■ Spark RDD(Resilient Distributed Dataset)

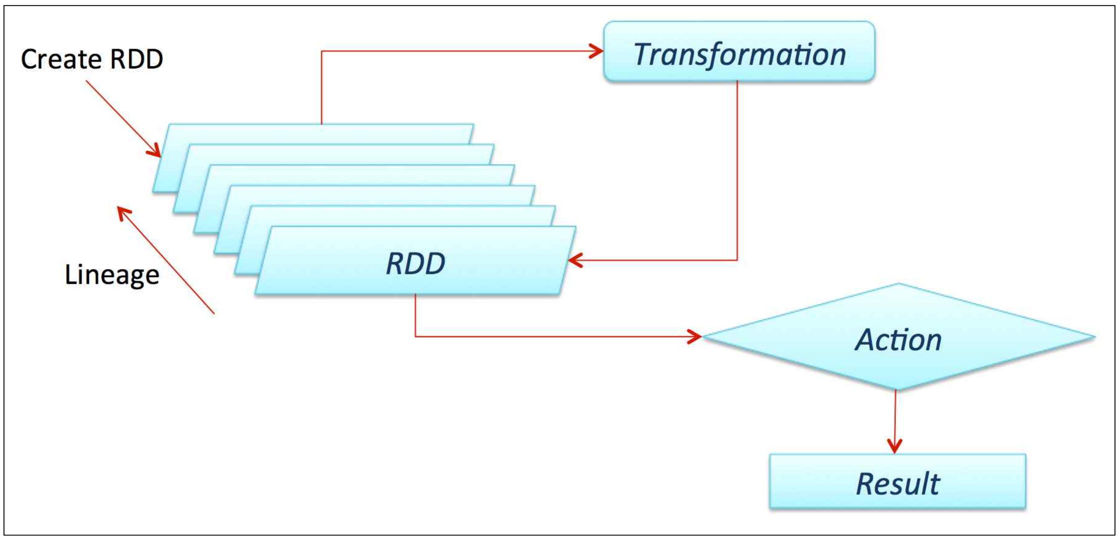
1) RDD의 성질

- 불변성 : 읽기 전용
- 복원성 : 장애 내성
- 분산 : 노드 한 개 이상에 저장된 데이터 셋

2) RDD의 유형

- 변환(Transformation) : 데이터를 조작해 새로운 RDD를 생성
- 행동(Action) : 계산결과를 반환하거나 특정 작업을 수행하려고 실제 계산을 시작하는 역할을 함

3) Lazy Evaluation



- 여러번 transformation을 해도 실제로 작동되지는 않고, 액션 메서드를 호출할 때 transformation을 수행하는 개념으로, 계획에 따라 빠르고 효율적으로 transformation을 가능하게 함

4) 자바의 람다식(java 8에서 추가됨)

- 식별자 없이 실행 가능한 함수 표현식

```
( parameters ) -> expression body
( parameters ) -> { expression body }
() -> { expression body }
() -> expression body
```

- 예제

```
package org.jbm.spark_0626;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;

public class LambdaTest {

        public static void main(String[] args) {

                List<Integer> list = new ArrayList<>();

                list.add(1);
                list.add(2);
                list.add(3);
                list.add(4);
                list.add(5);

                //기존의 for each문
                for(int value : list) {
                        System.out.println(value);
                }//for end

                //람다식의 for each 문
                list.forEach(value->System.out.println(value));

        }
}
```

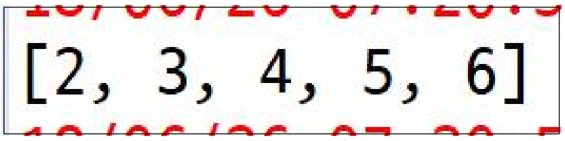
■ 변환(Transformation) 연산자

1) map 연산자 : 원본 RDD의 모든 요소에 임의의 함수를 적용할 수 있는 변환 연산자

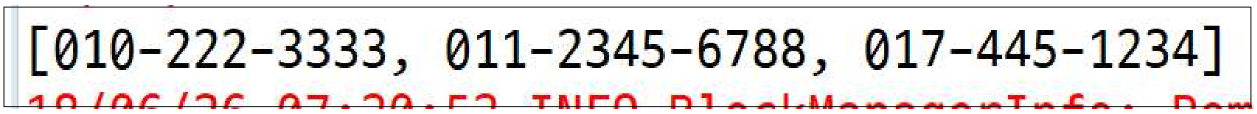
```java
package org.jbm.spark_0626;

import java.util.Arrays;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI3_map {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                executeMap1();
                executeMap2();
                executeMap3();
        }

        //1,2,3,4,5인 RDD를 +1해서 2,3,4,5,6으로 만드는 예제
        private static void executeMap1() {

                //1,2,3,4,5인 JavaRDD를 생성
                JavaRDD<Integer> rdd1 = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5));

                // -> 함수는 아래와 같은 코드임 즉, 인자() {} return 등을 생략가능함
                JavaRDD<Integer> rdd2 = rdd1.map(i -> i + 1);

                //JavaRDD<Integer> rdd2 = rdd1.map((Integer i) -> {
                //      return i + 1;
                //});

                System.out.println(rdd2.collect());
        }

        //전화번호를 쪼개서 '-'을 넣어주는 예제
        private static void executeMap2() {
```

```
        //1,2,3,4,5인 JavaRDD를 생성
        JavaRDD<String> rdd =
        sc.parallelize(Arrays.asList("0102223333","01123456788","0174451234"))
          .map(phone -> {

                String p1 = phone.substring(0, 3);
                String p2 = phone.substring(3,phone.length()-4);
                String p3 = phone.substring(phone.length()-4,phone.length());

                return p1+"-"+p2+"-"+p3;
        });

        System.out.println(rdd.collect());
    }


    //23살...이런 String형을 23...이런 Integer형으로 변환하는 예제
    private static void executeMap3() {

        JavaRDD<String> rdd = sc.parallelize(Arrays.asList("15살","23살","34살","47살"));

        //맵은 RDD타입이 같을 수도 있고, 다를 수도 있음
        JavaRDD<Integer>          rdd2          =          rdd.map(age          ->
Integer.parseInt(age.substring(0,age.length()-1)));

        System.out.println(rdd2.collect());

    }

}
```

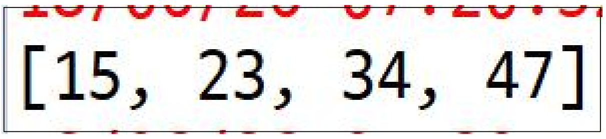
- 결과



```
[2, 3, 4, 5, 6]
```



```
[010-222-3333, 011-2345-6788, 017-445-1234]
```

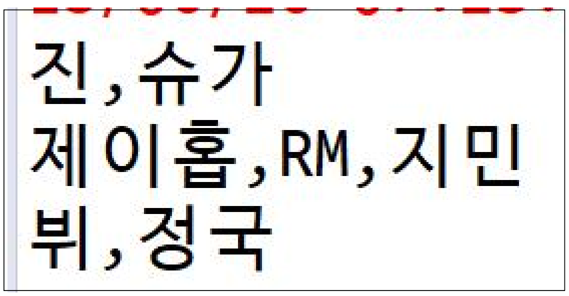


```
[15, 23, 34, 47]
```

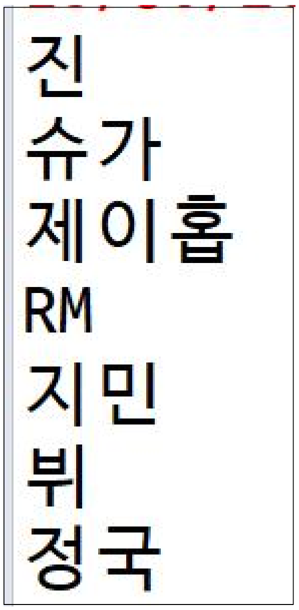
2) flatMap 연산자 : map과 유사하지만 iterator를 리턴

```
package org.jbm.spark_0626;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI4_flatMap {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                List<String> data = new ArrayList<String>();

                data.add("진,슈가");
                data.add("제이홉,RM,지민");
                data.add("뷔,정국");

                JavaRDD<String> rdd1 = sc.parallelize(data);

                System.out.println("rdd1은");
                for (String s : rdd1.collect()) {
                        System.out.println(s);
                } // for end

                JavaRDD<String> rdd2 = rdd1.flatMap(s -> Arrays.asList(s.split(",")).iterator());

                System.out.println("");
                System.out.println("rdd2는");
                for (String s : rdd2.collect()) {
                        System.out.println(s);
                } // for end
        }
}
```

- 결과



```
진,슈가
제이홉,RM,지민
뷔,정국
```

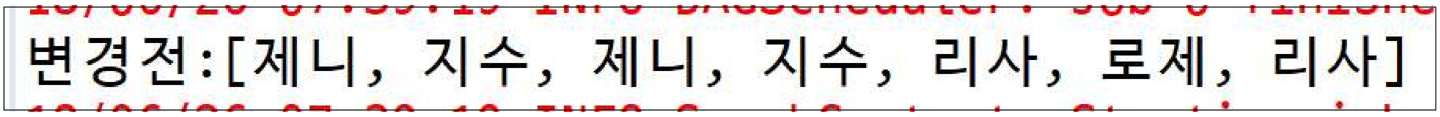


```
진
슈가
제이홉
RM
지민
뷔
정국
```

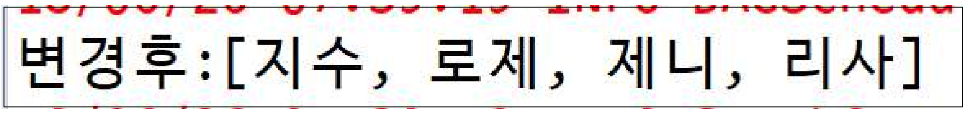
3) distinct 연산자 : 중복 제거

```
package org.jbm.spark_0626;

import java.util.Arrays;
import org.apache.commons.lang3.ArrayUtils;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI5_distinct {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                JavaRDD<String> rdd =
                                sc.parallelize(Arrays.asList("제니","지수","제니","지수","리사","로제","
리사"));

                System.out.println("변경전:"+rdd.collect());

                rdd =rdd.distinct();

                System.out.println("변경후:"+rdd.collect());

        }

}
```

- 결과



```
변경전:[제니, 지수, 제니, 지수, 리사, 로제, 리사]
```

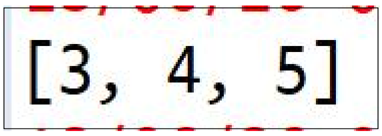


```
변경후:[지수, 로제, 제니, 리사]
```

4) filter 연산자 : 리턴값이 true인 경우만 필터링

```
package org.jbm.spark_0626;

import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI7_filter {
        private static JavaSparkContext sc;
        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5));

                //2보다 큰 값만
                JavaRDD<Integer> result = rdd.filter((v) -> v > 2);

                System.out.println(result.collect());
        }
}
```

- 결과



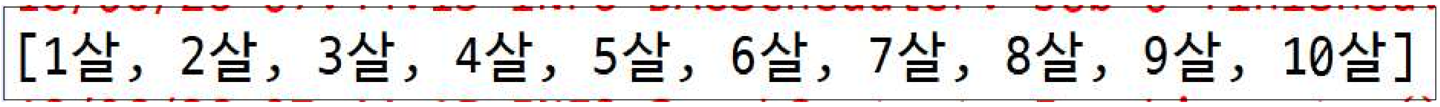
```
[3, 4, 5]
```

5) mapPartitions 연산자 : flatMap과 비슷하나 파티션별로 나눠서 작동됨

```
package org.jbm.spark_0626;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI6_mapPartitions {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                //파티션이 4개
                JavaRDD<Integer> rdd1 = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), 4);

                JavaRDD<String> rdd2 = rdd1.mapPartitions(numbers -> {

                        List<String> result = new ArrayList<>();

                        while (numbers.hasNext()) {
                                result.add(numbers.next() +"살");
                        }

                        return result.iterator();
                });

                System.out.println(rdd2.collect());

        }
}
```

- 결과

```
TaskSchedulerImpl: Adding task set 0.0 with 4 tasks
```

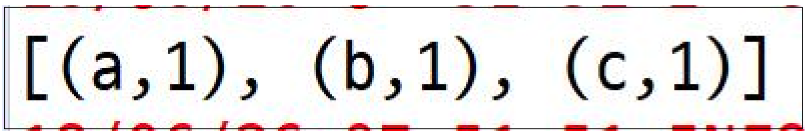


```
[1살, 2살, 3살, 4살, 5살, 6살, 7살, 8살, 9살, 10살]
```

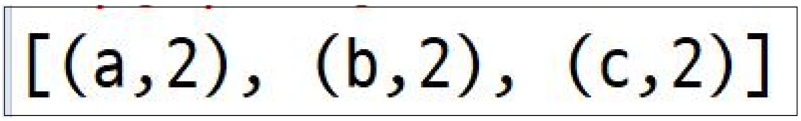
6) mapValues 연산자 : value값을 변형시킬 수 있는 연산자

```
package org.jbm.spark_0626;

import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;

public class RDDAPI8_mapValues {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "b", "c"));

                //mapToPair는 하나의 RDD를 PairRDD로 변환시켜주는 연산자
                JavaPairRDD<String, Integer> rdd2 =
                                rdd1.mapToPair((String t) -> new Tuple2<String, Integer>(t, 1));

                System.out.println(rdd2.collect());

                //mapValues는 값을 변환시켜주는 연산자
                rdd2 = rdd2.mapValues((Integer v1) -> v1 + 1);

                System.out.println(rdd2.collect());

        }
}
```

-결과



```
[(a,1), (b,1), (c,1)]
```

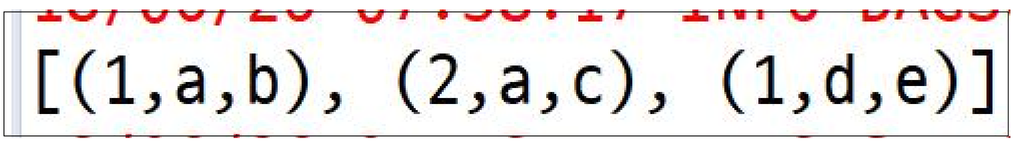


```
[(a,2), (b,2), (c,2)]
```

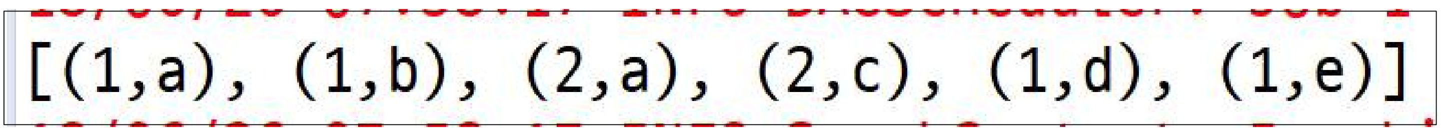
7) flatMapValues 연산자 : flatMap과 흡사함(value값에 대해서)

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;

public class RDDAPI9_flatMapValues {
        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                List<Tuple2<Integer, String>> data =
                                Arrays.asList(
                                                new Tuple2<Integer, String>(1, "a,b"),
                                                new Tuple2<Integer, String>(2, "a,c"),
                                                new Tuple2<Integer, String>(1, "d,e")
                                                );

                JavaPairRDD<Integer, String> rdd1 = sc.parallelizePairs(data);

                System.out.println(rdd1.collect());

                JavaPairRDD<Integer, String> rdd2 =
                                rdd1.flatMapValues((String v1) -> Arrays.asList(v1.split(",")));

                System.out.println(rdd2.collect());
        }
}
```

- 결과



```
[(1,a,b), (2,a,c), (1,d,e)]
```

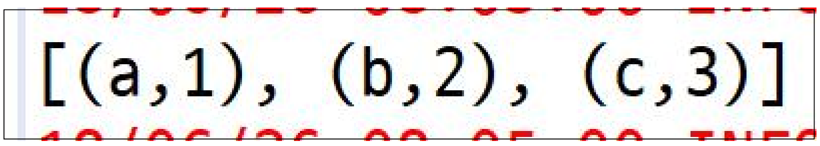


```
[(1,a), (1,b), (2,a), (2,c), (1,d), (1,e)]
```

8) zip 연산자 : 두개의 RDD를 PairRDD로 합침

```
package org.jbm.spark_0626;

import java.util.Arrays;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI10_zip {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "b", "c"));

                JavaRDD<Integer> rdd2 = sc.parallelize(Arrays.asList(1, 2, 3));

                JavaPairRDD<String, Integer> result = rdd1.zip(rdd2);

                System.out.println(result.collect());

        }

}
```

- 결과



```
[(a,1), (b,2), (c,3)]
```

9) 그 외의 연산자들 : 예제에서 하나씩 실행해서 공부함

```
package org.jbm.spark_0626;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.stream.Collectors;

import org.apache.spark.HashPartitioner;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaDoubleRDD;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.Optional;
import org.apache.spark.storage.StorageLevel;
import org.apache.spark.streaming.util.BatchedWriteAheadLog.Record;

import scala.Tuple2;

public class RDDAPI12_ETC {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {
                                doMapValues(sc);
                                doFlatMapValues(sc);
                                doGroupBy(sc);
                                doGroupByKey(sc);
                                doCogroup(sc);
                                doDistinct(sc);
                                doCartesian(sc);
                                doSubtract(sc);
                                doUnion(sc);
                                doIntersection(sc);
                                doJoin(sc);
                                doLeftOuterJoin(sc);
                                doSubtractByKey(sc);
                                doFoldByKey(sc);
                                doPipe(sc);
                                doCoalesceAndRepartition(sc);
                                doRepartitionAndSortWithinPartitions(sc);
                                doPartitionBy(sc);
                                doSortByKey(sc);
                                doKeysAndValues(sc);
                                doSample(sc);
                                doFirst(sc);
                                doTake(sc);
                                doTakeSample(sc);
                                doCountByValue(sc);
                                doFold(sc);
                                doSum(sc);
                                doForeach(sc);
                                doForeachPartition(sc);
                                doDebugString(sc);
                                doCache(sc);
                                doGetPartitions(sc);

        }

        public static void doMapValues(JavaSparkContext sc) {

                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "b", "c"));

                JavaPairRDD<String, Integer> rdd4 =
                                rdd1.mapToPair((String    t)    ->    new    Tuple2<String,    Integer>(t,
1)).mapValues((Integer v1) -> v1 + 1);

                System.out.println(rdd4.collect());
        }


        public static void doFlatMapValues(JavaSparkContext sc) {

                List<Tuple2<Integer, String>> data =
                                Arrays.asList(
                                                new Tuple2<Integer, String>(1, "a,b"),
                                                new Tuple2<Integer, String>(2, "a,c"),
                                                new Tuple2<Integer, String>(1, "d,e"));

                JavaPairRDD<Integer, String> rdd1 = sc.parallelizePairs(data);

                JavaPairRDD<Integer,    String>    rdd3    =    rdd1.flatMapValues((String    v1)    ->
Arrays.asList(v1.split(",")));

                System.out.println(rdd3.collect());
        }

        public static void doGroupBy(JavaSparkContext sc) {
                JavaRDD<Integer> rdd1 = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
                JavaPairRDD<String, Iterable<Integer>> rdd3 = rdd1.groupBy((Integer v1) -> (v1 % 2 == 0)
? "even" : "odd");

                System.out.println(rdd3.collect());
        }


        public static void doGroupByKey(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data = Arrays.asList(
                                new Tuple2<String, Integer>("a", 1),
                                new Tuple2<String, Integer>("b", 2),
                                new Tuple2<String, Integer>("c", 3),
                                new Tuple2<String, Integer>("b", 4),
                                new Tuple2<String, Integer>("c", 5));

                JavaPairRDD<String, Integer> rdd1 = sc.parallelizePairs(data);

                JavaPairRDD<String, Iterable<Integer>> rdd2 = rdd1.groupByKey();

                System.out.println(rdd2.collect());
        }

        public static void doCogroup(JavaSparkContext sc) {
                List<Tuple2<String, String>> data1 = Arrays.asList(
                                new Tuple2<String, String>("k1", "v1"),
                                new Tuple2<String, String>("k2", "v2"),
                                new Tuple2<String, String>("k1", "v3")
                                );

                List<Tuple2<String,  String>>  data2  =  Arrays.asList(new  Tuple2<String,  String>("k1",
"v4"));

                JavaPairRDD<String, String> rdd1 = sc.parallelizePairs(data1);
                JavaPairRDD<String, String> rdd2 = sc.parallelizePairs(data2);

                JavaPairRDD<String,     Tuple2<Iterable<String>,     Iterable<String>>>     result     =
rdd1.<String>cogroup(rdd2);

                System.out.println(result.collect());
        }

        public static void doDistinct(JavaSparkContext sc) {
                JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 1, 2, 3, 1, 2, 3));
                JavaRDD<Integer> result = rdd.distinct();
                System.out.println(result.collect());
        }

        public static void doCartesian(JavaSparkContext sc) {
                JavaRDD<Integer> rdd1 = sc.parallelize(Arrays.asList(1, 2, 3));
                JavaRDD<String> rdd2 = sc.parallelize(Arrays.asList("a", "b", "c"));
                JavaPairRDD<Integer, String> result = rdd1.cartesian(rdd2);
                System.out.println(result.collect());
        }

        public static void doSubtract(JavaSparkContext sc) {
                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "b", "c", "d", "e"));
                JavaRDD<String> rdd2 = sc.parallelize(Arrays.asList("d", "e"));
                JavaRDD<String> result = rdd1.subtract(rdd2);
                System.out.println(result.collect());
        }

        public static void doUnion(JavaSparkContext sc) {
                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "b", "c"));
                JavaRDD<String> rdd2 = sc.parallelize(Arrays.asList("d", "e", "f"));
                JavaRDD<String> result = rdd1.union(rdd2);
                System.out.println(result.collect());
        }

        public static void doIntersection(JavaSparkContext sc) {
                JavaRDD<String> rdd1 = sc.parallelize(Arrays.asList("a", "a", "b", "c"));
                JavaRDD<String> rdd2 = sc.parallelize(Arrays.asList("a", "a", "c", "c"));
                JavaRDD<String> result = rdd1.intersection(rdd2);
                System.out.println(result.collect());
        }

        public static void doJoin(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data1 = Arrays.asList(new Tuple2("a", 1), new Tuple2("b",
1), new Tuple2("c", 1),
                                new Tuple2("d", 1), new Tuple2("e", 1));
                List<Tuple2<String, Integer>> data2 = Arrays.asList(new Tuple2("b", 2), new Tuple2("c",
2));

                JavaPairRDD<String, Integer> rdd1 = sc.parallelizePairs(data1);
                JavaPairRDD<String, Integer> rdd2 = sc.parallelizePairs(data2);

                JavaPairRDD<String, Tuple2<Integer, Integer>> result = rdd1.<Integer>join(rdd2);
                System.out.println(result.collect());
        }

        public static void doLeftOuterJoin(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data1 = Arrays.asList(new Tuple2("a", 1), new Tuple2("b",
"1"),
                                new Tuple2("c", "1"));
                List<Tuple2<String, Integer>> data2 = Arrays.asList(new Tuple2("b", 2), new Tuple2("c",
"2"));

                JavaPairRDD<String, Integer> rdd1 = sc.parallelizePairs(data1);
                JavaPairRDD<String, Integer> rdd2 = sc.parallelizePairs(data2);

                JavaPairRDD<String,      Tuple2<Integer,      Optional<Integer>>>      result1      =
rdd1.<Integer>leftOuterJoin(rdd2);
                JavaPairRDD<String,      Tuple2<Optional<Integer>,      Integer>>      result2      =
rdd1.<Integer>rightOuterJoin(rdd2);
                System.out.println("Left: " + result1.collect());
                System.out.println("Right: " + result2.collect());
        }

        public static void doSubtractByKey(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data1 = Arrays.asList(new Tuple2("a", 1), new Tuple2("b",
1));
                List<Tuple2<String, Integer>> data2 = Arrays.asList(new Tuple2("b", 2));

                JavaPairRDD<String, Integer> rdd1 = sc.parallelizePairs(data1);
                JavaPairRDD<String, Integer> rdd2 = sc.parallelizePairs(data2);

                JavaPairRDD<String, Integer> result = rdd1.subtractByKey(rdd2);
                System.out.println(result.collect());
        }


        public static void doFoldByKey(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data = Arrays.asList(new Tuple2("a", 1), new Tuple2("b",
1), new Tuple2("b", 1));

                JavaPairRDD<String, Integer> rdd = sc.parallelizePairs(data);

                JavaPairRDD<String, Integer> result2 = rdd.foldByKey(0, (Integer v1, Integer v2) -> v1 +
v2);
                System.out.println(result2.collect());
        }

        public static void doPipe(JavaSparkContext sc) {
                JavaRDD<String> rdd = sc.parallelize(Arrays.asList("1,2,3", "4,5,6", "7,8,9"));
                JavaRDD<String> result = rdd.pipe("cut -f 1,3 -d ,");
                System.out.println(result.collect());
        }

        public static void doCoalesceAndRepartition(JavaSparkContext sc) {
                JavaRDD<Integer> rdd1 = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 0), 10);
                JavaRDD<Integer> rdd2 = rdd1.coalesce(5);
                JavaRDD<Integer> rdd3 = rdd2.coalesce(10);
                System.out.println("partition size:" + rdd1.getNumPartitions());
                System.out.println("partition size:" + rdd2.getNumPartitions());
                System.out.println("partition size:" + rdd3.getNumPartitions());
        }
        public static List<Integer> fillToNRandom(int n) {
                ArrayList<Integer> rst = new ArrayList<>();
                Random random = new Random();
                return random.ints(n, 0, 100).boxed().collect(Collectors.toList());
        }

        public static void doRepartitionAndSortWithinPartitions(JavaSparkContext sc) {
                List<Integer> data = fillToNRandom(10);
                JavaPairRDD<Integer, String> rdd1 = sc.parallelize(data).mapToPair((Integer v) -> new
Tuple2(v, "-"));
                JavaPairRDD<Integer, String> rdd2 = rdd1.repartitionAndSortWithinPartitions(new
HashPartitioner(3));
                rdd2.count();

                rdd2.foreachPartition( it ->{
                                System.out.println("==========");
                                while (it.hasNext()) {
                                        System.out.println(it.next());
                                }
                });
        }

        public static void doPartitionBy(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data = Arrays.asList(new Tuple2("apple", 1), new
Tuple2("mouse", 1),new Tuple2("monitor", 1));
                JavaPairRDD<String, Integer> rdd1 = sc.parallelizePairs(data, 5);
                JavaPairRDD<String, Integer> rdd2 = rdd1.partitionBy(new HashPartitioner(3));
                System.out.println("rdd1:" + rdd1.getNumPartitions() + ", rdd2:" +
rdd2.getNumPartitions());
        }

        public static void doSortByKey(JavaSparkContext sc) {
                List<Tuple2<String, Integer>> data = Arrays.asList(new Tuple2("q", 1), new Tuple2("z",
1), new Tuple2("a", 1));
                JavaPairRDD<String, Integer> rdd = sc.parallelizePairs(data);
                JavaPairRDD<String, Integer> result = rdd.sortByKey();
                System.out.println(result.collect());
        }

        public static void doKeysAndValues(JavaSparkContext sc) {
                List<Tuple2<String, String>> data = Arrays.asList(new Tuple2("k1", "v1"), new
Tuple2("k2", "v2"),new Tuple2("k3", "v3"));
                JavaPairRDD<String, String> rdd = sc.parallelizePairs(data);
                System.out.println(rdd.keys().collect());
                System.out.println(rdd.values().collect());
        }


        public static ArrayList<Integer> fillToN(int n) {
```

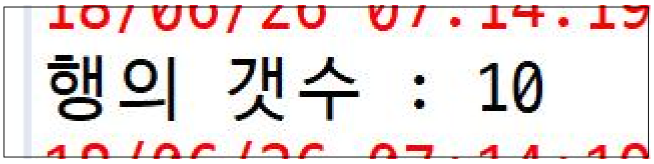
```
		ArrayList<Integer> rst = new ArrayList<>();
		for (int i = 0; i < n; i++)
			rst.add(i);
		return rst;
	}

	public static void doSample(JavaSparkContext sc) {
		List<Integer> data = fillToN(100);
		JavaRDD<Integer> rdd = sc.parallelize(data);
		JavaRDD<Integer> result1 = rdd.sample(false, 0.5);
		JavaRDD<Integer> result2 = rdd.sample(true, 1.5);
		System.out.println(result1.take(5));
		System.out.println(result2.take(5));
	}

	public static void doFirst(JavaSparkContext sc) {
		List<Integer> data = Arrays.asList(5, 4, 1);
		JavaRDD<Integer> rdd = sc.parallelize(data);
		int result = rdd.first();
		System.out.println(result);
	}

	public static void doTake(JavaSparkContext sc) {
		List<Integer> data = fillToN(100);
		JavaRDD<Integer> rdd = sc.parallelize(data);
		List<Integer> result = rdd.take(5);
		System.out.println(result);
	}

	public static void doTakeSample(JavaSparkContext sc) {
		List<Integer> data = fillToN(100);
		JavaRDD<Integer> rdd = sc.parallelize(data);
		List<Integer> result = rdd.takeSample(false, 4);

		result.forEach(x->System.out.println(x));
	}

	public static void doCountByValue(JavaSparkContext sc) {
		JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 1, 2, 3, 3));
		Map<Integer, Long> result = rdd.countByValue();
		System.out.println(result);
	}

	public static void doFold(JavaSparkContext sc) {
		List<Integer> data = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
		JavaRDD<Integer> rdd = sc.parallelize(data, 3);

		int result2 = rdd.fold(0, (Integer v1, Integer v2) -> v1 + v2);
		System.out.println(result2);
```

```
        }

        public static void doSum(JavaSparkContext sc) {
                List<Double> data = Arrays.asList(1d, 2d, 3d, 4d, 5d, 6d, 7d, 8d, 9d, 10d);
                JavaDoubleRDD rdd = sc.parallelizeDoubles(data);
                double result = rdd.sum();
                System.out.println(result);
        }

        public static void doForeach(JavaSparkContext sc) {
                List<Integer> data = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                JavaRDD<Integer> rdd = sc.parallelize(data);

                rdd.foreach((Integer t) -> System.out.println("Value Side Effect: " + t));
        }

        public static void doForeachPartition(JavaSparkContext sc) {
                List<Integer> data = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                JavaRDD<Integer> rdd = sc.parallelize(data, 3);

                rdd.foreachPartition(it -> {
                        System.out.println("Partition Side Effect!!");
                        it.forEachRemaining(v -> System.out.println("Value Side Effect:" + v));
                });
        }

        public static void doDebugString(JavaSparkContext sc) {
                JavaRDD<Integer> rdd1 = sc.parallelize(fillToN(100), 10);
                JavaRDD<Integer> rdd2 = rdd1.map((Integer v1) -> v1 * 2);
                JavaRDD<Integer> rdd3 = rdd2.map((Integer v1) -> v1 * 2);
                JavaRDD<Integer> rdd4 = rdd3.coalesce(2);
                System.out.println(rdd4.toDebugString());
        }

        public static void doCache(JavaSparkContext sc) {
                JavaRDD<Integer> rdd = sc.parallelize(fillToN(100), 10);
                rdd.cache();
                rdd.persist(StorageLevel.MEMORY_ONLY());
        }

        public static void doGetPartitions(JavaSparkContext sc) {
                JavaRDD<Integer> rdd = sc.parallelize(fillToN(1000), 10);
                System.out.println(rdd.partitions().size());
                System.out.println(rdd.getNumPartitions());
        }

}
```

■ 행동(Action) 메서드

1) count() : 갯수를 세는 메서드

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class RDDAPI2_count {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));

                System.out.println("행의 갯수 : "+rdd.count());
        }

}
```

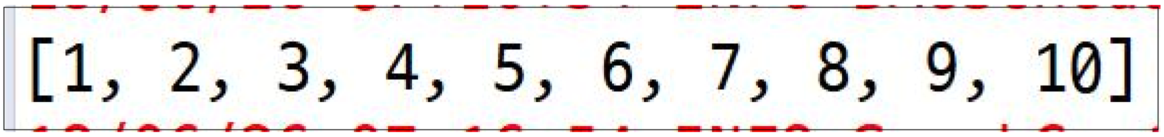


```
행의 갯수 : 10
```

2) collect() : 자바의 리스트로 변환해주는 메서드

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
```

```
public class RDDAPI1_collect {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {
                JavaRDD<Integer> rdd = sc.parallelize(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));

                //자바의 리스트로 변환시켜주는 메서드
                List<Integer> result = rdd.collect();

                for (Integer i : result) {
                        System.out.println(i);
                }//for end
        }

}
```



```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3) reduce 연산자 : 결과를 줄이는(맵리듀스의 리듀스와 같음) 연산

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

import scala.Tuple2;

public class RDDAPI11_reduce {

        private static JavaSparkContext sc;

        static {
                SparkConf conf = new SparkConf().setAppName("RDDAPIEx").setMaster("local");
                sc = new JavaSparkContext(conf);
        }

        public static void main(String[] args) {

                executeReduce();
                executeReduceByKey();

        }

        private  static void executeReduce() {
                List<Integer> data = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                JavaRDD<Integer> rdd = sc.parallelize(data, 3);

                int result = rdd.reduce((Integer v1, Integer v2) -> v1 + v2);
                System.out.println(result);
        }

        private  static void executeReduceByKey() {
                List<Tuple2<String, Integer>> data = Arrays.asList(
                                new Tuple2<String, Integer>("a", 0),
                                new Tuple2<String, Integer>("a", 1),
                                new Tuple2<String, Integer>("b", 1),
                                new Tuple2<String, Integer>("b", 2),
                                new Tuple2<String, Integer>("b", 3),
                                new Tuple2<String, Integer>("b", 4)
                                );

                JavaPairRDD<String, Integer> rdd = sc.parallelizePairs(data);

                JavaPairRDD<String, Integer> result2 = rdd.reduceByKey((Integer v1, Integer v2) ->
                {
                        System.out.println("v1:"+v1);
                        System.out.println("v2:"+v2);

                        return v1 + v2;
                }
                );
                System.out.println(result2.collect());
        }

}
```

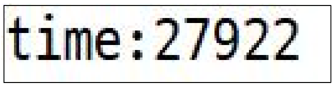
■ WordCount 비교

1) MyWordCount 직접 구현

```
package org.jbm.spark_0626;

import java.io.FileInputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class MyWordCount {

        public static void main(String[] args) throws Exception {

                long start = System.currentTimeMillis();

                FileInputStream fis = new FileInputStream("src/assets/test.txt");

                List<String> sList = new ArrayList<>();

                List<Integer> tmp = new ArrayList<>();

                Map<String, Object> result = new HashMap<>();

                int data = 0;

                int oldData = 0;

                while ((data = (byte)fis.read()) != -1) {

                        if (data != 32 && data != 13 && data !=10) {

                                tmp.add(data);

                        }else {

                                if((oldData!=32   &&   oldData!=13   &&   oldData!=10)   &&   (data==32   ||
data==13)) {

                                        Integer[] b = tmp.toArray(new Integer[tmp.size()]);

                                        byte[] bb = new byte[b.length];

                                        for (int i = 0; i < b.length; i++) {
                                                bb[i]=(byte)(int)b[i];
                                        }

                                        String s = new String(bb, "UTF-8");

                                        sList.add(s);

                                        Integer size = (Integer)result.get(s);

                                        if(size!=null) {
                                                result.put(s,size+1);
                                        }else {
                                                result.put(s, 1);
                                        }

                                        tmp.clear();
                                }

                        }//else

                        oldData = data;

                } // while end

                fis.close();

                Integer[] b = tmp.toArray(new Integer[tmp.size()]);

                byte[] bb = new byte[b.length];

                for (int i = 0; i < b.length; i++) {
                        bb[i]=(byte)(int)b[i];
                }

                String s = new String(bb, "UTF-8");

                sList.add(s);

                Integer size = (Integer)result.get(s);

                if(size!=null) {
                        result.put(s,size+1);
                }else {
                        result.put(s, 1);
                }

                PrintWriter out = new PrintWriter("src/assets/result.txt");

                Set<String> keys = result.keySet();
```

```
                for(String key : keys) {

                        out.println("("+key+","+result.get(key)+")");
                }

                out.close();

                long end = System.currentTimeMillis();

                System.out.println("time:"+(end-start));

        }
}
```

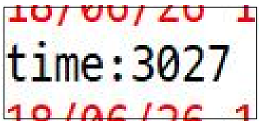
- 결과


```
time:27922
```

2) RDD로 구현

```
package org.jbm.spark_0626;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

import scala.Tuple2;

import java.util.Arrays;

public class WordCountRDD {

        public static void main(String[] args) throws Exception {

                SparkConf conf = new SparkConf().setAppName("WordCount").setMaster("local");

                // 1) SparkContext 생성
                JavaSparkContext sc = new JavaSparkContext(conf);

                Thread.sleep(2000);

                try {
                        long start = System.currentTimeMillis();

                        // 2) 입력 소스로부터 RDD 생성
                        JavaRDD<String> inputRDD = sc.textFile("src/assets/test.txt");

                        // 3) 한칸 띄우는 문자들로 나눠서 RDD 생성
                        JavaRDD<String>  words  =  inputRDD.flatMap((String  s)  ->  Arrays.asList(s.split("
")).iterator());

                        // 4) RDD를 PairRDD로
                        JavaPairRDD<String,    Integer>    wcPair    =    words.mapToPair(w    ->    new
Tuple2<String,Integer>(w, 1));

                        JavaPairRDD<String,  Integer>  result  =  wcPair.reduceByKey((Integer  c1,  Integer
c2) -> c1 + c2);

                        result.saveAsTextFile("src/result");

                        long end = System.currentTimeMillis();

                        System.out.println("time:"+(end-start));

                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        // Step5: Spark와의 연결 종료
                        sc.stop();
                }
        }

}
```

- 결과



```
18/06/26 1
time:3027
18/06/26 1
```

■ DataSet(DataFrame도 포함)

- RDD는 분산환경에서 메로리 기반으로 빠르고 안정적으로 동작
- RDD는 풍부한 데이터 처리 연산을 제공
- 하지만 데이터에 대한 메타데이터, ‘스키마’는 표현 못함
- 스파크 SQL은 RDD의 한계를 극복
- 스파크 SQL에는 1.SQL, 2. 데이터셋(DataSet) API가 있음
- 스파크 2.0부터 데이터 프레임 클래스가 데이터셋 클래스로 통합되면서 자바에서는 데이터셋 클래스만 사용

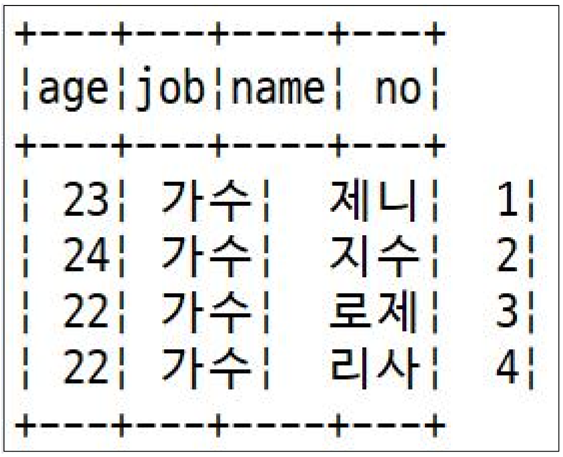
vo 생성 : Person.java

```
package vo;

import java.io.Serializable;

public class Person implements Serializable {
  private int no;
  private String name;
  private int age;
  private String job;

  public Person() {
  }

  public Person( int no, String name, int age, String job) {
        this.no= no;
    this.name = name;
    this.age = age;
    this.job = job;
  }
  public int getNo() {
        return no;
  }
  public void setNo(int no) {
        this.no = no;
  }
  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public String getJob() {
    return job;
  }

  public void setJob(String job) {
    this.job = job;
  }
}
```

1) DataFrame 생성

```
package org.jbm.spark_0626;

import java.util.*;

import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;

import vo.Person;

public class DatasetAPI_dataframe {

        private static  SparkSession spark;
        private static JavaSparkContext sc;


        public static void main(String[] args) {

                spark = SparkSession
                                .builder()
                                .appName("DataFrameSample")
                                .master("local[*]")
                                .getOrCreate();

                sc = new JavaSparkContext(spark.sparkContext());

                createDataframe();

                 spark.stop();
        }

        public static void createDataframe() {

                 Person row1 = new Person(1, "제니", 23, "가수");
                 Person row2 = new Person(2, "지수", 24, "가수");
                 Person row3 = new Person(3, "로제", 22, "가수");
                 Person row4 = new Person(4, "리사", 22, "가수");

                 //1. 로컬 컬렉션으로부터 데이터프레임 생성
                 List<Person> data = Arrays.asList(row1, row2, row3, row4);
                 Dataset<Row> df1 = spark.createDataFrame(data, Person.class);

                 //2. RDD를 이용해 데이터프레임 생성
                 JavaRDD<Person> rdd = sc.parallelize(data);
                 Dataset<Row> df2 = spark.createDataFrame(rdd, Person.class);

                 df1.show();
                 df2.show();
        }
}
```

- 결과



```
+---+---+----+---+
|age|job|name| no|
+---+---+----+---+
| 23| 가수|  제니|  1|
| 24| 가수|  지수|  2|
| 22| 가수|  로제|  3|
| 22| 가수|  리사|  4|
+---+---+----+---+
```

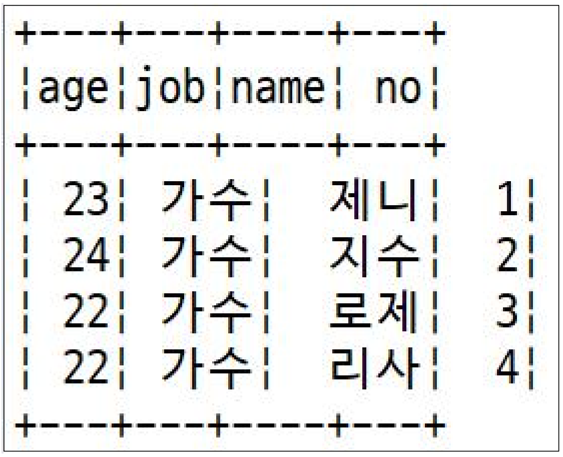
2) DataSet 생성

- 자바 객체를 이용해 생성

```
package org.jbm.spark_0626;

```

```
import java.util.Arrays;
import java.util.List;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.*;


import vo.Person;

public class DatasetAPI1_createDataSet{
        private static SparkSession spark;
        private static JavaSparkContext sc;

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());

                 createDataSet();

                 spark.stop();
        }
         public static void createDataSet() {
                 Person row1 = new Person(1, "제니", 23, "가수");
                 Person row2 = new Person(2, "지수", 24, "가수");
                 Person row3 = new Person(3, "로제", 22, "가수");
                 Person row4 = new Person(4, "리사", 22, "가수");

                 //자바 객체를 이용해 생성
                 List<Person> data = Arrays.asList(row1, row2, row3, row4);
                 Dataset<Person> ds= spark.createDataset(data, Encoders.bean(Person.class));

                 //결과 확인
                 ds.show();


         }
}
```

- 결과

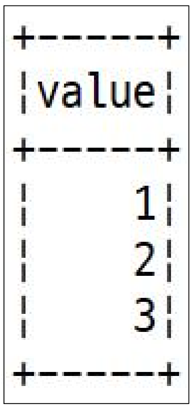


```
+---+---+----+---+
|age|job|name| no|
+---+---+----+---+
| 23| 가수|  제니|  1|
| 24| 가수|  지수|  2|
| 22| 가수|  로제|  3|
| 22| 가수|  리사|  4|
+---+---+----+---+
```

- RDD를 이용해 생성

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.*;

import vo.Person;

public class DatasetAPI2_createDataSet {
        private static SparkSession spark;
        private static JavaSparkContext sc;

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());

                 createDataSet();

                 spark.stop();
        }
         public static void createDataSet() {

                 //RDD를 이용해 생성
                 JavaRDD<Integer> javaRDD = sc.parallelize(Arrays.asList(1,2,3));
                 Dataset<Integer> ds = spark.createDataset(javaRDD.rdd(), Encoders.INT());

                 ds.show();

         }
}
```

- 결과

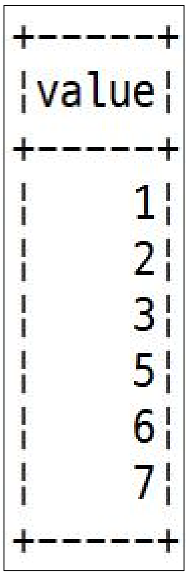


```
+-----+
|value|
+-----+
|    1|
|    2|
|    3|
+-----+
```

- 데이터 프레임을 이용해 생성

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.*;


import vo.Person;

public class DatasetAPI2_createDataSet {
        private static SparkSession spark;
        private static JavaSparkContext sc;

        public static void main(String[] args) {
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());
                createDataSet();

                spark.stop();
        }
         public static void createDataSet() {

                //RDD를 이용해 생성
                JavaRDD<Integer> javaRDD = sc.parallelize(Arrays.asList(1,2,3,4,5,6,7));
                Dataset<Integer> ds = spark.createDataset(javaRDD.rdd(), Encoders.INT());

                ds.show();

         }
}
```

- 결과

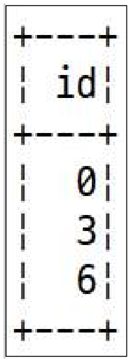


```
+-----+
|value|
+-----+
|    1|
|    2|
|    3|
|    5|
|    6|
|    7|
+-----+
```

3) Range : 범위에 포함되는 모든 숫자 리스트 생성

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.*;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
public class DatasetAPI4_createDataSet {
        private static SparkSession spark;
        private static JavaSparkContext sc;
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());

                 createDataSet();

                 spark.stop();
        }
         public static void createDataSet() {

                // range()로 생성

                 Dataset<Long> ds = spark.range(0, 7, 3);

                 //결과 확인
                 ds.show();
         }
}
```

- 결과

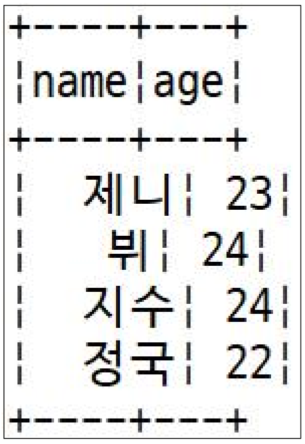


```
+---+
| id|
+---+
|  0|
|  3|
|  6|
+---+
```

4) Select

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.TypedColumn;

import vo.Person;

public class DatasetAPI_select {
        private static  SparkSession spark;
        private static JavaSparkContext sc;
        private static Dataset<Person> ds;

        public static void main(String[] args) {
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .config("spark.driver.host","192.168.0.112")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());
                 Person row1 = new Person(1, "제니", 23, "모델");
                 Person row2 = new Person(2, "뷔", 24, "댄서");
                 Person row3 = new Person(3, "지수", 24, "학생");
                 Person row4 = new Person(4, "정국", 22, "가수");
                 List<Person> data = Arrays.asList(row1, row2, row3, row4);
                 ds = spark.createDataset(data, Encoders.bean(Person.class));
                 runSelectEx();

                 spark.stop();

        }
         public static void runSelectEx() {
                //출력하고 싶은 컬럼만 지정해서 출력
                 TypedColumn<Object, String> c1 = ds.col("name").as(Encoders.STRING());
                 TypedColumn<Object, Integer> c2 = ds.col("age").as(Encoders.INT());
                 ds.select(c1, c2).show();
         }

}
```

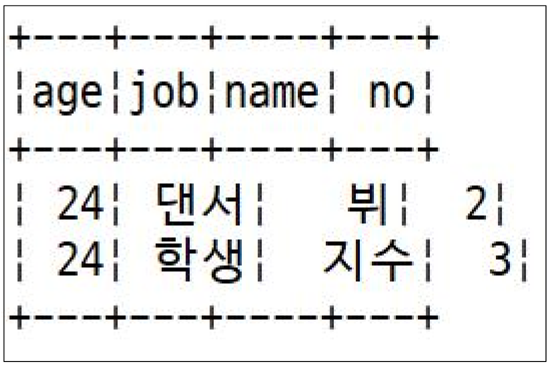
- 결과


```
+----+---+
|name|age|
+----+---+
|  제니| 23|
|   뷔| 24|
|  지수| 24|
|  정국| 22|
+----+---+
```

5) Where

```
package org.jbm.spark_0626;

import java.util.Arrays;
import java.util.List;

import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.TypedColumn;

import vo.Person;

public class DatasetAPI_select {
        private static  SparkSession spark;
        private static JavaSparkContext sc;
        private static Dataset<Person> ds;

        public static void main(String[] args) {
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .config("spark.driver.host","192.168.0.112")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());
                 Person row1 = new Person(1,"제니", 23, "모델");
                 Person row2 = new Person(2,"뷔", 24, "댄서");
                 Person row3 = new Person(3,"지수", 24, "학생");
                 Person row4 = new Person(4,"정국", 22, "가수");
                 List<Person> data = Arrays.asList(row1, row2, row3, row4);
                 ds = spark.createDataset(data, Encoders.bean(Person.class));
                 runSelectEx();
                spark.stop();

        }
         public static void runSelectEx() {
                ds.select("*").where(ds.col("age").equalTo(24)).show();
         }

}
```

- 결과

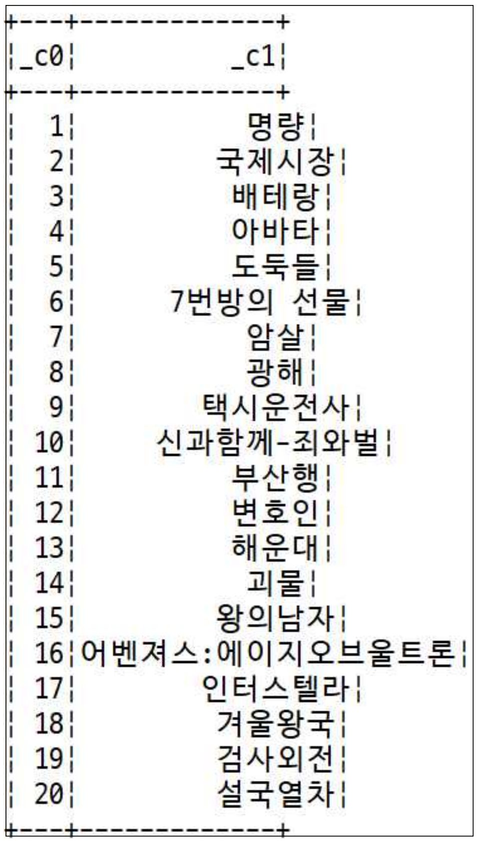


```
+---+---+----+---+
|age|job|name| no|
+---+---+----+---+
| 24| 댄서|   뷔|  2|
| 24| 학생|  지수|  3|
+---+---+----+---+
```

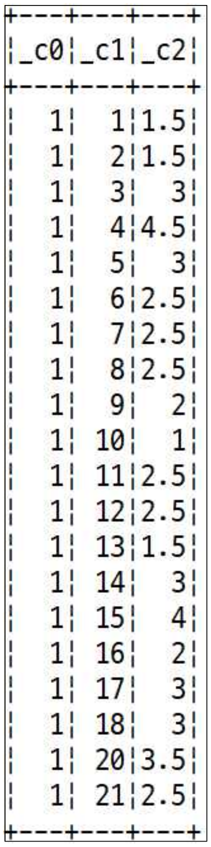
5) 외부에서 csv파일을 불러온 뒤 join

```
package org.jbm.spark_0626;

import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.functions;

public class DatasetAPI_join {
        private static  SparkSession spark;
        private static JavaSparkContext sc;
        private static Dataset<Row> films;
        private static Dataset<Row> filmRatings;

        public static void main(String[] args) {
                spark = SparkSession
                                .builder()
                                .appName("DatasetSample")
                                .master("local[*]")
                                .config("spark.driver.host","192.168.0.112")
                                .getOrCreate();
                sc = new JavaSparkContext(spark.sparkContext());

                 createDataSet();
                 runAsEx();
                 spark.stop();

        }

         public static void createDataSet() {
                String dir = "file:/C:/jbm/data/movie/";
                //파일로 데이터셋 생성
                films = spark.read().csv(dir+"films.csv").as("films");
                filmRatings = spark.read().csv(dir+"film_ratings.csv").as("filmRatings");
                //데이터셋 확인
                films.show();
                filmRatings.show();

         }
         public static void runAsEx() {
               //join
                films.join(filmRatings,functions.expr("films._c0 = filmRatings._c1")).show();
        }

}
```

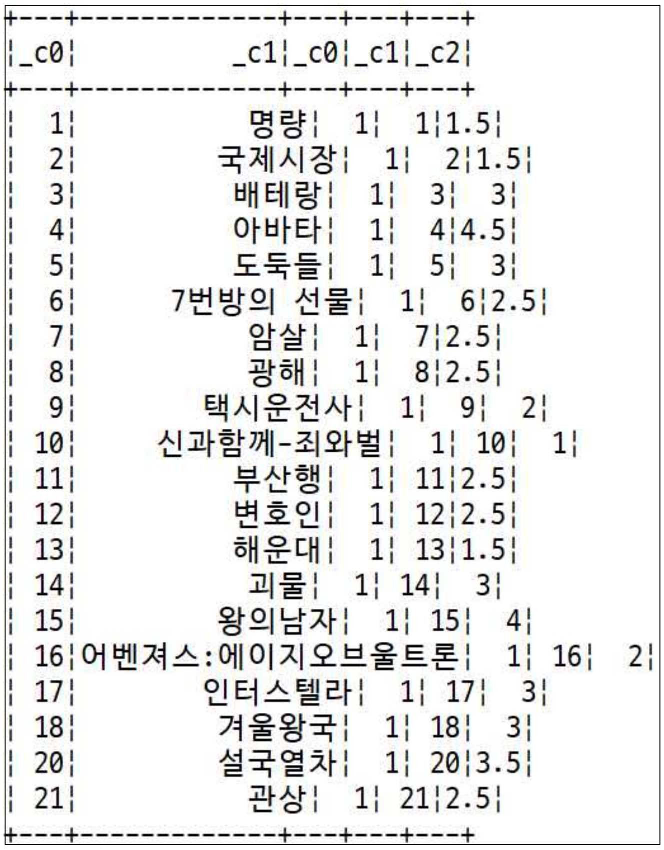
- 결과



```
+---+--------------+
|_c0|           _c1|
+---+--------------+
|  1|            명량|
|  2|          국제시장|
|  3|           배테랑|
|  4|           아바타|
|  5|           도둑들|
|  6|        7번방의 선물|
|  7|            암살|
|  8|            광해|
|  9|          택시운전사|
| 10|        신과함께-죄와벌|
| 11|           부산행|
| 12|           변호인|
| 13|           해운대|
| 14|            괴물|
| 15|          왕의남자|
| 16|어벤져스:에이지오브울트론|
| 17|          인터스텔라|
| 18|          겨울왕국|
| 19|          검사외전|
| 20|          설국열차|
+---+--------------+
```



```
+---+---+---+
|_c0|_c1|_c2|
+---+---+---+
|  1|  1|1.5|
|  1|  2|1.5|
|  1|  3|  3|
|  1|  4|4.5|
|  1|  5|  3|
|  1|  6|2.5|
|  1|  7|2.5|
|  1|  8|2.5|
|  1|  9|  2|
|  1| 10|  1|
|  1| 11|2.5|
|  1| 12|2.5|
|  1| 13|1.5|
|  1| 14|  3|
|  1| 15|  4|
|  1| 16|  2|
|  1| 17|  3|
|  1| 18|  3|
|  1| 20|3.5|
|  1| 21|2.5|
+---+---+---+
```



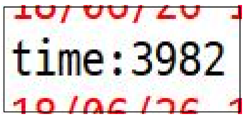
```
+---+--------------+---+---+---+
|_c0|           _c1|_c0|_c1|_c2|
+---+--------------+---+---+---+
|  1|            명량|  1|  1|1.5|
|  2|          국제시장|  1|  2|1.5|
|  3|           배테랑|  1|  3|  3|
|  4|           아바타|  1|  4|4.5|
|  5|           도둑들|  1|  5|  3|
|  6|        7번방의 선물|  1|  6|2.5|
|  7|            암살|  1|  7|2.5|
|  8|            광해|  1|  8|2.5|
|  9|          택시운전사|  1|  9|  2|
| 10|        신과함께-죄와벌|  1| 10|  1|
| 11|           부산행|  1| 11|2.5|
| 12|           변호인|  1| 12|2.5|
| 13|           해운대|  1| 13|1.5|
| 14|            괴물|  1| 14|  3|
| 15|          왕의남자|  1| 15|  4|
| 16|어벤져스:에이지오브울트론|  1| 16|  2|
| 17|          인터스텔라|  1| 17|  3|
| 18|          겨울왕국|  1| 18|  3|
| 20|          설국열차|  1| 20|3.5|
| 21|            관상|  1| 21|2.5|
+---+--------------+---+---+---+
```

▲ films ▲ filmRatings ▲ Join 결과

- WordCount를 DataSet으로 구현

```
package org.jbm.spark_0626;

import java.util.Arrays;
import org.apache.spark.sql.*;

import scala.Tuple2;

public class WordCountDataSet {

        public static void main(String[] args) throws Exception{

                SparkSession                                    spark                                          =
SparkSession.builder().master("local").appName("WordCount").getOrCreate();

                Thread.sleep(2000);

                long start = System.currentTimeMillis();

                Dataset<String> ds = spark.read().text("src/assets/test.txt").as(Encoders.STRING());

             Dataset<String> wordDF2 = ds.flatMap((String v) -> Arrays.asList(v.split(" ")).iterator(),
Encoders.STRING()).filter(s -> !s.isEmpty()).coalesce(1);
               Dataset<Tuple2<String,  Object>>  result2  =  wordDF2.groupByKey((String  value)  ->  value,
Encoders.STRING()).count();

            result2.toJavaRDD().saveAsTextFile("src/assets/result2");

                long end = System.currentTimeMillis();

                System.out.println("time:"+(end-start));

        }

}
```

- 결과



```
time:3982
```