Querying Collections

Effective Programming in Scala

Querying Collections

In this section we'll see methods that allow us to query properties of collections. For example we can determine the number of elements in a collection or reduce a collection to just those elements that match a predicate.

Querying: Simple Properties

Several methods allow us to query simple properties of a collection, such as the number of elements it contains.

Operation	Description
`xs.size`	Number of elements of the collection `xs`
`xs.isEmpty`, `xs.nonEmpty`	Is the collection `xs` empty (or not empty)?
`xs.contains(x)`	Does the collection `xs` contain the element `x`?

Querying: Simple Properties

Several methods allow us to query simple properties of a collection, such as the number of elements it contains.

Operation	Description
`xs.size`	Number of elements of the collection `xs`
`xs.isEmpty`, `xs.nonEmpty`	Is the collection `xs` empty (or not empty)?
`xs.contains(x)`	Does the collection `xs` contain the element `x`?

Note: when calling `contains` on a `Map` we pass an example of a key.

Querying: Finding Elements

Sometimes we want to find the first or all the elements in a collection that satisfy a predicate. For this we can use the `find` and `filter` methods respectively.

```
val data = List(1, 2, 3, 4)

// Find first even number
data.find(x => x % 2 == 0) // Some(2)

// Find all even numbers
data.filter(x => x % 2 == 0) // List(2, 4)
```

Querying: Finding Elements

Sometimes we want to find the first or all the elements in a collection that satisfy a predicate. For this we can use the `find` and `filter` methods respectively.

```
val data = List(1, 2, 3, 4)

// Find first even number
data.find(x => x % 2 == 0) // Some(2)

// Find all even numbers
data.filter(x => x % 2 == 0) // List(2, 4)
```

Note: `find` returns an `Option`. This indicates we may not find an element that matches the predicate. Let's take a quick look at `Option`.

Aside: Option

We will learn all about `Option` soon. For now we need to know that `Option` is a special collection that contains zero or one element. Thus it can represent an "optional" value; a value that might be missing.

There are two cases to `Option`: `Some` when there is a value, and `None` otherwise.

When we call `find` it will return the `Some` case of `Option` if a value is found.

```
List(1, 2, 3, 4).find(x => x == 1) // Some(1)
```

If no value is found the `None` case is returned.

```
List(1, 2, 3, 4).find(x => x == 5) // None
```

Querying: Summary

We have seen the following methods to query properties of a collection

- `size` to get the number of elements in a collection;
- `isEmpty` and `nonEmpty` to determine if a collection contains elements;
- `contains` to test if a collection contains an element;
- `find` to return the first element that matches a predicate;
- `filter` to return all the elements that match a predicate;

We also briefly discussed `Option`.

Querying: Summary

We have seen the following methods to query properties of a collection

- `size` to get the number of elements in a collection;
- `isEmpty` and `nonEmpty` to determine if a collection contains elements;
- `contains` to test if a collection contains an element;
- `find` to return the first element that matches a predicate;
- `filter` to return all the elements that match a predicate;

We also briefly discussed `Option`.

Note: There are many other querying methods, such as `filterNot` and `findLast`, that we have not discussed. Check the API documentation for more: `https://scala-lang.org/api`.