1 Visualized

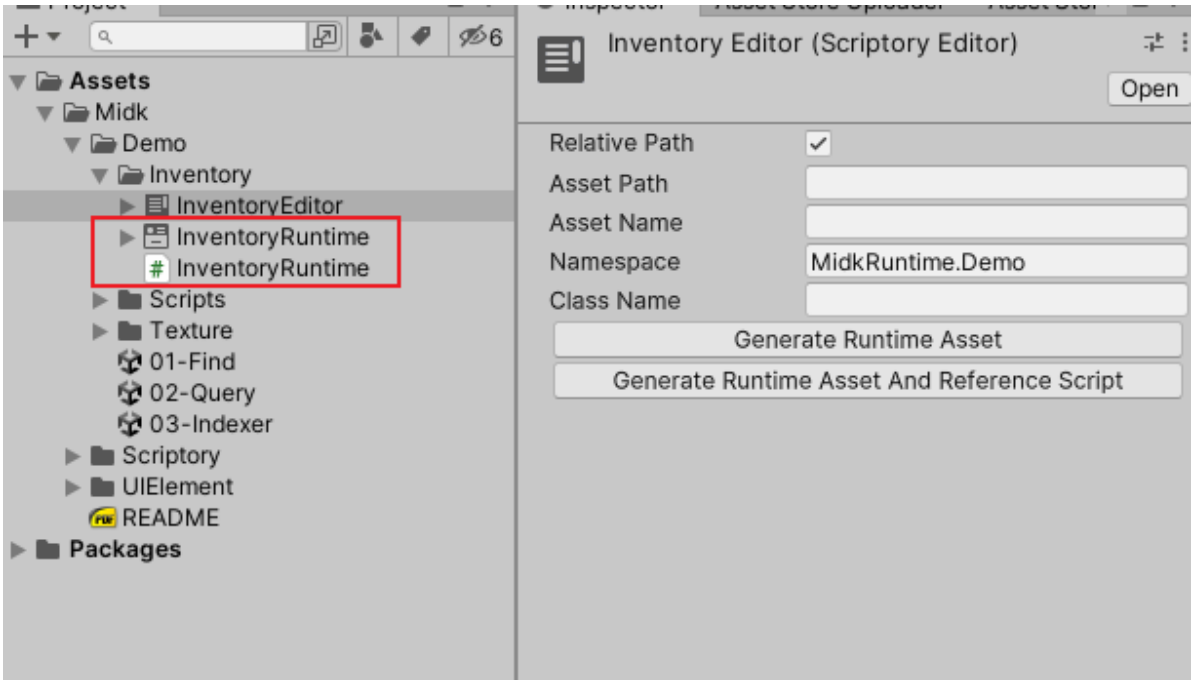
1.1 Asset

Project 窗口右键单击 > Create > Scriptory Editor 即可创建 ScriptoryEditor 资产，双击创建的 Asset 会打开 Scriptory Window 窗口。

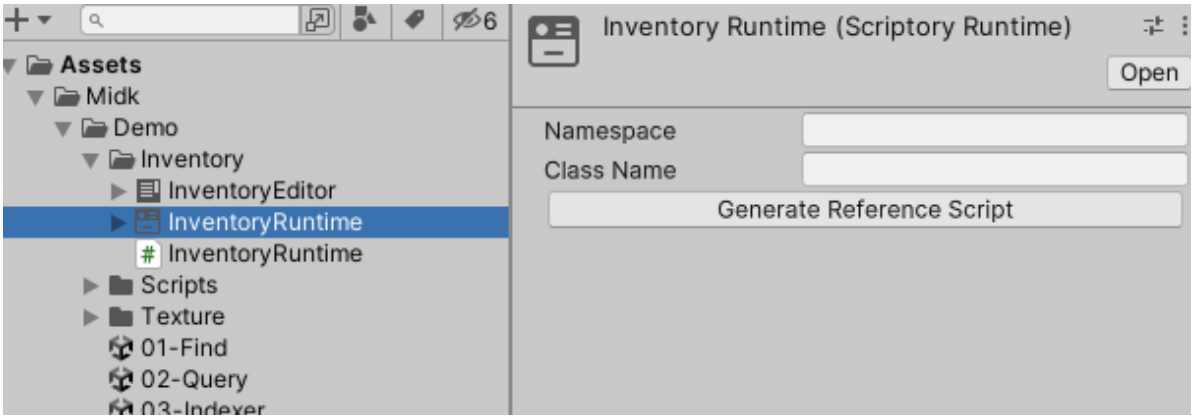
直接通过 ScriptoryEditor 资产来查找其中的子资产会有一定的效率问题，而且可能会错误的使用一些用于 Scriptory Window 的方法。为了避免这些问题，不直接使用 ScriptoryEditor 资产，而是通过 ScriptoryEditor 资产生成 ScriptoryRuntime 资产。

选中 ScriptoryEditor 资产，在 Inspector 窗口点击 Generate Runtime Asset 按钮会生成 ScriptoryRuntime 资产，点击 Generate Runtime Asset And Reference Script 按钮会生成 ScriptoryRuntime 资产和对应的子资产索引脚本。子资产索引脚本仅用于通过索引查找，手册末尾可查看 Demo 中生成的索引脚本。

- Relative Path：为 true 时，根路径为当前 ScriptoryEditor 资产所在的路径，为 false 时，根路径为 Assets 文件夹。
- Asset Path：根路径之后的路径，不包含名称。
- Asset Name：要创建的 ScriptoryRuntime 资产的名字，为空时会根据 ScriptoryEditor 资产的名字生成对应的名字。
- Namespace：要创建的子资产索引类所在的命名空间，会传递给创建的 ScriptoryRuntime 资产。
- Class Name：要创建的子资产索引脚本的名字，会传递给创建的 ScriptoryRuntime 资产。

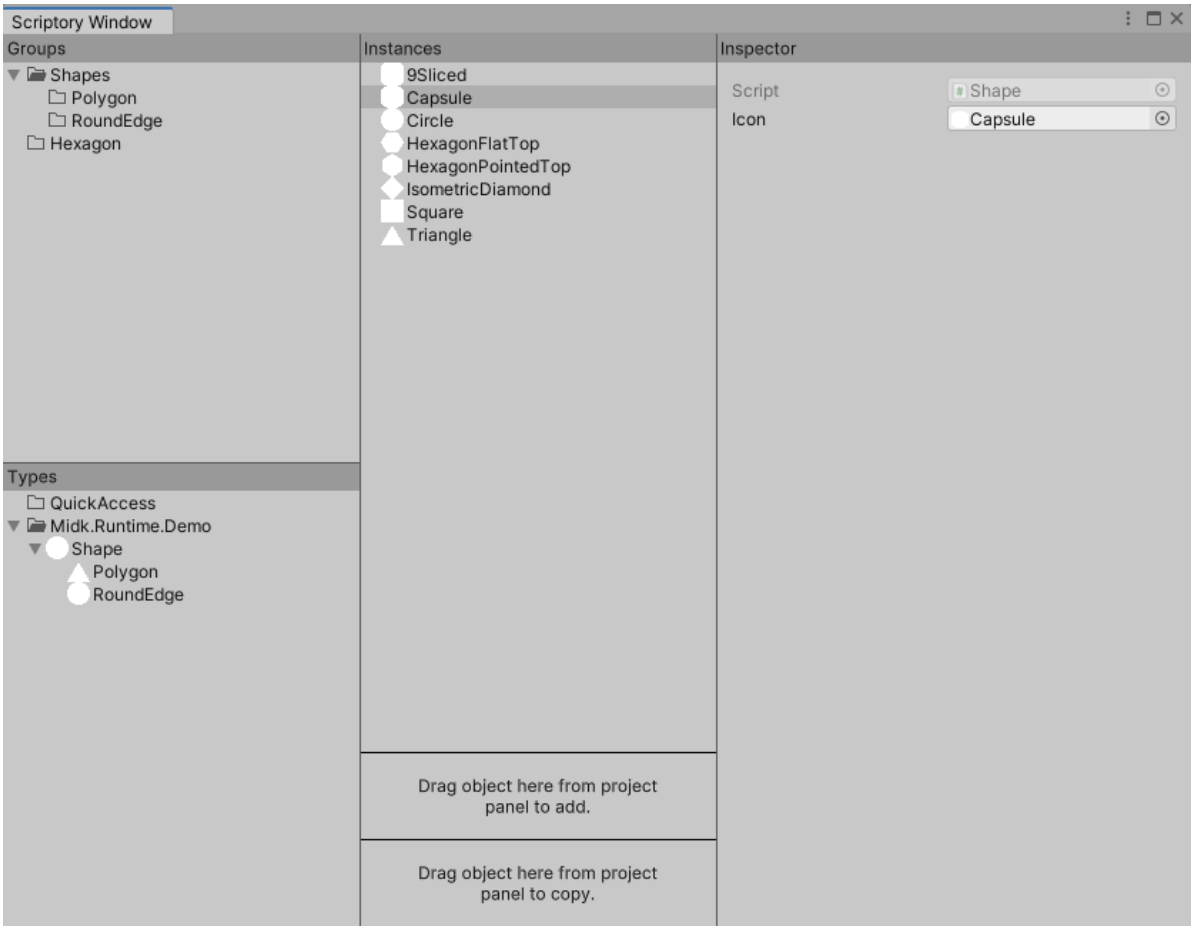


选中 ScriptoryRuntime 资产，在 Inspector 窗口点击 Generate Reference Script 也会生成索引脚本。



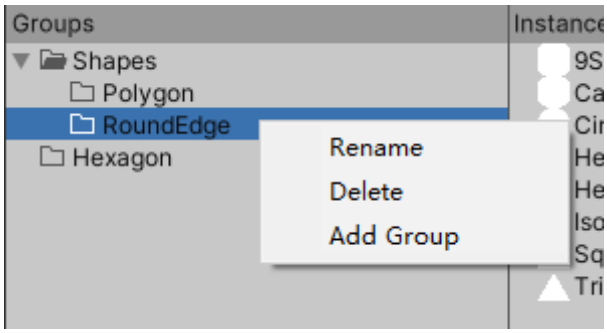
1.2 Window

双击 ScriptoryEditor 资产会打开 Scriptory Window 窗口。



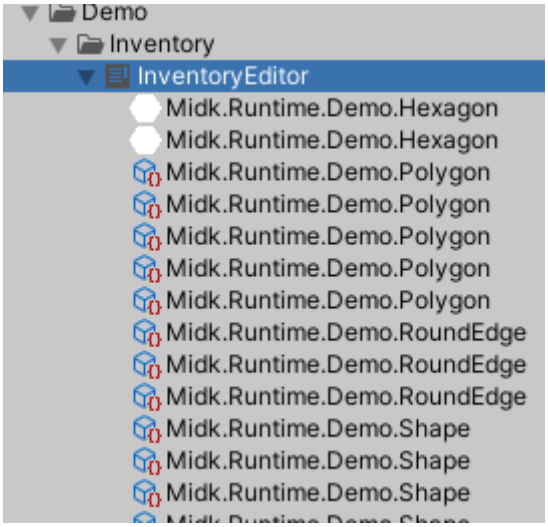
1. Groups

用于显示当前的文件夹结构，右键菜单可添加新文件夹、改名、删除文件夹，左键按住可拖拽文件夹来移动位置。

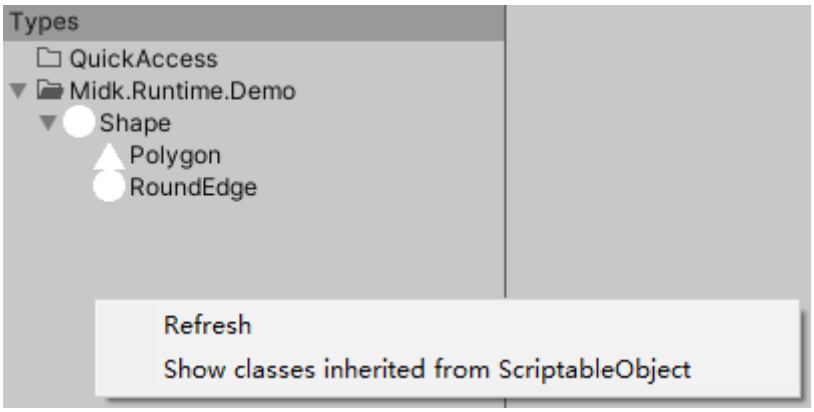


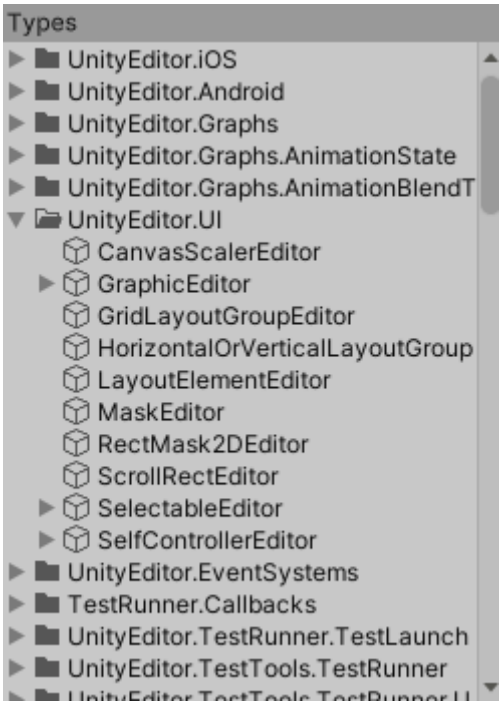
2. Types

默认显示继承自 ScriptoryRuntime.Object 的类型，选中类型拖拽到 Instances 窗口会在选中的文件夹创建此类型的实例，创建的实例同时会保存为对应 ScriptoryEditor 资产的子资产。

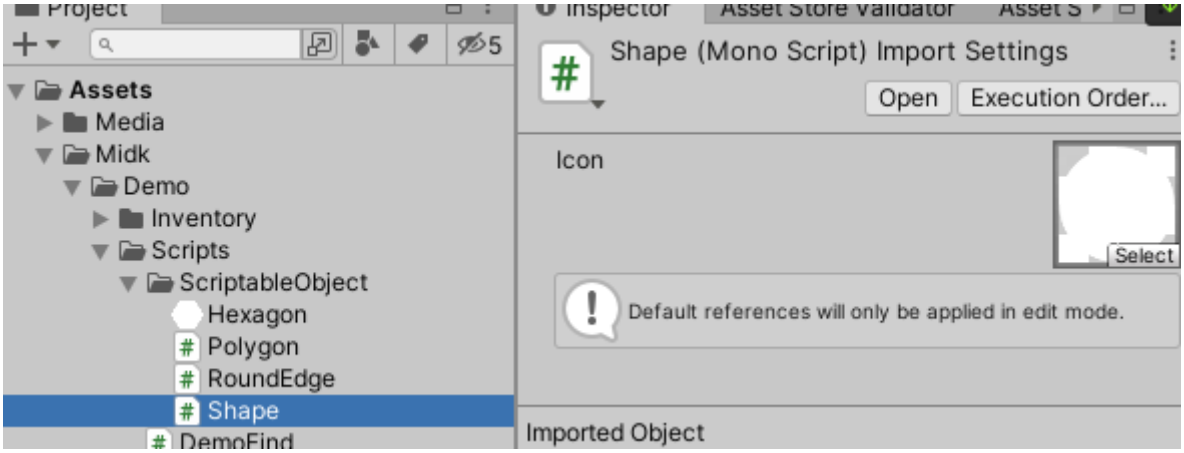


右键菜单可刷新此列表，或者显示继承自 ScriptableObject 的类型，显示的列表包含继承自 ScriptoryRuntime.Object 的类型，这些类型同样可以通过拖拽到 Instances 窗口来创建实例。





> 选中继承自 ScriptoryRuntime.Object 的类型所在的 C# 文件，在 Inspector 窗口可设置默认 Icon，此处设置的图标会在 Types 窗口显示。
>
> 不要随意实例化不认识的继承自 ScriptableObject 的脚本，这可能导致编辑器或插件的设置被异常修改而报错。

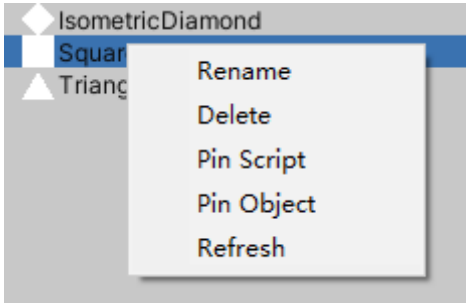


在类型上单击右键，弹出的菜单包含将此类型添加到 QuickAccess 文件夹的功能。

3. Instances

显示选中的文件夹所包含的实例，在 Groups 窗口点击空白处可选中根文件夹。

右键菜单可删除实例、重命名、刷新、选中实例类型的脚本、选中实例。

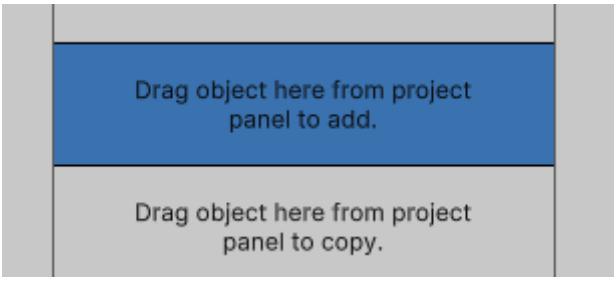


左键拖拽到 Groups 窗口上的文件夹以改变此实例的父文件夹，拖拽到 Groups 窗口的空白处会修改父文件夹为根文件夹。

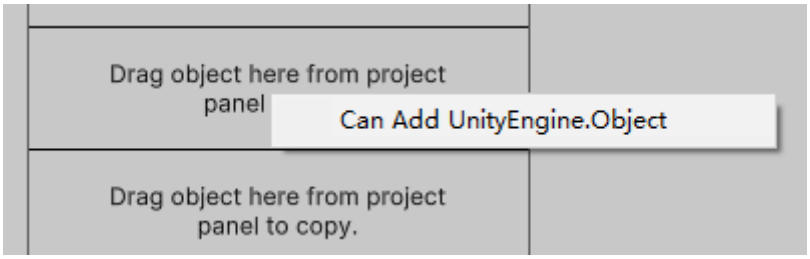
左键拖拽到能接收此实例类型的变量字段上可为变量赋值，因为 ScriptoryEditor 生成 ScriptoryRuntime 时会将所有子资产复制一份，并且 ScriptoryEditor 资产不会被打包到游戏中，所以不要在 Scriptory 资产外直接引用 Scriptory 的子资产。

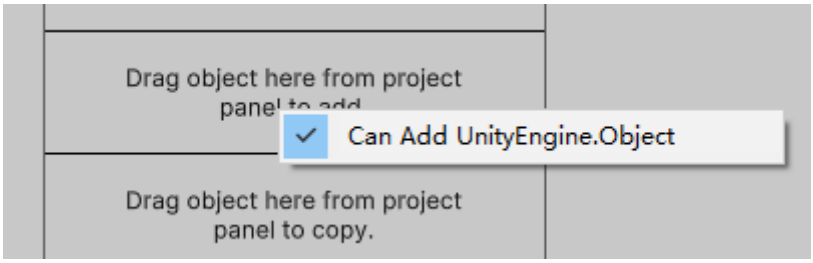
左键拖拽到此窗口里其它实例上可移动在此文件夹中的位置，鼠标弹起时会根据位置将拖动的实例移动到选中的实例之上或之下。

Instances 窗口最下方存在两个区域，可拖拽 Project 窗口中的资产到这两个区域来将其加入到 ScriptoryEditor 资产中。拖拽到 Add 区域会直接将拖拽的资产添加到 ScriptoryEditor 资产中，拖拽到 Copy 区域会将拖拽资产的复制添加到 ScriptoryEditor 资产中。这两个区域默认只接收继承自 ScriptableObject 的资产，可同时拖拽多个资产，拖拽的资产中包含可接收的资产时，对应的区域会变为蓝色，指针样式也会改变。不要通过此种方式添加包含子资产的资产。



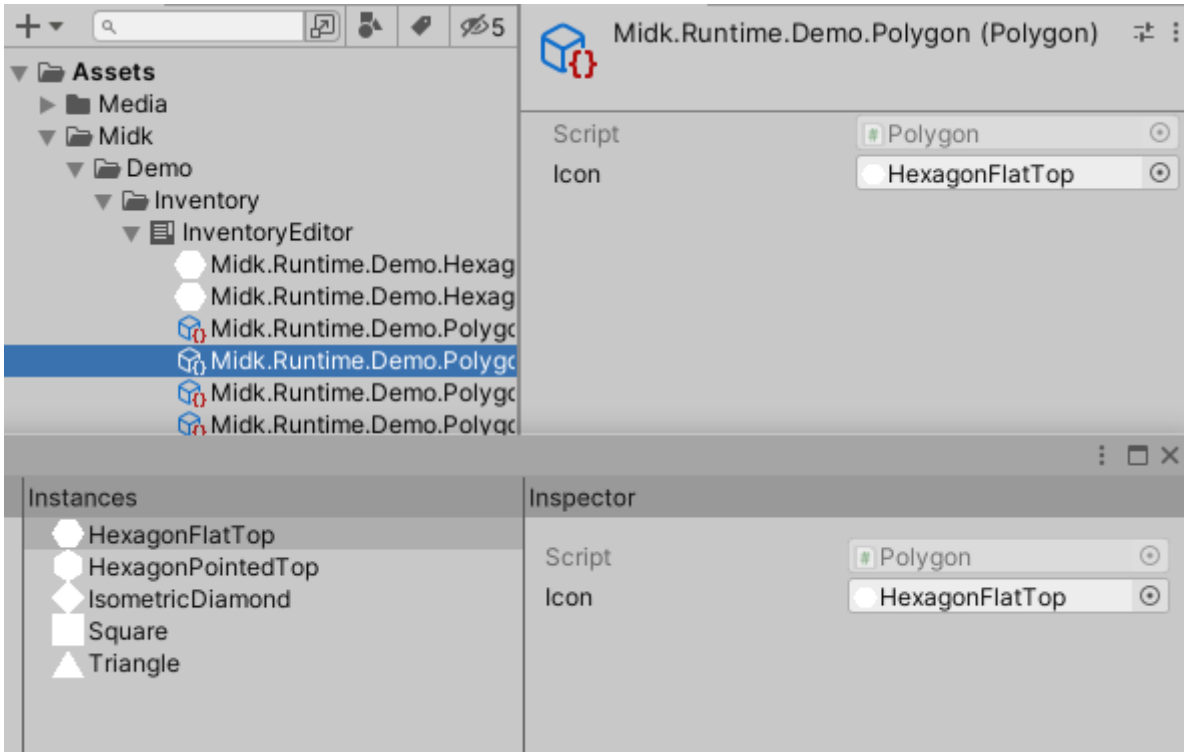
这两个区域的右键菜单的选项可设置接收的资产从继承自 ScriptableObject 的资产变为继承自 UnityEngine.Object 的资产，请谨慎使用此功能。



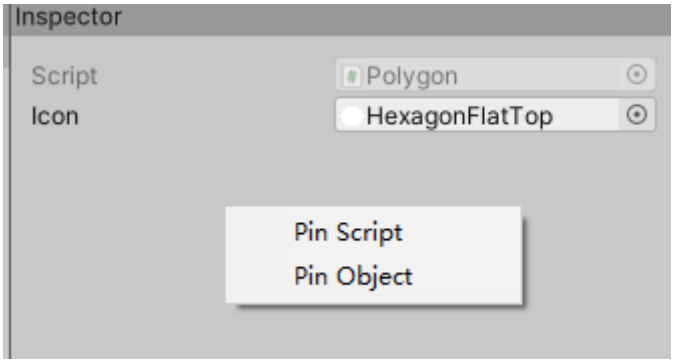


4. Inspector

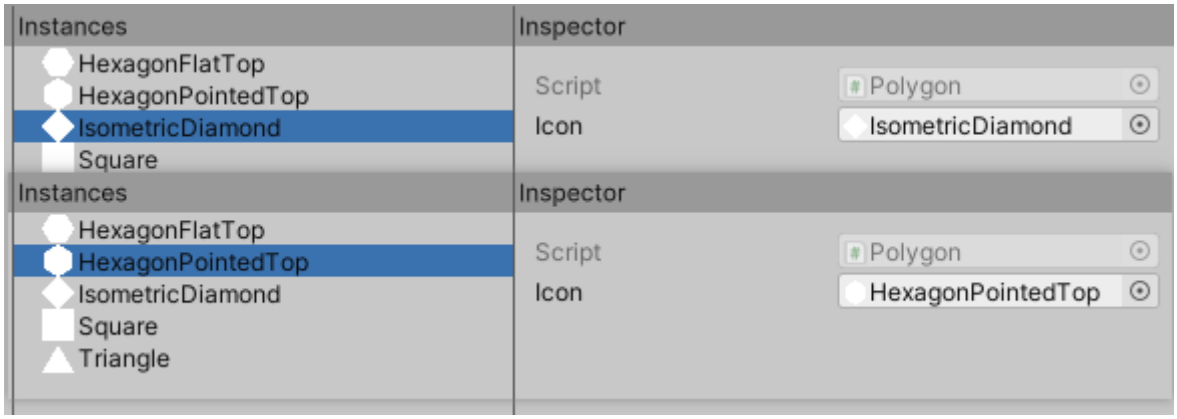
显示的内容与编辑器里 Inspector 窗口的内容一致，因为 Unity 内置格式的原因，带有 Foldout 元素的部分排版会有些许差异。



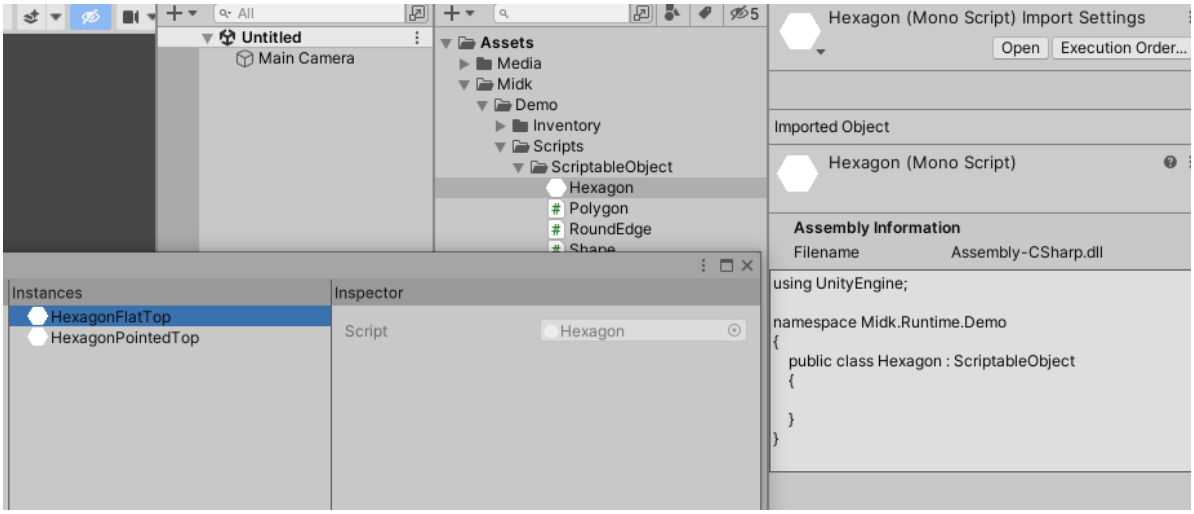
右键菜单可选中实例类型的脚本、选中实例。



继承自 ScriptoryRuntime.Object 的类型通过在此窗口修改 Icon 的值，来修改 Instances 窗口中显示的图标，修改完后需在 Instances 窗口右键刷新来刷新图标。

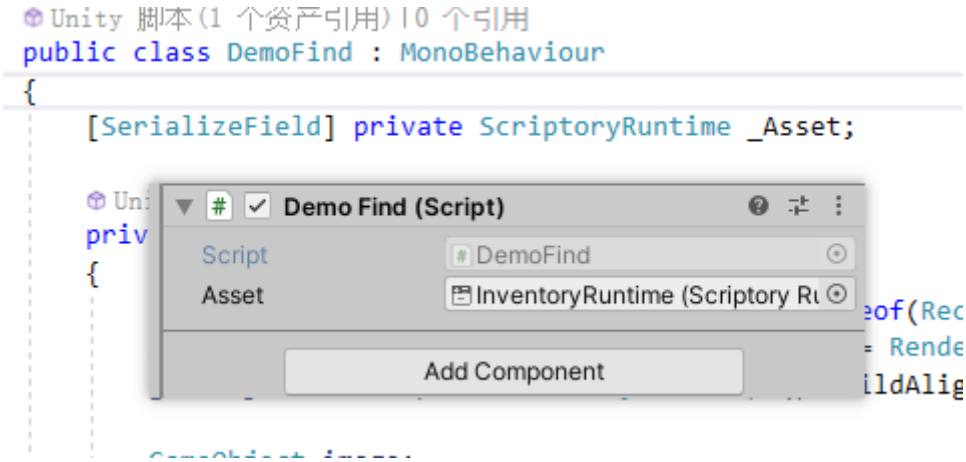


继承自 ScriptableObject 的类型在 Instances 窗口会显示 Select Icon。



2 Coded

在脚本中引用 ScriptoryRuntime 资产，然后通过此资产查找子资产。



可通过路径、类型、名称、索引等多种方式查找子资产。

1. 公共属性

ScriptoryRuntime：

```
public Folder Root; // 根文件夹
```

ScriptoryRuntime.Folder：

```
public int Index; // 文件夹索引，无用，可忽略
public string Name; // 文件夹名称
public List<Folder> ChildFolderList; // 子文件夹列表
public List<ScriptableObject> ChildObjectList; // 子资产，不包含子文件夹下的资产
```

2. 通过路径查找

通过路径查找文件夹：

- 从根文件夹查找：

  ```
  public ScriptoryRuntime.Folder ScriptoryRuntime.FindFolder(string path);
  public ScriptoryRuntime.Folder ScriptoryRuntime.Folder.FindFolder(string
  path);
  ```

- 从当前文件夹查找：

  ```
  public ScriptoryRuntime.Folder
  ScriptoryRuntime.Folder.FindFolderRelative(string path);
  ```

```
ScriptoryRuntime.Folder polygon = _Asset.FindFolder("Shapes.Polygon");

ScriptoryRuntime.Folder shapes = _Asset.FindFolder("Shapes");
ScriptoryRuntime.Folder polygon = shapes.FindFolderRelative("Polygon");
```

通过路径查找资产：

- 从根文件夹查找：

  ```
  public Object ScriptoryRuntime.FindObject(string path);
  public Object ScriptoryRuntime.Folder.FindObject(string path);
  ```

- 从当前文件夹查找：

  ```
  public Object ScriptoryRuntime.Folder.FindObjectRelative(string path);
  ```

```
Polygon triangle = _Asset.FindObject("Shapes.Polygon.Triangle") as Polygon;
Polygon triangle =
_Asset.FindFolder("Shapes").FindObject("Shapes.Polygon.Triangle") as Polygon;
Polygon triangle =
_Asset.FindFolder("Shapes").FindObjectRelative("Polygon.Triangle") as Polygon;
Polygon triangle =
_Asset.FindFolder("Shapes.Polygon").FindObjectRelative("Triangle") as Polygon;
```

3. 通过类型和名称查找

从根文件夹查找：

```
public List<T> ScriptoryRuntime.Query<T>(string name = null) where T :
UnityEngine.Object;
public T ScriptoryRuntime.Q<T>(string name = null) where T : UnityEngine.Object;
```

从当前文件夹查找：

```
public List<T> ScriptoryRuntime.Folder.Query<T>(string name = null) where T :
UnityEngine.Object;
public T ScriptoryRuntime.Folder.Q<T>(string name = null) where T :
UnityEngine.Object;
```

其中 Query 为查询所有符合条件的对象，Q 为查找第一个符合条件的对象，当 name 为空时仅通过类型查找。

```
List<Shape> shapes = _Asset.Query<Shape>();
List<Shape> polygon = _Asset.FindFolder("Shapes.Polygon").Query<Shape>();
Shape triangle = _Asset.Q<Shape>("Triangle");
```

4. 通过索引查找

查找文件夹：

```
ScriptoryRuntime.Folder shapes = _Asset[InventoryRuntime.Shapes.folder];
ScriptoryRuntime.Folder polygon =
_Asset[InventoryRuntime.Shapes.Polygon.folder];
```

查找子资产对象：

```
Shape triangle = _Asset[InventoryRuntime.Shapes.Polygon.Triangle] as Shape;
Shape circle = _Asset[InventoryRuntime.Shapes.Circle] as Shape;
```

> 此方法无查询消耗。

Demo 中生成的索引脚本：

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was auto-generated by Scriptory
//     version x.x
//     from Assets/Midk/Demo/Inventory/InventoryRuntime.asset
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Midk.Runtime.Demo
{
    public class InventoryRuntime
    {
        public const int folder = 4;
        public class Shapes
        {
            public const int folder = 2;
            public const long _9Sliced = 8589934592;
            public const long Capsule = 8589934593;
            public const long Circle = 8589934594;
```

```
            public const long HexagonFlatTop = 8589934595;
            public const long HexagonPointedTop = 8589934596;
            public const long IsometricDiamond = 8589934597;
            public const long Square = 8589934598;
            public const long Triangle = 8589934599;
            public class Polygon
            {
                public const int folder = 0;
                public const long HexagonFlatTop = 0;
                public const long HexagonPointedTop = 1;
                public const long IsometricDiamond = 2;
                public const long Square = 3;
                public const long Triangle = 4;
            }
            public class RoundEdge
            {
                public const int folder = 1;
                public const long _9Sliced = 4294967296;
                public const long Capsule = 4294967297;
                public const long Circle = 4294967298;
            }
        }
        public class Hexagon
        {
            public const int folder = 3;
            public const long HexagonFlatTop = 12884901888;
            public const long HexagonPointedTop = 12884901889;
        }
    }
}
```