This release brings enhancements to type promotion, interop capabilities, DevTools, and more.

Q

Operators >

Contents

< Basics

Variables

Null safety

Default value Late variables Final and const

var name = 'Bob';

Variables store references. The variable called name contains a reference to a String object with a value of "Bob". The type of the name variable is inferred to be String, but you can change that type by

dynamic if necessary). Object name = 'Bob';

specifying it. If an object isn't restricted to a single type, specify the Object type (or

Another option is to explicitly declare the type that would be inferred: String name = 'Bob';

i Note: This page follows the style guide recommendation of using var, rather than type annotations, for local variables.

Null safety

runtime error, but Dart's compiler prohibits these actions. Therefore, Dart apps can't cause runtime errors. Null safety introduces three key changes: 1. When you specify a type for a variable, parameter, or another relevant component, you

a property or call a method on an expression that evaluates to null. An exception to this rule

String? name // Nullable type. Can be `null` or string. // Non-nullable type. Cannot be `null` but can be string. String name 2. You must initialize variables before using them. Nullable variables default to null, so

3. You can't access properties or call methods on an expression with a nullable type. The

Sound null safety changes potential runtime errors into edit-time analysis errors. Null safety

same exception applies where it's a property or method that null supports like

This check allows you to fix these errors before deploying your app.

hashCode or toString().

flags a non-null variable when it has been either:

• Not initialized with a non-null value.

Default value Uninitialized variables that have a nullable type have an initial value of null. Even variables with numeric types are initially null, because numbers—like everything else in Dart—are objects.

details, check out Assert.

lineCount = countLines();

int lineCount = 0;

assert(lineCount == null);

int? lineCount;

With null safety, you must initialize the values of non-nullable variables before you use them:

You don't have to initialize a local variable where it's declared, but you do need to assign it a value before it's used. For example, the following code is valid because Dart can detect that lineCount is non-null by the time it's passed to print():

print(lineCount);

Often Dart's control flow analysis can detect when a non-nullable variable is set to a non-null

variables and instance variables: Dart often can't determine whether they're set, so it doesn't

value before it's used, but sometimes analysis fails. Two common cases are top-level

超起是可知

If you're sure that a variable is set before it's used, but Dart disagrees, you can fix the error by marking the variable as late:

void main() {

String description;

description = 'Feijoada!';

print(description);

When you mark a variable as late but initialize it at its declaration, then the initializer runs the first time the variable is used. This lazy initialization is handy in a couple of cases: The variable might not be needed, and initializing it is costly.
You're initializing an instance variable, and its initializer needs access to this.

A If you fail to initialize a late variable, a runtime error occurs when the variable is

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

If you never intend to change a variable, use final or const, either instead of var or in

addition to a type. A final variable can be set only once; a const variable is a compile-time

Note: Instance variables can be final but not const.

OLATER TOPOLE THAT I SUPPRIME THE STATE STATES

Use const for variables that you want to be compile-time constants. If the const variable is at the class level, mark it static const. Where you declare the variable, set the value to a compile-time constant such as a number or string literal, a const variable, or the result of an arithmetic operation on constant numbers:

You can omit const from the initializing expression of a const declaration, like for baz

You can change the value of a non-final, non-const variable, even if it used to have a const

const bar = 1000000; // Unit of pressure (dynes/cm2)

const double atm = 1.01325 * bar; // Standard atmosphere

name = 'Alice'; // Error: a final variable can only be set once.

foo = [1, 2, 3]; // Was const [] You can't change the value of a const variable:

x static analysis: error/warning

spread operators (... and ...?):

const baz = []; // Equivalent to `const []`

above. For details, see DON'T use const redundantly.

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.

• Note: Although a final object cannot be modified, its fields can be changed. In

comparison, a const object and its fields cannot be changed: they're immutable.

const set = {if (list is List<int>) ...list}; // ...and a spread.

baz = [42]; // Error: Constant variables can't be assigned a value.

You can define constants that use type checks and casts (is and as), collection if, and

For more information on using const to create constant values, see Lists, Maps, and Classes. < Basics

Operators >

Here's an example of creating a variable and initializing it:

The Dart language enforces sound null safety. Null safety prevents an error that results from unintentional access of variables set to null. The error is called a null dereference error. A null dereference error occurs when you access

is when null supports the property or method, like toString() or hashCode. With null safety, the Dart compiler detects these potential errors at compile time. For example, say you want to find the absolute value of an int variable i. If i is null, calling i.abs() causes a null dereference error. In other languages, trying this could lead to a

can control whether the type allows null. To enable nullability, you add a ? to the end of the type declaration.

they are initialized by default. Dart doesn't set initial values to non-nullable types. It forces you to set an initial value. Dart doesn't allow you to observe an uninitialized variable. This prevents you from accessing properties or calling methods where the receiver's type can be null but null doesn't support the method or property used.

• Assigned a null value.

i Note: Production code ignores the assert() call. During development, on the other hand, assert (condition) throws an exception if condition is false. For

int lineCount; if (weLikeToCount) {

} else {

lineCount = 0;

Top-level and class variables are lazily initialized; the initialization code runs the first time the variable is used. Late variables The late modifier has two use cases: • Declaring a non-nullable variable that's initialized after its declaration. • Lazily initializing a variable.

try.

}

used.

Final and const

constant. (Const variables are implicitly final.)

final String nickname = 'Bobby';

x static analysis: error/warning

var foo = const [];

final bar = const [];

Here's an example of creating and setting a final variable:

final name = 'Bob'; // Without a type annotation

In the following example, if the temperature variable is never used, then the expensive readThermometer() function is never called:

You can't change the value of a final variable:

The const keyword isn't just for declaring constant variables. You can also use it to create constant values, as well as to declare constructors that create constant values. Any variable can have a constant value.

value:

Dart

O X Except as otherwise noted, this site is licensed under a Creative Commons Attribution 4.0 International License, and code samples are licensed under the 3-Clause BSD License.

Privacy Terms Security