**UNIVERSITA DEGLI STUDI DI MILANO BICOCCA**

Elaborato Complimenti Basi di dati

**RELAZIONE PROGETTO SPARK**

**Studenti**

Fabio Maiocchi 762486

Hugo Aldo Reynoso Quispe 765261

**Introduzione**

Il progetto consiste nello sviluppo di una piccola applicazione utilizzando la tecnologia Spark, questa applicazione dovrà:

- profilare dati RDF (contare numero e tipo di proprietà)

- definire frequenze di parole all’interno di testi

Per lo sviluppo del progetto abbiamo approfondito alcuni argomenti come i big data, l’utilizzo del framework Apache spark, RDF/xml , e utilizzato le conoscenze pregresse del linguaggio di programmazione java.

**BIG DATA**

**Introduzione ai Big Data**

Si parla di Big data quando si ha un dataset talmente grande da richiedere strumenti non convenzionali per estrapolare, gestire e processare informazione entro un tempo ragionevole. Non esiste una dimensione di riferimento, ma questa cambia sempre, poiché le macchine sono sempre più veloci e i dataset sono sempre più grandi.

I Big data si distinguono per le seguente caratteristiche “le quattro V “:

* **Volume**: fa riferimento alla capacità di acquisire, memorizzare ed accedere a grandi volumi di dati, non gestibili con i database tradizionali;
* **Velocita**: è riferita al fatto che l’analisi dei dati deve essere effettuata a ritmo sostenuto o addirittura in tempo reale;
* **Varietà**: riferita alle varie tipologie di dati, provenienti da fonti diverse (strutturate e non), come per esempio: audio, video, web, social network, smartphone, ecc;
* **Variabilità**: le stesse informazioni hanno significato differente a seconda del contesto e del “luogo virtuale” in cui vengono reperite. Occorre contestualizzare il dato, in modo da capire se è indispensabile filtrarlo o meno.



Figura 1: Big data 4v

**Provenienza dei Big data**

Le fonti di generazione di dati potenzialmente utilizzabili sono molteplici e tra loro eterogenee, possiamo suddividerli come:

**Dati strutturati**: I dati rappresentati in un formato tabellare all’interno di un database relazionale sono dati strutturati.  
**Dati semi strutturati**: Sono dati che presentano parte dei loro dati in formato strutturato.  
**Dati non strutturati**: Sono dati che non possiedono una struttura predefinita, non conducibili ad un formato tabellare.

Alcuni esempi:

* **Social networks e social media (Twitter, Facebook, blogs, forum, etc.);**
* **Documenti cartacei digitalizzati;**
* **Dati di geo-posizionamento (GPS);**
* **Dati generati da trasmettitori e sensori (cellulari, wifi, bluetooth, Rfid, NFC, etc.);**
* **M2M (Machine to Machine) data – Internet of Things;**
* **Automazione processi produttivi;**
* **Database, registrazioni video e audio, immagini, email, ecc.**

Alcuni studi realizzati, come quello della Emc Digital Universe nel 2013, che fotografa e stima la quantità di dati digitali prodotti in un anno. Prevedono una crescita dell’universo digitale di dieci volte tra il 2013 e il 2020 (da 4,4 a 44 zettabyte). Grazie all’impatto delle tecnologie wireless e alla diffusione di prodotti intelligenti (IoT).

Inoltre grazie allo sviluppo di tecnologie come: Cloud computing, machine learning, l’industria 4.0, e gli Opendata, Crescono anche i big data è faciliteranno la loro implementazione e il loro utilizzo.

Eric schmidt, presidente del consiglio di amministrazione di google, cosi affermava nel 2010: *“Dall’alba della civiltà al 2003, l’umanità ha registrato 5 exabytes di dati. Oggi la stessa quantità di dati e registrata ogni due giorni”.*

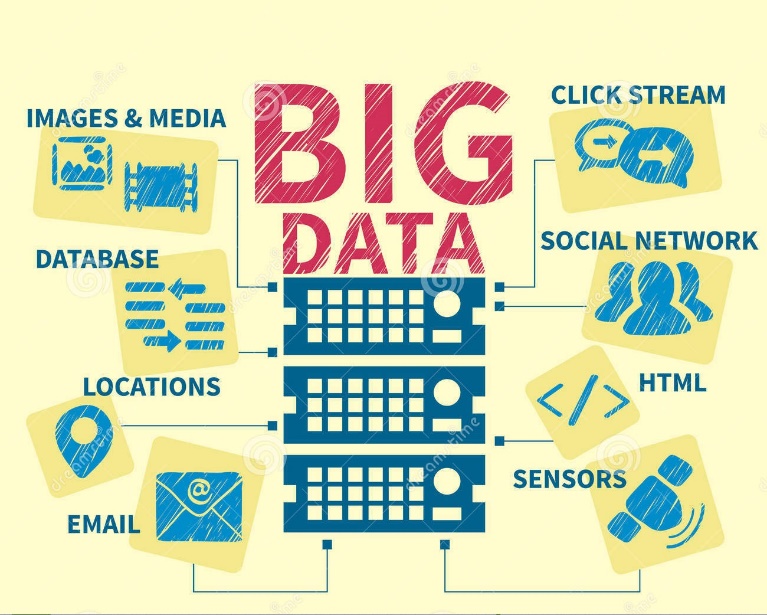


Figura 2: Big data source

**RDF**

**Resource Description Framework**

RDF (Resource Description Framework) è uno standard proposto dal W3C come set di linguaggi dichiarativi basato su sintassi XML ed adatto a descrivere la struttura di una parte della realtà.

Per realtà intendiamo qualsiasi risorsa sia possibile identificare sulla rete con un indirizzo univoco, mentre per descrizione indichiamo l’insieme delle proprietà, degli attributi e delle relazioni con altre realtà.

RDF è lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati, e consente l’interoperabilità tra applicazioni che si scambiano sul Web informazioni.

L'RDF si basa su tre principi chiave:

* Qualunque cosa può essere identificata da un Universal Resource Identifier (URI).
* The least power: utilizzare il linguaggio meno espressivo per definire qualunque cosa.
* Qualunque cosa può dire qualunque cosa su qualunque cosa.

**Rappresentazione fisica del modello**

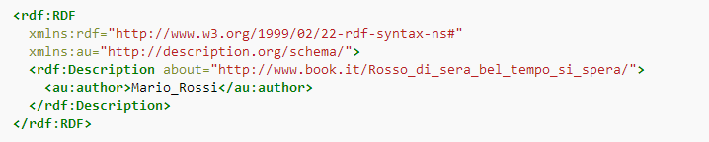
Un modello RDF è rappresentabile da un grafo orientato sui cui nodi ci sono risorse o tipi primitivi e i cui archi rappresentano le proprietà. Un grafo RDF è rappresentato fisicamente mediante una serializzazione.

Le principali serializzazioni adottabili per un grafo RDF sono:

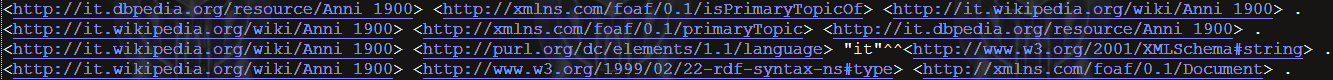
* RDF/XML: documento RDF è serializzato in un file XML.
* N-Triples: serializzazione del grafo come un insieme di triple soggetto - predicato - oggetto.
* Notation3: serializzazione del grafo descrivendo, una per volta, una risorsa e tutte le sue proprietà.

Esempio serializzazione RDF/XML:

vogliamo serializzare la frase "Mario\_Rossi" "è\_autore\_di" "Rosso\_di\_sera\_bel\_tempo\_si\_spera"



Esempio ttl:



**SPARK**

**Definizione Apache Spark**

Spark è un framework open-source per l’analisi di grandi quantità di dati su cluster, nato per essere veloce e flessibile. Caratterizzato dalla capacità di memorizzare risultati in memoria centrale si offre come valida alternativa a Hadoop MapReduce il quale memorizza obbligatoriamente i risultati delle computazioni su disco.

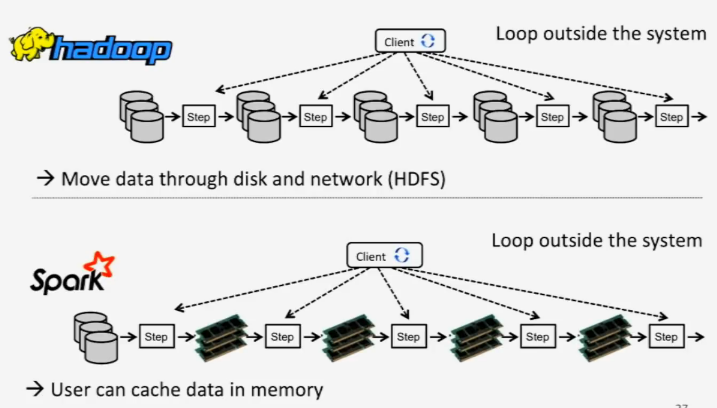


Figura 3: Hadoop vs Spark

Nel suo core, Spark offre un motore di computazione che viene sfruttato da tutti gli altri suoi componenti. Ciò significa che un’ottimizzazione al core di Spark si manifesta su tutte le applicazioni che sfruttano questo framework. Il core di Spark si occupa delle funzionalità base, come lo scheduling, la distribuzione e il controllo dell’esecuzione dell’applicazione utente. A sfruttare il core ci sono vari tools.

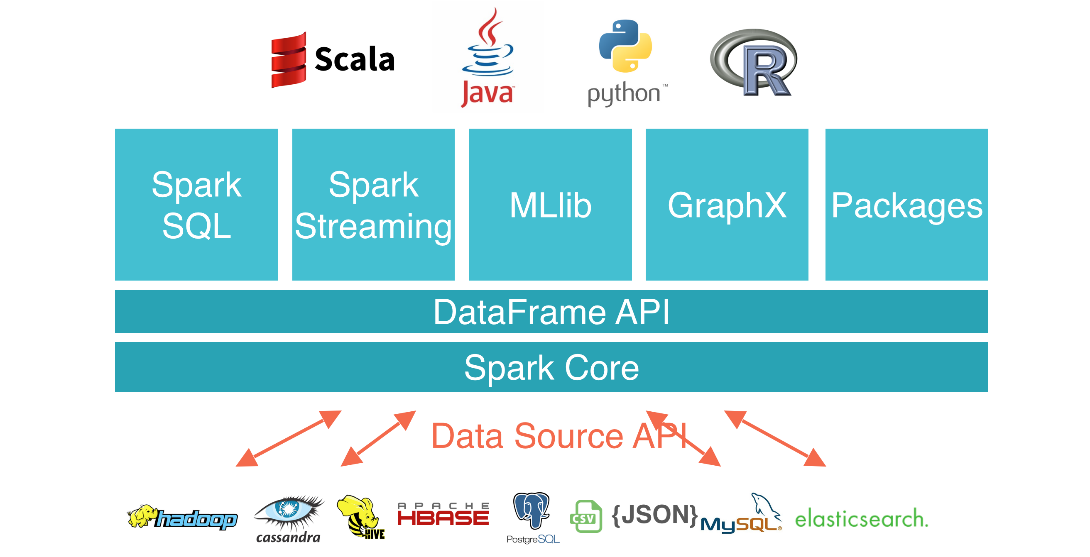


Figura 4: spark ecosystem

Spark è implementato in Scala , un linguaggio di programmazione ad alto livello che integra caratteristiche sia dei linguaggi funzionali sia della programmazione orientata agli oggetti. Spark fornisce le API per i linguaggi Scala, Java e Python, e offre dei preziosi tool di sviluppo come:

* **Spark SQL**: per creare delle query con Spark
* **MLlib**: una libreria ricca di algoritmi e strumenti classici per il machine learning.
* **Spark Streaming**: Spark Streaming permette di analizzare flussi di dati in tempo reale, ad esempio log di errori o un flusso di tweet.
* **GraphX**: è una libreria per l’analisi di grafi talmente grandi che non potrebbero essere analizzati su una singola macchina (ad esempio i grafi dei social network).

**Archittetura Spark**

Spark può lavorare sia in un singolo nodo che in cluster. Nel caso generale vi sono una serie di processi in esecuzione per ogni applicazione Spark: un driver e molteplici executor. Il driver è colui che gestisce l’esecuzione di un programma Spark, decidendo i compiti da far svolgere ai processi executor, i quali sono in esecuzione nel cluster. Il driver, invece, potrebbe essere in esecuzione nella macchina client. Nel programma principale di un’applicazione Spark (il programma driver) è presente un oggetto di tipo SparkContext, la cui istanza comunica con il gestore di risorse del cluster per richiedere un insieme di risorse (RAM, core, ecc.) per gli executor. Sono supportati diversi cluster manager tra cui YARN, Mesos, EC2 e lo Standalone cluster manager di Spark.

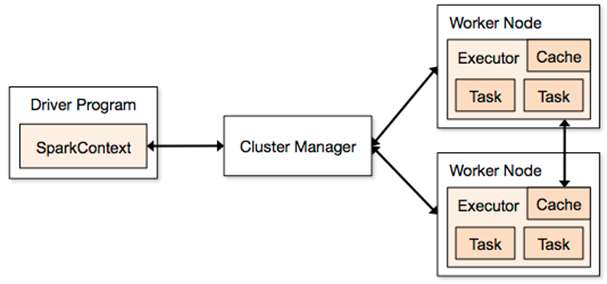


Figura 5: spark architecture

**Resilient distribuited dataset (RDD)**

Spark si basa sul concetto di resilient distribuited dataset (RDD), che è una collezione di elementi che possono essere elaborati in parallelo e che possiede le proprietà di fault tolerance. Gli RDD vengono memorizzati in maniera distribuita, spezzata in parti chiamate partizioni, ognuna delle quali sta in un esecutore di Spark. Per processare i dati, il sistema creerà un task per ogni partizione. Gli RDD possono contenere elementi facenti parte di una qualsiasi classe Java, anche definita da un utente. Per capire il meccanismo di funzionamento di un RDD, si consideri il seguente esempio:

JavaRDD<String> linee = sc.textFile("data.txt “) ;

JavaRDD<Integer> lunghezzaLinee = linee.map( s -> s.length ( ) ) ;

int lunghezzaTotale = lunghezzaLinee.reduce ( ( a , b) -> a + b ) ;

La prima riga definisce un RDD contenente stringhe a partire da un file esterno. Tramite il metodo textFile, ogni riga del file di partenza diventa un elemento di un RDD contenente stringhe. Il file non viene toccato a questo punto, in quanto non sono stati richiesti risultati. La variabile linee è quindi una sorta di puntatore al file. La seconda linea trasforma l’RDD di partenza creandone un altro: per ogni riga del file di partenza, si ottiene un intero contenente la lunghezza della linea. L’ultima riga calcola attraverso il metodo reduce la lunghezza totale del file.

Solo a questo punto vengono svolti calcoli paralleli, in quanto il metodo reduce è un’azione. Per svolgere la computazione, il file data.txt viene caricato parallelamente dai nodi del cluster, e ogni macchina esegue una parte del metodo map (che è una trasformazione). A questo punto ogni macchina fa una reduce locale e manda il risultato al programma driver di Spark, che si preoccuperà di raccogliere i risultati parziali e di sommarli per ottenere la lunghezza totale del file. Il motivo per il quale i dati vengono letti solo alla richiesta di un’azione è che non sempre l’intero file è necessario per il risultato. Supponendo di volere ottenere in output le righe con più di 100 caratteri, solo queste saranno tenute in memoria durante la lettura del file. Dopo il calcolo, il dataset non rimane in memoria nei nodi del cluster. Se ci fosse servito per una futura operazione, avremmo dovuto chiamare il metodo persist sull’oggetto lunghezzaLinee prima di chiamare il metodo reduce. In alternativa il dataset verrebbe ricalcolato ad ogni azione il cui risultato dipende da esso. Se il dataset è molto grande e di difficile ricomputazione è anche possibile memorizzarlo su disco.

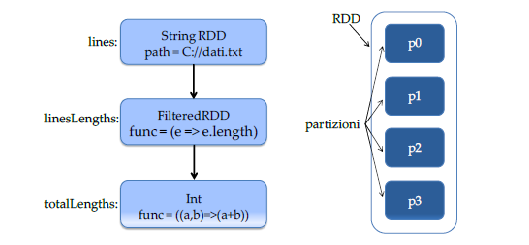


Figura 5: RDD

**PROGETTO SPARK**

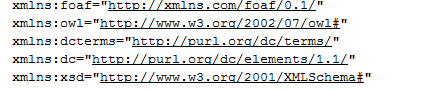
**Riassunto**

L’obiettivo di questo progetto era l’analisi e l’implementazione di una piccola applicazione che permettesse di contare il numero e tipo di proprietà dato come input un file in formato RDF, e definire frequenze di parole all’interno di testi.

Abbiamo creato una variabile chiamata Logdata di tipo RDD, che contiene il nostro file , il quale andremo ad analizzare.

**Analisi RDF**

Abbiamo creato un metodo extract\_prefix che estrae i prefissi che si trovano nel file. Esempio:



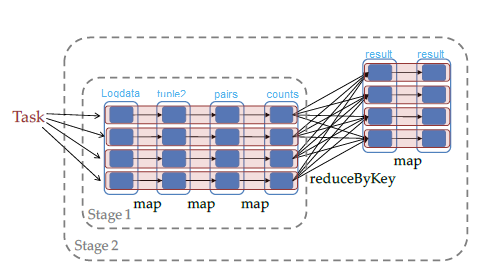
Per contare quante proprietà ha il file RDF abbiamo dovuto creare il metodo count\_property per analizzare le singole righe e con l'aiuto del metodo extract\_property vengono estratte le proprietà riconosciute dai prefissi.

**Analisi Testo**

Abbiamo creato il metodo refine\_text per eliminare la punteggiatura e isolare le singole parole.

**Conta Risultati**

Dopo di che, il metodo count, viene utilizzato per contare i risultati dei metodi precedenti salvati in un RDD. Inizialmente viene creato un pairRDD con valore 1 per ogni elemento, poi gli elementi con chiavi uguali vengono uniti e il loro valori vengono sommati. Infine il risultato dell'unione degli elementi uguali viene salvato e restituito.



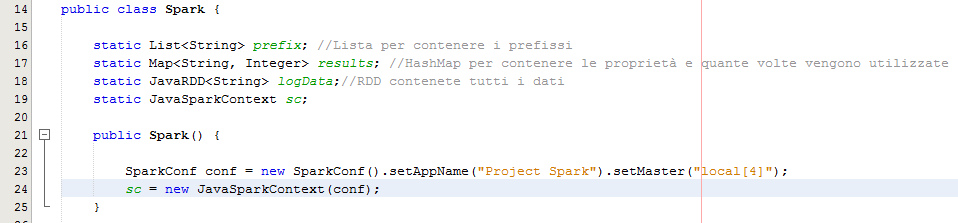
**DETTAGLIO PROGETTO**

Nella nostra Classe Spark vengono dichiarate le variabile prefix, results, logdata, sc.

**Public Spark()**

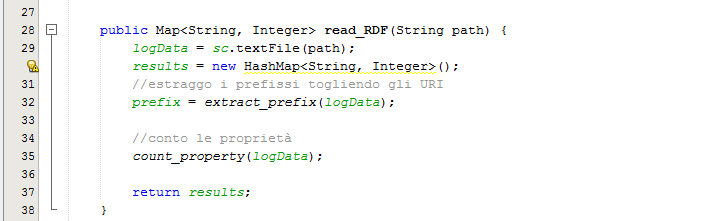
**SparkConf**: istanza un nuova sparkconf chiamato “conf” per impostare il nome della applicazione e abilitare il registro degli eventi inoltre, visto che lavoriamo in locale, dobbiamo impostare il setMaster = “local[4]” con il numero di core posseduti dal nostro pc.

**JavaSparkContext:** Crea un nuovo JavaSparkContext “sc” con la configurazione definita in “conf”, nel momento in cui viene creato lo SparkContext si definisce l’ambiente di lavoro relativo all’applicazione: quanti sono gli executor, quanta memoria associare ad ogni executor, quanto core usare per ogni macchina e tutti gli eventi e parametri definiti in sparkconf.



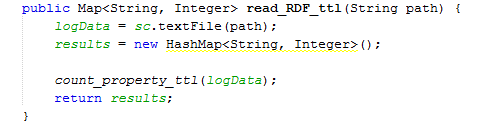
**Public Map<String, Integer>read\_RDF(String path)**

Metodo che riempie Logdata con i dati di un file in formato RDF/xml, inizializza la variabile results per i nuovi risultati, riempie la lista prefix utilizzando il metodo extract\_prefix, ed avvia il conto delle proprietà



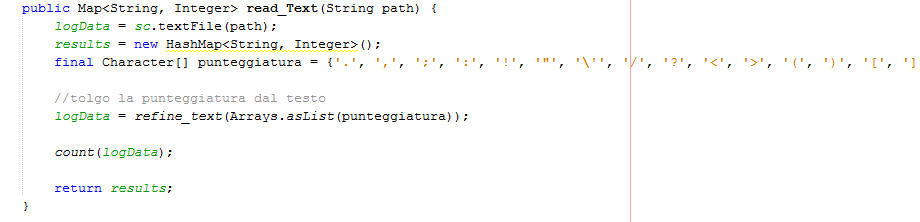
**public Map<String, Integer> read\_RDF\_ttl(String path)**

Metodo che riempie Logdata con i dati di un file in formato RDF/ttl, inizializza la variabile results per i nuovi risultati, ed avvia il conto delle proprietà.

****

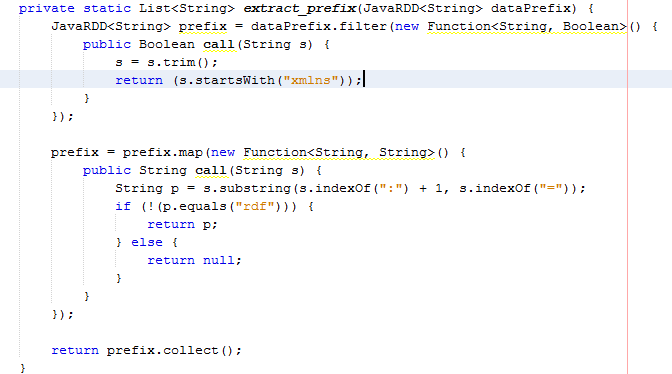
**public Map<String, Integer> read\_Text(String path)**

Metodo che riempie il Logdata con un file testo, inizializza la variabile results, crea un array di caratteri contenete la punteggiatura e i simboli da eliminare, richiama il metodo refine\_text per eseguire operazioni sul testo, ed avvia il conto delle parole.

****

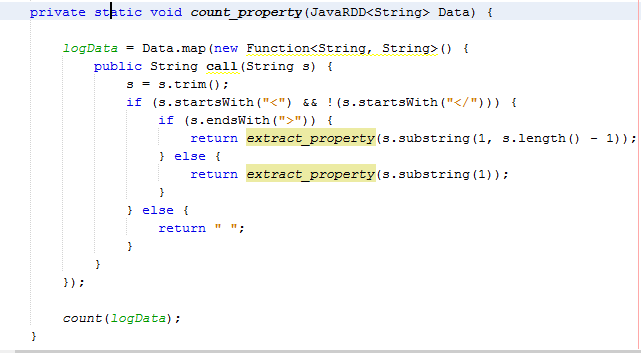
**private static List<String> extract\_prefix(JavaRDD<String> dataPrefix)**

Il metodo che estrae i prefissi, nel nostro caso i prefissi sono quelli che vengono preceduti dalla scritta xmlns, viene escluso il prefisso "rdf" perchè non rappresenta una proprietà.



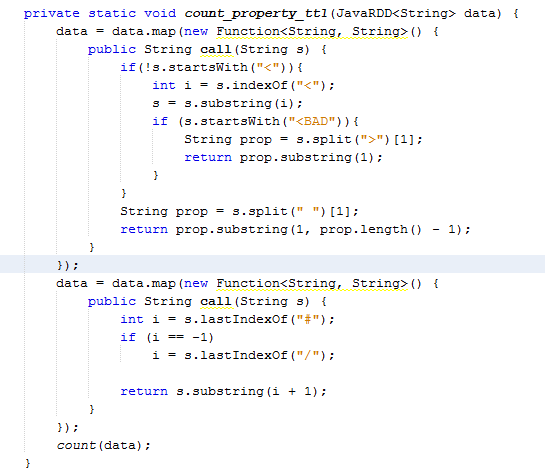
**private static void count\_property(JavaRDD<String> Data)**

Metodo che prepara le singole stringhe distinguendo i tag normali ("<"), i tag di chiusura ("</") e i tag mal indentati, cioè le stringhe che non iniziano con <. Le singole stringhe vengono passate a extract\_property che le analizza. Infine il risultato viene passato a count per contare le proprietà.



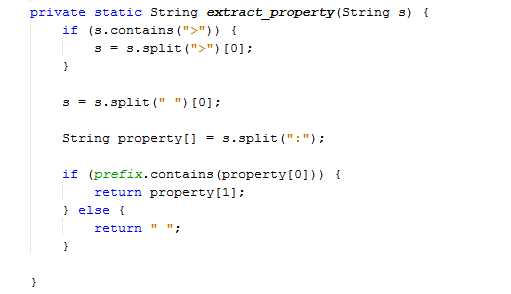
**private static void count\_property\_ttl(JavaRDD<String> data)**

Metodo che prepara le singole stringhe, estraendo il tag centrale che contiene la proprietà, dopo di che si analizza la stringa ottenuta prendendo il nome e escludendo il prefisso. I risultato viene passato a count che conta le proprietà.



**private static String extract\_property(String s)**

Il metodo che estrae le proprietà, che si trova all'inizio del tag, quindi la stringa viene splittata prima in base alla presenza del simbolo “>”, poi in base agli spazi e infine in base a ":" in modo da distinguere il prefisso e il nome.



**private static void count (JavaRDD<String> data)**

**JavaPairRDD**=permette di lavorare con coppie (chiave, valori)

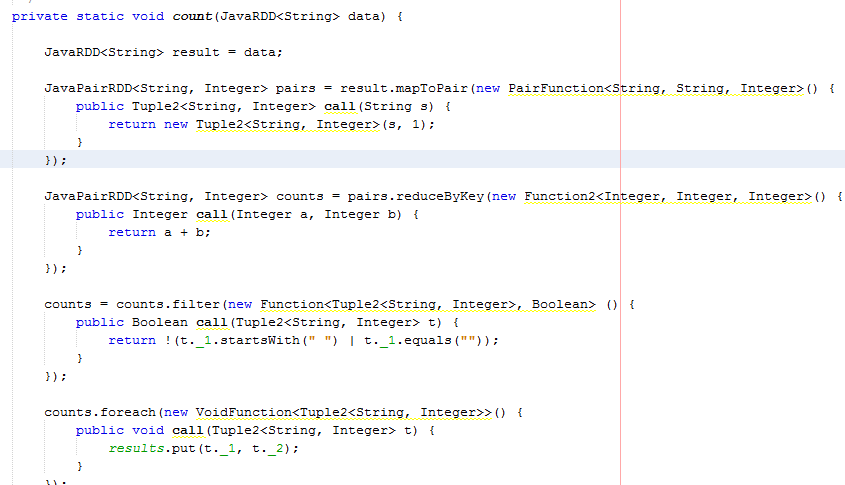
**MapToPair**= restituisce un nuovo RDD applicando una funzione ad ogni partizione di questo RDD

**Tuple2**= è tipo di dato presente nel linguaggio scala, si crea semplicemente scrivendo (key, value).

**ReduceBykey**=otteniamo un RDD che contiene, per ogni chiave presente in pairs, la somma dei valori associati a quella chiave.

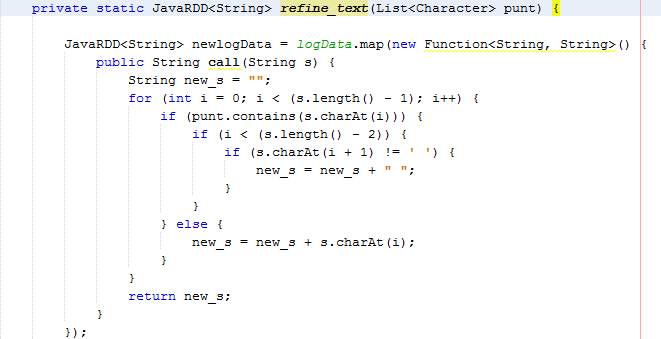
**Filter**= vengono mantenuti quelli elementi per cui e valida una certa condizione

Metodo per contare gli elementi di un RDD. Inizialmente viene creato un pairRDD con valore 1 per ogni elemento, poi gli elementi con chiavi uguali vengono uniti e il loro valori vengono sommati. Infine il risultato dell'unione degli elementi uguali viene messo in results.

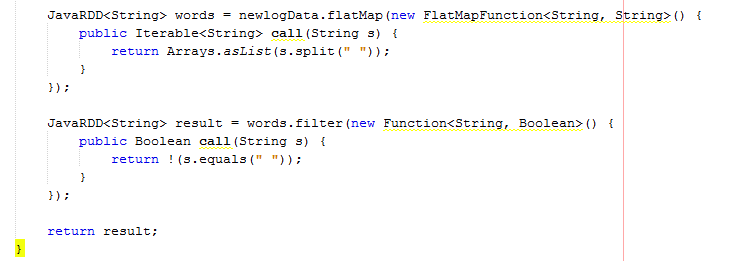


**private static JavaRDD<String> refine\_text(List<Character> punt)**

Metodo che prende le singole stringhe del testo e ne ricrea delle nuove escludendo punteggiatura ed altri caratteri speciali,



Dopo di che vengono divise le parole in base agli spazi e si escludono gli elementi vuoti che potrebbero essere creati dalla divisione.



**Ambiente di lavoro**

La macchina utilizzata per realizzare il progetto ha le seguenti caratteristiche:

* Sistema operativo: Linux Mint 17 Cinnamon 64 bit
* Versione Cinnamon 2.2.16
* Kernel Linux 3.13.0-24-generic
* Processore intel Core i7-3612QM CPU @ 2.10 GHz x 4
* Memoria 7.8 GB
* Disco fisso 712.7 GB
* Scheda grafica Advanced Micro Devices, inc. [AMD/ATI] Thames [Radeon HD 7550M/7570M/7650M] (prog-if 00 [VGA controller])

Software installati:

* IDE Eclipse Mars
* Netbeans
* Apache spark 1.6