ここが変わる！Unity 5のスマホ開発

～アセットバンドル、ビルド、プラグイン

ユニティ・テクノロジーズ・ジャパン
エバンジェリスト
伊藤周

AssetBundle

今までのAssetBundle問題点

- ビルドスクリプトが複雑
- ビルドに時間がかかる
- アセットの依存関係が面倒
- 互換性

解決方針

- シンプルな感じに
  - Simple UI
  - Simple script
- 追加ビルド可能に
- 依存性はUnityが解決
  - Manifest file
- バージョン間の互換性
- 新旧混ぜるな危険

変更点

- 新しいAssetBundleビルド方式
- 新しいAssetロード方式

unity

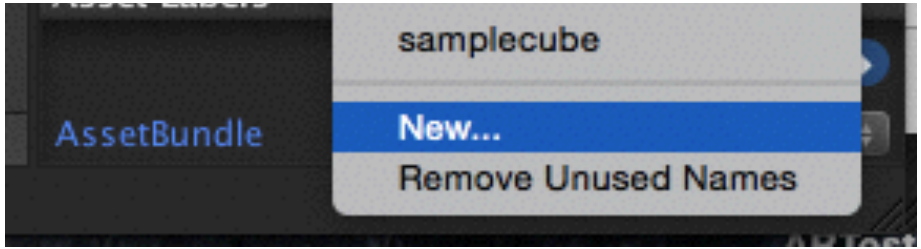
新しいAssetBundleビルド

unity

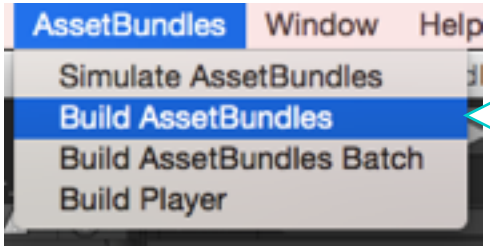
まずは
ざっくり

新しいAssetBundleビルド

Step1　Asset と AssetBundle 間を結びつける



Step2　ビルドするスクリプトを呼ぶ




```
public static void BuildAssetBundles(){
    BuildPipeline.BuildAssetBundles();
}
```

以上！

unity

ちょっと
深く掘り下げ

AssetBundle名前付け

- 小文字のみ
- 同名禁止
- パス構成
  - Editor拡張で一気に**「名前」＝「パス」**と変更も可能

```
public class CsutomImporter : AssetPostprocessor {
        void OnPostprocessTexture(Texture2D texture) {
                assetImporter.assetBundleName = assetPath;
        }
}
```


unity

Buildスクリプト

- 出力パス
- BuildAssetBundleOptions
- 出力ターゲット

```
public static void BuildAssetBundles()
{
    BuildPipeline.BuildAssetBundles(
        outputPath,
        BuildAssetBundleOptions.IgnoreTypeTreeChanges,
        EditorUserBuildSettings.activeBuildTarget);
}
```

BuildAssetBundleOptions

- **IngoreTypeTreeChanges(New!)**
  タイプツリーが変わっても無視

  ＝ソースコード変更しても**変化したとみなされない**

- **DisableWriteTypeTree**
  タイプツリー自体を削除（サイズ削減用途として）

  （こちらをオンにしてIngoreTypeTreeChangesをオンはできない）

- CollectDependencies　→いつでもON
- DeterministicAssetBundle　→いつでもON
- CompleteAssets　→廃止。いつでもコンプリート
- ForceRebuildAssetBundle(New!)　→強制的にリビルド
- AppendHashToAssetBundleName(New!)　→ハッシュを追加

unity

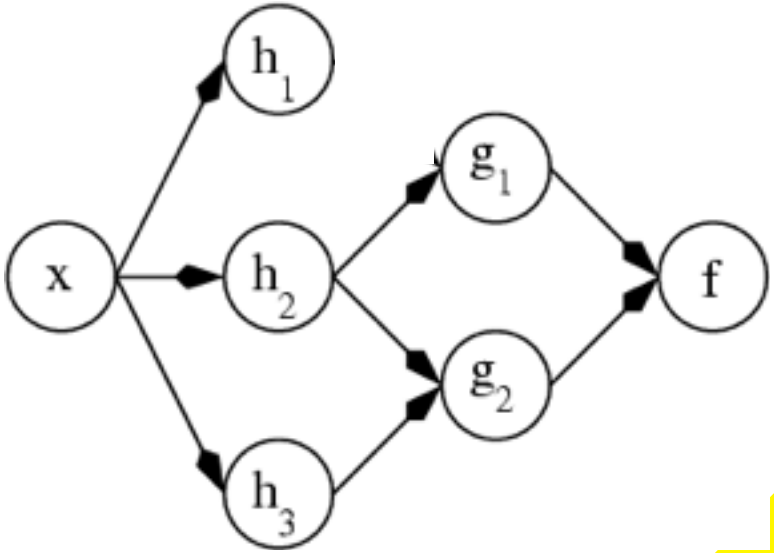
バージョン互換性について

バージョン互換性について

- タイプツリーを持っているのであればバージョンアップしても古いAssetBundleは使用可
- 要Unity5

依存関係

- Push/Pop いらない
- 勝手に依存関係を調べてくれる
- 「依存関係があるアセット群を全てビルド」が必要なくなる



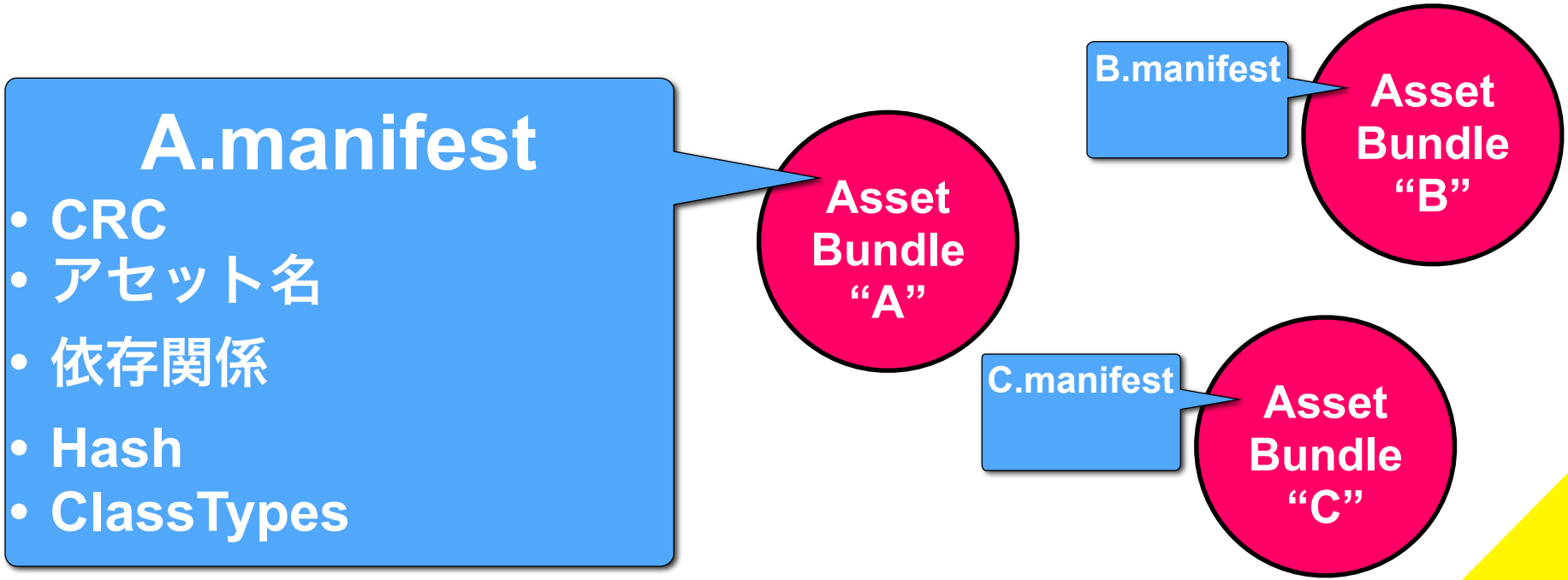
unity

Manifestファイルについて

- Single Manifestファイル　→親玉。**依存関係を解決**できる。
  (ビルド時のフォルダ名が　**ランタイムに読んで**使用
  AB名になる)
- 他の.manifestファイル　→**インクリメンタルビルドにだけ**
  関係する。
  ランタイム時は読み込まない

unity

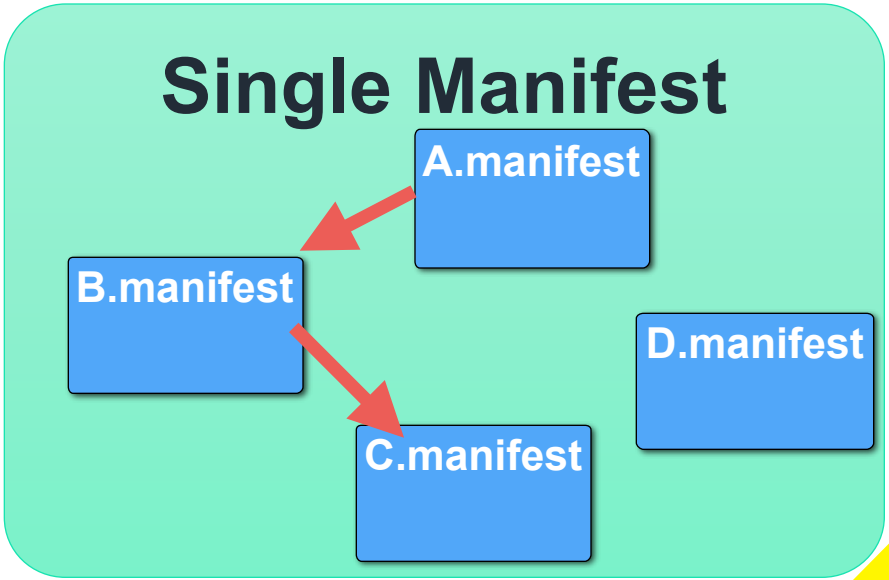
.manifest ファイル



unity

Single Manifestファイル

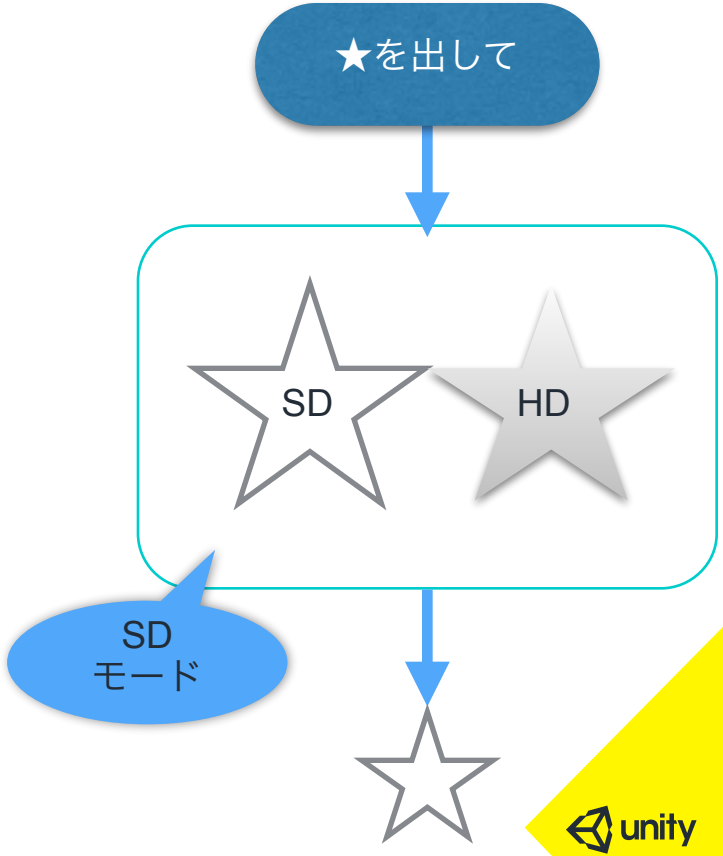
- 全AssetBundle名
- Assetの依存関係



unity

Variantとは

- HDとSDを同じアセットとして扱える仕組み
  - 例: ロースペックなスマホではSD、ハイスペックならHDで表示
- エディタ右下で設定
- 設定しているアセットと設定していないアセットを**混ぜてはいけない**
- 違うABで同じVariantも設定可



新しいAsset口ード方式

まずは
ざっくり

サンプルプロジェクトを使おう

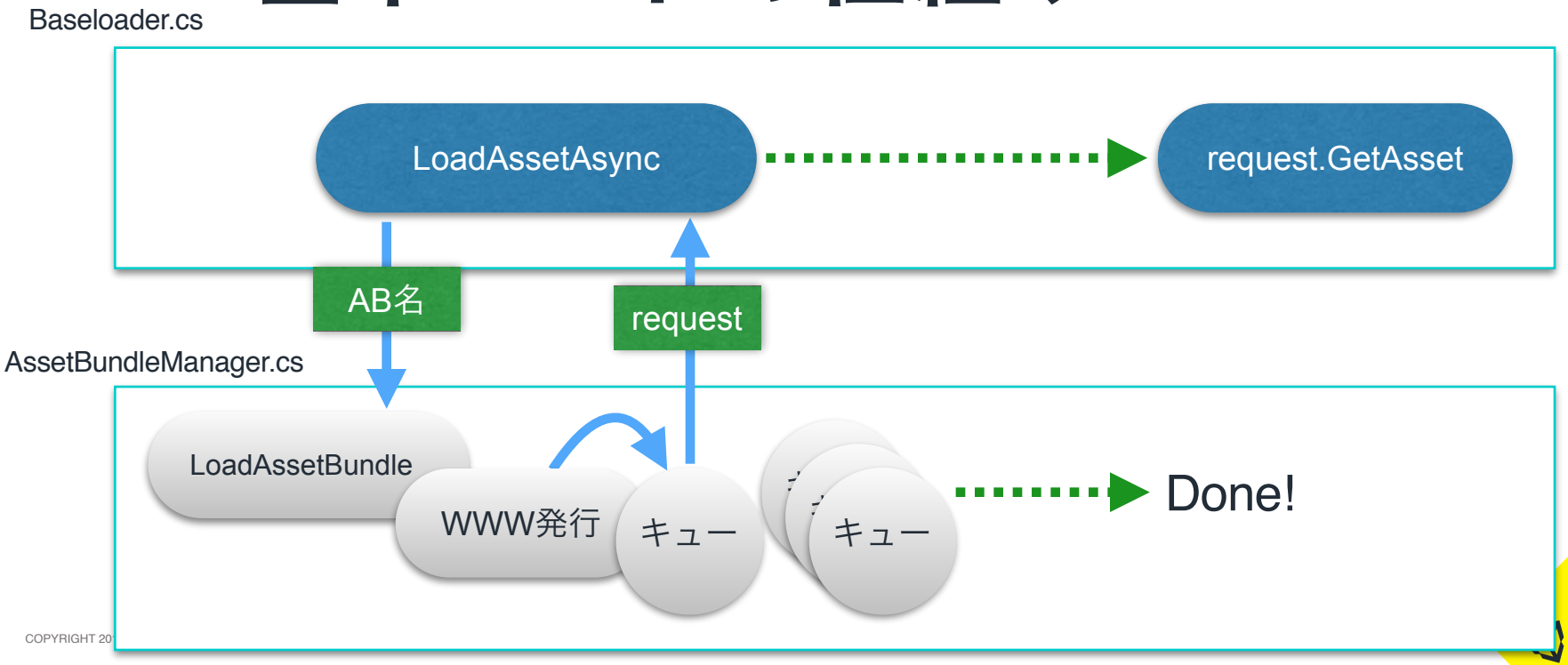
- サンプルプロジェクトをDL
  - http://files.unity3d.com/vincent/assetbundle-demo/users_assetbundle-demo.zip
- サンプルシーン
  - AssetLoader(Assetロードのサンプル)
  - SceneLoader(シーンロードのサンプル)
  - VariantLoader(Variantのサンプル)
- スクリプト
  - AssetBundleManager.cs (マネージャークラス)
  - BaseLoader.cs (基本ローダー)

ちょっと
深く掘り下げ

AssetBundleManager.cs

- AssetBundle取り扱いクラスのサンプル
- 機能
  1. ABの基本的なロードの仕組み
  2. 依存関係の解消
  3. エディタシミュレーション
  4. Variantsを使った切り替え

AssetBundleManager.cs
1. 基本ロードの仕組み



AssetBundleManager.cs
1. 基本ロードの仕組み

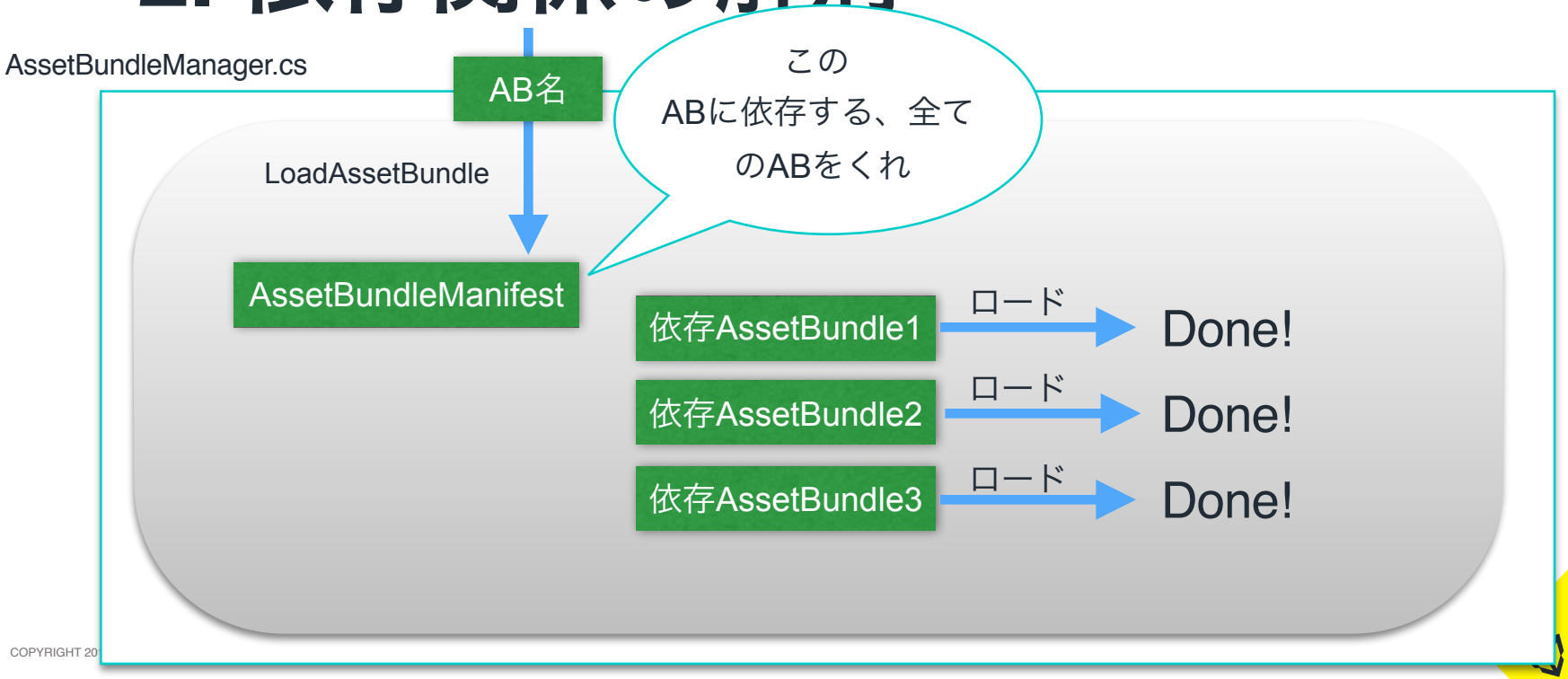
1.アプリ側からLoadAssetAsyncを呼び出される

2.LoadAssetBundleInternalで
WWW.LoadFromCacheOrDownloadを発行して
m_DownloadingWWWsにキューを突っ込む

3.Updateでキューを回して、終わったらDispose

4.依存関係のロードも

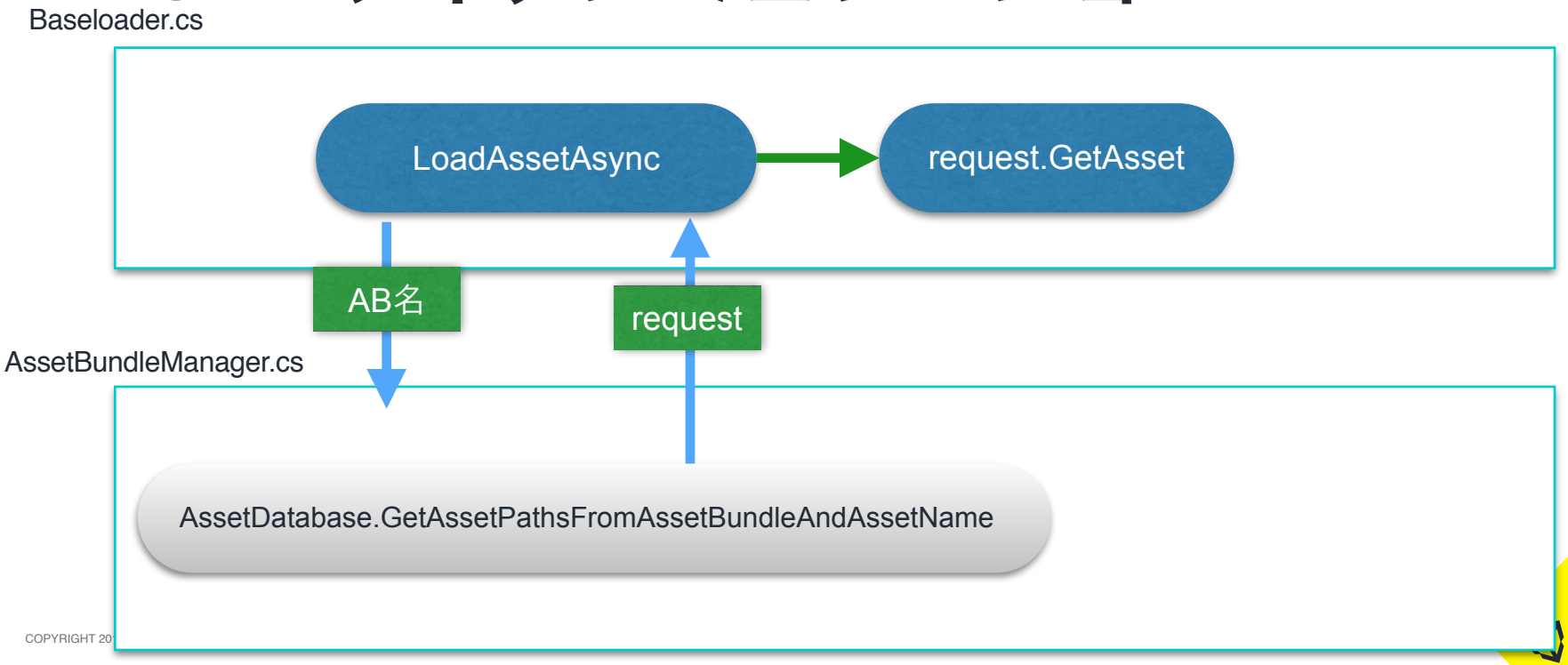
AssetBundleManager.cs
2. 依存関係の解消



AssetBundleManager.cs
2. 依存関係の解消

1. m_AssetBundleManifest.GetAllDependencies(abName) で依存しているAB名を取得
2. 1で取得したABをLoadAssetBundleInternalでロード

AssetBundleManager.cs
3. エディタシミュレーション



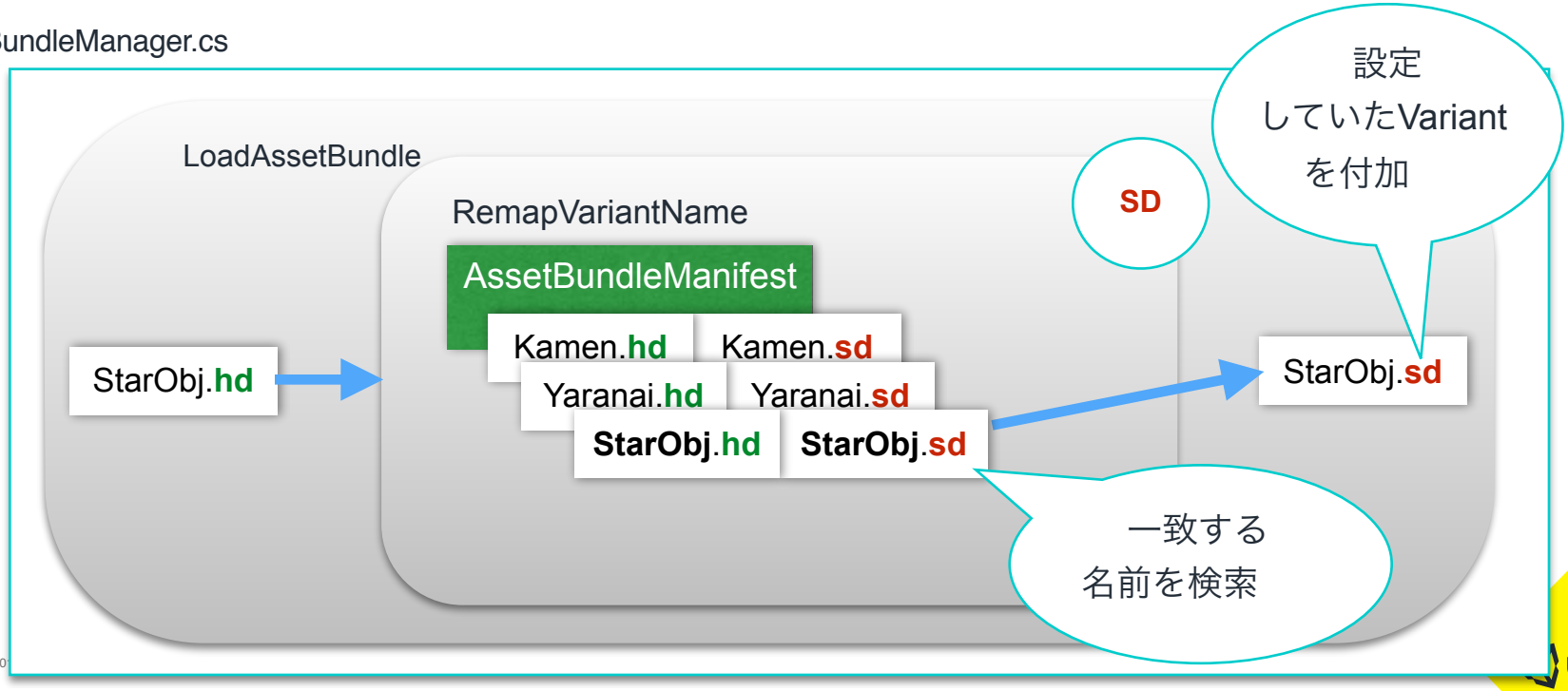
AssetBundleManager.cs
3. エディタシミュレーション

1. エディタ上で可否をトグル
2. Initialize,LoadAssetBundle,UnloadAssetBundleを何もしないで返す
3. LoadAssetAsyncでは

   AssetDatabase.GetAssetPathsFromAssetBundleAndAssetNameでアセットパスを取得して、そこからアセットを取り出す
4. いかにもロードしているかのように返す

AssetBundleManager.cs
4. Variantsを使った切り替え

AssetBundleManager.cs



unity

AssetBundleManager.cs
4. Variantsを使った切り替え

1.m_AssetBundleManifest.GetAllAssetBundlesWithVariant で全Variant付きABを取得

2. 1から該当のアセットバンドル名を検索

3. 最後のvariant文字列(.hd/.sd)を差し替えてアセットバンドル名を返す

デモ

- https://github.com/makoto-unity/NewAssetBundleSample/

- ジュエルセイバーFREE

unity

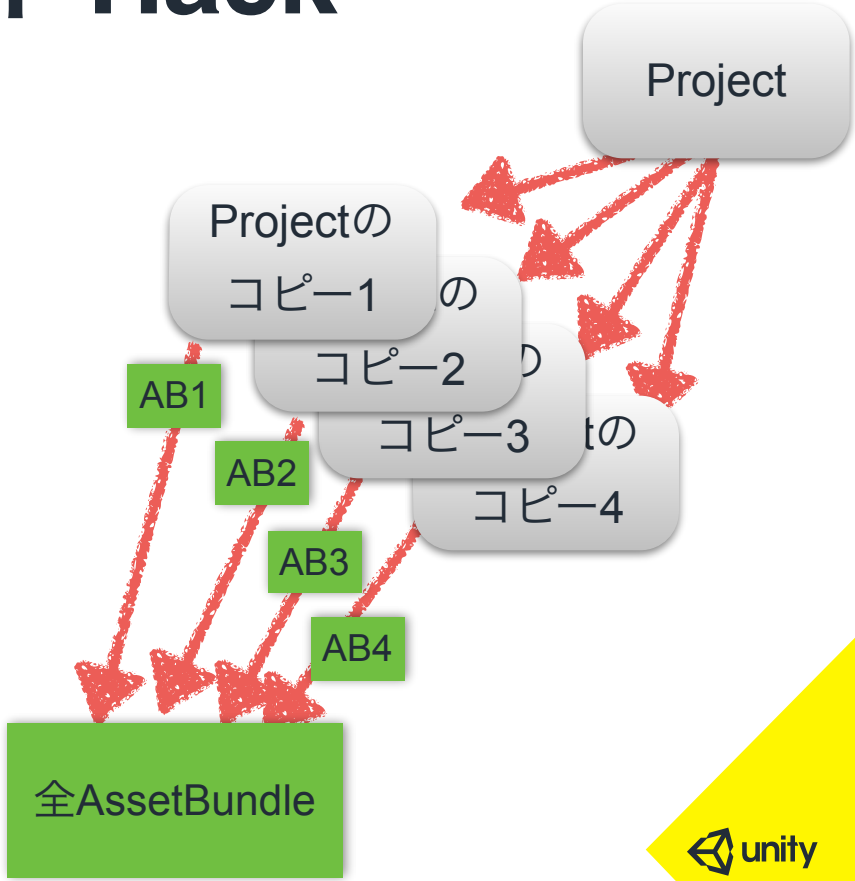
ビルドHack

ところで
AssetBundleビルド
時間かかりません？

そんな
あなたのための
いいHack

マルチプロセスビルドHack

- 何個もビルドプロセスを走らせて、並行でビルド
- 複数のプロジェクトを用意
- プロセスごとに別のプロジェクトを使う
- 最終成果物(AssetBundle)だけを集積



マルチプロセスビルドHack

- プロジェクトファイルを複数作成
  - シンボリックリンク
    - Library, ProjectSettings フォルダをリンク
    - Assets は**その中にある**フォルダとファイルをリンク
    - Tempは**リンクしない**

unity

マルチプロセスビルドHack

- Unityはバッチモードを使う
  - Unity -quit -batchmode -projectPath [path]
    -executeMethod BuildScript.BuildAssetBundlesBatch
- ビルドするAssetBundleは分担を分ける
  - 例：100個あるABのうち、1~25はプロセス1、26〜50はプロセス2…

マルチプロセスビルドHack

- 用法：ABBuildScript.bat [src] [dst] [from] [to]

ビルドスクリプトの例

```
rm -rf $2
mkdir $2
cd $2
ln -s ../$1/AssetBundles .
ln -s ../$1/Library .
ln -s ../$1/ProjectSettings .
mkdir Assets
cd Assets
ln -s ../../$1/Assets/* .
/Applications/Unity5_Latest/Unity.app/Contents/MacOS/Unity -quit -batchmode -
projectPath /Users/makoto/Unity/GitHub/$2 -logFile /Users/makoto/Unity/
GitHub/tmp.log -executeMethod BuildScript.BuildAssetBundlesBatch $3 $4
```

マルチプロセスビルドHack

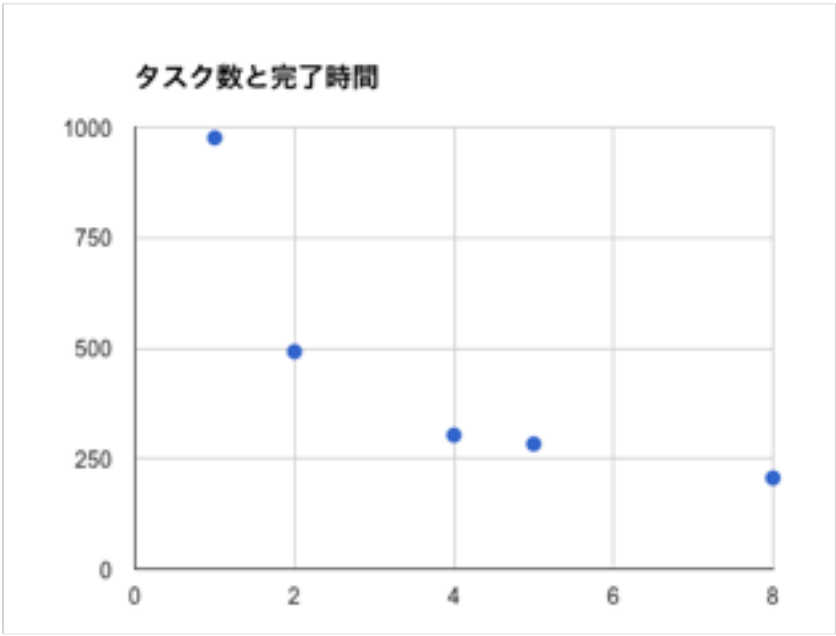
BuildScript.cs

```
public static void BuildAssetBundlesBatch()
{
      // Choose the output path according to the build target.
      string outputPath = Path.Combine(kAssetBundlesOutputPath,
BaseLoader.GetPlatformFolderForAssetBundles(EditorUserBuildSettings.activeBuildTarget) );
      if (!Directory.Exists(outputPath) )
            Directory.CreateDirectory (outputPath);

      string [] args = System.Environment.GetCommandLineArgs();
      int startRatio = int.Parse(args[9]);
      int endRatio   = int.Parse(args[10]);
      string [] allAssetBundleNames = AssetDatabase.GetAllAssetBundleNames();
      int counter = allAssetBundleNames.Length * startRatio / 100;
      int counter_end = allAssetBundleNames.Length * endRatio / 100;
      AssetBundleBuild[] buildMap = new AssetBundleBuild[counter_end - counter];
      int num = 0;
      for( int i=counter ; i<counter_end ; i++ ) {
            string abname = allAssetBundleNames[i];
            buildMap[num].assetBundleName = abname;
            buildMap[num].assetNames = AssetDatabase.GetAssetPathsFromAssetBundle(abname);
            num++;
      }
      BuildPipeline.BuildAssetBundles (outputPath, buildMap, 0, EditorUserBuildSettings.activeBuildTarget);
}
```

マルチプロセスビルドHack

- 結果！
  - x1 16:16
  - x2 8:12
  - x4 5:03
  - x5 4:43
  - x8 3:26
    (MacBookPro Retinaコア4)
- ほぼ比例して時間短縮できる



unity

マルチプロセスビルドHack

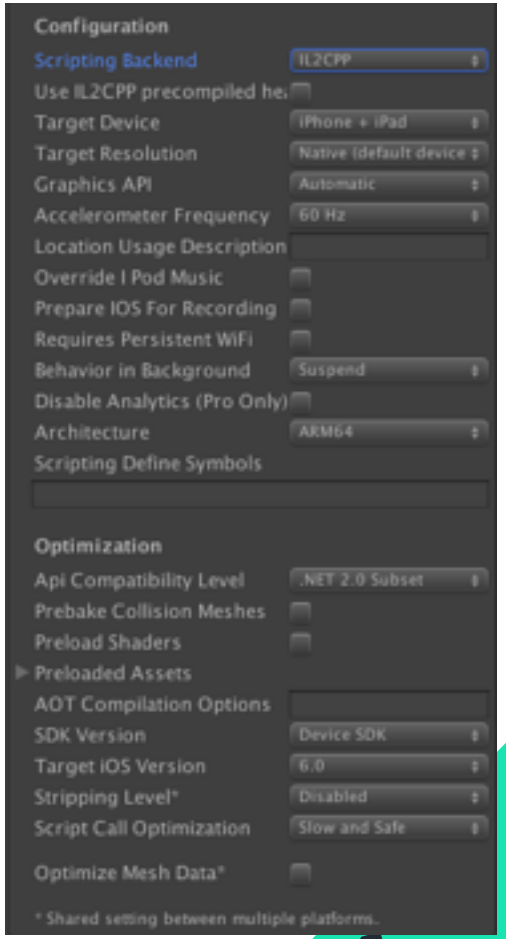
- 注意点
  - 未来永劫使えるとも限らない！
  - そもそもUnityのAssetBundleビルドがマルチスレッド化すればいいのでは…
  - 運用には自己責任で…

ビルド

ビルドの変更点

- IL2CPP
  - 64bit対応のため
  - ビルドサイズ問題は？

    →まさに今、対応中
- Use IL2CPP precompiled header
  - Xcodeビルド時間短縮(例: 63sec→54sec)
- Prebake Collision Meshes
  - 事前にMeshColliderの計算をしておく
  - サイズが若干大きくなるトレードオフ



UnityEditor.iOS.Xcode

- 待望のXcodeプロジェクトを直接編集可能に！
- frameworkの追加／コンパイルオプションの追加／lib,headerパスの追加
- ドキュメントがまだない
- https://bitbucket.org/Unity-Technologies/iosnativecodesamples/src/a0bc90e7d6358e456caf25d717134864218740a7/NativeIntegration/Misc/UpdateXcodeProject/Assets/Editor/XcodeProjectUpdater.cs?at=stable
- http://bit.ly/1CSDBYo (短縮URL)

UnityEditor.iOS.Xcode

```
[PostProcessBuild]
public static void OnPostprocessBuild(BuildTarget buildTarget, string path) {

     if (buildTarget == BuildTarget.iOS) {
          string projPath = path + "/Unity-iPhone.xcodeproj/project.pbxproj";
          PBXProject proj = new PBXProject();
          proj.ReadFromString(File.ReadAllText(projPath));

          string target = proj.TargetGuidByName("Unity-iPhone");
          // こうやって他frameworkとかを追加
          CopyAndReplaceDirectory("Assets/OpenCVForUnity/iOSforXcode/opencv2.framework",
                                  Path.Combine(path, "Frameworks/opencv2.framework"));
          proj.AddFileToBuild(target, proj.AddFile("Frameworks/opencv2.framework",
                                                   "Frameworks/opencv2.framework",
                                                   PBXSourceTree.Source));
          File.WriteAllText(projPath, proj.WriteToString());
     }
}
```

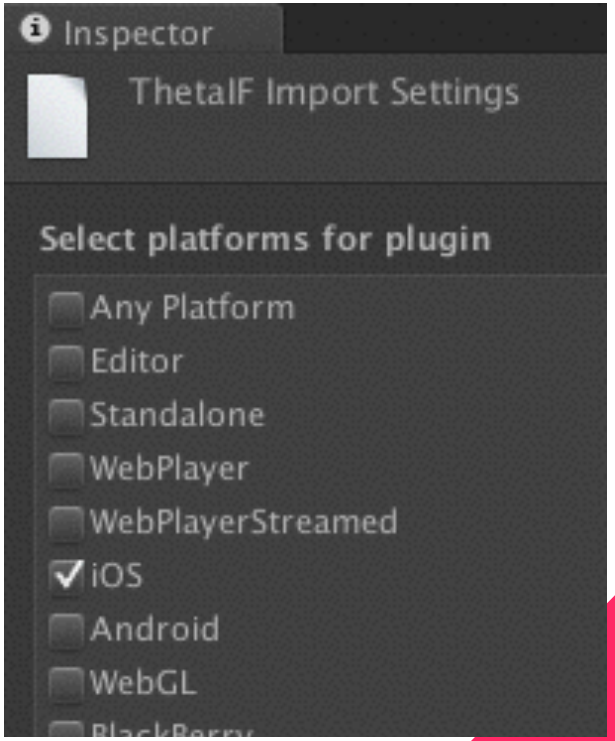
http://bit.ly/1CSDBYo

unity

Plug-in

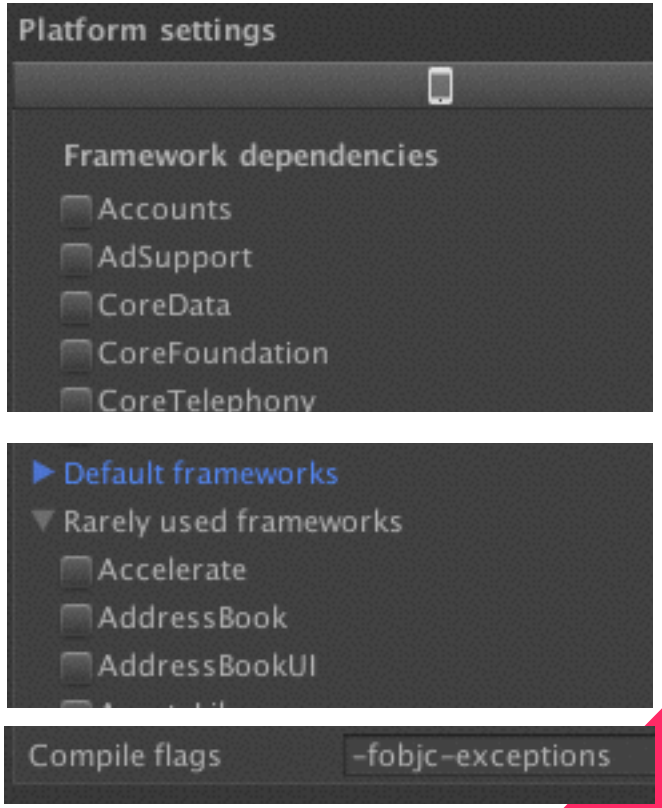
Plug-in の変更点

- Plugins フォルダ以外でも追加可能
- それぞれを個別で選択可能
- 重複も可能()



Plug-in の変更点

- 依存するFrameworkを選択可
- コンパイルオプションを設定可



Plug-in の変更点

- 「全部再設定しなくちゃいけないの？」
  →インポート時に Plugins/iOS/ に置けば自動的に認識
- 「画像とかPluginとして認識されないけど」
  → Plugins フォルダに置きましょう

Native Code Plug-in (復習)

```
[DllImport("__Internal")]
private static extern void callNativeFunc();
```

Unityからネイティブの関数を呼び出し

```
void callNativeFunc()
{
    [[TheNativeClass instance] theFunc];
}
```

CからObjCを呼び出し

```
@implementation TheNativeClass
static TheNativeClass *instance_ = nil;
+ (TheNativeClass *)instance{
    if (!instance_) {
        instance_ = [TheNativeClass new];
    }
    return instance_;
}
- (void)theFunc { NSLog(@“call func”); }
@end
```

ObjCを呼び出し

Native Code Plug-in (直接呼出し型)

- ObjC.cs

```
public class ObjC
{
    [DllImport("__Internal")]
    public static extern IntPtr objc_getClass (string name);

    [DllImport("__Internal")]
    public static extern IntPtr sel_registerName (string name);

    [DllImport("__Internal")]
    public static  extern IntPtr objc_msgSend (IntPtr self, IntPtr op);
}
```

Native Code Plug-in (直接呼出し型)

```
IntPtr instance;
IntPtr classInfo;
void Start() {
  classInfo = ObjC.objc_getClass("TheNativeClass");
  instance = ObjC.objc_msgSend(ObjC.objc_msgSend(classInfo,
                               ObjC.sel_registerName(“alloc")),
                               ObjC.sel_registerName("init"));
}
```

ObjCを
直接呼んで
instance生成

```
void Update() {
  ObjC.objc_msgSend(instance,ObjC.sel_registerName(“theFunc"));
}
```

ObjC関数を
直接呼び出し

```
@implementation TheNativeClass
- (void)theFunc { NSLog(@“call func”); }
@end
```

ObjCを
呼び出し

unity

Native Code Plug-in (直接呼出し型)

- 利点
  - C → ObjCの橋渡し部分のコードを書かなくても良い
  - Singletonを使わない
- 注意点
  - 引数が使えない

- http://tsubakit1.hateblo.jp/entry/2014/08/14/022012