Introduction to the Dart Null Safety

Null means no value or absence of value. Dart has supported sound null safety since version 2.12.

In null safety, [variables](https://www.darttutorial.org/dart-tutorial/dart-variables/) cannot be null unless you explicitly specify that they can.

The following program declares a string variable called message, initializes its value to 'Hello', and displays the length property:

void main() {

String message = 'Hello';

print(message.length);

}

Code language: Dart (dart)

Before Dart 2.12, you can assign null to the message variable and access the length property. However, it’ll cause a runtime error:

void main() {

String messge = null;

print(message.length); *// runtime error*

}

Code language: Dart (dart)

With null safety, types in the code are non-nullable by default. If you attempt to assign null to a variable, the code editor will issue an error. In other words, null safety turns the runtime errors into edit-time errors. This makes your code more robust.

To specify that a variable can be null, you add a question mark (?) to the type in variable declaration. For example:

void main() {

String? message = 'Hello';

print(message);

message = null; *// OK*

}

Code language: Dart (dart)

In this example, the message variable is a string variable that can accept either null or a non-null value.

Nullable types

A nullable type contains null in addition to its own values of the type.

To mark an existing type nullable, you place a question mark after the type. For example:

* int? – a nullable integer such as 1, 2, and null.
* double? – a nullable double such as 3.14, 2.5, and null.
* bool? – nullable boolean such as true, false, and null.
* String? – a nullable string such as ‘Hello’, ‘Bye’, and null.
* Point? a nullable user-defined class Point. For example, point(10,20) and null.

In Dart, every non-nullable type has a corresponding nullable type. By adding the ?, you specify when you want to allow null and when you don’t.

Working with nullable types

Before null safety, it’s easy to forget to add the code that handles null. However, with null safety, Dart makes it impossible to forget. Because you really cannot do much with null unless you deal with the null possibility.

For example, Dart will not allow you to run the following code:

void main() {

String? message;

print(message.length);

}

Code language: Dart (dart)

It issues a compile-time error.

Dart has a tool called Dart analyzer that is smart enough to tell if a nullable variable contains null or not.

Type promotion

Type promotion allows you to assign a value to a nullable variable without requiring any extra work. For example:

void main() {

String? message;

message = 'Hello';

print(message.length);

}

Code language: Dart (dart)

In this example, the variable message is a nullable string type. However, Dart can see that the message is not null because we assign a value to it before accessing the length property.

Therefore, Dart implicitly promotes the type of the message variable from String? to String automatically.

Flow analysis

Besides type promotion, Dart uses a sophisticated flow analysis to check every possible case the code would take. And if none of these cases come up with the possibility of being null, it promotes the variable to a non-nullable type using type promotion:

bool isEven(int? x) {

if (x == null) {

return false;

}

return x.isEven;

}

Code language: Dart (dart)

In this example, once the last line of the function is reached, the parameter x cannot be null. Therefore, Dart promotes the x to int instead of using the int? type.

Summary

* In Null safety, variables cannot be null unless you explicitly specify that they can.
* Null safety turns possible null-related runtime errors into compile-time errors.
* Every non-nullable type has a corresponding nullable type.
* Add a question mark (?) to a non-nullable type to turn it into a nullable type.
* Dart uses a sophisticated flow analysis to automatically promote a type from nullable to non-nullable when necessary.

Introduction to Dart null-aware operators

To deal with null values, Dart uses flow analysis and type promotion. In addition, it provides you with various null-aware operators:

| **Operator** | **Meaning** |
| --- | --- |
| ?? | If-null operator |
| ??= | Null-aware assignment operator |
| ?. | Null-aware access & method invocation operator |
| ! | Null assertion operator |
| ?.. | Null-aware cascade operator |
| ?[] | Null-aware index operator |
| ...? | Null-aware spread operator |

The if-null operator (??)

The following example uses an [if](https://www.darttutorial.org/dart-tutorial/dart-if/) statement to check if a variable is null and assign an error message if it is:

void main() {

String? input;

String message;

if (input != null) {

message = input;

} else {

message = 'Error';

}

print(message);

}

Output:

Error

To make this code more precise, you can use the if-null operator:

value ?? value\_if\_

The ?? operator returns the value if the value is not null. Otherwise, it returns the value\_if\_null. By using the if-null operator, you can turn the example above into a more concise code:

void main() {

String? input;

String message = input ?? 'Error';

print(message);

}

Output:

Error

The null-aware assignment operator (??=)

Sometimes, you have a single nullable variable:

String? input;

And you want to assign it a value if it is null like this:

input = input ?? 'Error';

This code works well. However, it is quite redundant to reference the input variable on both sides of the expression.

To avoid this, you can use the null-aware assignment operator:

input ??= 'Error';

In this syntax, if the input is null, it is assigned the string 'Error'.

The null-aware access operator (?.)

Suppose you have a nullable string variable called input like this:

String? input;

To access the property and method of the String type, you need to check if it’s not null:

void main() {

String? input;

if (input != null) {

print(input.length);

print(input.toLowerCase());

}

}

To avoid using the if statement, you can use the null-aware access operator (?.):

objectName?.property

The null-aware access operator returns null if the objectName is null. For example:

void main() {

String? input;

print(input?.length); *// null*

print(input?.toLowerCase()); *// null*

}

Null assertion operator (!)

The following code causes a compile-time error:

bool? isTextFile(String? filename) {

if (filename != null) {

return filename.endsWith('.txt') ? true : false;

}

return null;

}

void main() {

bool result = isTextFile('readme.txt');

print(result);

}

Error:

A value of type 'bool?' can't be assigned to a variable of type 'bool'.Code language: Dart (dart)

The isTextFile() function returns a value of the bool? type not bool. Therefore, you cannot assign the result of the isTextFile() function to the result variable.

Since you’re sure that the isTextFile('readme.txt') doesn’t return a null value, you can use the null assertion operator (!) like this:

bool result = isTextFile('readme.txt')!;

The program will look like the following:

bool? isTextFile(String? filename) {

if (filename != null) {

return filename.endsWith('.txt') ? true : false;

}

return null;

}

void main() {

bool result = isTextFile('readme.txt')!;

print(result);

}

Null-aware index operator

The null-aware index operator ?[] allows you to access an element of a list when the list might be null. For example:

void main() {

List? scores = [1, 2, 3, 4, 5];

*// somewhere in the code*

scores = null;

print(scores?[3]); *// null*

}

In this example, we use the null-aware index operator to access the 4th element of the scores list. Because the scores list is null, the ?[] returns a null value.

Summary

* Use null-aware operators to deal with null values.