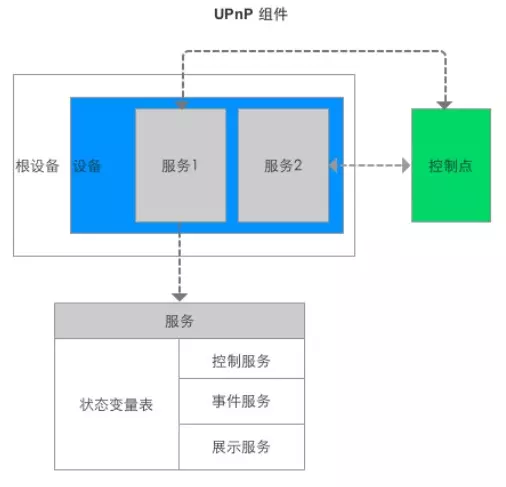
**基于DLNA协议的android设备投屏技术**

# 什么是DLNA

DLNA(Digital Living Network Alliance) 由索尼、英特尔、微软等发起成立、旨在解决个人PC，消费电器，移动设备在内的无线网络和有线网络的互联互通。  
DLNA 2003年诞生是为了能够把当时相互独立的电视机及其相关设备（功放、音响、DVD/蓝光播放机）、PC及其相关设备（数码相机、音乐播放器）和移动电话在家庭中有机地整合起来，使得多媒体内容能够在各个设备上便捷地播放

# UPnP组件

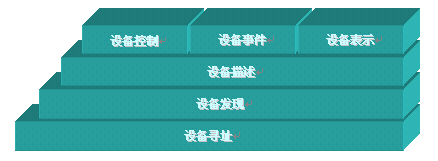


DLNA 标准包括多项协议及标准，其中最重要的部分是 UPnP 协议

**UPnP协议栈：**  
UPnP是由“通用即插即用论坛”（UPnP Forum）推广的一套网络协议，它的目标是实现设备间网络互联。

UPnP协议定义了设备之间，设备和控制点，控制点之间通信的协议。

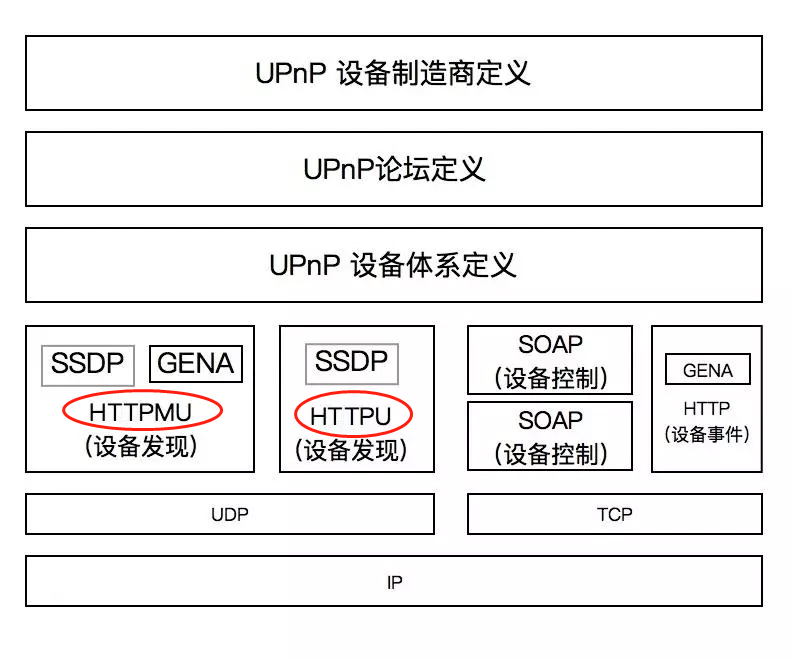
完整的UPnP协议栈由设备寻址、设备发现、设备描述、设备控制、事件通知以及基于Html的描述等几部分构成。



 设备发现：  
android设备投屏，首先android设备就要找到所有支持投屏的设备，找到设备的过程就是设备发现。

 设备控制：  
例如：android设备将视频播放投屏 到显示器上，设备需要对显示器上的视频进行 播放、暂停、进度拖拽等操作，这些操作就是设备控制。

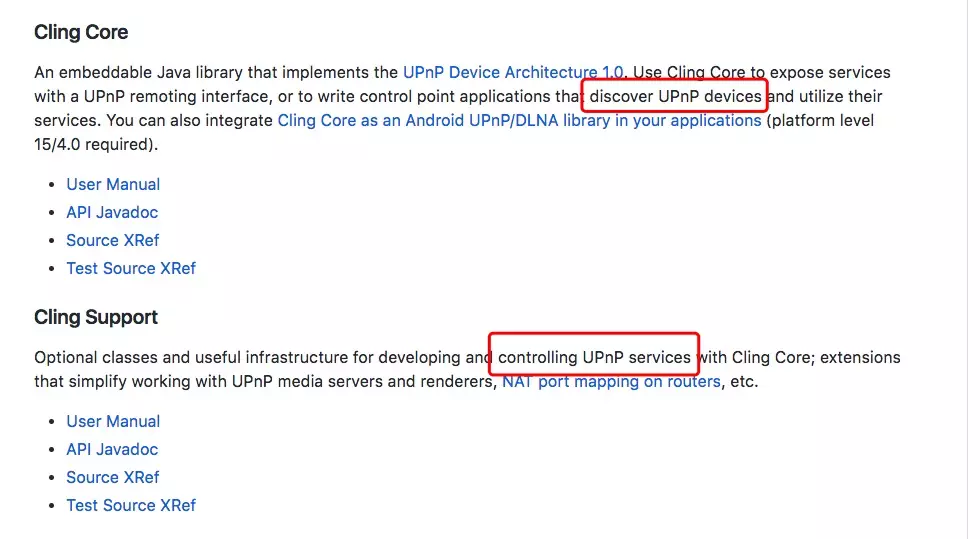
 设备事件：  
例如：android 设备需要随时知道 支持 投屏 设备在局域网中的出现，当出现一个新的设备 android 设备就通知到，要实现这点 android 设备就要订阅这个事件，就是设备事件。



1. 设备发现 SSDP 协议  
   简单服务发现协议（Simple Service Discovery Protocol：SSDP）
2. 控制设备 SOAP 协议  
   简单对象访问协议（Simple Object Access Protocol：SOAP）
3. 设备事件 GENA 协议  
   通用事件通知架构（Generic Event Notification Architecture：GENA）

# Cling开源库

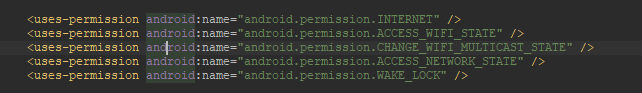
## <https://github.com/4thline/cling>



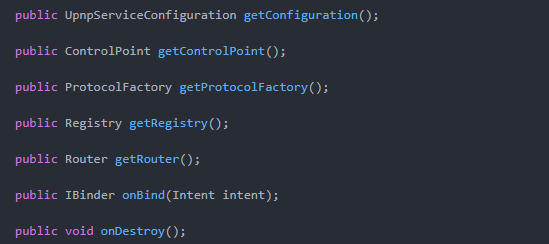
* core:发现设备部分
* support:控制设备部分

## 发现设备功能实现

首先在manifest 里加上以下权限（网络相关权限）：



然后在 manifest 里面定义这个AndroidUpnpServiceImpl，它有以下方法：



 ControlPoint getControlPoint() 获取到控制点，通过控制点执行 search() 方法就可以发现设备了

 然后就是 我们可以根据需求在 activity 里 bind 或者 unbind AndroidUpnpServiceImpl 这个 Service

在activity中使用要实现发现设备和设备监听代码：

 upnpService.getControlPoint().search(); //发现设备

 BrowseRegistryListener registryListener;  
upnpService.getRegistry().addListener(registryListener);//监听设备

## 发现设备源码分析：

upnpService.getControlPoint().search();

控制点执行serch方法，ControlPoint 的实现类是 ControlPointImpl:  
ControlPointImpl.search() 如下：

public void search() {

search(new STAllHeader(), MXHeader.DEFAULT\_VALUE);

}

STAllHeader：

public class STAllHeader extends UpnpHeader<NotificationSubtype> {

public STAllHeader() {

setValue(NotificationSubtype.ALL);

}

public void setString(String s) throws InvalidHeaderException {

if (!s.equals(NotificationSubtype.ALL.getHeaderString())) {

throw new InvalidHeaderException("Invalid ST header value (not "+NotificationSubtype.ALL+"): " + s);

}

}

public String getString() {

return getValue().getHeaderString();

}

}

NotificationSubtype.ALL = "ssdp:all" ssdp 这个就是发现设备的协议， ":" 后面就是一个筛选。继续返回到 search 中。ControlPointImpl.search() 实际调用的是

public void search(UpnpHeader searchType, int mxSeconds) {

log.fine("Sending asynchronous search for: " + searchType.getString());

getConfiguration().getAsyncProtocolExecutor().execute(

getProtocolFactory().createSendingSearch(searchType, mxSeconds)

);

}

getConfiguration() 返回的对象是 UpnpServiceConfiguration  
getAsyncProtocolExecutor() 返回的对象是一个执行者 Executor  
getProtocolFactory() 返回的对象是 ProtocolFactory

最后执行的是 ProtocolFactory.createSendingSearch() 方法进行的设备发现。

这个 UpnpServiceConfiguration 在 AndroidUpnpServiceImpl 中实际上是 AndroidUpnpServiceConfiguration。在 AndroidUpnpServiceImpl onCreate 时构造的。  
其实这个东西，它是用于配置环境的，比如 AndroidUpnpServiceConfiguration 它就用于配置 android 环境，比如一些网络、xml解析、全局使用的一些方法之类的。从这里面获取执行者(这个执行者是：ClingExecutor)， ClingExecutor 继承 ThreadPoolExecutor(线程池) 说明它就是执行线程池相关配置等工作的。如果我们有什么特殊的需要，也可以自己定义一个配置，然后增加一些自己需要的方法等。

在 发现设备 中，是通过获取 AndroidUpnpServiceConfiguration 的执行者，返回的就是这个 ClingExecutor

public Executor getAsyncProtocolExecutor() {

return getDefaultExecutorService();

}

执行过程的话，简单来说就是发一个指令（这个指令要么是 发现设备协议 要么是 控制设备协议 的指令），Executor 执行的 就是一个 Runnable 了。在 发现设备中 这个 Runnable 的实现类是 SendingSearch；控制设备 中实现类是 ActionCallback。

重要代码：getUpnpService().getRouter().send(msg);

在 AndroidUpnpServiceImpl 里，getRouter() 是 AndroidRouter 对象。  
这个消息 msg 是 OutgoingSearchRequest 的实例，OutgoingSearchRequest 里面就封装了 发现设备 的请求内容。

这些设备是保存在 RegistryImpl 中：  
保存的设备有两种：一个是 RemoteItems(远程设备，不是当前设备);另一个是 LocalItems(本地设备，就是当前设备)。他们就相当于列表。然后就是监听，发现设备之后 会回调到我们定义的监听。其实在 lan 层会截获到路由发的消息，然后会通知到我们。

## 发现设备总结：

 AndroidUpnpServiceConfiguration 它就用于配置 android 环境，比如一些网络、xml解析、全局使用的一些方法之类的。那么如果我们有什么特殊的需要，也可以自己定义一个配置，然后增加一些自己需要的方法等。

 ClingExecutor 是一个执行者，发现设备、控制设备命令都是由它来执行

 发现设备实际是通过 SendingSearch 来实现的，控制设备则是 ActionCallback

 SendingSearch 其实就是向路由发消息，告诉路由 我需要什么样的设备，路由会筛选，通过我们定义的监听返回给我们。

# 控制设备功能实现

 获取tv设备控制服务：通过选中的设备执行 device.findService(serviceType);

 获取控制点：通过执行 UpnpService.getControlPoint()

 执行指定控制命令：通过执行 ControlPoint.execute(命令)

所有的控制方法，包括：播放、暂停、停止等，都是通过这三步完成。  
我们通过绑定 Service，然后将 Service 绑定到 ClingUpnpServiceManager中，ClingUpnpServiceManager 是一个单例对象，因为 Service 里面有获取控制点等方法，绑定到 ClingUpnpServiceManager 之后，我们可以直接通过 ClingUpnpServiceManager 来获取控制点等。这样更方便。  
同时，在 ClingPlayControl 中封装了控制设备的各种方法(例如：播放、暂停...)

# 控制设备源码分析

Service service = device.findService(serviceType);

这段代码返回的是Service， serviceType 就是服务类型，比如 AVTransport ...

执行一个控制设备播放操作：

controlPoint.execute(new Play(avtService)

controlPoint.execute(..) 是通过 ExecutorService.submit(...) 执行的。 最后的执行者是 ClingExecutor。  
ClingExecutor extends ThreadPoolExecutor。而 ThreadPoolExecutor 也是继承于 ExecutorService。那么 最后执行的就是 ClingExecutor.submit(Runnable)。submit 里面的参数类型是 Runnable。这个 Play 就是一个 Runnable

public abstract class Play extends ActionCallback {…}

public abstract class ActionCallback implements Runnable {…}

所有类似 Play 的指令都是继承ActionCallback

这些指令最后的执行是 ActionCallback 的 run 方法；在 run 方法中执行了 SendingAction 的方法来发送指令  
SendingAction 它也是继承 Runnable，它最后通过 sendRemoteRequest 这个方法完成指令的发送。  
SendingAction prot = getControlPoint().getProtocolFactory().createSendingAction(actionInvocation, controLURL);

prot.run();

getControlPoint()//获取到控制点。

getProtocolFactory() //返回一个协议工厂。

public SendingAction createSendingAction(ActionInvocation actionInvocation, URL controlURL) {

return new SendingAction(getUpnpService(), actionInvocation, controlURL);

}

Run方法执行了execute，execute 最后执行的是 SendingAction 的 executeSync()

protected IncomingActionResponseMessage executeSync() throws RouterException {

return invokeRemote(getInputMessage());

}

invokeRemote 中手机发送控制命令给投屏端的方法就是 sendRemoteRequest

protected IncomingActionResponseMessage invokeRemote(OutgoingActionRequestMessage requestMessage) throws RouterException {

StreamResponseMessage streamResponse = sendRemoteRequest(requestMessage);

..

}

protected StreamResponseMessage sendRemoteRequest(OutgoingActionRequestMessage requestMessage){…

getUpnpService().getConfiguration().getSoapActionProcessor().writeBody(requestMessage, actionInvocation);

return getUpnpService().getRouter().send(requestMessage);

…

}

writeBody(requestMessage, actionInvocation)//写入控制命令  
getUpnpService().getRouter().send(requestMessage);//通过路由发送出去

## 控制设备总结：

 ControlPointImpl 是控制点的实现类，它有一个 execute 的方法，来执行控制命令。

 执行 Play 其实就是执行控制点的 execute(ActionCallback callback) 这个方法  
ActionCallback 它继承 Runnable，所有类似 Play 的指令都是继承它的。这些指令最后的执行是 ActionCallback 的 run 方法；在 run 方法中执行了 SendingAction 的方法来发送指令，SendingAction 它也是继承 Runnable，它最后通过 sendRemoteRequest 这个方法完成指令的发送。