|  |
| --- |
| 博士課程教育リーディングプログラム  実世界データ循環学リーダー人材育成プログラム |
| データツールファースト |
| Android入門 |

|  |
| --- |
| 名古屋大学  2014/02/27 |

* Androidは、Google Incの商標または登録商標です。
* Linuxは、Linux Torvaldsの米国およびその他の国における登録商標あるいは商標です。
* その他会社名、各製品名は、一般に各社の商標または登録商標です。

目次

[1. はじめに 1](#_Toc381557409)

[1.1 Androidとは 1](#_Toc381557410)

[1.1.1　Androidの歴史 1](#_Toc381557411)

[1.1.2　アプリケーション実行環境とは 2](#_Toc381557412)

[1.1.3　Androidのバージョン 2](#_Toc381557413)

[1.1.4　バージョン毎の割合 3](#_Toc381557414)

[1.1.5　Androidアプリケーションの概要 4](#_Toc381557415)

[1.1.6　VMについて 4](#_Toc381557416)

[1.2 適用事例・活用例 5](#_Toc381557417)

[1.3　Androidアプリケーションの構造 6](#_Toc381557418)

[1.3.1　コンポーネントの実行 8](#_Toc381557419)

[1.3.2　コンポーネントの終了 8](#_Toc381557420)

[1.3.3　コンポーネントの情報 9](#_Toc381557421)

[1.3.4　インテントフィルタ 11](#_Toc381557422)

[1.3.4　タスク 11](#_Toc381557423)

[1.3.5　プロセスとスレッド 13](#_Toc381557424)

[1.3.6　RPC（リモート・プロシージャ・コール） 13](#_Toc381557425)

[1.3.7　スレッドセーフ 14](#_Toc381557426)

[1.3.8　コンポーネントのライフサイクル 14](#_Toc381557427)

[1.4　Android開発環境の構築 22](#_Toc381557428)

[1.4.1　Android SDKを使った開発環境の構築 22](#_Toc381557429)

[1.4.2　JDKのインストール 23](#_Toc381557430)

[1.4.3　AndroidSDKのダウンロードと展開 25](#_Toc381557431)

[1.4.4　SDKのセットアップ 27](#_Toc381557432)

[1.4.5　Eclipseの起動 28](#_Toc381557433)

[1.4.6　SDK　パッケージのインストール 29](#_Toc381557434)

[1.4.7　AVDの作成 30](#_Toc381557435)

[1.4.7　AVDの実行 32](#_Toc381557436)

[1.5　サンプルの作成手順の確認 33](#_Toc381557437)

[1.5.1　Helloworldの作成 33](#_Toc381557438)

[1.5.2　エミュレータでの実行 38](#_Toc381557439)

[1.5.3 実機での実行 41](#_Toc381557440)

[2. AndroidセンサAPIの構造 42](#_Toc381557441)

[2.1 Android APIの概略 42](#_Toc381557442)

[2.2 センサAPIの種類 44](#_Toc381557443)

[2.2.1 GPS 44](#_Toc381557444)

[2.2.2 各種センサ 45](#_Toc381557445)

[2.3　センサAPIの具体的な使用方法 47](#_Toc381557446)

[2.3.1　プロジェクトの作成 47](#_Toc381557447)

[2.3.2　画面デザイン 48](#_Toc381557448)

[2.3.3　リソースのインポート 49](#_Toc381557449)

[2.3.4　レイアウト 50](#_Toc381557450)

[2.3.5リソースの編集 53](#_Toc381557451)

[2.3.6　プログラムの編集 57](#_Toc381557452)

[2.3.7　サーバへの送信 63](#_Toc381557453)

[3　データ保存 73](#_Toc381557454)

[3.1　データ保存方法の検討 73](#_Toc381557455)

[4　ネットワーク通信 74](#_Toc381557456)

[4.1　サーバへの転送フォーマット 74](#_Toc381557457)

[4.2　認証 74](#_Toc381557458)

[4.3　同期管理 77](#_Toc381557459)

[5　その他(補足) 80](#_Toc381557460)

[5.1　Debug/DDMS 80](#_Toc381557461)

[5.1.1　デバッグ 80](#_Toc381557462)

[5.2　リソースの扱い（多言語化など） 84](#_Toc381557463)

[6. 参考文献 89](#_Toc381557464)

[７． 変更履歴 90](#_Toc381557465)

# はじめに

## Androidとは

すでにAndroidがスマートフォンで動作するOSであることはご存じの事と思います。

しかし、携帯電話のOSというだけでは、実際にAndroidとは何かということについて、何もわからないと思います。

AndroidはLinuxの上に独自のドライバを組み込んで、BionicなどのライブラリなどをベースにDalvikVM上に構築されたJavaアプリケーションの実行環境です。

Linuxは皆さんご存じの通りリーナス・トーバルスが開発したUnix互換なOSカーネルです。

世の中には多くのLinuxのディストリビューションが存在しますが、Androidも見方によってはそのディストリビューションの一つとも考えられます。

通常のLinuxディストリビューションはLinuxカーネル上にglibcなどのネイティブライブラリや各種アプリケーションがインストールされた状態で提供されています。

ユーザが使用するアプリケーションもネイティブアプリケーションが主で、それ以外にRubyやPythonといったスクリプト言語を使用したアプリケーションやその他のツールで構成されています。

AndroidはそういったLinuxのディストリビューションとは異なり、Linuxのカーネル上にBionicとよばれるFreeBSD由来のCライブラリを使用し、アプリケーションは基本的にJavaで開発されたものが提供される環境です。

　AndroidではDalvikVMというJavaVMが稼働しており、ユーザから見えるアプリケーションはそのほとんどがJavaで開発されています。

そこで、ここではAndroidの生い立ちから説明します。

### 1.1.1　Androidの歴史

2003年にPalo AltoにAndroid社が発足しました。そのAndroid社が開発した携帯デバイス用のアプリケーション実行環境がAndroidです。

2005年にGoogleがAndroid社を買収し、カーネルをLinuxとし、携帯電話のOSとして開発を継続しました。

2007年にはOHA（Open Handset Alliance)を設立し、広くハードウェアメーカーやキャリアに提供することを発表しました。

OHAの目的はオープンで標準的なモバイルデバイス用の環境を開発することでした。そして、2008年10月にT-Mobile G1という形で、初めてのAndroid携帯電話が発売されました。

日本では2009年7月にdocomo PRO Series HT-03Aという台湾HTC社製の端末が発売されました。

　その後Sony Xperiaシリーズの発売などを経て、日本でも多くのメーカーがAndroid端末を発売するようになり日本でも多くのユーザが使用するようになりました。

Androidのバージョンの歴史については後述します。

### 1.1.2　アプリケーション実行環境とは

一般にAndroidは携帯電話用OSと言われています。OSとは何でしょうか。OSとは「Operation System」の略なのですがそれだけでは何をするものか全くわかりません。OSとはハードウェアの違いを吸収し、アプリケーションに対して各種資源（メモリや画面、タッチパネル、センサなど）を使えるようにし、プロセス（プログラムの実行単位）を管理する為のソフトウェアの集合体を指します。

Androidは一般にOSと呼ばれるものにJavaの仮想マシンやBinderと呼ばれるRPC（Remote Procedure Call)などのドライバ、いろいろなデバイスやデータベースなどを利用するためのライブラリ、フレームワークと呼ばれるJavaライブラリを加えたものですので、より一層抽象化されたものと言えるでしょう。

そこで、本書ではOSの上にJavaアプリケーションを実行する環境を加えたものとしてAndroidをデバイス用アプリケーション実行環境と呼びます。

ここでいうデバイスとは、携帯電話、タブレット、その他の機器を総称したもので、汎用のパソコンとはちょっと違うコンピュータ機器を指します。

### 1.1.3　Androidのバージョン

2008年に公開された Android Version 1.0以降 Androidは頻繁にバージョンを更新してきました。主な更新は以下の通りです。

|  |  |  |
| --- | --- | --- |
| SDK公開日 | バージョン | コードネーム |
| 2009/2/9 | 1.1 |  |
| 2009/4/30 | 1.5 | Cupcake |
| 2009/9/15 | 1.6 | Donut |
| 2009/10/26 | 2.0 | Eclair |
| 2010/5/20 | 2.2 | Froyo |
| 2010/12/6 | 2.3 | Gingerbread |
| 2011/2/22 | 3.0 | Honeycomb |
| 2011/10/18 | 4.0 | Ice Cream Sandwich |
| 2012/6/27 | 4.1 | Jerry Bean |
| 2013/10/31 | 4.4 | KitKat |

Androidカーネルのソースコードの公開はSDKの公開からしばらく経ってから行われているので上記の日付ではまだAndroidカーネルのソースコードは公開されていません

この間にも細かなバージョンの更新（2.3から2.3.1,2.3.2・・・などのサブリビジョンと呼ばれる更新）も頻繁に行われています。

コードネームは1.5以降のバージョンにはCから順にDEFGHIJKで始まるお菓子の名前が付いています。

現在の最新版は　Version 4.4 (KitKat)です。

もちろんご存じの通りネスレのKitKatというチョコレートの名前です。もともとはKey Lime Pieという名前だったと言われていますが、そもそもキーライムパイの味を知っている人は少なすぎるのではないか？KといったらKitKatだろう。ということで、ネスレから正式にブランド名の使用許諾を得て4.4はKitKatとなりました。

広告やキャンペーンなどでご覧なった方も多いのではないかと思います。

### 1.1.4　バージョン毎の割合

日本では、DocomoのHT-03AがVersion 1.5(Cupcake)で発売されましたが、発売直後に1.6(Donut)に更新されましたので、現在1.5の端末は存在しないと考えてもいいでしょう。

実際AndroidマーケットにアクセスしたAndroidのバージョン別の割合を見ると2014/2/4時点では2.2未満のバージョンはほぼ０で、2.2ですら1.3％となっています。

http://developer.android.com/intl/ja/resources/dashboard/platform-versions.html

2014/2/4時点の各バージョンの割合

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Codename | API Level | 割合 |
| 2.2 | Froyo | 8 | 1.3% |
| 2.3.3 -2.3.7 | Gingerbread | 10 | 20.0% |
| 3.2 | Honeycomb | 13 | 0.1% |
| 4.0.3 -4.0.4 | Ice Cream Sandwich(ICS) | 15 | 16.1% |
| 4.1.x | Jelly Bean | 16 | 35.5% |
| 4.2.x | Jelly Bean | 17 | 16.3% |
| 4.3 | Jelly Bean | 18 | 8.9% |
| 4.4 | KitKat | 19 | 1.8% |

これを見るとほとんどの端末が2.3以上のバージョンになっていると言えますので、このコースではVersion2.3.3(Gingerbread)以上対応で進めます。

この中でVersion 3.2のHoneycombだけが極端に少ないのですが、これはHoneycombがTablet用として開発され携帯電話では採用されることが無かったこと

Googleが公開できるようなソースではないという理由でソースコードを公開しなかったこと、その時点ではまだタブレットというものはほとんど売れておらず、その後出てきたタブレットはほとんどが次期バージョンのICS以降を採用するようになったことなどが大きな原因です。

### 1.1.5　Androidアプリケーションの概要

これまでにも簡単に紹介しましたが、Androidのアプリケーションは一部の例外を除いてほとんどJavaという言語で作成します。

Javaは旧Sun Microsystems（現Oracle)が開発した言語です。本書はJavaの入門書ではありませんので、Java言語についてのまとまった説明は行いません。

（ご存じとは思いますが、Webアプリの作成などで使用されているJavaScriptとは全く異なる言語です。）

Javaの基本的な構文はC言語に近く、オブジェクト指向の言語ですので、これまでにC++などのオブジェクト指向の言語を使用されたことがある方ならそれほど違和感無くプログラムすることが出来るのではないでしょうか。

また、Javaではクラスやインタフェースというオブジェクト指向の概念を多用します。説明の中でも用語としてインスタンス、継承、オーバーライドなどのオブジェクト指向の用語を用いますので、オブジェクト指向についてある程度理解しておいていただく必要があります。

加えてAndroidのアプリケーションではリソース（データ）を表現するのにXMLを多用します。後で簡単に説明をしますが画面レイアウトや文字列の定義、テーブルやデータベース的なものを含めてXMLのファイルとして作成することが多いので、XMLの文法についても簡単に理解しておく必要が有ります。

### 1.1.6　VMについて

　Javaはソースコードをバイトコードと呼ばれる中間形式にコンパイルしVirtual Machine上で実行するのが普通です。

　現在のAndroidではGoogleが開発したDalvikVMと呼ばれるVirtual Machine上で実行されます。

　DalvikVMの中間コードはOracleの提供するJavaVMとは異なりますのでAndroidアプリケーションをそのまま別のJavaVM上で実行することはできません。

## 適用事例・活用例

　このコースではAndroid端末に内蔵されるGPSや各種センサなどを使用したアプリの作成を中心に進めます。

　GPSはほとんどの端末に搭載されており、Locationマネージャとして独立したAPIが用意されています。

　それ各種センサはSensorマネージャとしてまとめられており、内蔵されているセンサは端末によって異なりますがここではAndroidが標準APIで対応している以下のセンサの使い方を中心に説明します。

　・Accelerometer （加速度センサ、重力の影響を含む）

　・Temperature　（温度センサ）

　・Gravity　（重力センサ）

　・Gyroscope　（回転センサ）

　・Light　（照度センサ）

　・Linear Accelerometer　（加速度センサ、重力の影響を含まない）

　・Magnetic　Field (磁場センサ)

　・Orientation (方位センサ)

　・Pressure　（気圧センサ）

　・Proximity　(近接センサ)

　・Relative Humidity (相対湿度センサ)

このコースではNexus７に搭載されているGPSと重力センサを用いてそのセンサ情報をサーバに転送するアプリを作成します。

転送する情報は時刻、位置情報、重力加速度となります。

また本体を固定して、加速度センサの情報を収集することで簡易な地震計として動作させることなども可能かもしれません。

端末によっては上記の通り気圧センサなどを搭載しているものなどもありますので、それを使って時間毎の気圧変化などを記録したりすることも可能です。

## 1.3　Androidアプリケーションの構造

　Androidのアプリケーションは以下の4種類のコンポーネントで構成されます。

　アプリケーションにはこれらの部品が最低限一つ以上含まれている必要は有りますが、一つのアプリケーションに複数のコンポーネントを組み込むことに制限はありません。

・アクティビティ(Activity)

　ユーザが操作を行う為に画面に表示される単位で、主にユーザインタフェースが必要なアプリケーションで使用します。一つの画面が一つのアクティビティだと思ってかまいません。複数の画面を使用するアプリケーションの場合は複数のアクティビティを使用することも、一つのアクティビティで表示内容を変えることで実行することもできます。ただし一つの画面でいくつもの表示を持たせてしまうのはプログラムが必要以上に複雑になりますので、似たような画面はかまいませんが根本的に違う操作をするような画面では別のアクティビティにした方がいいでしょう。

・サービス(Service)

　ユーザインタフェースとしての画面表示を持たず、バックグラウンドで黙々と処理だけを行うものです。たとえば音楽再生など一度再生を開始したら、別のアプリケーションに切り替えても再生を継続しておいて欲しいようなものや、定期的にインターネットにアクセスし情報をとってくるようなものが考えられます。

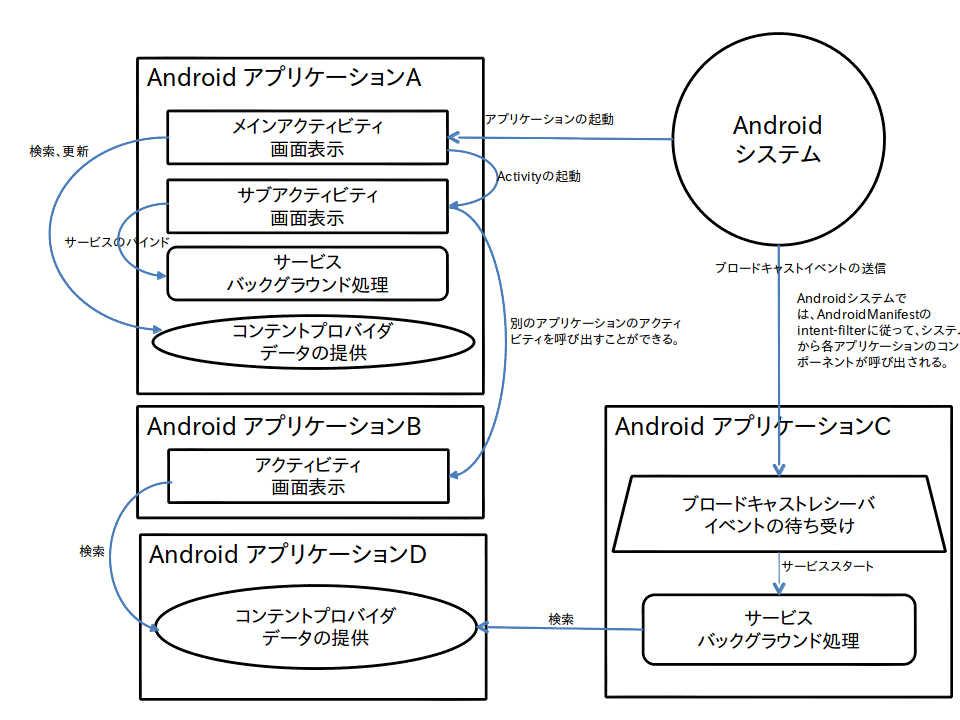
　ただし、サービスは自分で終了しない限り、よほどの事がなければ終了されませんので、メモリを占有しますし、電池の消耗をまねく可能性もありますので、注意が必要です。

・ブロードキャストレシーバ(BroadcastReceiver)

　「Androidの起動が完了した」、「電池の残量が減った」、「カメラで撮影が行われた」など、システムで何かが行われたときに発せられるメッセージや、アプリケーションから発行されるメッセージなどのブロードキャストメッセージを受信して処理を行うコンポーネントです。ブロードキャストレシーバを登録することで、たとえばAndroidの起動が完了した時、必ず実行させるような処理などを実現することができます。

・コンテントプロバイダ(ContentProvider)

　アプリケーションが管理しているデータを他のアプリケーションなどから使用できる形で提供することができます。この機能を使うことであるアプリケーションで作成されたデータを他のアプリケーションで参照したり更新したりすることが出来ます。



この図は複数のコンポーネントが連携して処理を実行していることを表しています。

ユーザがアイコンをタッチするなどしてアプリケーションAを起動します。

アプリケーションCはタイマーイベントや起動イベントなどの内部イベントに従って自動的に起動されるブロードキャストレシーバです。

ブロードキャストレシーバは起動して何かの処理を行ってすぐ終了することもできますし、内部でサービスを起動し常駐して処理を行うことも可能です。

アプリケーションAは起動されるとメインのアクティビティを開きユーザの操作を待ちます。操作によっては別の内部のアクティビティを起動して画面をきりかえたり、アプリケーション内部でバックグラウンド処理を行うサービスを起動したりすることもあります。

また内部でデータベースを使用したい場合はアプリ内部にコンテントプロバイダを起動しデータベースを管理することもできます。

アプリケーションAは処理の内容によってはほかのアプリケーションBが提供するアクティビティを起動することもあります。

よくあるのはブラウザの画面をほかのアプリから起動したりする場合です。

また公開しているコンテントプロバイダをもつアプリケーションDはほかのアプリケーションからのデータベースの問い合わせなどにも答えることができます。

このように、いろいろな処理を行う複数のアプリケーションが連携して、サービスを提供しているのがAndroidアプリケーションです。

### 1.3.1　コンポーネントの実行

　アクティビティ、サービス、ブロードキャストレシーバはインテントと呼ばれるメッセージを受信して実行されます。

　たとえば、ブラウザを使用してあるURLのページを表示場合はページのURLを保持するインテントオブジェクトを生成して、「startActivity」メソッドなどに渡します。「startActivity」メソッドはRPC（Remote Procedure Call)を使用して、そのインテントを処理することができるアプリケーションを探して、実行します。同じインテントを処理する複数のアプリケーションが存在する場合には、ユーザにどのアプリケーションを使用するか問い合わせます。同様にサービスを起動するためには、インテントを作成して「startService」メソッドを呼び出します。

　ブロードキャストレシーバを起動するには、同様に「sendBraodcast」メソッドをコールします。

　コンテントプロバイダはアプリケーションから「ContentResolver」のメソッドが実行されると起動します。

### 1.3.2　コンポーネントの終了

　コンテントプロバイダとブロードキャストレシーバはそれぞれのリクエストへの応答が終了すると自動的に終了します。

　アクティビティは、アクティビティ自身が「finish」メソッドを実行するか、呼び出した側から「finishActivity」をコールすることで終了します。

　サービスもほぼ同じで、サービス自身が「stopSelf」をコールするか外部から「stopService」をコールすることで終了します。

### 1.3.3　コンポーネントの情報

　アプリケーションがどのようなコンポーネントを含んでおり、どういったインテントに応答できるかは、マニフェスト(AndroidManifest.xml)というファイルに記載します。

　たとえば、アクティビティを持つアプリケーションでは、一般的に以下のようなマニフェストファイルを用意します。

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.sample"

android:versionCode="1"

android:versionName="1.0">

<uses-sdk android:minSdkVersion="4" />

<application android:icon="@drawable/icon" android:label="@string/app\_name">

<activity android:name=".SampleApplication"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

　マニフェストファイルは、XML(Extensible Markup Language)で書かれています。

　XMLは、エレメントとアトリビュートで構成され、コンテンツを含むことができます。コンテンツには別のエレメントや文字列を記述する事が出来ますので、エレメントを何重にも重ねて、複雑なデータ構造を表す事ができます。

　一つのエレメントは以下のような構文で表されます。

<エレメント名 アトリビュート名="値">コンテンツ</エレメント名>

　はじめの「<」と「>」で囲まれた部分を「開始タグ」最後の「</」と「>」で囲まれた部分を終了タグと呼びます。

　例外として、コンテンツが無い場合は

<エレメント名 アトリビュート名="値" />

　という書き方で終了タグを省略することができます。

　そのほかに特殊な表現として、「<?」と「?>」で囲まれた宣言と呼ばれるものと「<!--」と「-->」で囲まれたコメントがあります。

開始タグと終了タグは必ず１対１で対応していなければなりません。コンテンツ内のタグに対応する終了タグは必ずそのコンテンツを含んでいるタグの終了タグよりさきになければいけません。つまりAというエレメントがコンテンツにBというエレメントを含んでいる場合は、以下のように、Aの開始タグ、Bの開始タグ、Bの終了タグ、Aの終了タグの順番に書かなければいけません。

<A>

<B>

</B>

</A>

　この順番を間違っていると、XMLが正しく解釈できず、アプリケーションのビルドができません。

　この例では、いくつかの項目が@string/app\_nameの様になっていますが、これは、この内容をマニフェストファイルに直接記述するのではなく「res/string.xml」などのリソースファイルと呼ばれる別のファイルに記述してあるという事を表しています。

　リソースファイルは言語や画面サイズ、解像度などの種別毎に用意する事が出来るので、多言語対応や画面サイズの異なる端末にきめ細かく対応することが出来ます。

つまり、日本語の設定で使用しているときは、アプリケーションの名前として"サンプル"と表示され、英語で使用している時は"Sample"と表示される様な事ができます。

　また、iconなどの様に画像ファイルを指定しなければならない場合は、@drawable/iconという風に書くことでdrawableディレクトリ内のファイルを指定する事ができます。

　この例ではactivityエレメントのみですが、その他のコンポーネントもサービスならば<service>タグ、ブロードキャストレシーバは<receiver>タグ、コンテントプロバイダは<provider>タグを使用して定義します。アプリケーションに含まれるすべてのコンポーネントはマニフェストファイルに記述する必要があります。

　例外として、ブロードキャストレシーバについては、プログラム内部で「registerReceiver」メソッドをコールして登録することができますので、マニフェストファイルに記述しない事も出来ます。

### 1.3.4　インテントフィルタ

　先のマニフェストファイルに含まれる「activity」エレメントには「intent-filter」エレメントがあります。

「intent-filter」ではシステムに対して、このアクティビティがどういったインテントに応答することが出来るかということを指定します。

　このサンプルで記述しているインテントフィルタは、通常のアプリケーションではほとんど指定されているものです。

　11行目の「action」エレメントでは、このアプリケーションをシステムから起動する際には、このアクティビティから起動することを指定しています。

　12行目の「category」エレメントでは、このアクティビティをアプリケーションメニューに表示するべきものであるということを指定しています。

　たとえば一つのアプリケーションに複数のアクティビティが存在している場合、両方のアクティビティに上述の様なcategoryエレメントを記述しておくと、アプリケーションは一つなのに、システムのアプリケーションのリストには二つのアクティビティが表示される様になります。

　逆にどのアクティビティにもcategoryエレメントを記述しないようにしておくと、ランチャーにはアイコンが表示されず、直接実行することができないアプリケーションとなります。

　また、マニフェストファイルには複数のインテントフィルタを記述することが出来ます。インテントフィルタにはこのアプリケーションがどういった種類のインテントに対して、どういうアクションを行うことができるかということを記述します。

　Androidのシステムはインテントを処理する際に、マニフェストに記載されたインテントフィルタを調べて、どのコンポーネントがそのインテントに応答できるかを確認し、アクティビティやサービス、ブロードキャストレシーバを自動的に起動します。

### 1.3.4　タスク

　一般にコンピュータシステムにおいてタスクというと一つの処理の実行単位を指します。マルチタスクという言葉がありますが、これは複数の処理を平行して実行している状態を指します。

Androidではこのタスクとはちょっと違った意味でタスクという言葉を使います。Androidで言うタスクは、「ひとまとまりのアクティビティの集まり」を意味します。一つのタスクには複数のアクティビティが含まれますが、これは一つのアプリケーションに含まれるアクティビティに限るわけではなく、動作上ひとまとまりに見えるようなアクティビティの集合体を指します。

　こう書くとわかりにくいのですが、コンポーネントの説明の所で述べたようにアクティビティとは画面上でユーザインタフェースを司る単位でアクティビティから別のアクティビティを起動することができます。ユーザから見た場合、これは一連の画面遷移として見えますので、一つのアプリケーションを操作しているように見えるはずです。

　たとえば、あるアプリケーションで、webページを表示したいと思ったとき、アプリケーションの内部にブラウザアクティビティを作って表示する事も出来ますがただ単に表示するだけでいいのであれば、URLを持ったインテントを生成して外部のブラウザを開始することができます。そうすると、ブラウザが自動的に開始され、画面上にwebページが表示されることになります。

　このような場合、ユーザから見るとこれは一つのアプリケーション内の処理で、別の画面に切り替わっただけの様にみえるでしょう。こういった一連の画面遷移のかたまりをタスクと呼んでひとまとまりとしてあつかいます。

　一つのタスクは、ひとつのアプリケーションかのごとく動作します。上述のブラウザのアクティビティを表示している状態で、ホームボタンを押して、ホームに戻ると、ブラウザのアクティビティと元のアプリケーションのアクティビティは一緒にバックグラウンドに移動します。

　再度このアクティビティをフォアグラウンドに戻してきたときは、一緒にフォアグラウンドに戻ってきます。従って、戻ってきたブラウザのアクティビティで戻るボタンを押すと、元のアクティビティに戻ることになります。

　ある処理を行っている画面をちょっと置いておいて別の画面に切り替えるような処理をスタックにプッシュするという表現をします。このちょっと置いておいた画面の集まりをスタックと呼びます。

　前の画面に戻る処理で、ちょっと置いておいた画面を戻してくることをスタックからポップするといいます。スタックはプッシュまたはポップすることしかできません。そうでないと戻るというボタンを押したときどの画面に戻ったらいいのかわからなくなってしまいます。タスクというのはこの一つ一つのスタックの事を指していると言ってもいいでしょう。

　プログラムの現在の状態はこのスタック上に保持されていますので、複数のアプリケーションから同じアクティビティが呼び出されたとしても、プログラムとしては同じものが実行されますが、それぞれが持っているデータは異なったものになります。

### 1.3.5　プロセスとスレッド

　Androidではアプリケーションのコンポーネントを起動する際にLinuxのプロセスを一つ開始します。基本的に一つのアプリケーションはこのプロセスとして実行されますが、明示的に他のプロセスで実行するようにしたり、別のスレッドを起動したりすることもできます。

　ご存じのようにLinuxはマルチタスクが可能なOSです。Androidではアクティビティでユーザの操作に応答しますが、中には終了まで時間がかかる処理が必要になることがあります。たとえば、ネットワークを経由してサーバからデータを受信したり、SDカードにデータを保存したりする処理です。こういった時間のかかる可能性のある処理をユーザの操作を受け持っているアクティビティでそのまま処理してしまうと、処理が終了するまで、ユーザの操作に一切応答する事が出来なくなってしまいます。

　そういった状況では、ユーザは何も出来なくなり、大変イライラすることになります。そういったユーザの不満を解消する為に、時間のかかる処理を行う場合は、可能であれば、あとどのくらいで処理が終了するかをリアルタイムで表示するとか、どのくらいで終わるかがわからない場合でも、現在処理中であることを表示する必要が有ります。また、処理を途中で中断したり、別のアプリケーションに切り替えたりする事ができるように、ユーザの操作は常に有効にしておかねばいけません。そのために、時間のかかる処理は別のスレッドで実行し、アクティビティでは、常に処理スレッドの状態をモニターし、中断できる様な処理ならば、いつでもユーザが中断出来るようにしておく必要が有ります。

　スレッドが必要になった場合は「Java Thread」オブジェクトを使用してThreadを生成します。Androidではスレッド管理用のユーティリティとして「Looper」「Handler」「HandlerThread」などのクラスが用意されているので、必要に応じてこれらを使うことができます。

### 1.3.6　RPC（リモート・プロシージャ・コール）

　Androidでは、RPC(Remote Procedure Call)と呼ばれるアプリケーションとアプリケーションの通信の仕組みを持っています。色々な処理がRPCを使って実行されます。リモートプロセスはサービスで管理されるものがほとんどです。つまりRPCとはアプリケーションからサービスに対して要求を送信し結果を受信する仕組みだと思っていいでしょう。システムで用意されたサービスをRPC経由で実行する場合もあり、アプリケーションからは、RPCを使っている様には見えない場合でも、内部ではRPCを使って実行されている場合があります。

### 1.3.7　スレッドセーフ

　上記のように複数のアプリケーションから実行されるサービスメソッドを提供する場合は、そのサービスをスレッドセーフな形に作っておかねばなりません。簡単に言えば、同時に複数のプロセスから呼び出されてもそれぞれに正しい結果を返すことができる様にすることです。サービスやコンテントプロバイダで実装する外部から呼び出されるメソッドについてはこの点を注意する必要があります。

　といっても何をどうすればスレッドセーフになるのかわかりにくいと思います。簡単にいえば、内部の状態に依存した処理を行わないということになるのですがそれでもよく分かりません。いくつかの注意点はあるのですが、単純にはクラスの静的メンバ（static と宣言されたメンバ変数やメソッド）を使ったプログラムは問題を招きやすいので、避けるのが望ましいでしょう。通常のメンバ変数はオブジェクト毎に生成されますし、ローカル変数はプロセス毎に生成されますので問題有りませんが、「static」変数や「static」メソッドはすべてのオブジェクトで共有されてしまいますので、同時に複数のオブジェクトが生成されて実行されるような状況では互いに影響を及ぼしてしまう可能性が有ります。「ContetProvider」などでは「static」なメンバは使用しない方がいいでしょう。

### 1.3.8　コンポーネントのライフサイクル

　コンポーネント（アプリケーションの部品）にはそれぞれにライフサイクルと呼ばれる実行状態の遷移があります。コンポーネント毎にそれぞれのライフサイクルを説明します。アプリケーションを作成する際には、このライフサイクルを十分理解しておく必要があります。

#### 1.3.8.1　アクティビティのライフサイクル

　アクティビティは、これまでにも何度も出てきたとおり、画面上にユーザインタフェースを表示して、ユーザの操作を受け付ける為のコンポーネントです。ユーザからどういう風に見えるかという状態の変化がそのままライフサイクルにおける状態の変化となっています。アクティビティは、基本的に以下の三つの状態をとります。

・アクティブ状態

　画面に最前面に表示されてユーザのアクションのターゲットになっている状態です。画面上のボタンなどが操作可能で、ハードウェアボタンを操作した際の挙動もこのアクティビティに通知される様になっています。今まさにユーザが操作している状態です。

・一時停止状態

　ユーザのアクションのターゲットから外れている状態です。画面上にはまだ見えるのですが、その上に別のアクティビティが表示されていることで、ユーザの操作が直接このアクティビティには及ばない状態です。この状態でもアクティビティ自体は完全に動作中の状態です。ただし、極端にメモリが不足した場合などには強制的に終了される可能性があります。

・停止状態

　画面上から完全に見えなくなっている状態です。この状態のアクティビティはメモリ上には存在していていつでも実行できる状態にありますが、ユーザからは見えなくなっているので、他の処理でメモリが必要になると、強制終了されてしまう可能性が高くなります。

　アクティビティの状態が遷移する際には、それを通知するために特定のメソッドが呼び出されます。たとえば、アクティビティが実行を開始した直後には、「onCreate」メソッドが、画面に表示される直前には「onStart」メソッドが呼び出されます。これらのメソッドが呼び出されるタイミングとその後の状態を図にしたのが以下の状態遷移図です。

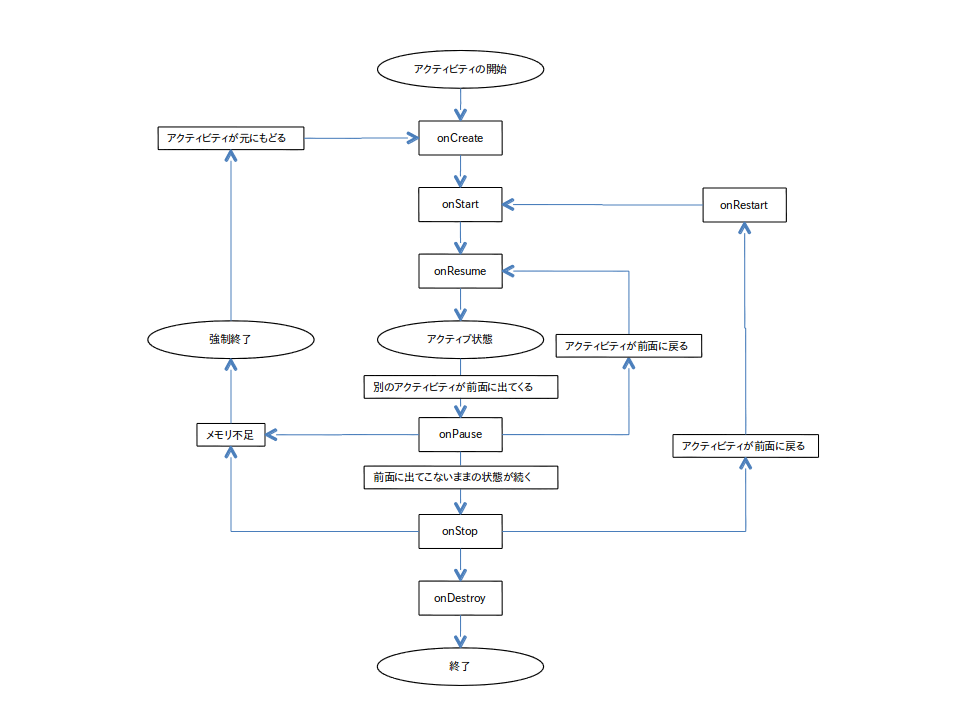


図1.1　アクティビティのライフサイクル

#### 1.3.8.1　状態遷移時に呼ばれるメソッド

・「onCreate」メソッド

　アクティビティが開始されるときに最初に呼び出されるメソッドです。動作に必要な初期化はここで行う必要が有ります。強制終了されたアクティビティもこのメソッドから開始されますが、そのときは状態を保存したBundleオブジェクトが引数として渡されますので、それに応じた初期化を行う必要があります。

・「onStart」メソッド

　ユーザから見える状態になる直前に呼び出されます。通常は、「onCreate」が呼び出された後に呼び出されますが、一端停止したアクティビティが再度表示される場合は後述の「onRestart」の後に呼び出されます。

・「onResume」メソッド

　ユーザとの対話（ユーザからの操作を受け付ける状態）になる直前に呼び出されます。この時すでに画面には表示されていて、ユーザには画面が見える様になっています。この後は基本的にユーザの操作やその他のイベントに応答することでプログラムは進行します。

・「onPause」メソッド

　システムが別のアクティビティを開始しようとする場合に呼び出されます。このメソッドは「onResume」が呼び出されていれば画面から消える前に必ず呼び出されますので、ユーザからの操作やプログラムの状態を永続的に保存する場合などは、このメソッドで保存する必要が有ります。このメソッドが呼び出された後は、一時停止状態になるので、強制終了される可能性があります。逆にこのメソッドが呼び出される前には、アプリケーション自体のエラーによる場合を除いて強制終了されることはありません。

・「onStop」メソッド

　アクティビティが完全にユーザから見えなくなった時に呼び出されます。このメソッドが実行された後は停止状態になりますので、強制終了される可能性がかなり高くなります。

・「onDestroy」メソッド

　アクティビティが破棄される前に呼び出されます。アクティビティが自身で「finish」メソッドを呼んだ場合等に呼び出されます。

・「onRestart」メソッド

　「onStop」メソッドが呼び出された後、停止状態から復帰した時、戻るボタンで別のアクティビティからもどってきた場合などに呼び出されます。「onRestart」の直後には「onStart」メソッドが呼び出されます。この時点ではメモリは開放されていません。メモリが開放されてしまった後再度呼び出される場合は、「onCreate」から呼び出される事になります。

　以上のメソッドの中で、特に注意すべきなのはアプリケーションが終了する場合、特に強制終了される場合には「onStop」,「onDestroy」のメソッドは呼び出されない可能性が有るということです。つまり、終了時に必ず行わねばならないような終了処理は、これらのメソッドで行うことはできません。「onPause」メソッドにも書きましたがユーザの操作の結果を保存するなどの永続的な状態の保存は必ず「onPause」メソッドで行う必要があります。

　資源の開放などの終了処理はほとんどの場合各オブジェクトが開放される際に自動的に行われますので、あまり気にする必要はありませんが、終了する時点で何かを他のプロセス等に通知する必要があるような場合は別途検討が必要になりますが、そのあたりの問題は複雑になりますし、あまり頻繁に発生することではありませんので本書では扱いません。

#### 1.3.8.2　アクティビティから別のアクティビティを実行する場合の処理

　アクティビティ「A」からアクティビティ「B」を実行した場合は、以下の順番で処理されることが決まっています。

１．アクティビティ「A」の「onPause」メソッド

２．アクティビティ「B」の「onCreate」,「onStart」,「onResume」メソッド

３．アクティビティ「A」の「onStop」メソッド

つまり、アクティビティから別のアクティビティを実行すると、最初に「onPause」メソッドが呼び出され、「onStop」メソッドが呼ぶ出される前に、新しいアクティビティの「onResume」メソッドまでが実行されていることになります。

#### 1.3.8.3　サービスのライフサイクル

　コンポーネントの種類の所で説明したとおり、通常のサービスは一度開始したら自身で「stopSelf」メソッドで終了するか、外部から「stopService」で終了させられるまでは勝手には終了しません。サービスの開始は明示的に「startService」メソッドで開始します。

　もう一つの方法として、サービスで定義されているインタフェースをexportしておくことで外部から「bindService」メソッドでサービスに接続することができます。この場合は「unbindService」メソッドで接続を切断します。サービスが開始されていない場合は「bindService」メソッドが呼ばれた時点で開始します。

　ただ、これらの方法は完全に独立したものではなく、たとえば、Serviceを明示的に起動してから「bindService」メソッドで接続することもできます。その場合はすべての接続が「unbindService」メソッドでクローズされない限り、「stopService」メソッドでサービスを終了することはできません。

　ライフサイクル関数としては、「onCreate」「onStart」「onDestroy」の三つが存在します。「onCreate」はサービスが起動した時に呼び出されるもので、Activityと同様このメソッドで初期化などを行います。「onStart」メソッドは「startService」によって起動された際に呼び出されるメソッドで、「bindService」で起動された場合には呼び出されません。「onDestroy」はサービスが終了する際に呼び出されますので、ここで後処理を行うことができます。

　また、サービスをexportしている場合は先に述べたように「bindService」でサービスを開始する場合もあります。この場合は「onCreate」は呼び出されますが「onStart」は呼び出されません。代わりに「onBind」が呼び出され、「unbindService」メソッドが呼び出された時には「onUnbind」が呼び出されます。

　先に述べたようにこれらの状態は別の状態ではありませんので、「onStart」メソッドが呼び出された後で「onBind」メソッドが呼び出されるケースもあることを注意しておく必要があります。

Serviceのライフサイクルを図にすると以下のようになります。

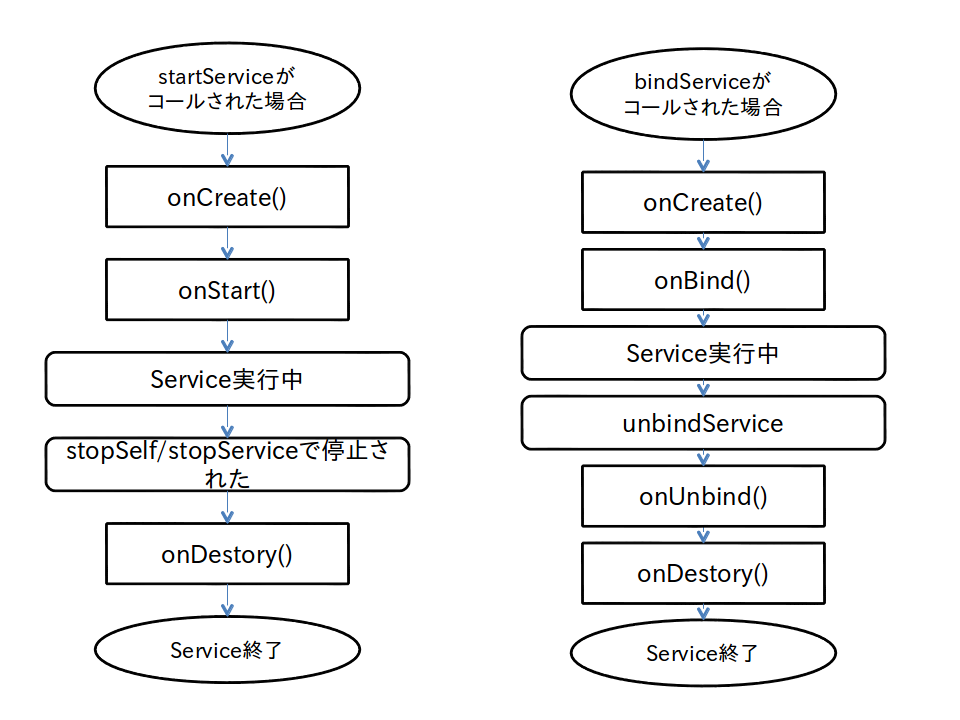


図1.2　サービスのライフサイクル

#### 1.3.8.4　ブロードキャストレシーバのライフサイクル

　ブロードキャストレシーバのライフサイクルはとても単純です。レシーバーで定義された「onReceive」メソッドがコールされ、実行している間のみアクティブになります。つまり、「onReceive」メソッドが終了するとその時点でアクティブではなくなります。

　従ってアプリケーション内に他にアクティブなコンポーネントが無い場合、アプリケーションはメモリが不足するとすぐにメモリを解放されてしまう可能性があります。しかし、ブロードキャストレシーバで時間のかかる処理を実行するのは他のプロセスの動作を妨げてしまうことになるので推奨されません。

　それではブロードキャストレシーバでは時間のかかる処理が行えないことになってしまいます。そういった処理が必要な場合はブロードキャストレシーバでサービスを起動し、別スレッドで実行する様にします。そうすることでサービスがアクティブな間は勝手に開放されてしまうこともなく、ブロードキャストレシーバはすぐに終了することができます。もちろんサービスはずっと動作する必要がないのであれば処理が終了したらすぐに「stopSelf」メソッドで自身を終了させるべきです。

#### 1.3.8.4　ライフサイクルと強制終了

　コンポーネントのライフサイクルについて説明してきましたが、最後にプロセスの強制終了について簡単にまとめます。

　アプリケーションを作成する際にはユーザのデータを保存するなど終了する前に必ず処理しておかねばならないことなどを管理する為にもプロセスがいつどういう状態の時に終了するのかについてある程度理解しておくことが必要です。

まず、プロセスは以下の5つの階層に分類されます。最も重要度の高いものから

１．フォアグラウンド・プロセス

　　・ユーザが使用しているアクティビティ

　　・ユーザが使用しているアクティビティにバインドされているサービス

　　・ライフサイクルコールバックを実行中のサービス

　　・onReceiveを実行中のブロードキャストレシーバ

　　見ても分かるとおり、確かに勝手に終了されては困るものばかりです。これらのプロセスをフォアグラウンド・プロセスと呼び最も優先度の高いプロセスになります。

２．可視プロセス

　　・画面に見えているアクティビティ

　　・画面に見えて言えるアクティビティにバインドされているサービス

　　「画面に見えている」とはダイアログなどの陰になっているアクティビティなどで完全には表示が見えなくなっているわけではないけれど、ユーザからの操作は受け取らないようなアクティビティのことです。（「onPause」メソッドが呼び出されています）

　　このアクティビティが突然消えるとユーザは何が起こったのかと思うでしょうし、ダイアログなどの陰になっているアクティビティはダイアログが消えたとき戻るべきアクティビティなので、消してしまうと再度起動する必要があり、ユーザインタフェース上も不都合が多いので、可視プロセスとして優先度が上がっています。

３．サービス・プロセス

　　直接ユーザの操作する対象とはつながっていないけれど、音楽再生などユーザへのサービスを提供している可能性のあるプロセスになります。

　　ユーザのインタフェースを司っている訳ではないので優先順位は下がりますが、それでも急に止めて欲しくないプロセスとなります。

４．バックグラウンド・プロセス

　　完全に画面から見えなくなっているアクティビティを保持しているプロセスです。アクティビティ実行中にHomeボタンを押してホームに戻ってしまった場合や、通知から別のアクティビティを実行した結果画面上に表示されなくなってしまったアクティビティなどです。（「onStop」メソッドが呼び出されています）

　　画面から完全に消えていますので停止してしまってもユーザにはほとんど気付かれないでしょう。再度表示しなければならなくなった場合は再起動となります。

５．空プロセス

　完全にアクティブなコンポーネントの存在しないプロセスです。メモリが十分に余っているのであればこのプロセスを残しておくことで次回同じアプリケーションを実行しようと思ったとき素早く起動することができますので残しておいてもかまわないでしょうが、通常は他のプロセスがメモリを要求することですぐに削除されてしまうでしょう。

　以上のようにサービスなどではバインドのされ方などで優先順位がころころ変わりますので、終了処理などが必要な場合は十分考えておく必要があります。

　アクティビティについては単純に「onPause」で状態を保存することにしておけばそれほど致命的なことにはならないでしょう。

　以上でAndroidのアプリの構造について説明しましたが、まだ今ひとつどんなものかつかみにくいと思います。このあたりの構造は言葉で説明してもなかなか理解するのは難しいので、詳しいことは実際のプログラムを作りながら理解していくのが一番の近道でしょう。

　ここでは出てきた言葉の意味をおおざっぱにつかんでもらえれば十分だと思います。これらの内容はこのあとの実際のプログラムの説明の中で必要に応じて再度説明しますので、ここでは気にせず次のステップに進んでください。

## 1.4　Android開発環境の構築

### 1.4.1　Android SDKを使った開発環境の構築

　Androidアプリケーションについての簡単な説明を行いましたが、ここからはAndroid SDKを使ったアプリケーション作成について説明します。

　Androidのアプリケーションを作成するには、開発用のPCを準備し、そのPCにAndroid SDKをインストールする必要があります。Android SDKのインストールについての情報は、公式に以下のURLで提供されています。

Get the Android SDK

http://developer.android.com/sdk/index.html

このページではアクセスしたPCのOSの種類に従ったADT Bundle SDKのダウンロードができるようになっています。

　基本的にはこのSDKをダウンロードして展開するだけでAndroidアプリケーションの作成環境は構築できるのですが、SYSTEM REQUIREMENTSに書かれている準備を行う必要があるのでここでは先にREQUIREMENTSに書かれている準備作業から説明します。

　まずハードウェア、OSとして必要な環境は以下のいずれかです。

・Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)

・Mac OS X 10.5.8 or later (x86 only)

・Linux (tested on Ubuntu Linux, Lucid Lynx)

GNU C Library (glibc) 2.7 or later is required.

On Ubuntu Linux, version 8.04 or later is required.

64-bit distributions must be capable of running 32-bit applications.

最も簡単に入手できるのはWindows環境もしくはMac環境だと思います。

Androidのシステムまで含めたビルド環境を用意したい場合はUbuntuまたはMacの環境が必要となりますが、このコースではAndroidアプリの作成のみを行いますのでWindows環境をもとに説明を行います。

次にソフトウェアとしては以下のものが必要です。

・Eclipse 3.6.2 (Helios) or greater

Note: Eclipse 3.5 (Galileo) is no longer supported with the latest version of ADT.

・Eclipse JDT plugin (included in most Eclipse IDE packages)

・JDK 6 (JRE alone is not sufficient)

・Android Development Tools plugin (recommended)

ただしこの中でJDK以外はすべて上記URLのページからダウンロードできるADT Bundle に含まれていますので、ここではJDKのインストールのみ行います。

### 1.4.2　JDKのインストール

JDKはJava Development Kitのことで、ここではOracleが配布している最新のJDKをインストールします。

　上記のRequirementsではJDK６が必要と書かれていますが、現時点でのJDKの最新はJDK7となっており、現在のAndroidSDKはJDK7での動作も可能ですのでここでは素直に最新のJDK7をインストールします。

　OpenJDKなどLinuxでは互換JDKが存在するのですがこれらは動作が確認されていませんので、Linux上で環境を構築するさいにはご注意ください。

　また、現在のMacOSXではJavaがインストールされていませんので、MacOSXにおいてもJavaのインストールは必要となります。

　OracleのJDKは以下のURLで配布されています。

http://www.oracle.com/technetwork/java/javase/downloads/index.html

このページにアクセスすると以下のような画面が表示されます。（2014/2/8時点）

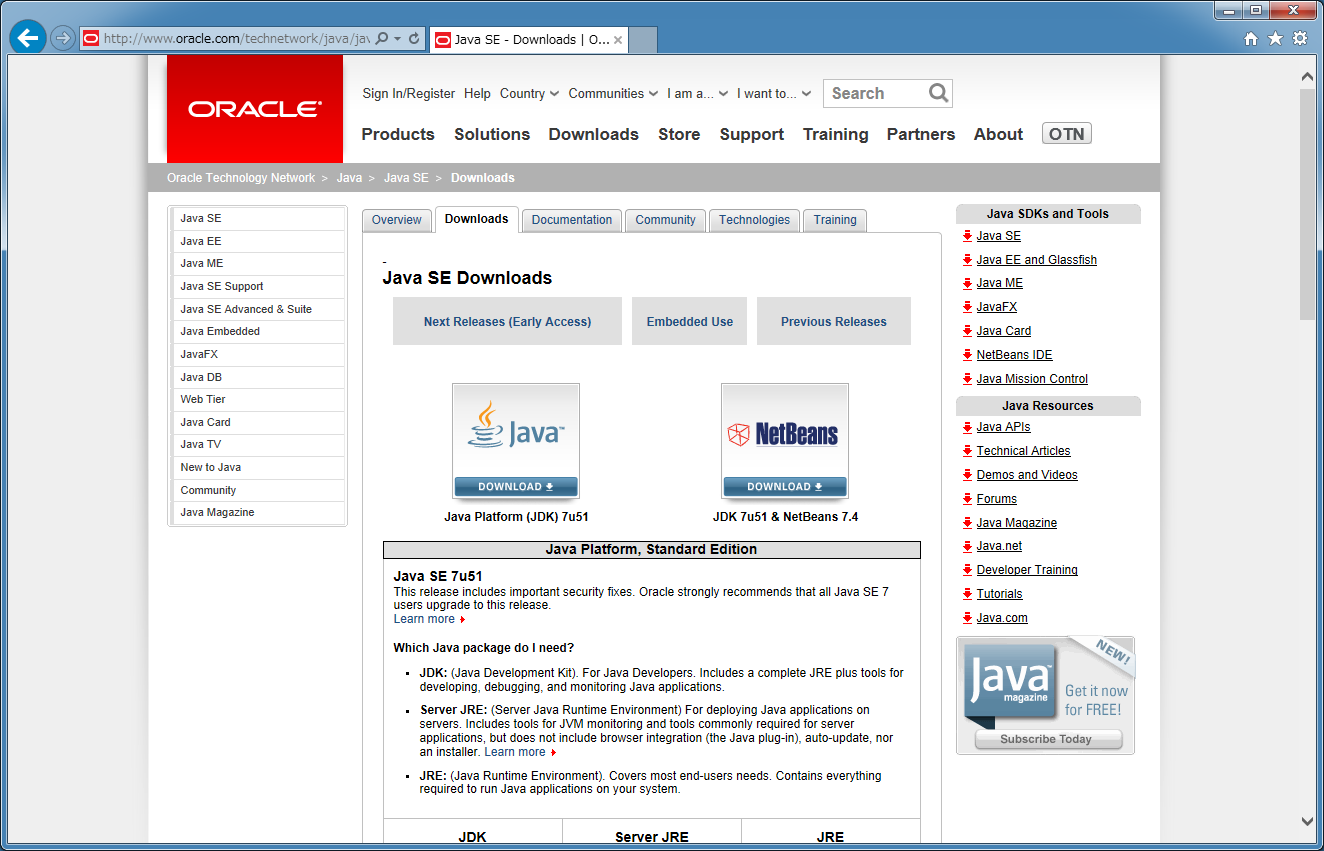


図1.3　OracleのJDKダウンロードページ

この時点での最新版はJava SE 7u51となっています。

少しスクロールすると以下のようにJDKとJREを選択することができますので、JDKの下のDOWNLOADボタンをクリックします。（図中赤枠）



図1.4　OracleのJDKダウンロードページ(2)

続けて以下の画面になるので、「 Accept License Agreement  」のラジオボタン（図中赤枠）をクリックし、システムに合ったファイルをクリックしてダウンロードします。

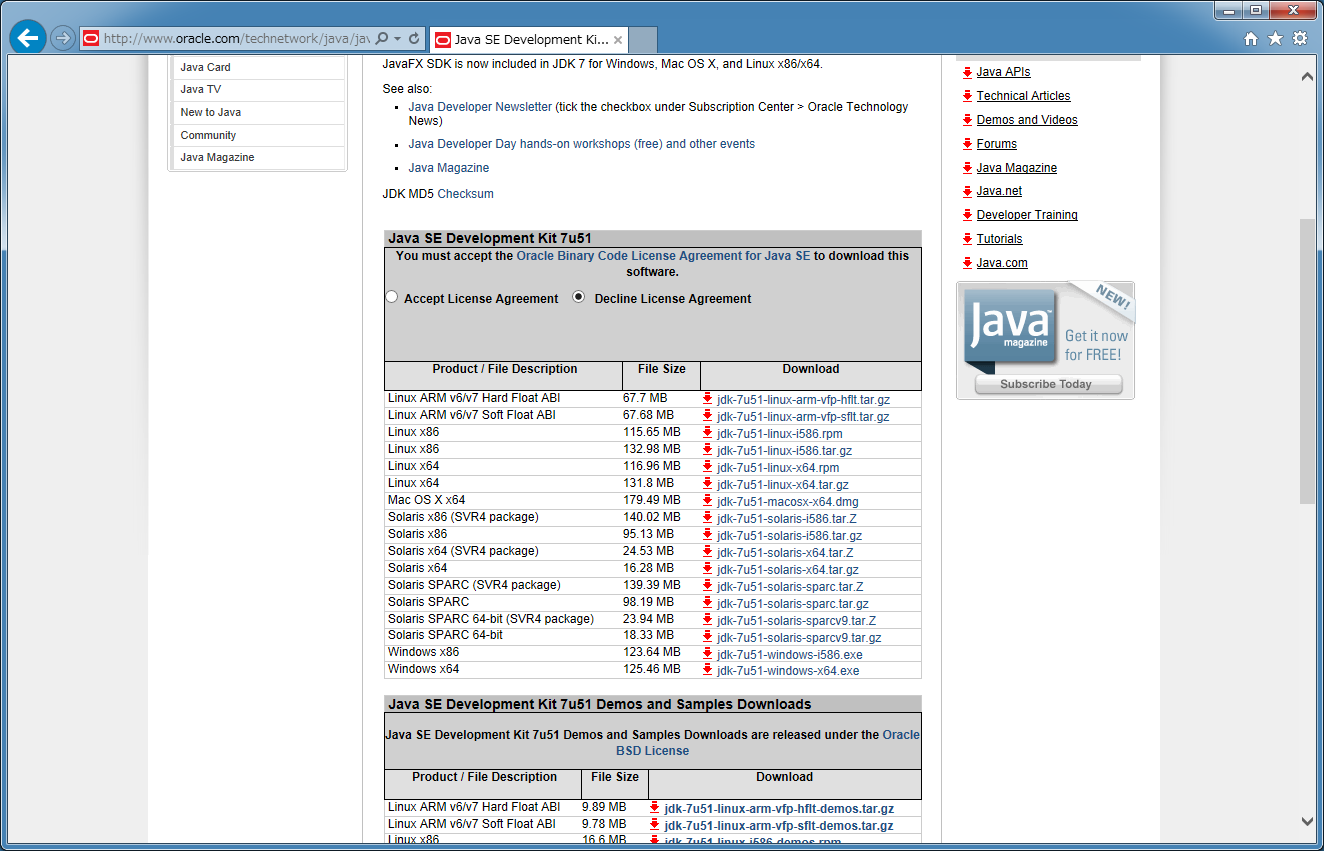


図1.5　OracleのJDKダウンロードページ(3)

ここではWindows　32bitを使用するため、[Windows x86]と書かれたファイル、jdk-7u51-windows-i586.exe　（図中緑枠）をクリックして実行を選択します。

JDKのインストール画面が表示されますので標準のままインストールを実行してください。

実行時に「次のプログラムにこのコンピュータへの変更を許可しますか？」というダイアログが表示されますので、「はい」を選択してください。

その後は「次へ」を何度かクリックしインストールを完了してください。

### 1.4.3　AndroidSDKのダウンロードと展開

再び以下のURLに戻ります。

Get the Android SDK

http://developer.android.com/sdk/index.html

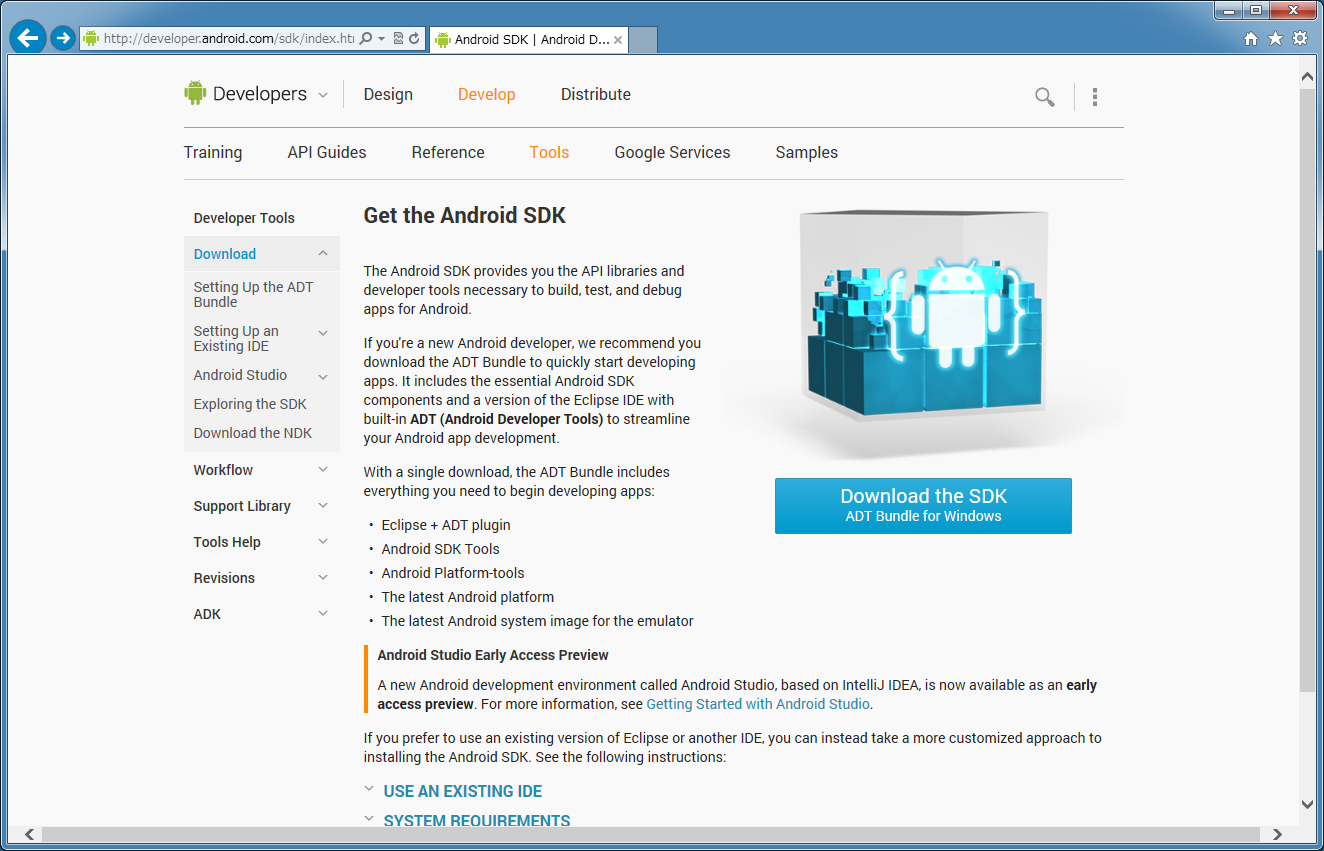


図1.6　AndroidSDKダウンロードページ

このページの　Download the SDK ADT Bundle for Windowsのボタンをクリックします。

続けて表示される下記のページのTerms and Conditionsに一通り目を通した上で　[I have read and agree with the above terms and conditions]のチェックボックスをクリックして、32-bitか64-bitのラジオボタンを選択して[Download the SDK ADT Bundle for Windows]と書かれているボタンをクリックしてダウンロードします。

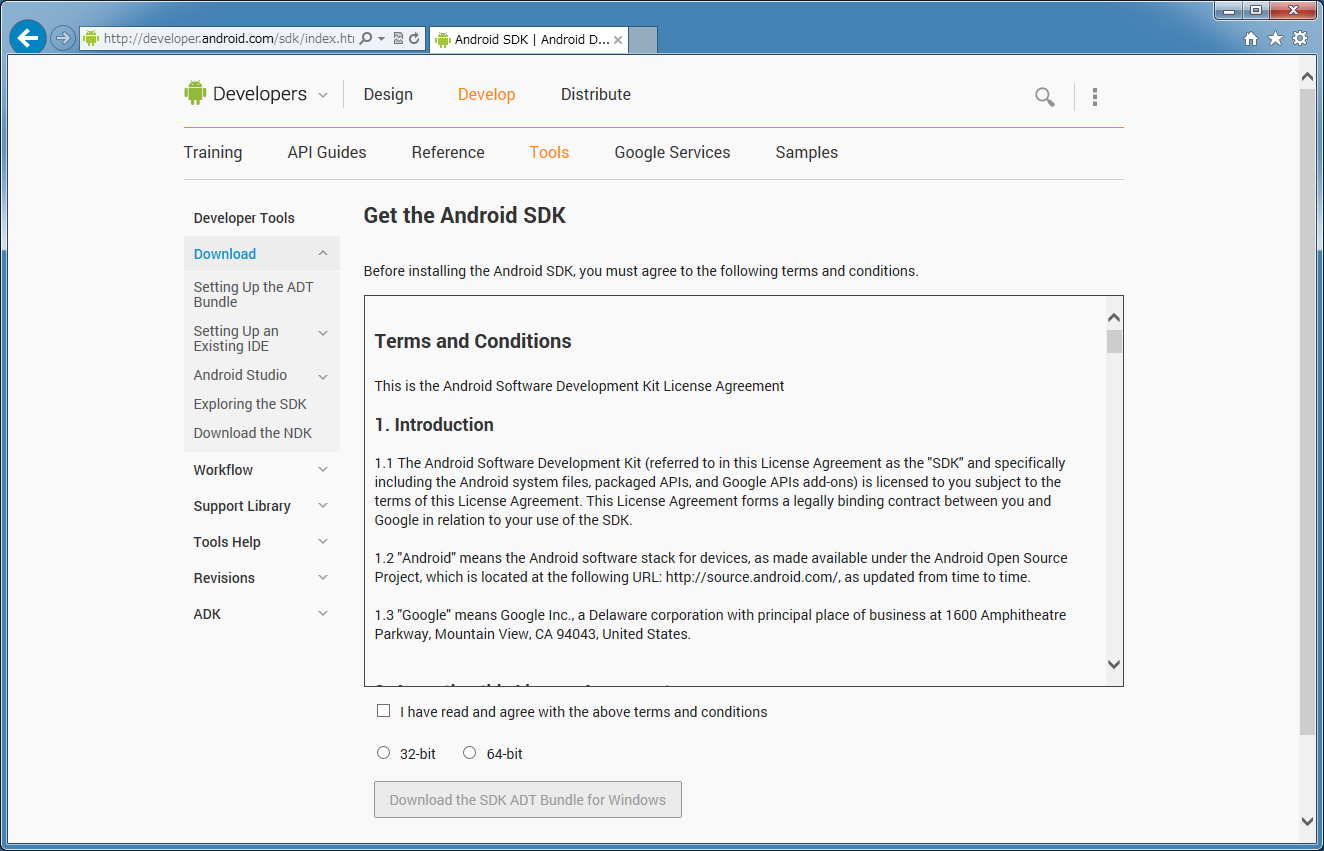


図1.7　AndroidSDKダウンロードページ(2)

ダウンロードが完了したら、アーカイブファイルをひらき、適当なフォルダーに展開します。

展開先は特に決まりはないので自分で使いやすいところに展開してください。

実習用のPCでは　ユーザディレクトリの中に展開しています。

ダウンロードしたファイル名はadt-bundle-windows-x86-20131030.zipで、ファイルをユーザディレクトリに展開するとadt-bundle-windows-x86-20131030というディレクトリが作成されて、中には以下のようにeclipse,sdkというディレクトリとSDK Managerというプログラムが含まれています。

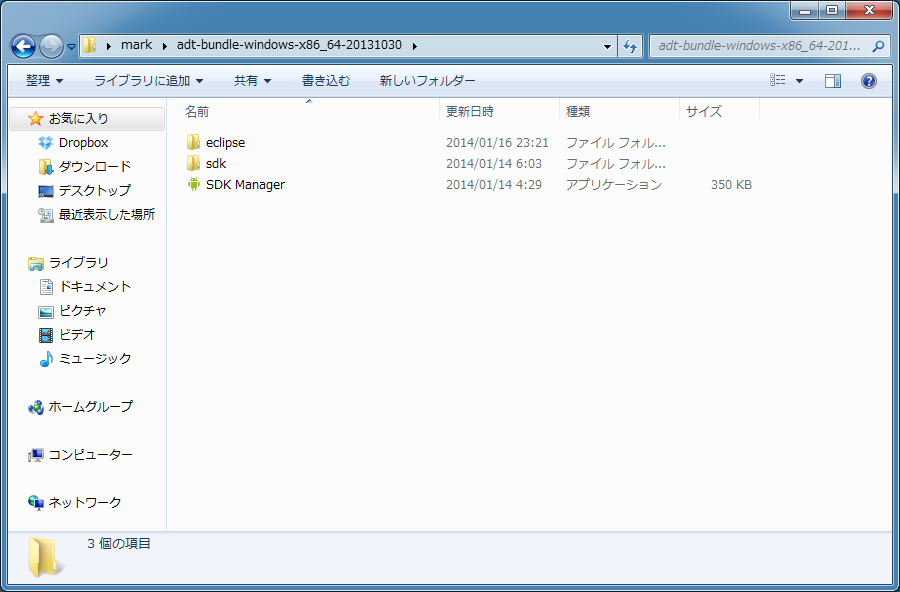


図1.8　AndroidSDKを展開したフォルダー

※このイメージはWindows7 64-bitなのでディレクトリ名が異なりますが中のファイルは同じです

### 1.4.4　SDKのセットアップ

ここまででSDKのインストールは完了していますが、実際にAndroidアプリケーションを開発するには以下のSDKのセットアップが必要です。

・SDKパッケージの導入

・AVDの作成

SDKパッケージとは、Androidアプリケーションを開発するために使用するライブラリやツール、フレームワーク、デバイスドライバなどです。

インストール直後のSDKでは最新のAndroidに対応したパッケージのみがインストールされていますが、先に述べたようにここではAndroid 2.2に対応したアプリを作成したいので、最小限Android2.2のライブラリをインストールしておく必要があります。

またNexus7の実機を接続して動作確認を行いますので、WindowsではそのためのUSBデバイスドライバのインストールが必要となります。

　AVD（Android Virtual Device）は作成したアプリをPC上でデバッグするためのAndroidのエミュレータのことです。

　このエミュレータは画面サイズやCPU、物理キーの有無などを設定して作成することができますので、自分が必要とするタイプのデバイスを作成しておく必要があります。

### 1.4.5　Eclipseの起動

　SDKをインストールしたディレクトリのeclipseディレクトリに含まれるeclipseといファイルをダブルクリックして実行します。

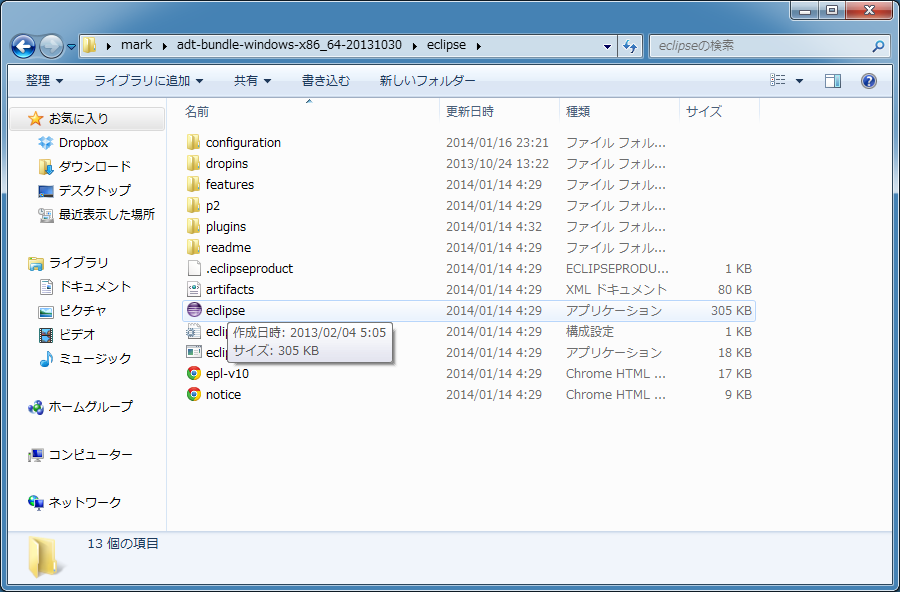


図1.8　eclipseフォルダー

起動するとセキュリティの警告が表示されますが、正しく上記のURLからダウンロードしたものであれば、実行して問題はありません。

そのとき「このファイル開く前に常に警告する」というチェックボックスがチェックされていた場合はチェックを外しておくと次回から表示されなくなります。

実行すると　workspace Launcher というダイアログが表示されworkspaceを作成する場所を確認されますが、そのままOKをクリックしてかまいません。

　その際　Use this as the default and do not ask againというチェックボックスをチェックしておけば次回以降は表示されません。

　また初回起動時にはGoogleへのレポートの送信の確認ダイアログが表示されますがこれは各自の判断で送信するかどうかを決めてください。

　送信する、しないのどちらを選択してもアプリケーション作成には関係ありません。

　加えて初回起動時は以下の画面のようなWelcome!画面が表示されますが、特に必要無いのでタブのX（図中赤枠）をクリックして閉じてかまいません。

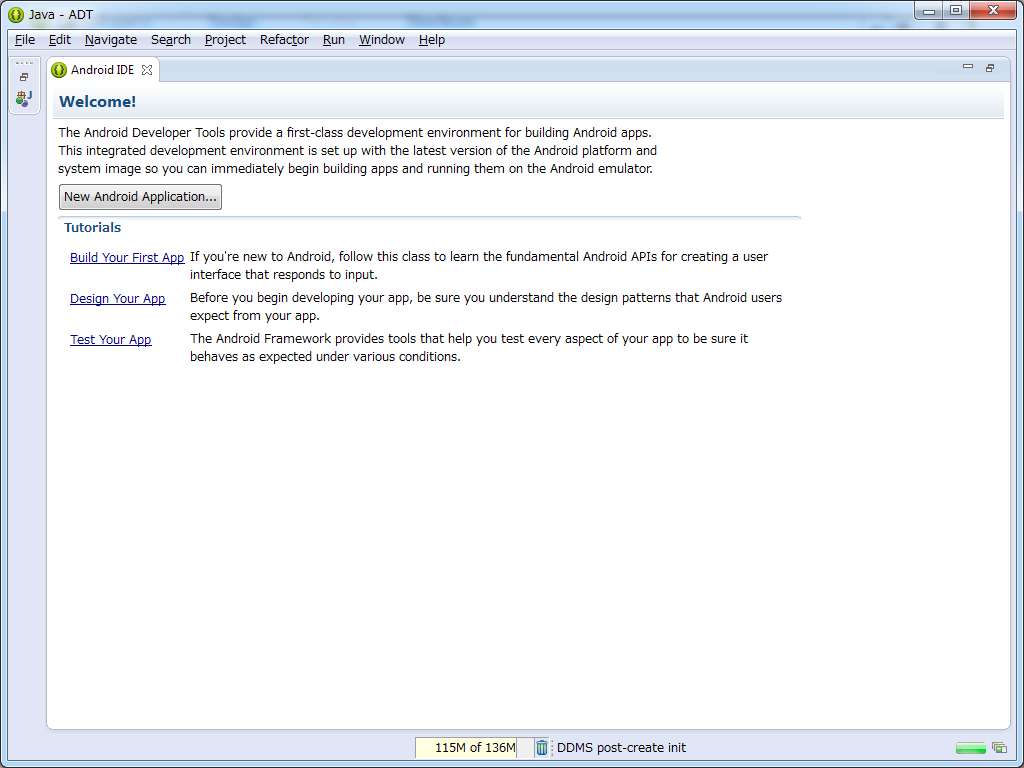


図1.9 Welcome 画面

### 1.4.6　SDK　パッケージのインストール

メニューのWindow を開き「Android　SDK　Manager」を選択すると下記のようなアプリケーションが起動します。

　これはSDKのパッケージを管理するためのアプリケーションで、このアプリケーションでパッケージのインストールや更新を行います。

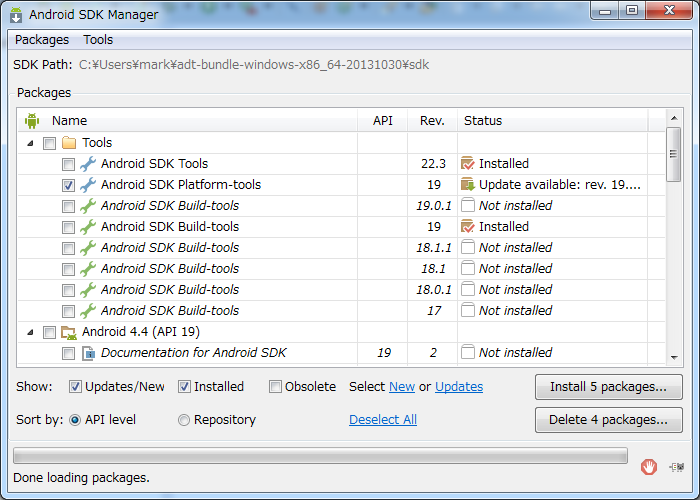


図1.10　SDK Manager

初期状態では最新のVersion（この場合は4.4）の　SDK Platformのみがインストールされています。

　今回のコースでは先に行ったとおりVersion2.2以降を対象としたいので、Android 2.2のSDK　Platformのインストールが必要です。

下図の様にリストをスクロールしAndroid 2.2(API 8)のチェックボックスをチェックしてください。

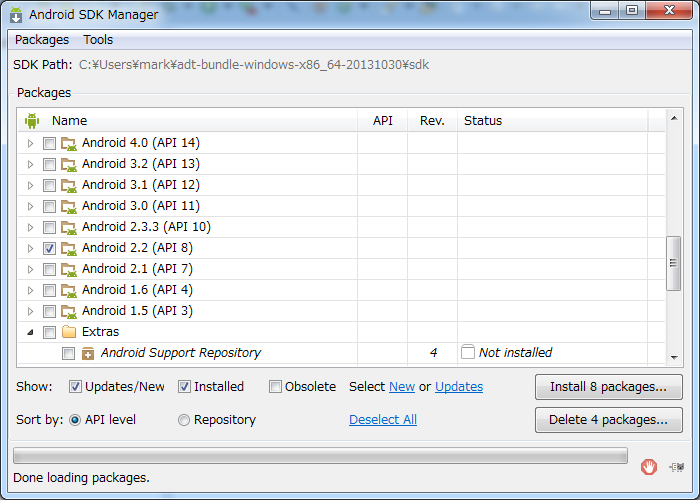


図1.11　SDK Manager(2)

　この状態で　[Install X packages…]というボタンをクリックするとライセンスの確認画面が表示されて必要なソフトが自動的にインストールされます。

　ライセンスの確認画面では[Accept License]ボタンをクリックすることでInstallが可能になります。

　PCに十分な容量と時間がある場合はすべてのパッケージをインストールしておいてもかまいません。

　SDKパッケージのインストールが完了すればSDK Manager のウィンドウは閉じてかまいません。

### 1.4.7　AVDの作成

SDK Managerの起動と同様　メニューから[Window]->[Android Virtual Device Manager]を選択してAVD Managerを起動します。

　AVD Managerは先に述べたとおり、AndroidのアプリをPC上でデバッグするためのエミュレータを管理するためのソフトです。

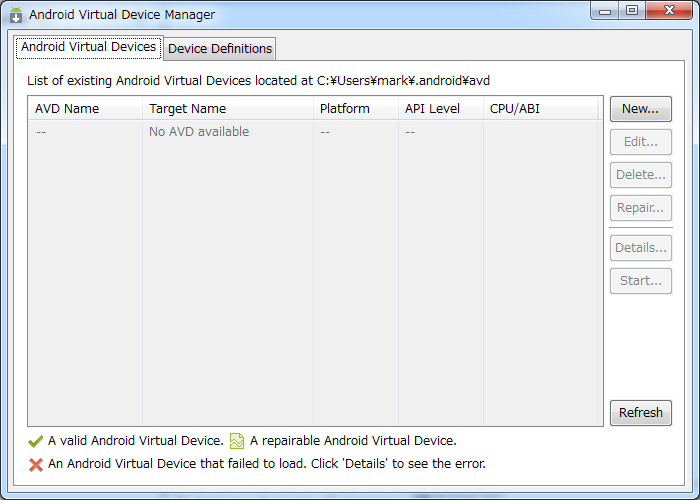


図1.11　AVD Manager

AVD Managerを使用して今回ターゲットするNexus7と同等のAVDを作成します。

　[New…]ボタンをクリックし、以下の設定を行います。

AVD Name: Nexus7AVD

Device:Nexus 7 (7.27’’,800 x 1280:tvdpi)

Target:Android 4.4 – API Level 19

CPU/ABI: ARM(armeabi-v7a)

Keyboard: □Hardware keyboard present（チェックを外す）

Skin:Display a skin with hardware controls(チェックをする)

Front Camera:None

Back Camera:None

Memory Options: RAM:768 VM Heap:32

Internal Storage: 200 [MiB]

SD Card: Size: 1024[MiB]

Emulation Options: Snapshot（チェックする）

　　　　　　　　　Use Host GPU（チェックしない）

　内容を確認しOKをクリックします。

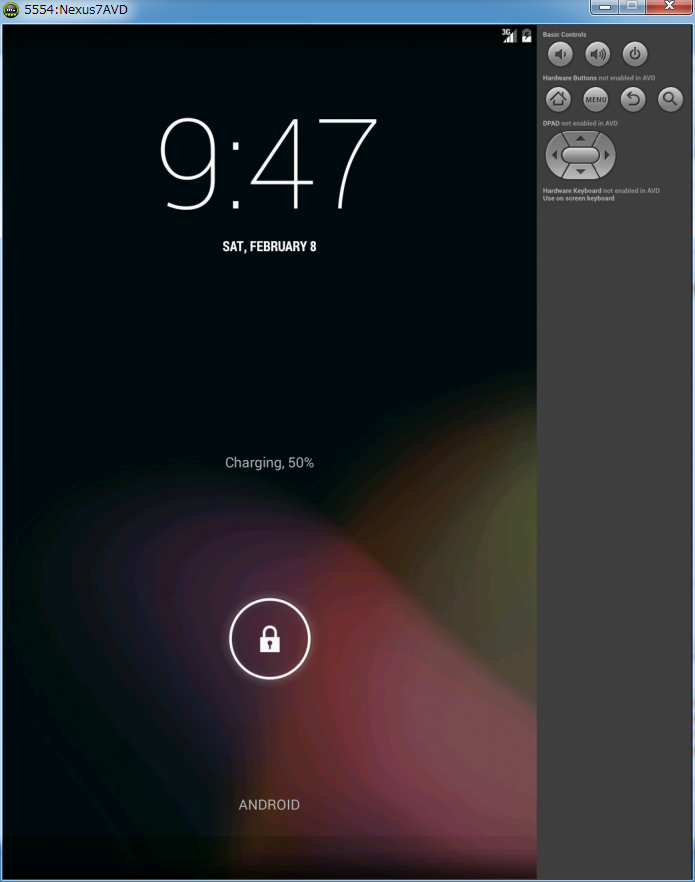
### 1.4.7　AVDの実行

　AVDの起動には時間がかかるので一度実行を行います。

　AVDManagerのリストから先ほど作成したNexus7AVDを選択し[Start…]ボタンをクリックすると、LaunchOptionsのダイアログが開くのでそのまま[Launch]ボタンをクリックして起動します。

　ウィンドウが開きエミュレータでAndroidの起動が始まります。これにはかなりの時間がかかりますのででAndroidが起動するまでしばらくまちます。

　起動が完了すると以下のような画面が表示されますので、画面下部の鍵アイコンを右にドラッグして初期画面を表示します。



この状態でウィンドウの閉じるボタンをクリックして一旦終了します。

　これによってSnapshotのバイナリイメージを保存することができます。

　次回起動時にはこのSnapshotイメージを使って起動することができますので起動時間が若干短縮されます。

　以上で開発環境の準備はほぼ完了です。

続けて簡単なサンプルを作成します。

## 1.5　サンプルの作成手順の確認

新しい環境やコンピュータ言語を学習する際、最初の作成するプログラムの一つにHello worldと呼ばれるものがあります。

画面にHello worldという文字列を表示することだけが目的のプログラムですが、ソフト開発における最小限の環境の確認として使われています。

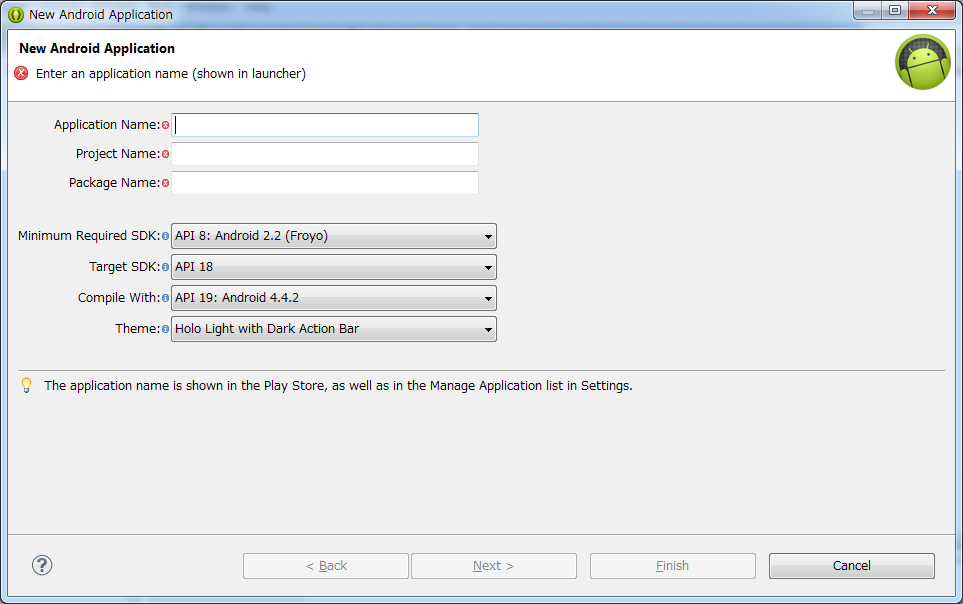
AndroidのアプリケーションでHello worldを作成するのは至極簡単です。プログラムを書く必要は全くありません。

では、Helloworldの作成を始めます。

### 1.5.1　Helloworldの作成

セクション1.4.5の説明に従ってEclipseを起動します。

新しいAndroidアプリケーションを作成するためにFile -> New -> Android Application Project を選択します。以下のようなダイアログが開きます。



ここではアプリケーション名などを設定します。

今回はHelloworldのテストなので、

Application Name:　Helloworld

Project Name:Helloworld

Package Name:com.example.helloworld

とします。

Application NameにHelloworldと入力するだけ、自動的にほかの所は設定されます。

このときPackage Nameに黄色い三角形のアラートが表示されていると思います。

com.exampleというのはただのプレースホルダーだから書き換えろと言っていますが、ここではただのサンプルで、エミュレータ上で実行するだけなのでこのまま進めます。

　これ以外の項目について簡単に説明します。

・Minimum Required SDK:

　　このアプリケーションを実行することができる最も古いSDKバージョンを指定します。

　　これを指定することで、どこまでの互換性を保つかを決めます。

　　今回は先にも書いたとおりAPIレベル８　Android 2.2 （Froyo）以上をターゲットにします。

・TargetSDK:

　　使用するSDKの最も上位のレベルを指定します。

　　このSDKレベルがメインターゲットとなります。Androidのアプリケーションは基本的に互換性があり、新しいバージョンの端末で古いバージョンのアプリケーションを実行した場合、画面表示などのイメージが変わる場合があります。TargetSDKのバージョンを古いものにすることで、大画面対応などが自動的に行われるのを防ぐこともできます。

　　通常はアプリケーション作成時点で最も普及しているバージョンを選んでおけばいいでしょう。最新を選んでもかまいません。

　　ここではデフォルトのAPI レベル１８を選択しておきます。

・Compile With:

　　コンパイラをどのSDKのバージョンにするかを指定します。

　　どうしても新しいバージョンでは問題が発生する様な場合を除いて、基本的に最新版を使用するのを推奨します。

・Theme:

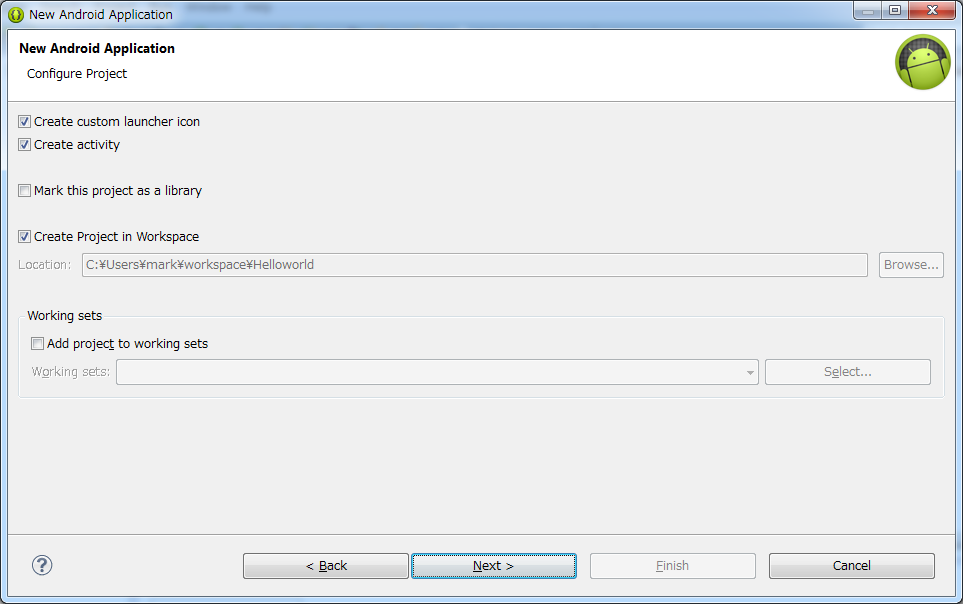
　　画面レイアウトのテーマを選択します。

　　画面デザインをすべて自分で行う場合は特に気にすることはありませんが、デフォルトのボタンなどを使用して作成するような場合は、このテーマを変更することで、イメージを変えることができます。

　　とくにどれを使えばいいというものではありませんので、自由に選ぶことができます。

　　ここでは特に変更しないでデフォルトのままで使用します。

それぞれの設定を確認したら　[Next>]ボタンをクリックします。



ここではCreate custome launcher Iconのチェックを外します。

このチェックをつけておくと、アイコン次の画面でアイコン用のデータを指定する必要があります。

　CreateActivityはチェックしたままにします。

　このチェックを外すと自分でアクティビティの作成をしなければなりません。

　アクティビティを必要としないようなアプリケーションの場合はこのチェックを外して次に進みます。

　それ以外の項目について

　Make this project as a library

　　ほかのアプリケーションで使用するためのライブラリを作成する際にチェックします。

　Create Project in Workspace

　　プロジェクトをワークスペースに作成するかどうかを選びます。

　　このチェックを外した場合はプロジェクトを作成する場所をLocationに設定します。

　Working sets

　　いくつかのプロジェクトをまとめてWorking setを作成する際に、どのワーキングセットに登録するかを指定します。

　　複数の関連するアプリケーションをまとめてビルドするような場合にワーキングセットを作成すると便利です。

　　ここでは単体のアプリケーションなのでこの項目は指定しません。

以上の設定が完了したら[Next>]をクリックします。



この画面ではActivityのタイプを選択します。

・BlankActivity

画面にHelloworldとだけ表示されるアクティビティです。

通常のアプリケーションではこれで十分でしょう。

・Fullscreen Activity

画面上部と下部のバーを表示しない様に設定されたActivityです。

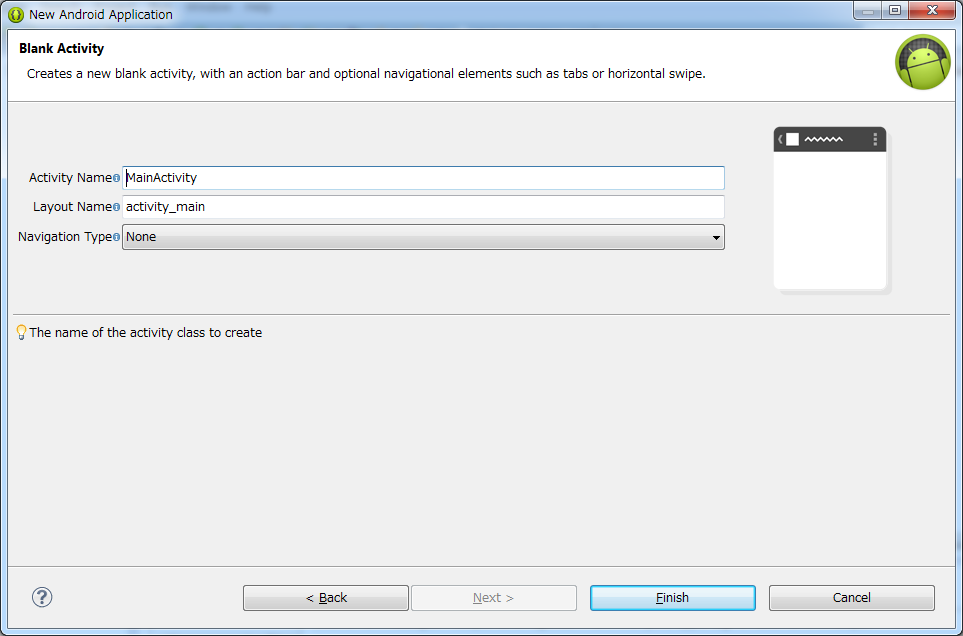
ゲームなど画面全体を使用したい場合に選択します。

・Master/Detail Flow

　　画面の左側に項目リスト、その右に詳細を表示するようなアプリケーションを用のアクティビティです。

　　特にタブレットなどの画面の大きなものや、横置きが標準となるようなアプリケーションで使用します。

ここではBlankActivityを選択して　[Next>]をクリックします。



最後にActivityの名前を設定します。

ここでは自動的にMainActivityとなっていますので、そのままでいいでしょう。

　メインのアクティビティが複数あるようなアプリケーションを作成する場合や、アクティビティのクラス名を変更したい場合はここで指定します。

　また、Layout Nameも同様です。

・Navigation Type

複数のトップ画面を切り替える方法を選択できます。

　トップ画面で複数のタブ表示をしたい場合、複数の画面を左右にスクロールして使いたい場合、画面上部のドロップダウンメニューで画面を切り替えたい場合などの方法を選ぶことができます。

　画面が一つしか無いアプリでは特に必要は無いので、Noneを選択します。

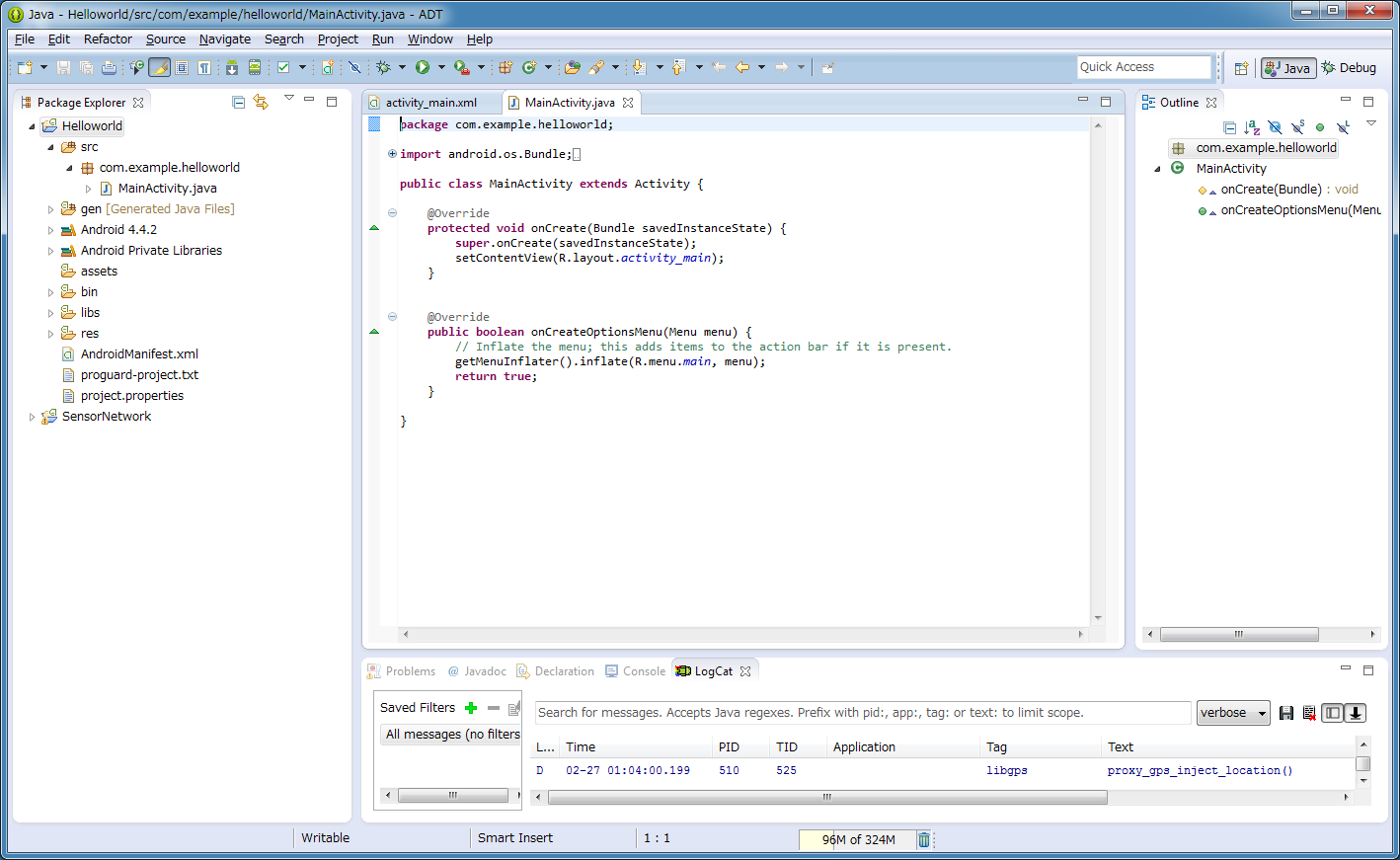
以上で設定は完了です。　[Finish]をクリックすると自動的にHelloworldのプロジェクトが作成されます。

　画面左にPackageExplorerの所にHelloworldがあることが確認できます。

　これでHelloworldアプリケーションは完成です。

　エミュレータ上で実行してみましょう。

### 1.5.2　エミュレータでの実行



デフォルトでは画面の左側のペインがPackage Explorer、中央のペインがソースファイルなどの編集ペイン、右のペインがアウトラインなど、現在編集中のコンテンツの情報、右下がログやコンソール、エラー表示などとなります。

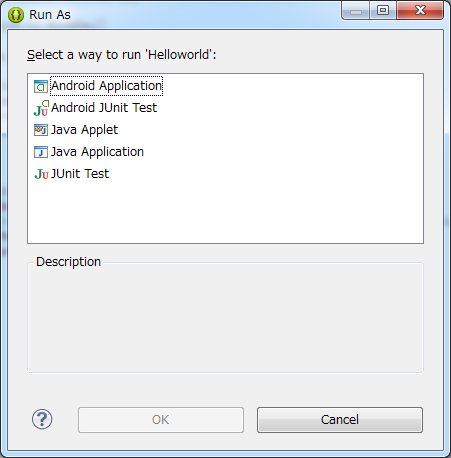
　このレイアウトは固定ではありませんので、自分で使いやすい様に調整していくことができます。

実行するにはRunを選ぶだけで十分です。

他面上のプレイボタンの様なアイコンをクリックするか、RunメニューからRunを選択します。

RunメニューからRunを選ぶと以下のダイアログが表示されます。

これはプロジェクトを作成して1回目の起動のときのみ表示されますので、



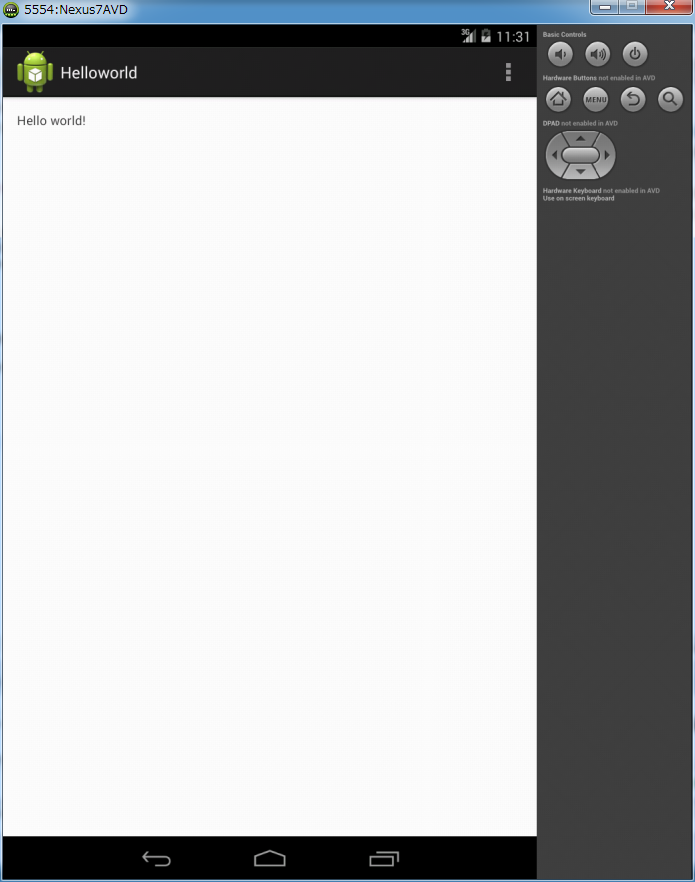
ここではAndroid Applicationを選択し[OK]をクリックします。

自動的にエミュレータが起動し、Helloworldアプリが実行されます。

次のような画面が表示されていると思います。

もし、正しく実行できていない場合は、再度Runメニューを実行してみてください。

初回起動時など、エミュレータの起動に時間がかかりすぎて、アプリケーションの起動に失敗したと判断される場合があります。



上記のウィンドウの閉じるボタンをクリックして閉じてください。

終了にも少々時間がかかかりますので、しばらく待つ必要があります。

続けて実機の上で実行してみましょう。

### 1.5.3 実機での実行

まずNexus７をUSBケーブルでPCに接続します。

初めて接続した時はデバイスドライバのインストール画面が表示されます。

　インストールの途中でドライバの確認が表示されたらGoogleが提供するUSBドライバを指定する必要があります。

　デバイスドライバはAndroid SDKのインストールを行ったディレクトリの中の

　\SDK\extras\google\usb\_driver

　に格納されていますので、このディレクトリを指定します。

　ファイルを確認されずに、インストールに失敗したと表示されてしまう場合は、コントロールパネルを開き、デバイスマネージャーの中のほかのデバイスにNesus7という項目があるはずですので、それをダブルクリックしてデバイスドライバの更新を選択し、先ほどのUSBドライバを指定してインストールします。

　デバイスドライバのインストールが完了したらNexus７で先ほどのアプリケーションを実行してみましょう。

　RunメニューのRunを選ぶだけで、実機上で先ほど作成したHelloworldが実行できます。

以上でサンプルアプリケーションの動作確認は完了です。

# AndroidセンサAPIの構造

　Android端末には各種センサが搭載されており、AndroidのAPI でも数多くのセンサがサポートされています。

　AndroidのセンサAPIについての説明の前にAndroidAPIの概略について説明します。

## Android APIの概略

　AndroidAPIは大きく分けて

　　・UserInterface

　　・Animation and Graphics

　　・Computation

　　・Media and Camera

　　・Location and Sensors

　　・Connectivity

　　・Text and Input

　　・Data Storage

　　・Administration

　　・Web Apps

　などがあります。

　それぞれのAPIについて簡単に説明します。

・User Interface

　Layoutやメニュー、イベントの処理、ダイアログの処理など、画面表示や操作にかかわるフレームワークです。

・Animation and Graphics

　表示の中でも特に動画やOpenGL、グラフィックスアクセラレータなどに関するフレームワークです。

・Computation

　RenderScriptを中心としたハイパフォーマンスな演算処理サービスのフレームワークです。

・Media and Camera

　内蔵カメラによる写真撮影や動画の撮影、オーディオやビデオの再生などを行うフレームワークです。

・Location and Sensors

　GPSなどによる位置情報や重力や磁力、温度や明るさのセンサなど各種センサを扱うフレームワークです。

・Connectivity

　Bluetooth,Wifi,USB、NFCなど外部のデバイスとの接続に関するフレームワークです。

・Text and Input

　文字入力、表示など、コピーペースト、IME、スペルチェッカーなどのフレームワークです。

・Administration

　システム管理のためのフレームワークです。

・WebApps

　WebViewを中心とした、Webアプリケーションのためのフレームワークです。

## センサAPIの種類

　ここではセンサAPIの種類とその内容について説明します。

まずセンサAPIは先のAPI概略にあったとおり大きく分けてLocationとSensorにわかれます。

　どちらも内蔵したセンシングデバイスからの情報を収集するという点が共通するので、フレームワークとしては一つにカテゴライズされています。

以下ではそれぞれのセンサについて、具体的なクラスおよびその使い方について説明します。

### GPS

Locationサービスとしてまとめられています。

この機能はそのままGoogleMapsと連携して簡単んに使用できるようになっています。

基本的な使い方の流れは以下のようになります。

・LocationManagerの取得

　LocationManagerのインスタンスをSystemServiceから取得します。

android.content.Contextクラスのメソッド

getSystemService(Context.LOCATION\_SERVICE);

を呼び出して、LocationManagerを取得します。

ActivityがContextクラスを継承していますので、アプリケーションのメインアクティビティのメソッドを実行することで取得することができます。

・ロケーションリスナーの登録

LocationListenerインタフェースをインプリメントしたクラスをLocationManagerのインスタンスに登録し、位置情報などの更新情報を受け取るようにします。

LocationListenerでは最低限以下のメソッドを実装しておく必要があります。

public void onLocationChanged(Location loc)

　位置情報が更新されるたびに呼び出されます。

public void onProviderDisabled(String arg0)

　ロケーションプロバイダ（ロケーションマネージャに登録されているGPS機能）が向こうになったとき呼び出されます。

public void onProviderEnabled(String arg0)

　ロケーションプロバイダが有効になったとき呼び出されます。

public void onStatusChanged(String arg0, int arg1, Bundle arg2)

　ロケーションプロバイダの状態が変化した際に呼び出されます。

public void onAccuracyChanged(Sensor arg0, int arg1)

　位置情報の精度が変化したときに呼び出されます。

・Location情報の更新への対応

　位置情報が更新されたときによびだされるonLocationChagnedで、Locationの情報を取得し保存したり、地図を更新したりします。

### 各種センサ

各種センサはSensorManagerという形でまとめられています。

SensorManagerには先に説明したとおり以下のTypeが存在します。

　・Accelerometer （加速度センサ、重力の影響を含む）

　・Temperature　（温度センサ）

　・Gravity　（重力センサ）

　・Gyroscope　（回転センサ）

　・Light　（照度センサ）

　・Linear Accelerometer　（加速度センサ、重力の影響を含まない）

　・Magnetic　Field (磁場センサ)

　・Orientation (方位センサ)

　・Pressure　（気圧センサ）

　・Proximity　(近接センサ)

　・Relative Humidity (相対湿度センサ)

仕様方法はすべてのセンサについて同じ、以下のようになります。

・SensorManagerの取得

android.content.Contextクラスのメソッド

　　　　getSystemService(Context.SENSOR\_SERVICE);

　　　を呼び出してSensorManagerを取得します。

　・Sensorの取得

　　　SensorManagerクラスのメソッド

　　　　　getDefaultSensor(Sensor.TYPE\_XXXXXX);

　　　を呼び出して、Sensorを取得します。

・SensorListenerの登録

　　SensorListenerインタフェースを実装したクラスを作成し、SsensorManagerに登録します。

　　ここではSensorに登録するのではなくSensorManagerに上記で取得したSensorのインスタンスとともに登録します。

　SensorListenerでは最小限以下のメソッドを実装する必要があります。

public void onSensorChanged(SensorEvent event)

　センサの情報が変化したときに呼び出されます。

・Sendor情報の更新の処理

　　onSensorChangedメソッドで、更新された情報などを保存したり、表示を更新したりします。

## 2.3　センサAPIの具体的な使用方法

センサAPIを使った具体的なアプリケーションの作成を行います。

このプログラムではSensorの情報とGPSの情報を取得し、サーバに送信することを目的とします。

### 2.3.1　プロジェクトの作成

新しいプロジェクトを作成します。

　Helloworldのプロジェクトを作成したときと同様にメニューからFile->Newを選択し以下のような値を設定します。

Application Name：SensorNetwork

Project Name：SensorNetwork

Package Name：jp.ac.nagoya\_u.engg.sensornetwork

PackageNameの部分に注意してください。

GooglePlayなどで公開するアプリケーションについてはPackageNameが同じものは作成できません。

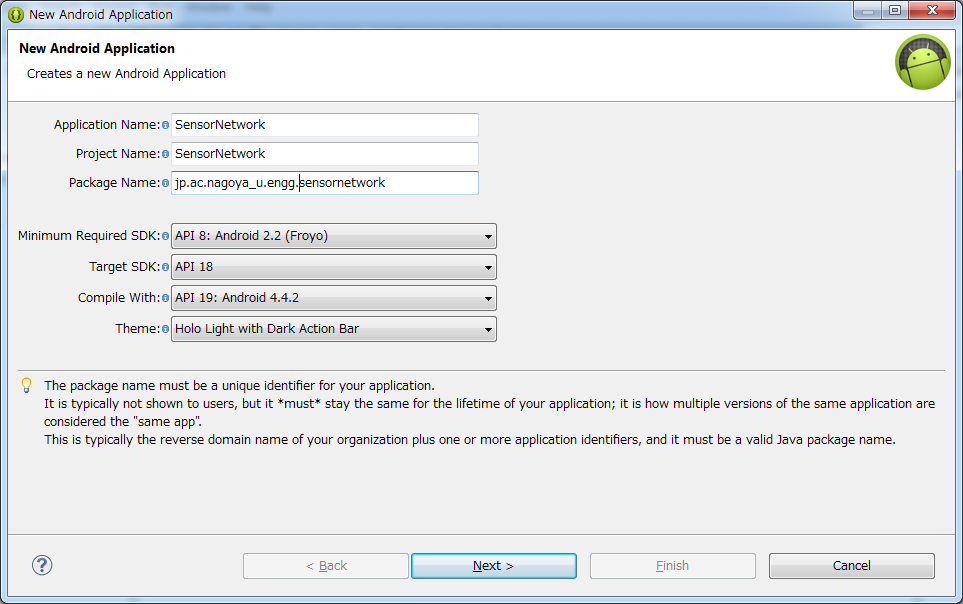
そこで、PackageNameについてはアプリケーションを作成する人が管理できるドメインを逆順で並べたものを使用するのが慣例となっています。

　ここでは、engg.nagoya-u.ac.jpというドメインを使用して作成することにします。もしGooglePlay等に公開する場合は必ずそのドメインの所有者の確認をとるか、自身が管理しているドメインを使用する様にしてください。

　つまりここで作成したアプリはそのままでは公開することはできませんのでご注意ください。

　注：nagoya-uなのにPackageNameではnagoya\_uになっていることが気になった方もいるかもしれません。PackgaeNameには – は使えないため、-の変わりに\_を使用しています。逆にドメイン名では\_は使えませんのでこれによって別の誰かのアプリと重複することはありません。

設定を行った画面は以下のようになります。

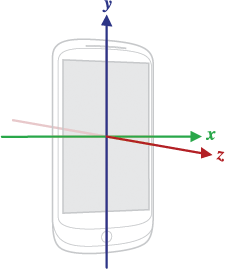


残りの設定はHelloworldと同じですので省略します。

### 2.3.2　画面デザイン

次に画面のデザインを行います。

Nexus7に搭載されている加速度センサの値を取得して表示したいので、画面上に次のようなイメージを表示し、図のX,Y,Zの矢印のあたりに数値を表示するようにしてみます。



この画像はAndroidDevleoperサイトの加速度センサの所で使用されているものです。

ライセンスはCreativeCommons 2.5となっています。

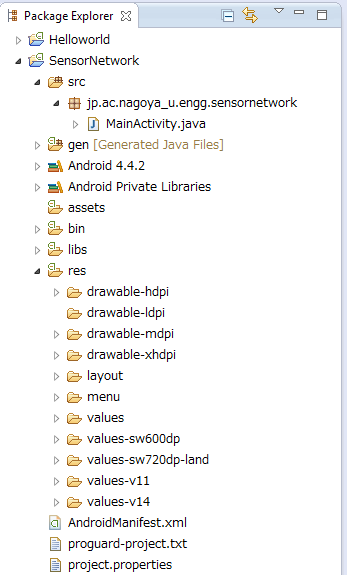
http://developer.android.com/license.html

　上記の画像のファイル名はaxis\_device.pngとなっています。

　サーバにおいてありますので、まずダウンロードしてください。

### 2.3.3　リソースのインポート

　ダウンロードしたフォルダーを開き、axis\_device.pngをドラッグして、Package ExplorerのSensorNetwork -> res -> drawable-xhdpi にドロップしてください。



このときファイルのコピーをするかどうかの確認ダイアログが表示されますのでCopy filesを選択してOKをクリックしてください。



データが大きい場合などはコピーしないこともできますが、後々プロジェクトをExportしたりする場合などに、トラブルの元となりますので基本はコピーする様にしたほうがいいでしょう。

### 2.3.4　レイアウト

続けて画面レイアウトを行います。

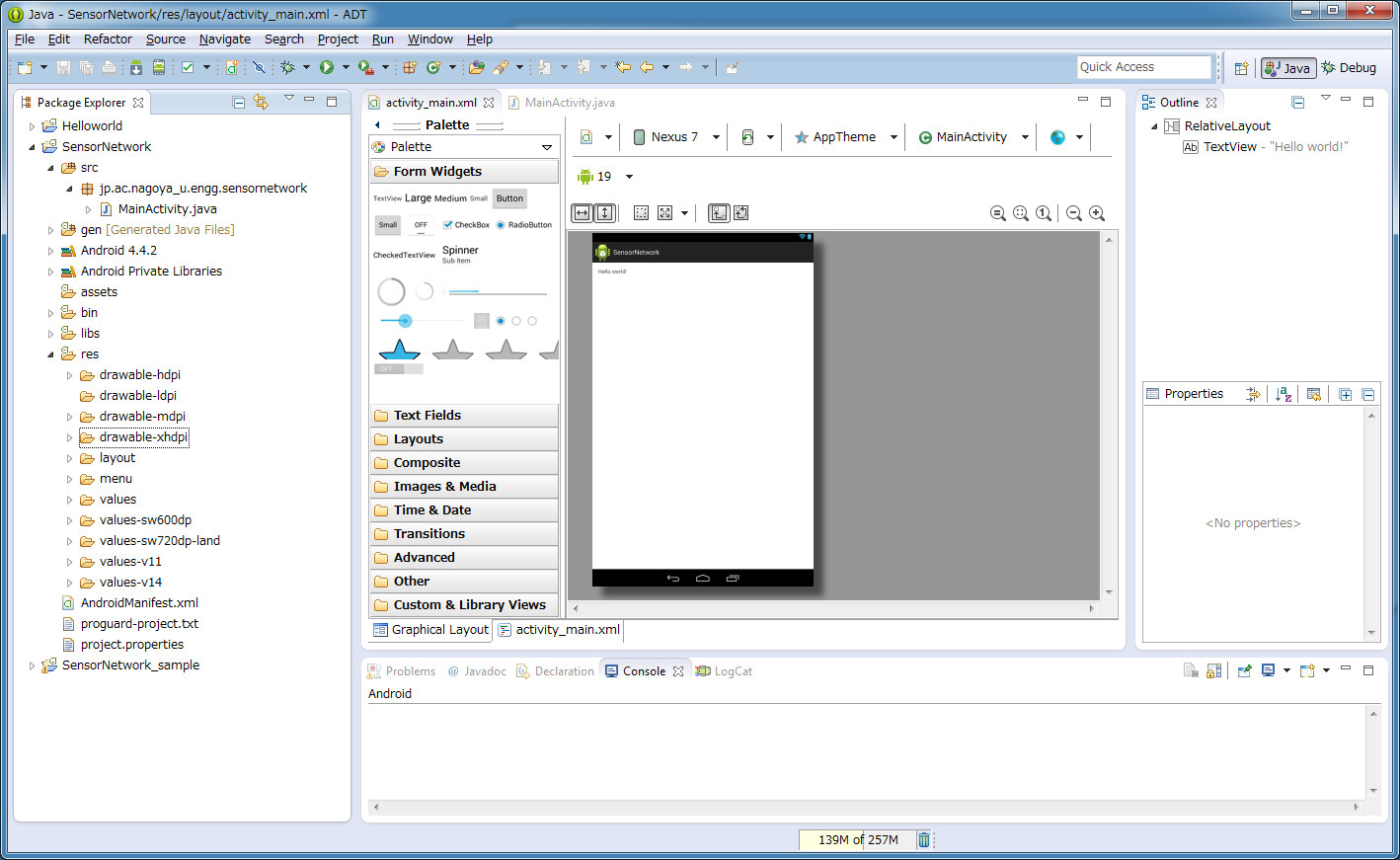
Eclipseの中央ペインのタブからactivity\_main.xmlをクリックします。

もし画面がグラフィカルインタフェースになっていなければ、ペインの下部のGraphical Layout タブをクリックします。

この時点では、レイアウトベースがNexus Oneになっていて画面サイズが小さいのでNexus７に合わせるためにペイン丈夫のNexus Oneをクリックし

Nexux 7(7.3”,800x1280:tvpi)を選択します。

この状態で以下のような画面になっていることを確認してください。



続けて画面レイアウトを行います。

まず画面上にあるHello worldの文字列を表示しているTextViewを削除します。

右ペインのOutline内のTextVeiw – “Hello world!” を右クリックしてDeleteを選択します。

次にイメージを画面中央に配置します。

中央ペインの中のPaletteの[Image & Media]というタブをクリックし、ImageViewを画面の中央にドラッグ＆ドロップします。

リソース選択のダイアログが表示されますので、先ほどインポートしたaxis\_deviceを選択します。



画面の中央に画像が配置されていればOKです。

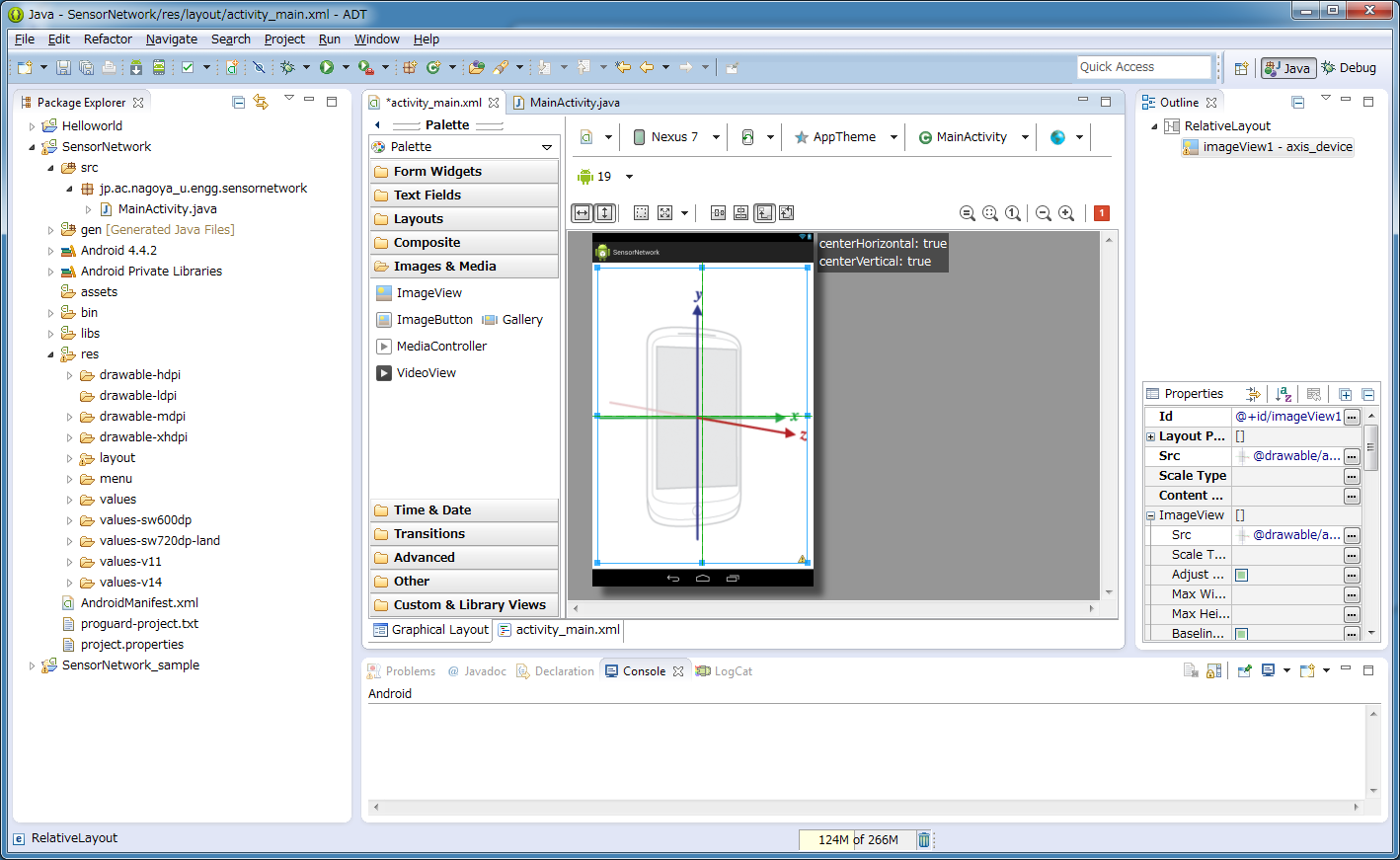


次に画像のサイズを変更します。

画面中央の画像を右クリックしLayout Width -> Match Parentを選択します。

続けて、同様にLayout Height -> Match Parentを選択します。

これで全体に表示されるようになります。

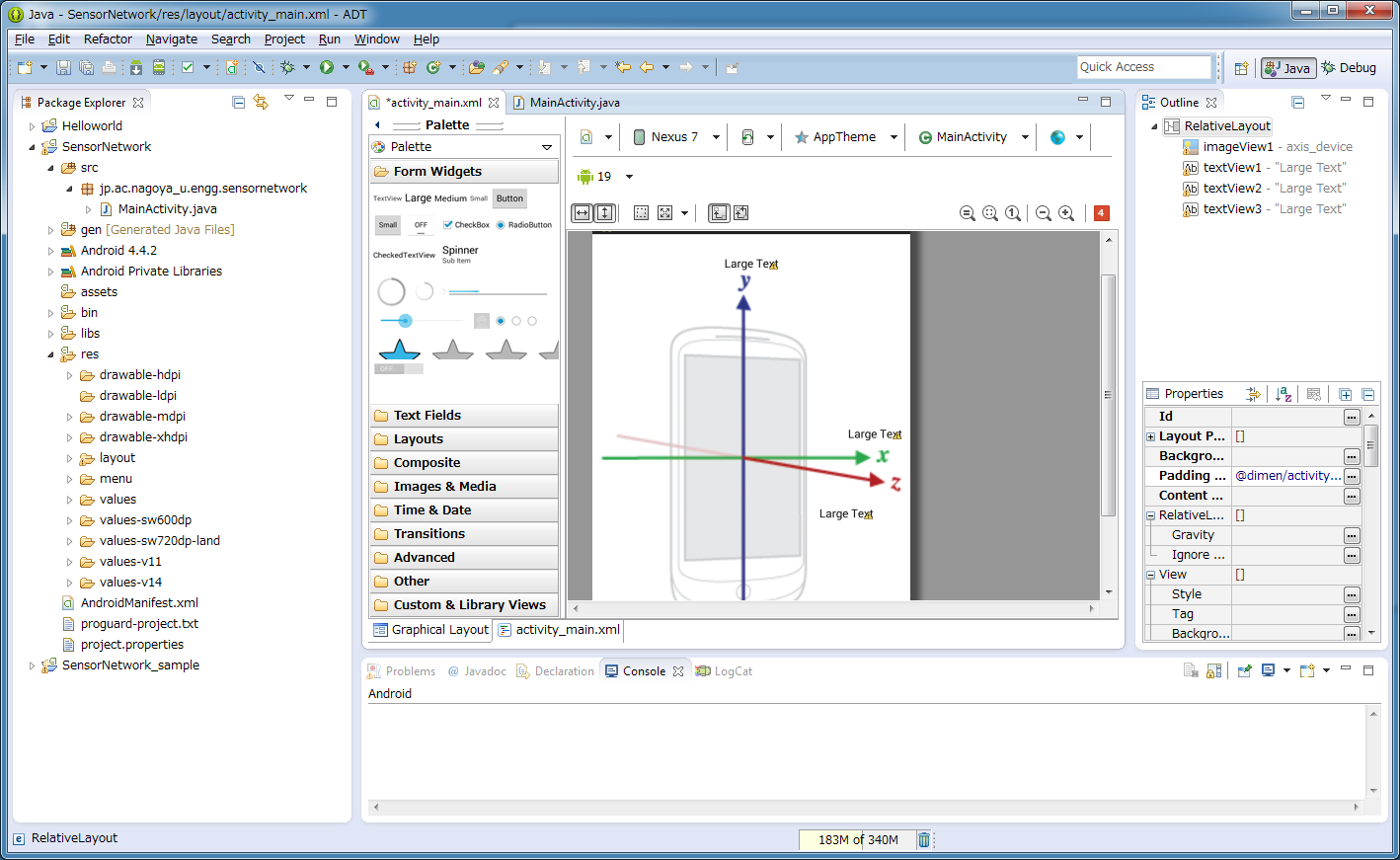


次にセンサの値を表示するためのTextViewを配置します。

PaletteからForm Widgetsを選びLargeをドラッグ＆ドロップして、画面のX,Y,Z

の近くに配置します。

位置は適当でかまいません。



次に現在の情報をサーバに転送するためのボタンを作成します。

PaletteのForm WidgetsからButtonを画面下部にドロップしてください。

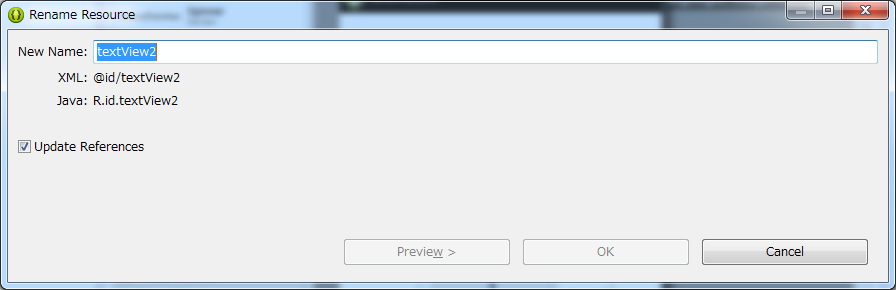
### 2.3.5リソースの編集

以上で画面の配置は完了です。

次にプログラムからアクセスしやすいように各画面上のアイテムに名前をつけて、いきます。

画面上のXの位置に配置したTextViewを右クリックしEdit IDを選びます。

以下のダイアログが表示されますので、TextView2の文字列を消して代わりにAccel\_X と入力します。



下のように変わったことを確認しOKをクリックします。



ここでXMLおよびJavaのプログラムからアクセスする際の書き方が表示されていることいに注意してください。

　このTextViewのIDはJavaのプログラムから見るとR.id.Accel\_Xという表記になります。

　あとでソースを作成するさいに必要となりますので、覚えておいてください。

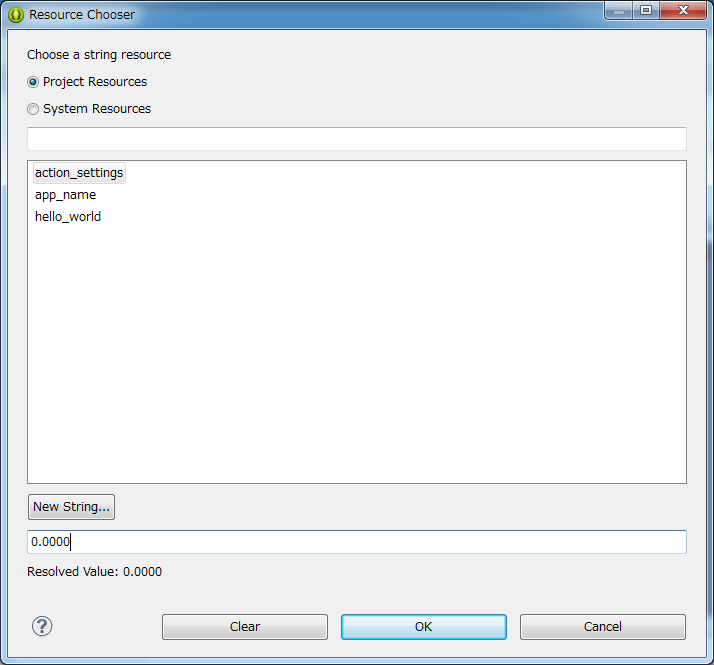
以下同様にY,Zの座標を表すTextViewのIDをAccel\_Y,Accel\_Zと変更します。

　変更が完了しても画面レイアウトは変化していません。

このままでは起動直後に一瞬だけLarge Textという文字が表示されてしまうので、見た目にうつくしくないので表示されている文字列を変更しておきます。

　先ほどと同様LargeTextとと書いてあるTextViewを右クリックし、今度は[Edit Text]を選択します。

テキストリソースを選択する画面が表示されますが、リソースにするほどのものでもないので、画面下部のテキストフィールドに0.0000と入力しOKをクリックします。



これをAccel\_Y,AccelZのそれぞれに対しても行います。

次にボタンのリソースID等を変更します。

画面上のボタンを右クリックしEdit IDを選択します。New Nameにsend\_buttonと入力しOKをクリックします。

　同様にEdit Textをクリックしてダイアログを開きます。

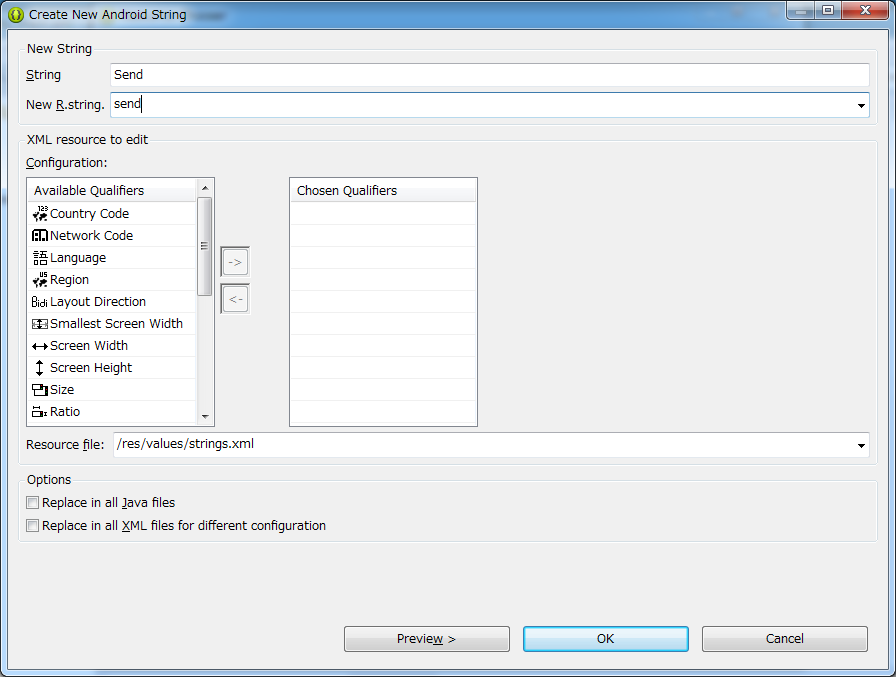
　ただし今度は少し操作が異なります。

　文字列を設定する際にはリソースを使用するべきなので、ここでは文字列リソースを作成してみます。

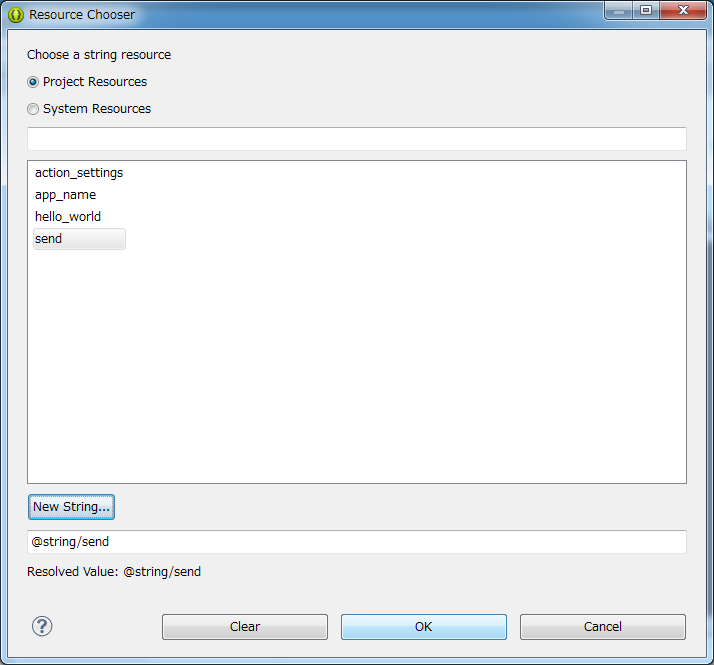
　リソースの作成にはNew Stringボタンをクリックします。

次のようなStringリソースを作成する画面が表示されますので、Stringに「Send」

New R.string. に　「send」と入力しOKをクリックします。

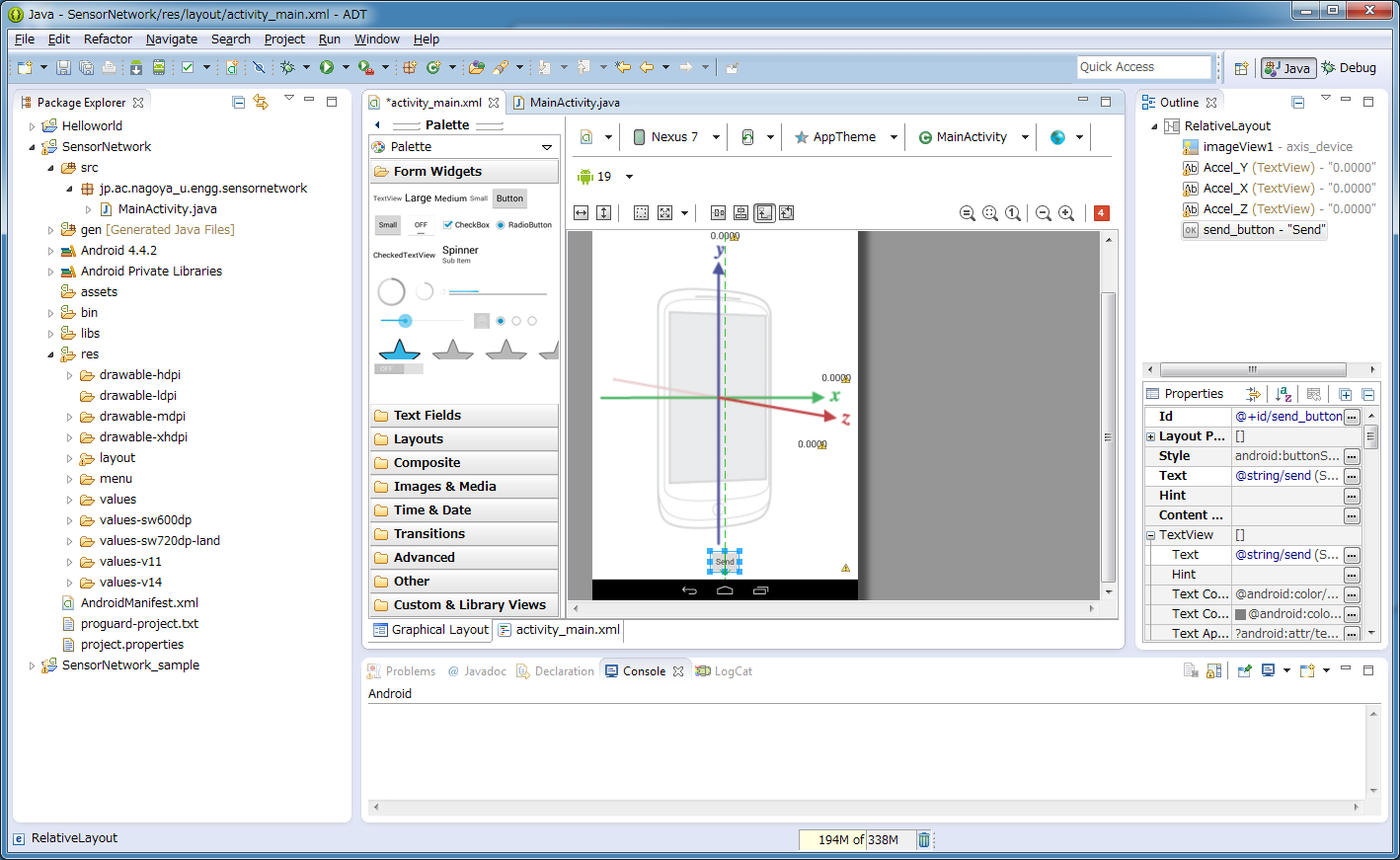


Resource Chooserの画面にsendという選択肢が増えていることを確認し、それを選択してOKをクリックします。



最終的に以下のような画面になっていることを確認してください。

変わっているのは、中央ペインの画面と右のOutlineの内容です。



以上で画面レイアウトは完了です。

一旦保存しましょう。ファイルメニューからSave Allを選択するかキーボードでCtrl+Sを押してください。

この時点でRunを実行すると画面ひょうじのみが行えますので、実機やエミュレータで画面表示を確認してみてください。

テキストの表示位置などが気になるようであれば、レイアウトを修正してください。

続けてプログラムを作成します。

### 2.3.6　プログラムの編集

今回のプログラムは以下のような機能を実装します。

　・継続的に自動的に位置情報を取得する

・継続的にセンサ情報を取得して画面の表示を更新する。

　・Sendボタンにタッチされたら、現時点の位置情報、加速度情報をサーバに転送する。

Androidのアプリケーションではパーミションを設定しないとアクセスできない機能がありますので、それらのパーミションの設定を行う必要があります。

このアプリケーションで使用するパーミッションはGPSの情報取得、ネットワークアクセスです。

それぞれ、ACCESS\_FINE\_LOCATION、INTERNETとなりますので、それを設定する必要があります。

#### 2.3.6.1　AndroidManifestの修正

AndroidManifestではアプリケーションのいろいろな設定を行います。　Manifestはこのアプリケーションはこういう機能を持っている、こういう許可を求めているなどの情報をAndroidのシステムに通知するためのものです。

　先に書いたとおりこのアプリケーションでは、GPS機能とネットワークを使用しますのでそれらの情報をManifestni記述します。

　プロジェクトを作成する際に指定した情報などもここに書かれているので、以下にAndroidManifest.xmlの内容を簡単に説明します。

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="jp.ac.nagoya\_u.engg.sensornetwork"  android:versionCode="1"  android:versionName="1.0" >  <uses-sdk  android:minSdkVersion="8"  android:targetSdkVersion="18" />  <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  <uses-permission android:name="android.permission.INTERNET" />  <application  android:allowBackup="true"  android:icon="@drawable/ic\_launcher"  android:label="@string/app\_name"  android:theme="@style/AppTheme" >  <activity  android:name="jp.ac.nagoya\_u.engg.sensornetwork.MainActivity"  android:label="@string/app\_name" >  <intent-filter>  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  </application>  </manifest> |

1行目の<?xml で始まる行は　このファイルがXMLファイルであることを表しています。基本的にAndroidアプリケーションで使用されるすべてのXMLファイルはこの行から始まっています。

2行目から最後までがManifestになります。

3行目のpackage= がこのアプリケーションのパッケージ名を表しています。

4行目のversionCodeはこのアプリの現在のバージョンを表しています。

　この数字は１から始まる整数で指定することになっておりGooglePlay等に登録した場合、アプリの更新があった場合はこの数字を大きくしていくことになっています。

　途中で数字が元に戻ったり、以前に登録したバージョンと同じだったりすると、新しい更新プログラムの登録に失敗しますので、GooglePlayに登録する際には必ず変更するようにします。

5行目のversionNameはこのアプリのバージョンを表す文字列です。

　ここは任意の文字列を書いてもかまいません。versionCodeとは違って必ず更新しんなければならないものでもありませんし、更新のルールは自分で決めることができます。

　ただこのバージョンの文字列がユーザがGooglePlayで見た時に表示されるものになりますので、あまり変な内容を書いているとユーザを混乱させるので、基本的にはバージョンが更新されたらより大きな値になるようにするのがいいでしょう。

　このアプリケーションはテストなので、1.0のままでかまいません。

8行目のminSdkVersionと9行目のtargetSdkVersionがプロジェクト作成時に指定したSDKバージョンです。

10行目はこのアプリケーションがGPS情報を使用することを宣言しています。

11行目はこのアプリケーションがネットワークを使用することを宣言しています。

この二つのパーミションはGooglePlayなどに登録した際、ユーザに確認として表示されるものです。

　14行目はアプリのアイコンのデータ名

　15行目はアプリの名称を表しています。

　それぞれリソースへのポインタですので実際に表示されるのはそれぞれのポインタが指すリソースの内容になります。

　16行目からactivityセクションとになります。

　このactivityセクションはこのアプリケーションがアクティビティを持っていること

　アクティビティのクラス名、アクティビティの名前を設定しています。

　19行目のintent-filterは1章で説明したインテントが投げられたとき、このアプリケーションがどういったインテントを処理することができるかを指定しています。

　ここでは直接アプリケーションの起動を指定された場合にのみ処理を行うようになっています。

　特定の処理を行うアプリケーションなどであれば、それに応じたインテントフィルタを設定することで、別のアプリから起動することも可能になります。

　最後に　categoryの所で、LAUNCHERを指定していますがこれをしていているアクティビティはランチャーのアプリリストにアイコンが並びます。

　10,11行目のパーミッションの追加が変更点です。

#### 2.3.6.2　画面上のアイテムの準備

画面上に並んでいるテキストビューなどの内容を更新するためには、アプリケーションの中からこれらのViewにアクセスする必要があります。そのためには、これらのViewのインスタンスを取得しなければいいけません。

1章で説明したとおりActivityはonCreateから実行されます。

そこでonCreateで画面上のTextViewおよびButtonを取得しアクセスできるようにメンバとして保存しておきます。

　画面上のアイテムのインスタンスを保持するためのメンバ変数を追加します。

|  |
| --- |
| private TextView mAccel\_X\_View;  private TextView mAccel\_Y\_View;  private TextView mAccel\_Z\_View;  private Button mSendButton; |

　次にonCreateの中でこれらのインスタンスをリソースから取得し設定します。

　onCreateのsetContentViewの直後に、以下のコードを追加します。

|  |
| --- |
| mAccel\_X\_View = (TextView)findViewById(R.id.Accel\_X);  mAccel\_Y\_View = (TextView)findViewById(R.id.Accel\_Y);  mAccel\_Z\_View = (TextView)findViewById(R.id.Accel\_Z);  mSendButton = (Button)findViewById(R.id.send\_button);  mSendButton.setOnClickListener(this); |

　次にボタンクリックの処理を追加します。

ボタンのクリック処理をMainActivityで行うために、MainActivityにimplements OnClickListener を追加します。

　onClickメソッドを追加します。

|  |
| --- |
| @Override  public void onClick(View arg0) {  } |

このメソッドは今は何もしません。実際にはボタンが押された時の処理を行いますので、後で処理を追加します。

必要なクラスをインポートします。

|  |
| --- |
| import android.view.View.OnClickListener;  import android.widget.Button;  import android.widget.TextView; |

以上で、テキストビューとボタンを使う準備が終わりました。

この状態でビルドして動作させることができますが、見た目には何も変化がないのでこのまま次の処理に写ります。

#### 2.3.6.3　加速度の取得

　センサの情報を取得し、TextViewに表示してみます。

まず、SensorMangerを取得します。

センサーマネージャーのインスタンスを保存するためのメンバを追加します。

|  |
| --- |
| private SensorManager mSensorManager = null; |

センサを初期化するメソッドを追加します。

|  |
| --- |
| private void initSensorManager(){  mSensorManager = (SensorManager)getSystemService(Context.SENSOR\_SERVICE);  Sensor = mSensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER);  if(sensor == null){  Toast.makeText(this, "Cannot find Accelerometer sensor.",Toast.LENGTH\_LONG).show();  }else{  mSensorManager.registerListener(this, sensor,SensorManager.SENSOR\_DELAY\_NORMAL);  }  Log.d(TAG,"SensorManager Initialized");  } |

このメソッドではgetSystemServiceをでSensorManagerを取得し、SensorManagerから加速度センサを取得します。

もし加速度センサが取得できなければ画面にエラーを表示します。

取得できれば、SensorManagerにSensorListenerとしてMainActivityをセットします。

そのときに、更新頻度をSENSOR\_DELAY\_NORMALに設定します。

ゲームなどで使用する場合は更新頻度を上げるためにSENSOR\_DELAY\_GAMEやSENSOR\_DELAY\_FASTEST等に設定します。

続けてMain Activity にSensorEventListenerをインプリメントします。

|  |
| --- |
| public class MainActivity extends Activity implements OnClickListener,SensorEventListener { |

追加したクラスをインポートします。

|  |
| --- |
| import android.hardware.Sensor;  import android.hardware.SensorEvent;  import android.hardware.SensorEventListener;  import android.hardware.SensorManager; |

MainActivityにSensorEventListenerのメソッドを追加します。

|  |
| --- |
| public void onSensorChanged(SensorEvent event) {  mAccel\_X = event.values[0];  mAccel\_Y = event.values[1];  mAccel\_Z = event.values[2];  mAccel\_X\_View.setText(String.format("%.6f",mAccel\_X));  mAccel\_Y\_View.setText(String.format("%.6f",mAccel\_Y));  mAccel\_Z\_View.setText(String.format("%.6f",mAccel\_Z));  }  public void onAccuracyChanged(Sensor arg0, int arg1) {  // TODO Auto-generated method stub  } |

onSensorChangedはセンサの値が変わるたびに呼び出されます。

加速度のデータを保持するためのメンバを追加します。

|  |
| --- |
| private Float mAccel\_X;  private Float mAccel\_Y;  private Float mAccel\_Z; |

onCreateからinitSensorManager()を呼び出します。

以上加速度センサの値が変わると画面上の表示を更新することができるようになりました。

加速度センサはエミュレータでは使用できないので、実機でテストをします。

起動したら画面が表示されて、各軸の加速度が表示され、本体に向きを変えたり、動かすことで加速度が変化することを確認してください。

#### 2.3.6.4　GPS情報の取得

続けてＧＰＳから場所を取得する処理を追加します。

やりかたはほとんど同じですので、書き換えた部分だけを列挙します。

追加クラスのインポート

|  |
| --- |
| import android.location.Location;  import android.location.LocationListener;  import android.location.LocationManager; |

メンバ変数の追加

|  |
| --- |
| private LocationManager mLocationManager = null;  private Double mLatitude = 0.0;  private Double mLongtitude = 0.0; |

MainActivityへのLocationListenerのインプリメント

|  |
| --- |
| public class MainActivity extends Activity implements SensorEventListener,OnClickListener,LocationListener { |

initLocationManagerメソッドとLocationListenerのメソッドの追加

|  |
| --- |
| private void initLocationManager(){  mLocationManager = (LocationManager)this.getSystemService(Context.LOCATION\_SERVICE);  mLocationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 0, 0, this);  }  @Override  public void onLocationChanged(Location loc) {  mLatitude = loc.getLatitude();  mLongtitude = loc.getLongitude();  }  @Override  public void onProviderDisabled(String arg0) {  // when provider is disabled  ｝  @Override  public void onProviderEnabled(String arg0) {  // when provider is enabled  }  @Override  public void onStatusChanged(String arg0, int arg1, Bundle arg2) {  // when status changed  } |

以上で、GPSからの情報を取得する処理の追加ができました。

### 2.3.7　サーバへの送信

最後にサーバにデータを送信するための仕組みを組み込みます。

ここまでの処理では特別な処理はほとんどありませんでしたが、サーバとの通信を行う際には、いくつか注意が必な点があります。

Androidのユーザインタフェースを担っているプロセス(UIプロセス)の中で通信などのレスポンスを待つ必要のある処理を行うと通信状況が悪い場合など、ユーザの操作にたいして反応ができなくなってしまう可能性があるため、現在のAndroid端末ではそういった処理をUIプロセスで行うことは禁止されています。

　そのため、通信処理を行うためには、そのために新しいスレッドを起動する必要があります。

　また通信処理などはエラーが発生しやすく、そのエラーも深いところで発生することが多いので、エクセプションの処理をちゃんと行わないといけない場合が多くあります。

　それらの処理を行うため、見た目に煩雑なプログラムになりますが、基本的な流れはそれほど面倒なものではありません。

　まずはじめに基本的な流れを説明し、それが実際のコードでどのように表現することになるかを説明していきます。

サーバとの通信を行うためにはサーバ側と通信のプロトコルを設計しなければなりませんが、今回はあくまで実運用ではなくサンプルとしてのプログラムと割り切って、HTTP GETのパラメータで値を渡す形で処理を行います。

　そのため送信するデータはすべてURLとして表現されますので、見た目にわかりやすいものになると思います。

　●処理の流れ

　　・送信すべきデータの準備

　　・URLを指定してHTTPで通信を行います。

　　・エラー処理など

送信すべきデータを準備するためには、URLの文字列を生成することになるのですがURLに入れてよい文字、URLでエスケープしなければならない文字などいろいろな制約があります。

また不用意な文字列を渡すことによってほかのURLと解釈されてしまうような可能性もありますので、URLの文字列をただの文字列連結で作成するのはあまりよい方法ではありません。

ここではAndroidのフレームワーク　android.net.Uriで用意されているUri.Builderという仕組みを使ってURLを生成します。

Uri.BuilderはURIに必要な要素を加えていくことで、最終的なURIを生成してくれます。必要名要素は、主に以下の項目となります。

Scheme ：HTTP,HTTPSなど

Authority ：サーバのロケーション、認証情報など

Path：URIの場所

QueryParameter：引数

これらの処理を行うbuildURLというメソッドを追加します。

|  |
| --- |
| private String buildURL(){  Uri.Builder builder = new Uri.Builder();  builder.scheme(PROTOCOL);  builder.authority(DOMAIN);  builder.path(PATH);  builder.appendQueryParameter(ID, mUserID);  builder.appendQueryParameter(PASSWORD,mPassword);  builder.appendQueryParameter(LATITUDE,Double.toString(mLatitude));  builder.appendQueryParameter(LONGTITUDE,Double.toString(mLongtitude));  builder.appendQueryParameter(ACCEL\_X,Float.toString(mAccel\_X));  builder.appendQueryParameter(ACCEL\_Y,Float.toString(mAccel\_Y));  builder.appendQueryParameter(ACCEL\_Z,Float.toString(mAccel\_Z));  builder.appendQueryParameter(DATE,DateFormat.format("yyyy-MM-dd",Calendar.getInstance()).toString());  builder.appendQueryParameter(TIME,DateFormat.format("kk:mm:ss",Calendar.getInstance()).toString());  return builder.toString();  } |

はじめにUri.Builderのインスタンスを生成し、scheme,authority,path,parameter を順番に設定していきます。

すべてのパラメータの設定が終わったら、最後にbuilder.toString()をすることで、URLを返します。

このメソッドをonClick()から呼び出してURLを生成します。

次にURLを指定してHTTPの通信を行います。

AndroidのアプリからHTTPで通信するために最も簡単な方法はHttpURLConnectionクラスを使用することだと思います。

ここではHttpURLConnectionを使用して、指定したURLにアクセスするだけのプログラムを作成します。

先に述べたようにAndroidではUIプロセスの中でネットワーク通信処理などは行えないようになっています（以前のバージョンでは行えました）

そこで、非同期通信用のスレッドを起動して処理を行うためのユーティリティクラスが用意されています。

今回はAsyncTaskというクラスを使用してネットワーク通信を行います。

具体的にはAsyncTaskというクラスを継承したAsyncNetworkAccessというクラスを作成し、そのクラスのdoBackGroundメソッドに処理を記述することで別プロセスでの実行を行います。

AsyncTaskは三つの引数の型を指定して継承します。

一つ目はdoInBackgroundメソッドの引数

二つ目はonProgressUpdateメソッドの引数

三つ目はonPostExecuteの引数

今回はdoInBackgroundのみ使用しますので二つ目以降は適当な型でかまいまんので、以下のようにAsyncNetworkAccessクラスを作成します。

|  |
| --- |
| class AsyncNetworkAccess extends AsyncTask<String,Integer,String>{  public AsyncNetworkAccess(Context context){  }  @Override  protected String doInBackground(String... params) {  String line;  URL url=null;  HttpURLConnection uc=null;  try {  url = new URL(params[0]);  uc = (HttpURLConnection) url.openConnection();  uc.setRequestMethod("GET");  uc.setRequestProperty("Accept-Language", "ja");  uc.setDoOutput(true);  BufferedReader reader;  reader = new BufferedReader(new InputStreamReader(uc.getInputStream()));  while ((line = reader.readLine()) != null) {  Log.d(TAG,line);  }  reader.close();  uc.disconnect();  } catch (Exception e) {  e.printStackTrace();  }  return null;  }  } |

このクラスはMainActivityの内部クラスとして作成します。

はじめにコンストラクタは特にやることもありませんので、何も行いません。

メインの処理はdoInBackgroundの中で行います。

doInBackgroundの中の処理は、HttpURLConnectionを作成し、引数として受け取った文字列をURLに変換してHTTP GETでアクセスを行うだけです。

これらのメソッドはそれぞれエクセプションを起こす可能性がありますので、try/catchでくくって、まとめて例外処理を行っています。

本来はエラーの種類を区別して例外処理すべきところですが、エラーが発生したら一律何もせず終わる用にしています。

|  |
| --- |
| @Override  public void onClick(View arg0) {  String url = buildURL();  AsyncNetworkAccess task = new AsyncNetworkAccess(this);  task.execute(url);  } |

最後にonClickメソッドでAsyncNetworkAccessのインスタンスを生成し、task.executeを実行します。

必要なインポートを追加したすべてのソースは以下のようになります。

|  |
| --- |
| package jp.ac.nagoya\_u.engg.sensornetwork;  import java.io.BufferedReader;  import java.io.InputStreamReader;  import java.net.HttpURLConnection;  import java.net.URL;  import java.util.Calendar;  import android.hardware.Sensor;  import android.hardware.SensorEvent;  import android.hardware.SensorEventListener;  import android.hardware.SensorManager;  import android.location.Location;  import android.location.LocationListener;  import android.location.LocationManager;  import android.net.Uri;  import android.os.AsyncTask;  import android.os.Bundle;  import android.app.Activity;  import android.content.Context;  import android.text.format.DateFormat;  import android.util.Log;  import android.view.Menu;  import android.view.View;  import android.view.View.OnClickListener;  import android.widget.Button;  import android.widget.TextView;  import android.widget.Toast;  public class MainActivity extends Activity implements OnClickListener,SensorEventListener,LocationListener {  final String TAG = "SensorNetwork";  final String PROTOCOL = "http";  final String DOMAIN = "www.watana.be";  final String PATH = "sensornetwork.html";  final String ID = "id";  final String PASSWORD = "pw";  final String LATITUDE = "lat";  final String LONGTITUDE = "lon";  final String ACCEL\_X = "ax";  final String ACCEL\_Y = "ay";  final String ACCEL\_Z = "az";  final String DATE = "date";  final String TIME = "time";    private TextView mAccel\_X\_View;  private TextView mAccel\_Y\_View;  private TextView mAccel\_Z\_View;  private Button mSendButton;  private SensorManager mSensorManager = null;  private Float mAccel\_X;  private Float mAccel\_Y;  private Float mAccel\_Z;  private LocationManager mLocationManager = null;  private Double mLatitude = 0.0;  private Double mLongtitude = 0.0;  private String mUserID = "1111";  private String mPassword = "pw";  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  mAccel\_X\_View = (TextView)findViewById(R.id.Accel\_X);  mAccel\_Y\_View = (TextView)findViewById(R.id.Accel\_Y);  mAccel\_Z\_View = (TextView)findViewById(R.id.Accel\_Z);  mSendButton = (Button)findViewById(R.id.send\_button);  initSensorManager();  initLocationManager();  mSendButton.setOnClickListener(this);  }  @Override  public boolean onCreateOptionsMenu(Menu menu) {  // Inflate the menu; this adds items to the action bar if it is present.  getMenuInflater().inflate(R.menu.main, menu);  return true;  }  @Override  public void onClick(View arg0) {  String url = buildURL();  AsyncNetworkAccess task = new AsyncNetworkAccess(this);  task.execute(url);  }    private void initSensorManager(){  mSensorManager = (SensorManager)getSystemService(Context.SENSOR\_SERVICE);  Sensor sensor = mSensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER);  if(sensor == null){  Toast.makeText(this, "Cannot find gravity sensor.",Toast.LENGTH\_LONG).show();  }else{  mSensorManager.registerListener(this, sensor,SensorManager.SENSOR\_DELAY\_NORMAL);  }  }  @Override  public void onAccuracyChanged(Sensor arg0, int arg1) {    }  @Override  public void onSensorChanged(SensorEvent event) {  mAccel\_X = event.values[0];  mAccel\_Y = event.values[1];  mAccel\_Z = event.values[2];  mAccel\_X\_View.setText(String.format("%.6f",mAccel\_X));  mAccel\_Y\_View.setText(String.format("%.6f",mAccel\_Y));  mAccel\_Z\_View.setText(String.format("%.6f",mAccel\_Z));  }    private void initLocationManager(){  mLocationManager = (LocationManager)this.getSystemService(Context.LOCATION\_SERVICE);  mLocationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 0, 0, this);  }  @Override  public void onLocationChanged(Location loc) {  mLatitude = loc.getLatitude();  mLongtitude = loc.getLongitude();  }  @Override  public void onProviderDisabled(String arg0) {  // when provider is disabled  }  @Override  public void onProviderEnabled(String arg0) {  // when provider is enabled  }  @Override  public void onStatusChanged(String arg0, int arg1, Bundle arg2) {  // when status changed  }  private String buildURL(){  Uri.Builder builder = new Uri.Builder();  builder.scheme(PROTOCOL);  builder.authority(DOMAIN);  builder.path(PATH);  builder.appendQueryParameter(ID, mUserID);  builder.appendQueryParameter(PASSWORD,mPassword);  builder.appendQueryParameter(LATITUDE,Double.toString(mLatitude));  builder.appendQueryParameter(LONGTITUDE,Double.toString(mLongtitude));  builder.appendQueryParameter(ACCEL\_X,Float.toString(mAccel\_X));  builder.appendQueryParameter(ACCEL\_Y,Float.toString(mAccel\_Y));  builder.appendQueryParameter(ACCEL\_Z,Float.toString(mAccel\_Z));  builder.appendQueryParameter(DATE,DateFormat.format("yyyy-MM-dd",Calendar.getInstance()).toString());  builder.appendQueryParameter(TIME,DateFormat.format("kk:mm:ss",Calendar.getInstance()).toString());  return builder.toString();  }  class AsyncNetworkAccess extends AsyncTask<String,Integer,String>{  public AsyncNetworkAccess(Context context){  }  @Override  protected String doInBackground(String... params) {  String line;  URL url=null;  HttpURLConnection uc=null;  try {  url = new URL(params[0]);  uc = (HttpURLConnection) url.openConnection();  uc.setRequestMethod("GET");  uc.setRequestProperty("Accept-Language", "ja");  uc.setDoOutput(true);  BufferedReader reader;  reader = new BufferedReader(new InputStreamReader(uc.getInputStream()));  while ((line = reader.readLine()) != null) {  Log.d(TAG,line);  }  reader.close();  uc.disconnect();  } catch (Exception e) {  e.printStackTrace();  }  return null;  }  }  } |

センサから取得したデータをサーバに送信するためのプログラムは完成です。

# 3　データ保存

センサーデータを取得してサーバに送信するだけであれば、ここまでで説明したアプリケーションで十分です。

しかし、実際のアプリケーションでは送信するたびにボタンを押すのは現実的ではありませんし、言って期間毎にデータを取得して保存し、あとでまとめて送信するという形の方が現実的でしょう。

そこでAndroidで内部にデータを保存する方法について少し説明を行います。

## 3.1　データ保存方法の検討

Androidのアプリケーションでデータを保存する方法は、「プリファレンス」「ファイル」「データベース」と大きく分けて3つあります。

それぞれについて簡単に説明すると以下のようになります。

・プリファレンス

ソフトの設定情報など、簡単な名前と値という組み合わせで管理できる情報を保存します。情報は本体メモリに記録され簡単に保存、更新、取り出しができます。大きなデータを保存したり、検索を行ったりするのには向いていません。

・ファイル

パソコンのアプリケーションと同様に独自にファイルを開いて保存する方法です。データ構造などを自分で設計できるため、自由にデータを保存することができます。ただし、単純なファイルなので、中身については自分ですべて管理しなければなりませんし、検索や更新は自分でプログラムを作成して実行する必要があります。

・データベース

Androidでは「SQLite」というデータベースを簡単に使うことができます。「SQLite」を使用するためのインタフェースなども用意されており、検索や更新を行う必要のあるデータを格納するのに向いています。標準で用意されていているとはいうものの、データベースのテーブル設計が必要になること、SQLなどの知識が必要になることなどから、プリファレンスやファイルに比べると面倒です。

プログラムの難易度という点ではプリファレンスが最も単純で、次にファイル、データベースが一番面倒だと思います。

　それぞれメリットデメリットがありますので、それぞれの目的にあった方法を選択してください。

　それぞれの具体的な使い方については参考文献[1]で説明してありますので必要があればそちらを参照してください。

# 4　ネットワーク通信

サンプルプログラムではネットワーク通信といいつつ、ごく単純なHTTPのパラメータで値を渡す方法で実装しました。

　しかし実際にはもっと複雑なデータや大量なバイナリデータなどを送受信したいケースも考えられます。

　その場合使用するプロトコルとしてGETでは限界があります。

　そこである程度以上のデータの送受信を行う際にはHTTPのPOSTメソッドを使用する様なことも必要になってきます。

　もちろんデータの種類や量、頻度によってHTTPではなくほかのプロトコルを使用する必要がある場合も考えられるでしょう。

## 4.1　サーバへの転送フォーマット

　まず基本的なプロトコルとしてHTTPを使用することを前提としても、データのフォーマットについてはいろいろなものが考えられます。

　現在よく使われているフォーマットはXMLまたはJSONと呼ばれるテキスト型のフォーマットです。

　どちらのフォーマットもデータを階層化し、名前とデータの組み合わせで表現するものですので、ほとんどのデータベースの構造をそのままの形で表現することができます。

　人間が見た場合はXMLの方が読みやすいですが、データサイズはJSONの方が若干小さくできるようです。

　どちらにしても、Androidには内部にパーサのライブラリがありますので、処理を行う際はそれらのパーサを使用しますので、自前のプログラムで解析を行っていく必要はありません。

## 4.2　認証

　ネットワークを通してデータのやりとりをするためには必ず認証の処理が必要となります。WebサーバのBASIC認証の様なプレーンなパスワードによる認証で十分な場合もありますが、パスワードの漏洩などのリスクを考えると、十分にセキュアな認証方法をとる必要があります。

　現在Google等のサービスプロバイダで採用されているのがOAuthと呼ばれるチケット交換型の認証方式です。

　OAuthのメリットは認証を行うサーバと、認証された処理を行うサービスが別でもかまわないところです。

たとえば、GoogleカレンダーのデータにアクセスするようなWebサービスやアプリケーションを作成することを考えてみましょう。（今後Webサービスも含めてアプリケーションと呼びます。）

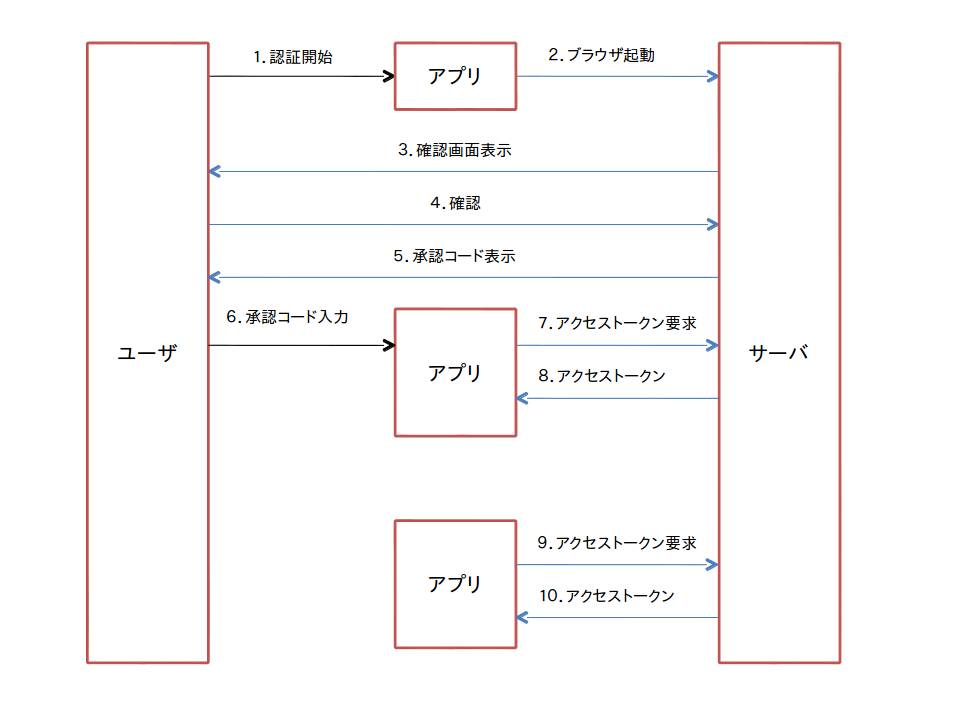
これまでの認証方式ではGoogleカレンダーのデータにアクセスするためにはGoogleのIDとパスワードをアプリケーションに設定してやる必要がありました。

　アプリケーションを自分で作成するか、アプリケーションの作成者を１００％信用できるのであればこれでも問題内かもしれませんが実際には作成者の意図しない方法でアプリケーションからパスワードが漏洩したり、同じアプリケーションなのに運営者が変わったりする可能性が否定できません。

　また、Google等はカレンダー以外にもいくつものサービスを同じアカウントシステムで提供しているのでカレンダーのデータだけを使いたいアプリケーションにGmailへのアクセスまで許可してしまうことになるのはあまりいい方法とはいえません。

　OAuthなどの認証方式はアプリケーション側には一切パスワードなどの情報を登録せず、アプリケーションが自動的にGoogleなどのサービスに接続し、許可されたサービスだけを使用することができるような仕組みを提供しています。

　具体的には以下の様な流れで認証を行います。



この図はOAuthの承認コードを使用した認証の様子です。

具体的な流れは以下の様になります。

まずはじめにユーザはアプリケーションに対してサーバへの認証要求を出すように指示します。

アプリケーションは自分の持っているアプリケーショントークンをつけて、ブラウザを起動し、認証画面を表示させます。

　アプリケーションは事前にサーバにたいして使用するデータの範囲や認証方法をサーバに登録してありますので、この時点でサーバから表示されたブラウザの認証画面では、どういうアプリケーションがどういったサービスを使用するかが表示されています。

　ユーザはアプリケーション名や提供者情報、アクセスを許す範囲などがアプリケーション側の説明と齟齬がないことを確認し、認証を行います。

　サーバは認証が正しく認証が行われてことを確認し、アプリケーション用の承認コードを発行します。

　発行された承認コードをアプリケーションに入力すると、アプリケーションはサーバに対して承認コードを送りアクセストークンを受け取ります。

　これ以降アプリケーションはこのアクセストークンを使用してサーバに対してアクセスを行うことができます。

　このアクセストークンはアプリケーションとユーザとサービスごとに固有で別のアプリケーションが同じトークンを使用することもできませんし、別のユーザが使用することももちろんできません。

　また、アクセストークンには有効期限があり、一定時間すぎたアクセストークンは二度と使用することができなくなります。

　アプリケーションは初回認証時にアクセストークンを更新するためのキーを受け取っていますので、アクセストークンの有効期限が切れた場合は新しいキーを更新することで、サービスを継続することができます。

　これによって、アプリケーションはその内部にユーザのパスワードなどの情報を持つ必要が無くなり、パスワード情報の管理に気を遣う必要もなくなります。

　ユーザはアプリケーションを使用なくなったら、サーバにアクセスして認証を停止することもできますので、一度使ったアプリケーションがずっとアクセス可能な状態で残るという問題も発生しません。

　OAuth以外にもこういったアクセス制御の仕組みはありますが、今後はOAuth2.0が主流になってくるのではないかと思われます。

## 4.3　同期管理

　サーバとのデータのやりとりを行う場合はローカル側のデータとサーバ側のデータの同期を管理することが重要になります。

　サーバ、クライアント両方でデータの更新が行えるような場合は特に、どちらのデータが新しくて、どちらのデータが正しいのかというのを正しく管理しておく必要があります。

　またデータの削除についても、どちらかで削除したデータは相手側でも削除する必要がありますが、場合によっては一度削除したデータを復活させたものの可能性もあるので、その場合は削除されているからといって勝手に消してしまっても困ります。

　データの同期を管理するためには各データについて、更新のタイムスタンプなどを持って、削除した場合でもデータを削除するのではなく、削除フラグで対応するなどの処理が必要です。

　たとえば、Googleカレンダーのデータと同期する様なクライアント上のカレンダーアプリがあるとすると、データの状況は多岐にわたります。

　状況を整理したのが次の表です。

　○は有効なデータがある。

　△は削除済みデータがある。

　×はデータが存在しない。

という状況を表しています。

「更新の有無」の項は、前回同期してからサーバ側端末側で更新があったかどうかを表します。

　受信の項はGoogleカレンダーにアクセスした際に受信したかどうかを表します。

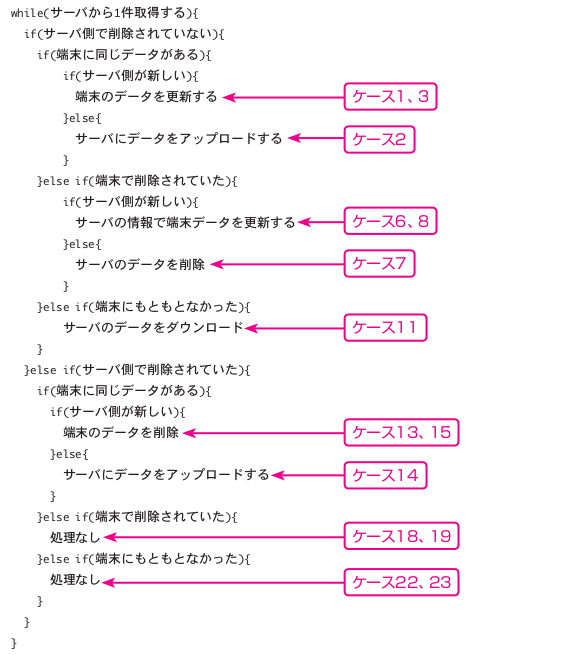
　これはサーバ側にデータが存在し、前回同期以降に更新、削除などが行われたことを表します。

状態をすべて確認していくと、以下の様に27通りの組み合わせが発生します。

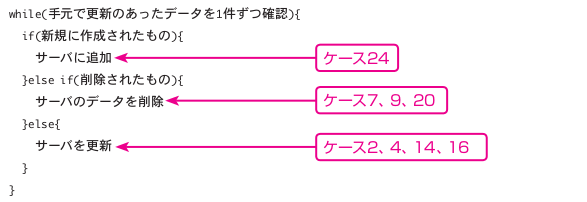
それぞれについてどういった処理を行うべきかが処理の欄に書いてあります。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | データの状態 | | 更新の有無 | |  |  |  |
|  | サーバ | 端末 | サーバ | 端末 | 受信 | 新 | 処理 |
| 1 | ○ | ○ | ○ | ○ | ○ | サーバ | ダウロード |
| 2 | ○ | ○ | ○ | ○ | ○ | 端末 | アップロード |
| 3 | ○ | ○ | ○ | × | ○ | サーバ | ダウンロード |
| 4 | ○ | ○ | × | ○ | × | 端末 | アップロード |
| 5 | ○ | ○ | × | × | × |  | なし |
| 6 | ○ | △ | ○ | ○ | ○ | サーバ | ダウンロード |
| 7 | ○ | △ | ○ | ○ | ○ | 端末 | サーバ側データ削除 |
| 8 | ○ | △ | ○ | × | ○ | サーバ | ダウンロード |
| 9 | ○ | △ | × | ○ | × | 端末 | サーバ側データ削除 |
| 10 | ○ | △ | × | × | × |  | 異常 |
| 11 | ○ | × | ○ | × | ○ | サーバ | ダウンロード |
| 12 | ○ | × | × | × | × |  | 異常 |
| 13 | △ | ○ | ○ | ○ | ○ | サーバ | 端末側データ削除 |
| 14 | △ | ○ | ○ | ○ | ○ | 端末 | アップロード |
| 15 | △ | ○ | ○ | × | ○ | サーバ | 端末側データ削除 |
| 16 | △ | ○ | × | ○ | × | 端末 | アップロード |
| 17 | △ | ○ | × | × | × |  | 異常 |
| 18 | △ | △ | ○ | ○ | ○ |  | なし |
| 19 | △ | △ | ○ | × | ○ |  | なし |
| 20 | △ | △ | × | ○ | × | 端末 | サーバ側データ削除 |
| 21 | △ | △ | × | × | × |  | なし |
| 22 | △ | × | ○ | × | ○ |  | なし |
| 23 | △ | × | × | × | × |  | なし |
| 24 | × | ○ | × | ○ | × | 端末 | アップロード |
| 25 | × | ○ | × | × | × |  | 異常 |
| 26 | × | △ | × | ○ | × | 端末 | なし |
| 27 | × | △ | × | × | × |  | なし |

この27通りの組み合わせをすぐそのままプログラムにしていくのは難しいので、まず疑似コードで表現したのが次の図になります。



サーバからダウンロードされなかったデータ(サーバ側で更新がなかったもの)については別に処理が必要です。



この疑似コードで処理を行わなければならないすべてのパターンが網羅されていることを確認して、実際のプログラムを作成することになります。

# 5　その他(補足)

## 5.1　Debug/DDMS

ここでは簡単にAndroidSDKを使用したソースラインデバッグの方法およびDDMSの簡単な使い方を説明します。

### 5.1.1　デバッグ

ソースコードの表示とブレークポイントの設置

簡単なデバッグの操作をやってみましょう。デバッグのためにプログラムのソースを見る必要があります。

　2章で作成したアプリケーションをターゲットにデバッグを行ってみます。

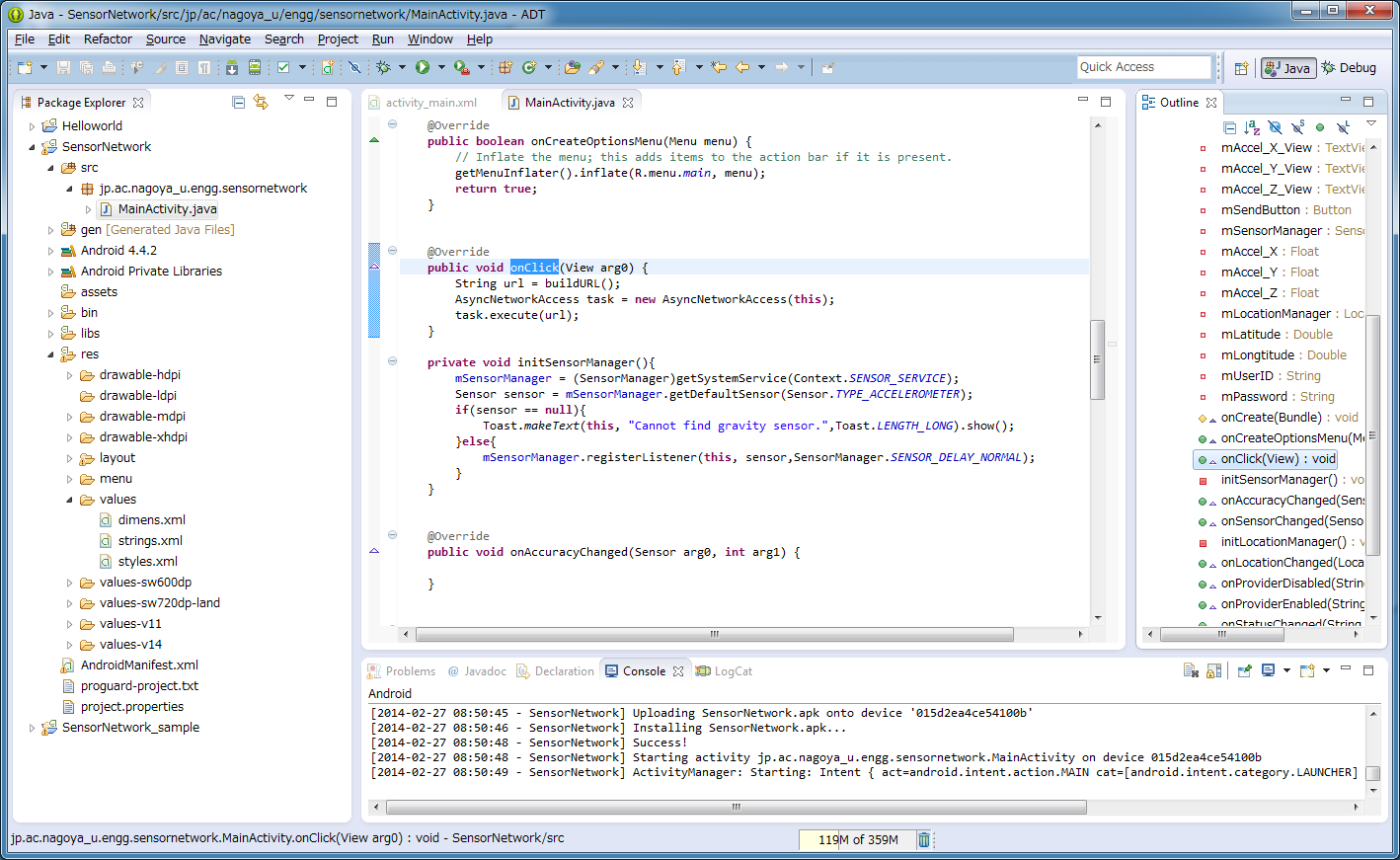
まずソースを開きます。

　ボタンにタッチしたらそこで処理を一旦止めるようにしてみましょう。

　画面右のOutlineのペインを見ると実装されている関数が並んでいると思います。

　そこからonClickをクリックしてください。

　中央のペインにソースが表示されてonClickの関数が選択されます。



この状態でonClickの中のString url = buildURL()の行の左の水色のエリアでダブルクリックします。

青い●が表示されているのが確認できると思います。

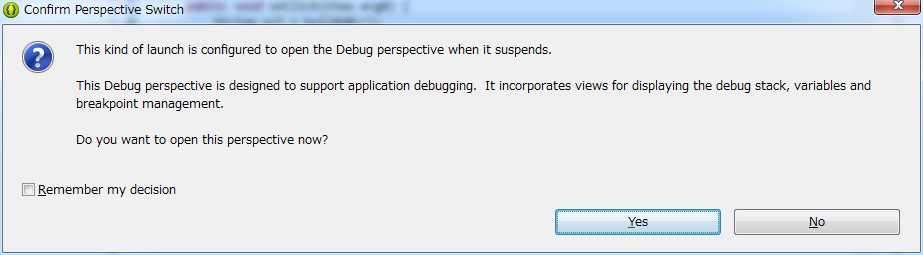
これでこの行にブレークポイントをセットすることができました。

RunメニューからDebugを選択してください。

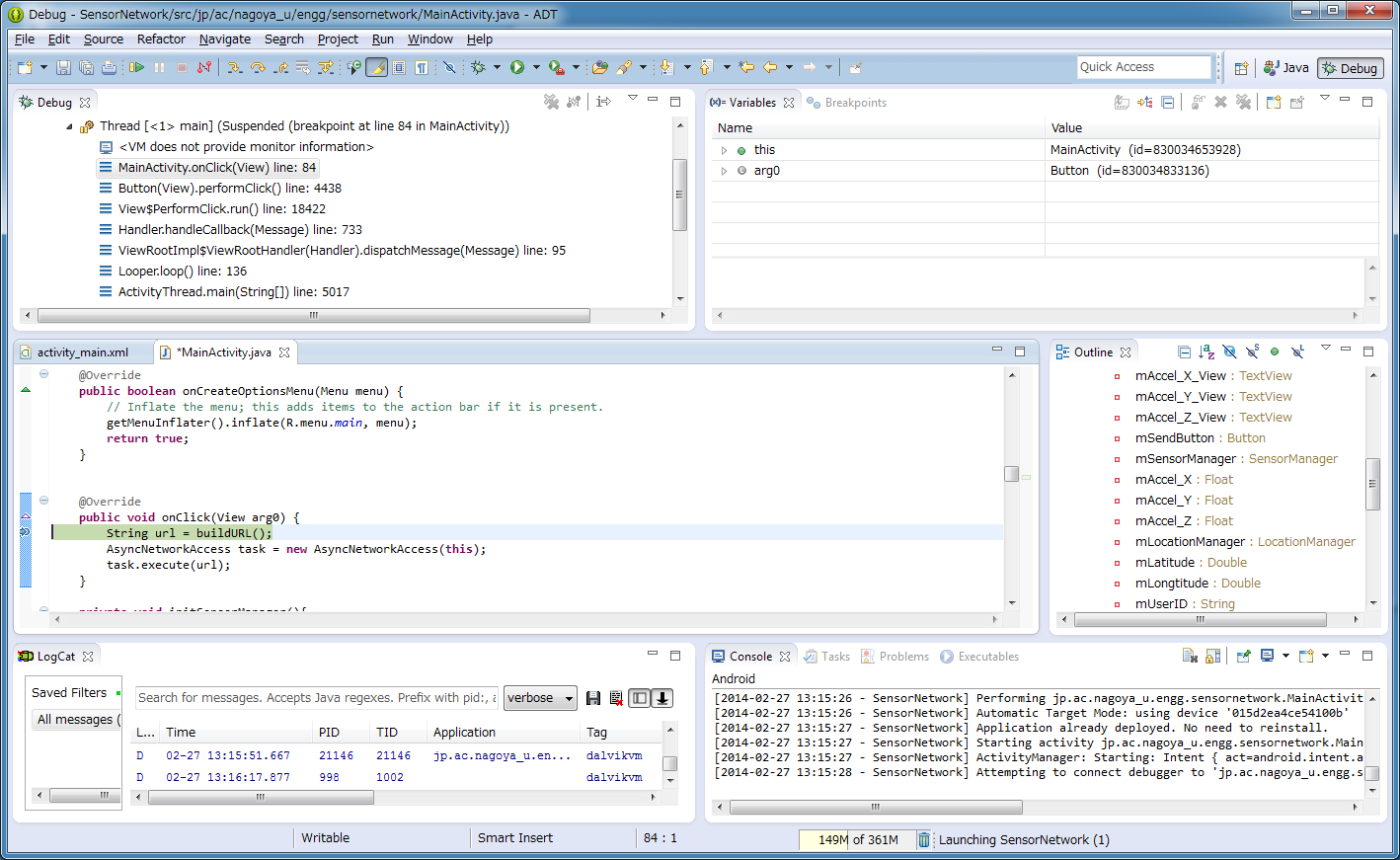
アプリケーションがデバッグモードで起動しました。

Sendボタンにタッチしてみてください。

画面レイアウトを変更してもよいか？というダイアログが表示されますのでYesをクリックしてみましょう。



画面レイアウトが切り替わって、Debugモードの画面になりました。



この画面では左上から右にプロセスのリスト、変数リスト、ソースコード、Outline,Logcat,Consoleとなっています。

これが標準的なデバッグ画面のレイアウトになります。

この画面でメニューバーの下のアイコンをクリックすることで、一行づつ実行したり、関数から出るところまでを実行したり、そのときの変数の値を確認したりすることができます。

まず右上のペインのthisの左の三角形をクリックしてみてください。

MainActivityクラスのメンバ変数の一覧が表示されるはずです。

それぞれの値がどうなっているかが確認できます。

ここでRun->Step　Intoを選んでみると、buildURL関数の中に処理がすすむのが確認できます。

次にStepReturnを選ぶとbuildURLの終了時点まで処理を進めることができます。

続けてStep Overを選択すると次の行に処理が進んでurlという変数に値が格納されるのが確認できます。

確認が完了して処理を進めるときはResumeを選択してください。

再び処理が始まります。

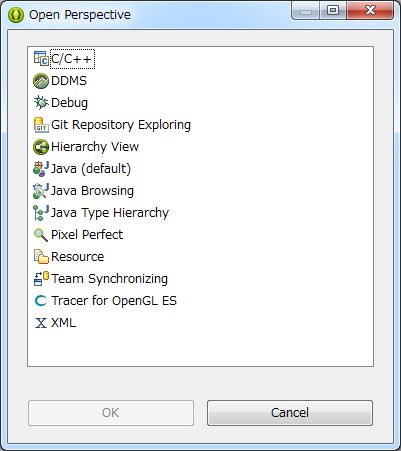
このように、ソースラインデバッガをつかって、任意の場所で処理をとめたり変数の内容を確認したりすることができます。

Debugが終わってもとの画面に戻るには右上の端にあるJavaというパースペクティブ選択のボタンをクリックしてください。

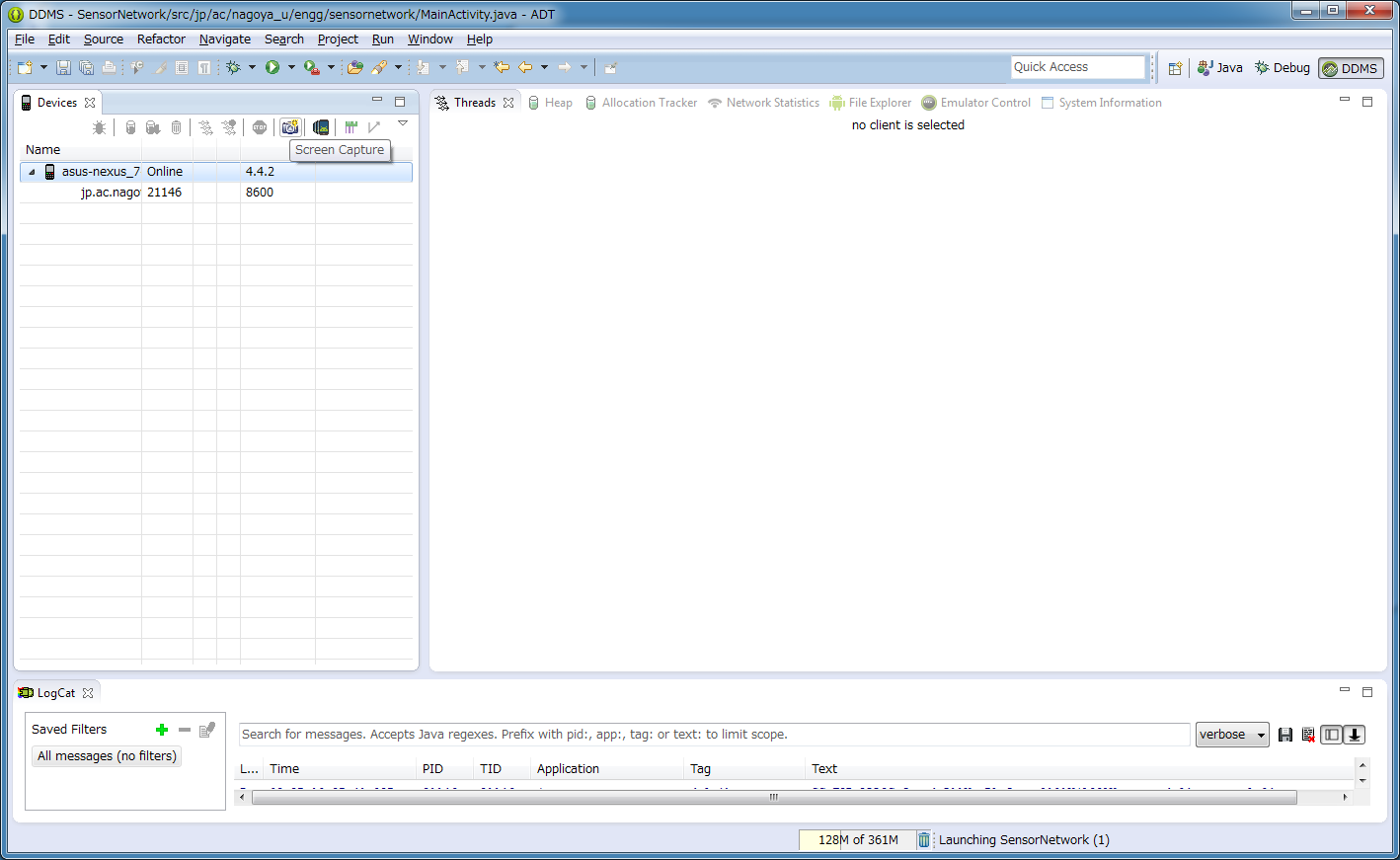
●DDMSの使い方

Androidの開発環境ではこれ以外にも、DDMS(Dalvik Debug Monitor Server)というツールが使用できます。DDMSではlogの確認や実行画面の撮影、メモリの使用状況の確認、ターゲットへのファイルの交換などが行えます。

DDMSの画面を起動するには画面右上のパースペクティブを追加するアイコンをクリックしDDMSを選んでください。



選択するとつぎのような画面が開きます。

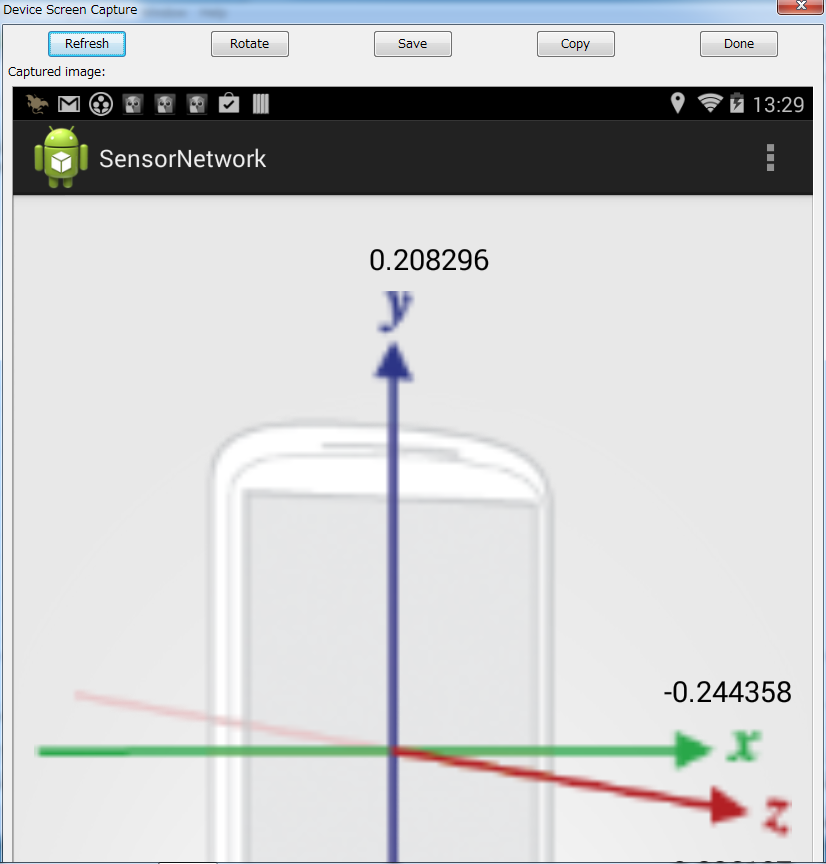


○ターゲット画面の撮影

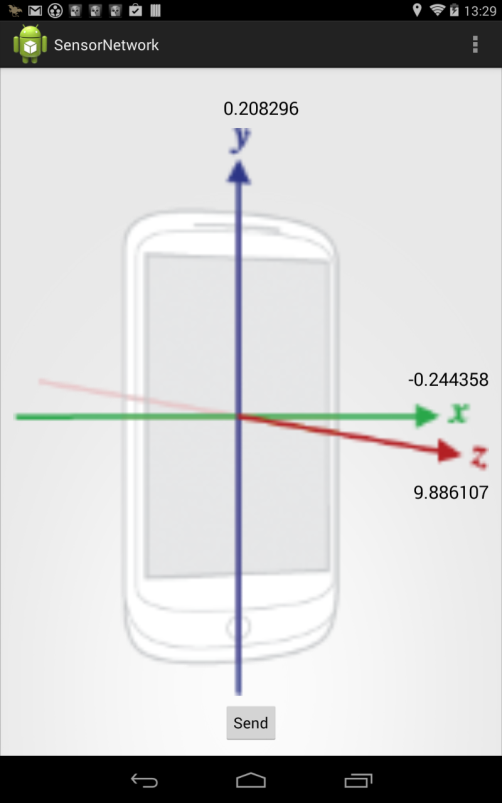
アプリケーションの実行画面を撮影するのは、アプリケーションの公開の際にもかならず必要となります。

画面を撮影するには画面左上あたりのScreen Captureという文字が表示されているカメラアイコンをクリックします。

次のようなウィンドウが表示されて、デバイスの画面が撮影されます。



Save や　Copyのボタンをクリックしてファイルを保存したり、クリップボードにコピーしたりすることができます。



このような画像を撮影することができます。

## 5.2　リソースの扱い（多言語化など）

サンプルアプリケーションの説明の中で、0.000という文字列はそのまま入力したのに、ボタンの文字列であるSendはわざわざリソースを使用しました。

これはリソースを利用することで、多言語対応などを行うことができるためです。

　たとえばSendというボタンを「送信」という名前に変えたいとします。

　しかし、GooglePlayなどで公開するのであれば、日本以外のユーザにも使ってもらいたいので、できればユーザの言語選択によって、表示が変わるようにできれば便利です。

　言語毎にリソースを持っておくことでそういったことができるようになります。

　具体的には、文字列リソースはres/values/strings.xmlというファイルになるのですがこれはデフォルト言語のフォルダーとなります。

　日本語対応のファイルを用意するときは、res/values-jaというディレクトリを用意します。

　そしてこの中にstrings.xmlをコピーし、文字列を日本語に変更しておきます。

　アプリケーションはリソースを探す際に端末の状態にあったリソースディレクトリを探して、そこにファイルが見つからなければデフォルトのディレクトリを探しに行きます。

　したがって、別の設定をする必要が無ければ、そのファイルは置いておく必要はありません。

　そのほかにも、画面サイズなどいろいろな条件でリソースファイルを自動的に選択してくれる機能があります。

　リソースを選択するためのディレクトリの作成方法の詳細については以下のURLに説明されています。

http://developer.android.com/intl/ja/guide/topics/resources/providing-resources.html

この中で一般的によく使われるものについてのみ簡単に説明します。

まず、リソースディレクトリres/以下には以下のディレクトリがあります。

|  |  |
| --- | --- |
| ディレクトリ名 | リソースの種類 |
| animator/ | プロパティアニメーション　プロパティを適当な間隔で変化させることによって行うアニメーションの定義ファイル(XML) |
| anim/ | トゥィーンアニメーション　描画命令を使ったアニメーションの定義ファイル（XML)プロパティアニメーションを入れることもできる。 |
| color/ | 色定義ファイル(XML) |
| drawable/ | イメージファイル(png,9.png,jpg,gif) |
| layout/ | レイアウトファイル(XML) |
| menu/ | メニュー定義ファイル(XML) |
| raw/ | raw(生）のデータ、ムービーファイルやサウンドファイルなどの直接アクセスするファイル |
| values/ | 文字列や配列などいろいろな値を定義したファイル |
| xml/ | その他のXMLファイル |

　このディレクトリ名に様々な属性情報をつけくわえることで特定に属性の端末に応じたリソースを区別することができます。

　実は詳しく説明せず使ってきましたが多言語化の対応のところで言語を区別するためにvalues-jaなどのディレクトリを使ったのもこの方法の一つでした。他にも自動的に作成されているdrawable-hdpiの様に、画面の解像度を選択することもできます。

　これらの追加する属性には、以下の表のような種類があります。

|  |  |  |
| --- | --- | --- |
| 意味 | 代表的な値 | 簡単な説明 |
| キャリアなどの情報(MCC/MNC) | mcc440-mcc10(Docomo), mcc440-mnc54(KDDI), mcc440-mnc20(SoftBank), mcc440-mnc00(EMOBILE) | 端末のキャリアによってリソースを変えたい場合などmcc440だけだと、日本のキャリアの場合の値になる |
| 言語、国（エリア）など | ja(日本語),　en(英語),　fr(フランス語）,　en-rUS(アメリカ英語）,　fr-rCA(カナダのフランス語) | 言語や国（エリア）によって区別したい場合 |
| 画面の最小幅(API 13) | sw320dp | 画面の短い方の辺の長さ（縦横関係なく短い方）、たとえばレイアウト上の理由で、短辺が600dp以上必要な場合などはsw600dpとする。 |
| 画面の横幅(API 13) | w720dp,w1024dp | 画面の横幅で選択する場合（横幅なので向きによって変わる）、たとえば、w720dpとw1024dpという二つのディレクトリを作ってリソースを入れておいた場合、画面の幅がどちらか近い方のリソースが選択される。 |
| 画面の高さ（API 13) | h720dp,h1024dp | 画面の高さで選択する場合（高さなので向きによって変わる）幅と同様、近い方の値が使用される |
| スクリーンサイズ(API 4) | small,normal,large,xlarge | small　320x426dp程度 normal　320x470dp程度 large　480x640dp程度 xlarge　720x960程度 |
| スクリーンの縦横比(API 4) | long,notlong | long　ワイド画面(WQVGA,WVGA,FWVGA) notlong　QVGA,HVGA,VGA |
| スクリーンの向き | port,land | port 縦長 land 縦長 |
| ドックモード(API 8) | car/desk | ドックにセットされている場合に、ドックの種類によって選択する場合 |
| 夜間モード(API 13) | night,notnight | 昼と夜で表示をかえたい場合（時刻によって切り替わる） |
| スクリーン解像度 | ldpi,mdpi,hdpi,xhdpi,nodpi,tvdpi | ldpi　120dpi程度 mdpi　160dpi程度 hdpi　240dpi程度 xhdpi　320dpi程度 nodpi 解像度に関係なく使用する場合 tvdpi 213dpi程度(API 13) |
| タッチスクリーンのタイプ | notouch,stylus,finger | notouch タッチスクリーンなし stylus スタイラスペン finger 指で操作する場合 |
| キーボードの状態 | keysexposed,keyshidden,keyssoft | keysexposed ハードウェアキーボードが出ている時 keyshidden ハードウェアキーボードが隠れている時 keyssoft ソフトウェアキーボードが有効な時（表示されていない場合も含む） |
| テキスト入力の最優先デバイス | nokeys,qerty,12key | nokeys: ハードウェアキーボードを持っていない qwerty: フルキーボードが使用できるとき（出ていなくてもよい） 12key: 携帯電話キー（数字キーと#＊）が有効な時（出ていなくてもよい） |
| ナビゲーションキー | navexposed,navhidden | navexposed:ナビゲーションキー（方向キーなど）が有効な時 navhidden:ナビゲーションキーが隠れているとき |
| タッチスクリーン以外の操作 | nonav,dpad,trackball,wheel | nonav:画面タッチ以外のナビゲーションキーがない dpad:方向キー trackball:トラックボール wheel:ホイール |
| バージョン | v4,v7,v11など | APIレベルの値  v4:Android1.6 v7:Android 2.1 v11:Android 3.0 |

　これらの値を必要に応じて組み合わせてディレクトリを作成します。

　たとえば、「英語(en)」で「大画面(xlarge)」で「横向き(land)」の時に限定のレイアウトなら

　「layout-en-xlarge-land」というディレクトリを作成して保存します。

　それ以外の条件（一つでも条件が違ったとき）たとえば、日本語だった場合はそれに適合する最も条件の多いものが合致します。たとえば「layout-ja」と「layout」があれば、この場合は「layout-ja」が優先されます。どれにもあうものがなければ、「layout」ディレクトリのファイルが最終的に使用されることになります。つまり、少なくとも何も属性の着いていないディレクトリを用意しておく必要があります。

　最も注意すべきは属性名を並べる順番です。これは上の表の上から順番に並べる必要があります。先ほどの例では「layout-xlarge-en-land」などという順番で書いてしまうと適合しないことになります。

　また、ディレクトリは「res」ディレクトリに作成しなければなりません。「res/drawable/drawable-en/」というような階層ディレクトリは使用できません。　詳細は先のURLを確認してください。

# 参考文献

[1] 改訂2版 基礎から学ぶAndroidアプリ開発

ISBN-13: 978-4863541139

[2] Android Developer Google

http://developer.android.com/

[3] Android ADKで学ぶハードウェア制御

ISBN-13: 978-4863541078

# 変更履歴

|  |  |  |
| --- | --- | --- |
| **版数** | **変更内容** | **年月日** |
| 1.0 | 新規作成 | 2014/Feb/27 |