Spark - Scala Methods

Scala - Spark methods

1. **foreach**
2. **map**
3. **flatMap**
4. **filter**
5. **withFilter**
6. **collect**
7. **reduce**
8. **fold**
9. **groupBy**
10. **partition**

1. Overview of the Methods

Method	Purpose	Returns	When to Use
foreach	**Iterates** over each element and performs a side effect (like printing or writing to DB).	**Unit** (**no** return **value**)	When you only want to perform actions, not transform data.
map	**Transforms** each element into something else.	**Collection** of the same size as input.	When every element maps to exactly one new element.
flatMap	**Transforms** each element into **zero or more** elements and flattens the result.	**Collection** of potentially different size.	When each element might become multiple elements.
filter	Keeps only elements that **satisfy** a **condition**.	Smaller (or equal) size **collection**.	When you want to remove unwanted elements.
withFilter	Like filters but **lazy**, used with comprehensions for efficiency.	A **filtered view** (not an immediate new collection).	When chaining filters in a for comprehension.
collect	Applies a **partial function** (matching & transforming at once).	**Collection**	When you want to filter + transform in one step.
reduce	Combines all elements using a binary function.	**Single** value	When you want to aggregate (sum, product, min, max).
fold	Like reduce but starts with an initial value.	**Single** value	When aggregation needs a start value.

groupBy	Groups elements into a Map based on a key function.	**Map**[K, **Seq**[V]]	When categorizing items.
partition	Splits collection into two based on predicate.	**Tuple** of two **collections**	When you want "pass" and "fail" groups.

2. Complete Scala Example

```
object CollectionMethodsDemo {
  def main(args: Array[String]): Unit = {

    val numbers = List(1, 2, 3, 4, 5, 6)

    // foreach: Just perform an action (no transformation)
    println("foreach:")
    numbers.foreach(n => println(s"Number: $n"))

    // map: Transform each element
    val squares = numbers.map(n => n * n)
    println(s"map (squares): $squares")

    // flatMap: Transform each element into a collection, then flatten
    val duplicates = numbers.flatMap(n => List(n, n))
    println(s"flatMap (duplicates): $duplicates")

    // filter: Keep only even numbers
    val evens = numbers.filter(_ % 2 == 0)
    println(s"filter (evens): $evens")

    // withFilter: Lazy filtering - useful with for comprehension
    println("withFilter example:")
    val withFilterResult = for {
      n <- numbers.withFilter(_ % 2 == 0) // only even numbers
    } yield n * 10
    println(withFilterResult)

    // collect: Filter and transform together
    val evenSquares = numbers.collect { case n if n % 2 == 0 => n * n }
    println(s"collect (even squares): $evenSquares")
```

```
    // reduce: Aggregate (sum)
    val sum = numbers.reduce(_ + _)
    println(s"reduce (sum): $sum")

    // fold: Aggregate with initial value
    val sumWithInit = numbers.fold(10)(_ + _)
    println(s"fold (sum with initial 10): $sumWithInit")

    // groupBy: Group numbers by odd/even
    val grouped = numbers.groupBy(n => if (n % 2 == 0) "Even" else "Odd")
    println(s"groupBy (odd/even): $grouped")

    // partition: Split into evens and odds
    val (evenNums, oddNums) = numbers.partition(_ % 2 == 0)
    println(s"partition evens: $evenNums, odds: $oddNums")
  }
}
```

https://alvinalexander.com/scala/scala-cheat-sheet-basic-quick-reference-syntax/

https://www.scala-exercises.org/std_lib/classes