Spark : Session et Context

Note: ref = `https://sparkbyexamples.com`

Session et Context

Pour demarrer une application spark, il faut creer une session spark et lui fournir un context spark.

La session represente l'objet d'interface avec le cluster.

Il contient le contexte contenant les parametres et configurations du cluster.

La Session

SparkSession

> C'est une instance de travail d'une application spark permettant d'interagir avec le cluster.

La Session

Le role de la session Spark est de fournir un point d'entrée à l'aplication.

C'est l'interface entre l'utilisateur et le cluster.

Il en existe 2 types:

- Spark Session SQL : pour les traitements batch.
- Spark Session Streaming : pour les traitements en temps réel.

La Session

C'est à partir de la session que sont chargées et manipulées les données.

Il existe sous forme d'un objet de type **SparkSession** importé du module **spark.sql**.

```
from pyspark.sql import SparkSession
```

La Session

La creation d'une session peut se faire :

- Automatiquement (par défaut, environnement de développement, ect...)
- Manuellement (environnement de production, ect...)

La Session

Creer à partir de la méthode **builder()** de la classe SparkSession.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
  .master("local[1]")\
  .appName("SparkSessionExample")\
  .getOrCreate()
```

La Session

```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
  # indique le nom du master selon le mode de deploiement
  # local[1] : mode local ou standalone
  # local[1] : le nombre correspond au nombre de coeurs donc de partitions RDD
  .master("local[1]")\
  # indique le nom de l'application
  .appName("SparkSessionExample")\
  # methode qui permet de recuperer la session
  # ou de la creer si elle n'existe pas
  .getOrCreate()
```

La Session

Il est possible de creer plusieurs sessions.

La methode **newSession()** permet de creer une nouvelle session possedant les memes configurations (nom,master, context) que la session courante.

```
from pyspark.sql import SparkSession
# creation d'une nouvelle session spark
spark2 = SparkSession.newSession
```

La Session

Par defaut, la SparkSession va creer :

- un objet **SparkContext**
- un objet **SparkConfig**
- tous les autres contextes, selon la configuration (HiveContext, StreamingContext, ect...).

La Session

Remarque :

- Dans de nombreux environements (pyspark-shell, databricks, ect...), la session est crée automatiquement avec son contexte.
- Elle peut alors etre utiliser directement à partir de l'objet **spark**.

```
# recuperation d'une session spark existante
spark = SparkSession.getOrCreate
```

La Session

Remarque :

- Sans indication contraire, le contexte est crée automatiquement à partir de la creation de la session.
- Il accessible à partir de l'objet **sc** ou la propriété **spark.sparkContext**.

```
# recuperation du contexte spark
sc = spark.sparkContext
```

La Session

Au besoin, des configurations peuvent etre ajoutées à la session.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
  .master("local[1]")\
  .appName("SparkSessionExample")\
  # Ajout de configurations avec la methode .config()
  .config("spark.some.config.option", "some-value")\
  .config("spark.some.other.config", "other-value")\
  .getOrCreate()
```

La Session

Les configurations peuvent etre ajoutées ou recuperer à partir de la propriété **spark.conf**.

```
# Ajout de configurations avec la methode .set()
spark.conf.set("spark.executor.memory", "6g")
# Recuperation de configurations avec la methode .get()
partitions = spark.conf.get("spark.sql.shuffle.partitions")
print(partitions)
```

La Session

Dans le cas d'utilisation de Hive (moteur de gestion de données distribuées), il est possible d'ajouter le contexte.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
  .master("local[1]")\
  .appName("SparkSessionExample")\
  # Ajout du contexte Hive
  .enableHiveSupport()\
  .getOrCreate()
```

La Session

Methode les plus communes :

```
# recupere la version de spark en cours d'utilisation
spark.version
# Creation d'un dataframe a partir de données (collection ou RDD)
spark.createDataFrame()
# recupere la session courante
spark.getActiveSession()
# retourne une instance de DataFrameReader,
# utiliser pour lire des fichiers en batch
spark.read()
# retourne une instance de DataStreamReader,
# utiliser pour lire des fichiers en streaming
spark.readStream()
# recupere le contexte spark
spark.sparkContext()
# recupere un dataframe apres execution de la requete SQL
spark.sql("requete sql")
# retourne une instance de SQLContext
spark.sqlContext()
# arrete le contexte spark courant
```

```
spark.stop()
# retourne un dataframe a partir d'une table ou d'une vue
spark.table()
# Cree une fonction utilisateur (UDF) pour l'utiliser
spark.udf()
```

Le contexte

SparkContext

> C'est un objet de la SparkSession qui contient les configurations et informations de l'application spark.

Le contexte

Le contexte est accessible à partir de la propriété **spark.sparkContext** lorsqu'il n'est pas crée automatiquement dans l'objet **sc**.

```
# recuperation du contexte spark
sc = spark.sparkContext
```

Le contexte

Le SparkContext contient toutes les configurations de l'application.

C'est lui qui soumet les taches au cluster.

C'est egalement lui qui recupere les informations sur l'etat du cluster.

Le contexte

Le SparkContext est liée à l'execution de spark par la JVM (Java Virtual Machine).

Il ne peut y en avoir qu'**une seule instance** en meme temps !

Si il y a plusieurs sessions en cours, elles partagent le meme contexte.

Si plusieurs contextes sont crées, un message d'erreur est affiché :

```
ValueError: Cannot run multiple SparkContexts at once;
```

Le contexte

Il est possible d'arreter le contexte avec la methode **stop()**.

```
# arrete le contexte spark courant
spark.sparkContext.stop()
```

ATTENTION : l'execution va declancher un message dans les logs spark qui n'est plus utilisable tant qu'un context n'as pas été recrée.

```
INFO SparkContext: Successfully stopped SparkContext
```

Le contexte

Variables et methodes les plus communes :

```
# Variable qui contient un id unique à l'application
spark.sparkContext.applicationId
# Variable qui contient la version du cluster
spark.sparkContext.version
# Variable qui contient l'URL de l'interface web de Spark
spark.sparkContext.uiWebUrl
# Methode pour creer une variable accumulator avec une valeur initiale
# variable du driver qui sert à stocker des resultats intermediaires
spark.sparkContext.accumulator(value[, accum_param])
# Creer une variables broadcast et la distribue à tous le cluster.
# variable en lecture seule qui ne peut etre distribué qu'une fois.
spark.sparkContext.broadcast(value)
# Methode pour creer un RDD vide
spark.sparkContext.emptyRDD()
# Methode pour creer et recuperer un SparkContext
spark.sparkContext.getOrCreate()
# Methode pour recuperer un RDD des fichiers hadoop
spark.sparkContext.hadoopFile()
# Methode pour recuperer un RDD des fichiers hadoop
# avec une clé donnée et une valeur typée
spark.sparkContext.sequenceFile()
# Methode pour creer un RDD pour un fichier Hadoop
# avec un nouveau format InputFormat
spark.sparkContext.newAPIHadoopFile()
# Methode pour changer le niveau de log (debug, info, warn, fatal, error)
spark.sparkContext.setLogLevel()
# Methode pour lire et recuperer un RDD depuis un fichier texte (locale ou HDFS)
spark.sparkContext.textFile()
# Methode pour faire une union de deux RDD
spark.sparkContext.union()
# Methode similaire à textFile() mais qui lit un fichier texte
# directement en tant que dossier de fichiers distribués (HDFS)
spark.sparkContext.wholeTextFiles()
```

Note:

- une variable broadcast est une variable distribué à tous les noeuds du cluster.
- woleTextFiles() garde la liste des shards de fichiers et leur contenu. à la difference de textFile() qui regroupe en un seul RDD.

Le RDD et le Dataframe