Android 物联网师资培训暨教学研讨

物联网概述

- 物联网 (Internet of Things) 是继计算机、互联网移动通信网之后的又一次信息产业浪潮。物联网对促进互联网的发展、带动人类的进步发挥着重要的作用，并将成为未来经济发展的新增长点。
- 它将各种信息传感设备，如射频识别 (RFID) 装置、红外感应器、全球定位系统、激光扫描器等种种装置与互联网结合起来而形成的一个巨大网络拓扑。其目的是让所有的物品都与网络连接在一起，系统可以自动的、实时的对物体进行识别、定位、追踪、监控并触发相应事件。

- 物联网是一个新生概念，在这个领域，中国和其他发达国家站在一个起跑线上。
- 早在 1999 年，中科院就启动了传感网的研究和开发。
- 2005 年 11 月 27 日，在突尼斯举行的信息社会峰会上，国际电信联盟（ITU）发布了《ITU 互联网报告 2005 ：物联网》，正式提出了物联网的概念。

基于智能家居的物联网系统分析

- 智能家居系统是利用先进的计算机技术、网络通讯技术、综合布线技术、医疗电子技术依照人体工程学原理，融合个性需求，将与家居生活有关的各个子系统如安防、灯光控制、窗帘控制、煤气阀控制、信息家电、场景联动、地板采暖、健康保健、卫生防疫、安防保安等有机地结合在一起，通过网络化综合智能控制和管理，实现“以人为本”的全新家居生活体验。又称智能住宅，在国外常用 Smart Home 表示。
- 与智能家居系统含义近似的有家庭自动化（Home Automation）、电子家庭（Electronic Home、 E-home）、数字家园（Digital Family）、家庭网络（Home Net/Networks for Home）、网络家居（Network Home）、智能家庭 / 建筑（Intelligent Home/Building），在我国香港和台湾等地区，还有数码家庭、数码家居等称法。

智能家居的系统要求

- 实用性
- 便利性
- 可靠性
- 标准性
- 安全性

智能手机在智能家居中的应用分析

- 早期的智能家居系统，并无手机的位置。原因在于应用的复杂性、网络的成本等等。
- 随着智能手机的发展，手机在整个系统中的作用越来越重要，对提升用户使用的方便性、可操控性，都有很重要的作用。

- 基于 WIFI 、蓝牙、云的系统
- 基于 zigbee 的系统

物联网通信模式分析

- 感知
- 网络
- 应用

物联网常用的通信协议

- 物联网的通信环境有Ethernet， Wi-Fi， RFID， NFC（近距离无线通信）， Zigbee， 6LoWPAN（IPV6低速无线版本），Bluetooth， GSM， GPRS， GPS， 3G， 4G等网络，而每一种通信应用协议都有一定适用范围。 AMQP、 JMS、 REST/HTTP都是工作在以太网，COAP协议是专门为资源受限设备开发的协议，而DDS和MQTT的兼容性则强很多。
-
- zigbee目前在工业控制领域应用广泛，在智能家居领域也有一定应用。它有以下主要优势：
- 1. 低成本。数据传输速率低，协议简单，所以开发成本也比较低。并且zigbee协议还免收专利费用
- 2. 低功耗。由于zigbee协议传输速率低，节点所需的发射功率仅1mW，并采用休眠+唤醒模式，功耗极低。
- 3. 自组网。通过zigbee协议自带的mesh功能，一个子网络内可以支持多达65000个节点连接，可以快速实现一个大规模的传感网络。
- 4. 安全性。使用crc校验数据包的完整性，支持鉴权和认证，并且采用aes-128对传输数据进行加密。

zigbee 协议的不足

- zigbee 虽然可以方便的组网，但不能接入互联网，所以网络中必须有一个节点充当路由器的角色，这提高了成本并且增加了用户使用门槛。同时由于 zigbee 协议数据传输速率低，对于大流量应用如流媒体、视频等，基本是不可能。

- 相对 wifi 和蓝牙协议这些年的快速发展和商业普及， zigbee 协议尽管在技术设计和架构上拥有很大优势，但是技术更新太慢，同时在市场推广中也被竞争对手拉开了差距。后续 zigbee 协议在行业领域还是有很大空间，但是家用及消费领域要挑战 wifi 及蓝牙协议不是那么容易了。

蓝牙

- 蓝牙目前已经成为智能手机的标配通信组件，他的优势包括：
- 1. 低功耗。蓝牙 4.0 引入了 BLE 协议，使用纽扣电池可运行数月乃至一年以上，这对可穿戴设备具有十分大的吸引力。
- 2. 智能手机的普及。近年来支持蓝牙协议基本成为智能手机的标配，用户无需购买额外的接入模块，降低了用户的使用门槛。

- 另外值得关注的是蓝牙 4.2 版本加入了 mesh 组网功能，向 zigbee 发出了强有力的挑战。

WIFI

- wifi 协议目前也得到了非常大的发展。由于家用 wifi 路由器以及智能手机的迅速普及，wifi 协议在智能家居领域也得到了广泛应用。
- 它最大的优势是可以直接接入互联网。相对于 zigbee ，采用 wifi 协议的智能家居方案省去了额外的网关，相对于蓝牙协议，省去了对手机等移动终端的依赖。
- 当然，wifi 协议的功耗较大，这成为其在物联网领域应用的一大瓶颈。但是随着现在各大芯片厂商陆续推出低功耗、低成本的 wifi soc （如 esp8266），这个问题也在逐渐被解决。
- 在对功耗较不敏感的产品上，wifi 的优势更是不可替代，如智能插座、智能灯具等等。

物联网常用的通讯协议

- 比较常见的通讯协议包括：HTTP、 web socket、 XMPP、 COAP、 MQTT。
-
- 物联网的通信架构构建在传统互联网基础架构之上。因此，在当前的互联网通信协议中，HTTP协议由于开发成本低，开放程度高，几乎占据大半江山。
- 但在物联网应用中他的劣势也很明显：
- 1. 由于必须由设备主动向服务器发送数据，难以主动向设备推送数据。对于数据采集等场景还勉强适用，但若频繁的操控，只能通过设备定期主动拉取的的方式，实现成本和实时性都大打折扣。
- 2. 安全性不高。HTTP是明文协议，在很多要求高安全性的物联网场景，必须作很多安全准备工作（如采用https等）。
- 3. 不同于用户交互终端如pc、手机，物联网场景中的设备多样化，对于某些运算和存储资源都十分受限的设备，很难实现http协议、 XML/JSON数据格式的解析。

- 所以，目前众多物联团队在设计物联网云平台时，也只是在针对手机或PC的用户时，采用HTTP协议，针对设备的物联网接入，一般不采用HTTP协议。

- 由于物联网设备通信的模式和互联网中的即时通讯应用非常相似，互联网中常用的即时通讯协议也被大量运用于物联网系统构建中，这其中的典型是 XMPP 。
- 类似的还有 COAP 、 MTQQ 等即时通讯协议，也在一些系统中得到应用。
-
- 当前的物联网通信协议，没有任何协议能够在市场上占有统治地位。但要实现物联网设备互联互通（不同厂商、不同平台、不同架构），关键点并不在上述接入协议或通讯协议的统一，而在于上层业务应用层协议的统一。无论是 wifi 、蓝牙、亦或是 mqtt 、 http 都是设备进行数据通讯和交换的通道，规定的是通讯的格式；而通讯的内容的统一才是实现互联互通的关键。

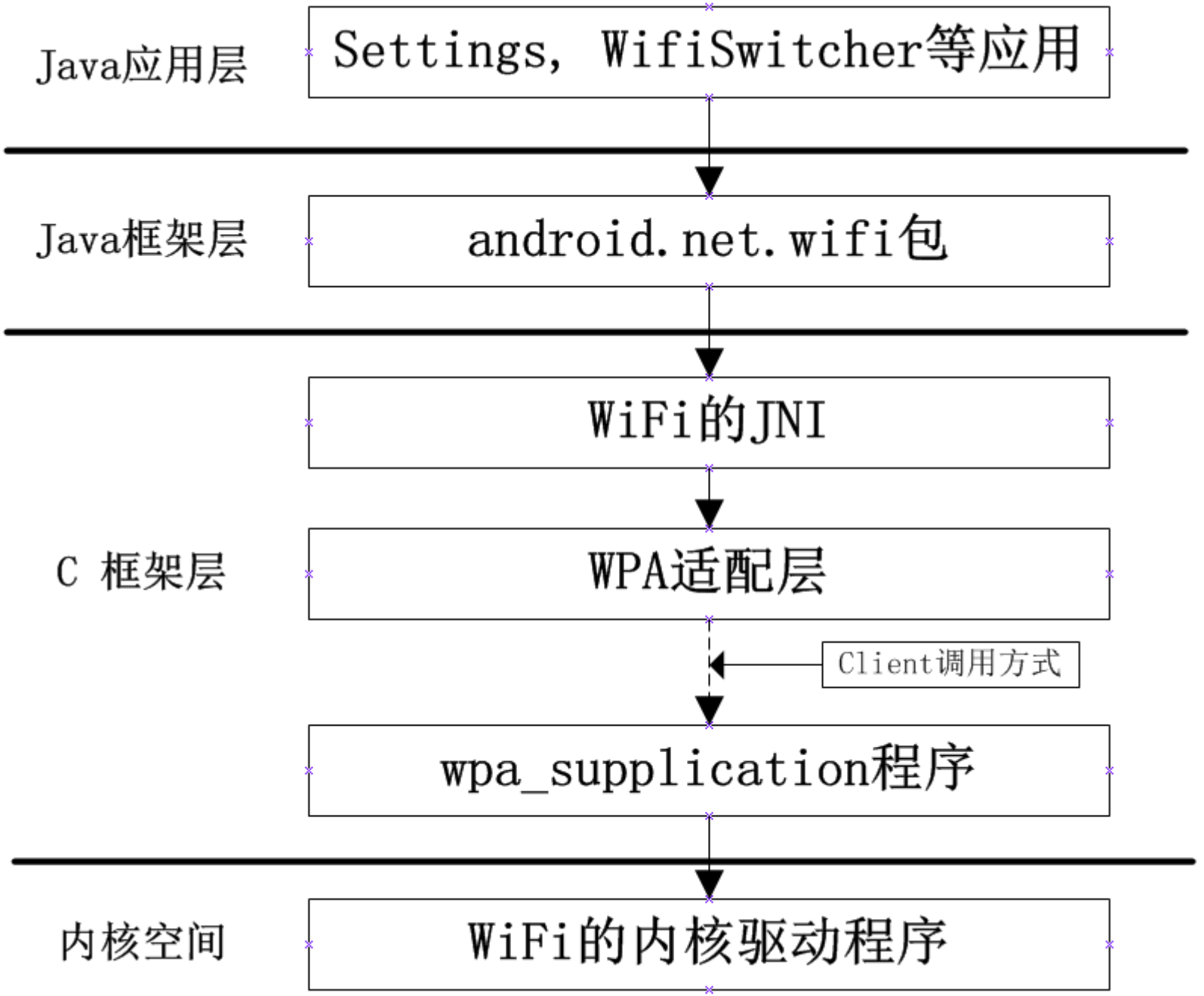
PHILIPS HUE 简介

- HUE 是 Philips 推出的智能互联照明系统。
- 他本身是完整的照明系统，可以较为完善的实现智能控制功能。
- 另外他也提供了 REST API ，供第三方操作自己，这为 HUE 带来 了无数奇妙的应用，如提示天气、甚至股票、电子邮件等等。

- HUE 桥接器可以同时支持 50 个灯泡，而且 HUE 使用 LED 灯泡，节能性很好。另外飞利浦在 ios 、 android 上均提供了应用和 SDK 供终端用户和二次开发。

1. WIFI 通信协议简介

Android 的 WiFi 系统



- WiFi 作为网络部分使用的方法和通常的网络相同。
- 唯一特殊的部分是在 Settings 程序中的 WiFi 相关设置内容，它们调用了 WiFi 提供了 WiFi 架构提供的 Java 层接口。

- wpa_supplicant 是一个独立运行的守护进程，其核心是一个消息循环，在消息循环中处理 WPA 状态机、控制命令、驱动事件、配置信息等。 wpa_supplicant 有很多控制接口，也提供命令行和通行界面的控制模式：而 Android 与 wpa_supplicant 的通信通过 Socket 完成。
-
- wpa_supplicant 适配层是通用的 wpa_supplicant 的封装，在 Android 中作为 WiFi 部分的硬件抽象层来使用。 wpa_supplicant 适配层主要用于与 wpa_supplicant 守护进程的通信，以提供给 Android 框架使用，它实现了加载、控制和消息监控等功能。

- WiFi 系统 Java 层的核心是根据 IWifiManger 接口所创建的 Binder 服务器端和客户端，服务器端是 WifiService ，客户端是 WifiManger 。
- WiFiManger 是 WiFi 部分与外界的接口，用户通过它来访问 WiFi 的核心功能。 WifiWatchdogService 这一系统组件也是用 WifiManger 来执行一些具体操作。
- WifiService 是服务器端的实现，作为 WiFi 部分的核心，处理实际的驱动加载、扫描、链接、断开等命令，已经底层上报的事件。对于主动的命令控制， WiFi 是一个简单的封装，针对来自客户端的控制命令，调用相应的 WifiNative 底层实现。
- Android 的 Settings 应用程序对 WIFI 的使用，是典型的 WiFi 应用方式，也是用户可见的 Android WiFi 管理程序。
- WifiEnabler 和 WifiLayer 都是 WifiSettings 的组成部分，同样通过 WifiManger 来完成实际的功能，也同样注册一个 BroadcastReceiver 来响应 WifiStateTracker 所发出的通知消息。 WifiEnabler 其实是一个比较简单的类，提供开启和关闭 WiFi 的功能，设置里面的外层 WiFi 开关菜单，就是直接通过它来做到的。

3. Android WIFI 编程

- // 获得 WifiManager 对象
- wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
- // 获得连接信息对象
- wifiInfo = wifiManager.getConnectionInfo();
- // 根据当前 WIFI 的状态（是否被打开）设置复选框的选中状态
- if (wifiManager.isWifiEnabled()) {
-

- // 获得 WIFI 信息
- StringBuffer sb = new StringBuffer();
- sb.append("Wifi 信息 \n");
- sb.append("MAC 地址： " + wifiInfo.getMacAddress() + "\n");
- sb.append(" 接入点的 BSSID ： " + wifiInfo.getBSSID() + "\n");
- sb.append("IP 地址（int）： " + wifiInfo.getIpAddress() + "\n");
- sb.append("IP 地址（Hex）： " + Integer.toHexString(wifiInfo.getIpAddress()) + "\n"); sb.append("IP 地址： " + ipIntToString(wifiInfo.getIpAddress()) + "\n");
- sb.append(" 网络 ID ： " + wifiInfo.getNetworkId() + "\n");

- wifiConfigurations = wifiManager.getConfiguredNetworks();
- tvWifiConfigurations.setText(" 已连接的无线网络 \n");
- for (WifiConfiguration wifiConfiguration : wifiConfigurations) {
- tvWifiConfigurations.setText(tvWifiConfigurations.getText() + wifiConfiguration.SSID + "\n");
- }

- wifiManager.setWifiEnabled(true);
-
- 在 AndroidManifest.xml 文件中要使用如下的代码打开相应的权限。
- <uses-permission android:name="android.permission.ACCESS_WIFI_STATE">
- </uses-permission>
- <uses-permission android:name="android.permission.WAKE_LOCK">
- </uses-permission>
- <uses-permission android:name="android.permission.CHANGE_WIFI_STATE">
- </uses-permission>

创建一个 wifi 热点

```
//获取wifi管理服务
        wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);

//wifi和热点不能同时打开，所以打开热点的时候需要关闭wifi
            wifiManager.setWifiEnabled(false);

//热点的配置类
            WifiConfiguration apConfig = new WifiConfiguration();
            //配置热点的名称(可以在名字后面加点随机数什么的)
            apConfig.SSID = "YRCCONNECTION";
            //配置热点的密码
            apConfig.preSharedKey="12122112";
                //通过反射调用设置热点
            Method method = wifiManager.getClass().getMethod(
                    "setWifiApEnabled", WifiConfiguration.class, Boolean.TYPE);
            //返回热点打开状态
            return (Boolean) method.invoke(wifiManager, apConfig, enabled);
```

搜索热点

```
wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
        wifiReceiver = new WifiReceiver();
// 注销广播 unregisterReceiver(wifiReceiver);
wifiManager.startScan();

/* 监听热点变化 */
    private final class WifiReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            wifiList = wifiManager.getScanResults();
            if (wifiList == null || wifiList.size() == 0 || isConnected)
                return;
            onReceiveNewNetworks(wifiList);
        }
    }
```

```
/* 当搜索到新的wifi热点时判断该热点是否符合规格 */
    public void onReceiveNewNetworks(List<ScanResult> wifiList){
        passableHotsPot=new ArrayList<String>();
        for(ScanResult result:wifiList){
            System.out.println(result.SSID);
            if((result.SSID).contains("YRCCONNECTION"))
                passableHotsPot.add(result.SSID);
        }
        synchronized (this) {
            connectToHotpot();
        }
    }
```

```
* 连接到热点 */
    public void connectToHotpot(){
        if(passableHotsPot==null || passableHotsPot.size()==0)
            return;
        WifiConfiguration wifiConfig=this.setWifiParams(passableHotsPot.get(0));
        int wcgID = wifiManager.addNetwork(wifiConfig);
        boolean flag=wifiManager.enableNetwork(wcgID, true);
        isConnected=flag;
        System.out.println("connect success? "+flag);
    }
```

```
/*设置要连接的热点的参数*/
    public WifiConfiguration setWifiParams(String ssid){
        WifiConfiguration apConfig=new WifiConfiguration();
        apConfig.SSID="\""+ssid+"\"";
        apConfig.preSharedKey="\"12122112\"";
        apConfig.hiddenSSID = true;
        apConfig.status = WifiConfiguration.Status.ENABLED;
        apConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
        apConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
        apConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
        apConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
        apConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
        apConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);
        return apConfig;
    }
```

Android Socket 编程

- 基本与 PC 上的 java 一致
- 需要注意的是：
- 网络部分绝对不能使用 UI 线程，因此需要使用 Thead 或者 Runnable 来完成。

- Server
-

```
ServerSocket serverSocket = new ServerSocket( PCPORT );
while (true) {
    Socket client = serverSocket.accept();
    BufferedReader in = new BufferedReader(
            new InputStreamReader(client.getInputStream()));
    String clientConent = in.readLine();
}
```

- Client

- InetAddress serverAddr = serverAddr = InetAddress.getByName ("192.168.0.254");
- Socket socket = new Socket(serverAddr, 55555);
- PrintWriter out = new PrintWriter(new BufferedWriter(
- new OutputStreamWriter(socket.getOutputStream())), true);
- out.println(“This is from client”);
-

Android 多播编程

- 组播使用 UDP 对一定范围内的地址发送相同的一组 Packet ，即一次可以向多个接受者发出信息，其与单播的主要区别是地址的形式。 IP 协议分配了一定范围的地址空间给多播（多播只能使用这个范围内的 IP ）， IPv4 中组播地址范围为 224.0.0.0 到 239.255.255.255 。
-
- java 中通过 MulticastSocket 实例进行通信，使用时涉及到几个概念
  ① TTL （ Time To Live ）, 每个 IP 报文都包含一个 TTL （是一个数字），报文每被一个路由转发一次它的 TTL 减 1 ，当 TTL 变为 0 时，该报文被丢弃
- ② 多播组（ multicast group ），接受者只有加入这个组才能获取发送到该组的报文（这就确定了组播的对象）
-
- 他们在智能家居系统中，常用来解决发现设备这个问题。

- 发送端（Android手机）：
- //----------
- MulticastSocket mSocket = new MulticastSocket(30001);//生成套接字并绑定30001端口
- InetAddress group=InetAddress.getByName("239.0.0.1");//设定多播IP
- byte[] buff = "QQ".getBytes("utf-8");//设定多播报文的数据
- mSocket.joinGroup(group);//加入多播组，发送方和接受方处于同一组时，接收方可抓取多播报文信息
- mSocket.setTimeToLive(4);//设定TTL
- //设定UDP报文（内容，内容长度，多播组，端口）
- DatagramPacket packet = new DatagramPacket(buff,buff.length,group,30001);
- mSocket.send(packet);//发送报文
- mSocket.close();//关闭套接字

- 接收端（PC）：
- //----------
- MulticastSocket s = new MulticastSocket(30001);// 生成套接字并绑定端口
- InetAddress group = InetAddress.getByName("239.0.0.1");// 设定多播 IP
- s.joinGroup(group);// 接受者加入多播组，需要和发送者在同一组
- DatagramPacket packet = new DatagramPacket(buffer，100);// 创建接收报文，以接收通过多播传递过来的报文
- s.receive(packet);// 接收多播报文，程序停滞等待直到接收到报文
-
- s.close();// 关闭套接字

- 使用广播，本地网络中所有的主机都会受到一份数据副本。广播使用 UDP 报文， IPv4 使用（255.255.255.255）地址发送广播，本地广播绝不会被路由器转发，即广播信息会被限制在本地网络之内。

- 发送端（Android手机）
- //----------
- byte[] buff = "QQ".getBytes("utf-8");//设定报文信息
- DatagramSocket socket=new DatagramSocket();//建立套接字，参数端口号不填写，系统会自动分配一个可用端口
- //创建报文，包括报文内容，内容长度，报文地址（这里全1地址即为广播），端口号（接受者需要使用该端口）
- DatagramPacket packet=new DatagramPacket(buff,buff.length,InetAddress.getByName("255.255.255.255"), 30000);
- socket.send(packet);//发送报文
- socket.disconnect();//断开套接字
- socket.close();//关闭套接字

- //----------
- 接受端（PC）：
- //----------
- DatagramSocket socket=new DatagramSocket(30000);//创建套接字
- byte[] buffer;//创建接收字符串
- buffer=new byte[35];
- DatagramPacket packet = new DatagramPacket(buffer，buffer.length);//创建接收报文，以接收通过广播传递过来的
- System.out.println("Listening at UDP(30000)....");
- socket.receive(packet);//接收报文，程序停滞等待直到接收到报文
- socket.disconnect();//断开套接字
- socket.close();//关闭套接字

- HUE 支持 UPNP ，在 SDK 中使用 PHHueSDK.SEARCH_BRIDGE 服务， search 方法可以使用 upnp 搜索。
- 但实践中，往往需要用到 search 的 IPScan 方式才可以搜索到桥接器。估计和实际网络的设置有关。

- 无论何种方法，都需要最终搜索到桥接器的 IP ，随后才能展开其他操作。

Philips HUE API 简介

- 结合 HUE 的开发文档，介绍 HUE 的一些概念

REST API

- REST即表述性状态传递（英文： Representational State Transfer ，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种软件架构风格。它是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。
- 目前在三种主流的Web服务实现方案中，因为REST模式的Web服务与复杂的SOAP和XML-RPC对比来讲明显的更加简洁，越来越多的web服务开始采用REST风格设计和实现。
- REST是设计风格而不是标准。 REST通常基于使用HTTP ， URI ，和XML（标准通用标记语言下的一个子集）以及HTML （标准通用标记语言下的一个应用）这些现有的广泛流行的协议和标准。

- 结合HUE开发文档，介绍REST API、接口解释

Philips HUE 编程与控制

- 如何开始？

  1. 连接上网络，用标准的 Philips HUE app 判断是否正确安装。
  2. 找到桥接器的 IP ，你可以用各种方式

  １） upnp 搜索

  ２） 访问 https://www.meethue.com/api/nupnp ，用 Philips 的网页服务搜索

  ３） 在路由器里去找 DHCP 分配表

  ４） 其他方法

  3. 使用 REST API 、 SDK 等，进行各种管理和控制

Philips HUE 综合案例实践

- 写一个 android 程序实现 hue 的综合应用
  1. 使用 REST API
  2. 使用提供的 SDK

- 功能
  1. 开关控制
  2. 颜色控制
  3. 亮度控制