Transformaciones sobre un RDD

Al aplicar una transformación sobre un RDD original,se devolverá un RDD nuevo resultado de dicha transformación. De esta forma las transformaciones no modifican el RDD original.

Un punto importante a tener en cuenta es que las **transformaciones Spark son perezosas**. Es decir, invocar una transformación en un RDD no activa inmediatamente un cálculo. En cambio, las transformaciones se encadenan entre sí y sólo se calculan efectivamente cuando se llama a una acción. Esto permite que Spark sea más eficiente devolviendo resultados al controlador cuando sea necesario para que la mayoría de las operaciones se realicen en paralelo en el clúster.

- **map(func):** Devuelve un nuevo RDD ,resultado de pasar cada uno de los elementos del RDD original como parámetro de la función func. Aplica una función a cada elemento de la colección.

    - **numbers_rdd.map(lambda x: x**2).collect()**
      Salida:[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

- **distinct():** devuelve un nuevo RDD que contiene una sola copia de los diferentes elementos del RDD original. Elimina duplicados.
- **filter(func):** devuelve un nuevo RDD que sólo contiene los elementos del RDD original que satisfacen el predicado especificado en la función func. Sólo procesa aquellos elementos que cumplen la condición.

    - **num=sc.parallelize([1,2,3,4,5,6,100,2000,4000])**
      num_mayor_10=num.filter(lambda x:x>10)
      num_mayor_10.collect()
      Salida: [100,2000,4000]

- **groupByKey():** devuelve un nuevo RDD de pares clave/valor,donde las claves guardan cada una de las diferentes claves del RDD original y los valores asociados se agrupan en un objeto iterable con los valores que estaban asociados a la misma clave del RDD original
- **reduceByKey(func):** devuelve un RDD de pares clave/valor donde cada clave única se corresponde con las diferentes claves del RDD original y el valor es el resultado de aplicar una operación reduce sobre los valores correspondientes a una misma clave.En resumen,combina valores con la misma clave en cada procesado.

    - **rdd=sc.parallelize([('a',1),('b',2)('a',3),('b',4)])**
      rdd.reduceByKey(lambda x,y:x+y).collect()
      Salida: [('a',4),('b',6)]

- **flatMap(function):** Esto devuelve un RDD formado por aplanamiento de la salida de la función para cada elemento de la entrada RDD. Se utiliza cuando cada valor en la entrada se puede asignar a 0 o más elementos de salida.

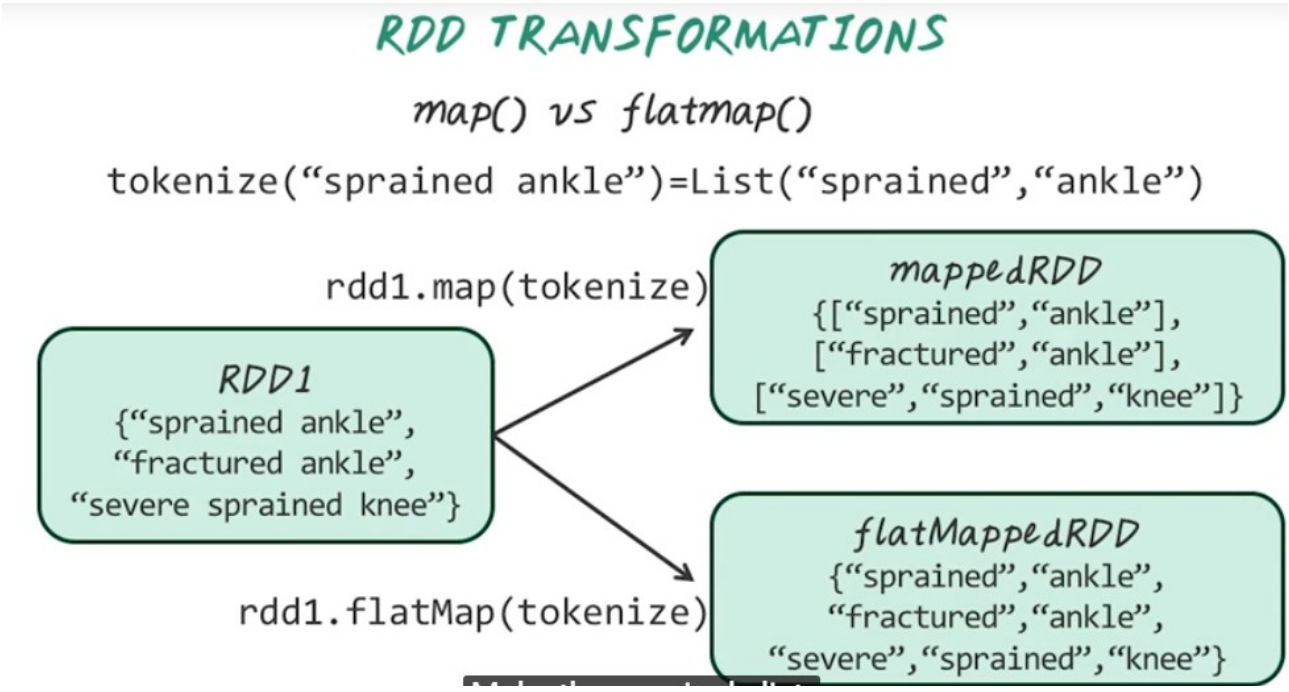
- **Uso de flatMap:** flatMap permite partir una frase en palabras

- **lines=sc.parallelize([“hello world”])**
  words=lines.flatMap(lambda line:line.split(“ ”)
  words.first()

- **filter(function):** Devuelve un conjunto de datos compuesto por todos los valores donde la función devuelve true.
- **distinct():** Devuelve un RDD que contiene elementos distintos de la entrada RDD.
- **repartition(numPartitions):** Esto cambia el número de particiones en el RDD. Este método siempre baraja todos los datos a través de la red.
- **groupByKey():** Crea un RDD donde, para cada clave, el valor es una secuencia de valores que tienen esa clave en el conjunto de datos de entrada.
- **reduceByKey (function):** Esto agrega la entrada RDD por clave y luego aplica la función reduce a los valores de cada grupo
- **sortByKey(ascending):** Esto ordena los elementos en el RDD por clave en orden ascendente o descendente
- **union(otherRDD):** Esto fusiona dos RDDs juntos
- **intersection(otherRDD):** Esto devuelve un RDD compuesto por sólo los valores que aparecen tanto en la entrada como en el RDD pasado como argumento.
- **join(otherRDD):** Devuelve un conjunto de datos en el que las entradas de valor-clave se unen (en la clave) RDD pasado como argumento.

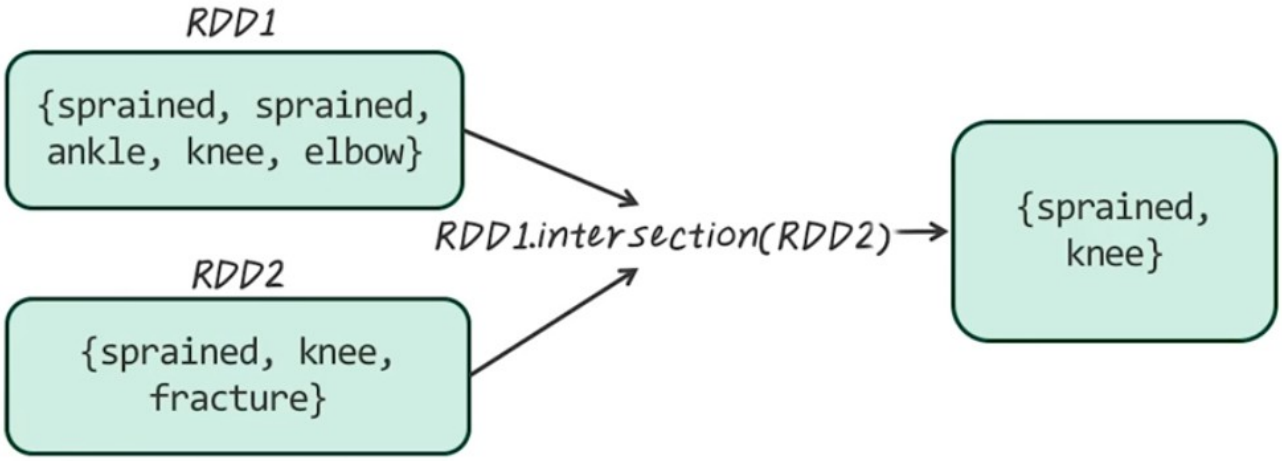
Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic



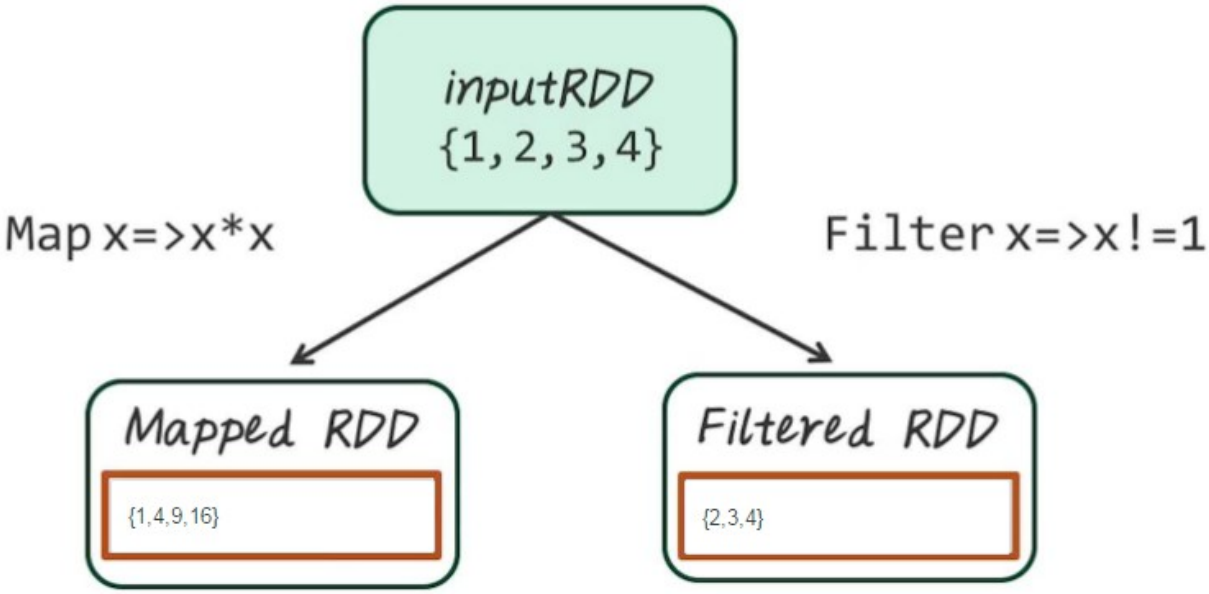
RDD TRANSFORMATIONS

Operation: Intersection()



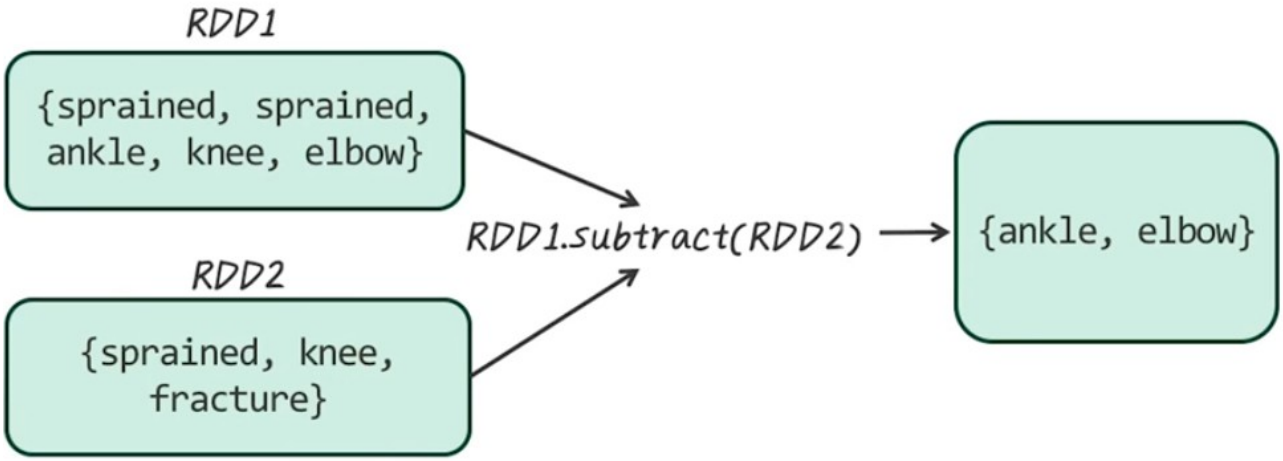
RDD TRANSFORMATIONS QUIZ

What is the output of each of the given transformations of inputRDD?



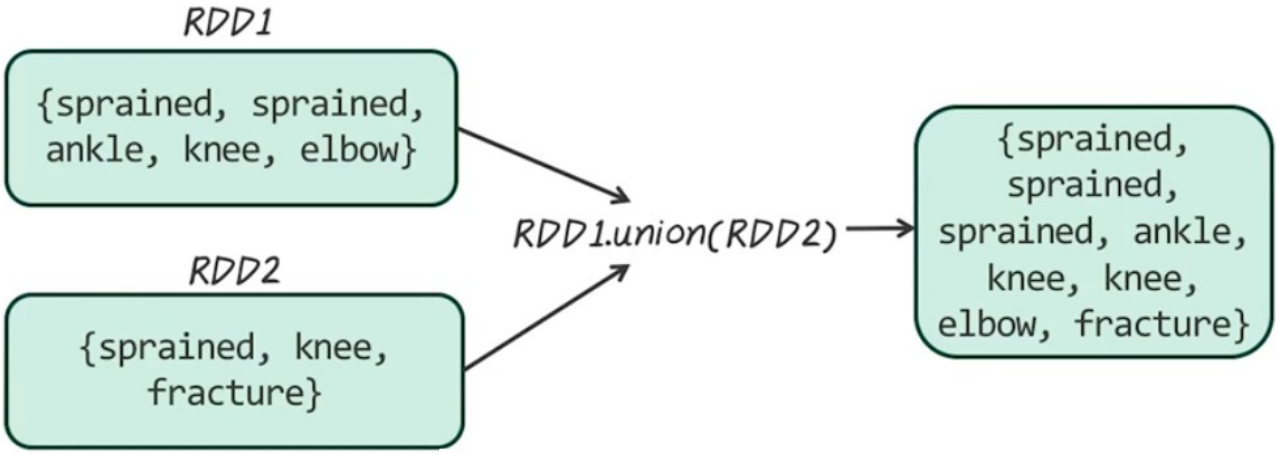
RDD TRANSFORMATIONS

Operation: Subtract()



RDD TRANSFORMATIONS

Operation: Union()



El código fuente de la clase rdd para ver lo que hace por debajo se puede ver en el respositorio de código oficial de spark

https://apache.googlesource.com/spark/+/master/python/pyspark/rdd.py

```
class RDD(object):

    """
    A Resilient Distributed Dataset (RDD), the basic abstraction in Spark.
    Represents an immutable, partitioned collection of elements that can be
    operated on in parallel.
    """

    def __init__(self, jrdd, ctx, jrdd_deserializer=AutoBatchedSerializer(PickleSerializer())):
        self._jrdd = jrdd
        self.is_cached = False
        self.is_checkpointed = False
        self.ctx = ctx
        self._jrdd_deserializer = jrdd_deserializer
        self._id = jrdd.id()
        self.partitioner = None
```