SIProp

Androidとその可能性

SIProp Project
代表：いまむらのりつな

自己紹介

SIProp

- 氏名：今村謙之（いまむらのりつな）
- 年齢：29歳＋18ヶ月
- IT業界歴：約7年
- SIP歴：約5年
  - SIPropプロジェクト運営中
- 得意言語：Java、C
- 得意分野：SIP、ネットワーク層
  （開発～運用、セキュリティー）

- 特記事項：PCサーバ タワー8台運営中(自宅にて)

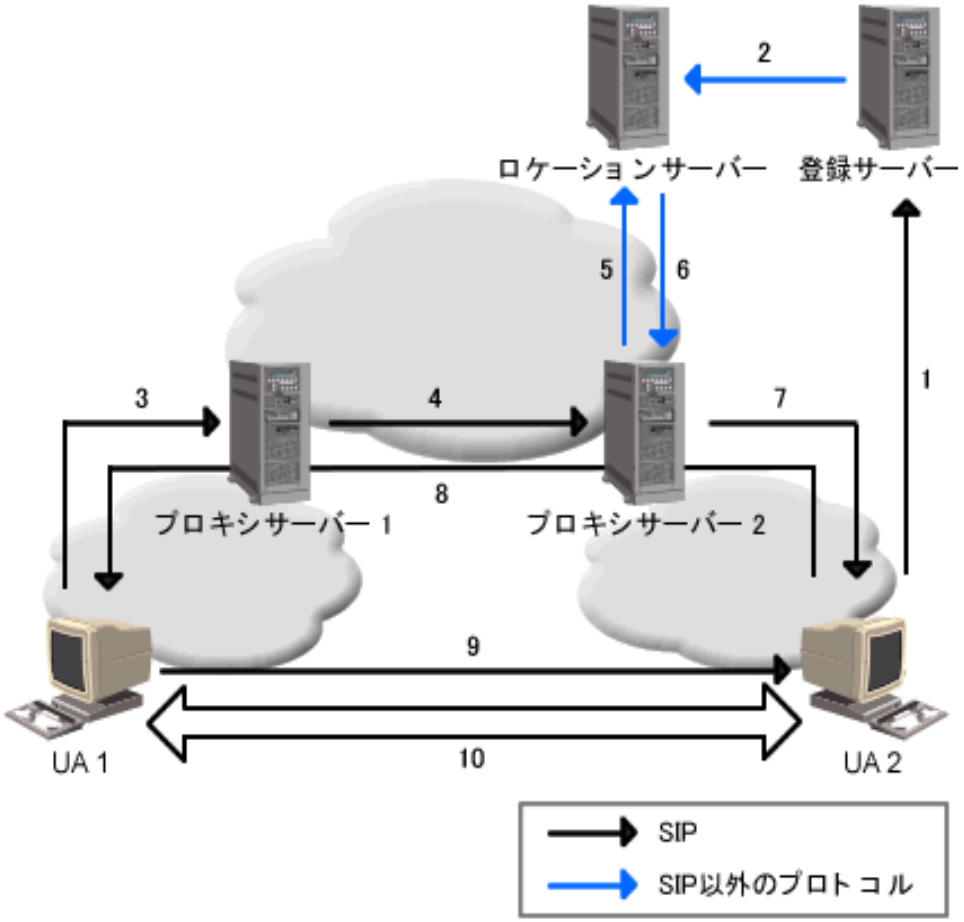
Agenda

- デモ
- なぜ？
- 事例
- ポイント

SIProp

シグナリングとセッション

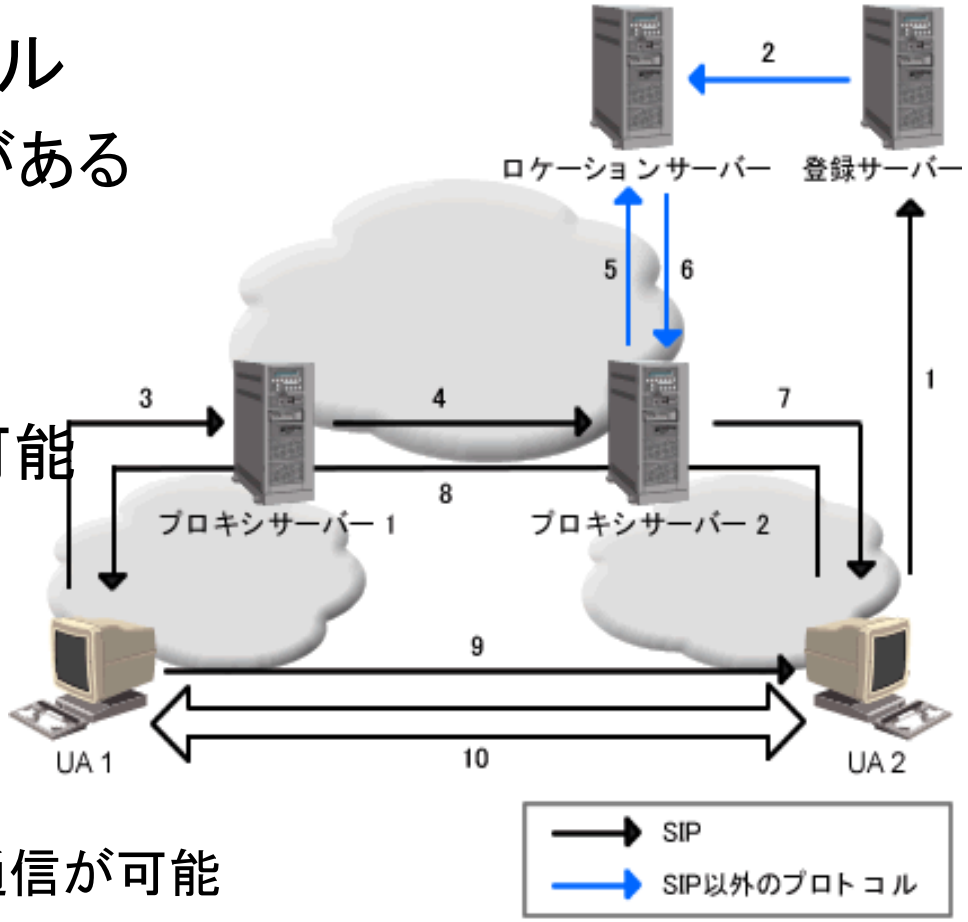
- シグナリング(呼制御)とは？
  - UA間などでロケーション情報をやり取りして、ユーザー間にセッションを確立する仕組み
- セッションとは？
  - メディアの送信側と受信側の組み合わせ、および送信側から受信側へのデータストリームの流れ



出典：ソフトフロント社HPより

HTTPとSIPの違い

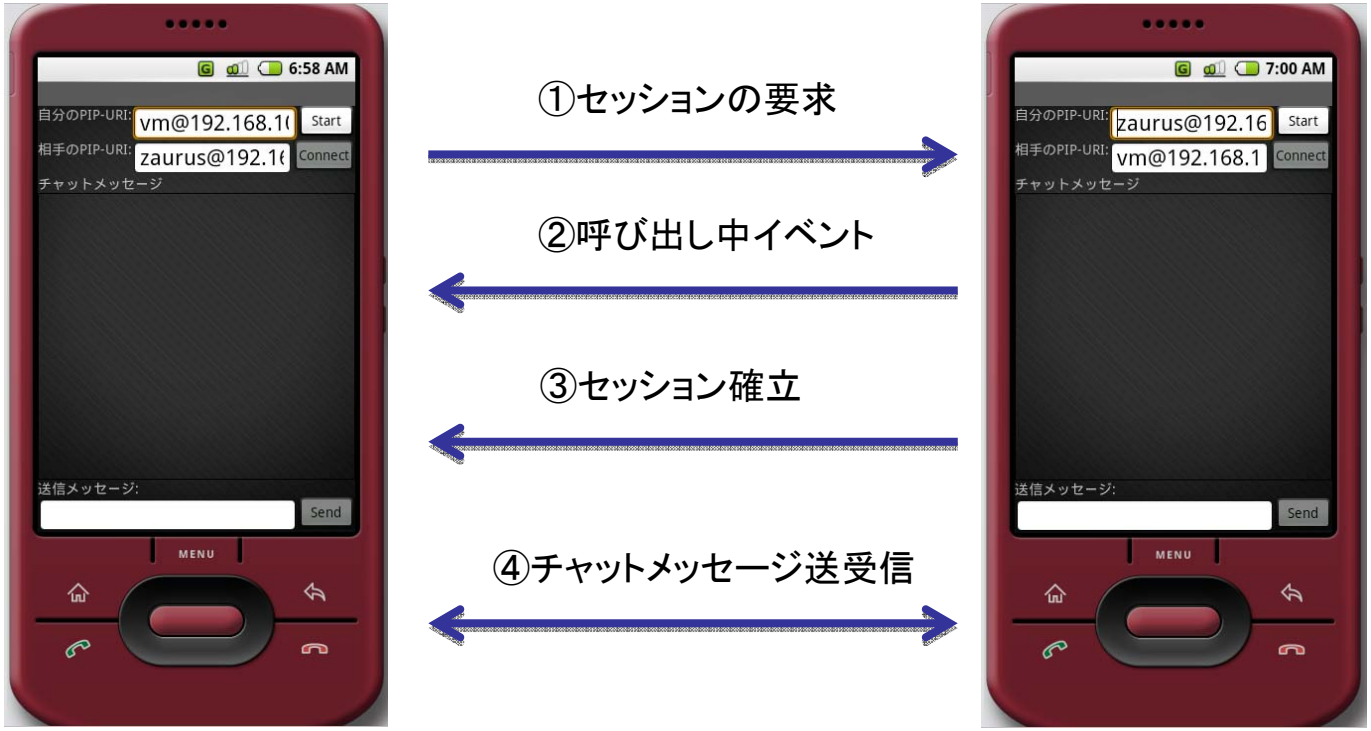
- SIPの特徴(HTTPとの相違点)
  - ステートフルプロトコル
    - クライアントの状態がある
  - クライアント間通信
    - 相手からの通知が可能
      - Cometのようなもの
  - ロケーション情報
    - ルーティング可能
      - クロスドメイン間の通信が可能
      - JSONPのようなもの



出典:ソフトフロント社HPより

デモアプリ概要

- Android用のチャットアプリケーション
  - 特徴
    - Android(=Peer)間で直接通信する
    - Android(=Peer)間でセッションを確立する



SIProp

SIProp

デモ見てね（はぁと

なぜ、こんなものを？ 1/2

SIProp

- Androidの適用分野
  - スマートフォン、携帯ゲーム機
    - Windows Mobile や iPhone(iPod touch) の世界
      - サービス
      - ユーザエクスペリエンス
  - 低価格携帯電話
    - 世界レベルで携帯電話が普及する世界
      - 丸山先生の資料参照のこと
  - 家電向けプラットホーム
    - 本格的に無線デバイスがPC化する世界
      - プラットホームの統一化
      - 台数の桁が違う

なぜ、こんなものを？ 2/2

SIProp

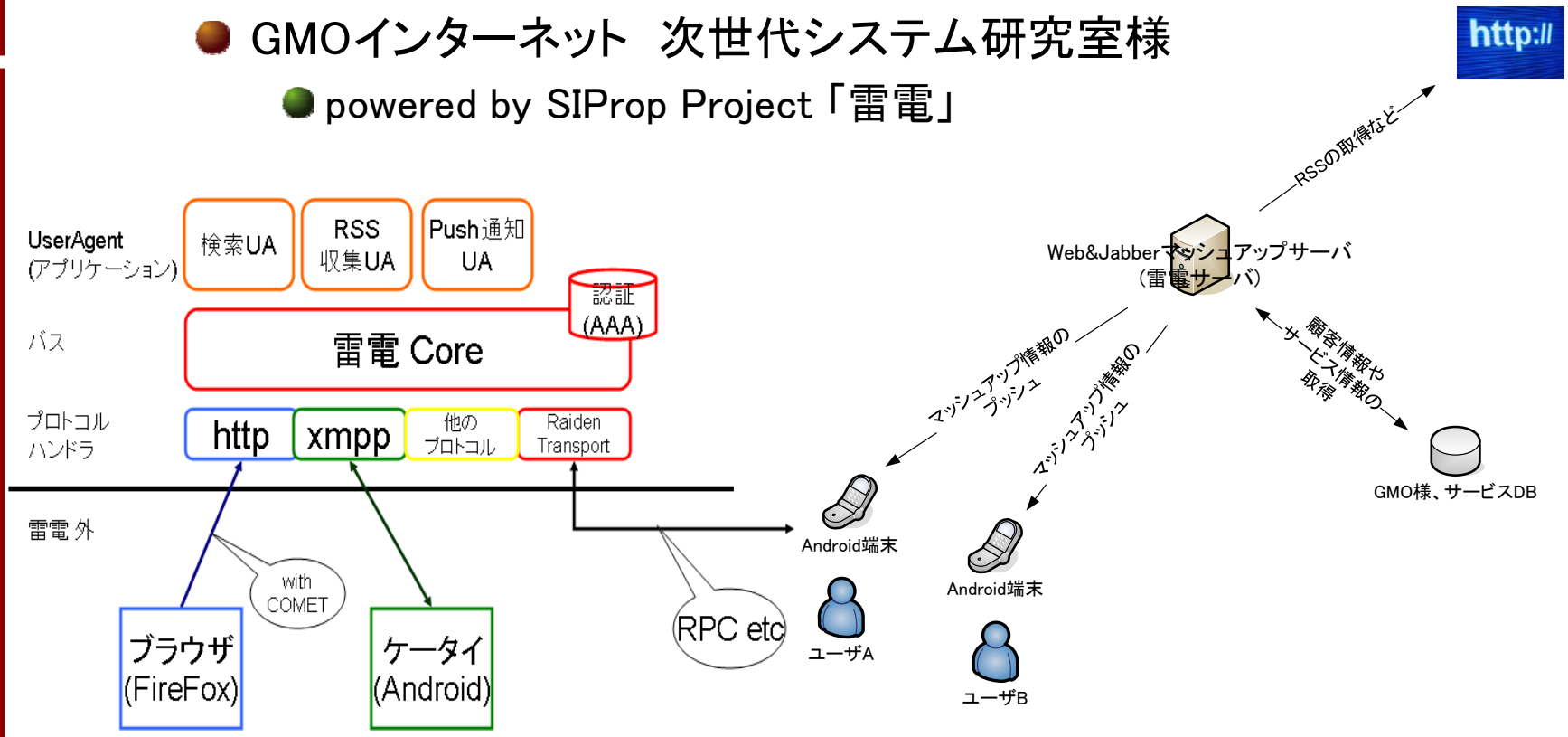
- ハードウェアとソフトウェア(物理領域)の融合
  - 超低価格デバイス
    - 無線デバイス版の100ドルPCのようなもの
  - 単一機能デバイス
    - 液晶が無くスピーカーだけのデバイスで、タクシーが近づいてきたら、タクシーからの無線を受けて、Beep音がなるデバイス
- 無線デバイスのメリット
  - 移動し、肌身離さず持っている
    - 車車間通信による、事故の回避用のデータ
      - 局所的なデータ処理だけできればよい

⇒アドホックに通信することにメリットがある！

⇒デバイス間SNSを作りたい！

応用事例：概要

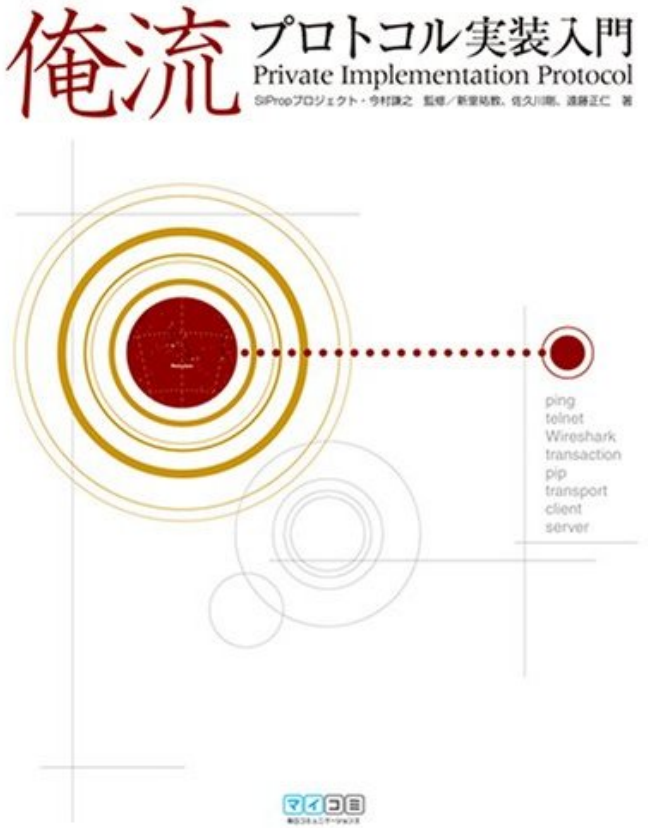
- リアルタイム・イベント配信
  - Webサービスなどから拾った情報を、リアルタイムにAndroidへ配信するシステム
- 共同開発
  - GMOインターネット　次世代システム研究室様
    - powered by SIProp Project「雷電」



CMです。

SIProp

- 書籍
  俺流プロトコル実装入門
  - 内容
    - nRFCの定義
    - Stackの設計・実装
    - Stackを利用したIMアプリ実装
  - 著作
    - SIProp Project
  - 発売中
    - 約500ページ
    - 約4000円



開発Tips 1/2

SIProp

- デモアプリを作る上で、はまった箇所の解説
- 注意点
  - 基本的なAndroid開発知識がある
  - ActivityやViewなどのGUI系の話はない
  - よって、Android開発入門的な話はありません
  - 入門的な話は、『Android勉強会』で！

開発Tips 2/2

SIProp

- jarの利用
  - Java5.0でコンパイルしたjarのライブラリ
- Serviceプログラミング
  - Activityとの連携方法
- Socket, ServerSocketプログラミング
  - 複数スレッドの扱い方
- 直接通信環境の構築
  - qemu の tap 機能の利用
  - Linux Zaurus 上 で、Android

jarの利用

SIProp

- 書籍のCD-ROMに付属しているjarをリンク
  - Java5.0でコンパイルしたもの
- 結果
  - 何事もなく、利用可能だった。
  - いうことなし。<(_ _)>

```
java.io.InputStream;
java.io.OutputStream;
java.io.ByteArrayOutputStream;
java.io.Serializable;

java.net.InetAddress;
java.net.ServerSocket;
java.net.Socket;
```

```
java.util.Hashtable;
java.util.HashMap;
java.util.ArrayList;
java.util.LinkedList;
java.util.Iterator;
java.util.ListIterator;
java.util.Collections;

java.util.regex.Matcher;
java.util.regex.Pattern;

java.util.Random;
```

Serviceプログラミング 1/4

SIProp

- Serviceとは？
  - バックグラウンドプロセス
    - デーモンのようなモノ
  - 使用例
    - 音楽プレーヤー
- 今回のお題
  - AIDL(Android Interface Definition Language)を介して通信するIPC(Inter Process Communication)プログラミング
    - 基本的にはプリミティブな型しかつかえない
      - CPU依存、プログラム依存を排除する

Serviceプログラミング 2/4

SIProp

- AIDLファイルの定義
  - 要は、コールバック用のAPIを書いたモノ
  - Activity用とService用の2つが必要

AIDLファイルの中身:

```
interface IIMServiceCallback {
        void onNewSession(String id, String session_id, String fromAddr);
        void onProvisional(String id);
        void onConnected(String id);
}
```

自動生成されたJavaファイルの一部:

```
public interface IIMServiceCallback extends android.os.IInterface {

    /** Local-side IPC implementation stub class. */
    public static abstract class Stub extends android.os.Binder implements
    org.siprop.pip.android.service.IIMServiceCallback {

        private static final java.lang.String DESCRIPTOR =
        "org.siprop.pip.android.service.IIMServiceCallback";

        /** Construct the stub at attach it to the interface. */
        public Stub() {
```

Serviceプログラミング 3/4

SIProp

- Activityでやること
  - Service側のAIDLを呼び出して、Activity側のAIDLを登録する

Activity側でのAIDLインプリメント:

```
private IIMServiceCallback mCallback = new IIMServiceCallback.Stub() {
        public void onNewSession(String id, String session_id, String fromAddr)
                throws DeadObjectException {
            mHandler.sendMessage(……
        }
```

Activity側でのAIDL呼び出し:

```
protected void onCreate(Bundle icicle) {
    super.onCreate(icicle);
      // サービスを起動する。
    bindService(new Intent(IIMService.class.getName()),
            mConnection, Context.BIND_AUTO_CREATE);

    protected ServiceConnection mConnection = new ServiceConnection() {
            public void onServiceConnected(ComponentName className, IBinder
        service) {
            mService = IIMService.Stub.asInterface(service);

            mService.registerCallback(mCallback);
```

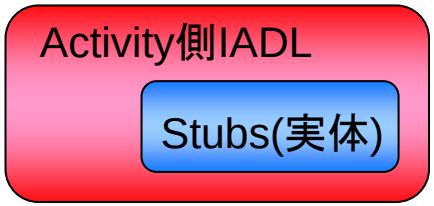
Serviceプログラミング 4/4

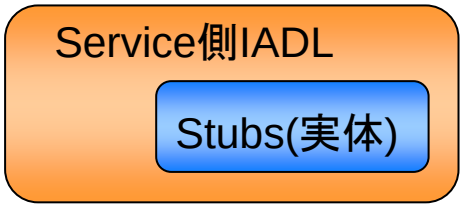
SIProp

Activity側(クラス)

Service側(クラス)

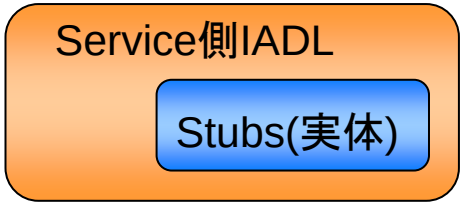
①Stubを用いて、AIDLインタフェースを無名クラスとして実装





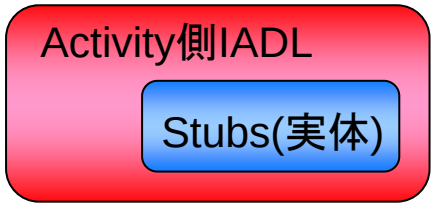
②ServiceConnection#onServiceConnectedの実装

③Service側のIADL実体を受け取る。
IMService.Stub.*asInterface()*



④Service側のIADL実体を受け取る
mService.registerCallback(mCallback);

＊正確には、独自で実装する必要がある



Socketプログラミング 1/3

SIProp

- java.netパッケージをそのまま利用可能
  - TCPによる接続
- 最大同時4接続
  - 複数のスレッドが動く

ServiceやActivityは、インスタンスを
生成したスレッド以外のスレッドから
操作することは出来ない！！！

- 生成スレッドにアタッチする必要がある！
- Handlerクラスを使用する

Socketプログラミング 2/3

SIProp

- Agent(Socket待ち受けスレッドを保持しているクラス)のコールバック用メソッドを変更
  - コールバック対象のインスタンスを、Handerにする
  - コールバック用のメソッドとして、Handler#sendMessage(Message)を用いる。

変更前:

```
public void incomingINVITE(Call c, PIPRequest request) {
    listener.onNewSession(c.getCallId(), request);
}
```

変更後:

```
public void incomingINVITE(Call c, PIPRequest request) {
  handler.sendMessage(
     handler.obtainMessage(0, 0, 0,
       new EventMessage(IMEvent.onNewSession,
         new Object[]{c.getCallId(), request})));
}
```

Socketプログラミング 3/3

SIProp

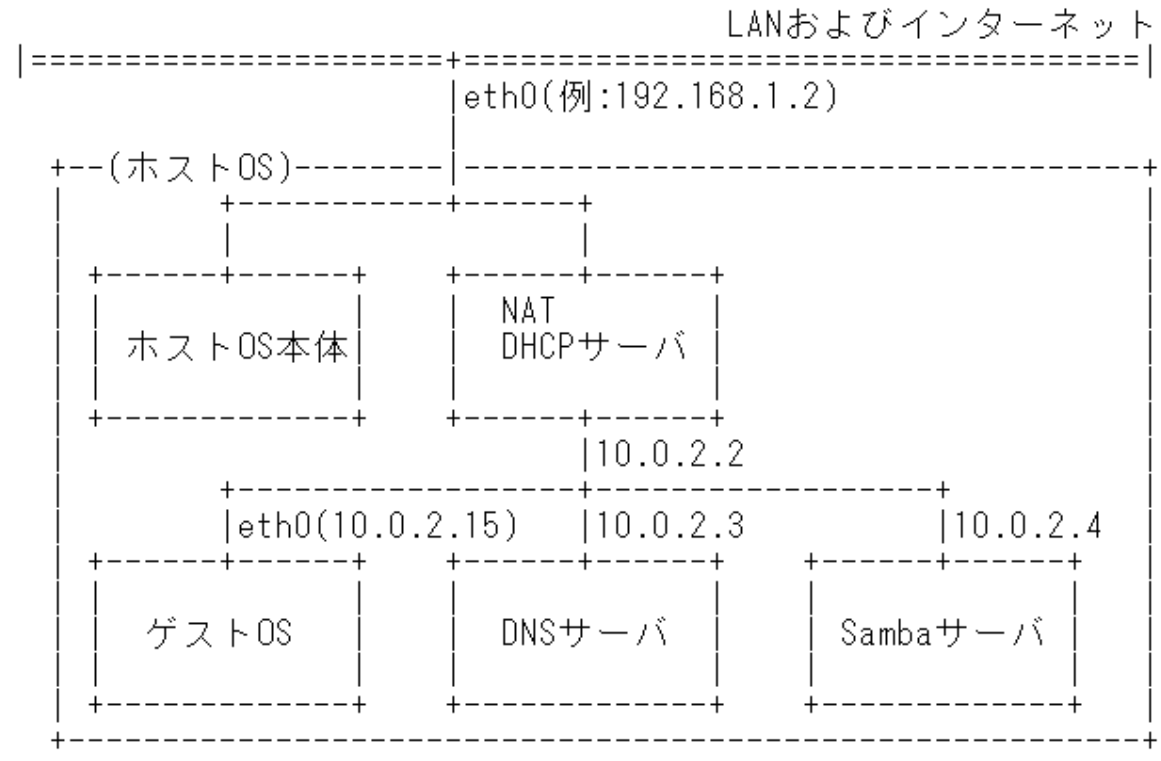
- Service内にHandlerを生成して、ここでコールバック先のメソッドを選択するように変更
  - Handlerは、Serviceのインスタンス生成時に生成する必要がある

```
##Service内
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message handlerMessage) {
        // EventMessageを取り出す。
        EventMessage msg = (EventMessage)handlerMessage.obj;
        Object[] objs = msg.getArgs();
        // イベントの種類により、IMServiceのコールバック先メソッドを選択する
        switch (msg.getEventType()) {
            case onNewSession:
                // 引数と共に、コールバックする
                imService.onNewSession((String)objs[0], (PIPMessage)objs[1]);
                break;
(以下略)
```

直接通信環境の構築 1/3

SIProp

- qemu の tap 機能の利用
  - ホストOS上の tap デバイスとブリッジすることにより、ゲストOS(Android)のNICが、あたかもホストOS側のNICのように使用できる機能



直接通信環境の構築 2/3

SIProp

1. Android上のNIC設定の書き換え
   1. etc/qemu-init.sh の編集
      - 例:
        - ifconfig eth1 192.168.1.2 netmask 255.255.255.0 up
        - route add default gw 192.168.1.1 dev eth1
2. ホストOSの tap とブリッジ デバイスの設定
   - 例:
     - brctl addbr br0
     - tunctl -u $USER -t net_android
     - brctl addif br0 eth0
     - brctl addif br0 net_android
3. Androidのqemu起動オプションの変更
   - ${android_sdk}/tools/emulator -console -qemu -net user -net nic -net nic,vlan=1 -net nic -net tap,vlan=1,ifname=net_android

直接通信環境の構築 3/3

SIProp

- Linux Zaurus 上 で、Android
  - cortez氏が、開発しているZaurus上で動作するAndroidパッケージを利用
    - http://www.omegamoon.com/blog/static.php?page=ZaurusAndroid
- 特徴
  - Zaurus上に、専用のLinuxがインストールされる
    - Zaurus Linuxには、リーチ不可能
  - qemu上ではなく、chrootによるjail上で動作
    - ベースOS上のリソースを利用できる

SIProp

SIProp

ご静聴ありがとうございました。

<(_ _)>

http://www.siprop.org/

Blog: http://noritsuna.siprop.org/