Understanding map, filter, and flatMap in Scala

Understanding map, filter, and flatMap in Scala

These functions are key components in functional programming and are widely used for working with collections and monads like Option, List, and Future.

---

1. map

The `map` function applies a given function to each element in a collection or monad and returns a new collection with the results.

Syntax

```
collection.map(f: A => B): Collection[B]
```

- **Input**: A collection and a function `f` to transform each element.
- **Output**: A new collection containing the results of applying `f` to each element.

Example

```scala
val numbers = List(1, 2, 3, 4)
```

```
val squares = numbers.map(x => x * x)

println(squares) // Output: List(1, 4, 9, 16)
```

---

2. filter

The `filter` function selects elements from a collection that satisfy a given condition (predicate).

Syntax

```
collection.filter(p: A => Boolean): Collection[A]
```

- **Input**: A collection and a predicate `p` that returns `true` or `false` for each element.
- **Output**: A new collection containing only elements where the predicate evaluates to `true`.

Example

```scala
val numbers = List(1, 2, 3, 4, 5)

val evenNumbers = numbers.filter(x => x % 2 == 0)

println(evenNumbers) // Output: List(2, 4)
```

---

3. flatMap

The `flatMap` function is a combination of `map` followed by `flatten`. It applies a function that produces a collection for each element, and then flattens the resulting nested collections into a single collection.

Syntax

```
collection.flatMap(f: A => Collection[B]): Collection[B]
```

- **Input**: A collection and a function `f` that transforms each element into a collection.
- **Output**: A single flattened collection containing all the elements from the collections produced by `f`.

Example

```scala
val numbers = List(1, 2, 3)
val expanded = numbers.flatMap(x => List(x, x * 2))
println(expanded) // Output: List(1, 2, 2, 4, 3, 6)
```

Key Differences Between map and flatMap

1. **Nested Results**:

   - `map` produces a collection of collections when the function returns a collection.

   - `flatMap` flattens the result into a single collection.

Example

Using `map`:

```scala
val numbers = List(1, 2, 3)
val mapped = numbers.map(x => List(x, x * 2))
println(mapped) // Output: List(List(1, 2), List(2, 4), List(3, 6))
```

Using `flatMap`:

```scala
val numbers = List(1, 2, 3)
val flatMapped = numbers.flatMap(x => List(x, x * 2))
println(flatMapped) // Output: List(1, 2, 2, 4, 3, 6)
```

Practical Use Cases

- **map**: Transforming data, e.g., converting strings to uppercase.

- **filter**: Selecting specific elements, e.g., filtering even numbers.

- **flatMap**: Flattening nested structures, e.g., parsing nested JSON or working with Option and Future.

Example Using All Three

```scala
val numbers = List(1, 2, 3, 4, 5)

// Double the even numbers and return a flat list
val result = numbers
  .filter(x => x % 2 == 0) // Keep only even numbers
  .flatMap(x => List(x, x * 2)) // Expand each number into two values
println(result) // Output: List(2, 4, 4, 8)
```

These functions form the backbone of functional transformations in Scala, making them essential for functional programming tasks!