Hadoop + Clojure

Hadoop World NYC
Friday, October 2, 2009

Stuart Sierra, AltLaw.org

JVM Languages

	Functional		**Object Oriented**
Native to the JVM	Clojure	Scala	Groovy
Ported to the JVM	Armed Bear CL Kawa		JRuby Jython Rhino

Java is dead, long live the JVM

Clojure

- a new Lisp,
  neither Common Lisp nor Scheme
- Dynamic, Functional
- Immutability and concurrency
- Hosted on the JVM
- Open Source (Eclipse Public License)

Clojure Primitive Types

String	`"Hello, World!\n"`
Integer	`42`
Double	`2.0e64`
BigInteger	`9223372036854775808`
BigDecimal	`1.0M`
Ratio	`3/4`
Boolean	`true, false`
Symbol	`foo`
Keyword	`:foo`
null	`nil`

Clojure Collections

List	`(print :hello "NYC")`
Vector	`[:eat "Pie" 3.14159]`
Map	`{:lisp 1  "The Rest" 0}`
Set	`#{2 1 3 5 "Eureka"}`

Homoiconicity

```
public void greet(String name) {
  System.out.println("Hi, " + name);
}

greet("New York");
Hi, New York
```

```
(defn greet [name]
  (println "Hello," name))

(greet "New York")
Hello, New York
```

```java
public double average(double[] nums) {
  double total = 0;
  for (int i = 0; i < nums.length; i++) {
    total += nums[i];
  }
  return total / nums.length;
}
```

```clojure
(defn average [& nums]
  (/ (reduce + nums) (count nums)))

(average 1 2 3 4)
5/2
```

Data Structures as Functions

```
(def m {:f "foo"
        :b "bar"})

(m :f)
"foo"

(:b m)
"bar"
```

```
(def s #{1 5 3})

(s 3)
true

(s 7)
false
```

```
(import '(com.example.package
                   MyClass YourClass))

(. object method arguments)

(new MyClass arguments)


(.method object arguments)

(MyClass. arguments)

(MyClass/staticMethod)
```

Syntactic
Sugar

```
...open a stream...
try {
   ...do stuff with the stream...
} finally {
   stream.close();
}
```

```
(defmacro with-open [args & body]
   `(let ~args
     (try ~@body
      (finally (.close ~(first args))))))


(with-open [stream (...open a stream...)]
   ...do stuff with stream...)
```

	synchronous	asynchronous
coordinated	`ref`	✗
independent	`atom`	`agent`
unshared	`var`	✗

```
(map function values)
```

↳ *list of values*

```
(reduce function values)
```

↳ *single value*

```
mapper(key, value)
```

↳ *list of key-value pairs*

```
reducer(key, values)
```

↳ *list of key-value pairs*

```
public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}



public static class Reduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

```
(mapper key value)
```

↳ *list of key-value pairs*

```
(reducer key values)
```

↳ *list of key-value pairs*

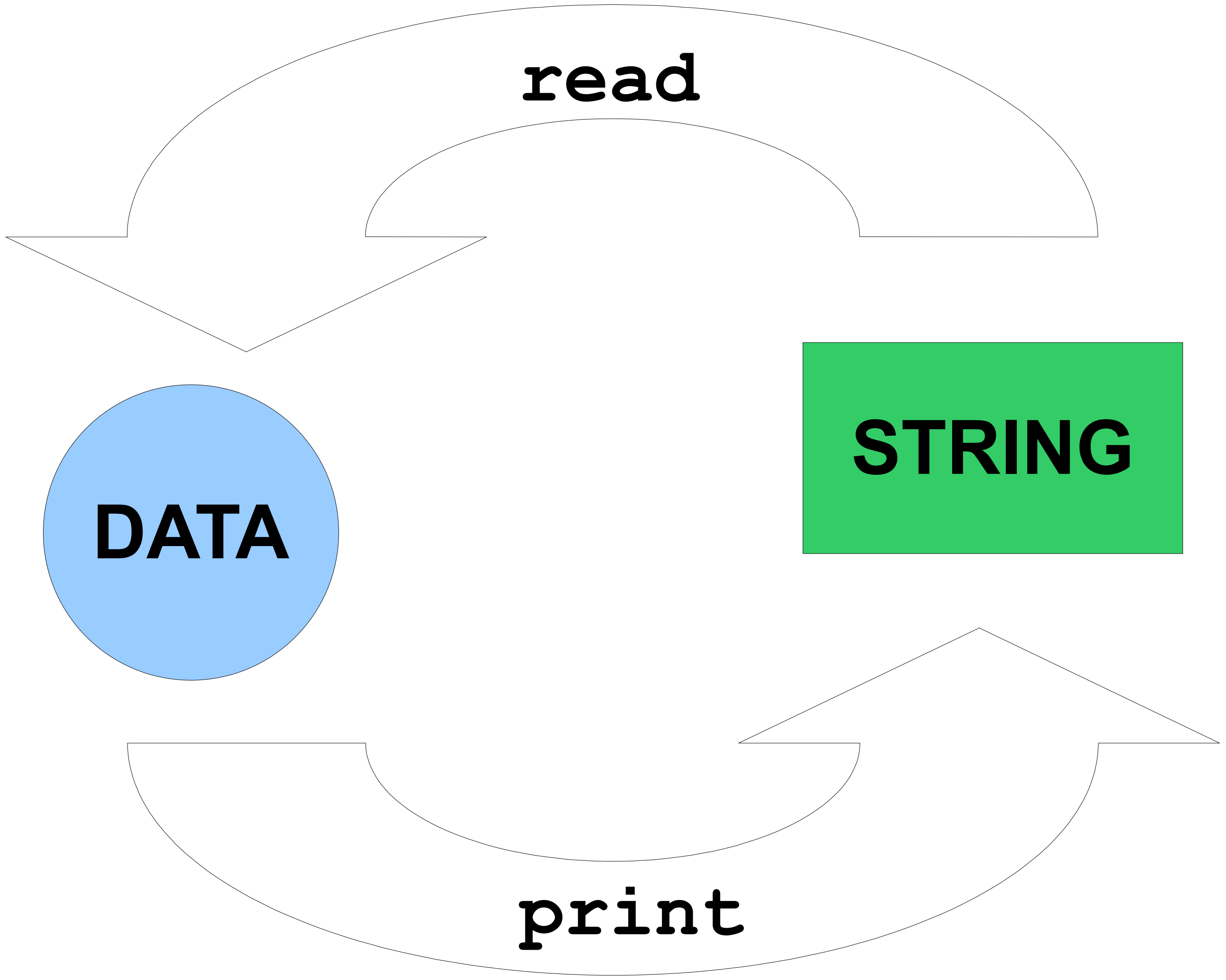
Clojure-Hadoop 1

```clojure
(defn mapper-map [this key val out reporter]
  (doseq [word (enumeration-seq
                (StringTokenizer. (str val)))]
    (.collect out (Text. word)
                  (IntWritable. 1))))

(defn reducer-reduce [this key vals out reprter]
  (let [sum (reduce +
              (map (fn [w] (.get w))
                   (iterator-seq values)))]
    (.collect output key (IntWritable. sum))))

(gen-job-classes)
```

Clojure-Hadoop 2

```
(defn my-map [key value]
   (map (fn [token] [token 1])
        (enumeration-seq (StringTokenizer. value))))

(def mapper-map
  (wrap-map my-map int-string-map-reader))

(defn my-reduce [key values]
   [[key (reduce + values)]])

(def reducer-reduce
  (wrap-reduce my-reduce))

(gen-job-classes)
```

Clojure print/read



Clojure-Hadoop 3

```
(defn my-map [key val]
 (map (fn [token] [token 1])
      (enumeration-seq (StringTokenizer. val))))

(defn my-reduce [key values]
  [[key (reduce + values)]])

(defjob job
  :map my-map
  :map-reader int-string-map-reader
  :reduce my-reduce
  :inputformat :text)
```

```
public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

```
public static class Reduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Clojure-Hadoop 3

```
(defn my-map [key val]
 (map (fn [token] [token 1])
      (enumeration-seq (StringTokenizer. val))))

(defn my-reduce [key values]
  [[key (reduce + values)]])

(defjob job
  :map my-map
  :map-reader int-string-map-reader
  :reduce my-reduce
  :inputformat :text)
```

More

- http://clojure.org/
- Google Groups: Clojure
- #clojure on irc.freenode.net
- #clojure on Twitter
- http://richhickey.github.com/clojure-contrib
- http://stuartsierra.com/
- http://github.com/stuartsierra
- http://www.altlaw.org/