Collections

Dart has built-in support for list, set, and map collections. To learn more about configuring the types collections contain, check out Generics.

Lists

Perhaps the most common collection in nearly every programming language is the *array*, or ordered group of objects. In Dart, arrays are `List` objects, so most people just call them *lists*.

Dart list literals are denoted by a comma separated list of expressions or values, enclosed in square brackets (`[]`). Here's a simple Dart list:

```dart
var list = [1, 2, 3];
```

ⓘ 提示

Dart infers that `list` has type `List<int>`. If you try to add non-integer objects to this list, the analyzer or runtime raises an error. For more information, read about type inference.

You can add a comma after the last item in a Dart collection literal. This *trailing comma* doesn't affect the collection, but it can help prevent copy-paste errors.

```dart
var list = [
  'Car',
  'Boat',
  'Plane',
];
```

Lists use zero-based indexing, where 0 is the index of the first value and `list.length - 1` is the index of the last value. You can get a list's length using the `.length` property and access a list's values using the subscript operator (`[]`):

```dart
var list = [1, 2, 3];
assert(list.length == 3);
assert(list[1] == 2);

list[1] = 1;
assert(list[1] == 1);
```

To create a list that's a compile-time constant, add `const` before the list literal:

```dart
var constantList = const [1, 2, 3];
// constantList[1] = 1; // This line will cause an error.
```

For more information about lists, refer to the Lists section of the `dart:core` documentation.

Sets

A set in Dart is an unordered collection of unique items. Dart support for sets is provided by set literals and the `Set` type.

Here is a simple Dart set, created using a set literal:

```dart
var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};
```

ⓘ 提示

Dart infers that `halogens` has the type `Set<String>`. If you try to add the wrong type of value to the set, the analyzer or runtime raises an error. For more information, read about type inference.

To create an empty set, use `{}` preceded by a type argument, or assign `{}` to a variable of type `Set`:

```dart
var names = <String>{};
// Set<String> names = {}; // This works, too.
// var names = {}; // Creates a map, not a set.
```

ⓘ Set or map?

The syntax for map literals is similar to that for set literals. Because map literals came first, `{}` defaults to the `Map` type. If you forget the type annotation on `{}` or the variable it's assigned to, then Dart creates an object of type `Map<dynamic, dynamic>`.

Add items to an existing set using the `add()` or `addAll()` methods:

```dart
var elements = <String>{};
elements.add('fluorine');
elements.addAll(halogens);
```

Use `.length` to get the number of items in the set:

```dart
var elements = <String>{};
elements.add('fluorine');
elements.addAll(halogens);
assert(elements.length == 5);
```

To create a set that's a compile-time constant, add `const` before the set literal:

```dart
final constantSet = const {
  'fluorine',
  'chlorine',
  'bromine',
  'iodine',
  'astatine',
};
// constantSet.add('helium'); // This line will cause an error.
```

For more information about sets, refer to the Sets section of the `dart:core` documentation.

Maps

In general, a map is an object that associates keys and values. Both keys and values can be any type of object. Each *key* occurs only once, but you can use the same *value* multiple times. Dart support for maps is provided by map literals and the Map type.

Here are a couple of simple Dart maps, created using map literals:

```dart
var gifts = {
  // Key:    Value
  'first': 'partridge',
  'second': 'turtledoves',
  'fifth': 'golden rings'
};

var nobleGases = {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};
```

ⓘ 提示

Dart infers that `gifts` has the type `Map<String, String>` and `nobleGases` has the type `Map<int, String>`. If you try to add the wrong type of value to either map, the analyzer or runtime raises an error. For more information, read about type inference.

You can create the same objects using a Map constructor:

```dart
var gifts = Map<String, String>();
gifts['first'] = 'partridge';
gifts['second'] = 'turtledoves';
gifts['fifth'] = 'golden rings';

var nobleGases = Map<int, String>();
nobleGases[2] = 'helium';
nobleGases[10] = 'neon';
nobleGases[18] = 'argon';
```

ⓘ 提示

If you come from a language like C# or Java, you might expect to see `new Map()` instead of just `Map()`. In Dart, the `new` keyword is optional. For details, see Using constructors.

Add a new key-value pair to an existing map using the subscript assignment operator (`[]=`):

```dart
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds'; // Add a key-value pair
```

Retrieve a value from a map using the subscript operator (`[]`):

```dart
var gifts = {'first': 'partridge'};
assert(gifts['first'] == 'partridge');
```

If you look for a key that isn't in a map, you get `null` in return:

```dart
var gifts = {'first': 'partridge'};
assert(gifts['fifth'] == null);
```

Use `.length` to get the number of key-value pairs in the map:

```dart
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds';
assert(gifts.length == 2);
```

To create a map that's a compile-time constant, add `const` before the map literal:

```dart
final constantMap = const {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};

// constantMap[2] = 'Helium'; // This line will cause an error.
```

For more information about maps, refer to the Maps section of the [`dart:core` documentation].

Operators

Spread operators

Dart supports the **spread operator** (`...`) and the **null-aware spread operator** (`...?`) in list, map, and set literals. Spread operators provide a concise way to insert multiple values into a collection.

For example, you can use the spread operator (`...`) to insert all the values of a list into another list:

```dart
var list = [1, 2, 3];
var list2 = [0, ...list];
assert(list2.length == 4);
```

If the expression to the right of the spread operator might be null, you can avoid exceptions by using a null-aware spread operator (`...?`):

```dart
var list2 = [0, ...?list];
assert(list2.length == 1);
```

For more details and examples of using the spread operator, see the spread operator proposal.

Control-flow operators

Dart offers **collection if** and **collection for** for use in list, map, and set literals. You can use these operators to build collections using conditionals (`if`) and repetition (`for`).

Here's an example of using **collection if** to create a list with three or four items in it:

```dart
var nav = ['Home', 'Furniture', 'Plants', if (promoActive) 'Outlet'];
```

Dart also supports if-case inside collection literals:

```dart
var nav = ['Home', 'Furniture', 'Plants', if (login case 'Manager') 'Inventory'];
```

Here's an example of using **collection for** to manipulate the items of a list before adding them to another list:

```dart
var listOfInts = [1, 2, 3];
var listOfStrings = ['#0', for (var i in listOfInts) '#$i'];
assert(listOfStrings[1] == '#1');
```

For more details and examples of using collection `if` and `for`, see the control flow collections proposal.