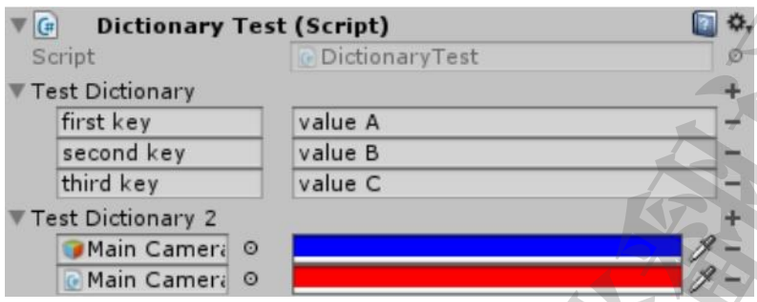
SerializableDictionary

一个用于 Unity 的可序列化字典类。

Unity 不能序列化标准字典。这意味着它们不会在检查器中显示或编辑，也不会在启动时实例化。一个经典的解决方法是将键和值存储在单独的数组中，并在启动时构造字典。

这个项目提供了一个通用的字典类和它的自定义属性抽屉来解决这个问题。



特性

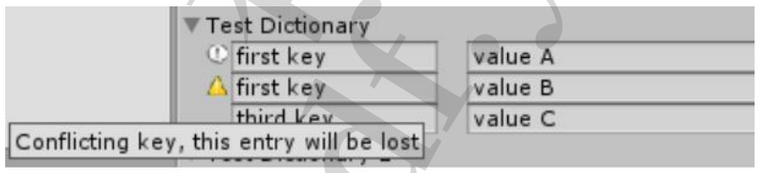
它继承自 Dictionary<TKey, TValue>

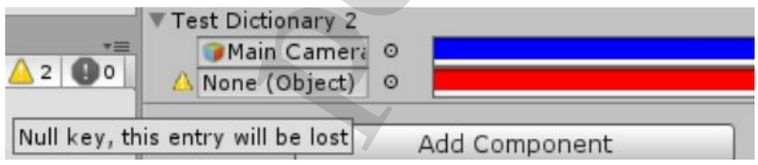
它实现了一个 CopyFrom(IDictionary<TKey, TValue>)方法来帮助从常规字典中赋值

你可以通过 unity 使用任何可序列化的类型作为键或值。

它可以在检查器中进行编辑，而不必实现自定义编辑器或属性抽屉。

检查器将处理无效的字典键，如重复或空键，并警告用户，如果键不固定，可能会发生数据丢失。





限制

必须为你想使用的每个<TKey, TValue>组合创建一个非泛型派生类。必须为这些类中的每个类声明 CustomPropertyDrawer。

不支持在检查器中使用 serializabledictionary 对脚本进行多次编辑。检查器将显示字典，但可能会发生数据丢失。

使用 LayerMask 作为键时，冲突键检测不起作用。LayerMask 值在 CustomPropertyDrawer 执行后改变。

列表或数组的字典必须使用 3 个参数 SerializableDictionary<TKey, TValue, TValueStorage>字典类和额外的 SerializableDictionary。Storage<TValue>类来保存值。详情请参见“Dictionary of lists or arrays”一节。

使用

简单的字典示例

创建一个类型为<string, string>的可序列化字典:创建一个 SerializableDictionary 子类

```
(序列化)
公共类 StringStringDictionary: SerializableDictionary<string, string> {}
```

在编辑器文件夹中创建 SerializableDictionaryPropertyDrawer 子类或使用现有的。
在这个类中添加一个 CustomPropertyDrawer 属性。

```
[CustomPropertyDrawer(typeof(StringStringDictionary))]
公 共 类 AnySerializableDictionaryPropertyDrawer:
SerializableDictionaryPropertyDrawer{}
```

所有 SerializableDictionary 子类的属性都可以添加到同一个 SerializableDictionaryPropertyDrawer 子类中。
在你的脚本中使用 StringStringDictionary 作为普通的 IDictionary<string, string>类型

列表字典示例

创建一个类型为<string, List<Color>>的可序列化字典:创建一个 SerializableDictionary。Storage 子类来保存列表

```
(序列化)
公共类 ColorListStorage: SerializableDictionary。存储列表<<颜色>> {}
```

在编辑器文件夹中创建一个 ListStoragePropertyDrawer 子类，或者使用已有的。在这个类中添加一个 CustomPropertyDrawer 属性。

```
[CustomPropertyDrawer typeof (ColorListStorage))]
公共类 AnySerializableDictionaryStoragePropertyDrawer:
```

```
SerializableDictionaryStoragePropertyDrawer {}
```

使用前面的子类创建一个 SerializableDictionary 子类

```
(序列化)
公共类 StringColorListDictionary:SerializableDictionary<string
,List<Color>, ColorListStorage> {}
```

在编辑器文件夹中创建 SerializableDictionaryPropertyDrawer 子类，或者使用已有的。

在这个类中添加一个 CustomPropertyDrawer 属性。

```
[CustomPropertyDrawer(typeof(StringColorListDictionary))]
公共类 AnySerializableDictionaryPropertyDrawer:
 SerializableDictionaryPropertyDrawer{}
```

使用 StringColorListDictionary 在你的脚本作为一个正常的 IDictionary<string, List<Color>> 类型

细节

由于 Unity 不能直接序列化泛型类型，为每个 SerializedDictionary 专门化创建一个派生类。

```
(序列化)
公共类 StringStringDictionary: SerializableDictionary<string, string> { }
[Serializa
```

你可以使用自己的 serializable 类。

```
(序列化)
公共类 MyClass
{
    公共 int i;
    公共字符串 str;
}

(序列化)
```

通过将 CustomPropertyDrawer 属性添加到 SerializableDictionaryPropertyDrawer 派生类之一来声明这些新类型的自定义属性抽屉。

```
[CustomPropertyDrawer(typeof(StringStringDictionary))]
[CustomPropertyDrawer(typeof(MyScriptColorDictionary))
]
[CustomPropertyDrawer(typeof(StringMyClassDictionary))
]
```

建议在单独的文件中创建一个派生类，并将属性添加到该类，而不是修改原始 SerializableDictionaryPropertyDrawer 类。你可以为所有 SerializableDictionary 专门化使用同一个类，没有必要为每个专门化创建一个新的类。

将字典添加到你的脚本中，并通过属性直接访问它们。字典可以通过类型为 IDictionary<TKey, TValue> 的属性访问，以便更好地封装。

```
public StringStringDictionary m_myDictionary1;

(SerializeField)
MyScriptColorDictionary m_myDictionary2;
publicIDictionary<MyScript, Color> MyDictionary2
{
    get {return m_myDictionary2;}
    set{m_myDictionary2.;CopyFrom(价值);}
public StringMyClassDictionary m_myDictionary3;
```

CopyFrom(value)方法清除 m_myDictionary2 字典，并向其添加值字典的每个内容，有效地将 value 复制到 m_myDictionary2。

SerializableDictionary 有一个从 IDictionary<TKey, TValue>复制构造函数。由于父类的构造函数不能直接使用，你必须给你的派生类添加一个复制构造函数，调用基构造函数才能使用它。

```
(序列化)
公共类 StringColorDictionary: SerializableDictionary<string, Color> {

    publicStringColorDictionary(IDictionary<string, Color> dict):base(dict){}
`
```

列表或数组的字典

因为 unity 不能序列化列表数组或数组数组，在脚本中使用 SerializableDictionary<TKey, TValue[]>或 SerializableDictionary<TKey, List<TValue>>将无法正常工作。字典不会显示在检查器中，值也不会被保存。

有必要创建一个包含列表或数组的中间类。然后这个类可以被包含在数组中，并被 Unity 序列化。

创建一个继承自 SerializableDictionary.Storage<List<TValue>的类。这个存储类将只包含 List<TValue>数据字段。

```
(序列化)
公共类 ColorListStorage:SerializableDictionary。 存储列表<<颜色> > {}
```

如果你直接与 SerializableDictionary 一起使用这个存储类，你将不得不通过 storage 类的.data 字段访问列表或数组，因为你的字典将继承 dictionary <TKey, storage <TValue>>而不是 dictionary <TKey, list <TValue>>。这远非理想。

```
//一个颜色列表字典的非最佳示例
(序列化)
公共类 ColorListStorage: SerializableDictionary。 Storage<List<Color>> {}
[Seriali
able]

public StringColorListDictionary m_colorStringListDict;

//你必须通过 ColorListStorage 的.data 字段访问颜色列表

List<Color> colorList = m_colorStringListDict[key].data;
```

要直接访问列表，使用特殊的 3 个参数 SerializableDictionary<TKey, TValue, TValueStorage>类，其中 TValueStorage 是之前创建的类。

```
(序列化)
公共类 ColorListStorage: SerializableDictionary。 Storage<List<Color>> {}
[Seriali
able]

m_colorStringListDict;

//你现在可以直接访问颜色列表 list
< color > colorList = m_colorStringListDict[key];
```

你必须声明字典的属性抽屉，就像经典的序列化字典一样。

```
[CustomPropertyDrawer(typeof(StringColorListDictionary))]
公共类 AnySerializableDictionaryPropertyDrawer:
```

```
SerializableDictionaryPropertyDrawer {}
```

你还必须为存储类声明一个属性抽屉，以便在检查器中隐藏中间的.data 字段。

```
[CustomPropertyDrawer typeof (ColorListStorage)))

公共类 AnySerializableDictionaryStoragePropertyDrawer:
SerializableDictionaryStoragePropertyDrawer {}
```