

実践 ANDROIDアプリ開発

@JX通信社

酒本伸也

2015/8/6

GIT

よく使うコマンド

- `git status`
- `git diff` ファイル
- `git checkout` ファイル
- `git add` ファイル
- `git commit -m “メッセージ”`
- `git pull “リモートリポジトリのURL” “ローカルのブランチ名”`
- `git push “リモートリポジトリのURL” “ローカルのブランチ名”`

- git status

- 現在のブランチの最新の状態から
変更があるかどうかを確認

- git diff ファイル

- 対象のファイルが最新の状態から
どこに変更があったのかを確認

- git checkout ファイル

- 対象のファイルを現在のブランチの最新の状態に戻す

- git add ファイル

- 対象のファイルをcommit対象にする

- `git commit -m “メッセージ”`
 - `commit`対象にしたファイルたちを
`push`対象にする
- `git pull “リモートリポジトリのURL” “ローカルの
ブランチ名”`
 - リモートリポジトリの変更をローカルに反
映させる

- git push “リモートリポジトリのURL” “ローカルのブランチ名”
 - push対象にしたファイルたちをリモートリポジトリへpushする

~/.GITCONFIG

- よく使うものは
登録しておく と便利

[user]

name = Shinya Sakemoto

email = sakebook@gmail.com

[color]

ui = auto

[alias]

st = status

br = branch -a

ch = checkout

cm = commit

stt = status -uno

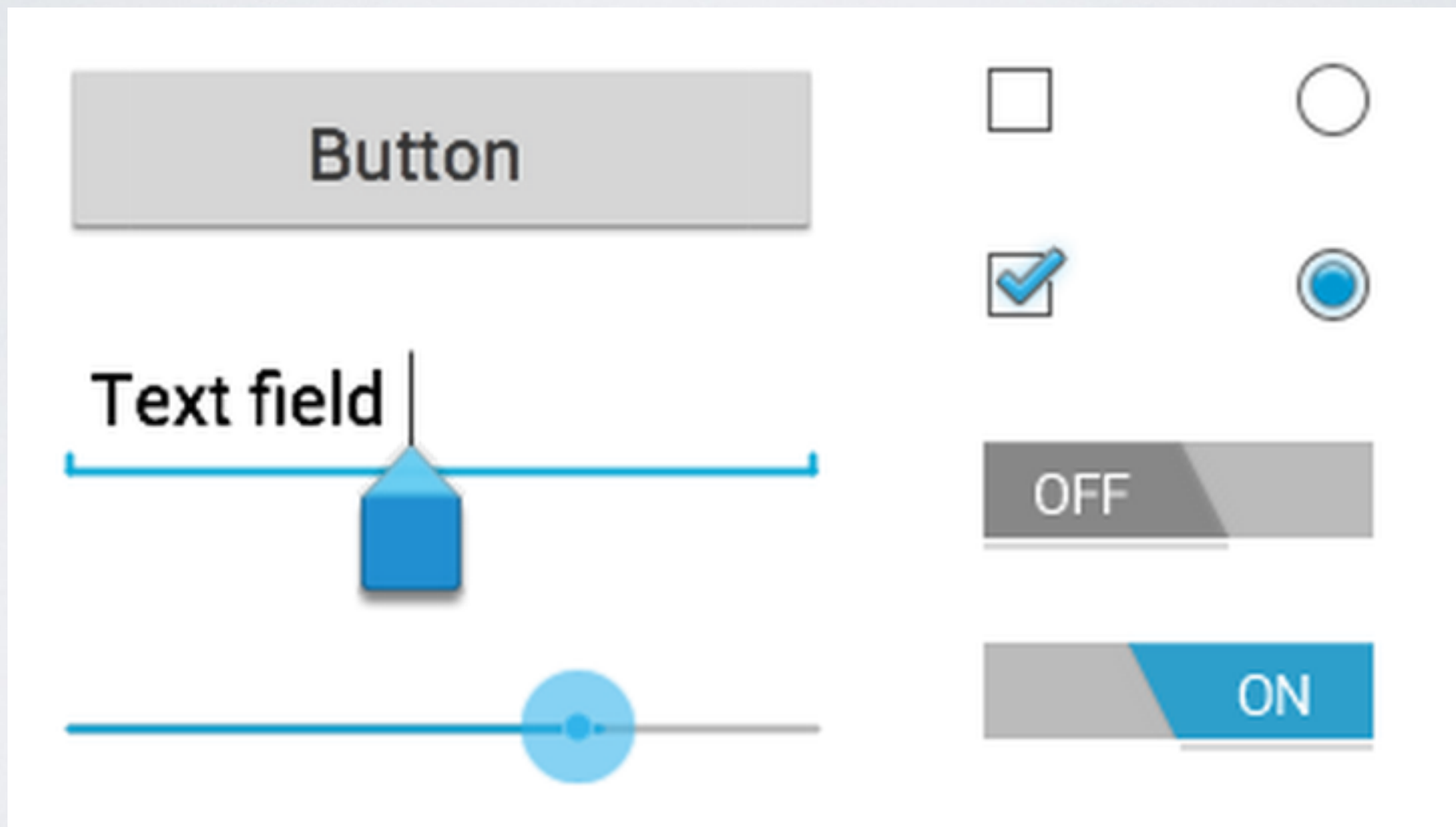
difff = diff --word-diff

詳しくは

- <http://blog.nanapi.co.jp/tech/2014/04/23/git-love/>

WIDGET

WIDGET



作りたいアプリレビュー

タスクに分解

- 画面ごと
 - 記事一覧、記事詳細
- 機能ごと
 - 通信周り、シェア、汎用処理

分けることで

- 複数人での作業分担
- 大まかな工数の概算ができる

工数

- 1日 = 8時間
- 1週間 = 5日
- 1ヶ月 = 4週間

工数

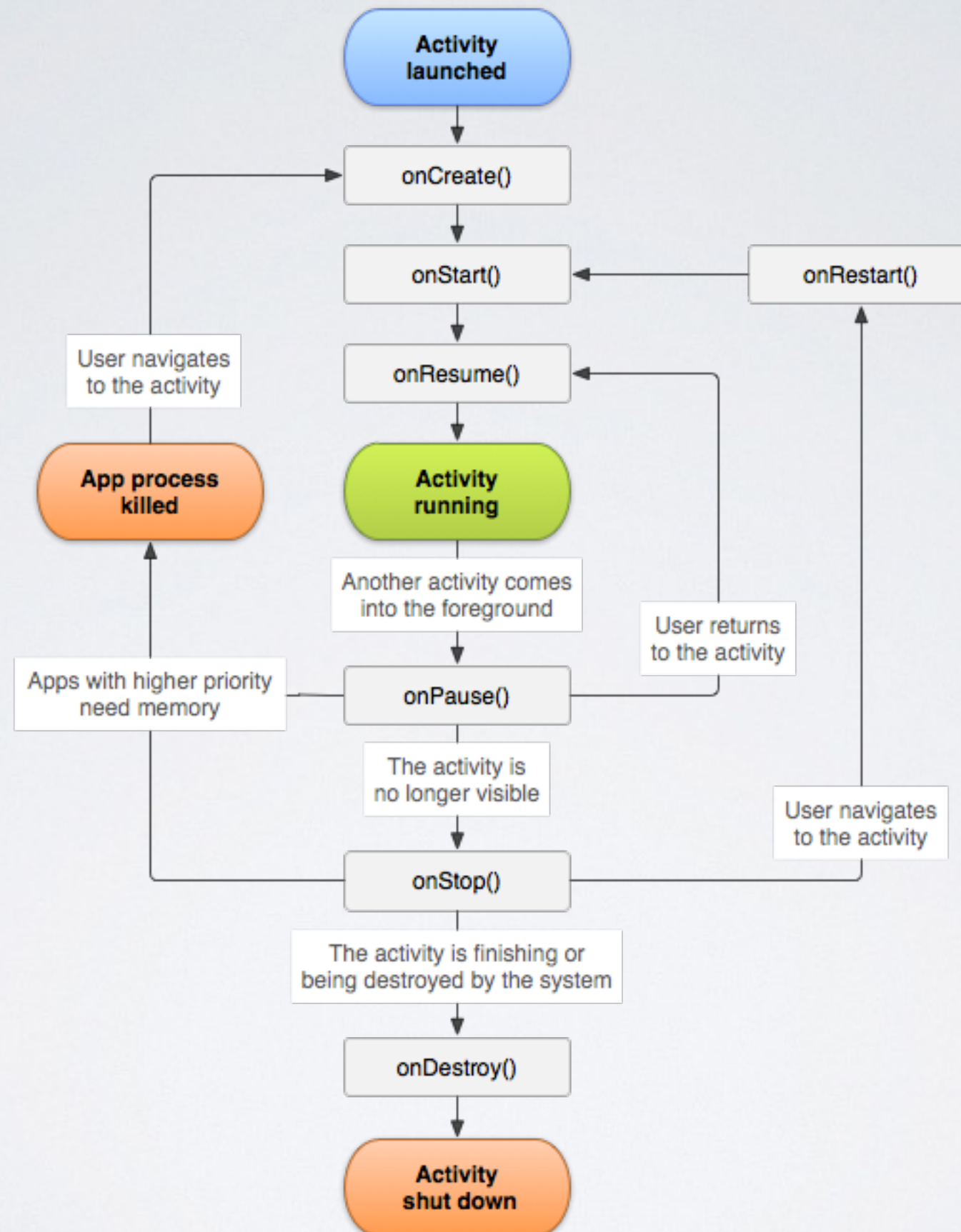
- 1日 = 8時間
- 1週間 = 5日
- 1ヶ月 = 4週間

今は忘れて

ライフサイクル

ライフサイクル

- ActivityはUIを持つ画面には必須
- Activity同士の円滑なやり取りを助長する
- ライフサイクル ≡ 状態遷移

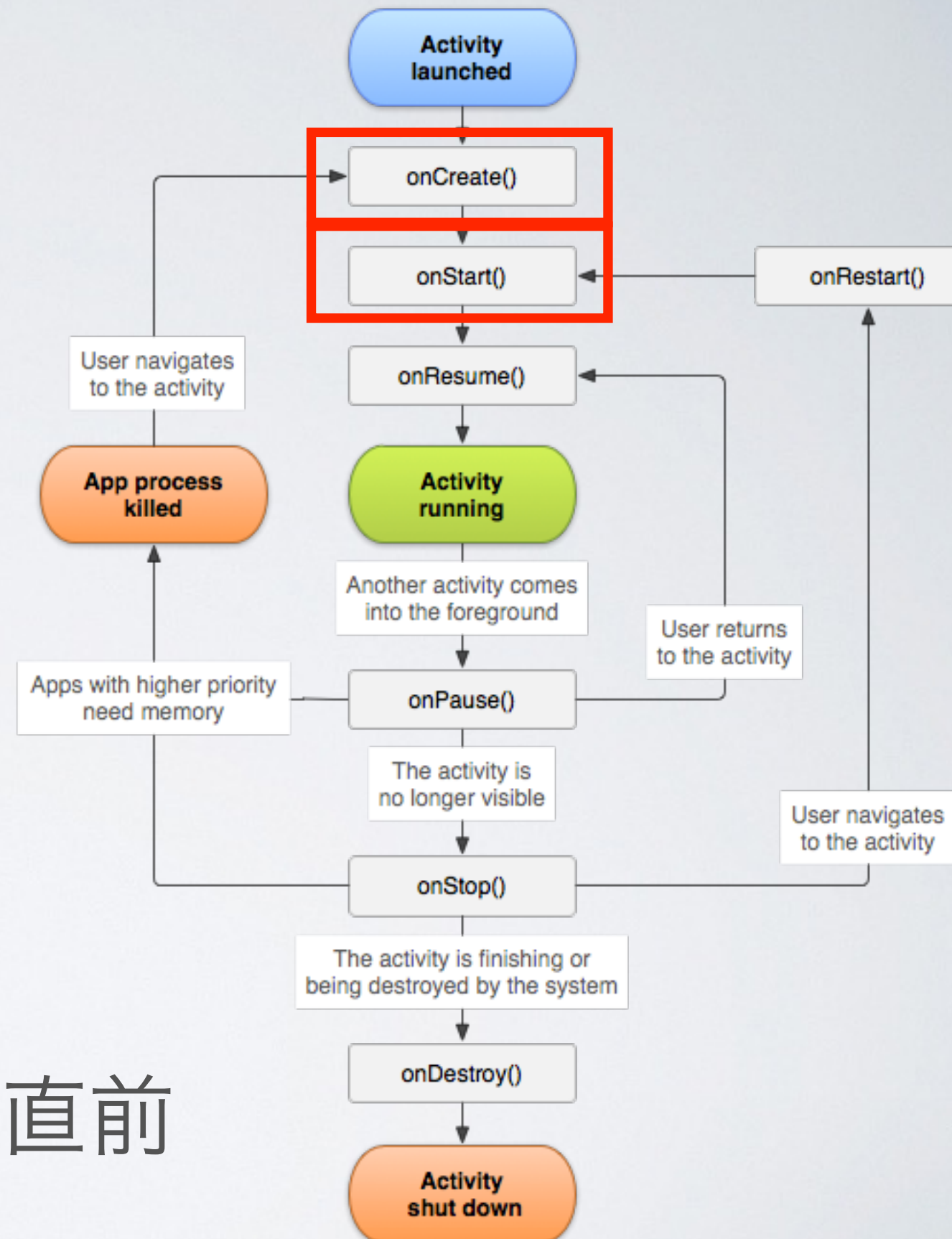


- onCreate

- Activityが初めて作られたとき

- onStart

- Activityがユーザに見えるようになる直前

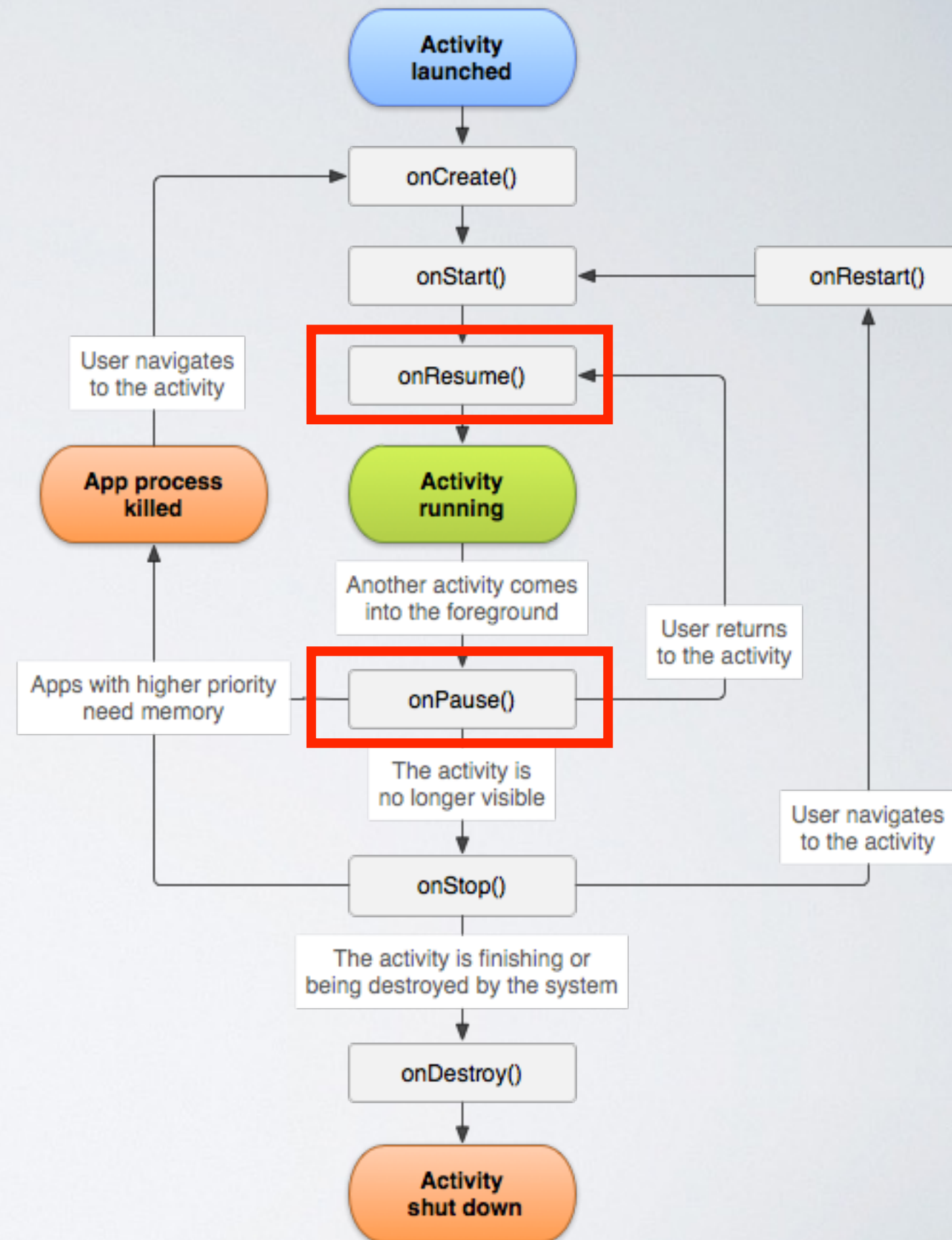


- onResume

- Activityが
表示されたとき

- onPause

- 別のActivityが
表示されるとき



- onStop

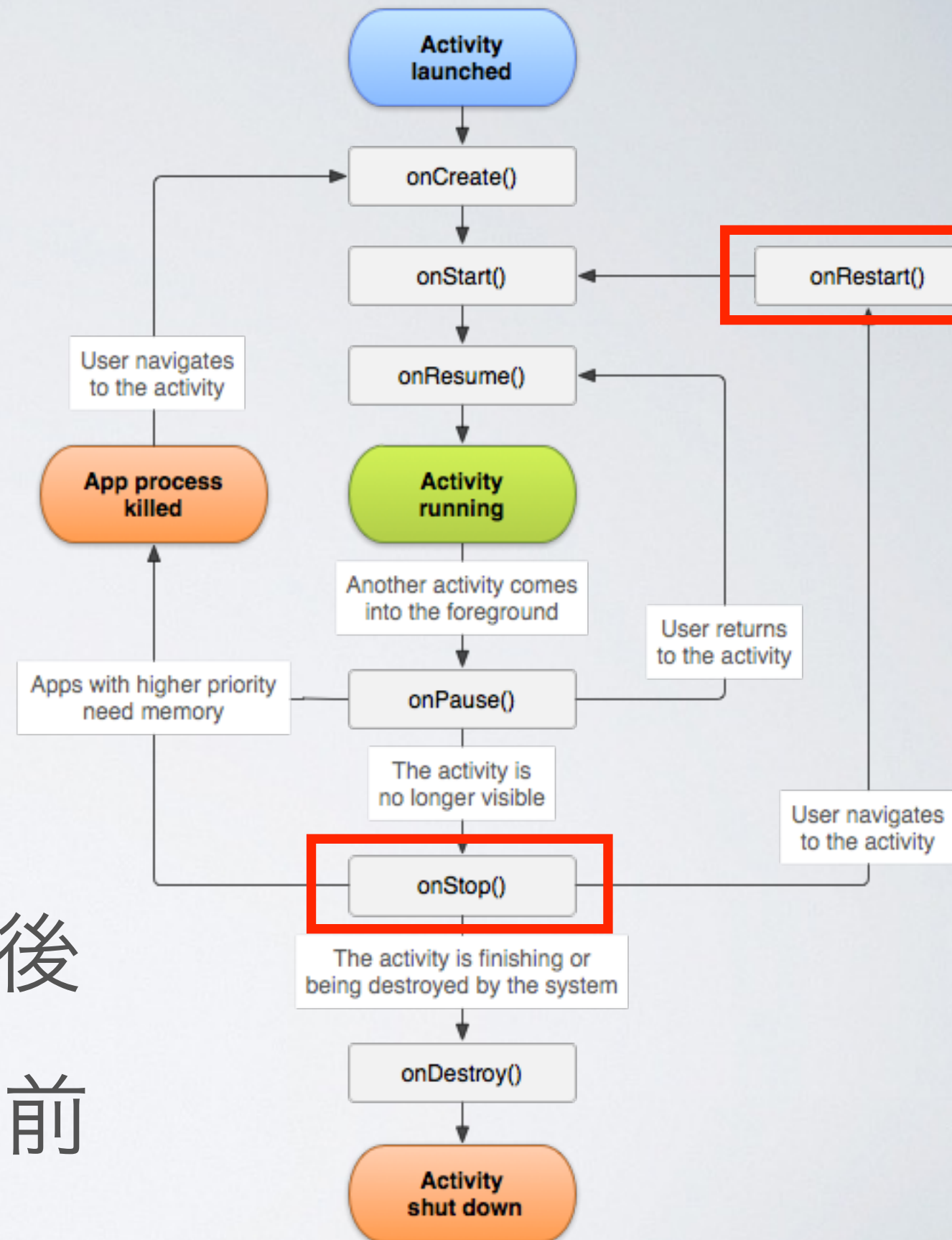
- Activityが

見えなくなった時

- onStart

- Activityが停止した後

再度開始される直前



詳細

- <http://dev.classmethod.jp/smartphone/android/android-tips-21-activity-lifecycle/>

イベントハンドリング

イベントハンドリング

- イベント
 - ユーザの何かしらの行動や
内部の処理の特定のタイミング
- イベントを適切なタイミングでキャッチし
適切な処理を行う。

ボタンをクリック

- ButtonのWidgetを設置
- Buttonがクリックされた時に行う処理を記述
- Buttonのクリックイベントをハンドリング

BUTTONのWIDGETを設置

- xml

```
activity_main.xml x
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:id="@+id/layout_main"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingLeft="16dp"
7      android:paddingRight="16dp"
8      android:paddingTop="16dp"
9      android:paddingBottom="16dp"
10     tools:context=".MainActivity">
11
12     <Button
13         android:id="@+id/button"
14         android:layout_centerInParent="true"
15         android:text="@string/hello_world"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"/>
18
19 </RelativeLayout>
20
```


BUTTONがクリックされた時 に行う処理を記述

- Callbackで

`OnClickListener#onClick`

が呼ばれるので、その実装を書く

```
10
11 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
12
13     public final static String TAG = "Trunk";
14
15     private MyCustomClick mMyCustomClick = new MyCustomClick();
16     private View.OnClickListener mOnClickListener = new View.OnClickListener() {
17         @Override
18         public void onClick(View v) {
19             Log.d(TAG, "mOnClickListener: onClick");
20         }
21     };
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27
28         Button button = (Button)findViewById(R.id.button);
29
30         button.setOnClickListener(mMyCustomClick);
31         button.setOnClickListener(new MyCustomClick());
32         button.setOnClickListener(mOnClickListener);
33
34         button.setOnClickListener(new View.OnClickListener() {
35             @Override
36             public void onClick(View v) {
37                 Log.d(TAG, "View.OnClickListener: onClick");
38             }
39         });
40
41         button.setOnClickListener(this);
42
43     }
44
```

実装

ハンドリング

ハンドリング

+
実装

```
10
11 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
12
13     public final static String TAG = "Trunk";
14
15     private MyCustomClick mMyCustomClick = new MyCustomClick();
16     private View.OnClickListener mOnClickListener = new View.OnClickListener() {
17         @Override
18         public void onClick(View v) {
19             Log.d(TAG, "mOnClickListener: onClick");
20         }
21     };
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27
28         Button button = (Button)findViewById(R.id.button);
29
30         button.setOnClickListener(mMyCustomClick);
31         button.setOnClickListener(new MyCustomClick());
32         button.setOnClickListener(mOnClickListener);
33
34         button.setOnClickListener(new View.OnClickListener() {
35             @Override
36             public void onClick(View v) {
37                 Log.d(TAG, "View.OnClickListener: onClick");
38             }
39         });
40
41         button.setOnClickListener(this);
42
43     }
44
```

実装?

ハンドリング

別クラスで実装

- MyCustomClick.class

```
public class MyCustomClick implements View.OnClickListener{  
    @Override  
    public void onClick(View v) {  
        Log.d(MainActivity.TAG, "MyCustomClick onClick");  
    }  
}
```



```
10
11 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
12
13     public final static String TAG = "Trunk";
14
15     private MyCustomClick mMyCustomClick = new MyCustomClick();
16     private View.OnClickListener mOnClickListener = new View.OnClickListener() {
17         @Override
18         public void onClick(View v) {
19             Log.d(TAG, "mOnClickListener: onClick");
20         }
21     };
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27
28         Button button = (Button)findViewById(R.id.button);
29
30         button.setOnClickListener(mMyCustomClick);
31         button.setOnClickListener(new MyCustomClick());
32         button.setOnClickListener(mOnClickListener);
33
34         button.setOnClickListener(new View.OnClickListener() {
35             @Override
36             public void onClick(View v) {
37                 Log.d(TAG, "View.OnClickListener: onClick");
38             }
39         });
40
41         button.setOnClickListener(this);
42     }
43
44
```

実装?

ハンドリング

同じクラスで実装

- 実装が記述されているのでthisが使える。

```
66  
67  
68       
69     @Override  
70     public void onClick(View v) {  
71         Log.d(TAG, "MainActivity: onClick");  
72     }  
73 }
```

SET_ON_CLICK以外のリスナー

- setOn○○○で用意されている。
- 登録/解除が必要なものは
add/remove
register/unregister
など、命名規則で判別できる

WIDGETの状態

BUTTONがクリックされた時

- なんどもクリックできる
- クリックできたかどうかわからない
- クリック可能な状態か不明

なんどもクリックできる

- メールを送るなど、複数回されては困るものがある
 - View#setEnabled(boolean)
 - Viewの状態を、有効/無効にする

クリックできたかどうか
わからない

クリック可能な状態か不明

- ユーザにフィードバックを与える

```
<Button
    android:id="@+id/button"
    android:layout_centerInParent="true"
    android:text="@string/hello_world"
    android:background="@drawable/button_state"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3      <item
4          android:state_enabled="false"
5          <shape
6              android:shape="rectangle"
7              <solid android:color="#00ff00" />
8          </shape>
9      </item>
10     <item
11         android:state_pressed="true"
12         android:state_enabled="true"
13         <shape
14             android:shape="rectangle"
15             <solid android:color="#ff0000" />
16         </shape>
17     </item>
18     <item
19         android:state_pressed="false"
20         android:state_enabled="true"
21         <shape
22             android:shape="rectangle"
23             <solid android:color="#0000ff" />
24         </shape>
25     </item>
26 </selector>
```

無効な状態

有効で
押されている状態

有効で
押されていない状態

ライブラリ

面倒なものは誰かが楽してる

- 公開されているものは積極的に使おう！
 - ソースコードを読むと勉強になる
 - かけるはずだった時間を別のところに
充てられる

GRADLE

- リモートリポジトリから
ライブラリをインポートし、
かつ依存関係の解決をしてくれる

探す

- <https://android-arsenal.com/>
- <https://github.com/trending?l=java>
- <https://gradleplease.appspot.com/>

使う

- app/build.gradleに使用したいライブラリを記載してGradleを同期するだけ

使う

Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

[Sync Now](#)

```
5      buildToolsVersion '22.0.1'
6
7      defaultConfig {
8          applicationId "com.sakebook.android.trunknews"
9          minSdkVersion 16
10         targetSdkVersion 22
11         versionCode 1
12         versionName "1.0"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18         }
19     }
20 }
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     compile 'com.android.support:appcompat-v7:22.2.0'
25     compile 'com.squareup.retrofit:retrofit:1.9.0'
26 }
27
```

依存関係

- Retrofitしかapp/build.gradle

に書いていないが

よしなになってる



通信

必須事項

- パーミッションの追加
- UIスレッド以外を用いる

パーミッションの追加

- アプリインストール時に聞かれるアレ
- AndroidManifest.xmlに一括して記述
- Android Mからは使う時にユーザが
任意に許可できるように

- AndroidManifest.xml
- <application>の前に記述

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.sakebook.android.trunknews" >
4
5      <uses-permission android:name="android.permission.INTERNET" />
6
7      <application
8          android:allowBackup="true"
9          android:icon="@mipmap/ic_launcher"
10         android:label="TrunkNews"
```

UIスレッド

- Androidはシングルスレッド
 - UIスレッド(メインスレッド)
 - 画面の描画を行う
 - UIスレッドが詰まると画面が固まる

通信はレスポンスを待つ

- レスポンスが帰ってくるまで描画が止まるので別スレッドを立てて実行させる
- 通信に限らず、その他の重い処理は別スレッドで行う

RETROFIT

- Square社製のライブラリ
 - RESTクライアント
 - 非同期
 - インタフェースを作成するのが特徴

インタフェースを作成

```
1  package com.sakebook.android.trunknews.network;
2
3  import com.sakebook.android.trunknews.models.Article;
4
5  import java.util.List;
6
7  import retrofit.Callback;
8  import retrofit.http.GET;
9  import retrofit.http.Path;
10
11  /**
12   * Created by sakemotoshinya on 15/08/06.
13   */
14  public interface ApiService {
15
16      @GET("/api/v1/tags/{tagName}/items")
17      void getArticles(@Path("tagName") String tagName,
18                      Callback<List<Article>> response);
19
20  }
21
```

クライアントを作成

```
ApiClient.java x
9
10 /**
11  * Created by sakemotoshinya on 15/08/06.
12  */
13 public class ApiClient {
14
15     private final static String API_HOST = "https://qiita.com";
16
17     public static void getArticles(Callback<List<Article>> callback) {
18         RestAdapter restAdapter = new RestAdapter.Builder()
19             .setEndpoint(API_HOST)
20             .build();
21         ApiService service = restAdapter.create(ApiService.class);
22         service.getArticles("Android", callback);
23     }
24 }
25
```


モデルを作成

- APIのレスポンスに準ずる

- Retrofit内のGsonで
よしなに変換

```
6      public class Article {  
7  
8          private String body;  
9          private String title;  
10         private String url;  
11  
12         public String getBody() { return body; }  
15  
16         public String getTitle() { return title; }  
19  
20         public String getUrl() { return url; }  
23  
24         @Override  
25         public String toString() {  
26             return "Article{" +  
27                 "body='" + body + '\'' +  
28                 ", title='" + title + '\'' +  
29                 ", url='" + url + '\'' +  
30                 '}';  
31         }  
32     }  
33
```

モデルを作成

- ネストしたものは別のモデルを作成して対応させる
- Keyを自分で設定したい場合は
`@SerializedName("〇〇")`
とアノテーションをつける

実行

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(final View v) {
        Log.d(TAG, "View.OnClickListener: onClick");
        ApiClient.getArticles(new Callback<List<Article>>() {
            @Override
            public void success(List<Article> articles, Response response) {
                Log.d(TAG, "success");
                Log.d(TAG, "success: dump= " + articles.get(0).toString());
                v.setEnabled(true);
            }

            @Override
            public void failure(RetrofitError error) {
                Log.d(TAG, "failure");
                v.setEnabled(true);
            }
        });
        v.setEnabled(false);
    }
});
```

実行

```
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(final View v) {  
        Log.d(TAG, "View.OnClickListener: onClick");  
        ApiClient.getArticles(new Callback<List<Article>>() {  
            @Override  
            public void success(List<Article> articles, Response response) {  
                Log.d(TAG, "success");  
                Log.d(TAG, "success: dump= " + articles.get(0).toString());  
                v.setEnabled(true);  
            }  
  
            @Override  
            public void failure(RetrofitError error) {  
                Log.d(TAG, "failure");  
                v.setEnabled(true);  
            }  
        });  
        v.setEnabled(false);  
    }  
});
```

コールバックの
実装

RETROFIT

- 取得を非同期で行ってくれるが
通信周りのライブラリではない
 - 内部ではデフォルトの通信を利用
 - いろいろカスタムできる