Package 'rkafka'

February 17, 2015

Type Package

Title R Package for KAFKA

Version 1.0

Date 2015-02-11

Author Shruti Gupta

Maintainer Shruti Gupta<shruti.gupta@mu-sigma.com>

Description This package enables the following:1.Creating KAFKA producer 2.Writing messages to a topic 3.Closing KAFKA producer 4.Creating KAFKA consumer 5.Reading messages from a topic 5.Closing KAFKA consumer

Depends rJava,RUnit

SystemRequirements Oracle Java 7,Apache KAFKA 2.8.0-0.8.1.1

License GPL-3

R topics documented:

`R package for KAFKA`

RKAFKA

Description

It provides functionalities of creating a KAFKA producer and consumer, and sending and receiving messages.

Details

Package:	rkafka
Type:	Package
Version:	1.0
Date:	2015-02-11
License:	GPL-3

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Shruti Gupta

Maintainer: Who to complain to shruti.gupta@mu-sigma.com

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
producer1=rkafka.startProducer("127.0.0.1:9092")
rkafka.send(producer1,"ind1","127.0.0.1:9092","this2")
rkafka.send(producer1,"ind1","127.0.0.1:9092","is21")
rkafka.closeProducer(producer1)
consumer1=rkafka.startConsumer("127.0.0.1:2181")
msgs=rkafka.read(consumer1,"ind1")
print(msgs)
rkafka.closeConsumer(consumer1)
```

---

rkafka.closeConsumer

Closing KAKFA consumer

Description

This functions shuts down the KAFKA consumer

Usage

```
rkafka.closeConsumer(HighConsumerObj)
```

Arguments

`HighConsumerObj`
: # @param HighConsumerObj:Consumer through which messages are to be read(Java Object)

Value

Function doesn't return anything

Author(s)

Shruti Gupta

Examples

```
consumer1=rkafka.startConsumer("127.0.0.1:2181")
rkafka.closeConsumer(consumer1)
```

`rkafka.closeProducer`

KAFKA producer shutdown

Description

This function closes the KAFKA producer

Usage

```
rkafka.closeProducer(producerObj)
```

Arguments

`producerObj`
: # * @param producerObj:producerObj(Java object) # * !!Mandatory: Producer which is to be terminated

Value

Doesn't return anything

Author(s)

Shruti Gupta

Examples

```
producer1=rkafka.startProducer("127.0.0.1:9092")
rkafka.closeProducer(producer1)
```

rkafka.read *KAFKA Consumer Reading*

Description

This function reads messages received by a KAFKA consumer

Usage

```
rkafka.read(HighConsumerObj, topicName)
```

Arguments

HighConsumerObj	# @param HighConsumerObj:Consumer through which messages are to be read(Java Object)
topicName	# @param topicName #* :The topic from which message is to be read

Value

Array Of Strings

Note

Warning: Ensure to close the consumer after reading messages. Won't work correctly next time otherwise

Author(s)

Shruti Gupta

Examples

```
consumer1=rkafka.startConsumer("127.0.0.1:2181")
print(rkafka.read(consumer1,"test"))
```

rkafka.send *KAFKA producer sending message*

Description

This function sends message to a particular name through a producer

Usage

```
rkafka.send(producer, topicName, ip, message)
```

Arguments

producer	# * @param producer:producer(Java object) # * !!Mandatory: Producer through which messages are to be sent
topicName	* @param topicName:String # * !!Mandatory: Topic to which messages are to be sent. If topicName doesn't exist, new topic is created # *
ip	# * @param ip:String # * !!Mandatory: ip on which producer is running
message	# * @param message:String # * !!Mandatory: message to be sent

Value

Doesn't return a value

Author(s)

Shruti Gupta

Examples

```
producer1=rkafka.startProducer("127.0.0.1:9092")
rkafka.send(producer1,"test","127.0.0.1:9092","Testing")
```

---

rkafka.startConsumer

Creating high level KAFKA consumer

Description

This function creates a high level KAFKA consumer

Usage

```
rkafka.startConsumer(zookeeperConnect,groupId="test-consumer-group",zookeeperCon
```

Arguments

zookeeperConnect	#@param zookeeperConnect #* !!Mandatory:Zookeeper connection string comma separated #* host:port pairs, each corresponding to a zk server. e.g. #* "127.0.0.1:3000,127.0.0.1:300 #* default:"127.0.0.1:2181"
groupId	#* @param groupId #* !!Mandatory:consumer group id default:test-consumer-group
zookeeperConnectionTimeoutMs	#* @param zookeeperConnectionTimeoutMs #* !!Mandatory:timeout in ms for connecting to zookeeper #* default:100000

`consumerTimeoutMs`
: #* @param consumerTimeoutMs #* !!Mandatory:Throw a timeout exception to the consumer if no #* message is available for consumption after the specified #* interval default:1000

`autoCommitEnable`
: #* –Optional:default:true If true, periodically commit to #* ZooKeeper the offset of messages already fetched by the #* consumer. This committed offset will be used when the process #* fails as the position from which the new consumer will begin.

`autoCommitInterval`
: #* @param autoCommitIntervalMs #* –Optional:default:60*1000 The frequency in ms that the #* consumer offsets are committed to zookeeper.

`autoOffsetReset`
: #* –Optional:default:largest * smallest : automatically reset #* the offset to the smallest offset largest : automatically #* reset the offset to the largest offset anything else: throw #* exception to the consumer

Value

Returns a consumer

Author(s)

Shruti Gupta

Examples

```
consumer1=rkafka.startConsumer("127.0.0.1:2181")
consumer2=rkafka.startConsumer("127.0.0.1:2181","test-consumer-group","50000","1000")
```

---

start.Producer *Creating producer*

Description

This function is used to create a KAFKA producer

Usage

```
rkafka.startProducer(metadataBrokerList,producerType="sync",compressionCodec="no
   serializerClass="kafka.serializer.StringEncoder",partitionerClass="NULL",comp
queueBufferingMaxTime="NULL",
queueBufferingMaxMessages="NULL",
queueEnqueueTimeoutTime="NULL",
batchNumMessages="NULL")
```

Arguments

`metadataBrokerList`
: # * @param metadataBrokerList:String # * !!Mandatory list of brokers used for bootstrapping knowledge # * about the rest of the cluster format: host1:port1,host2:port2 # * ... default:localhost:9092

`producerType` # * @param producerType:String # * !!Mandatory specifies whether the messages are sent # * asynchronously (async) or synchronously (sync) default:sync

`compressionCodec`
: # * @param compressionCodec:String # * !!Mandatory specify the compression codec for all data # * generated: none , gzip, snappy. default:none

`serializerClass`
: # * @param serializerClass:String # * !!Mandatory message encoder # * default:kafka.serializer.StringEncoder

`partitionerClass`
: # * @param partitionerClass:String # * –Optional name of the partitioner class for partitioning # * events; default partition spreads data randomly # default:NULL

`compressedTopics`
: # * @param compressedTopics:String # * –Optional allow topic level compression # * # default:NULL

`queueBufferingMaxTime`
: # * @param queueBufferingMaxTime:String # * –Optional(for Async Producer only) maximum time, in # * milliseconds, for buffering data on the producer queue # * # default:NULL

`queueBufferingMaxMessages`
: # * @param queueBufferingMaxMessages:String # * –Optional(for Async Producer only) the maximum size of the # * blocking queue for buffering on the producer # *# default:NULL

`queueEnqueueTimeoutTime`
: # * –Optional(for Async Producer only) 0: events will be enqueued # * immediately or dropped if the queue is full -ve: enqueue will # * block indefinitely if the queue is full +ve: enqueue will # * block up to this many milliseconds if the queue is full # *# default:NULL

`batchNumMessages`
: # * @param batchNumMessages:String # * –Optional(for Async Producer only) the number of messages # * batched at the producer # * # default:NULL

Value

* @return returns a Properties Object containing properties for the # * Producer, to be passed to MuProducer class

Author(s)

Shruti Gupta

Examples

```
producer1=rkafka.startProducer("127.0.0.1:9092")
producer2=rkafka.startProducer("127.0.0.1:9092","sync","none","kafka.serializer.StringEnc
```