Spark

数 据 嗨 客

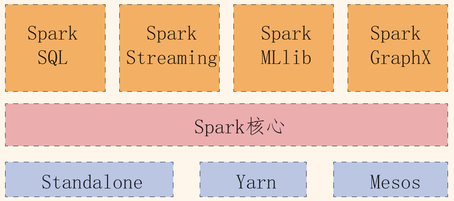
Spark是一种开源的集群计算框架，它将数据加载进内存中进行处理，适合机器学习算法的迭代运算.

Spark版本1.6.x，使用Python API.

```
from pyspark import SparkContext
from pyspark import SparkConf
```

基本概念

基本组件 | 在Hadoop HDFS的基础上，Spark提供数据分析的多种模块，应对不同使用场景下的数据处理需求.



Spark SQL | 基于结构化表格数据进行数据查询和处理.

Spark Streaming | 处理流式数据.

Spark MLlib | 基于Spark的机器学习算法库.

Spark GraphX | 基于图、网络数据进行数据处理.

Spark核心 | 基于结构化表格数据进行数据处理.

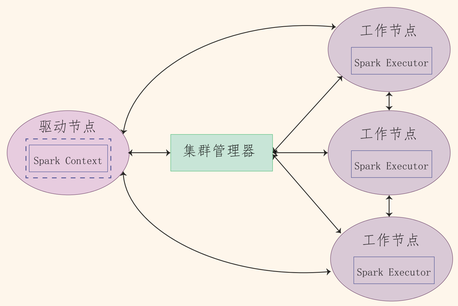
集群管理器 | 负责调度分配Spark的资源.

✓ Standalone，Spark内置的集群管理器；

✓ Yarn，Hadoop 2中的资源管理器；

✓ Mesos，一种通用的集群管理器，与Hadoop配合运行.

运行流程 | Spark程序在集群上以进程的形式独立运行，由SparkContext主导协调.



一个Spark程序由驱动程序和工作程序构成。在驱动节点，Spark创建Sparkcontext对象准备Spark的运行环境，通过集群管理器申请资源等；工作程序是工作节点中的一个进程，负责执行数据处理任务。

最后SparkContext将任务派给工作节点的Executor运行.

弹性分布式数据集 | Spark主要的数据类型，简写为RDD.

✓ 弹性. 数据分布在多个节点的内存中. 当节点的内存无法满足运算需求时，Spark可以将数据存放在硬盘中，避免内存不够而导致任务处理失败；

✓ 分布式. RDD将物理上分布在多个节点的数据集抽象为逻辑上完整的一个数据集.

Spark通过创建RDD将存储在磁盘中的数据加载进内存，构成RDD分区，然后调用RDD上的操作执行数据处理任务.

创建RDD | RDD的创建主要有以下三种方式：

✓将单机环境下的数据集并行化，转换为RDD对象；

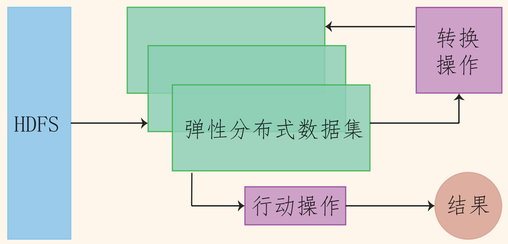
✓从HDFS分布式系统中读入数据，创建RDD对象；

✓由其他RDD对象经过转换操作生成新的RDD对象.

转换操作 | 构建多个RDD之间的逻辑关系，包括map、fliter、join和flatMap函数.

✓ 所有的转换操作都是延迟执行，会产生新的RDD. 新的RDD依赖于原来的RDD. 因此，数据分析流程构成了一个RDD组成的有向无环图.

行动操作 | 收集数据处理的结果，将其存储到外部存储系统，或者返回驱动节点.



初始化Spark

```
conf = SparkConf()
.setAppName(appName).setMaster(address)
```

首先创建SparkConf对象conf，配置Spark程序的名称appName，集群的地址address.

如果address取值为lcoal，那么Spark为单机本地模式.

`sc = SparkContext(conf=conf)` | 其次创建SparkContext对象sc，输入参数为SparkConf对象.

`sc.version`	查看SparkContext版本
`sc.pythonVer`	查看Python版本
`sc.appName`	查看Spark程序的名称
`sc.sparkHome`	查看工作节点中Spark的安装路径

使用Shell命令行运行Spark.

```
$./bin/pyspark --master address | 集群地址
               --py-file code.py | 代码文件
               --repository | 声明依赖
               --help | 查看帮助文档
```

RDD基本操作

加载数据

```
data = [2, 0, 1, 8, 0, 1, 0, 1]
rdd = sc.parallelize(data)
```

并行化已有数据，生成新的RDD对象dist_data.

`rdd_text = sc.textFile('data.txt')` | 加载外部文本数据，输入参数为文件地址，如hdfs://等.

`rdd_seq = sc.sequenceFile(path)` | 加载SequenceFile，输入参数为文件地址.

`sc.sequenceFile(path)`	加载sequenceFile
`sc.wholeTextFiles(dir)`	加载多个文本文件
`sc.pickleFile(path)`	加载pickle文件

统计数据信息

`rdd = sc.parallelize([2, 0, 1, 8])`

`rdd.getNumPartitions()` | 返回RDD的分区数.

`rdd.count()` | 返回RDD实例的个数.

`rdd.isEmpty()` | 判断RDD对象是否为空.

`rdd.countByKey()` | 按照Key返回RDD实例个数.

`rdd.countByValue()` | 按照value返回RDD实例个数.

`rdd.max()` | 最大值 `rdd.stdev()` | 标准差

`rdd.min()` | 最小值 `rdd.variance()` | 方差

`rdd.mean()` | 均值 `rdd.stats()` | 统计信息

切片数据

`rdd.first()` | 返回RDD的第一个元素.

`rdd.take(n)` | 返回RDD的前n个元素.

`rdd.collect()` | 返回RDD的元素列表

`rdd.distinct().collect()` | 返回不含重复RDD元素的列表 □

```
data = [('a',3),('c', 4), ('d', 2)]
rdd_kv = sc.parallelize(data)
rdd_kv.keys().collect()
```

返回RDD元素的key值列表$[a, c, d]$.

```
rdd1 = sc.parallelize(['a', 'b', 'c'])
rdd2 = sc.parallelize(['b', 'd']
```

`rdd1.union(rdd2)` | 求并集，[a, b, b, c, d].

`rdd1.intersection(rdd2)` | 求交集，[b].

`rdd1.subtract(rdd2)` | 从rdd1中移除rdd1和rdd2共有的元素，[a,c].

使用map、reduce等函数

`rdd.map(lambda x:x**2).collect()` | 将rdd中的元素求平方，[4,0,1,64,0,1,0,1].

`rdd_kv.filter(lambda x:'d' in x).collect()`

获取rdd_kv中key为'd'的元素，[('d', 2)].

```
rdd_kv.flatmap(lambda x:(x[1], x[0]))
      .collect()
```

将rdd_kv元素的key和value互换位置，并展平结果，[3, 'a', 4, 'c', 2, 'd'].

```
data1 = [('c',3),('c', 4), ('d', 2)]
rdd_kv1 = sc.parallelize(data1)
rdd_kv1.reduce(lambda x,y:x + y)
```

合并rdd_kv元素的value，['a', 3, 'c', 4, 'd', 2].

```
rdd_kv1.groupByKey()
       .mapValues(list).collect()
```

按key合并rdd_kv1元素，[('c',[3,4]),('d',2)].

`rdd_kv.sortBy(lambda x:x[1]).collect()`

按照value排序，[('d', 2),('a',3),('c', 4)].

RDD缓存

`rdd.persisit()` | 将rdd进行缓存，不再重复计算.

`rdd.cache()` | 将rdd缓存，缓存级别仅有MEMORY ONLY.

RDD分区

`rdd.repartition(5)` | 根据rdd生成5分区的RDD对象.

保存数据并停止

`rdd.saveAsTextFilefilter('rdd.txt')`

保存数据至文本文件.

`rdd.saveAsSequenceFile(path)`

保存为sequence文件.

`rdd.saveAsObjectFile(path)`

使用Java Serialization保存数据.

`rdd.saveAsPickleFile(path)` | 保存为picke文件.

`sc.stop()` | 停止运行.

Spark

数 据 嗨 客

MLlib的数据类型

在MLlib模块中，NumPy的ndarray和Python的list被认为是dense向量.MLlib自带的sparsVector和SciPy的csc_matrix（只有一列）为sparse向量.

Local向量|单机存储的数据类型，分为dense和sparse.

```
import numpy as np
from pyspark.mllib.linalg impoer Vectors
sv1 = Vectors.dense(np.array([0,2,0,4]))
sv2 = Vectors.dense([0,2,0,4])
sv3 = Vectors.sparse(4, 1:2,3:4)
```

创建长度为4的向量[0,2,0,4].

Labeled Point|带有标签或者响应的local向量.

```
from pyspark.mllib.regresson
import LabeledPoint
sv4 = LabeledPoint(1, [0,2,0,4])
```

生成标签为1的local向量[0,2,0,4].

Local矩阵|单机存储的矩阵类型，分为dense和sparse.

```
from pyspark.mllib.linalg import Matrix
from pyspark.mllib.linalg import Matrices
m1 = Matrices.dense(3, 2, [0,1,2,3,4,5])
m2 = Matrices.
    sparse(3,2, [0,1,3],[0,2,1],[9,6,8])
```

创建以下矩阵

$$m1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad m2 = \begin{bmatrix} 9 & 0 \\ 0 & 8 \\ 0 & 6 \end{bmatrix}$$

分布式矩阵 | 分布式得保存在一个或者多个RDD对象中的矩阵类型.

```
from pyspark.mllib.linalg.distributed
import RowMatrix
IndexedRow, IndexedRowMatrix, MatrixEntry
CoordinateMatrix, BlockMatrix
rows = sc.parallelize([[0, 1, 2], [3, 4,
5],[6, 7, 8]])
mat1 = RowMatrix(rows)
indexedRows = sc.parallelize([
              IndexedRow(0, [0, 1, 2]),
              IndexedRow(1, [3, 4, 5])])
mat2 = IndexedRowMatrix(indexedRows)
entries = sc.parallelize([
          MatrixEntry(0, 0, 1.2),
          MatrixEntry(1, 0, 2.1),
          MatrixEntry(6, 1, 3.7)])
mat3 = CoordinateMatrix(entries)
list1 = [1, 2, 3, 4, 5, 6]
list2 = [7, 8, 9, 10, 11, 12]
blocks = sc.parallelize([
  ((0, 0), Matrices.dense(3, 2, list1)),
  ((1, 0), Matrices.dense(3, 2, list2))])
mat4 = BlockMatrix(blocks, 3, 2)
```

上述代码分别创建4种类型的分布式矩阵

mat1	RowMatrix	行矩阵
mat2	IndexedRowMatrix	带行索引的行矩阵
mat3	CoordinateMatrix	坐标矩阵
mat4	BlockMatrix	块矩阵

MLlib基本统计

统计信息总结

```
from pyspark.mllib.stat import Statistics
stats = Statistics.colStats(mat)
```

mat为矩阵的一个RDD对象，生成统计报表summary，可以得到如下信息：

```
stats.count()  stats.max()  stats.normL1()
stats.mean()  stats.min()  stats.normL2()
stats.numNonzeros()   stats.variance()
```

相关关系

```
from pyspark.mllib.stat import Statistics
Statistics.corr(mat, method='pearson')
```

返回mat的相关系数矩阵，mat为矩阵的RDD对象.

假设检验

```
from pyspark.mllib.stat import Statistics
result = Statistics.chiSqTest(vec)
result1 = Statistics.chiSqTest(mat)
```

对vec进行拟合优度检验,vec为向量的RDD对象；对mat进行卡方检验，mat为矩阵的RDD对象.

分类回归算法

```
from pyspark.mllib.regression import
LinearRegressionWithSGD
model = LinearRegressionWithSGD
              .train(data)
pred = test_data.map(lambda x:(x.label,
                model.predict(x.features)))
```

线性回归模型，data和test_data为LabelPoint. 与回归模型的代码框架类似，Mllib支持以下模型（将线性模型替换为相应模型名称）：

逻辑回归模型	分类	LogisticRegressionWithLBFGS
支持向量机模型	分类	SVMWithSGD
朴素贝叶斯模型	分类	NavieBayes

```
from pyspark.mllib.tree
          import DecisionTree
model = DecisionTree.trainClassifier(
    data, numClasses=2,
    categoricalFeaturesInfo={},
impurity='gini', maxDepth=5, maxBins=32)
pred = model.predict(test_data.map(
          lambda x: x.features))
```

决策树分类模型，data和test_data为LabelPoint. 若决策树用于回归问题，则使用trainTregressor()方法. 与决策树模型的代码框架类似，Mllib支持以下模型（将决策树替换为相应模型名称）：

随机森林模型	分类	RandomForest
梯度提升树模型	分类	GradientBoostedTrees

聚类算法

```
from pyspark.mllib.clustering
          import KMeans
clusters = KMeans.train(data,
    culster_number, maxIterations=10,
    runs=10, initializationMode="random")
```

K均值模型，data为RDD对象.

```
from pyspark.mllib.clustering
import GaussianMixture
gmm = GaussianMixture.train(data, num)
```

高斯混合模型，data为RDD对象，参数num为类个数.

```
from pyspark.mllib.clustering import LDA
corpus = data.zipWithIndex()
     .map(lambda x:[x[1], x[0]]).cache()
model = LDA.train(corpus, k=5)
topics = model.topicsMatrix()
```

LDA主题模型，data为RDD对象.

特征抽取和转换

```
from pyspark.mllib.feature
import HashingTF, IDF, Word2Vec
htf = HashingTF()
tf = htf.transform(docs).cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

TFIDF模型，docs为RDD对象.

```
word2vec = Word2Vec()
model = word2vec.fit(docs)
synonyms = model.findSynonyms('data', 5)
```

Word2Vec模型，docs为RDD对象.

```
from pyspark.mllib.feature
import StandardScaler, Normalizer
label = data.map(lambda x: x.label)
features = data.map(lambda x: x.features)
sa = StandardScaler(withMean=True,
              withStd=True).fit(features)
sa.transform(features)
normalizer = Normalizer(p=float("inf"))
normalizer.transform(features)
```

数据标准化和数据归一化.

模型评价

```
import pyspark.mllib.evaluation as eval
```

输入数据为(score, label) 对.

✓ 二分类 eval.BinaryClassificationMetrics
areaUnderPR | areaUnderROC

✓ 多分类 eval.MulticlassMetrics
weightedFMeasure() | fMeasure()

输入数据为(prediction, observation) 对.

✓ 回归 eval.RegressionMetrics
meanSquaredError | meanSquaredError