 startService()函数，来取得 IBinder 通用性接口，进而与 ZigbeeAppActivity&硬件驱动互相通信。就完成两端整合的任务了。这 myTVServlet 类的代码如下：

范例代码

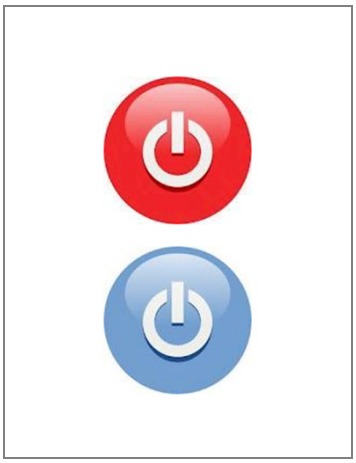
// myTVServlet.java

```
package com.nazca.px;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
```

```
import android.os.IBinder;
import android.os.Parcel;

@WebServlet("/ZigbeeCloudDemo")
public class myTVServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        private    String proofOfLife ;
        private myProxy pProxy = null ;

       public void init(ServletConfig config) throws ServletException {
            super.init(config);
            android.content.Context androidContext =
                  (android.content.Context)config.getServletContext().getAttribute
                                                      ("org.mortbay.ijetty.context");
            proofOfLife = androidContext.getApplicationInfo().packageName;
            androidContext.startService (
                 new Intent("com.google.android.ZigbeeApp.REMOTE_SERVICE"));
            androidContext.bindService (
                 new Intent("com.google.android.ZigbeeApp.REMOTE_SERVICE"),
                           mConnection, android.content.Context.BIND_AUTO_CREATE);
         }
        private ServiceConnection mConnection = new ServiceConnection() {
              public void onServiceConnected(ComponentName className, IBinder ibinder){
                    pProxy = new myProxy(ibinder);
               }
               public void onServiceDisconnected(ComponentName className) {}
              };
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
               String zigbeecmd = request.getParameter("zigbeecmd");
               response.setContentType("text/html;charset=gbk");
               PrintWriter out = response.getWriter();
               out.println("<html>");
              out.println("<h1>智能家庭 – 移动互联网<br></h1>" );
              out.println("<h3>控制操作: <br></h3>");
              if(Integer.valueOf(zigbeecmd)==1){
                       pProxy.setLedStatus(1);
                       out.println("<h3><li>你已开启电源插座</li></h3>");
                       out.println("<h3><A HREF=\"index.html\">返回</a></h3>");
              }
              if(Integer.valueOf(zigbeecmd)==0){
                       pProxy.setLedStatus(0);
                       out.println("<h3><li>你已关闭电源插座</li></h3>");
                       out.println("<h3><A HREF=\"index.html\">返回</a></h3>");
              }
              out.println("</html>");
              out.flush();
              out.close();
        }
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException {}
        //-----------------------------------------------------------------------------------------
        private class myProxy    {
               private IBinder ib ;
```

```
        private String mStatus,mLedStatus ;

        myProxy(IBinder ibinder) {
            ib = ibinder ;
        }
        public void setLedStatus(int key){
            Parcel data = Parcel.obtain();
           Parcel reply = Parcel.obtain();
           try{    ib.transact(key, data, replay, 0);
             }catch (Exception e) {    e.printStackTrace();    }
        }
    }
}
```

当 iPhone 手机的浏览器上网，就出现首页画面如下：



浏览器就来就透过 i-Jetty 的 Servlet 通用性接口，调用到这个 doGet()函数，再经由<T>来使用 Context 通用性接口，调用它的 startService()函数，进而与 ZigbeeAppActivity&硬件驱动互相通信。

G08_接口设计之美_通用性接口的组合应用

内容：

5.3 App 的 Context 接口与 myTVServlet 对接

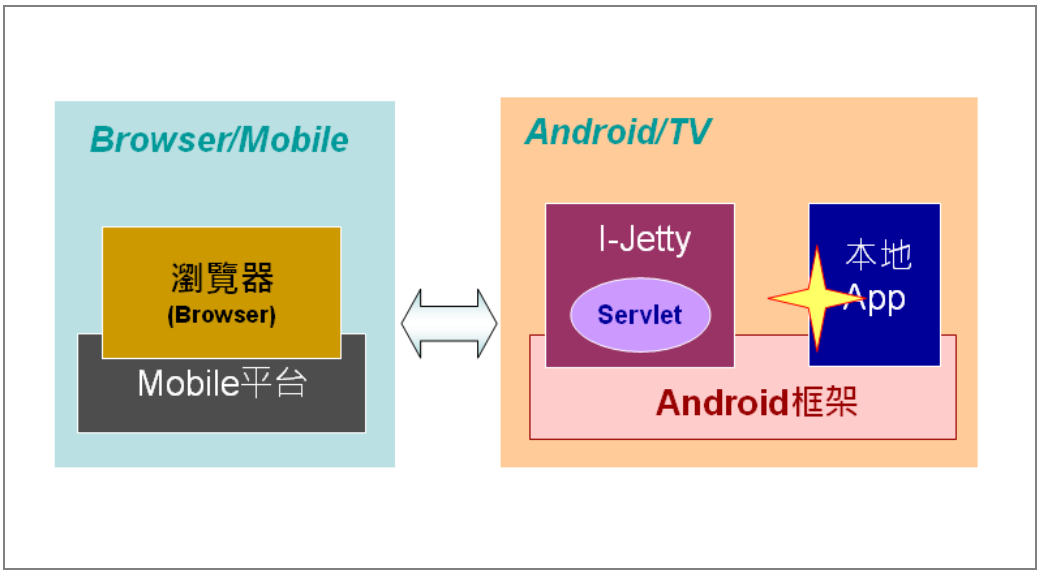
在上一节的 myTVServlet 里，使用了 Android 的 Context 和 IBinder 两个通用性接口，来与 Android 的 App 来沟通：

```
public class myTVServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private    String proofOfLife ;
    private myProxy pProxy = null ;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        android.content.Context androidContext =
            (android.content.Context)config.getServletContext().getAttribute
                                        ("org.mortbay.ijetty.context");
        proofOfLife = androidContext.getApplicationInfo().packageName;
        androidContext.startService (
            new Intent("com.google.android.ZigbeeApp.REMOTE_SERVICE"));
        androidContext.bindService (
            new Intent("com.google.android.ZigbeeApp.REMOTE_SERVICE"),
                mConnection, android.content.Context.BIND_AUTO_CREATE);
     }
     // .....
     }
```

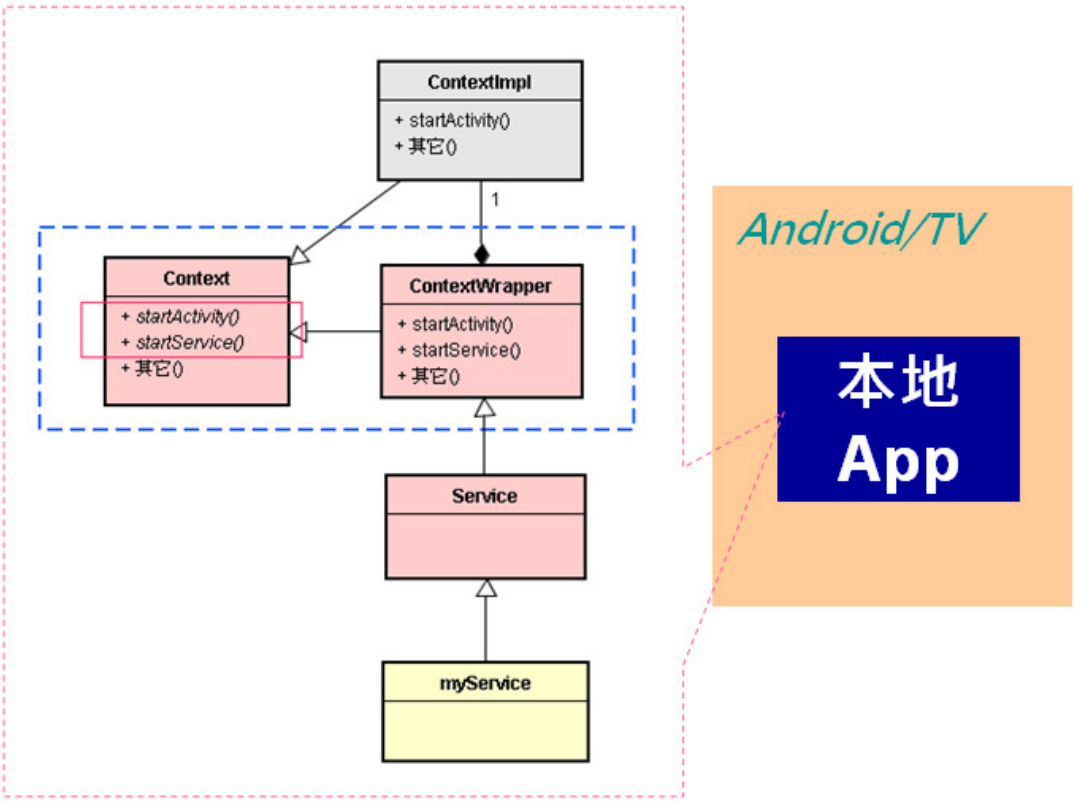
在上一节的 myTVServlet 里，也使用了 Android 的 IBinder 两个通用性接口，来与 Android 的 App 来沟通：

```
    private class myProxy    {
        private IBinder ib ;
        private String mStatus,mLedStatus ;

        myProxy(IBinder ibinder) {
            ib = ibinder ;
        }
        public void setLedStatus(int key){
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            try{    ib.transact(key, data, replay, 0);
             }catch (Exception e) {    e.printStackTrace();        }
        }
    }
```

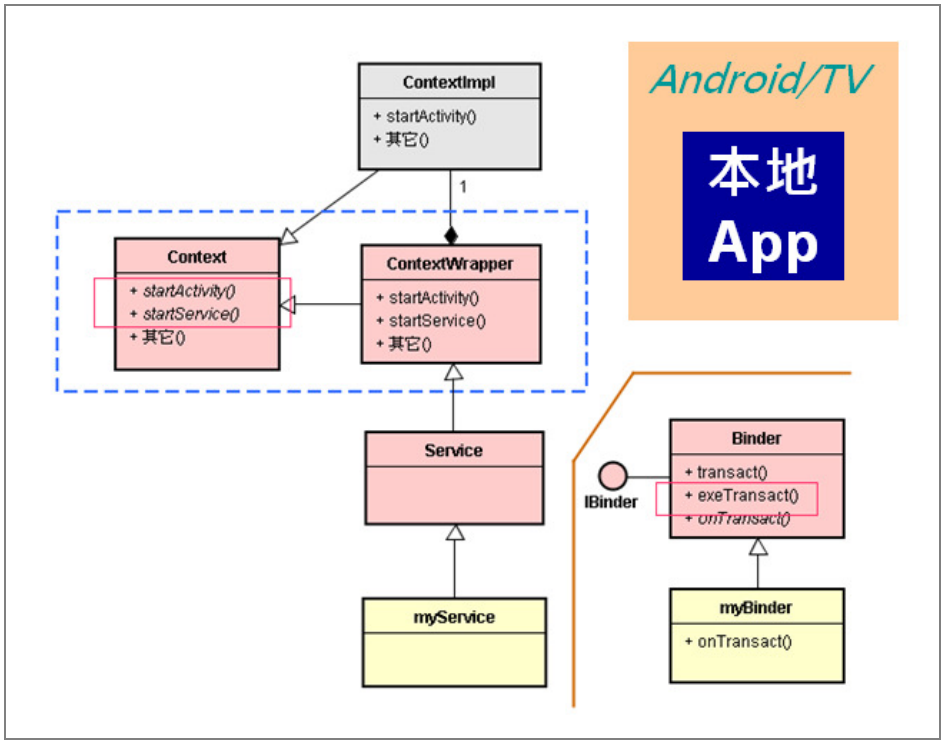
我们使用 Android 框架提供的这两个通用性接口，来与 i-Jetty 框架里的 myTVServlet 沟通。



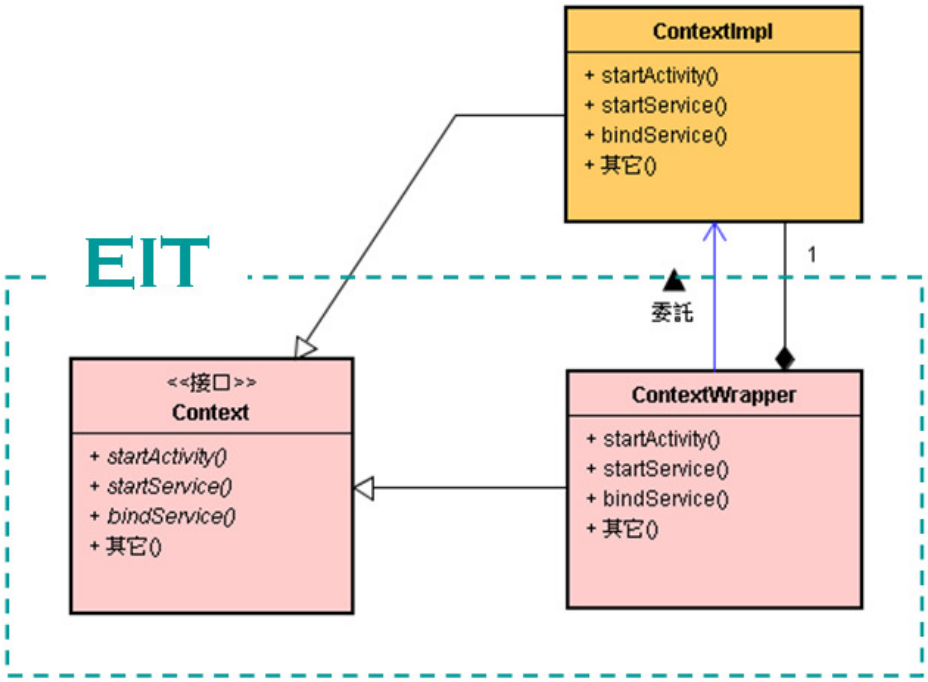
这 Context 接口的详细内涵如下图：



加上 IBinder 通用性接口，其内涵如下图：



这个 Context 通用性接口，是 Android 框架里最具有权力地位的接口。例如，只要你能取得这个 Context 接口，就能调用极具权力象征的 startActivity() 函数。



从上图可以看到，除了 startActivity()函数之外，还有 startService()函数、bindService()函数等，都是 Android 平台上最常用的函数。在 Android 里，上图的实现代码片段如下：

```
public class ContextWrapper extends Context {
        Context mBase;
        public ContextWrapper(Context base) {
              mBase = base;
        }
        //……
        @Override
         public void startActivity(Intent intent) {
                mBase.startActivity(intent);
         }
        @Override
         public ComponentName startService(Intent service) {
                return mBase.startService(service);
         }
          //……
    }
```

其中，mBase 是指针，用来指向 ContextImpl 对象的 Context 接口。也就是说，ContextWrapper 对象都会含有一个 ContextImpl 对象的指针。这让 ContextWrapper 能透过 mBase 而调用 ContextImpl 类里的实现代码。例如，

```
public void startActivity(Intent intent) {
            mBase.startActivity(intent);
    }
```

就是”委托”(Delegation)机制的实际运行了。

```
class ContextImpl extends Context {
        //.....
      @Override public void startActivity(Intent intent) {
            warnIfCallingFromSystemProcess();
            startActivity(intent, null);
      }
     @Override
      public void startActivity(Intent intent, Bundle options) {
             // 实现代码
      }
      //…….
     @Override
      public ComponentName startService(Intent service) {
            warnIfCallingFromSystemProcess();
            return startServiceAsUser(service, mUser);
```

```
    }
    @Override
     public ComponentName startServiceAsUser(Intent service, UserHandle user) {
         // 实现代码
     }
     // ……..
}
```

接下来，就撰写 myService 和 myBinder 两个类的代码了。如下：

范例代码

// myService.java

```
package com.google.android.ZigbeeApp;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.os.Parcel;
import android.util.Log;
import android.widget.Toast;

public class myService extends Service{
        private IBinder myBinder = null ;

        @Override public void onCreate() {
                myBinder = new myBinder(this.getApplicationContext());
        }
        @Override public IBinder onBind(Intent intent){
                return myBinder ;
        }
```

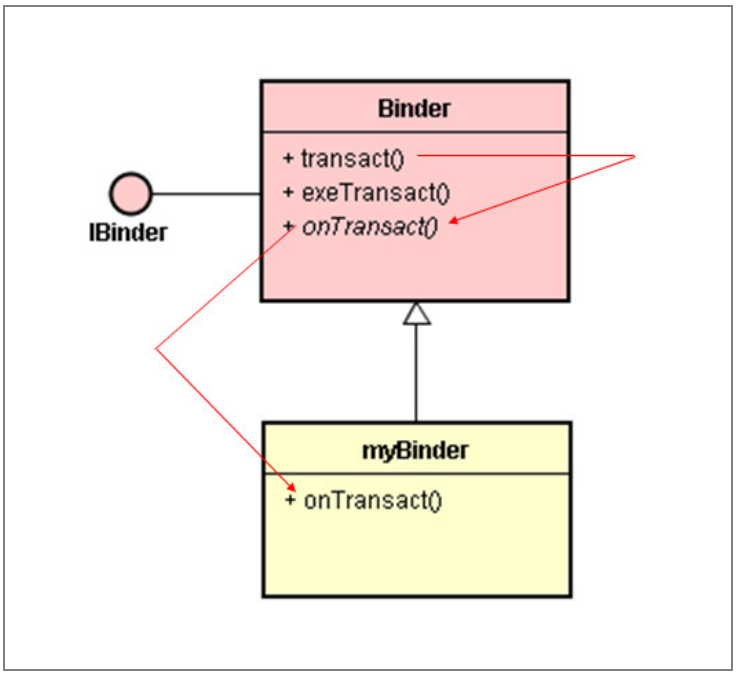
范例代码

// myBinder.java

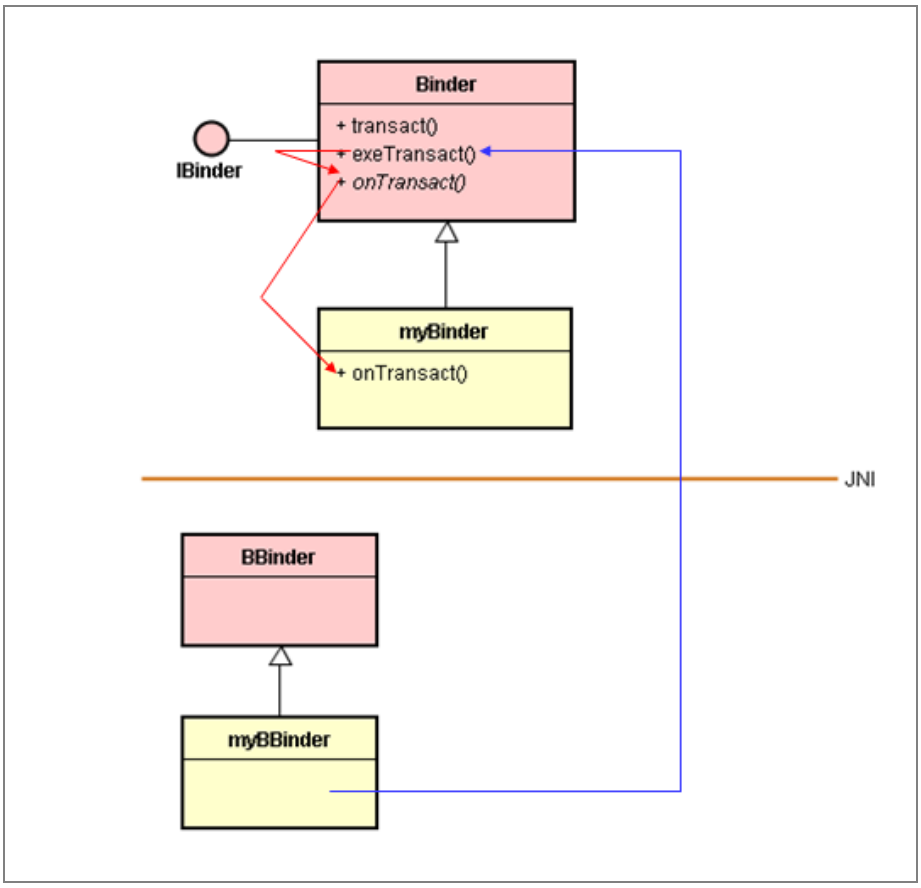
```
class myBinder extends Binder {
         private Context context ;
         private myService srv ;

        public myBinder(Context ctx, myService service){
                context = ctx ;
                srv = service;
        }
        @Override public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
                        throws android.os.RemoteException
                {
                   if (code == 0) {
                          Intent intent = new Intent("android.intent.action.MY_BROADCAST");
                          intent.putExtra("inta", 101) ;
                           sendBroadcast(intent);    // 传送给Activity
                   }
```

```
            return true;
        }
}
```

这里的 IBinder 接口是协助 IPC 的通用性接口。其典型的调用流程，如下图：



其实，这个典型的通用性接口设计模式，还有很多种变化的形式。例如下图：



虽然是一个简单的通用性接口设计模式，但是含有丰富的变化机制。例如上图的机制，就让 C/C++层的模块成为架构的掌控者。然而，必须理解到：因为设计(和掌控)了 IBinder 通用性”接口”设计，才能充分保为了 C/C++层的模块的”逻辑”控制权。这是许多人难以体会的。

G08_接口设计之美_通用性接口的组合应用

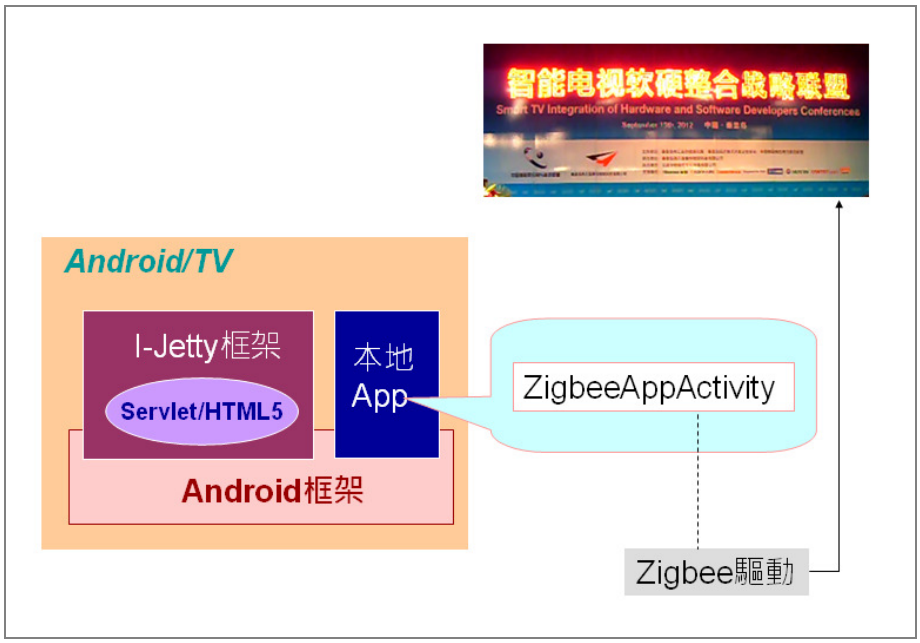
内容：

5.4 Android 本地 App 的設計

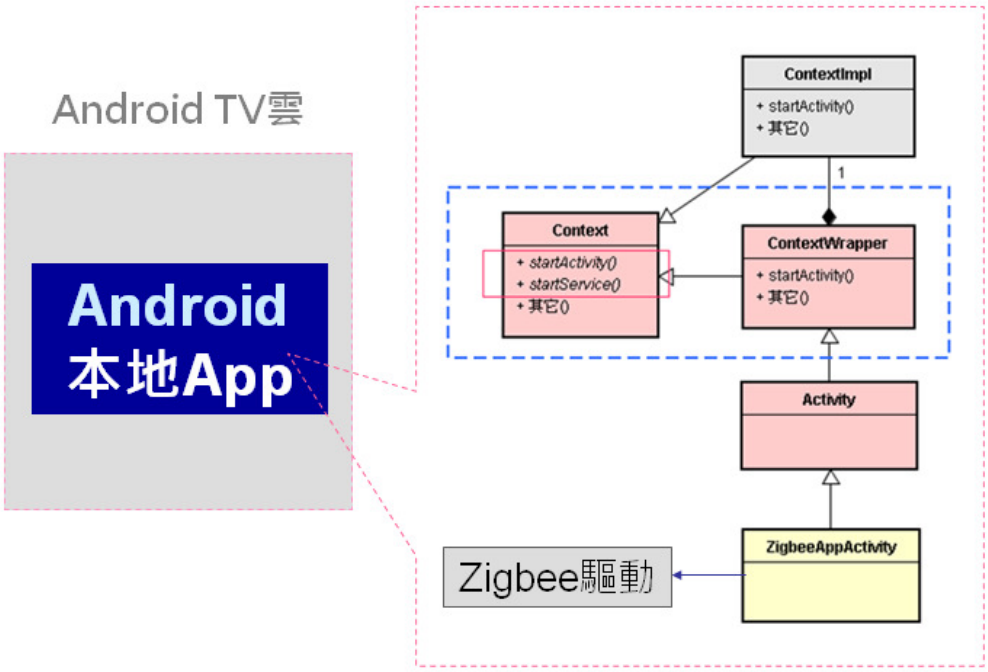
在 Android TV 里，首先规划一支 App，来与底层的 Zigbee 驱动沟通，控制其发出 Zigbee 信号给会议大厅的主灯开关。

撰写一 Activity 应用类：ZigbeeAppActivity

兹设计一个 Android 的本地应用程序(App)，其包含一个应用子类，就是：ZigbeeAppActivity.java。它衔接到底层的 Zigbee 驱动程序，发出 Zigbee 信号给会议大厅的主灯开关。如下图：



这个 ZigbeeAppActivity 应用子类，其幕后就继承了 Context 通用性接口，其架构就如下图：



ZigbeeAppActivity 类，它继承 Activity 类，也继承了 Context 通用性接口。如下述的代码：

范例代码

// ZigbeeAppAcitity.java

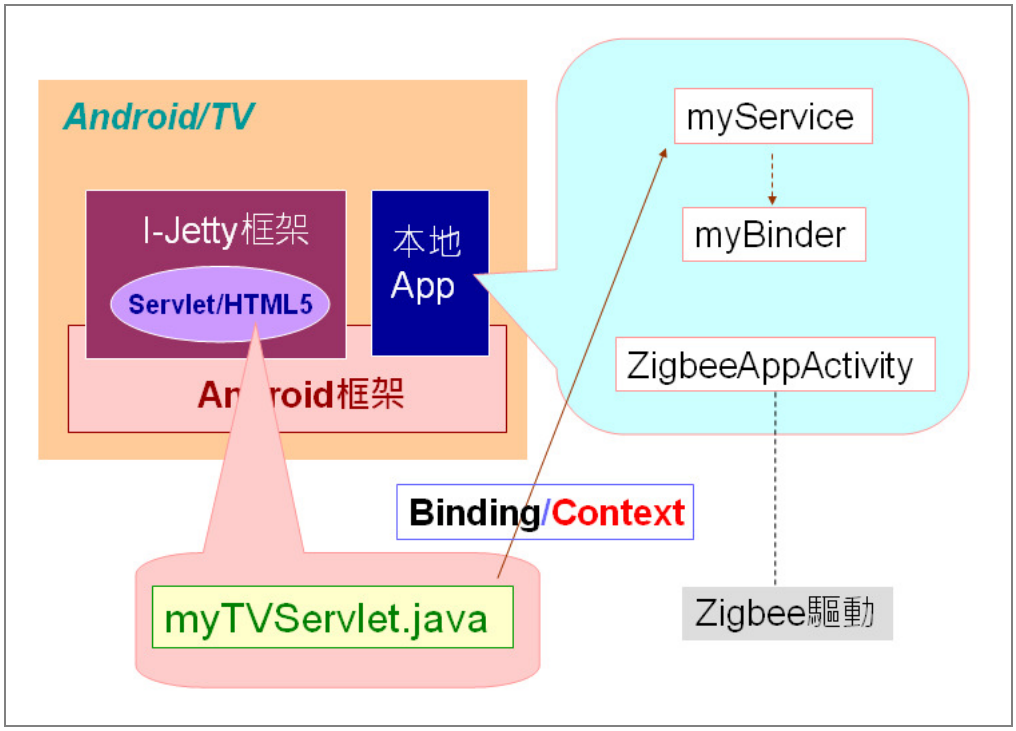
```
package com.misoo.ZigbeeApp;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;

public class ZigbeeAppActivity extends Activity implements OnClickListener    {
        protected static final long DelayMillis = 100;
        myReceiver receiver;
        private byte[] deviceId = new byte[8];
        private String deviceIdStr = "";
        private int nodeCount;
        private boolean mPowerStatus ;
        private final String VALUE = "value";
        String power_status, power_tips;

        @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
          }
    @Override protected void onResume() {
           super.onResume();
           //-----------------------------------------------------
          //  准备接收来自myService/myBinder的广播信息
           receiver = new myReceiver();
          IntentFilter filter = new IntentFilter();
                filter.addAction("android.intent.action.MY_BROADCAST");
                registerReceiver(receiver, filter);
                super.onResume();
            //-----------------------------------------------------
            initNative();    //初始化JNI本地函数
            getStatus = true;
            startGetStatus();
    }
    @Override protected void onPause() {
           super.onPause();
          uninitNative();
           getStatus = false;
    }
// ------------------------------------------------
    private boolean getStatus;
    private void startGetStatus(){
          new Thread(){
            public void run() {
                  while (getStatus) {
                        getStatusNative();    //调用本地函数
                        try {
                              sleep(DelayMillis);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        }
            };
       }.start();
    }
    // ----------- Callback from JNI  本地函数  -------------
  private void reportNodeId(int index, byte id0,
            byte id1, byte id2, byte id3, byte id4, byte id5, byte id6, byte id7)
       {
               deviceId = setByteArray(id0, id1, id2, id3, id4, id5, id6, id7);
                deviceIdStr = bytes2HexString(deviceId);
        }
  private void reportPowerStatus(byte id0, byte id1, byte id2, byte id3, byte id4, byte id5,
                                      byte id6, byte id7, int SocketNum , int SocketStatus)
        {
            String power_status = SocketStatus == 1 ? "Close" : "OPEN";
            String power_tips = "Power"+ SocketNum + " Status is " + status;
            // (SocketNum == 1)  代表  POWERSTATUS
            // (SocketNum == 0x81)  代表  POWERLEDSTATUS))
         }
```

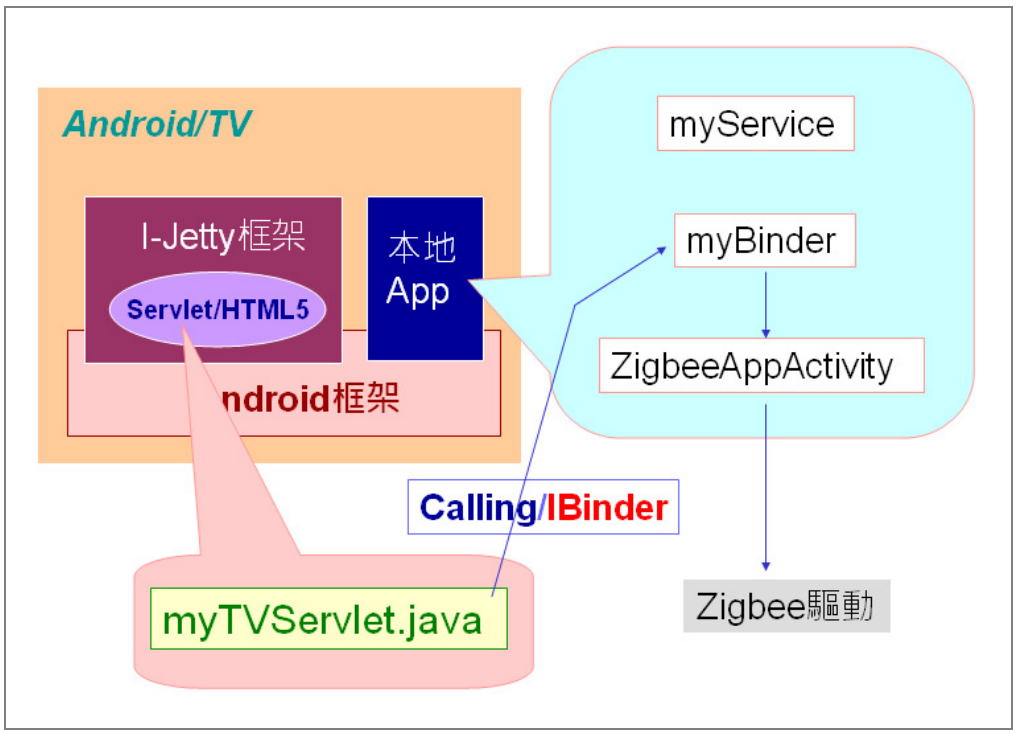
```
  //-----------------------------------------------------------------------------------------//
  public void SetPowerStatus( int status){
   switch(status){
                case 0:
               SetPowerStatusNative(deviceId, 1, 1);
               break;
                case 1:
               SetPowerStatusNative(deviceId, 1, 0);
               break;
                default:
               break;
           }
    }
// ----------------------------------------------------------------------------------
  static{
       System.loadLibrary("zigbee");
  }
//----------------------------------------------------------------------------------
// ---------------  宣告JNI本地函数  ------------------------------------------
private native int initNative();
private native int getStatusNative();
private native int uninitNative();
private native int resetNative();
private native int reconnetNative();
private native int getNodeCountNative();
private native int getNodeIdNative(int index);
private native int SetSwitchStatusNative(byte[] DeviceId, int num, int stauts);
private native int GetSwitchStatusNative(byte[] DeviceId, int num);
private native int SetBrightnessNative(byte[] DeviceId, int num, int vol);
private native int GetBrightnessNative(byte[] DeviceId, int num);
private native int SetPowerStatusNative(byte[] DeviceId, int num, int stauts);
private native int GetPowerStatusNative(byte[] DeviceId, int num);
private native int IRStudyNative(byte[] DeviceId, int keynum);
private native int IRSendNative(byte[] DeviceId, int keynum);
private native int GetWindowStatusNative(byte[] DeviceId);
private native int GetInputStatusNative(byte[] DeviceId, int num);
private native int SetOutputStatusNative(byte[] DeviceId, int num, int stauts);
private native int GetOutputStatusNative(byte[] DeviceId, int num);
private native int SetCurtainStatusNative(byte[] DeviceId, int status);
private native int GetCurtainStatusNative(byte[] DeviceId);
private native int GetTempHumiStatusNative(byte[] DeviceId);
// ----------------------------------------------------------------------------
private byte[] setByteArray(byte id0, byte id1, byte id2,
          byte id3, byte id4, byte id5, byte id6, byte id7){
   byte[] resultByte = new byte[8];
   resultByte[0] = id0;          resultByte[1] = id1;          resultByte[2] = id2;
   resultByte[3] = id3;          resultByte[4] = id4;          resultByte[5] = id5;
   resultByte[6] = id6;          resultByte[7] = id7;          return resultByte;
  }
private String bytes2HexString(byte[] bArray) {
          StringBuffer sb = new StringBuffer(bArray.length);
          String sTemp;
          for (int i = bArray.length - 1; i >= 0; i--) {
                sTemp = Integer.toHexString(0xFF & bArray[i]);
                if (sTemp.length() < 2)       sb.append(0);
                sb.append(sTemp.toUpperCase());
           }
```

```
            return sb.toString();
        }
    // ---------------------------------------------------------------------
    class myReceiver extends BroadcastReceiver {
        @Override public void onReceive(Context context, Intent intent) {
              int tempInt;
               tempInt =    intent.getIntExtra("inta", 101);
               if(tempInt != -1)
                      SetPowerStatus(tempInt);    // 设定开关状态
               }
       }
}
```

开发了 ZigbeeAppActivity 类之后，myTVServlet 就能透过 Context 通用性接口来绑定(Bind)这 Service 了。



绑定了 Service 之后，就能透过 IBinder 通用性接口，来与 myBinder 沟通，进而与 ZigbeeAppActivity 沟通了。



6. 結語

综上所述，由 i-Jetty 提供 Servlet 接口来与 iPhone 手机上的浏览器对接。在 i_Jetty 里，转换成为<doGet, doPost>接口，再经由<T>来使用 Context 接口，调用它的 startService()函数，来取得 IBinder 接口，进而透过 IBinder 接口与 ZigbeeAppActivity&硬件驱动互相通信。

藉由 3 个现成的通用性接口的联合应用，就完成了 iPhone 手机与 Android TV 两端的对接与整合任务了。◆