G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

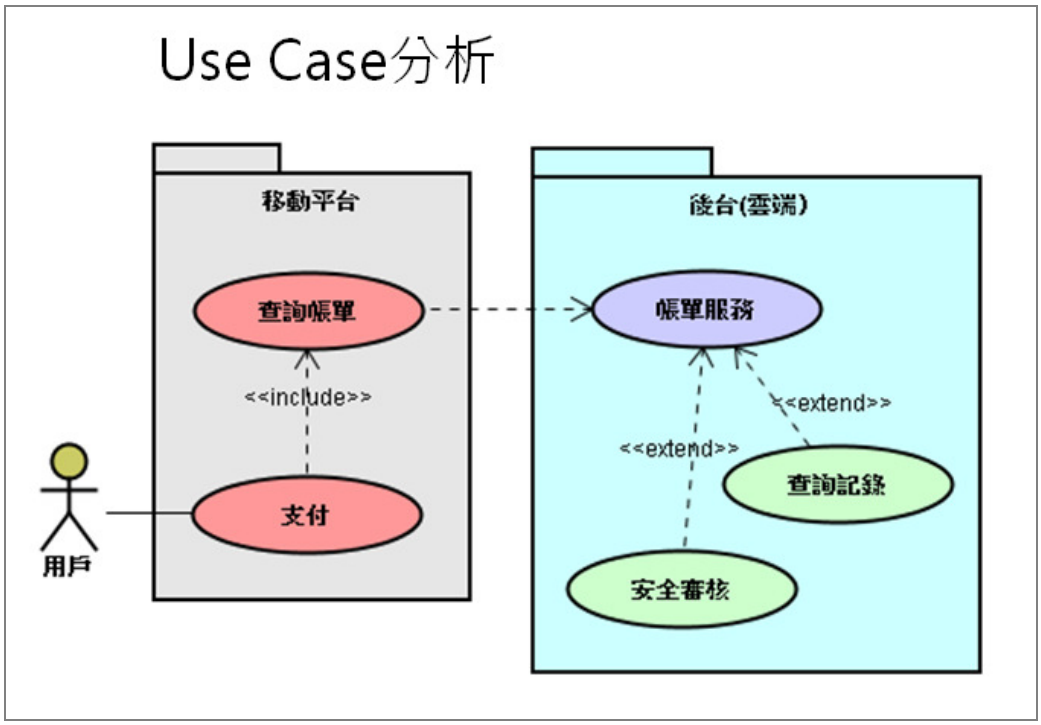
◇ Use Case 分析

前言：

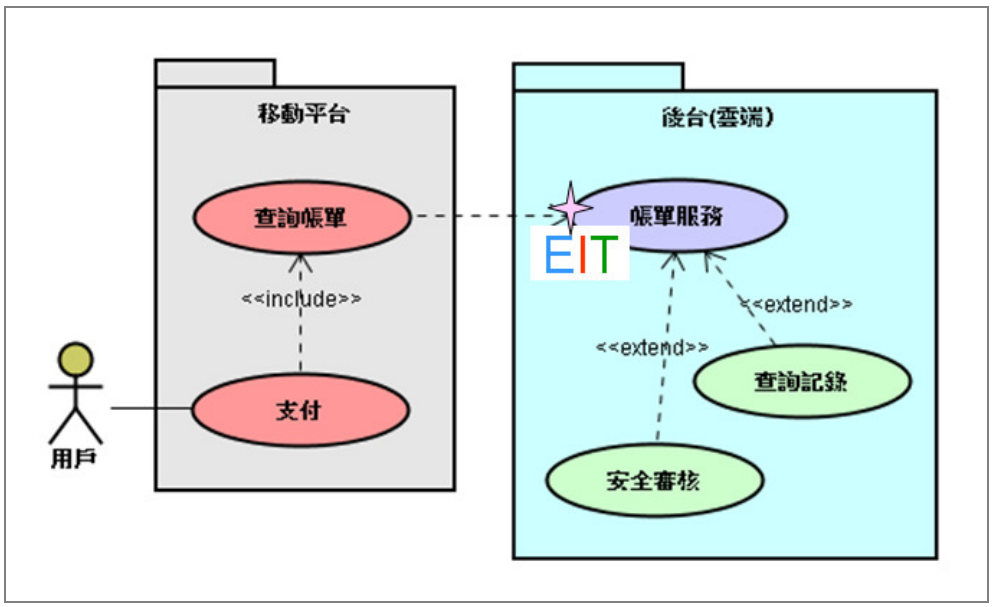
- 接口<I>是实的，代表一个须的空间，是空的，但用来容纳未来的变化花样。亦即，容纳改变（容易），让我们容易接纳、适应未来，更灵活、更具生命力。但是，这项虚实相依(或变与不变)，却又是另一个更大空间的内涵，这个更大空间称为”造形”。因之，架构师可学习做双层级抽象。
- <<双层级抽象>>例如你天天穿的鞋子，提供空间容纳不同袜子(或脚)的可变性，这是一层抽象。仅做到这层抽象是不够完美的。无论鞋子(相对上不变)或袜子(相对上可变)，其变与不变的虚实相依，都成为集装箱的内涵(Content)，于是将这复杂内涵塞入集装箱之"形"里，得到简单，才是架构设计的目标。
- 虽然从 EIT 代码造形来看，<E>、<I>和<T>三者是同位阶的，但从架构师角度上，<I>属于主角，而<E>和<T>是配角。搭配两个配角，才能将<I>表述的完整而清晰。

1. 范例(一)：Use Case 分析

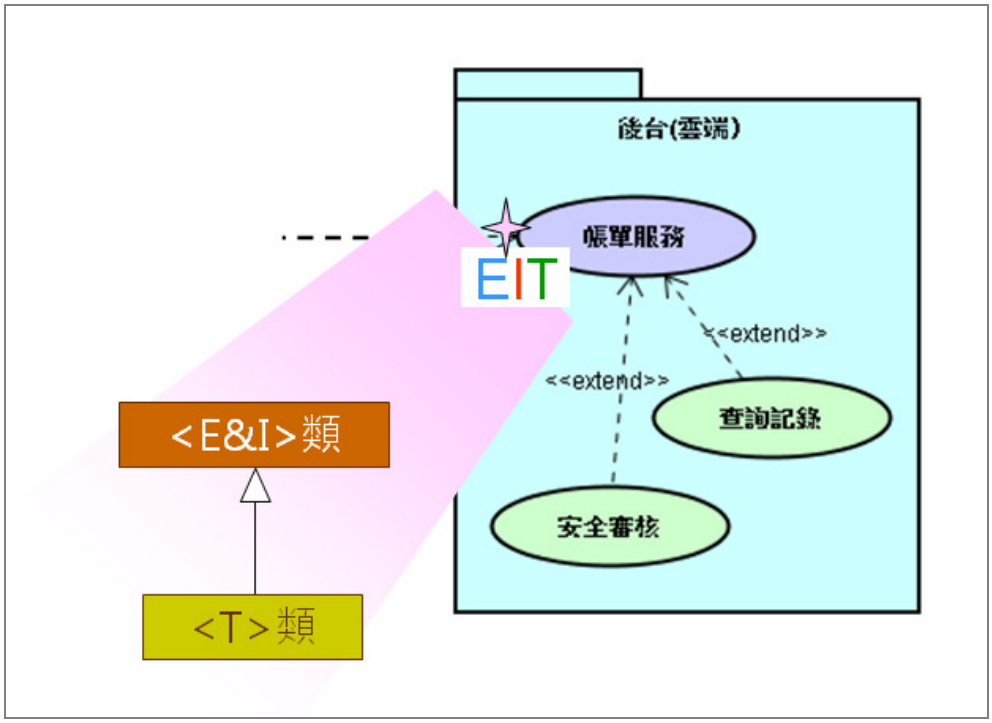
用例(Use Case)模型是以用例为基本概念来抽象描述一个系统。一个用例表达了使用者对系统的一项需求，也就是系统的一项责任(Responsibility)或功能(Function)。由于用例代表一项需求，经常成为开发的检验和交付的单元。



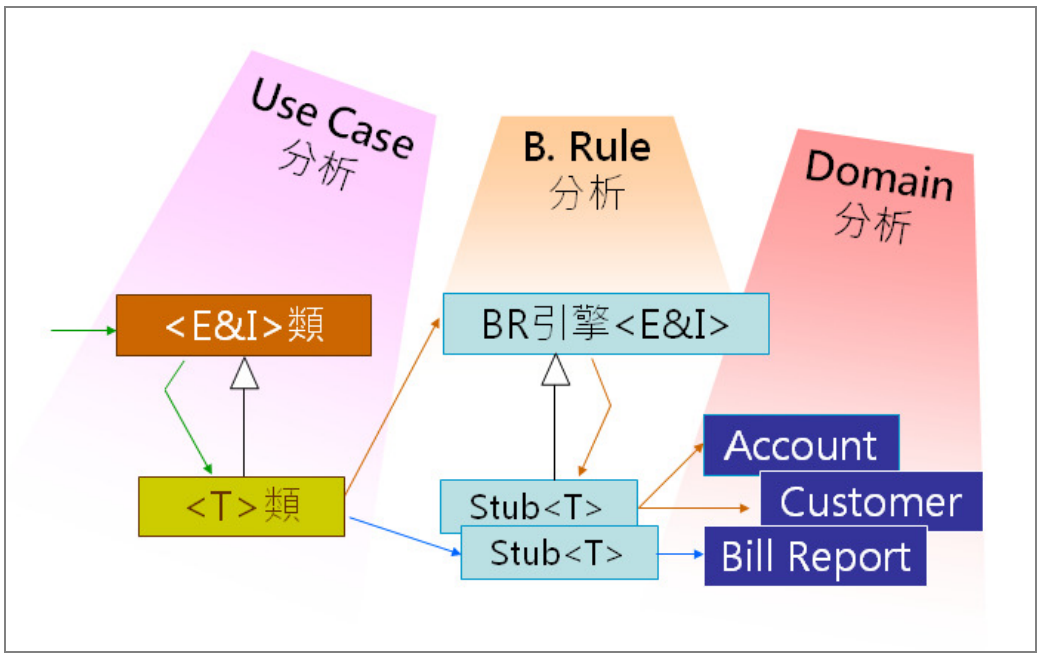
用例除了涵盖给用户使用的 UI 之外；也涵盖跨系统或跨模块(Module)的接口。此时，使用 EIT 造形来表述这种用例接口是最恰当的了。



架构师决定了<I>，等于将<E>和<T>两者”分解”开来，则开发者就能兵分二路，独立开发、迅速落实为代码。



一旦以 EIT 造形来表述这个用例接口之后，这个用例就能单独开发了，例如继续相关的企业规则(Business Rule)分析、领域(Domain)分析和细节设计了。



这可以迅速落实为代码，大力支持测试、迭代和交付。☆

G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

◇ iPhone 手机访问 Android 智能电视

前言：

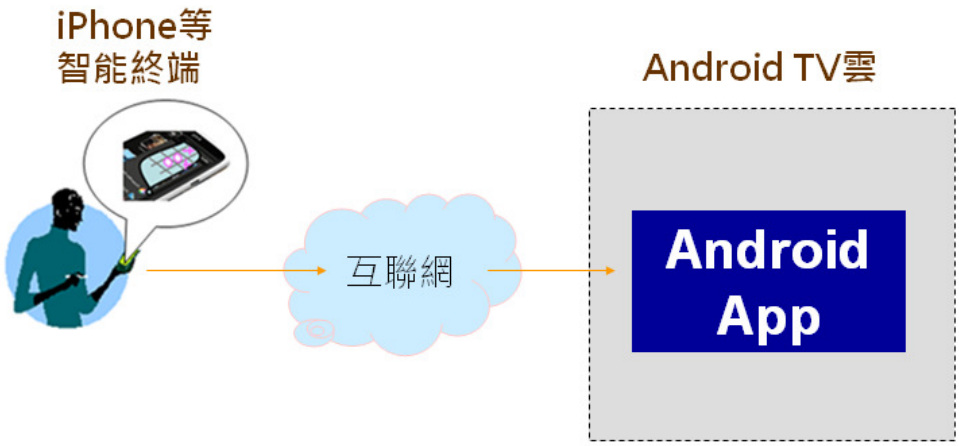
- EIT 造形的主角是接口<I>，只要架构师的眼中或心中有接口，就能以 EIT 造形来清晰表示出来。
- 这<I>就成为分解、重构系统的接口，或开发团队分工的界线。因此 EIT 造型创造了敏捷重构的自由度。
- 接下来，就请看看如何实践 EIT 造形？也就是将接口(透过 EIT 造形)落实为代码喔。
- 上个范例展示了 EIT 造形用来”分解”系统；下一个范例则展示如何”组合”系统，将 iPhone 手机与 Android 智能 TV 组合起来。

2. 范例(二)：iPhone 手机访问 Android 智能电视

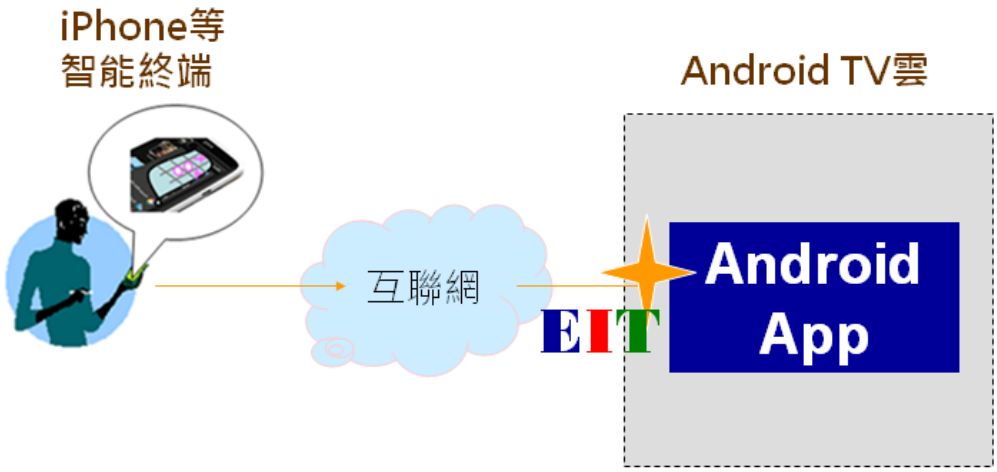
架构设计

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。

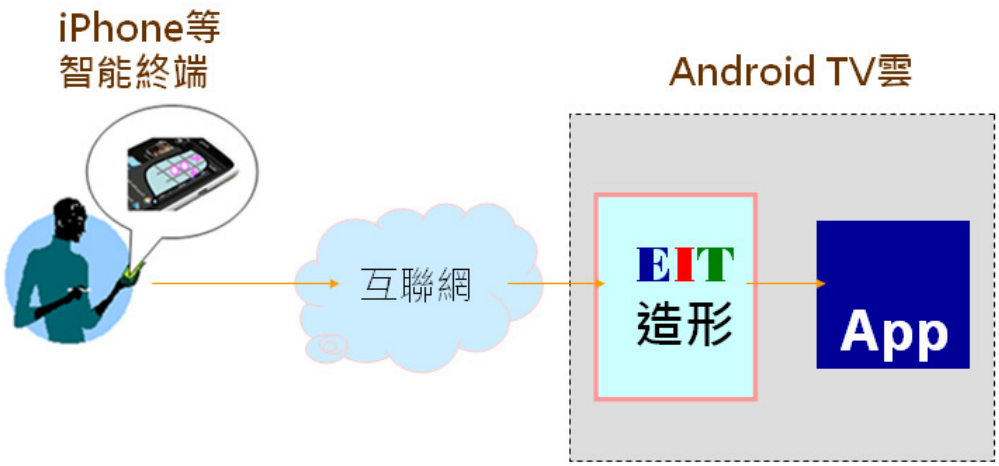
用户手机上跑浏览器(Browser)，用户上网访问会议厅里 Android 智能电视。Android 智能电视里有 App 软件和相关的 Zigbee 设备驱动，能控制会议厅里的主灯和电视墙。亦即，手机的浏览器软件要与 Android 框架通信，进而调用到其 App 软件。



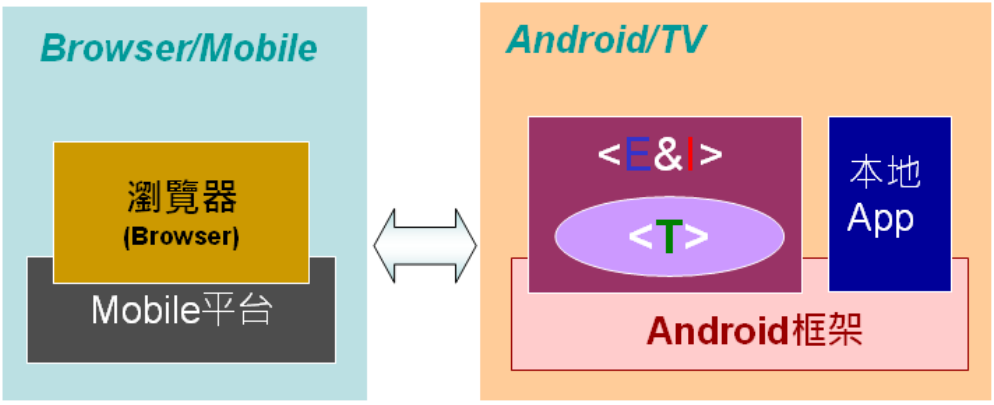
由于浏览器和 Android 框架都是现存的，其接口也都已经定义了。那么，该如何将两者”组合”起来呢？答案是：运用 EIT 造形。



<E>可以与手机的浏览器对接；而<T>可以与 Android 框架对接。



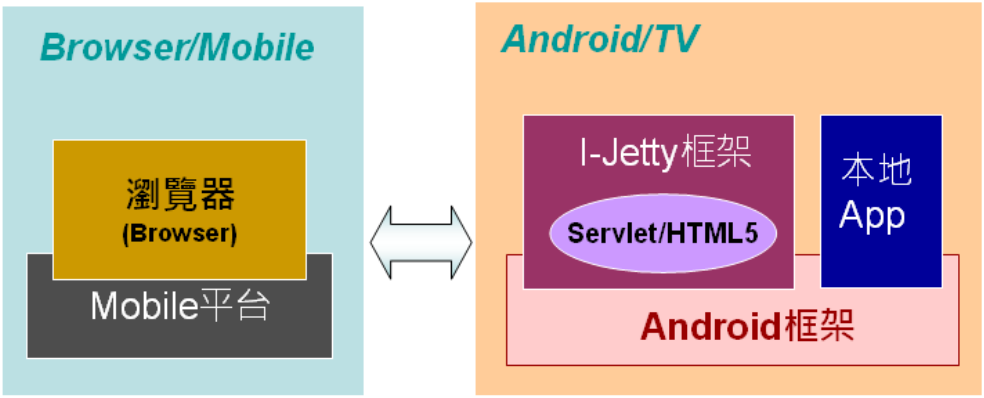
<E>接受来自手机浏览器的信息(如标准 HTTP 信息格式)，透过<I>接口来调用<T>，而<T>则将信息转化为 App 可理解的信息格式。



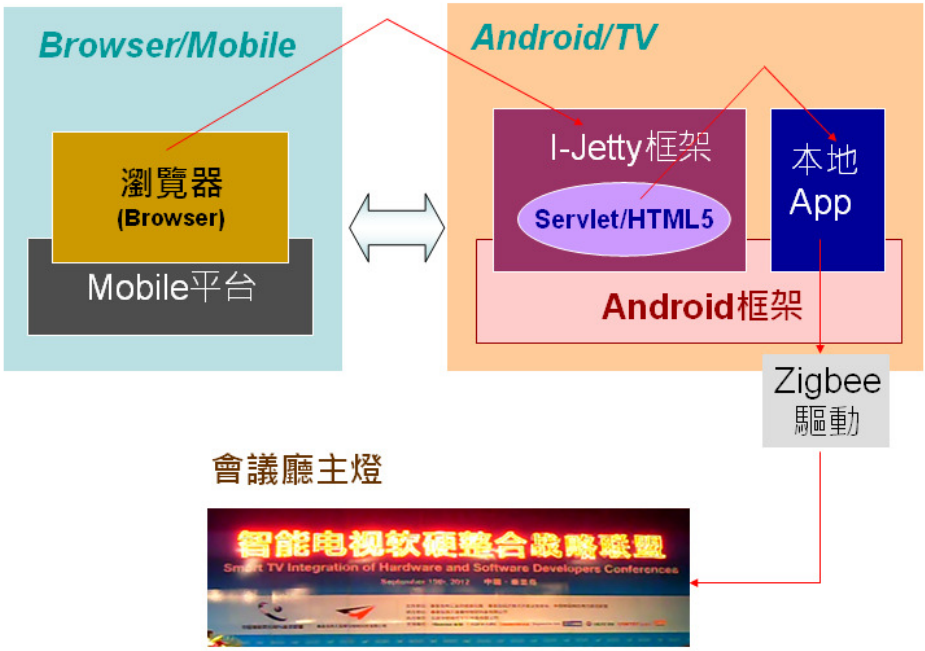
当我们心怀 EIT 造形时，就常常会发现身边早已存在形形色色的 EIT 造形了，不一定都要自己来开发 EIT 造形的代码，例如，想如上图的规划来整合 iPhone 手机与 Android 电视；就会发现 i-Jetty 是一个现成的 EIT 造形的代码。

- 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。

于是，在 Android 电视里，加入一个 i-Jetty 框架，扮演<E&I>的角色，再由开发者来撰写 Servlet 程序来扮演<T>角色。



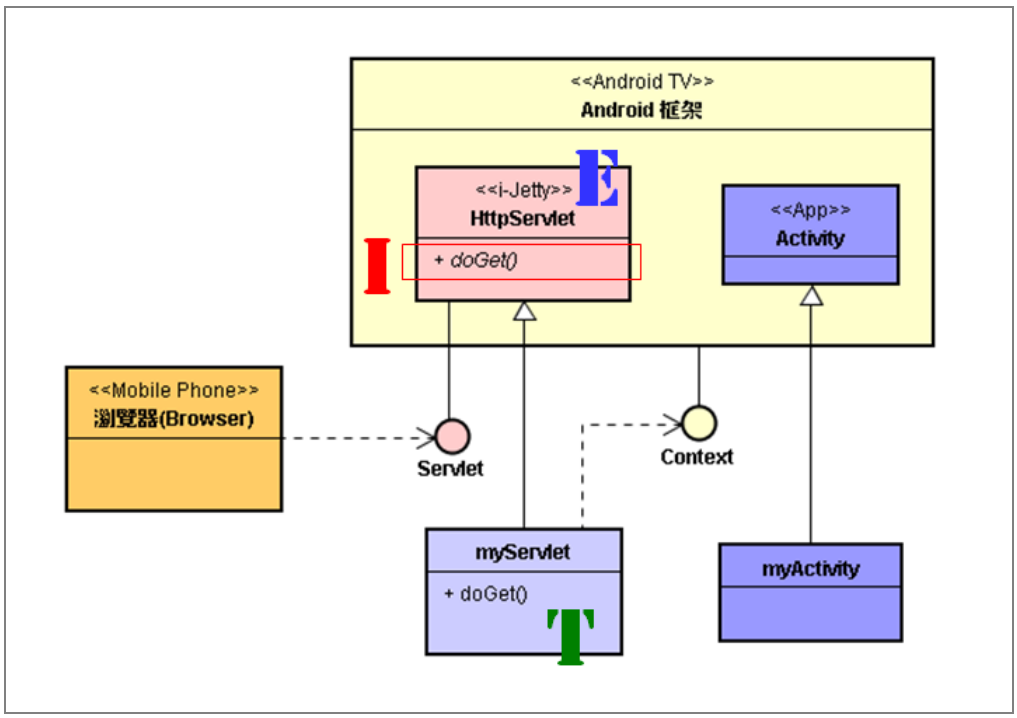
其通信的过程就如下图所示。



秦皇岛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。

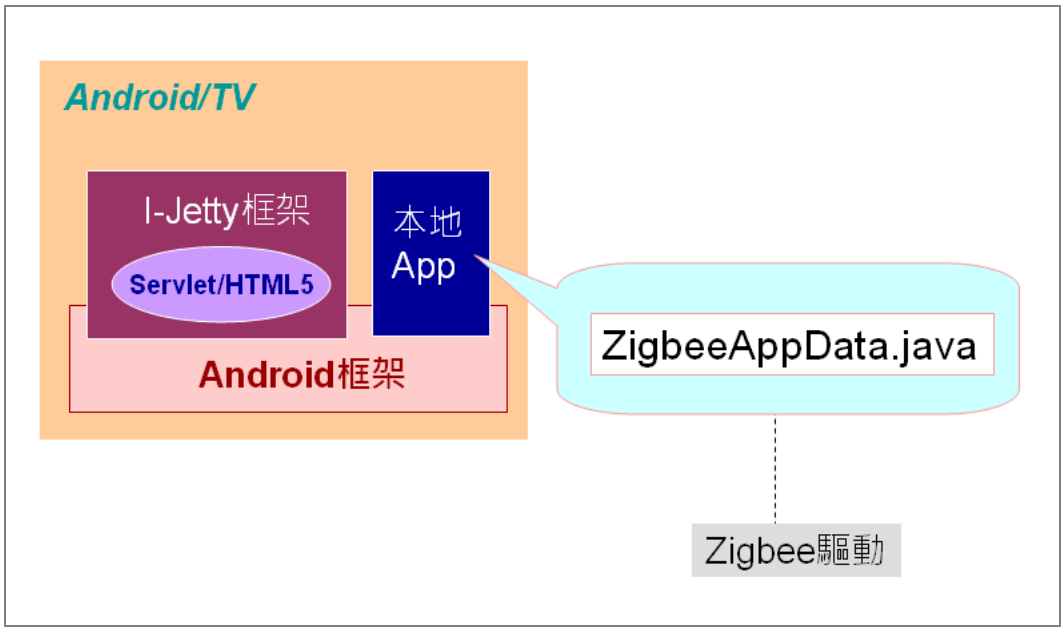
详细接口<I>设计

这 EIT 造形的<E>提供 Servlet 接口来与手机浏览器来对接。这 EIT 造形的<T>使用 Android 的 Context 接口来与 Android 框架通信。

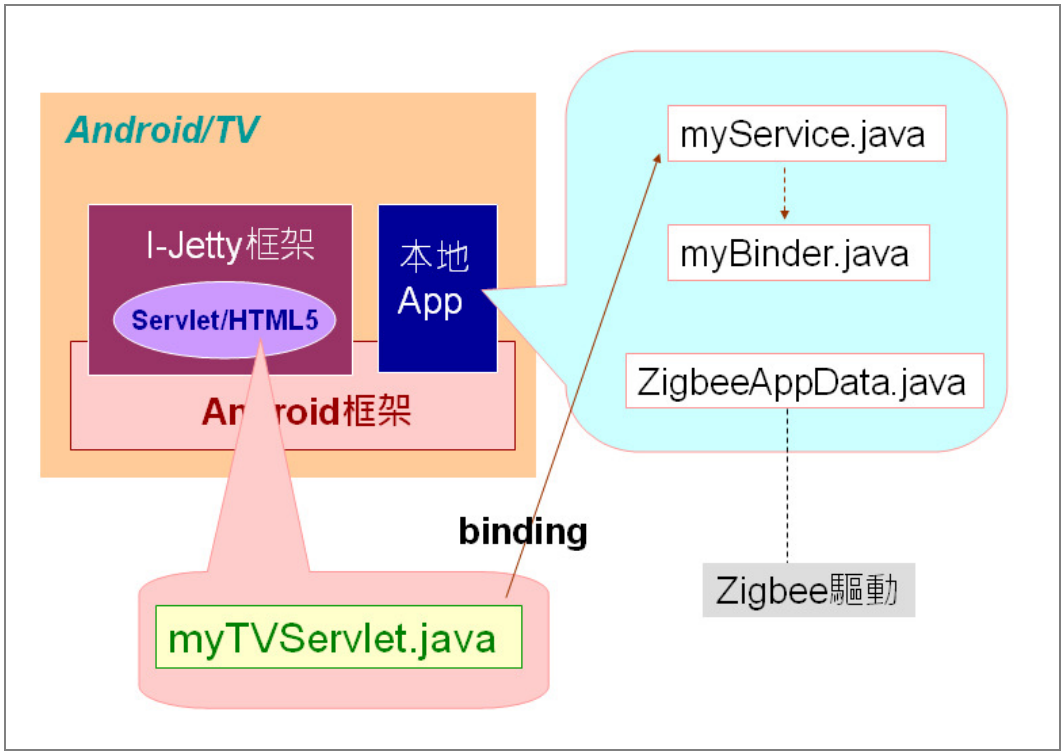


迅速落实为代码

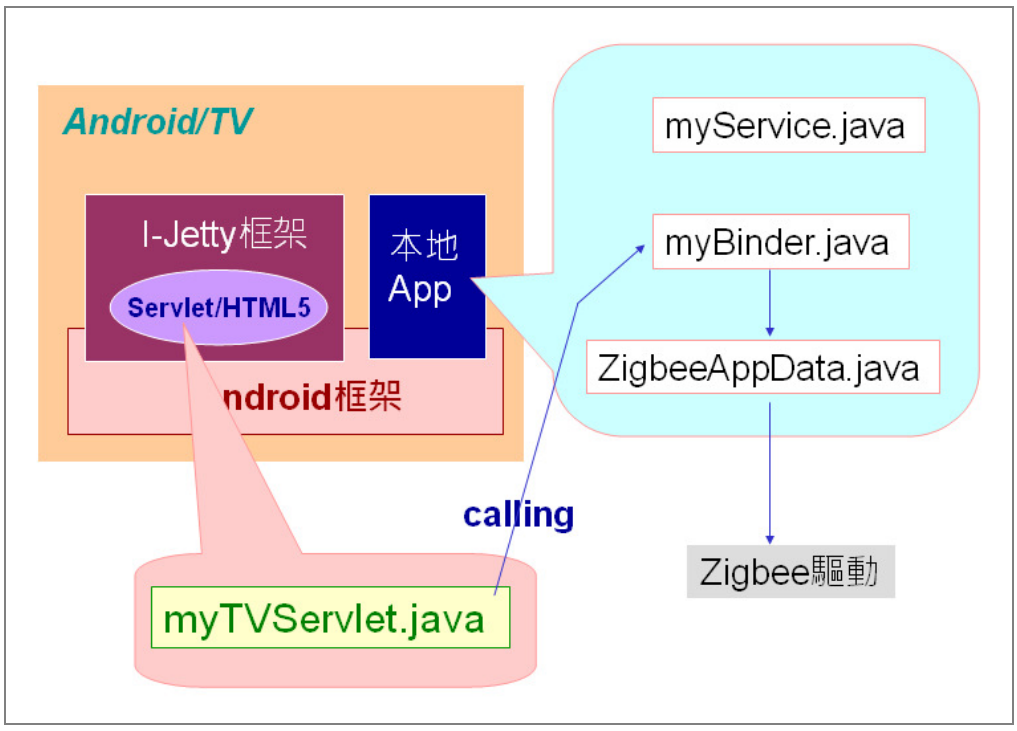
已经存在 Android App 和驱动软件，如下：



由开发者来撰写 Servlet 代码，透过 Context 接口来绑定(Bind)了 Android 的 Service 服务。如下：



绑定了 Service 之后，就能调用到 Android App 和驱动软件，如下：



代码范例

```
// myBinder 类
package com.google.android.ZigbeeApp;
import android.content.Context;
import android.os.Binder;
import android.os.Parcel;

public class myBinder extends Binder {
        private ZigbeeAppData mZigbeeAppData ;
        private Context context ;
        public myBinder(Context ctx){
                context = ctx ;
                mZigbeeAppData = new ZigbeeAppData();
        }
        @Override
           public boolean onTransact( int code, Parcel data, Parcel reply, int flags)
                   throws android.os.RemoteException {
                if (code == 1)
                        reply.writeString(
                                mZigbeeAppData.getAppData( data.readString()) );
                else if (code == 2)
                  {     String [] strings = {};
                        data.readStringArray(strings) ;
                         mZigbeeAppData.setAppdata(strings[0], strings[1]) ;
```

```
            }
            return true ;
    }}
```

```
// myService 类
package com.google.android.ZigbeeApp;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class myService extends Service{
        private IBinder mBinder = null ;
        @Override public void onCreate() {
                mBinder = new myBinder(getApplicationContext());
        }
        @Override public IBinder onBind(Intent intent){    return mBinder ;    }
}
```

```
// myTVServlet 类
package org.mortbay.ijetty;
import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
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;
import android.util.Log;

public class myTVServlet extends HttpServlet {
            private static final long serialVersionUID = 1L;
            private    String proofOfLife ;
            private myTVProxy 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.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 myTVProxy(ibinder);
                                }
                    public void onServiceDisconnected(
                                                    ComponentName className) {}
          };
          protected void doGet( HttpServletRequest request,
                                    HttpServletResponse response)
                        throws ServletException, IOException {
               response.setContentType("text/html");
               ServletOutputStream out = response.getOutputStream();
               out.println("<html>");
               out.println("<h1>TVZigbee Status From i-Jetty in FamilyCoud!
                                    RESULT is " + pProxy.getStatus(
                                    "Text_POWERSTATUS") + "</h1>");
               out.println("</html>");
               out.flush();
}
protected void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException { }
      private class myTVProxy    {
       private IBinder ib ;
       private String mStatus ;
       myTVProxy(IBinder ibinder)
                    { ib = ibinder ;}
       public String getStatus(String what) {
           Parcel pc = Parcel.obtain();
           Parcel pc_reply = Parcel.obtain();
           pc.writeString(what) ;
           try {
                  ib.transact(1, pc, pc_reply, 0);
                  mStatus =    pc_reply.readString();
           } catch (Exception e) {    e.printStackTrace();    }
           return mStatus ;
    }
   public void refreshStatus(String what, String value) {
           String[] strings = {"","On"};
           strings[0] = what;
           strings[1] = value;
           Parcel pc = Parcel.obtain();
           pc.writeStringArray(strings);
           Parcel pc_reply = Parcel.obtain();
           try { ib.transact(1, pc, pc_reply, 0);
           } catch (Exception e) {    e.printStackTrace();      }
}}}
```

G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

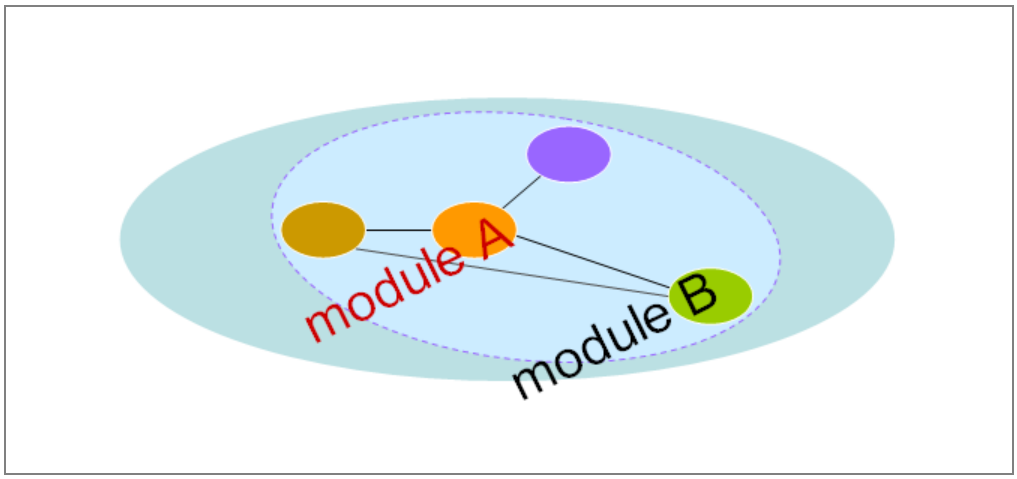
☆ 范例(三)：封装通信协议

前言：

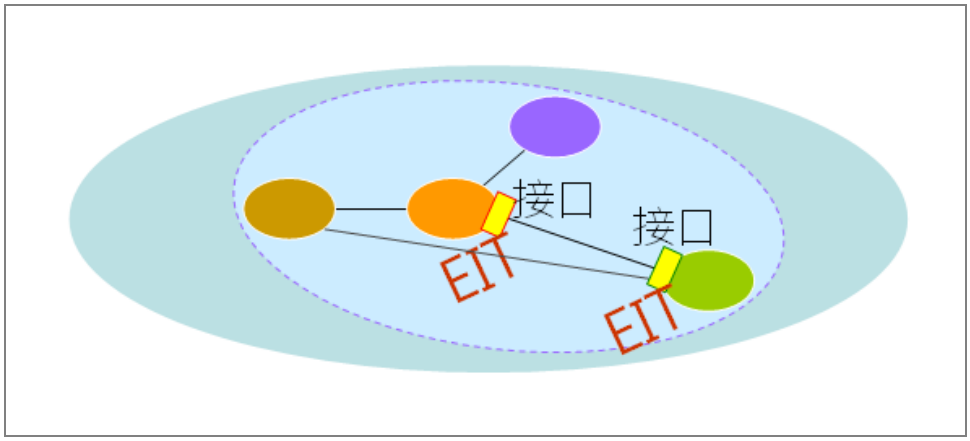
- 有人认为<E>是从<T>抽象出来，属于较为稳定不变的部分。这只是偏于”how-to”的视角。
- EIT 造形并无关于”how-to”。无论采取什么”how-to”，最后都以 EIT 造形表述出来。
- 也由于不局限特定”how-to”，所以代码造形不会局限人们的创意。就如同唐诗七言绝句的造形，并不会局限李白、杜甫、白居易大师人的创意。
- 接下来，就请看看<E>也能包容极为善变的通信协议和机制。

3. 范例(三)：封装通信协议

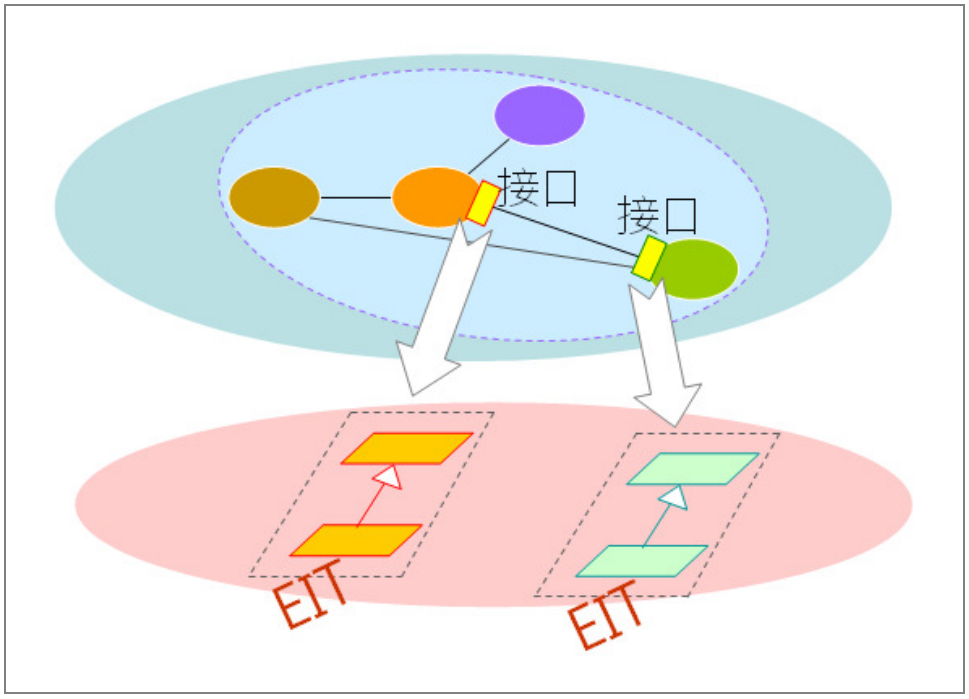
网络通信技术日新月异，通信协议是善变的。系统模块(Module)的设计与开发常常依赖于特定的通信协议，产生系统之间的高度偶合性(Highly-coupled)和相依性。



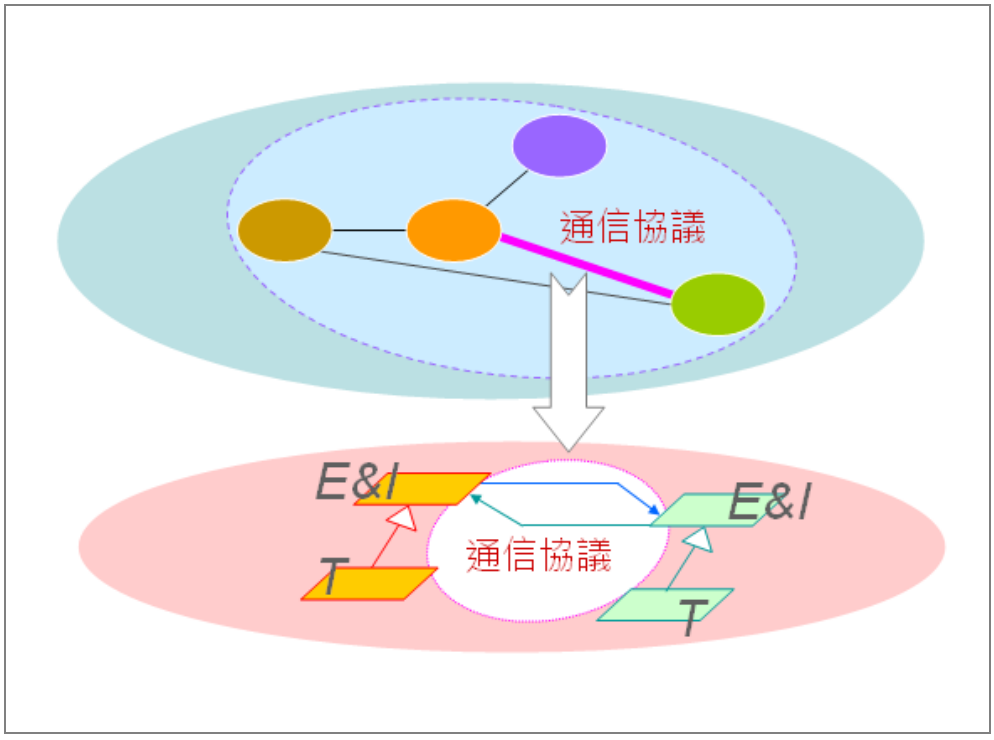
例如，上图里的模块”module A”与”module B”透过网络通信，如果两个模块分别由不同团队开发，通常两个团队会先会商，讨论两个模块之间的通信协议，确定了通信协议之后，才会各自展开其模块的细节开发工作。由于两个模块都依赖于特定的通信协议，一旦通信协议变动了，两个模块都会受到影响，也产生两个模块之间的高度偶合性和相依性。该如何面对这项问题呢？答案是：两边接口各设计一个 EIT 造形，如下图：



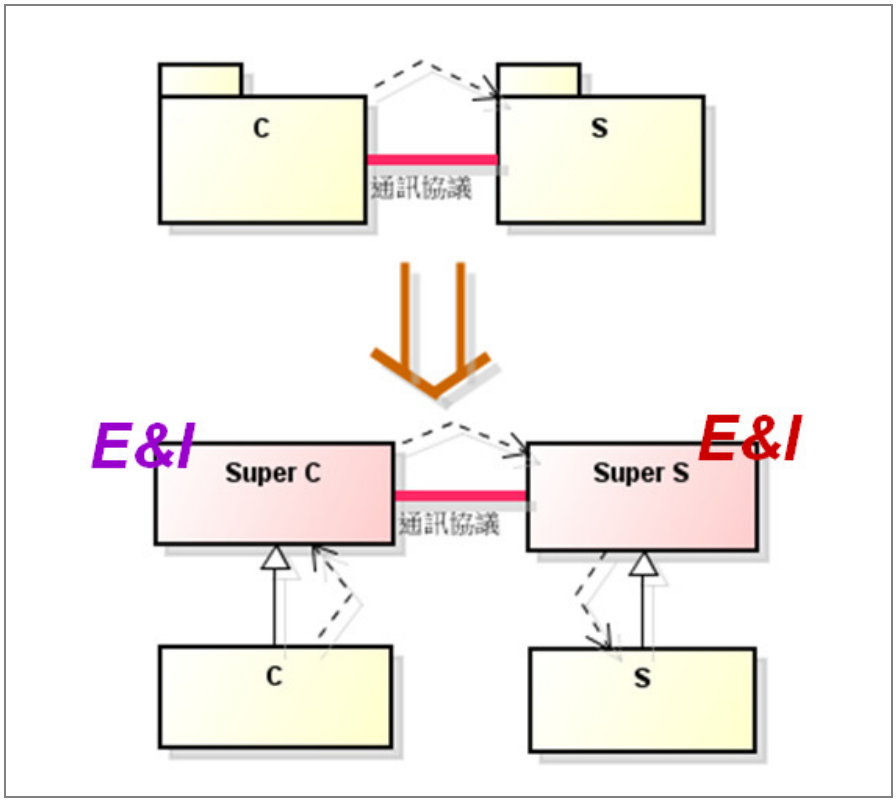
两个 EIT 造形的<T>，连接到各自模块的内部代码。



两个<E>则包装了通信协议；在还没开发模块内部代码之前，就能进行通信机制的检测。

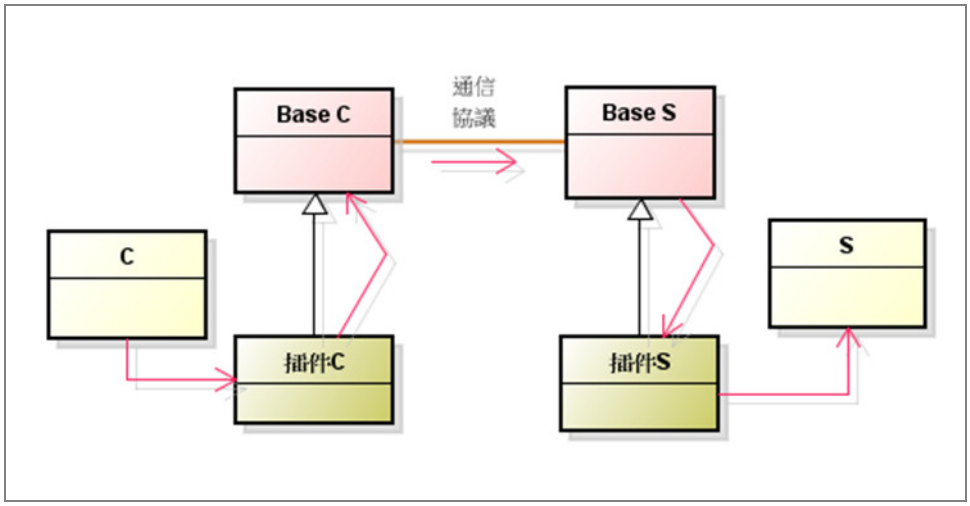


这是两 EIT 造形联合应用的典型范例。如下图，两个模块 C 和 S 是现有的，只是因为两者高度依赖于特定的通信协议。



一旦通信协议改变了，只需变更基类就可以了，并不必更动 C 和 S 模块的

内部代码。上述的”双 EIT 造形”架构，也常呈现如下图：



于是设计了两个基类(即<E&I>)，来包装通信机制和协议；并迅速落实为代码，立即检验通信机制和协议的可实现性。☆

G02_接口设计之美_7 个应用范例

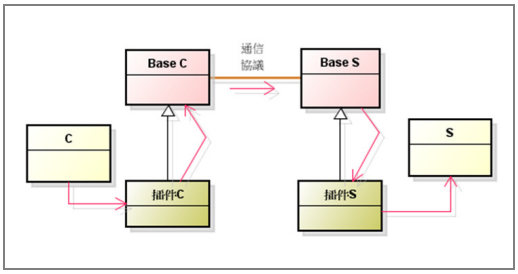
内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

☆ 范例(四)：维护底层或后台模块的变动自由度

前言：

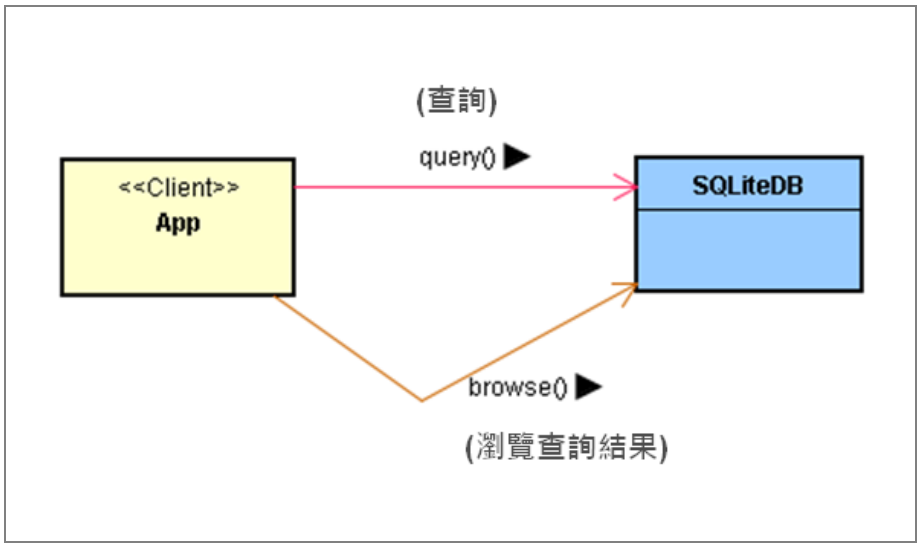
- 在上一个例子里，针对 Client-Server 架构，无中生有地创造了两个 EIT 造形，使用两个<E>来包容善变的通信协议和机制：



- 这保护了 Client 和 Server 模块的独立性和变动的自由度，创造了敏捷重构的空间。
- 这是架构师很常用的技能；例如，在 Android 平台里，也无中生有地创造了两个 EIT 造形来保护 SQLiteDB 引擎模块的变动自由度，创造了敏捷重构的空间，也替 SQLiteDB 等 Data Based 提供商带来<没钱就改版，改版就有钱>的商业机会。

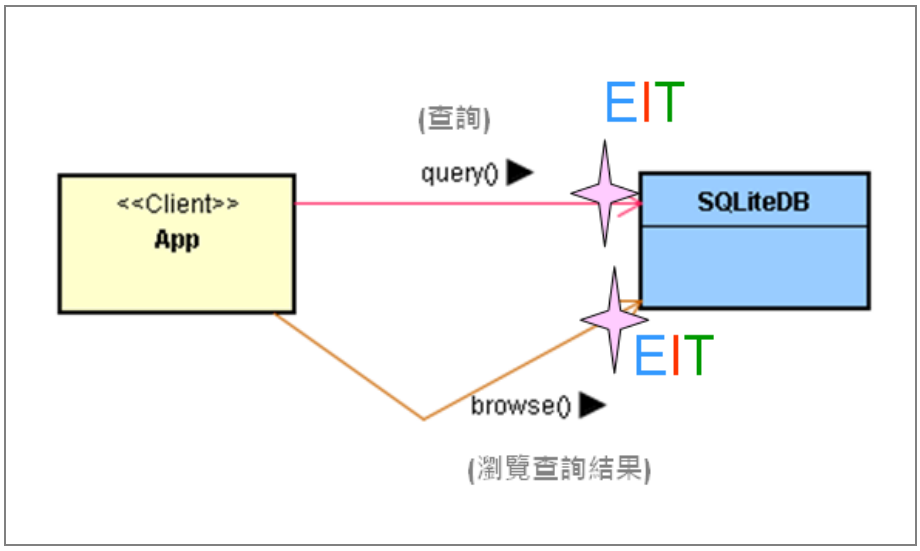
4. 范例(四)：维护底层或后台模块的变动自由度

保护底层和后端的软件模块，维护该模块的变动自由度是非常重要的。例如 SQLiteDB 数据库引擎模块的接口被写入于 Android App 代码里，因为受制于 App，导致该模块的变动自由度被局限了，因而减损了敏捷重构的自由度。

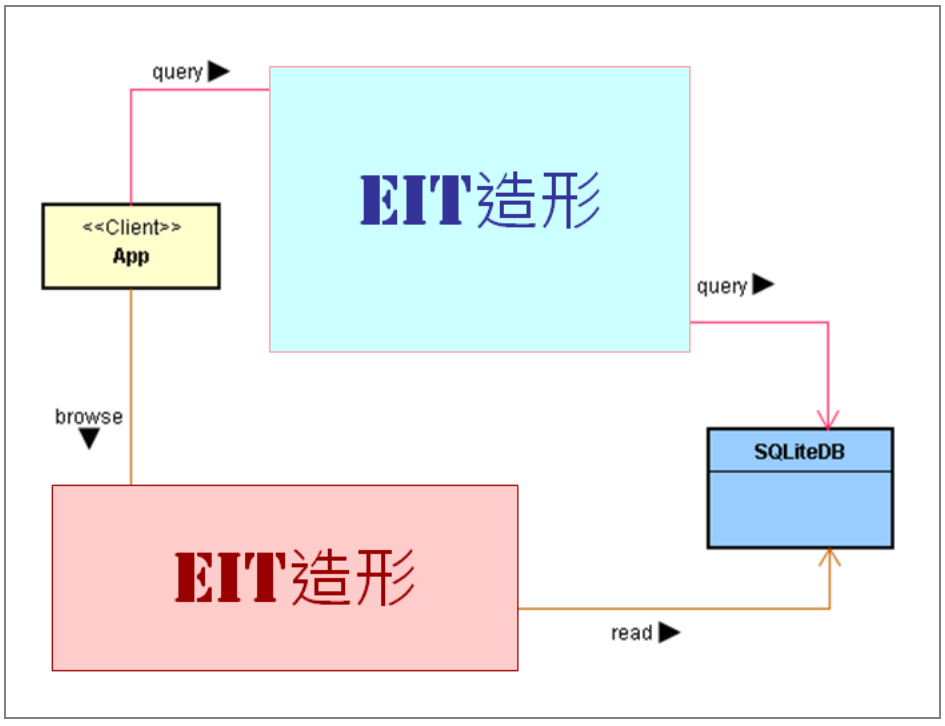


此时，可运用 EIT 造形来保护其接口。由于有两个接口：

- 查询接口：从 DB 查询数据，其结果放置于结果表格(Result Table)里。
- 浏览接口：浏览结果表格里的各笔数据。



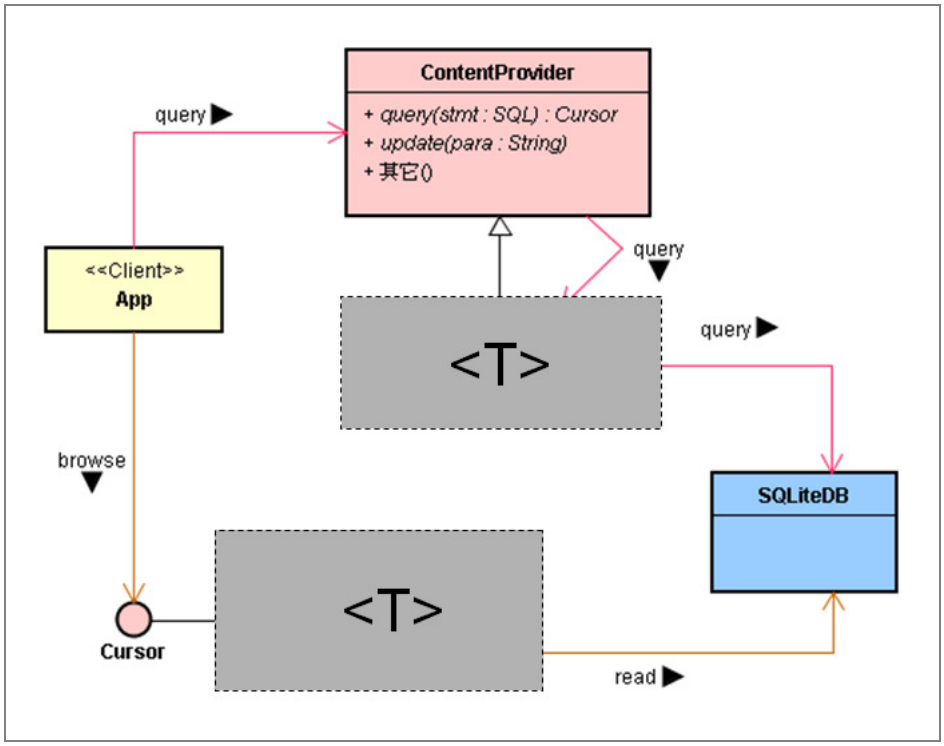
于是，设计了两个 EIT 造形，如下图：



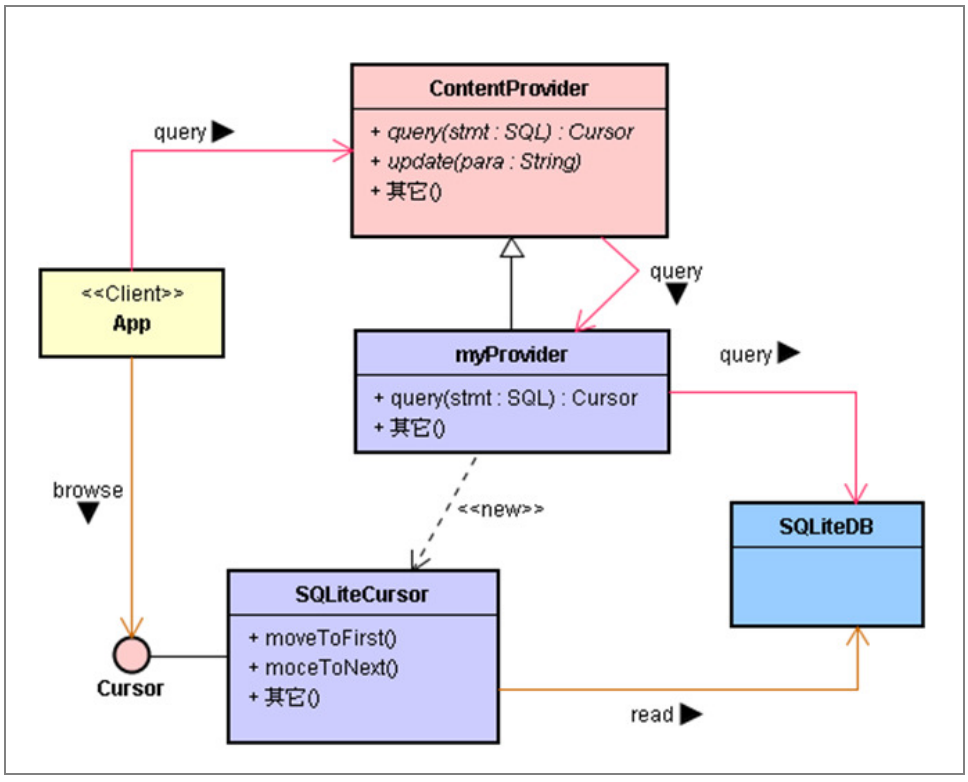
这两个 EIT 造形各提供一个新接口给 App 使用：

- 查询接口：兹设计一个基类 ContentProvider 来提供通用性的查询接口给 App 使用。

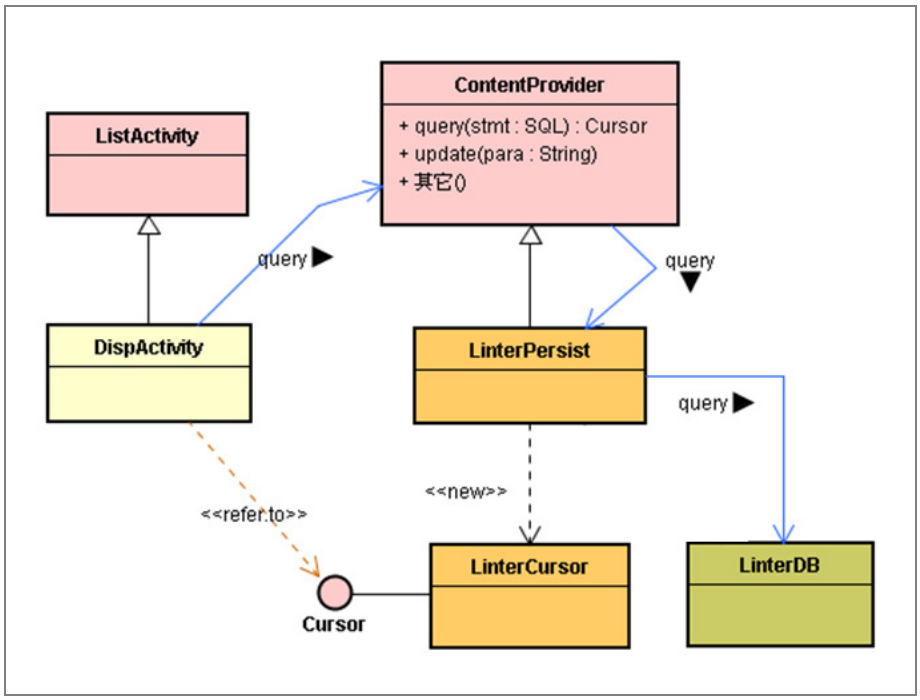
- 浏览接口：兹设计一个通用性接口 Cursor，来提供浏览接口给 App 使用。



其中的<T>则用来封装和保护 SQLiteDB 引擎的既有接口，如下的实际 EIT 造形设计：

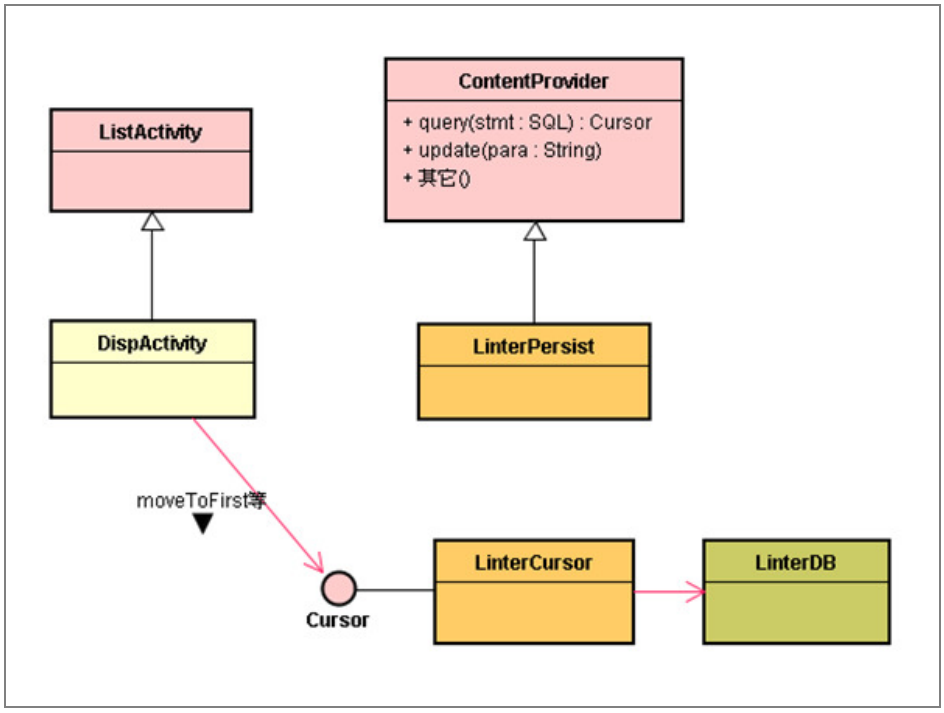


由于 SQLiteDB 引擎的接口不再受至于 App，则 SQLiteDB 引擎就很容易改版或抽换了。例如，我们很容易将 SQLiteDB 引擎更换成新的]LinterDB 引擎，如下图所示：



由于 ContentProvider 和 Cursor 提供的接口维持不变，在更换<T>和 DB 引擎时，并不影响 App。例如，上图的 DispActivity 仍然使用 ContentProvider 的通用性接口，来查询 DB 里的数据。由 LinterPersist 来实现查询任务，查询之后，就创建一个 LinterCursor 对象，并提供 Cursor 接口给 App(即 DispActivity)。

DispActivity 取得 Cursor 接口之后，就能调用 Cursor 接口的 moveToFirst()、moveToNext()等函数去浏览查询的结果了。如下图所示：



此范例展示了如何运用 EIT 造形来维护底层或后台(如 DB 引擎)模块的变动自由度；大幅促进模块的改版或抽换。

代码范例

```
// DataProvider.java
//……..
public class DataProvider extends ContentProvider {
        private static final String LINTER_TABLE_NAME = "Student123";
        private Connection con;
        @Override public boolean onCreate() {
            try {
                    Class.forName("com.relx.jdbc.LinterDriver").newInstance();
                    con = DriverManager.getConnection(
                       "jdbc:Linter:linapid:localhost:1070:local", "SYSTEM", "MANAGER");
             } catch(Exception e) { Log.e("Conn failed", e.toString());    return false;
            try{ Statement stmt = con.createStatement();
                  stmt.executeUpdate("drop table " + LINTER_TABLE_NAME);
            } catch (Exception e){ Log.e("drop table failed", e.toString()); }
            try {Statement stmt = con.createStatement();
                  stmt.executeUpdate(
                     "create table " + LINTER_TABLE_NAME + " (stud_no char(10),
                                  stud_name char(20));");
                     PreparedStatement prepstmt1 = con.prepareStatement(
                                "insert into " + LINTER_TABLE_NAME + " values (?,?);");
                    prepstmt1.setString(1, "linter_5");    prepstmt1.setString(2, "Lisa");
                    prepstmt1.executeUpdate();
                    prepstmt1.setString(1, "linter_8");    prepstmt1.setString(2, "Kitty");
                    prepstmt1.executeUpdate( );
              } catch (Exception e){ Log.e("create/insert table failed", e.toString());
                                        return false;    }
             return true;
        }
```

```
@Override public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
      ResultSet rs = null;
      try { Statement stmt = con.createStatement();
            rs = stmt.executeQuery("select * from " + LINTER_TABLE_NAME);
      } catch (Exception e) { e.printStackTrace();    return null; }
      Cursor c = new LinterCursor(rs, con);          return c;      }
 @Override public String getType(Uri uri) { return null; }
 @Override public Uri insert(Uri uri, ContentValues initialValues) {
          String field_1 = initialValues.get("stud_no").toString();
          String field_2 = initialValues.get("stud_name").toString();
          try{ PreparedStatement prepstmt1 = con.prepareStatement(
                        "insert into " + LINTER_TABLE_NAME + " values (?,?);");
               prepstmt1.setString(1, field_1);    prepstmt1.setString(2, field_2);
               prepstmt1.executeUpdate();
          } catch (Exception e) { Log.e("ERROR", e.toString()); }
          return uri;  }
   @Override public int delete(Uri uri, String where, String[] whereArgs) { return 0; }
   @Override public int update(Uri uri, ContentValues values, String where,
                                    String[] whereArgs) {    return 0;    }
  //-----------------   定义LinterCursor 类别 ----------------------------------
  class LinterCursor implements Cursor{
            ResultSet res;    Connection conn;
       LinterCursor(ResultSet rs, Connection con) { res = rs;    conn = con;    }
        @Override   public void close() {
              try {    res.close();    conn.close();
               } catch (java.sql.SQLException e) { e.printStackTrace();     }
        }
        @Override
        public void copyStringToBuffer(int columnIndex, CharArrayBuffer buffer) {}
        @Override public void deactivate() {}
        @Override public byte[] getBlob(int columnIndex) {    return null;    }
        @Override public int getColumnCount() {    return 0;    }
        @Override public int getColumnIndex(String columnName) {    return 0;       }
        @Override public int getColumnIndexOrThrow(String columnName)
                        throws IllegalArgumentException { return 0;    }
        @Override public String getColumnName(int columnIndex) {return null;       }
        @Override public String[] getColumnNames() { return null;         }
        @Override public int getCount() { return 0;      }
        @Override public double getDouble(int columnIndex) { return 0;    }
        @Override public Bundle getExtras() {    return null;    }
        @Override public float getFloat(int columnIndex) {   return 0;     }
        @Override public int getInt(int columnIndex) {    return 0;    }
        @Override public long getLong(int columnIndex) {   return 0;     }
        @Override public int getPosition() { return 0;   }
        @Override public short getShort(int columnIndex) { return 0;      }
        @Override public String getString(int columnIndex) {
              try {  return res.getString(columnIndex + 1);
              } catch (java.sql.SQLException e) { e.printStackTrace(); }
              return null;
        }
        @Override public boolean getWantsAllOnMoveCalls() { return false;    }
        @Override public boolean isAfterLast() {
              try { return res.isAfterLast();
              } catch (java.sql.SQLException e) { e.printStackTrace(); return false; }}
        @Override public boolean isBeforeFirst() {
              try {  return res.isBeforeFirst();
              } catch (java.sql.SQLException e) {    e.printStackTrace(); return false; }}
        @Override public boolean isClosed() { return false;    }
        @Override public boolean isFirst() { return false;       }
```

```
        @Override public boolean isLast() {return false;        }
        @Override public boolean isNull(int columnIndex) {    return false; }
        @Override public boolean move(int offset) {    return false;         }
        @Override public boolean moveToFirst() {
              try {  return res.first();
              } catch (java.sql.SQLException e) { e.printStackTrace(); return false; }}
        @Override public boolean moveToLast() {return false;       }
        @Override public boolean moveToNext() {
              try {  return res.next();
              } catch (java.sql.SQLException e) {e.printStackTrace(); return false; }}
        @Override public boolean moveToPosition(int position) {    return false;       }
        @Override public boolean moveToPrevious() {    return false;    }
        @Override public void registerContentObserver(ContentObserver observer) {}
        @Override public void registerDataSetObserver(DataSetObserver observer) {}
        @Override public boolean requery() {    return false;    }
        @Override public Bundle respond(Bundle extras) {    return null; }
        @Override public void setNotificationUri(ContentResolver cr, Uri uri) {}
        @Override public void unregisterContentObserver(ContentObserver observer) {}
        @Override public void unregisterDataSetObserver(DataSetObserver observer) {}
    }}
```

G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

☆ 范例(五)：软硬整合开发

☆ 微信(We Chat)平台整合家庭物联网

前言：

- 以上的例子里，都是纯软件的思维；其实，硬件世界也处处是接口，也处处可以对映到 EIT 造形。
- 于是，创造软件 EIT 造形来与硬件世界 EIT 造形互相辉映，带来了神奇的软硬整合效果，这是 Android 平台软硬整合架构师常用的技术。
- 此外，在网络世界里也处处是接口，也处处可以对映到 EIT 造形。例如，微信(WeChat)移动互联网与家庭 TV 客户端接口；再如家庭 TV 基于家庭物联网而与其它家庭智能设备的接口等。
- 于是，我们来看看如何建立双层 EIT 造形，来将微信移动互联网与家庭物联网，相互整合起来。

5. 范例(五)：软硬整合开发

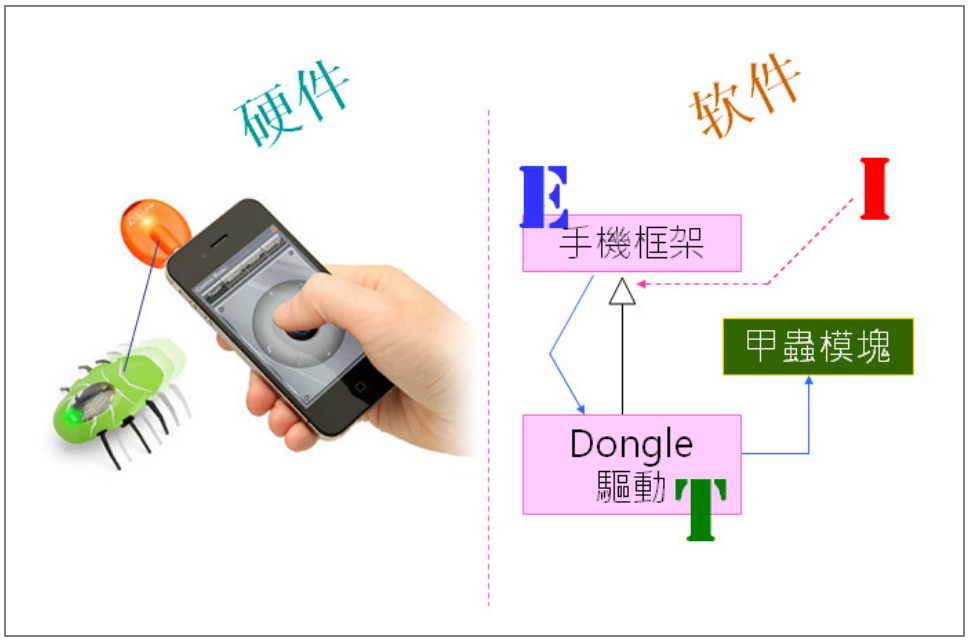
EIT 造形是软、硬件设计的共享性造形。例如，iPhone 手机要如何才能控制一只甲虫呢？如下图：

iPhone 手机扮演<E>角色，而其通用性的耳机接口则是<I>角色，如下图：

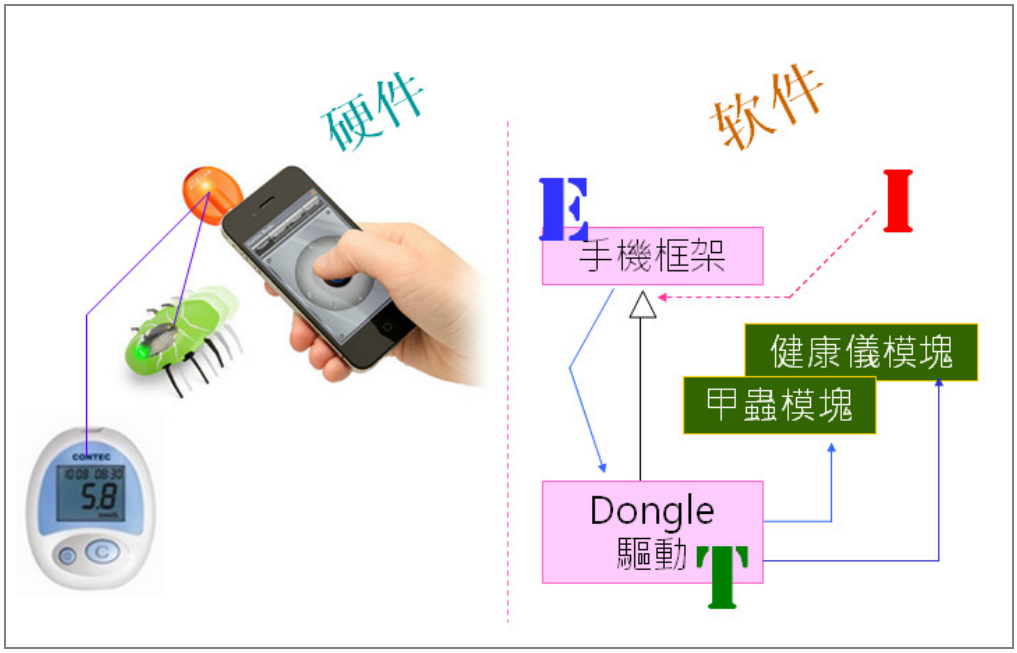
答案是：iPhone 手机提供通用性的耳机接口，只要插上檔口(Dongle)插件发射 Zigbee 信号，就能控制甲虫配件了。如下图所示：

以一般商业术语，手机是主件，耳机孔是接口，檔口(Dongle)是插件，而甲虫则是配件。

硬件 EIT 造形经常可对映到手机主件里的软件 EIT 造形。如下图：



硬件 EIT 造形与软件 EIT 造形的配对，能有效支持硬件配件的增加，如下图：



通常，手机侦测到新配件时，就会自动启动相对映的配件模块，来解析来自配件的信息。

G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

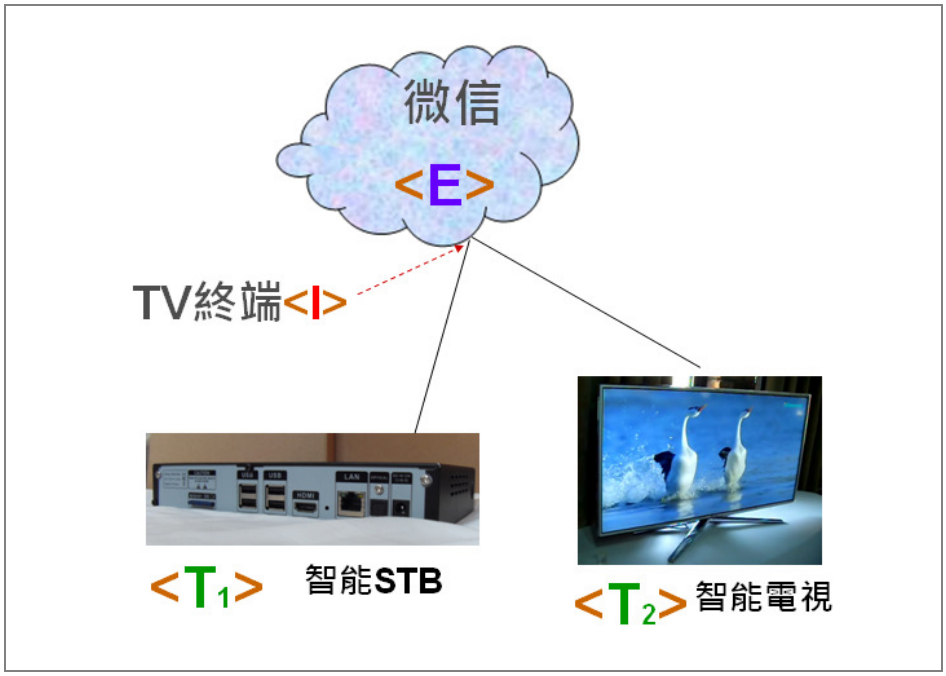
6. 范例(六)：微信(We Chat)平台整合家庭物联网

腾讯公司的微信(We Chat)平台已经拥有数亿个移动客户端。如下图：

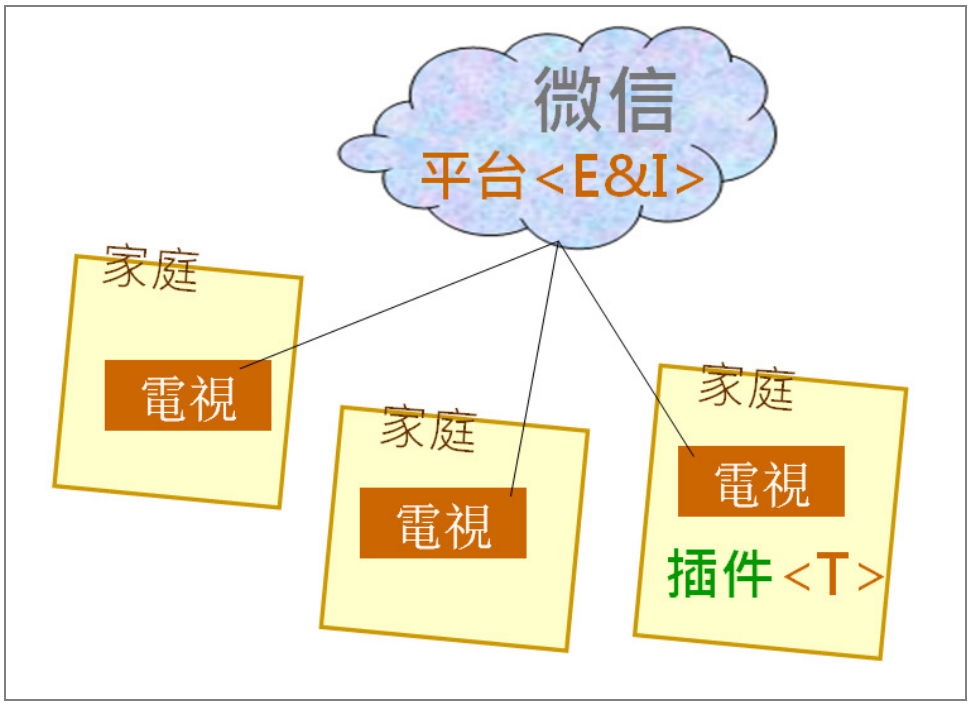
微信提供接口<I>来连结到数亿个移动终端设备。

于是，微信平台扮演<E&I>角色；而各移动终端都扮演<T>角色。

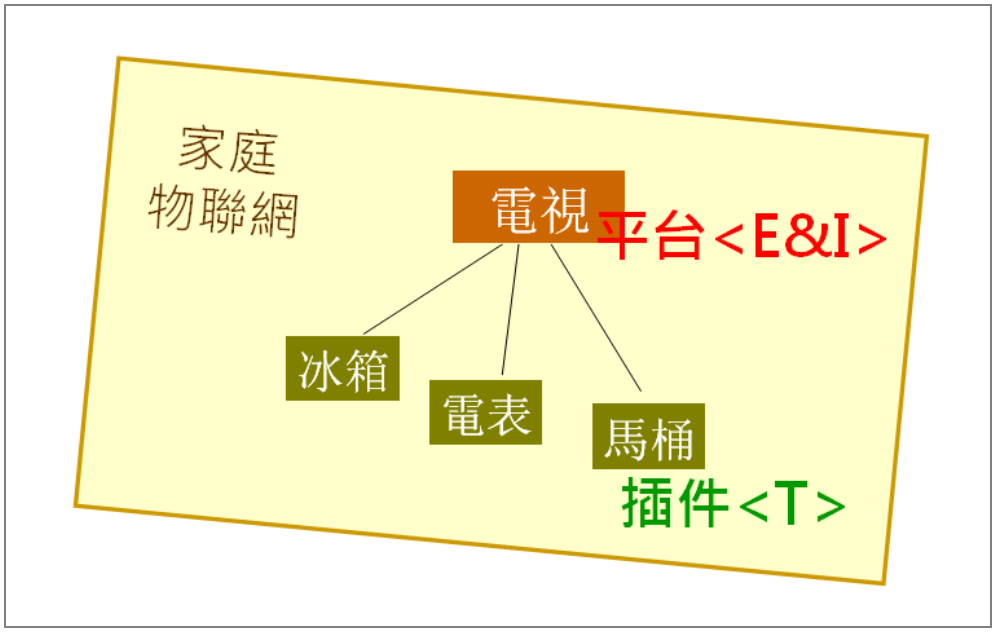
一个微信平台，其定义了接口<I>来连结到数亿个移动终端<T>。依样画葫芦，微信也能定义另一个<I>，来连结到家庭里的智能电视机(TV)或机顶盒(STB)。



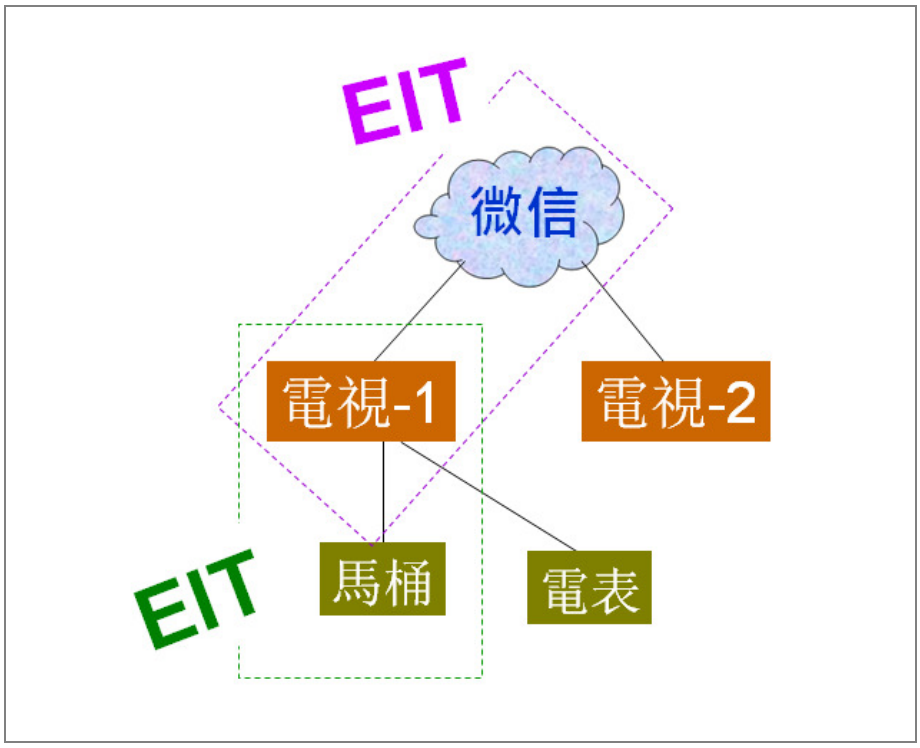
于是，微信平台扮演<E&I>角色；而各家庭里的智能电视(一种不移动终端)都扮演<T>角色。



以上的微信属于移动互联网，而智能电视则属于家庭物联网。藉由 EIT 造形来思考上述移动互联网与家庭物联网的整合。同样地，在家庭物联网里，智能电视也能扮演<E&I>角色；而家庭里的各项其它智能设备(如冰箱、电表、壁灯等)，则都扮演<T>角色。



于是，构成了两层 EIT 造形的树状结构体系。



这也意味着，是藉由移动互联网来整合家庭物联网。☆

G02_接口设计之美_7 个应用范例

内容：

1. 范例(一)：Use Case 分析
2. 范例(二)：iPhone 手机访问 Android 智能电视
3. 范例(三)：封装通信协议
4. 范例(四)：维护底层或后台模块的变动自由度
5. 范例(五)：软硬整合开发
6. 范例(六)：微信(We Chat)平台整合家庭物联网
7. 范例(七)：端&云整合的强龙策略

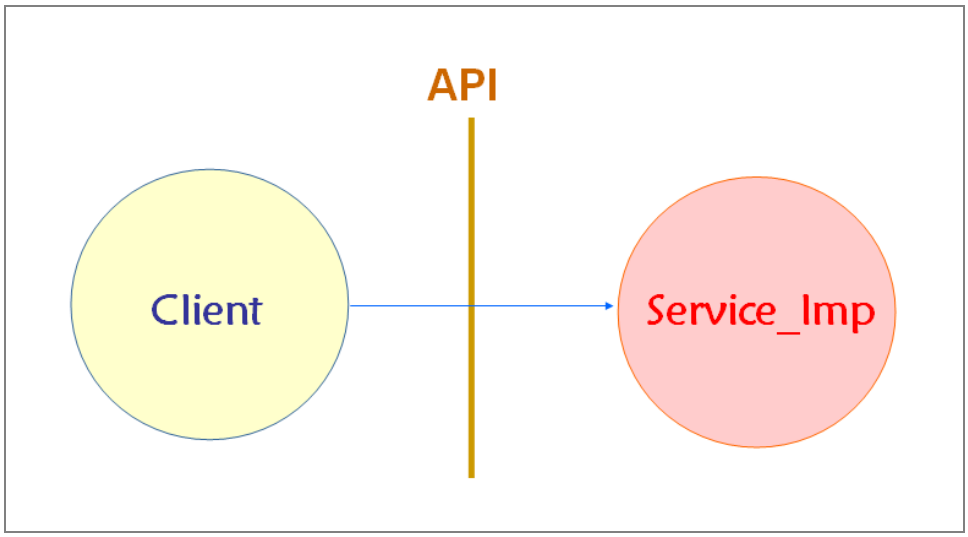
☆ 范例(七)：端&云整合的强龙策略(<I>就是话语权)

前言：

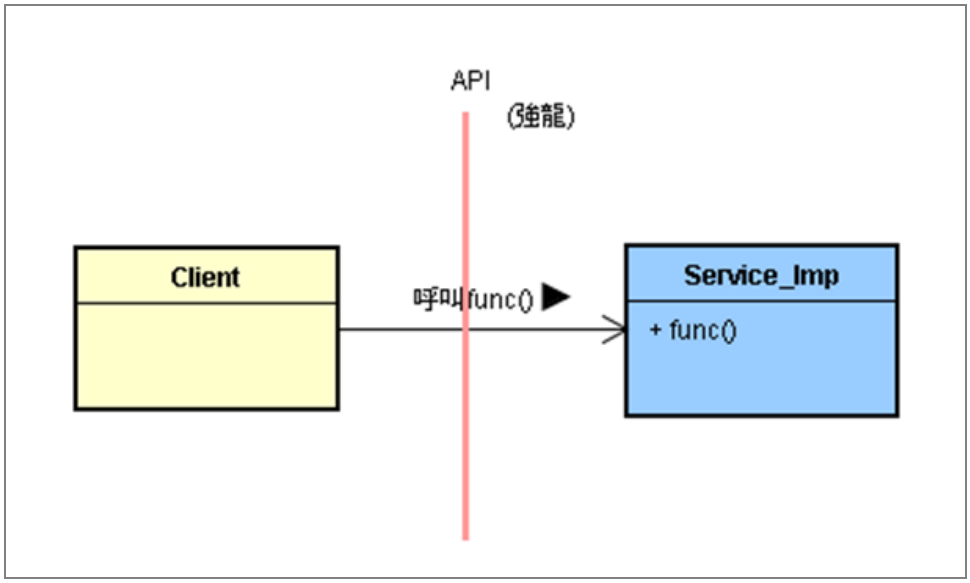
- 在以上的例子里，EIT 造形引导架构师去寻找”隐藏”的接口、设计接口(无中生有)和表述接口。
- 接下来，看看 EIT 造形的另一项重要用途：发挥<E&I>对<T>的控制力，让 Server 端(或云端)的大企业(如 Google)，除了掌控云平台之外，还能快速扩大地盘，掌控移动终端，成为横跨端云的强龙。
- 上述的寻找”隐藏”的接口、设计接口(无中生有)和表述接口，都不是终极目标，其最终目的是掌握<I>，取得整体架构(或系统)的”话语权”。

7. 范例(七)：端&云整合的强龙策略

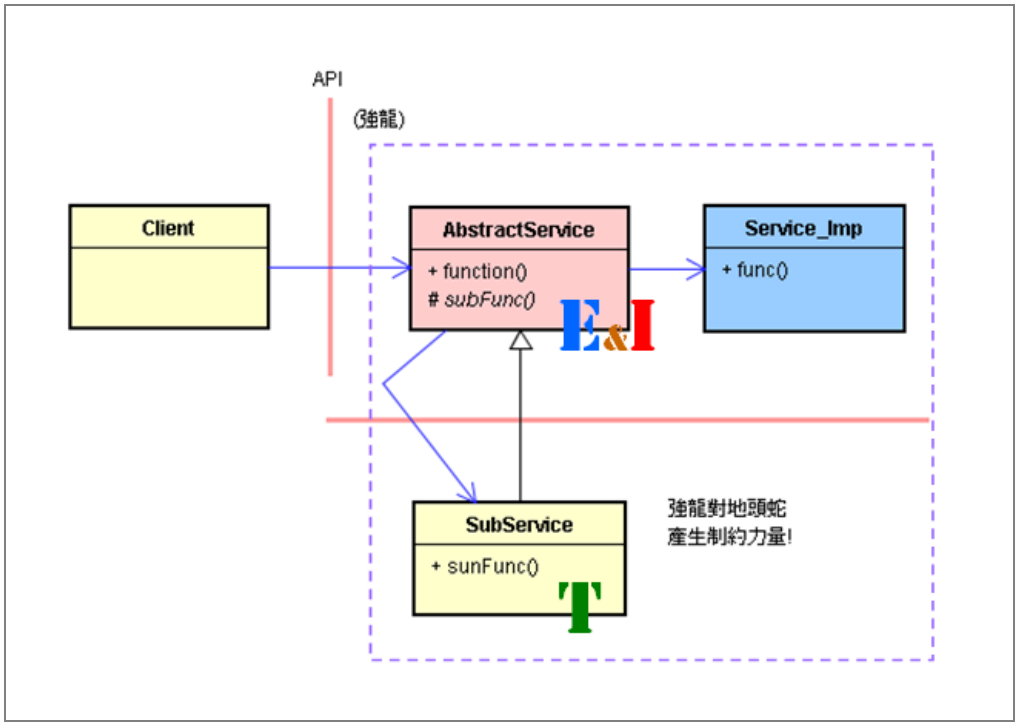
大家熟悉的 Client-Server 架构。



例如，Client 模块调用 Server 模块的 func()函数。



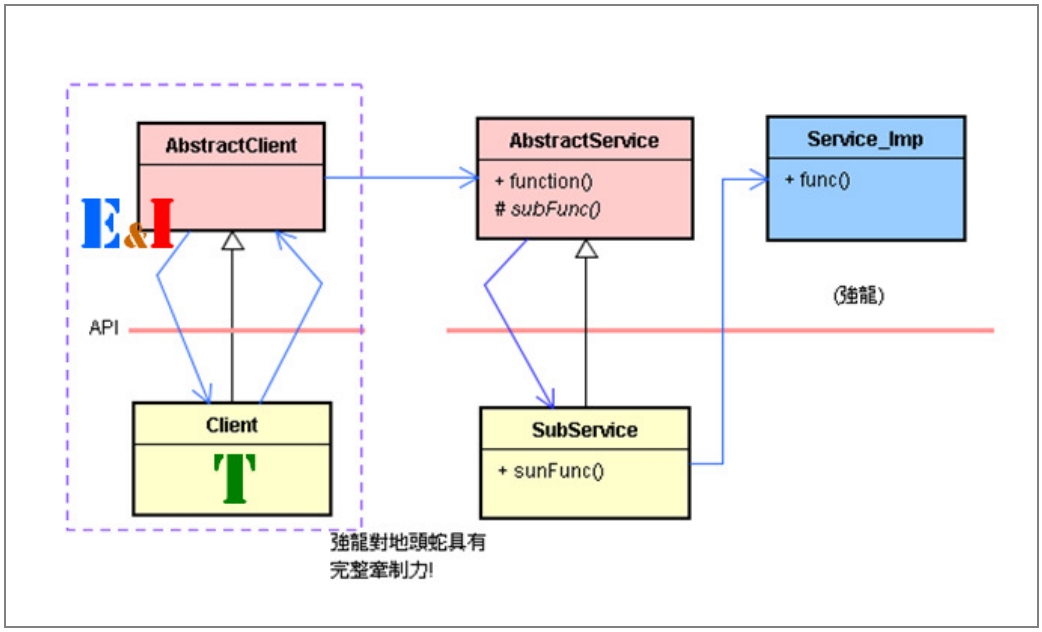
此时，Client 主动调用 Server 的 func()函数；其意味着，在此 Client-Server 架构里，Client 模块具有掌控权。因此，不利于 Server 开发者。那么，该如何才能反过来，让 Server 拥有主导权呢？答案是：加上一个 EIT 造形。



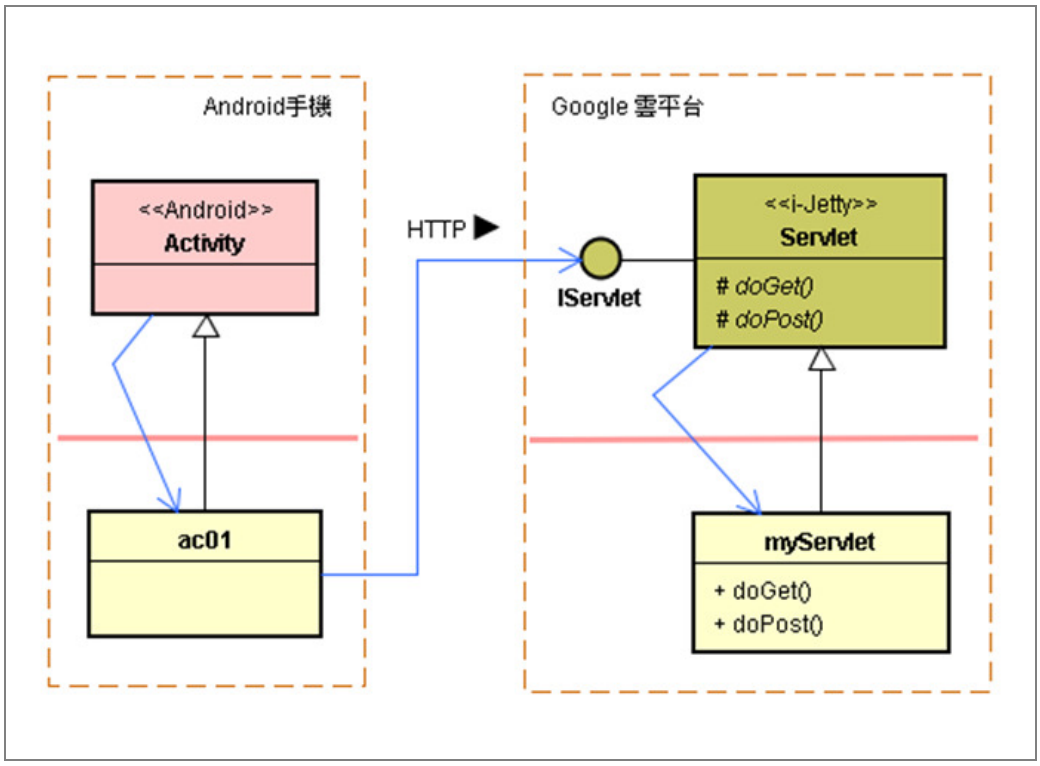
由Server开发者来撰写基类(即<E&I>)，并且让Client开发者来撰写<T>。此时，Server 开发者就具有主导权了。上图的 EIT 造形是位于 Server 端，就已经让 Server 开发者成为 Server 端的主导者了。

Server 开发者还可以运用 EIT 造形来扩大势力范围，也就是扩大到 Client

端，也成为 Client 端的主导者。此时，就在 Client 端设计一个 EIT 造形。



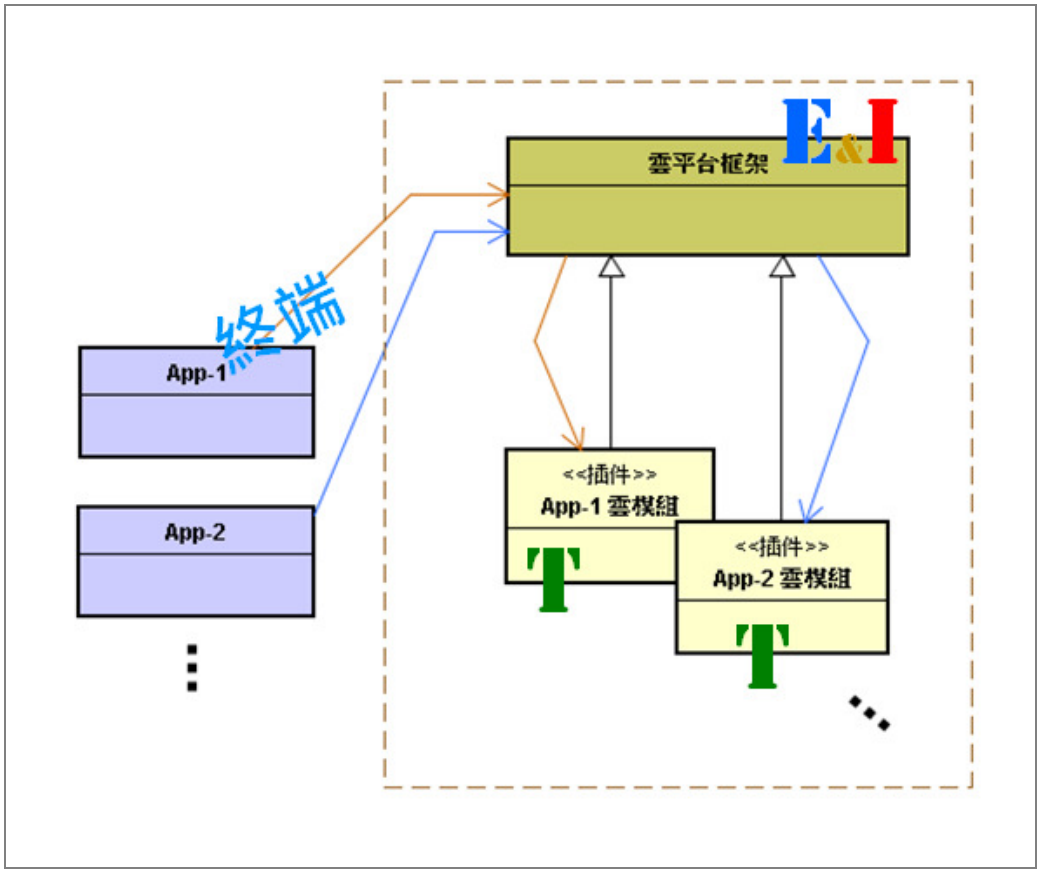
此时，两端各有一个 EIT 造形，而两个 EIT 造形的<E&I>都是由 Server 开发者来撰写，而两端的<T>则都由 Client 开发者撰写。于是，Server 开发者就成为雄踞两端的强龙了。例如，Google 就是典型的强龙，不仅在其 GAE 云平台上，撰写了 Servlet 基类；也在 Client 端撰写了 Activity 基类。



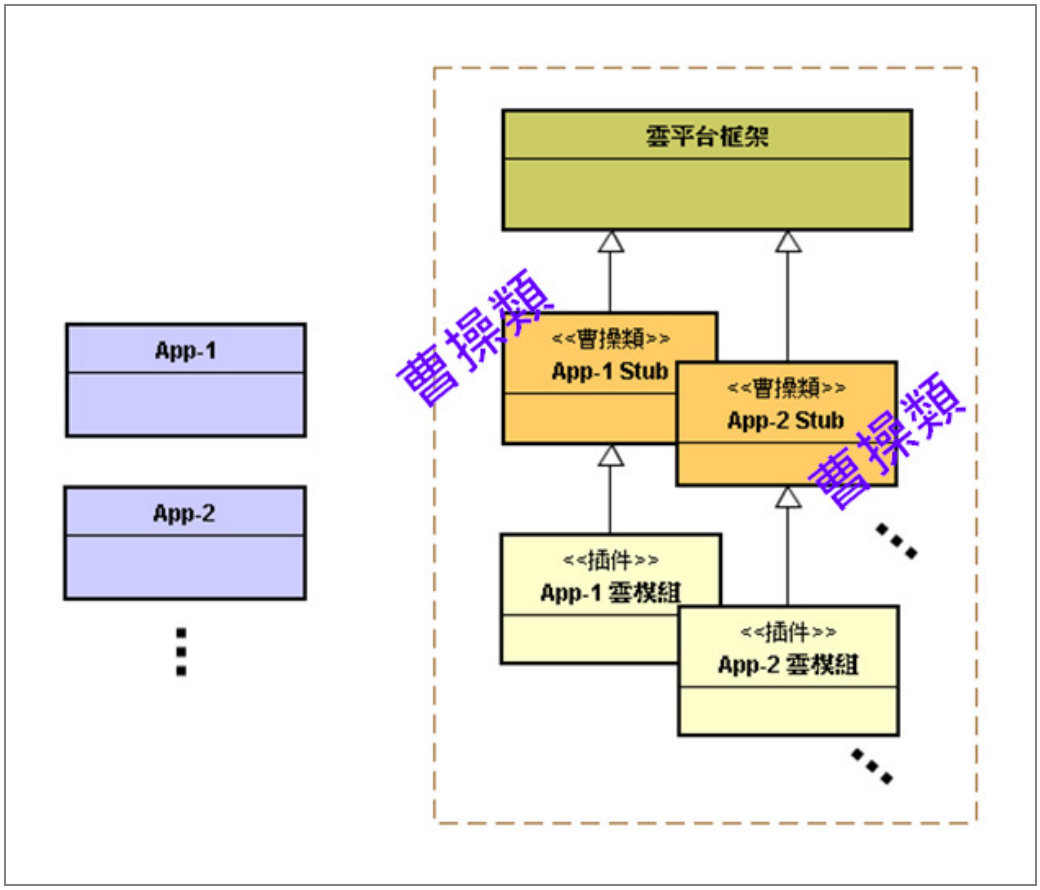
此时，两端各有一个 EIT 造形，而两个 EIT 造形的<E&I>都是由 Google 来撰写，而两端的<T>则都由 App 开发者所撰写。Google 就是典型的强龙了。

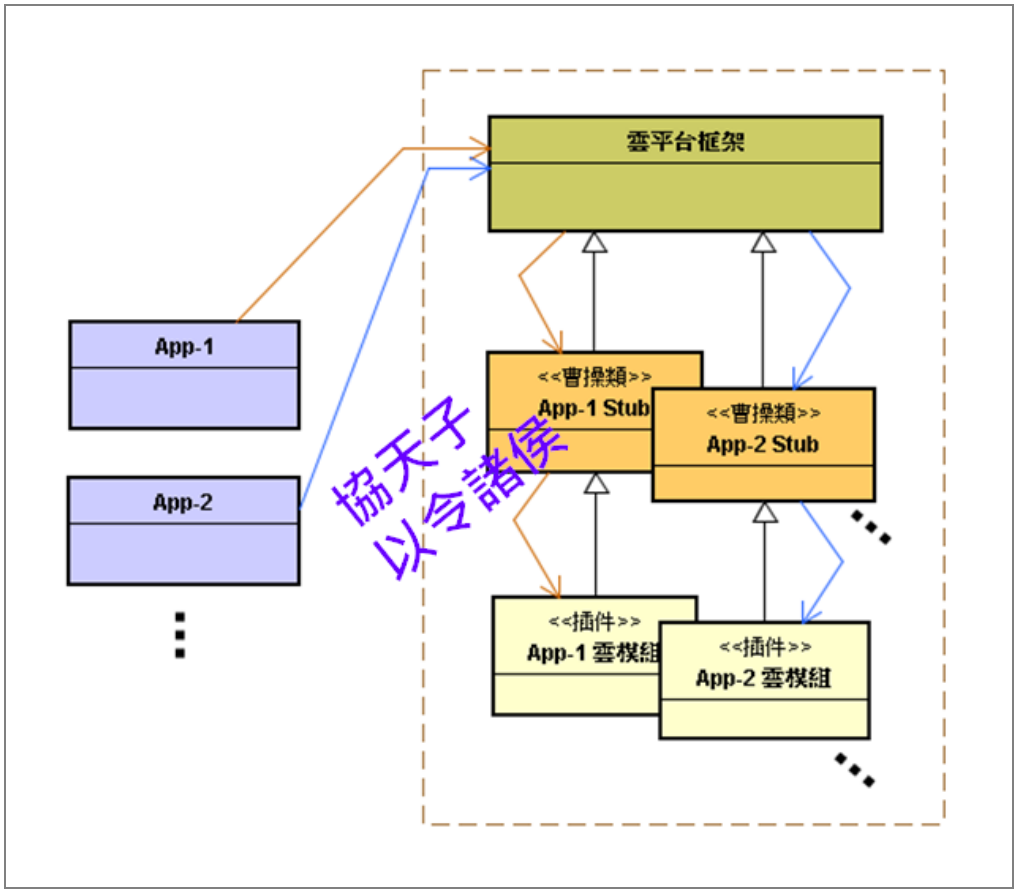
终端如何反制云端强龙呢?

在云端强龙策略下，终端厂商又有何反制策略呢？就终端 App 开发者来说，这个云平台<E&I>是别人的，(别人的)云平台要求 App 开发者去写插件<T>，让云平台跨终端了。



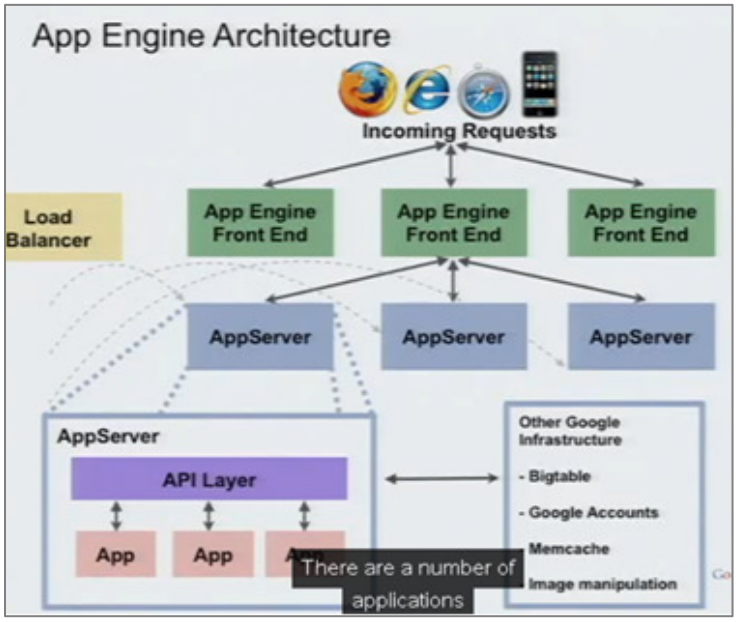
此时的应对策略是：挟天子以令诸侯。也就是设计云平台上的曹操类，如下图：


雲平台框架
曹操類
<<曹操類>>
App-1 Stub
<<曹操類>>
App-2 Stub
曹操類
App-1
App-2
<<插件>>
App-1 雲模組
<<插件>>
App-2 雲模組

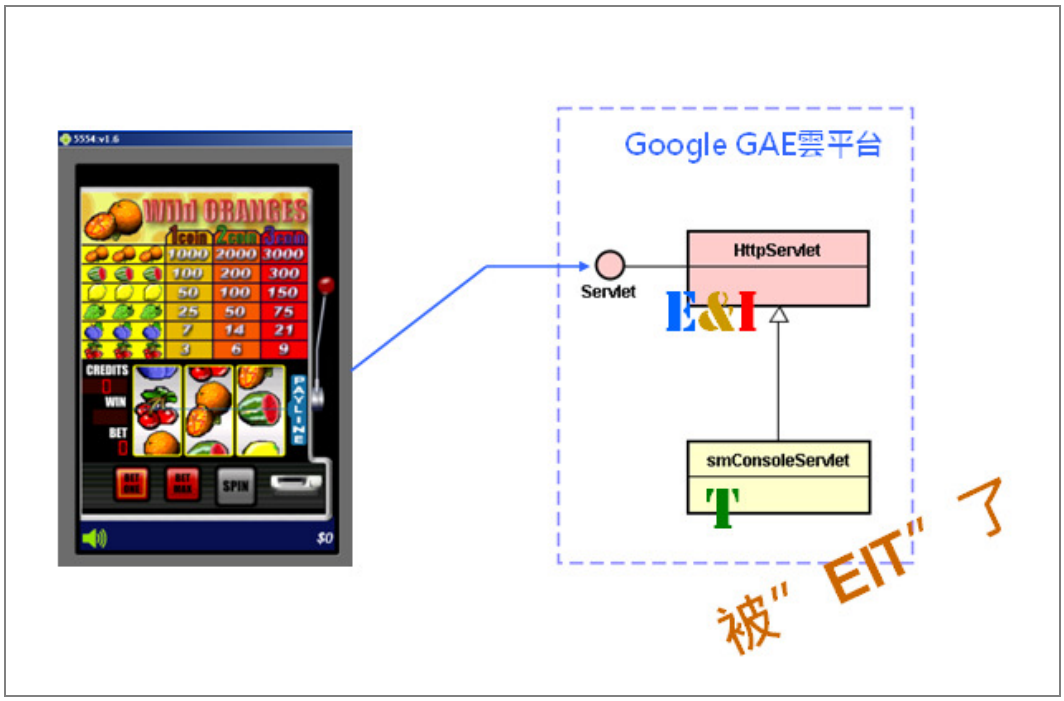

雲平台框架
App-1
App-2
協天子
以令諸侯
<<曹操類>>
App-1 Stub
<<曹操類>>
App-2 Stub
<<插件>>
App-1 雲模組
<<插件>>
App-2 雲模組

“挟天子以令诸侯”策略的范例 -- 以 SlotMachine 游戏机为例

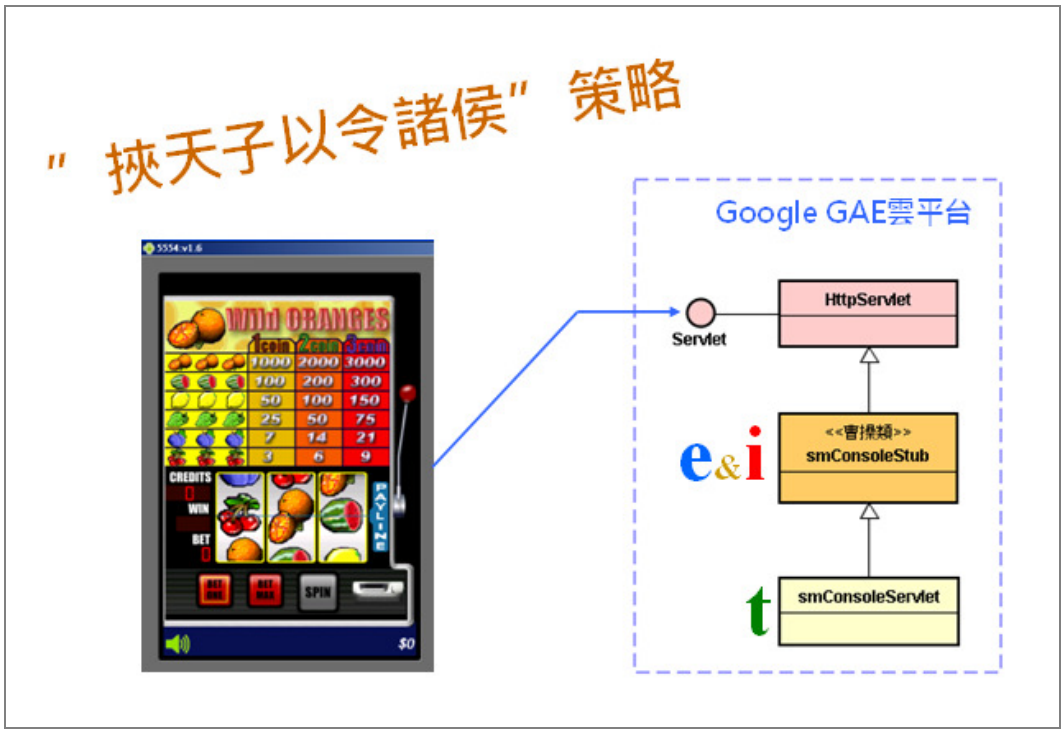
GAE(Google AppEngine)是 Google 的云服务引擎，第三方 App 开发者能开发 App 软件，然后放在 Google 服务器上执行，一切由 Google 代管。GAE 平台的系统架构如下图：



开放的 GAE 让我们撰写插件，放到 GAE 上执行。



此时我们(即终端 App 开发者)处于小弟的地位，也就是被云平台”EIT(设计)了”于是，可以设计设计”曹操类”，来实现”挟天子以令诸侯”策略。如下图：



~ End ~