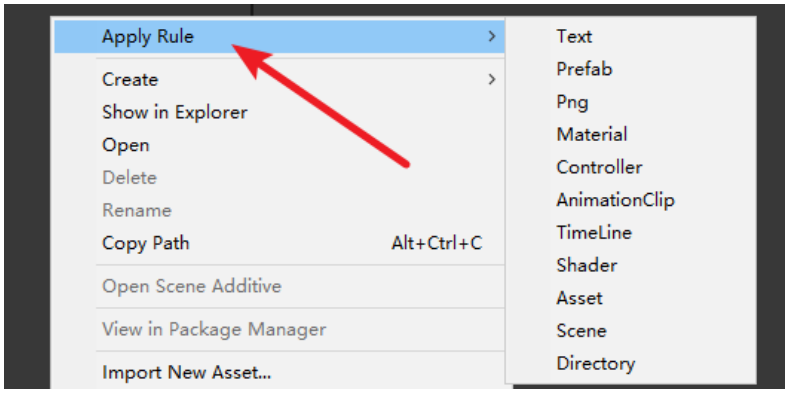
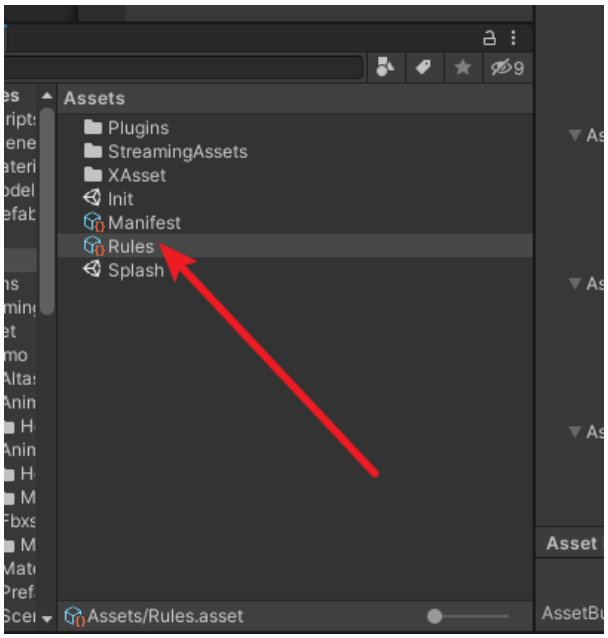
主要看MenuItems.cs 文件

1 Apply Rule，对应的是Rules.assets文件






```
internal static BuildRules GetBuildRules ()
{
    return GetAsset<BuildRules> ( path: "Assets/Rules.asset");
}
```

这些都可以自定义类型

```
[Header("Patterns")]
public string searchPatternAsset = "*.asset";
public string searchPatternController = "*.controller";
public string searchPatternDir = "*";
public string searchPatternMaterial = "*.mat";
public string searchPatternPng = "*.png";
public string searchPatternPrefab = "*.prefab";
public string searchPatternScene = "*.unity";
public string searchPatternAnimationClip = "*.anim";
public string searchPatternTimeLine = "*.playable";
public string searchPatternShader = "*.shader";
public string searchPatternText = "*.txt,*.bytes,*.json,*.csv,*.xml,*htm,*.html,*.yaml,*.fnt";
public static bool nameByHash = true;
```

最后都是加入到rules列表里

```
11 usages  hexinping
private static void AddRulesForSelection(BuildRules rules, string searchPattern)
{
    var isDir = rules.searchPatternDir.Equals(searchPattern);
    foreach (var item in Selection.objects)
    {
        var path = AssetDatabase.GetAssetPath(item);
        var rule = new BuildRule
        {
            searchPath = path,
            searchPattern = searchPattern,
            nameBy = isDir ? NameBy.Directory : NameBy.Path
        };
        ArrayUtility.Add(ref rules.rules, rule); //加到rules数组里
    }

    EditorUtility.SetDirty(rules);
    AssetDatabase.SaveAssets();
}
```

==========

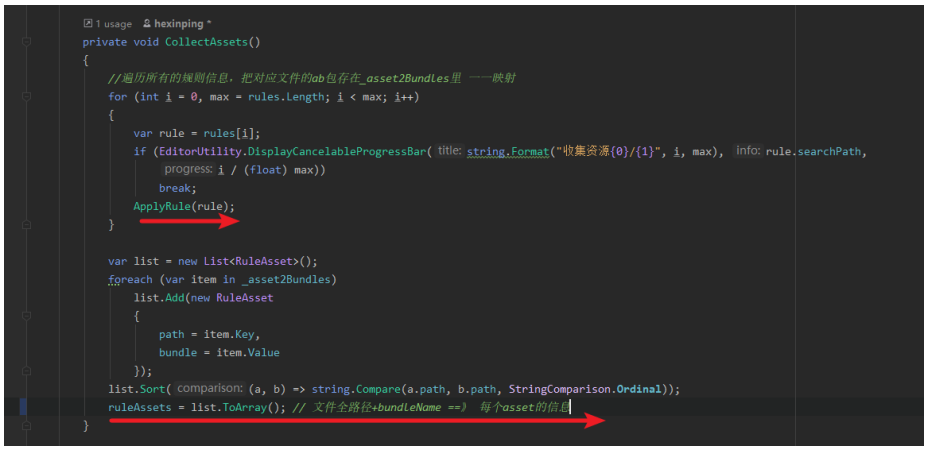
2 Build Rule，调用BuildScript.AppBuildRules方法，最后还是调用BuildRules.Apply方法

```
[MenuItem(KApplyBuildRules)]
hexinping
private static void ApplyBuildRules()
{
    var watch = new Stopwatch();
    watch.Start();
    BuildScript.ApplyBuildRules();
    watch.Stop();
    Debug.Log( message: "ApplyBuildRules " + watch.ElapsedMilliseconds + " ms.");
}
```

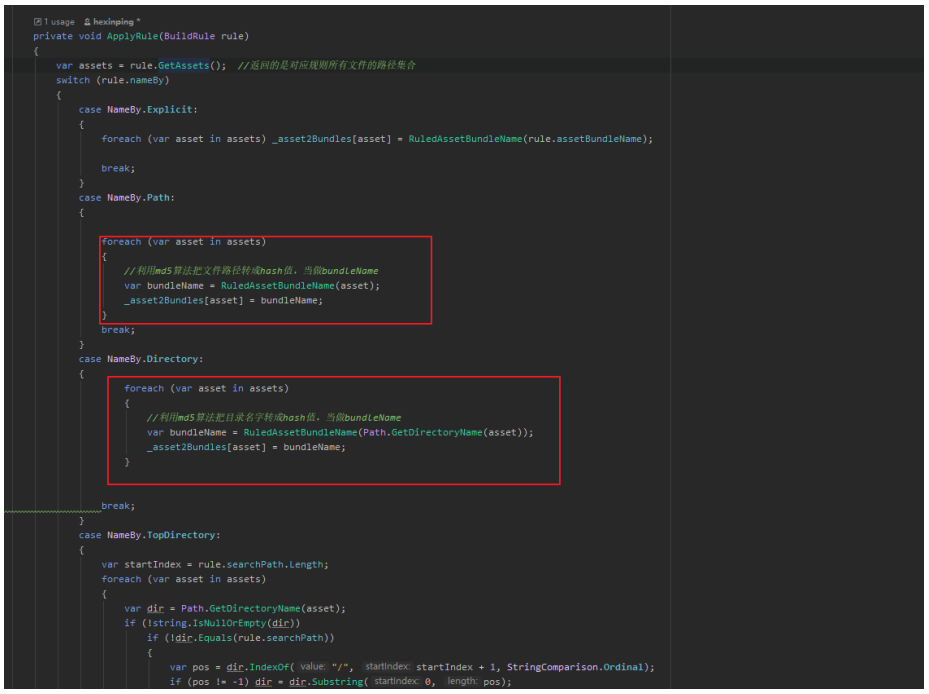
BuildRules.Apply==》获取每个rule的相关信息（ruleAssets和ruleBundles）

```
1 usage  hexinping
public void Apply()
{
    //先清一遍数据
    Clear();

    //收集Assets
    CollectAssets();

    //分析Assets，记录ab之间的依赖项
    AnalysisAssets();

    //优化Asset，处理重复资源
    OptimizeAssets();

    //保存数据
    Save();
}
```

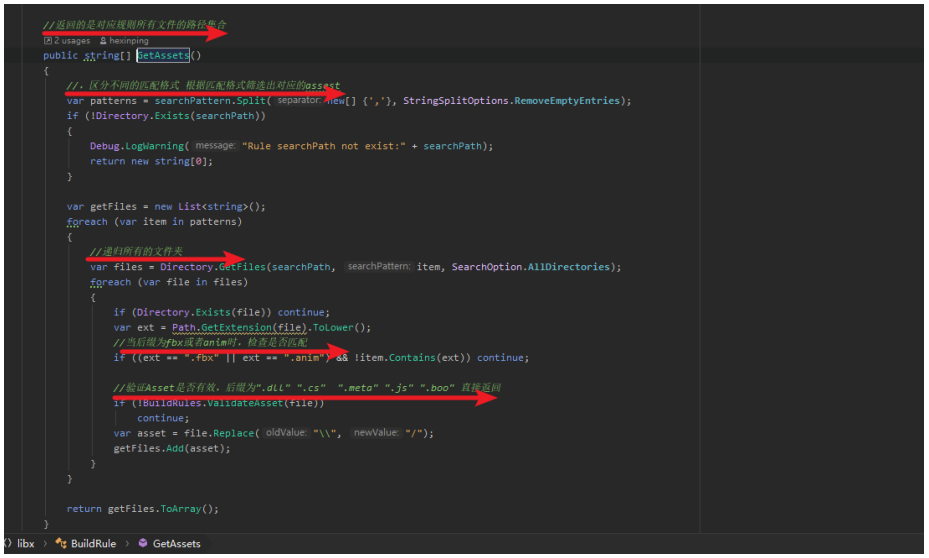
a. CollectAssets， ruleAssets里记录的是文件和对应bundle的信息



```
1 usage  hexinping *
private void CollectAssets()
{
    //遍历所有的规则信息，把对应文件的ab包存在_asset2Bundles里 一一映射
    for (int i = 0, max = rules.Length; i < max; i++)
    {
        var rule = rules[i];
        if (EditorUtility.DisplayCancelableProgressBar( title: string.Format("收集资源{0}/{1}", i, max),  info: rule.searchPath,
            progress: i / (float) max))
            break;
        ApplyRule(rule);
    }

    var list = new List<RuleAsset>();
    foreach (var item in _asset2Bundles)
        list.Add(new RuleAsset
        {
            path = item.Key,
            bundle = item.Value
        });
    list.Sort( comparison: (a, b) => string.Compare(a.path, b.path, StringComparison.Ordinal));
    ruleAssets = list.ToArray(); // 文件全路径+bundleName ==》 每个asset的信息
}
```



```
1 usage  hexinping *
private void ApplyRule(BuildRule rule)
{
    var assets = rule.GetAssets();  //返回的是对应规则所有文件的路径集合
    switch (rule.nameBy)
    {
        case NameBy.Explicit:
        {
            foreach (var asset in assets) _asset2Bundles[asset] = RuledAssetBundleName(rule.assetBundleName);

            break;
        }
        case NameBy.Path:
        {

            foreach (var asset in assets)
            {
                //利用md5算法把文件路径转成hash值，当做bundleName
                var bundleName = RuledAssetBundleName(asset);
                _asset2Bundles[asset] = bundleName;
            }
            break;
        }
        case NameBy.Directory:
        {
            foreach (var asset in assets)
            {
                //利用md5算法把目录名字转成hash值，当做bundleName
                var bundleName = RuledAssetBundleName(Path.GetDirectoryName(asset));
                _asset2Bundles[asset] = bundleName;
            }

            break;
        }
        case NameBy.TopDirectory:
        {
            var startIndex = rule.searchPath.Length;
            foreach (var asset in assets)
            {
                var dir = Path.GetDirectoryName(asset);
                if (!string.IsNullOrEmpty(dir))
                    if (!dir.Equals(rule.searchPath))
                    {
                        var pos = dir.IndexOf( value: "/",  startIndex: startIndex + 1, StringComparison.Ordinal);
                        if (pos != -1) dir = dir.Substring( startIndex: 0,  length: pos);
```



```
//返回的是对应规则所有文件的路径集合
2 usages  hexinping
public string[] GetAssets()
{
    //. 区分不同的匹配格式 根据匹配格式筛选出对应的asset
    var patterns = searchPattern.Split( separator: new[] {','}, StringSplitOptions.RemoveEmptyEntries);
    if (!Directory.Exists(searchPath))
    {
        Debug.LogWarning( message: "Rule searchPath not exist:" + searchPath);
        return new string[0];
    }

    var getFiles = new List<string>();
    foreach (var item in patterns)
    {
        //递归所有的文件夹
        var files = Directory.GetFiles(searchPath,  searchPattern: item, SearchOption.AllDirectories);
        foreach (var file in files)
        {
            if (Directory.Exists(file)) continue;
            var ext = Path.GetExtension(file).ToLower();
            //当后缀为fbx或者anim时，检查是否匹配
            if ((ext == ".fbx" || ext == ".anim") && !item.Contains(ext)) continue;

            //验证Asset是否有效，后缀为".dll" ".cs"  ".meta" ".js" ".boo" 直接返回
            if (!BuildRules.ValidateAsset(file))
                continue;
            var asset = file.Replace( oldValue: "\\",  newValue: "/");
            getFiles.Add(asset);
        }
    }

    return getFiles.ToArray();
}
libx  BuildRule  GetAssets
```

b. AnalysisAssets

```
//分析Assets，记录ab之间的依赖项
1 usage  hexinping
private void AnalysisAssets()
{
    var getBundles = GetBundles(); //返回的是一个Dictionary<BundleName, List<FilePath>>
    int i = 0, max = getBundles.Count;
    foreach (var item in getBundles)
    {
        var bundle = item.Key;
        if (EditorUtility.DisplayCancelableProgressBar( title: string.Format("分析依赖{0}/{1}", i, max),  info: bundle,
            progress: i / (float) max)) break;
        var assetPaths = getBundles[bundle];

        //判断list里有场景资源 && 并且所有的都为场景
        if (assetPaths.Exists(IsScene) && !assetPaths.TrueForAll(IsScene))
            _conflicted.Add(bundle, assetPaths.ToArray());

        //AssetDatabase.GetDependencies ==> 返回文件依赖项，第二个参数为true的话连间接依赖也能获取
        var dependencies = AssetDatabase.GetDependencies( pathNames: assetPaths.ToArray(),  recursive: true);
        if (dependencies.Length > 0)
            foreach (var asset in dependencies)
                if (ValidateAsset(asset))
                    Track(asset, bundle);
        i++;
    }
}
```

```
//asset 是全路径
1 usage  hexinping
private void Track(string asset, string bundle)
{
    HashSet<string> assets;
    if (!_tracker.TryGetValue(asset, out assets))
    {
        assets = new HashSet<string>();
        _tracker.Add(asset, assets);
    }

    assets.Add(bundle);
    if (assets.Count > 1)
    {
        //重复资源
        string bundleName;
        _asset2Bundles.TryGetValue(asset, out bundleName);
        if (string.IsNullOrEmpty(bundleName))
        {
            _duplicated.Add(asset);
        }
    }
}
```

c. OptimizeAssets

```
//优化Asset，处理重复资源
1 usage  hexinping
private void OptimizeAssets()
{
    //如果有冲突资源处理
    int i = 0, max = _conflicted.Count;
    foreach (var item in _conflicted)
    {
        if (EditorUtility.DisplayCancelableProgressBar( title: string.Format("优化冲突{0}/{1}", i, max),  info: item.Key,
            progress: i / (float) max)) break;
        var list = item.Value;
        foreach (var asset in list)
            if (!IsScene(asset))
                _duplicated.Add(asset);
        i++;
    }

    for (i = 0, max = _duplicated.Count; i < max; i++)
    {
        var item = _duplicated[i];
        if (EditorUtility.DisplayCancelableProgressBar( title: string.Format("优化冗余{0}/{1}", i, max),  info: item,
            progress: i / (float) max)) break;
        OptimizeAsset(item);
    }
}
```

d. Save 保存数据，ruleBundles记录的是bundle对应的文件信息

```
1 usage  hexinping *
private void Save()
{
    var getBundles = GetBundles(); //返回的是一个Dictionary<BundleName, List<FilePath>>
    ruleBundles = new RuleBundle[getBundles.Count]; //bundleName+文件列表信息 ==》每个Bundle的信息
    var i = 0;
    foreach (var item in getBundles)
    {
        ruleBundles[i] = new RuleBundle
        {
            name = item.Key, //BundleName
            assets = item.Value.ToArray() //文件列表
        };
        i++;
    }

    EditorUtility.ClearProgressBar();
    EditorUtility.SetDirty( target: this);
    AssetDatabase.SaveAssets();
}
```

==========

3 Builde Bundles，会强制调用一遍ApplyBuildRules，再调用BuildAssetBundles

```
[MenuItem(KBuildAssetBundles)]
hexinping
private static void BuildAssetBundles()
{
    var watch = new Stopwatch();
    watch.Start();
    //首先执行一遍apply rule
    BuildScript.ApplyBuildRules ();
    BuildScript.BuildAssetBundles();
    watch.Stop();
    Debug.Log( message: "BuildAssetBundles " + watch.ElapsedMilliseconds + " ms.");
}
```

a. 打不同的ab包

```
// Choose the output path according to the build target.
var outputPath = CreateAssetBundleDirectory ();

//使用LZ4压缩格式
/*
 LZMA ==》对整个ab包的字节数组进行压缩
LZ4 ==》对单独的Assets的字节进行压缩


AssetBundle.LoadFromFile==>加载LZ4的ab包，API将只加载AssetBundle的头部，并将剩余的数据留在磁盘上。

AssetBundle的Objects 会按需加载，比如加载方法(例如AssetBundle.Load)被调用或其InstanceID被间接引用的时候。在这种情况下，不会消耗过多的内存。
 */
const BuildAssetBundleOptions options = BuildAssetBundleOptions.ChunkBasedCompression;


var targetPlatform = EditorUserBuildSettings.activeBuildTarget;
var rules = GetBuildRules ();
var builds = rules.GetBuilds ();
//打不同的ab包
var assetBundleManifest = BuildPipeline.BuildAssetBundles (outputPath, builds, options, targetPlatform); //先构建Manifest
if (assetBundleManifest == null) {
    return;
}
```

b. 刷新Manifest.asset文件

```
//刷新Manifest.assets文件
var manifest = GetManifest();
var dirs = new List<string>();
var assets = new List<AssetRef>();
var bundles = assetBundleManifest.GetAllAssetBundles(); //拿到所有的bundle
var bundle2Ids = new Dictionary<string, int>();
for (var index = 0; index < bundles.Length; index++)
{
    var bundle = bundles[index];
    bundle2Ids[bundle] = index;
}

var bundleRefs = new List<BundleRef>();
for (var index = 0; index < bundles.Length; index++)
{
    var bundle = bundles[index];
    var deps = assetBundleManifest.GetAllDependencies(bundle); //连循环依赖也会拿到
    var path = string.Format("{0}/{1}", outputPath, bundle);
    if (File.Exists(path))
    {
        using (var stream = File.OpenRead(path))                    构建信息
        {
            bundleRefs.Add(new BundleRef
            {
                name = bundle,
                id = index,
                deps = Array.ConvertAll(deps, converter: input => bundle2Ids[input]), //依赖项
                len = stream.Length,
                hash = assetBundleManifest.GetAssetBundleHash(bundle).ToString(),
            });
        }
    }
    else
    {
        Debug.LogError( message: path + " file not exsit.");
    }
}
```

```
//rules.ruleAssets 文件全路径+bundleName ==》 每个asset的信息
for (var i = 0; i < rules.ruleAssets.Length; i++)
{
    var item = rules.ruleAssets[i];
    var path = item.path; //文件全路径
    var dir = Path.GetDirectoryName(path).Replace( oldValue: "\\", newValue: "/");
    var index = dirs.FindIndex( match: o => o.Equals(dir));
    if (index == -1)
    {
        index = dirs.Count;
        dirs.Add(dir);
    }

    //bundle下标，对应文件夹下标，文件名字
    var asset = new AssetRef { bundle = bundle2Ids[item.bundle], dir = index, name = Path.GetFileName(path) };
    assets.Add(asset);
}

manifest.dirs = dirs.ToArray();  //文件对应文件夹信息
manifest.assets = assets.ToArray();  // //bundle下标，对应文件夹下标，文件名字
manifest.bundles = bundleRefs.ToArray(); //bundle的信息（名字，大小，hash值，依赖项，文件夹下标）
```

游戏启动时会优先加载Manifest的ab包，记录bundle信息

```
1 usage  hexinping
internal static void OnLoadManifest(Manifest manifest)
{
    _activeVariants.AddRange(manifest.activeVariants);

    /*
        manifest.dirs = dirs.ToArray();  //文件对应文件夹信息
        manifest.assets = assets.ToArray();  // //bundle下标，对应文件夹下标，文件名字
        manifest.bundles = bundleRefs.ToArray(); //bundle的信息（名字，大小，hash值，依赖项，文件夹下标）
     */
    var assets = manifest.assets;
    var dirs = manifest.dirs;
    var bundles = manifest.bundles;

    //存储bundle的依赖信息
    //key：当前bundle
    //value：当前bundle依赖的bundle
    foreach (var item in bundles)
        _bundleToDependencies[item.name] = Array.ConvertAll(item.deps,  converter: id => bundles[id].name);

    //asset与bundle的对应关系
    //Key：asset的全路径
    //value：对应的bundleName
    foreach (var item in assets)
    {
        var path = string.Format("{0}/{1}", dirs[item.dir], item.name);
        if (item.bundle >= 0 && item.bundle < bundles.Length)
        {
            _assetToBundles[path] = bundles[item.bundle].name;
        }
        else
        {
            Debug.LogError( message: string.Format("{0} bundle {1} not exist.", path, item.bundle));
        }
    }
}
```

c. 打Manifest的ab包

```
var manifestBundleName = "manifest.unity3d";
builds = new[] {
    new AssetBundleBuild {
        assetNames = new[] { AssetDatabase.GetAssetPath (manifest), },
        assetBundleName = manifestBundleName
    }
};

//打manifest的ab包
BuildPipeline.BuildAssetBundles(outputPath, builds, options, targetPlatform);
ArrayUtility.Add(ref bundles, manifestBundleName);
```

d：写入版本信息

```
    //写入版本信息
    int version = GetBuildRules().AddVersion();
    Versions.BuildVersions(outputPath, bundles, version);
}
```

==========

4 Build Player， 调用BuildeScript的BuildStandalonePlayer方法，

EditorUserBuildSetting.activeBuildTarget

Application.platform还是editor

```
4 usages  hexinping *
public static string GetPlatformName ()
{
    //根据编辑器设置平台输出不同的文件夹名字，切换成android就是android
    return GetPlatformForAssetBundles (EditorUserBuildSettings.activeBuildTarget);
}
```

```
public static void BuildStandalonePlayer ()
{
    //不同平台不同输出路径
    var outputPath =
        Path.Combine (Environment.CurrentDirectory,
            "Build/" + GetPlatformName ()
                .ToLower ()); //EditorUtility.SaveFolderPanel("Choose Location of the Built Game", "", "");
    if (outputPath.Length == 0)
        return;
    if (Directory.Exists(outputPath))
        Directory.Delete(outputPath, recursive: true);

    var levels = GetLevelsFromBuildSettings ();
    if (levels.Length == 0) {
        Debug.Log ( message: "Nothing to build.");
        return;
    }

    //构建输出目标名称，包含资源版本号，打包时间
    var targetName = GetBuildTargetName (EditorUserBuildSettings.activeBuildTarget);
    if (targetName == null)
        return;
#if UNITY_5_4 || UNITY_5_3 || UNITY_5_2 || UNITY_5_1 || UNITY_5_0
    BuildOptions option = EditorUserBuildSettings.development ? BuildOptions.Development : BuildOptions.None;
    BuildPipeline.BuildPlayer(levels, path + targetName, EditorUserBuildSettings.activeBuildTarget, option);
#else
    var buildPlayerOptions = new BuildPlayerOptions {
        scenes = levels,
        locationPathName = outputPath + targetName,
        assetBundleManifestPath = GetAssetBundleManifestFilePath (),
        target = EditorUserBuildSettings.activeBuildTarget,
        options = EditorUserBuildSettings.development ? BuildOptions.Development : BuildOptions.None
    };
    BuildPipeline.BuildPlayer (buildPlayerOptions);

    Debug.Log( message: $"BuildPlayer Done, OutPut Path is {buildPlayerOptions.locationPathName.Replace( oldValue: "\\", newValue: "/")}");
    EditorUtility.OpenWithDefaultApp(outputPath);
#endif
}
```

==========

5 View Bundles ==> 打开bundle的下载路径

```
[MenuItem(KViewDataPath)]
hexinping
private static void ViewDataPath()
{
    EditorUtility.OpenWithDefaultApp(Application.persistentDataPath);
}
```

==========

6 Copy Bundles，仅仅把res和ver文件拷贝进去， ver文件是打包是生成的版本记录文件，里面记录了所有的ab信息，热更新阶段会使用

```
[MenuItem(KCopyBundles)]
hexinping
private static void CopyAssetBundles()
{
    BuildScript.CopyAssetBundlesTo(Application.streamingAssetsPath);
}
```

```
1 usage  hexinping
public static void CopyAssetBundlesTo (string path)
{
    var files = new[] {
        Versions.Dataname,
        Versions.Filename,
    };
    if (!Directory.Exists (path)) {
        Directory.CreateDirectory (path);
    }
    foreach (var item in files) {
        var src = outputPath + "/" + item;
        var dest = Application.streamingAssetsPath + "/" + item;
        if (File.Exists (src)) {
            File.Copy ( sourceFileName: src, dest,  overwrite: true);
        }
    }
}
```

```
public const string Dataname = "res";
public const string Filename = "ver";
```