

# 变量

下面是创建并初始化变量的例子：

```
var name = 'Bob';
```

dart

变量会保存引用。`name` 变量包含一个值为 "Bob" 的 `String` 对象的引用。

变量 `name` 的类型被推断为 `String`，但你可以通过指定类型来更改它。如果一个对象不受限于单一类型，可以指定为 `Object` 类型（或在必要时使用 `dynamic`）。

```
Object name = 'Bob';
```

dart

另一种选择是显式声明将要被推断的类型：

```
String name = 'Bob';
```

dart

## 提示

文章遵循 [代码风格指南](#)，使用的是 `var` 声明局部变量，而不是使用准确的类型。

## 空安全

Dart 语言要求以健全的空安全方式编写代码。

空安全能够防止意外访问 `null` 的变量而导致的错误。这样的错误也被称为空解引用错误。访问一个求值为 `null` 的expressions的属性或调用方法时，会发生空解引用错误。但是对于 `toString()` 方法和 `hashCode` 属性，空安全会体现出例外情况。Dart 编译器可以在空安全的基础上在编译期检测到这些潜在的错误。

例如，假设你想要查找 `int` 变量 `i` 的绝对值。如果 `i` 是 `null`，调用 `i.abs()` 会导致空解引用错误。在其他语言中，尝试这样做可能会导致运行时错误，但是 Dart 的编译器禁止这些操作。所以 Dart 应用程序不会引发运行时错误。

空安全引入了三个关键更改：

1. 当你为变量、参数或另一个相关组件指定类型时，可以控制该类型是否允许 `null`。要让一个变量可以为空，你可以在类型声明的末尾添加 `?`。

```
String? name // Nullable type. Can be `null` or string.

String name  // Non-nullable type. Cannot be `null` but can be string.
```

dart

2. 你必须在使用变量之前对其进行初始化。可空变量是默认初始化为 `null` 的。Dart 不会为非可空类型设置初始值，它强制要求你设置初始值。Dart 不允许你观察未初始化的变量。这可以防止你在接收者类型可以为 `null` 但 `null` 不支持的相关方法或属性的情况下使用它。
3. 你不能在可空类型的表达式上访问属性或调用方法。同样的例外情况适用于 `null` 支持的属性或方法，例如 `hashCode` 或 `toString()`。

空安全将潜在的 **运行时错误** 转变为 **编辑时** 分析错误。当非空变量处于以下任一状态时，空安全会识别该变量：

- 未使用非空值进行初始化。
- 赋值为 `null`。


此检查允许你在部署应用程序 **之前** 修复这些错误。

# 默认值

具有可空类型的未初始化变量的初始值为 `null`。即使是具有数值类型的变量，初始值也为空，因为数字（就像 Dart 中的其他所有东西一样）都是对象。

```
int? lineCount;
assert(lineCount == null);
```

dart

 提示

当你在生产环境中运行代码时，`assert()` 调用会被忽略。另外在开发过程中，`assert(condition)` 如果其 **条件** 为 `false`，会抛出一个异常。有关详细信息，请参阅 [断言](#)。

对于空安全，你必须在`使用非空变量之前初始化它们的值`：

```
int lineCount = 0;
```

dart

你不必在声明变量时初始化变量，但在使用之前需要为其赋值。例如以下代码是合法的，因为 Dart 可以检测到 `lineCount` 在传递给 `print()` 时是非空的：

```
int lineCount;

if (weLikeToCount) {
  lineCount = countLines();
} else {
  lineCount = 0;
}

print(lineCount);
```

dart

顶级变量和类变量是延迟初始化的，它们会在第一次被使用时再初始化。

# 延迟初始化变量

`late` 修饰符有两种用法：

- 声明一个非空变量，但不在声明时初始化。
- 延迟初始化一个变量。


通常 Dart 的语义分析可以检测非空变量在使用之前是否被赋值，但有时会分析失败。常见的两种情况是在分析顶级变量和实例变量时，Dart 通常无法确定它们是否已设值，因此不会尝试分析。

如果你确定变量在使用之前已设置，但 Dart 推断错误的话，可以将变量标记为 `late` 来解决这个问题：

```
late String description;

void main() {
  description = 'Feijoadada!';
  print(description);
}
```

dart

 Notice

如果你没有初始化一个 `late` 变量，那么当变量被使用时会发生运行时错误。

当一个 `late` 修饰的变量在声明时就指定了初始化方法，那么内容会在第一次使用变量时运行初始化。这种延迟初始化在以下情况很方便：

- （Dart 推断）可能不需要该变量，并且初始化它的开销很高。
- 你正在初始化一个实例变量，它的初始化方法需要调用 `this`。

在下面的例子中，如果 `temperature` 变量从未被使用，则 `readThermometer()` 这个开销较大的函数也永远不会被调用：

```
dart
// This is the program's only call to readThermometer().
late String temperature = readThermometer(); // Lazily initialized.
```

## 终值 (final) 和常量 (const)

如果你不打算更改一个变量，可以使用 `final` 或 `const` 修饰它，而不是使用 `var` 或作为类型附加。一个 `final` 变量只能设置一次，`const` 变量是编译时常量。（`const` 常量隐式包含了 `final`。）

❗ 提示

[实例变量](#) 可以是 `final` 但不能是 `const`。

下面是创建和设置 `final` 变量的示例：


```
dart
final name = 'Bob'; // Without a type annotation
final String nickname = 'Bobby';
```

你不能修改 `final` 变量的值：

```
dart
X static analysis: failure
name = 'Alice'; // Error: a final variable can only be set once.
```

请使用 `const` 修饰 **编译时常量** 的变量。如果 `const` 变量位于类级别，请将其标记为 `static const`（静态常量）。在声明变量的位置，将其值设置为编译时常量，比如数字、字符串、`const` 常量或在常量数字上进行的算术运算的结果：

```
dart
const bar = 1000000; // Unit of pressure (dynes/cm2)
const double atm = 1.01325 * bar; // Standard atmosphere
```

`const` 关键字不仅仅可用于声明常量，你还可以使用它来  创建常量 **值(values)**，以及声明 **创建(create)** 常量值的构造函数。任何变量都可以拥有常量值。

```
dart
var foo = const [];
final bar = const [];
const baz = []; // Equivalent to `const []`
```

你可以省略以 `const` 声明中的值的 `const` 修饰，就像上面的 `baz` 一样。更多详细信息请参考 [不要重复使用常量](#)。

如果一个变量没被声明为 `final` 或者 `const`，那么，即使它的值是 `const`，你仍然可以修改这个变量：

```
dart
foo = [1, 2, 3]; // Was const []
```

你不能修改 `const` 变量的值：

```
dart
X static analysis: failure
baz = [42]; // Error: Constant variables can't be assigned a value.
```

你可以在定义常量时使用 [类型检查和转换](#) (`is` 和 `as`)、[集合中的 if](#) 和 [展开操作符](#) (`...` 和 `...?`)：

dart

```
const Object i = 3; // Where i is a const Object with an int value...
const list = [i as int]; // Use a typecast.
const map = {if (i is int) i: 'int'}; // Use is and collection if.
const set = {if (list is List<int>) ...list}; // ...and a spread.
```

❗ 提示

虽然 `final` 对象不能被修改，但它的字段可能可以被更改。相比之下，`const` 对象及其字段不能被更改：它们是 不可变的。

虽然 `final` 对象不能被修改，但它的字段可能可以被更改。相比之下，`const` 对象及其字段不能被更改：它们是 **不可变的**。

有关使用 `const` 创建常量值的更多信息，请参见 [Lists](#)、[Maps](#) 和 [Classes](#)。