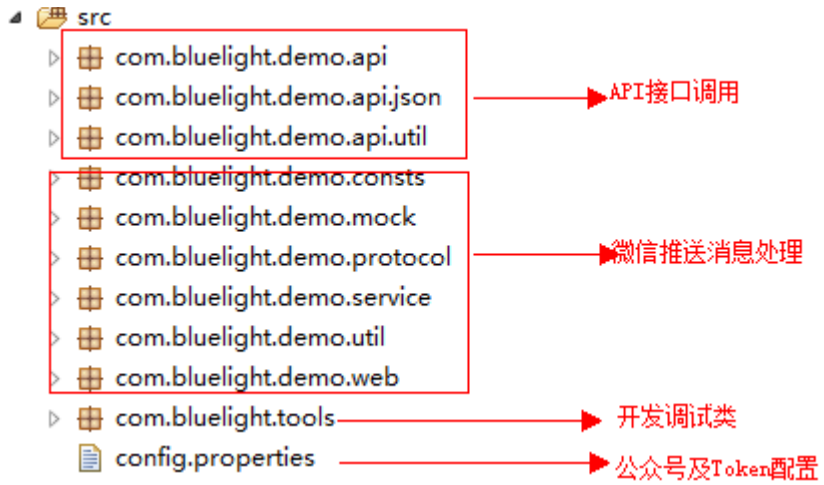
蓝牙 demo 服务端开发

Demo 源代码

蓝牙灯泡 demo 服务端为 java 语言开发。

代码目录介绍：



处理微信回调请求入口类：com.bluelight.demo.web.CallbackServlet

下面介绍 demo 服务端开发过程：

申请账号

针对个人开发者/企业测试账户： 申请微信

公众平台接口测试账号

http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=sandbox/login

打开网址，点击登录按钮后，显示二维码图片。使用微信扫描二维码，在弹出页面中点击确认登录，即可获得并登录测试账号。

在测试号界面，可以看到测试号相关信息：appID、appsecret、右上角的微信号（即原始 id，以 gh_开头）。

测试账号默认即有权限调用设备功能接口。

针对企业正式用户（在正式号申请期间，可以使用测试账户进行开发调试）：

1、 注册公众服务号（或使用现有服务号）

2、登录服务号后，点击左侧“添加功能插件”，在右侧界面选择“设备功能”，按照界面中的“设备接入流程”进行接入申请。

服务号信息查看：

原始 ID：点击左侧菜单列表设置中的公众号设置，在右侧界面即可查看。 appID、appsecret：点击左侧菜单列表最下的开发者中心，在右侧界面即可查看。

appID、appsecret 用于获取接口访问凭证，原始 id 为设备接口中的设备类型 DeviceType。

服务端接入

公众平台地址

https://mp.weixin.qq.com/

公众平台开发者文档地址

http://mp.weixin.qq.com/wiki/home/index.html

确定 Token：Token 可以任意填写，用作生成签名。通过比较签名是否相等，可以确

认请求是由微信服务器发送的，还是他人伪造的。

实现开发者接入验证：在处理 get 请求中进行签名验证，验证签名一致后，返回参数中的 echostr 参数。

将代码部署到服务器，微信公众号接口只支持 80 接口。 在测试账号界面，或者服务号的开发者中心界面，填写 Token 及服务器处理回调请求

的 URL 地址。提交后，微信会发送 get 请求到填写的地址。如果正常返回传入的 echostr 参数，则验证成功，否则提示失败。

Demo 中开发者接入验证代码见 CallbackServlet 类。

Demo 部署：

* demo 工程导入 Eclipse。

在 config.properties 中填入 appID、appsecret、token 的值。

生成 war 包：

在 build.xml 第 5 行，war.name 属性的值为 war 包名称。

打开 ant 界面（菜单栏 Window->Show View->Other->弹出窗口中输入 ant，选中 Ant，点击 OK），将 build.xml 拖入 ant 窗口，双击 make-war 命令，会在 dist 目录下生成对应的 war 包。

将 war 包部署到服务器。如果部署在自己的服务器不方便，也可以部署到第三方云平 台。

公众平台消息处理

公众平台的接口分为两类：

1、 消息接口：公众平台将用户操作的信息发送给开发者配置的 URL，并接收回复。

2、 API 接口：由开发者主动调用的接口。

消息接口将数据通过 POST 请求发送到开发者填写的 URL 上，数据格式为 XML。

不同的消息类型拥有不同的数据结构，但是所有的 XML 请求都包含以下几个元素：

ToUserName	接收方帐号（公众号 ID）
FromUserName	发送方帐号（OpenID）
CreateTime	消息创建时间（秒级）
MsgType	消息类型

通过 MsgType 可以区分具体的消息类型，参照对应类型的接口文档结构描述，可获取对应的元素数据。

Demo 中 CallbackServlet 中的 doPost 方法为回调处理的入口，CallbackService 类进行实际业务处理。

具体过程为：

1、 进行签名验证，不相等则为非法来源，直接返回空字符串。

2、 解析请求 XML 为 Map 对象，方便程序获取值。

3、 根据参数 MsgType 判断消息类型，进行相应业务处理（CallbackService 类）。4、返回空字符串或者 XML 格式的消息。

下面以 demo 中涉及的具体消息类型为例进行说明。

基本消息处理

用户关注公众号用户在关注与取消关注公众号时，微信会把这个事件推送给开发者。方便开发者给用户下发 欢迎消息或者做帐号的解绑。结构体例子如下：

```
<xml>
   <ToUserName><![CDATA[toUser]]></ToUserName>
   <FromUserName><![CDATA[FromUser]]></FromUserName>
   <CreateTime>123456789</CreateTime>
   <MsgType><![CDATA[event]]></MsgType>
   <Event><![CDATA[subscribe]]></Event>
</xml>
```

MsgType 为 event，Event 元素为 subscribe 表示关注，unsubscribe表示取消关注。通常在关注事件中返回欢迎提示语。

用户点击自定义菜单

用户点击自定义菜单后，微信会把点击事件推送给开发者。结构体例子如下：

```
<xml>
   <ToUserName><![CDATA[toUser]]></ToUserName>
   <FromUserName><![CDATA[FromUser]]></FromUserName>
   <CreateTime>123456789</CreateTime>
   <MsgType><![CDATA[event]]></MsgType>
   <Event><![CDATA[CLICK]]></Event>
   <EventKey><![CDATA[EVENTKEY]]></EventKey>
</xml>
```

MsgType 为 event，Event 元素为 CLICK 表示点击自定义菜单，EventKey 为创建菜单时指定的值，可以通过该值判断用户点击了哪一个菜单。

用户发送消息

当微信用户向公众账号发消息时，微信服务器会将消息发送给开发者。文本消息结构体例子：

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1348831860</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[this is a test]]></Content>
  <MsgId>1234567890123456</MsgId>
</xml>
```

消息类型有多种，文本消息的 MsgType 为 text，Content为消息内容。

在文本消息中可以进行关键字自动回复，demo 中返回“收到文本消息：”+用户输入。

基本消息处理，Demo 中代码见 CallbackService类：

```
public String handle(Map<String, String> reqMap) throws Exception {
    String msgType = reqMap.get("MsgType");
    String fromUser = reqMap.get("FromUserName");
    String toUser = reqMap.get("ToUserName");
    // 针对不同类型的消息和事件进行处理
    // 文本消息
    if (MsgType.TEXT.equals(msgType)) {
      // 可以在此处进行关键字自动回复
      String content = "收到文本消息：" + reqMap.get("Content");
      return XmlResp.buildText(fromUser, toUser, content);
    }
    // 基础事件推送
    if (MsgType.EVENT.equals(msgType)) {
      String event = reqMap.get("Event");
      // 关注公众号
      if (MsgType.Event.SUBSCRIBE.equals(event)) {
        // 回复欢迎语
        return XmlResp.buildText(fromUser, toUser, "欢迎关注蓝牙灯泡 demo 测试公众号
        ！ ");
      }
      // 菜单点击事件
      if (MsgType.Event.CLICK.equals(event)) {
        // 根据 key 值判断点击的哪个菜单
        String eventKey = reqMap.get("EventKey");
        // 点灯/灭灯
        if (V1001_LIGHT_ON.equals(eventKey)
            || V1002_LIGHT_OFF.equals(eventKey)) {
            //……
```

```
            }
        }
    }
    //.....
}
```

设备消息处理

用户进行设备绑定/解绑 用户通过扫描设备二维码绑定设备时，或取消关注公众号解除绑定时，微信会推送消息给开 发者。消息结构体如下：

```
<xml>
   <ToUserName><![CDATA[%s]]></ToUserName>
   <FromUserName><![CDATA[%s]]></FromUserName>
   <CreateTime>%u</CreateTime>
   <MsgType><![CDATA[%s]]></MsgType>
   <Event><![CDATA[%s]]></Event>
   <DeviceType><![CDATA[%s]]></DeviceType>
   <DeviceID><![CDATA[%s]]></DeviceID>
   <Content><![CDATA[%s]]></Content>
   <SessionID>%u</SessionID>
   <OpenID><![CDATA[%s]]></OpenID>
</xml>
```

MsgType 为 device_event，Event 取值为 bind/unbind，bind 表示绑定设备，unbind 表示解除绑定。

在绑定事件中，开发者需要将用户与设备的绑定关系进行存储。Demo 只存在了内存中，正式环境下需要存在数据库中。在用户通过微信控制设备时，需要根据 FromUserName 查询到设备信息（DeviceType、DeviceID、OpenID），才能推送数据和控制信息到设备。

在解除绑定事件中，开发者需要删除或失效用户与设备的绑定关系。 该接口可以直接回复空字符串。

接收设备消息

设备可以通过微信将数据发送给开发者，消息结构体如下：

```
<xml>
   <ToUserName><![CDATA[%s]]></ToUserName>
   <FromUserName><![CDATA[%s]]></FromUserName>
   <CreateTime>%u</CreateTime>
   <MsgType><![CDATA[%s]]></MsgType>
   <DeviceType><![CDATA[%s]]></DeviceType>
   <DeviceID><![CDATA[%s]]></DeviceID>
   <Content><![CDATA[%s]]></Content>
   <SessionID>%lu</SessionID>
   <MsgID>%lu</MsgID>
   <OpenID><![CDATA[%s]]></OpenID>
</xml>
```

MsgType 为 device_text，Content 为数据内容，经过了 BASE64 编码。

开发者收到数据后，处理过程如下：

1、进行 BASE64 解码。

2、将二进制数据反序列化为自定义的结构。该结构使用与设备约定好的协议解析数据。

3、进行业务逻辑处理。

在 demo 中，将设备发送的消息以文本消息的形式推送到用户微信公众号界面，

并响应一个 resp 包，返回设备发送来的数据。

4、将需要返回的数据序列化为二进制数据。

5、对二进制数据进行 BASE64 编码并进行回复。

该接口要求返回必须为符合协议的 XML 结构体（即不能直接返回空字符串），返回的数 据经过微信终端解码后发送给设备。

响应消息结构体如下：

```
<xml>
   <ToUserName><![CDATA[%s]]></ToUserName>
   <FromUserName><![CDATA[%s]]></FromUserName>
   <CreateTime>%u</CreateTime>
   <MsgType><![CDATA[%s]]></MsgType>
   <DeviceType><![CDATA[%s]]></DeviceType>
```

```
  <DeviceID><![CDATA[%s]]></DeviceID>
  <SessionID>%u</SessionID>
  <Content><![CDATA[%s]]></Content>
</xml>
```

设备消息处理 Demo 中代码见 CallbackService类：

```
// 设备消息或事件
if (MsgType.DEVICE_EVENT.equals(msgType)
      || MsgType.DEVICE_TEXT.equals(msgType)) {
    String reqContent = reqMap.get("Content");
    String deviceType = reqMap.get("DeviceType");
    String deviceID = reqMap.get("DeviceID");
    String sessionID = reqMap.get("SessionID");
    final String openID = reqMap.get("OpenID");
    // 设备事件推送
    if (MsgType.DEVICE_EVENT.equals(msgType)) {
        String event = reqMap.get("Event");
        // 绑定/解绑事件
        if (MsgType.DeviceEvent.BIND.equals(event)
              || MsgType.DeviceEvent.UNBIND.equals(event)) {
            // 存储用户和设备的绑定关系
            if(MsgType.DeviceEvent.BIND.equals(event)){
                DBMock.saveBoundInfo(reqMap);
            }else{
                DBMock.removeBoundInfo(reqMap.get("FromUserName"));
            }
            // 设备绑定/解绑事件可以回复空包体
            return "";
        }
    }
    // 收到设备消息
    if (MsgType.DEVICE_TEXT.equals(msgType)) {
        // Base64 解码
        byte[] reqRaw = Base64.decodeBase64(reqContent);
        // 反序列化
        BlueLight lightReq = BlueLight.parse(reqRaw);
        // 逻辑处理
        // demo 中 推送消息给用户微信
        String reqText = lightReq.body;
        System.out.println("recv text:" + reqText);
        String transText = "收到设备发送的数据：";

        byte[] reqTextRaw = reqText.getBytes("UTF-8");
        if (reqTextRaw.length > 0 && reqTextRaw[reqTextRaw.length - 1] ==
```

```
        0) {
            // 推送给微信用户的内容去掉末尾的反斜杠零'\0'
            transText = transText + new String(reqTextRaw, 0,
            reqTextRaw.length - 1, "UTF-8");
        } else{
            transText = transText + reqText;
        }
        // 推送文本消息给微信
        MpApi.customSendText(openID, transText);

        // demo 中 回复 收到的内容给设备
        BlueLight lightResp = BlueLight.build(BlueLight.CmdId.SEND_TEXT_RESP,
        reqText, lightReq.head.seq);
        // 序列化
        byte[] respRaw = lightResp.toBytes();
        // Base64 编码
        String respCon = Base64.encodeBase64String(respRaw);

        // 设备消息接口必须回复符合协议的 xml
        return XmlResp.buildDeviceText(toUser, fromUser, deviceType, deviceID,
        respCon, sessionID);
    }
}
```

公众平台 API 接口调用

除了对微信推送的消息接口进行被动响应，开发者也可以主动调用公众平台 API 接口与用 户和设备进行交互。

调用所有 API 接口时均需使用 https协议。

获取访问凭证 access_token

access_token 是公众号的全局唯一票据，公众号调用各接口时都需使用 access_token。获取访问凭证需要传入 appID 和appsecret。

access_token 有效期目前为 2 个小时，需要定时刷新，且刷新后将导致上次获取的凭证失

效。

由于获取访问凭证 API 接口有频率限制，频繁刷新可能导致超过限额，影响正常业务，因

此需要对访问凭证进行缓存。

所以需要对 access_token 的操作进行封装：

1、获取最新可用的 access_token

2、 access_token 过期前需要有自动刷新机制

3、 提供主动刷新接口给业务点调用，这是为了防止业务方的误刷新 access_token（如本地运行程序进行测试，导致服务器缓存的凭证失效），或者微信平台修改 access_token 的策略导致 access_token 提前过期

Demo 中：

AccessTokenUtil 类对 access_token 的访问进行了封装，使用定时任务进行刷新。

HttpUtil 类负责替换 url 参数中 access_token，使用 httpclient 类库发起 https 请求。并且在凭证无效时重新获取凭证并发起请求。

创建自定义菜单

Demo 中创建“点灯”、“灭灯”两个 CLICK 类型菜单。创建 CLICK 菜单时，需要指定每个菜单对应的 key 值，用于在服务端收到消息时区分用户点击了哪个菜单。

代码见 Tools 类中 createMenu 方法。

创建自定义菜单后，由于微信客户端缓存，需要 24 小时微信客户端才会展现出来。建议测试时可以尝试取消关注公众账号后再次关注，则可以看到创建后的效果。

更多类型菜单创建格式，见接口文档：

http://mp.weixin.qq.com/wiki/13/43de8269be54a0a6f64413e4dfa94f39.html

推送消息给用户微信

在收到消息接口推送时，不方便或无法在消息接口响应中进行处理时，可以通过客服消息接 口发送消息给用户。

在 demo 中，收到设备发送的消息后，响应内容只能回传给设备，这时还要发送消息给用户微信，就可以通过客服消息接口。

Demo 中发送的为文本消息，代码如下：

```
private static final String CustomSendUrl =
"https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=A
CCESS_TOKEN";
public static void customSend(String body) {
    System.out.println("customSend body=" + body);
    HttpUtil.doPost(CustomSendUrl, body);
}

public static void customSendText(String touser, String content) {
    JSONObject json = new JSONObject();
    json.put("touser", touser);
    json.put("msgtype", "text");
    JSONObject text = new JSONObject();
    text.put("content", content);
    json.put("text", text);
    customSend(json.toString());
```

```
}
```

更多客服消息类型见接口文档：

http://mp.weixin.qq.com/wiki/7/12a5a320ae96fecdf0e15cb06123de9f.html

推送消息给用户设备

用户通过微信公众号菜单控制设备过程：

1、 用户点击菜单

2、 微信推送菜单点击事件到开发者服务器

3、 开发者根据用户标识查询绑定的设备信息，未绑定则无法控制设备。

4、 构造要发送给设备的数据，转为二进制。

5、进行 Base64 编码。

6、调用发送设备消息 API 接口，发送消息。

7、微信客户端收到消息后，解码后发送给设备。

其中 3-6 步在开发者服务端完成。

Demo 中代码如下（CallbackService 类）：

```
// 菜单点击事件
if (MsgType.Event.CLICK.equals(event)) {
    // 根据 key 值判断点击的哪个菜单
    String eventKey = reqMap.get("EventKey");
    // 点灯/灭灯
    if (V1001_LIGHT_ON.equals(eventKey)
        || V1002_LIGHT_OFF.equals(eventKey)) {
        //是否点灯操作
        boolean open = V1001_LIGHT_ON.equals(eventKey);
        // 根据 fromUserName 获取绑定的信息
        Map<String, String> boundInfo = DBMock.queryBoundInfo(fromUser);
        // 未绑定
        if (boundInfo == null) {
```

```
            return XmlResp.buildText(fromUser, toUser, "未绑定");
        }
        String deviceType = boundInfo.get("deviceType");
        String deviceID = boundInfo.get("deviceID");
        String openID = boundInfo.get("openID");
        // 构造设备消息
        CmdId cmdId = open ? BlueLight.CmdId.OPEN_LIGHT_PUSH :
        BlueLight.CmdId.CLOSE_LIGHT_PUSH;
        byte[] respRaw = BlueLight.build(cmdId, null,
        (short)0).toBytes();
        // Base64 编码
        final String content = Base64.encodeBase64String(respRaw);
        // 推送消息给设备
        DeviceApi.transMsg(deviceType, deviceID, openID, content);
        // 回复
        boolean debug = true;
        if(debug){
            // 返回调试信息
            String debugText = "已发送" + (open ? "点灯" : "灭灯") + "消息:
            " + "deviceID 为" + deviceID + ",设备消息为" + content;
            return XmlResp.buildText(fromUser, toUser, debugText);
        }else{
            return "";
        }
    }
}
```

和设备联调

设备 id 由厂商指定，建议由字母、数字、下划线组成，以免 json解析失败。

生成设备二维码

调用 API 接口 create_qrcode，传入设备 id 获取对应的二维码生成串。 使用二维码生成串，

利用第三方库生成设备二维码图片。二维码需要在设备授权后才能进行 扫描绑定。

Demo 生成设备二维码代码见 Tools 类 createQrByDeviceId 方法。

进行设备授权

等到设备的参数确定后，就可以调用授权 API 接口对设备进行授权。 授权时需要的参数：设备 MAC 地址、加密 key（如果加密）、蓝牙类型、是否加密、连接策略、断开策略等。

设备授权后，还可以通过授权 API 更新授权属性。

在开发调试时，可以先用假的信息授权，测试服务端设备绑定事件的处理。在设备参数确定后，更新为正确的设备属性，再和设备联调。

授权后就可以通过微信扫描设备二维码，进行设备绑定以及与设备进行联调。

Demo 授权代码见 Tools 类 device_Auth 方法。

和设备联调测试：

首先，设备使用 DebugApp 调试通过，保证设备和微信客户端的通信正常，再和微信公众号进行联调测试：

1、 使用微信扫描生成的设备二维码，预期：关注公众号成功，微信收到欢迎提示语。服务器收到设备绑定推送事件，用户和设备绑定关系保存成功。

2、 手机打开蓝牙，在公众号界面等待设备连接成功。

3、 点击微信公众号自定义菜单：

点击“点灯”菜单，预期：设备收到消息，灯泡点亮。 点击“灭灯”菜单，预期：设备收到消息，灯泡熄灭。

4、 点击设备按钮，发送设备消息：服务器收到设备消息，推送文本消息给用户微信，预期：公众号界面显示消息：“收到 设备发送的数据：Hello, WeChat!” 同时回复应答消息给设备，预期：设备收到响应包，包体内容与设备发送的内容相同。

5、 取消关注公众号，服务器收到设备解绑推送事件，删除用户和设备绑定关系。

切换为正式服务号

开发调试通过后，在正式上线前，将服务器配置改为正式账号。

将 appID、appsecret 更新为正式服务号的值后，重新部署服务。 开发者中心—>修改开发者服务器配置：填入 Token 和服务器的回调地址。

配置示例如下图：

开发者中心

配置项　　接口报警

开发者中心 / 填写服务器配置

请填写接口配置信息，此信息需要你拥有自己的服务器资源。
填写的URL需要正确响应微信发送的Token验证，请阅读接入指南。

URL　　http://您的服务对应的域名/callback　　默认配置的是这个地址

请输入合法的URL

必须以http://开头，目前支持80端口。

Token　　wxboardplan　　默认配置的是这个token。如果有改动，请在相应的程序配置中同步修改

必须为英文或数字，长度为3-32字符。

什么是Token？

EncodingAESKey　　owbF581ue3Ai7qwWqDuHU28FZdLdhSX0Pzhs8M7d　43 /43　　随机生成

消息加密密钥由43位字符组成，可随机修改，字符范围为A-Z，a-z，0-9。

什么是EncodingAESKey？

消息加解密方式　　请根据业务需要，选择消息加解密类型，启用后将立即生效

- (●) 明文模式
  明文模式下，不使用消息体加解密功能，安全系数较低
- ( ) 兼容模式
  兼容模式下，明文、密文将共存，方便开发者调试和维护
- ( ) 安全模式（推荐）
  安全模式下，消息包为纯密文，需要开发者加密和解密，安全系数高