2018/6/4

目标：

1. 数据读入RDD时间
2. RDD落盘时间，序列化、反序列化时间
3. 序列化、反序列化时间

Java Instrumentation(javaagent)

Java Instrumentation能对class文件做拦截，对字节码做修改，比如在指定方法前插入一段代码。
因此，要获取数据读入RDD的时间，可以在相应方法前插入获取时间戳的代码，并在下一个要执行的方法也获取一次时间戳，print出来，查看两个时间戳并相减就能粗略获取读取所需时间。
第一个时间戳可以放在读取文件方法前加入，如hadoopFile方法、binaryFiles、textFile方法（调用了hadoopFile），第二个时间戳位置还没有调研，理想情况下希望能加入到BlockManager Master类中。

对于修改操作，现在有一个简单的示例，打包一个agent jar包，在用idea调试Spark程序时指定该agent包。更普遍的情况，可以在提交Spark应用时，通过增加spark.driver.extraJavaOptions参数指定agent包。

首先定义一个类，实现ClassFileTransformer的transform接口

```
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtMethod;

import java.lang.instrument.ClassFileTransformer;
import java.security.ProtectionDomain;

public class SparkIn implements ClassFileTransformer {
    final String injectedClassName = "org.apache.spark.SparkContext";
    final String injectedMethodName = "textFile";

    public byte[] transform(ClassLoader loader, String className, Class<?>
            classBeingRedefined, ProtectionDomain protectionDomain,
                            byte[] classfileBuffer) {
        className = className.replace("/", ".");
        if (className.equals(injectedClassName)) {
            try {
                CtClass ctclass = ClassPool.getDefault()
                        .get(className);
                CtMethod ctmethod = ctclass
                        .getDeclaredMethod(injectedMethodName);
                ctmethod.insertBefore("System.out.println" +
                        "(\"Start textFile Method at\")" +
```

```
                        ";System.out.println" +
                        "(System.currentTimeMillis());");
                return ctclass.toBytecode();
            } catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
        }
        return null;
    }

}
```

接下来定义一个类实现premain方法，并在打jar包时，将该类指定为Premain-Class

```
import java.lang.instrument.Instrumentation;

public class Agent {
    public static void premain(String agentArgs, Instrumentation inst) {
        System.out.println("=========premain方法执行========");
        inst.addTransformer(new SparkIn());
    }
}
```

MANIFEST.MF文件如下

```
Manifest-Version: 1.0
Premain-Class: com.shenyu.SparkIn.Agent
Can-Redefine-Classes: true
```

jar文件生成后，在Idea中设置Spark应用的VM options，添加agent jar包路径，如
-javaagent:/home/shenyu/IdeaProjects/PageRank/out/SparkIns1.jar

运行后，在输出中能看到agent插入的代码的输出

```
18/06/04 21:08:15 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(dr
18/06/04 21:08:15 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver
Start textFile Method at
1528117696090
18/06/04 21:08:16 INFO MemoryStore: Block broadcast_0 stored as values in memory (es
18/06/04 21:08:16 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memo
```

使用该方法可以在一定程度上获取目标1的信息。相应的，只要找到相应的方法和其执行顺序，应该就可以用类似方法来获取目标2，落盘以及序列化所需时间。

jamp

jamp工具可以获取某进程内所有对象的情况，GC使用的算法等信息
jamp -heap <进程id>
以下为获取master进程的内存情况

```
/usr/jdk/jdk1.8.0_131/bin: sudo jmap -heap 13649
Attaching to process ID 13649, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.131-b11

using thread-local object allocation.
Parallel GC with 8 thread(s)

Heap Configuration:
  MinHeapFreeRatio         = 0
  MaxHeapFreeRatio         = 100
  MaxHeapSize              = 1073741824 (1024.0MB)
  NewSize                  = 87031808 (83.0MB)
  MaxNewSize               = 357564416 (341.0MB)
  OldSize                  = 175112192 (167.0MB)
  NewRatio                 = 2
  SurvivorRatio            = 8
  MetaspaceSize            = 21807104 (20.796875MB)
  CompressedClassSpaceSize = 1073741824 (1024.0MB)
  MaxMetaspaceSize         = 17592186044415 MB
  G1HeapRegionSize         = 0 (0.0MB)

Heap Usage:
PS Young Generation
Eden Space:
  capacity = 196608000 (187.5MB)
  used     = 42190672 (40.23616027832031MB)
  free     = 154417328 (147.2638397216797MB)
  21.459285481770834% used
From Space:
  capacity = 7864320 (7.5MB)
  used     = 7364728 (7.023551940917969MB)
  free     = 499592 (0.47644805908203125MB)
  93.64735921223958% used
To Space:
  capacity = 10485760 (10.0MB)
  used     = 0 (0.0MB)
  free     = 10485760 (10.0MB)
  0.0% used
PS Old Generation
  capacity = 103809024 (99.0MB)
  used     = 41383512 (39.466392517089844MB)
```

```
   free     = 62425512 (59.533607482910156MB)
   39.8650429465554% used

 8011 interned Strings occupying 655288 bytes.
```

加上-dump:format=< format >,file=< filename >可以将heap内容输出到指定文件中，接下来可以用第三方工具进行分析

使用该方法能获取某进程在某时刻的堆内存信息，但是一次执行命令只能获取某一时刻的状态，有比较大的开销。使用该方法不能获取进程堆内存连续的状态

如果要获取GC时间占运行时间的比例，最简单也是最直观的办法是查看history server，history server中记录了每一个task的GC时长，以及GC时长的总和，可以轻松得到比例。如果要获取更详细的GC信息，可以对Spark进行相关配置，加上jvm参数。

比如XX:+PrintGCDetails -XX:+PrintGCTimeStamps，将输出所有的GC详情，接下来可以使用GCviewer等工具进行分析。