Spark MetricsSystem

MetricsSystem使用codahale提供的第三方测量仓库Metrics，其相关概念如下：

1. Instance： 指定谁在使用测量系统， 区分为Master、Worker、Application、Driver和Executor
2. Source: 指定了从哪里收集测量数据
3. Sink： 指定了往哪里输出测量数据，目前提供的Sink有ConsoleSink,CsvSink,JmxSink，MetricsServlet和GraphiteSink等。Spark目前使用MetricsServlet作为默认的Sink

MetricsSystem的启动过程包括如下步骤：

1. 注册Sources
2. 注册Sinks
3. 给Sink增加Jetty的ServletContextHandler

启动代码如下：

*\_env.metricsSystem.start()*

*// Attach the driver metrics servlet handler to the web ui after the metrics system is started.*

*\_env.metricsSystem.getServletHandlers.foreach(handler => ui.foreach(\_.attachHandler(handler)))*

MetricsSystem启动完毕后，会遍历与Sinks有关的ServletContextHandler，并调用attachHandler将其绑定到SparkUI上。MetricsSystem的启动如下：

*def start() {*

*require(!running, "Attempting to start a MetricsSystem that is already running")*

*running = true*

*StaticSources.allSources.foreach(registerSource)*

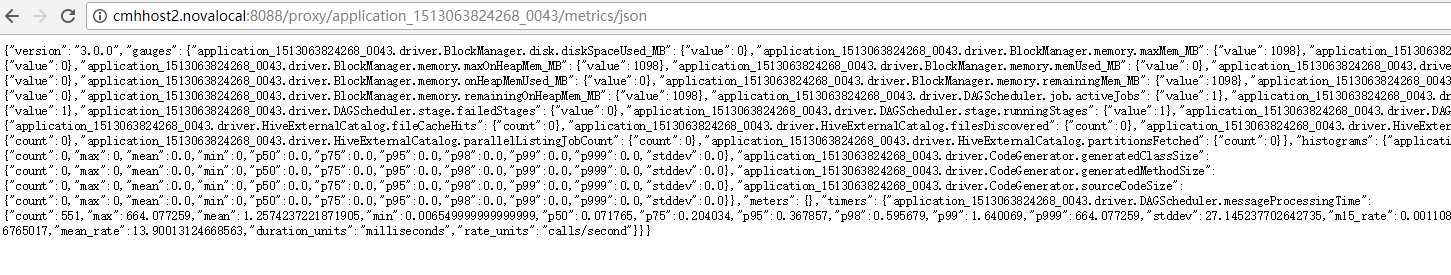
*registerSources()*

*registerSinks()*

*sinks.foreach(\_.start)*

*}*

通过页面获取如下所示：



MetricsSystem的类图如下所示：



1. 注册Source，registerSource方法用于注册Sources，告诉测量系统从哪里收集测量数据，代码如下：

*private def registerSources() {*

*val instConfig = metricsConfig.getInstance(instance)*

*val sourceConfigs = metricsConfig.subProperties(instConfig, MetricsSystem.SOURCE\_REGEX)*

*// Register all the sources related to instance*

*sourceConfigs.foreach { kv =>*

*val classPath = kv.\_2.getProperty("class")*

*try {*

*val source = Utils.classForName(classPath).newInstance()*

*registerSource(source.asInstanceOf[Source])*

*} catch {*

*case e: Exception => logError("Source class " + classPath + " cannot be instantiated", e)*

*}*

*}*

*}*

1. 从metricsConfig获取Driver的properties，默认为创建MetricsSystem的过程中解析：

*prop.setProperty("\*.sink.servlet.class", "org.apache.spark.metrics.sink.MetricsServlet")*

1. 用正则匹配Driver的properties中以source.开头的属性，然后将属性中的Source反射得到的实例加入ArrayBuffer[Source]
2. 将每个source的metricRegistry注册到ConcurrentMap<String,Metric> metrics
3. 注册Sinks

registerSinks方法用于注册Sinks，即告诉测量系统MetricsSystem往哪里输出测量数据，实现代码如下：

*private def registerSinks() {*

*val instConfig = metricsConfig.getInstance(instance)*

*val sinkConfigs = metricsConfig.subProperties(instConfig, MetricsSystem.SINK\_REGEX)*

*sinkConfigs.foreach { kv =>*

*val classPath = kv.\_2.getProperty("class")*

*if (null != classPath) {*

*try {*

*val sink = Utils.classForName(classPath)*

*.getConstructor(classOf[Properties], classOf[MetricRegistry], classOf[SecurityManager])*

*.newInstance(kv.\_2, registry, securityMgr)*

*if (kv.\_1 == "servlet") {*

*metricsServlet = Some(sink.asInstanceOf[MetricsServlet])*

*} else {*

*sinks += sink.asInstanceOf[Sink]*

*}*

*} catch {*

*case e: Exception =>*

*logError("Sink class " + classPath + " cannot be instantiated")*

*throw e*

*}}}}}*

1. 从Driver的properties中用正则匹配sink.开头的属性，如*{"\*.sink.servlet.class", "org.apache.spark.metrics.sink.MetricsServlet"}*
2. 将子属性class对应的类MetricsServlet反射得到MetricsServlet实例，如果属性为servlet，将其设置为metricsSystem；如果是Sink，则加入到ArrayBuffer[Sink]中
3. 给Sinks增加Jetty的ServletContextHandler

为了能够在SparkUI访问到测量数据，所以需要给Sinks增加Jetty的ServletContextHandler，这里用到的MetricsSystem的getServletHandlers方法实现如下：

*def getServletHandlers: Array[ServletContextHandler] = {*

*require(running, "Can only call getServletHandlers on a running MetricsSystem")*

*metricsServlet.map(\_.getHandlers(conf)).getOrElse(Array())*

*}*

调用了metricsServlet的getHandlers，其实现如下：

*def getHandlers(conf: SparkConf): Array[ServletContextHandler] = {*

*Array[ServletContextHandler](*

*createServletHandler(servletPath,*

*new ServletParams(request => getMetricsSnapshot(request), "text/json"), securityMgr, conf)*

*)}*

最终生成处理的/metrics/json请求的ServletContextHandler，而请求的真正处理由getMetricsSnapshot方法，利用fastjson解析，生成ServletContextHandler通过SparkUI的attachHandler方法，被绑定到SparkUI。