表4.4 检查null

|  |  |
| --- | --- |
| **说明** | **示例** |
| **is null 操作符模式匹配**  is 运算符提供了多种检查 null 的方法。从 C# 7.0 开始，可以使用 is null 表达式来检查一个值是否为 null。这是检查值是否为null的一种非常简单明了的方式。 | // 1.  if (uriString is null)  {  Console.WriteLine(  "Uri为null");  } |
| **is not null操作符模式匹配**  类似地，C# 9.0增加了对is not null的支持。如果要检查是否不为空，这就是首选方式。 | // 2.  if (uriString is not null)  {  Console.WriteLine(  "Uri非空");  } |
| **相等/不相等**  相等和不相等操作符适用于所有版本的 C#。  此外，用这种方式检查null具有较好的可读性。  这种方式的唯一缺点是可能要求重写相等/不相等运算符，从而可能引入轻微的性能损失。 | // 3.  if (uriString == null)  {  Console.WriteLine(  "Uri为null");  }  if (uriString != null)  {  Console.WriteLine(  $"Uri是: {uriString}");  } |
| **is object**  is object 是C# 1.0就有的一个功能，用于判断操作数是否不为 null，等价于is not null，但明显不如后者清晰。is object比下面的is {}表达式更好用，因为当操作数是非空的值类型时，它会发出警告。这很合理，既然操作数不能为null，检查null的意义何在呢？ | // 4.  int number = 0;  if ((uriString is object)  // Warning CS0183:给定表达式  // 始终不为null  && (number is object)  )  {  Console.WriteLine(  $"Uri是: {uriString}");  } |
| **is { }操作符模式匹配**  C# 8.0 新增了属性模式匹配表达式<操作数> is { }，它提供了与is object几乎相同的功能。除了可读性较差，它还有一个明显的缺点，也就是为不可空的值类型表达式使用is {}，它是不会发出警告的，但is object会。而且，对于不可空的值类型，最好是发出警告，因为检查不可空的值类型是否为 null 是没有意义的。因此，建议优先使用is object而不是is { }。 | // 5.  if (uriString is { })  {  Console.WriteLine(  $"Uri是: {uriString}");  } |
| **ReferenceEquals()**  object.ReferenceEquals()用于执行引用相等性检查。虽然对于一个如此简单操作而言，这个名字显得过长，但它适用于所有版本的 C# ，并具有不允许重写的优点。所以，它始终做到了“名符其实”。 | // 6.  if (ReferenceEquals(  uriString, null))  {  Console.WriteLine(  "Uri为null");  } |