核心类两个：Assets, AssetRequest

一个Asset对应的一个Request

Request的种类
ManifestRequest ==> 加载mainfest的
BundleRequest ==> 加载Bundle， 继承AssetRequest
BundleRequestAsync ==> 异步加载Bundle ，继承BundleRequest
BundleAssetRequest ==> 加载Bundle中的Asset，包含一个BundleRequest 对象，继承AssetRequest
BundleAssetRequestAsync ==> 异步加载Bundle中的Asset，包含一个BundleRequest 对象，继承BundleAssetRequest

SceneAssetRequest ==》场景加载， 继承AssetRequest
SceneAssetRequestAsync ==》异步场景加载， 继承SceneAssetRequest

WebAssetRequest ==》网络资源或者本地资源加载， 继承AssetRequest
WebBundleRequest ==》网络Bundle加载，继承BundleRequest

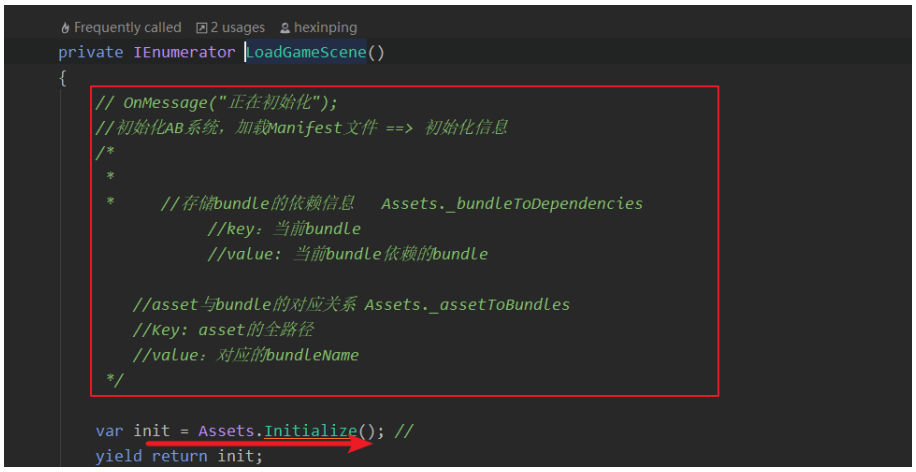
加载资源用状态控制

```
53 usages  hexinping  7 exposing APIs
public enum LoadState
{
    Init, //初始化
    LoadAssetBundle, //加载ab包
    LoadAsset, //加载对应asset资源
    Loaded, //加载完毕
    Unload // 卸载
}
```

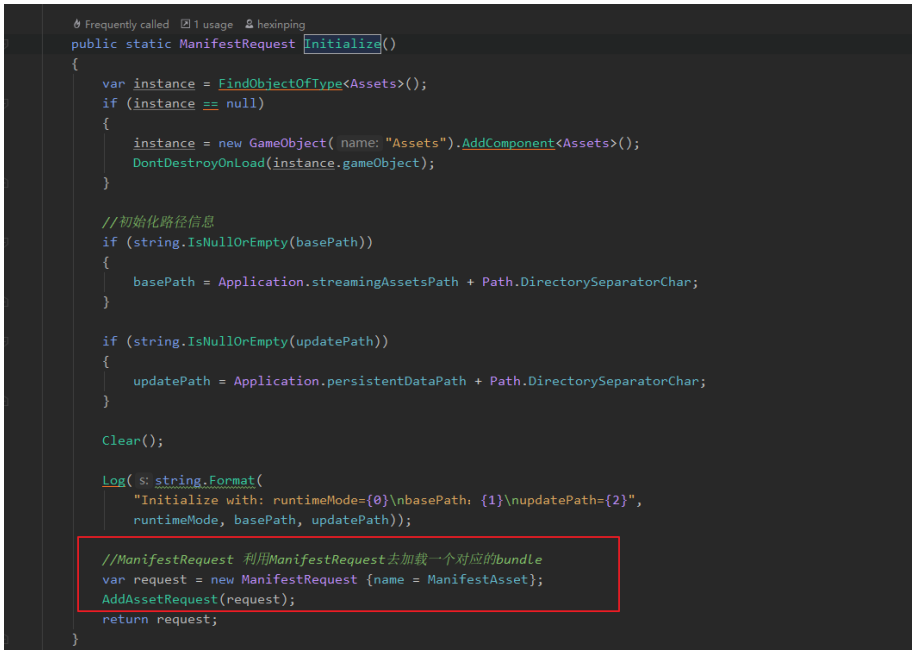
加载资源主要是这个接口，Asset.LoadAsset,会自动去加载asset对应的bundle

```
//异步加载资源，path为资源寻址路径，Type为资源类型
Frequently called  6 usages  hexinping
public static AssetRequest LoadAssetAsync(string path, Type type)
{
    return LoadAsset(path, type, async: true);
}

//同步加载资源，path为资源寻址路径，Type为资源类型
Frequently called  2 usages  hexinping  ··· More
public static AssetRequest LoadAsset(string path, Type type)
{
    return LoadAsset(path, type, async: false);
}
```

整体流程
1 先加载Mainfest文件，Assets.Initialize, 存储bundle和file的信息

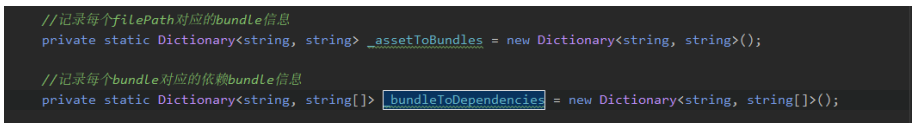


```
Frequently called  2 usages  hexinping
private IEnumerator LoadGameScene()
{
    // OnMessage("正在初始化");
    //初始化AB系统，加载Manifest文件 ==> 初始化信息
    /*
     *
     *     //存储bundle的依赖信息   Assets._bundleToDependencies
              //key：当前bundle
              //value：当前bundle依赖的bundle

        //asset与bundle的对应关系 Assets._assetToBundles
        //Key：asset的全路径
        //value：对应的bundleName
     */

    var init = Assets.Initialize(); //
    yield return init;
```



```
Frequently called  1 usage  hexinping
public static ManifestRequest Initialize()
{
    var instance = FindObjectOfType<Assets>();
    if (instance == null)
    {
        instance = new GameObject( name: "Assets").AddComponent<Assets>();
        DontDestroyOnLoad(instance.gameObject);
    }

    //初始化路径信息
    if (string.IsNullOrEmpty(basePath))
    {
        basePath = Application.streamingAssetsPath + Path.DirectorySeparatorChar;
    }

    if (string.IsNullOrEmpty(updatePath))
    {
        updatePath = Application.persistentDataPath + Path.DirectorySeparatorChar;
    }

    Clear();

    Log( s: string.Format(
        "Initialize with: runtimeMode={0}\nbasePath: {1}\nupdatePath={2}",
        runtimeMode, basePath, updatePath));

    //ManifestRequest 利用ManifestRequest去加载一个对应的bundle
    var request = new ManifestRequest {name = ManifestAsset};
    AddAssetRequest(request);
    return request;
}
```

```
Frequently called  2 usages  hexinping *
private static void AddAssetRequest(AssetRequest request)
{
    //放入列表里，Update不断遍历状态，并且调用不同request的Load方法
    _assets.Add(request.name, request);
    _loadingAssets.Add(request);
    request.Load();
}
```



```
//记录每个filePath对应的bundle信息
private static Dictionary<string, string> _assetToBundles = new Dictionary<string, string>();

//记录每个bundle对应的依赖bundle信息
private static Dictionary<string, string[]> _bundleToDependencies = new Dictionary<string, string[]>();
```

2 先加载Asset对应的Bundle, Asset.LoadBundle, Asset.LoadBundleAsync, 返回都是一个BundleRequest

```
Frequently called  2 usages  hexinping *
internal static BundleRequest LoadBundle(string assetBundleName, bool asyncMode)
{
    if (string.IsNullOrEmpty(assetBundleName))
    {
        Debug.LogError( message: "assetBundleName == null");
        return null;
    }

    assetBundleName = RemapVariantName(assetBundleName);

    //得到加载路径，persitantpath
    var url = GetDataPath(assetBundleName) + assetBundleName;

    BundleRequest bundle;
    //判断对应的bundle是否加载过
    if (_bundles.TryGetValue(url, out bundle))
    {
        bundle.Retain();
        _loadingBundles.Add(bundle); //为什么加载过的还要放进Loading列表里？？防止同一帧加载多个，然后update里用状态判断
        return bundle;
    }

    if (url.StartsWith( value: "http://", StringComparison.Ordinal) ||
        url.StartsWith( value: "https://", StringComparison.Ordinal) ||
        url.StartsWith( value: "file://", StringComparison.Ordinal) ||
        url.StartsWith( value: "ftp://", StringComparison.Ordinal))
        bundle = new WebBundleRequest(); //加载网络Bundle
    else
        bundle = asyncMode ? new BundleRequestAsync() : new BundleRequest();

    bundle.name = url;
    _bundles.Add(url, bundle);

    if (MAX_BUNDLES_PERFRAME > 0 && (bundle is BundleRequestAsync || bundle is WebBundleRequest))
    {
        _toloadBundles.Add(bundle);
    }
    else
    {
        bundle.Load();
        _loadingBundles.Add(bundle); //加入需要加载的bundle列表
        Log( s: "LoadBundle: " + url);
    }

    bundle.Retain();
    return bundle;
}
```

同步bundle加载

BundleRequest.Load ==> 直接调用接口AssetBundle.LoadFromFile

然后直接标记状态为Loaded，然后会直接调用complete方法

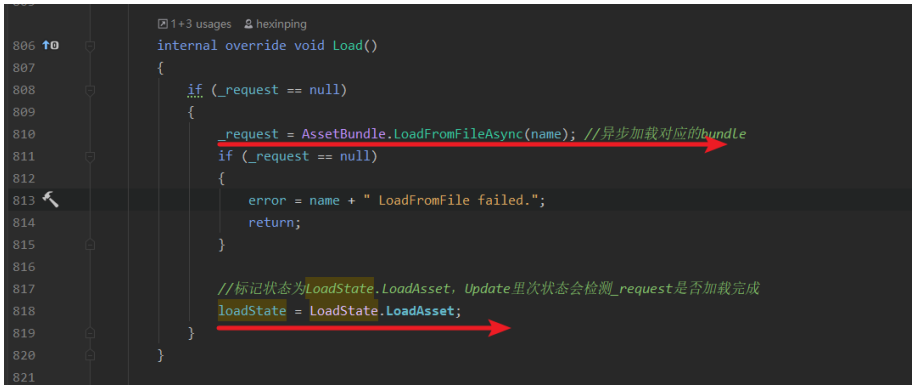
```
public class BundleRequest : AssetRequest
{
    public string assetBundleName { get; set; }

    15 usages  hexinping
    public AssetBundle assetBundle
    {
        get { return asset as AssetBundle; }
        internal set { asset = value; }
    }

    Frequently called  2+1 usages  2 overrides  hexinping *
    internal override void Load()
    {
        asset = AssetBundle.LoadFromFile(name); //同步加载bundle
        if (assetBundle == null)
            error = name + " LoadFromFile failed.";
        loadState = LoadState.Loaded;
    }
```

```
49 usages  hexinping *
public LoadState loadState
{
    get { return _loadState; }
    protected set
    {
        _loadState = value;
        if (value == LoadState.Loaded)
        {
            Complete(); //如果外部传入回调，会调用回调
        }
    }
}
```

异步bundle加载

BundleRequestAsync.Load ==》

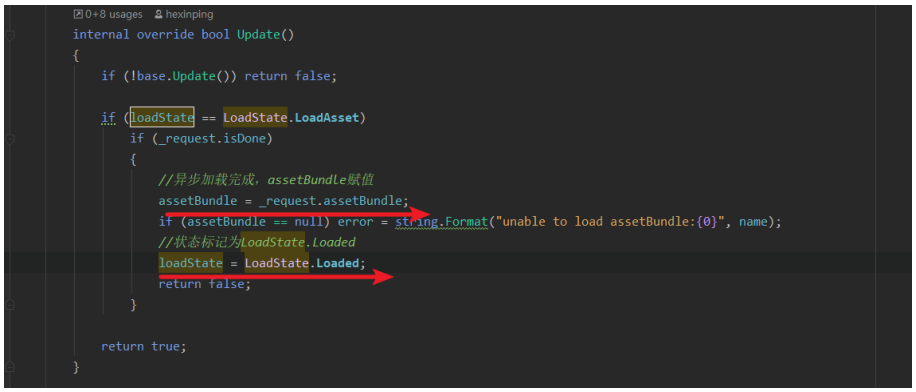
直接调用接口AssetBundle.LoadFromFileAsync，立即标记状态为LoadAsset，Update里检测Bundle加载完成后才会标记状态为Loaded



```
1+3 usages  hexinping
internal override void Load()
{
    if (_request == null)
    {
        _request = AssetBundle.LoadFromFileAsync(name); //异步加载对应的bundle
        if (_request == null)
        {
            error = name + " LoadFromFile failed.";
            return;
        }

        //标记状态为LoadState.LoadAsset，Update里次状态会检测_request是否加载完成
        loadState = LoadState.LoadAsset;
    }
}
```

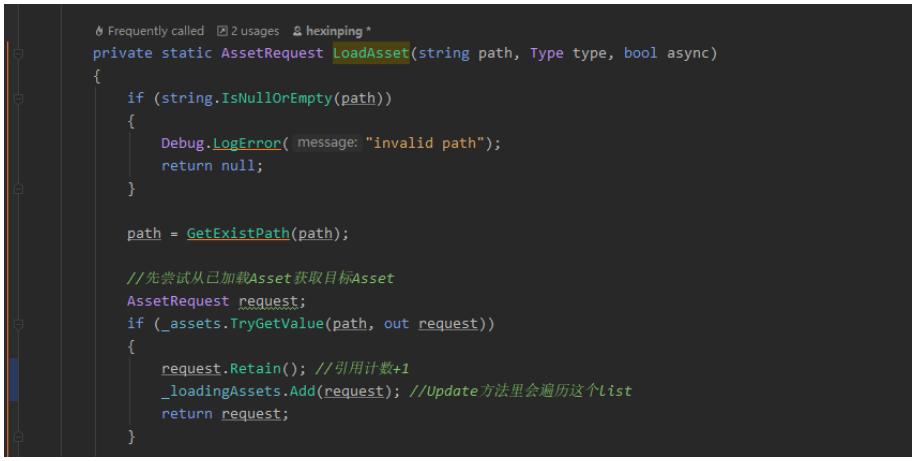
Assets.UpdateBundles里遍历_loadingBundles

```
for (var i = 0; i < _loadingBundles.Count; i++)
{
    var item = _loadingBundles[i];
    if (item.Update()) //刷新每个BundleRequest的状态，item加载完成后会返回false
        continue;
    _loadingBundles.RemoveAt(i); //加载完成从列表里移除
    --i;
}
```



```
0+8 usages  hexinping
internal override bool Update()
{
    if (!base.Update()) return false;

    if (loadState == LoadState.LoadAsset)
        if (_request.isDone)
        {
            //异步加载完成，assetBundle赋值
            assetBundle = _request.assetBundle;
            if (assetBundle == null) error = string.Format("unable to load assetBundle:{0}", name);
            //状态标记为LoadState.Loaded
            loadState = LoadState.Loaded;
            return false;
        }

    return true;
}
```

3 从bundle中加载Asset，Assets.LoadAsset



```
Frequently called  2 usages  hexinping *
private static AssetRequest LoadAsset(string path, Type type, bool async)
{
    if (string.IsNullOrEmpty(path))
    {
        Debug.LogError( message: "invalid path");
        return null;
    }

    path = GetExistPath(path);

    //先尝试从已加载Asset获取目标Asset
    AssetRequest request;
    if (_assets.TryGetValue(path, out request))
    {
        request.Retain(); //引用计数+1
        _loadingAssets.Add(request); //Update方法里会遍历这个List
        return request;
    }
```

```
//如果没找到就需要去获取AB
string assetBundleName;
//如果此AB已存在于本地记录（从Manifest文件读取的），就直接取得AB名，准备加载AB
if (GetAssetBundleName(path, out assetBundleName))
{
    request = async
        ? new BundleAssetRequestAsync(assetBundleName)
        : new BundleAssetRequest(assetBundleName);
}
else
{
    //如果此AB在本地记录未找到
    //如果是网络路径/本地路径（注意一定要有以下前缀之一，否则会被忽略而取不到对象）
    if (path.StartsWith( value: "http://", StringComparison.Ordinal) ||
        path.StartsWith( value: "https://", StringComparison.Ordinal) ||
        path.StartsWith( value: "file://", StringComparison.Ordinal) ||
        path.StartsWith( value: "ftp://", StringComparison.Ordinal) ||
        path.StartsWith( value: "jar:file://", StringComparison.Ordinal))
        request = new WebAssetRequest();
    else
        //如果是本地路径（事实上这个是用AssetDatabase.LoadAssetAtPath去编辑器找的，所以想要读取本地的非AB文件还是用上面的UWR吧，注意加上前缀）
        request = new AssetRequest();
}

request.name = path;
request.assetType = type;
//新增资产请求
AddAssetRequest(request);
//引用计数+1
request.Retain();
Log( s: string.Format("LoadAsset:{0}", path));
return request;
}
```

对应的request被加入到_loadingAssets列表里，在Update里检测

```
Frequently called  2 usages  hexinping *
private static void AddAssetRequest(AssetRequest request)
{
    //放入列表里，Update不断遍历状态，并且调用不同request的Load方法
    _assets.Add(request.name, request);
    _loadingAssets.Add(request);
    request.Load();
}
```

同步加载

BundleAssetRequest.Load ==> 同步加载对应的asset，状态标记为Loaded

```
0+1 usages  1 override  hexinping *
internal override void Load()
{
    //先加载对应bundle
    BundleRequest = Assets.LoadBundle(assetBundleName);
    var names = Assets.GetAllDependencies(assetBundleName);

    //bundle如果有依赖项优先加载依赖项
    foreach (var item in names)
        children.Add( item: Assets.LoadBundle(item));
    var assetName = Path.GetFileName(name);
    var ab = BundleRequest.assetBundle;

    //从bundle中加载asset
    if (ab != null)
        asset = ab.LoadAsset(assetName, assetType);
    if (asset == null) error = "asset == null";
    loadState = LoadState.Loaded; //标记为个状态，IsDone就为true了
}
```

BundleAssetRequest的Update是调用父类AssetRequest的Update方法

```
Frequently called  1 usage  hexinping *
private static void UpdateAssets()
{
    for (var i = 0; i < _loadingAssets.Count; ++i)
    {
        var request = _loadingAssets[i];
        if (request.Update()) //每一帧检测BundleAssetRequest的状态，返回true继续执行循环
            continue;
        _loadingAssets.RemoveAt(i);
        --i;
    }
    //移除无用资产
```

```
Frequently called  8 usages  5 overrides  hexinping *
internal virtual bool Update()
{
    if (checkRequires)
        UpdateRequires();
    if (!isDone)
        return true;
    if (completed == null)
        return false;
    try
    {
        //如果有外部回调，下一帧才会从列表里移除
        completed.Invoke( obj: this);
    }
    catch (Exception ex){...}

    completed = null;
    return false;       返回false后，就从_loadingAssets列表里移除了
}

IEnumerator implementation
}
```

异步加载

BundleAssetRequestAsync.Load==> 仅仅先加载对应bundle以及依赖bundle，状态标记为LoadAssetBundle

```
0+1 usages  hexinping *
internal override void Load()
{
    //异步加载对应的bundle
    BundleRequest = Assets.LoadBundleAsync(assetBundleName);

    //异步加载对应的bundle的依赖bundle
    var bundles = Assets.GetAllDependencies(assetBundleName);
    foreach (var item in bundles) children.Add( item: Assets.LoadBundleAsync(item));
    loadState = LoadState.LoadAssetBundle;
}
```

等所有bundle加载完成，跟不同加载一样Assets.UpdateAssets里遍历_loadingAssets, 但这里调用的是BundleAssetRequestAsync.Update,它重载了父类的Update方法

```
Frequently called  1 usage  hexinping *
private static void UpdateAssets()
{
    for (var i = 0; i < _loadingAssets.Count; ++i)
    {
        var request = _loadingAssets[i];
        if (request.Update()) //每一帧检测BundleAssetRequest的状态，返回true继续执行循环
            continue;
        _loadingAssets.RemoveAt(i);
        --i;
    }
```

```csharp
internal override bool Update()
{
    if (!base.Update()) return false;

    if (loadState == LoadState.Init) return true;

    if (_request == null)
    {
        //判断对应的bundle是否加载完成
        if (!BundleRequest.isDone) return true;
        if (OnError(BundleRequest)) return false;

        //判断对应bundle的依赖项是否加载完成
        for (var i = 0; i < children.Count; i++){...}

        var assetName = Path.GetFileName(name);
        //从bundle中异步加载asset
        _request = BundleRequest.assetBundle.LoadAssetAsync(assetName, assetType);
        if (_request == null)
        {
            error = "request == null";

            //状态标记为Loaded, 并且自动调用Complete()方法
            loadState = LoadState.Loaded;
            return false;
        }

        return true;
    }

    //判断asset是否加载完成
    if (_request.isDone)
    {
        asset = _request.asset;
        loadState = LoadState.Loaded; //标记状态并且调用complete方法
        if (asset == null) error = "asset == null";
        return false;
    }

    return true;
}
```

==

资源卸载，是自动管理的，利用引用计数来管理，只要调用对应AssetRequest.Release方法，Assets.Update方法里会去检查未使用的bundle和asset

```csharp
private static void UpdateAssets()
{
    for (var i = 0; i < _loadingAssets.Count; ++i)
    {
        var request = _loadingAssets[i];
        if (request.Update()) //每一帧检测BundleAssetRequest的状态，返回true继续执行循环
            continue;
        _loadingAssets.RemoveAt(i);
        --i;
    }
    //移除无用资产
    foreach (var item in _assets)
    {
        //IsUnused 中会去判断引用计数
        if (item.Value.isDone && item.Value.IsUnused())
        {
            _unusedAssets.Add(item.Value); //大部分加入的是BundleAssetRequest类型
        }
    }

    if (_unusedAssets.Count > 0)
    {
        for (var i = 0; i < _unusedAssets.Count; ++i)
        {
            var request = _unusedAssets[i];
            Log( string.Format("UnloadAsset:{0}", request.name));
            _assets.Remove(request.name);
            request.Unload();  //自身bundle引用计数减一，依赖的bundle引用计数减一
        }
        _unusedAssets.Clear();
    }
```

```
Frequently called  3 usages  hexinping *
public bool IsUnused ()
{
    if (_requires != null)
    {
        //如果有其他bundle依赖自己，其他bundle不存在 requires中的时候引用计数也要减一
        for (var i = 0; i < _requires.Count; i++)
        {
            var item = _requires[i];
            if (item != null)
                continue;
            Release();
            _requires.RemoveAt(i);
            i--;
        }
        if (_requires.Count == 0)
            _requires = null;
    }
    return refCount <= 0;
}
```

BundleAssetRequest.Unload==>自身引用计数以及依赖bundle引用计数都要减一

```
1+1 usages  1 override  hexinping *
internal override void Unload()
{
    //自身bundle引用计数减一
    if (BundleRequest != null)
    {
        BundleRequest.Release();
        BundleRequest = null;
    }

    if (children.Count > 0)
    {
        //bundle依赖项引用计数减一
        foreach (var item in children) item.Release();
        children.Clear();
    }

    asset = null;
}
```

BundleAssetRequestAsync.Unload

```
0+2 usages  hexinping *
internal override void Unload()
{
    _request = null;
    loadState = LoadState.Unload;
    base.Unload(); //调用父类的Unload,其实就是 BundleAssetRequest
}
```