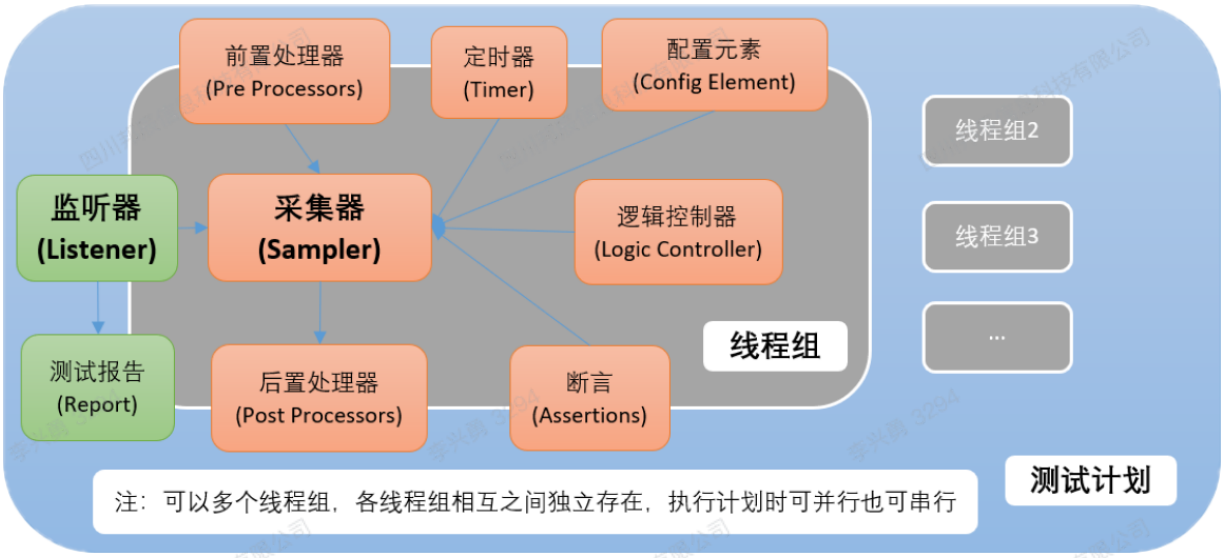
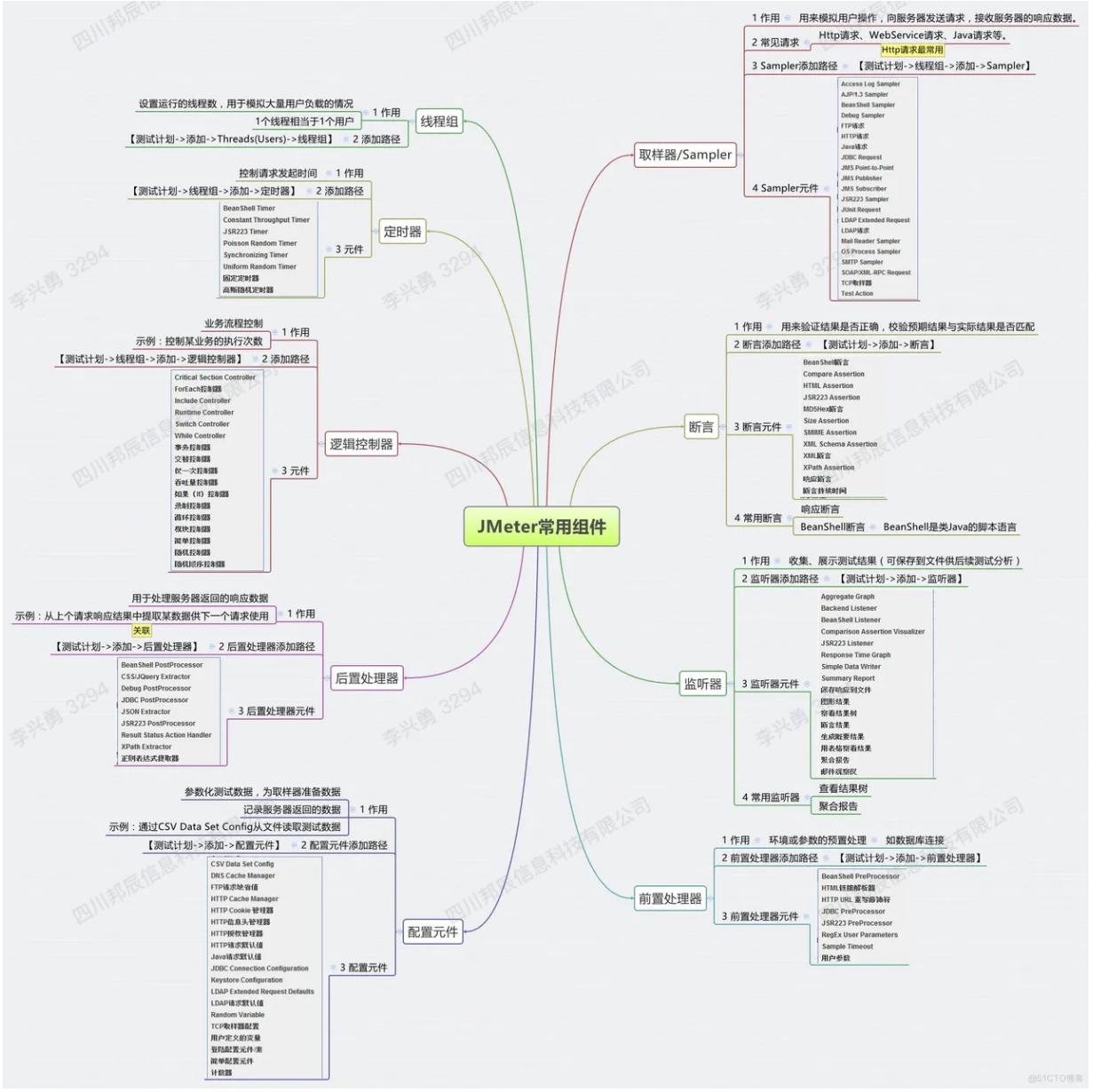
jmeter源码解读

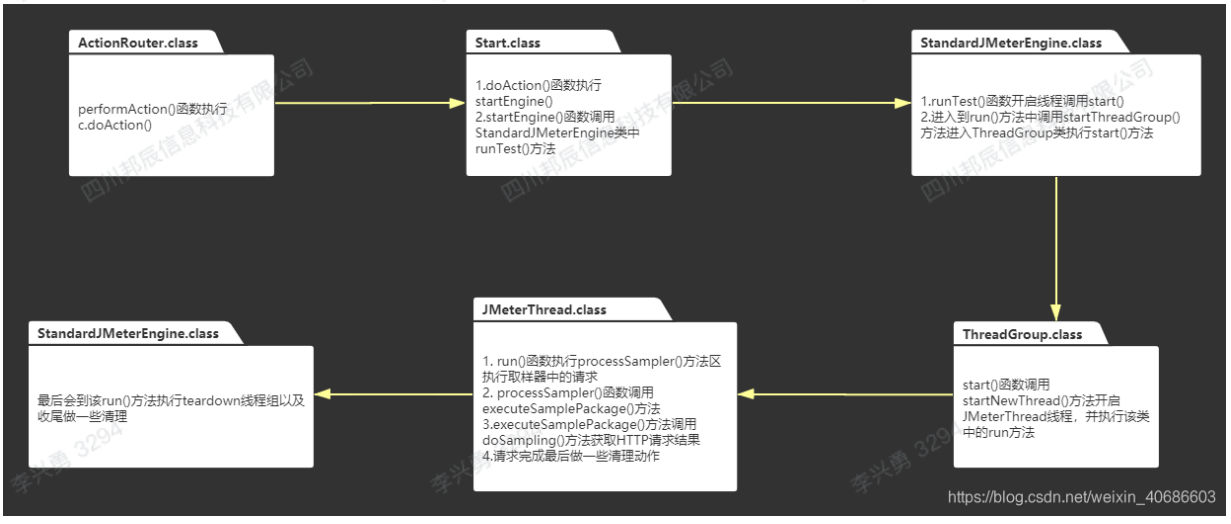
1. JMeter组成结构



2. JMeter常用组件

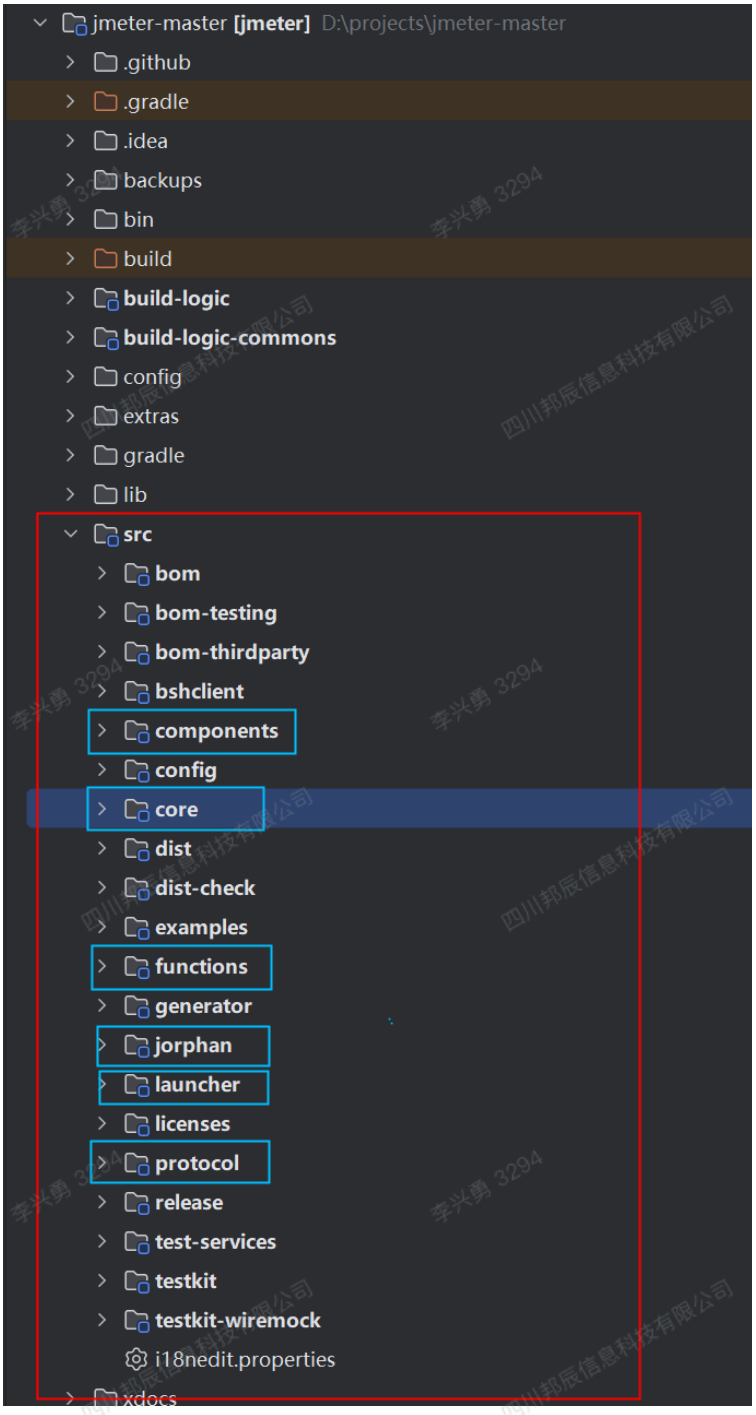


3. JMeter 引擎启动链路图



4. 源码解读

4.1 项目结构



源代码在 src 目录下以各个大模块存在，大模块中存在小模块，我们主要关注蓝色方框的模块

4.1.1 core

jmeter 核心包，引擎就在这里边实现（包含 engine、thread、threadgroup），另外还提供与整个执行相关的 sampler、controller、assertion、timer、samplelistener，processor 一些基类。

4.1.2 components

components 包含了assertion、controller、processor、timer、samplelistener 等的实现类

4.1.3 protocol

protocol 包含了按不同协议实现的 sampler。

4.1.4 jorphan

HashTree 实现库

4.1.5 launcher

包含了NewDriver 和 DynamicClassLoader，主要包含了 lib 包和 ext 包的类加载的初始化配置和加载逻辑

4.1.6 functions

脚本函数库

4.2 核心类解读

4.2.1 sampler

jmeter 主要的采样单元，如发送 http 请求、ftp、jdbc 请求等。在 jmeter 整个处理逻辑中最主要的模块。

```
 */
public interface Sampler extends Serializable, TestElement {  39 implementations
    /**
     * Obtains statistics about the given Entry, and packages the information
     * into a SampleResult.
     *
     * @param e
     *            the Entry (TODO seems to be unused)
     * @return information about the sample
     */
    SampleResult sample(Entry e); 31 implementations
}
```

里边包含了有一个最主要的接口方法 sample，在引擎执行过程中触发 sampler 逻辑的方法。

```
private void executeSamplePackage(Sampler current,  1 usage
        TransactionSampler transactionSampler,
        SamplePackage transactionPack,
        JMeterContext threadContext) {

    threadContext.setCurrentSampler(current);
    // Get the sampler ready to sample
    SamplePackage pack = compiler.configureSampler(current);
    runPreProcessors(pack.getPreProcessors());

    // Hack: save the package for any transaction controllers
    threadVars.putObject(PACKAGE_OBJECT, pack);

    delay(pack.getTimers());
    SampleResult result = null;
    if (running) {
        Sampler sampler = pack.getSampler();
        result = doSampling(threadContext, sampler);
    }
    // If we got any results, then perform processing on the result
    if (result != null) {
        if (!result.isIgnore()) {
            int nbActiveThreadsInThreadGroup = threadGroup.getNumberOfThreads();
            int nbTotalActiveThreads = JMeterContextService.getNumberOfThreads();
            fillThreadInformation(result, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
            SampleResult[] subResults = result.getSubResults();
            if (subResults != null) {
                for (SampleResult subResult : subResults) {
                    fillThreadInformation(subResult, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
                }
            }
```

```java
private SampleResult doSampling(JMeterContext threadContext, Sampler sampler) {  1 usage
    sampler.setThreadContext(threadContext);
    setSamplerThreadName(sampler);
    TestBeanHelper.prepare(sampler);

    // Perform the actual sample
    currentSamplerForInterruption = sampler;
    if (!sampleMonitors.isEmpty()) {
        for (SampleMonitor sampleMonitor : sampleMonitors) {
            if(sampleMonitor instanceof TestElement) {
                TestBeanHelper.prepare((TestElement) sampleMonitor);
            }
            sampleMonitor.sampleStarting(sampler);
        }
    }
    try {
        return sampler.sample( e: null);
    } finally {
        if (!sampleMonitors.isEmpty()) {
            for (SampleMonitor sampleMonitor : sampleMonitors) {
                sampleMonitor.sampleEnded(sampler);
            }
        }
        currentSamplerForInterruption = null;
```

4.2.2 controller

controller 在整个执行过程中个人理解充当了流程控制的功能，通过 next 按深度优先的方式，遍历 hashtree 上的 sampler 给到 engine 里的 jmeterthread 去一个一个执行。

```java
 */
public interface Controller extends TestElement {  25 implementations
    /**
     * Delivers the next Sampler or null
     *
     * @return org.apache.jmeter.samplers.Sampler or null
     */
    Sampler next();  13 implementations

    /**
     * Indicates whether the Controller is done delivering Samplers for the rest
     * of the test.
     *
     * When the top-level controller returns true to JMeterThread,
     * the thread is complete.
     *
     * @return boolean
     */
    boolean isDone();  6 implementations

    /**
     * Controllers have to notify listeners of when they begin an iteration
     * through their sub-elements.
```

GenericController 实现了controller 并让其他 controller 继承，对 next 做了基础实现，主要实现了深度优先遍历 sampler 的逻辑

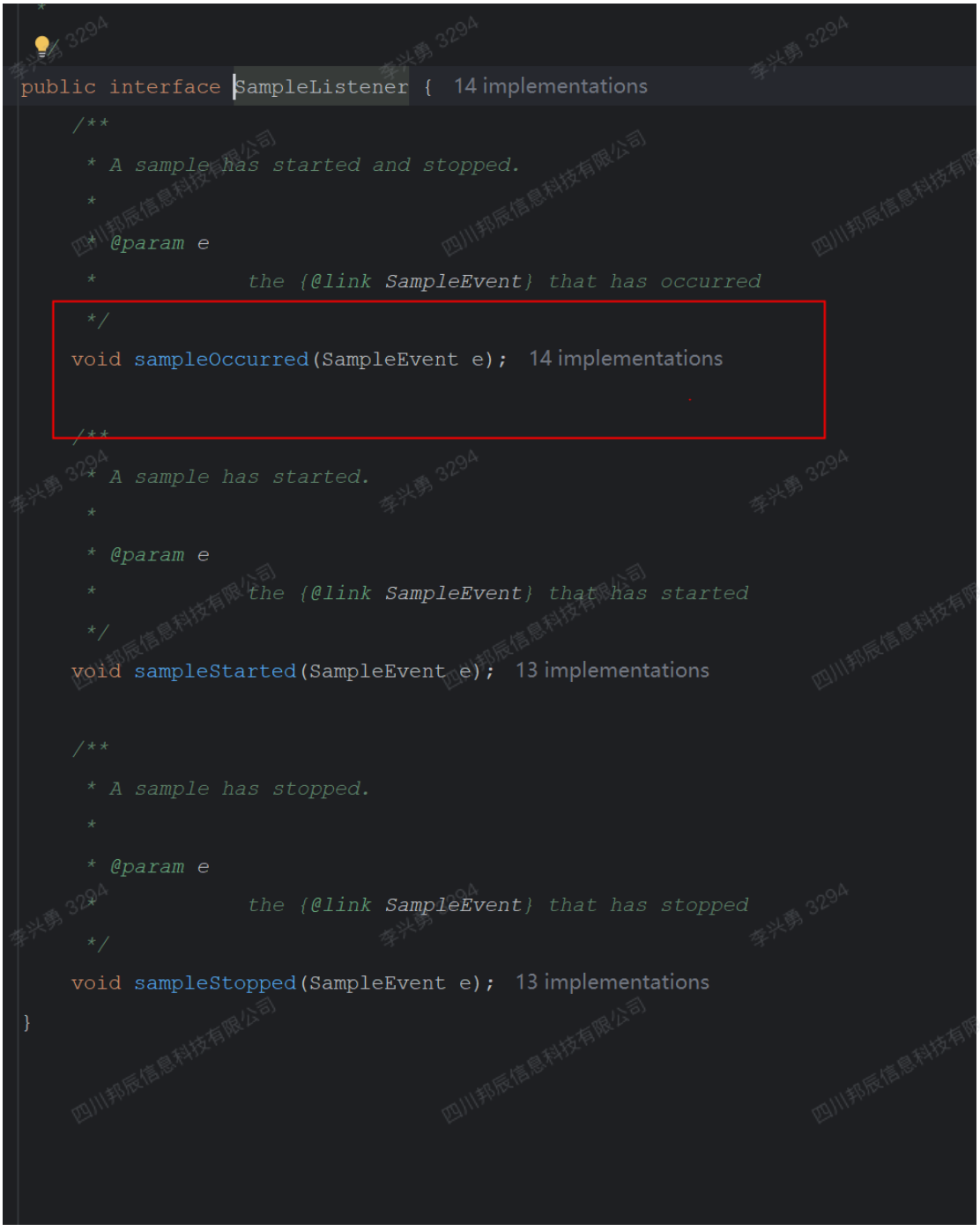
```
     */
    @Override  11 overrides
    public Sampler next() {
        fireIterEvents();
        log.debug("Calling next on: {}", GenericController.class);
        if (isDone()) {
            return null;
        }
        Sampler returnValue = null;
        try {
            TestElement currentElement = getCurrentElement();
            setCurrentElement(currentElement);
            if (currentElement == null) {
                returnValue = nextIsNull();
            } else {
                if (currentElement instanceof Sampler) {
                    returnValue = nextIsASampler((Sampler) currentElement);
                } else { // must be a controller
                    returnValue = nextIsAController((Controller) currentElement);
                }
            }
        } catch (NextIsNullException e) {
            // NOOP
        }
        return returnValue;
    }
```

如果 next 拿到的是 sampler 则直接返回，如果拿到的是 controller，则调用 nextTsAController 进行递归，去拿下一个 sampler

```
     */
    protected Sampler nextIsAController(Controller controller)  3 usages  2 overrides
            throws NextIsNullException { // NOSONAR false positive , throws is required by subclasse
        Sampler sampler = controller.next();
        if (sampler == null) {
            currentReturnedNull(controller);
            sampler = next();
        }
        return sampler;
    }
```

4.2.3 samplelistener

jmeter 引擎主要通过 samplelistener 的实现类对层级下的 sampler 等结果进行监听，获取到执行结果后通过 listener 的具体实现类（如BackendListener）展示或存储结果



```java
public interface SampleListener {  14 implementations
    /**
     * A sample has started and stopped.
     *
     * @param e
     *            the {@link SampleEvent} that has occurred
     */
    void sampleOccurred(SampleEvent e);  14 implementations

    /**
     * A sample has started.
     *
     * @param e
     *            the {@link SampleEvent} that has started
     */
    void sampleStarted(SampleEvent e);  13 implementations

    /**
     * A sample has stopped.
     *
     * @param e
     *            the {@link SampleEvent} that has stopped
     */
    void sampleStopped(SampleEvent e);  13 implementations
}
```

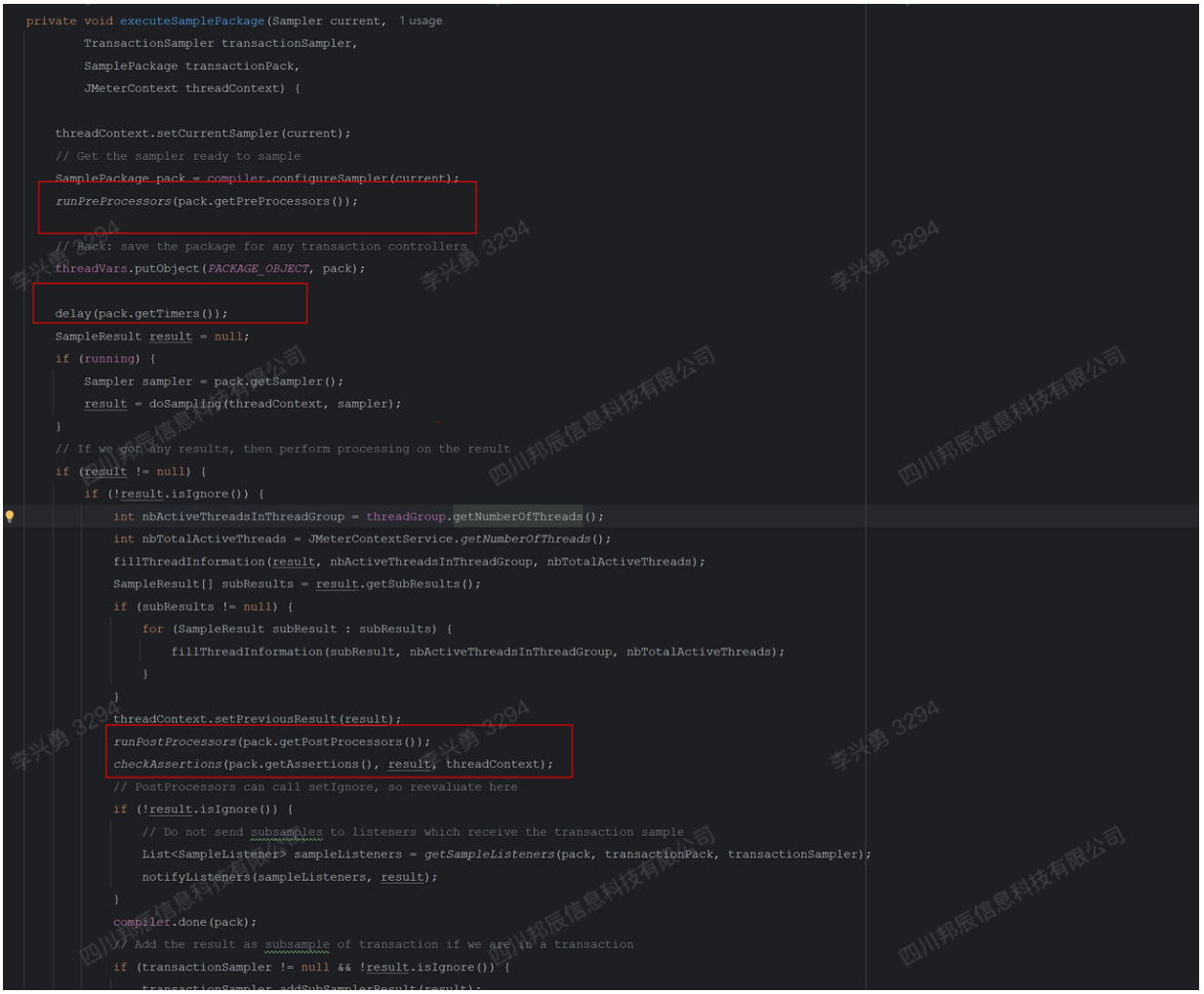
主要接口方法为 sampleOccurred，当 sampler 执行完成后，会在 JmeterThread 里调用 notifyListeners，并通过遍历调用 listener 的 sampleOccurred 方法将 sampler result 给到 listener。

```
        threadContext.setPreviousResult(result);
        runPostProcessors(pack.getPostProcessors());
        checkAssertions(pack.getAssertions(), result, threadContext);
        // PostProcessors can call setIgnore, so reevaluate here
        if (!result.isIgnore()) {
            // Do not send subsamples to listeners which receive the transaction sample
            List<SampleListener> sampleListeners = getSampleListeners(pack, transactionPack, transactionSampler
            notifyListeners(sampleListeners, result);
        }
        compiler.done(pack);
        // Add the result as subsample of transaction if we are in a transaction
        if (transactionSampler != null && !result.isIgnore()) {
            transactionSampler.addSubSamplerResult(result);
```

```
    */
    public void notifyListeners(SampleEvent res, List<SampleListener> listeners) {  3 usages
        for (SampleListener sampleListener : listeners) {
            try {
                TestBeanHelper.prepare((TestElement) sampleListener);
                sampleListener.sampleOccurred(res);
            } catch (RuntimeException e) {
                log.error("Detected problem in Listener.", e);
                log.info("Continuing to process further listeners");
            }
        }
    }
```

4.2.4 timer、assertion、preprocessor、postprocessor

这部分实现都比较简单，可以点进去看一下实现就很清楚了，具体的调用时机，是在 JmeterThread 中 executeSamplePackage 方法到具体流程会发起调用



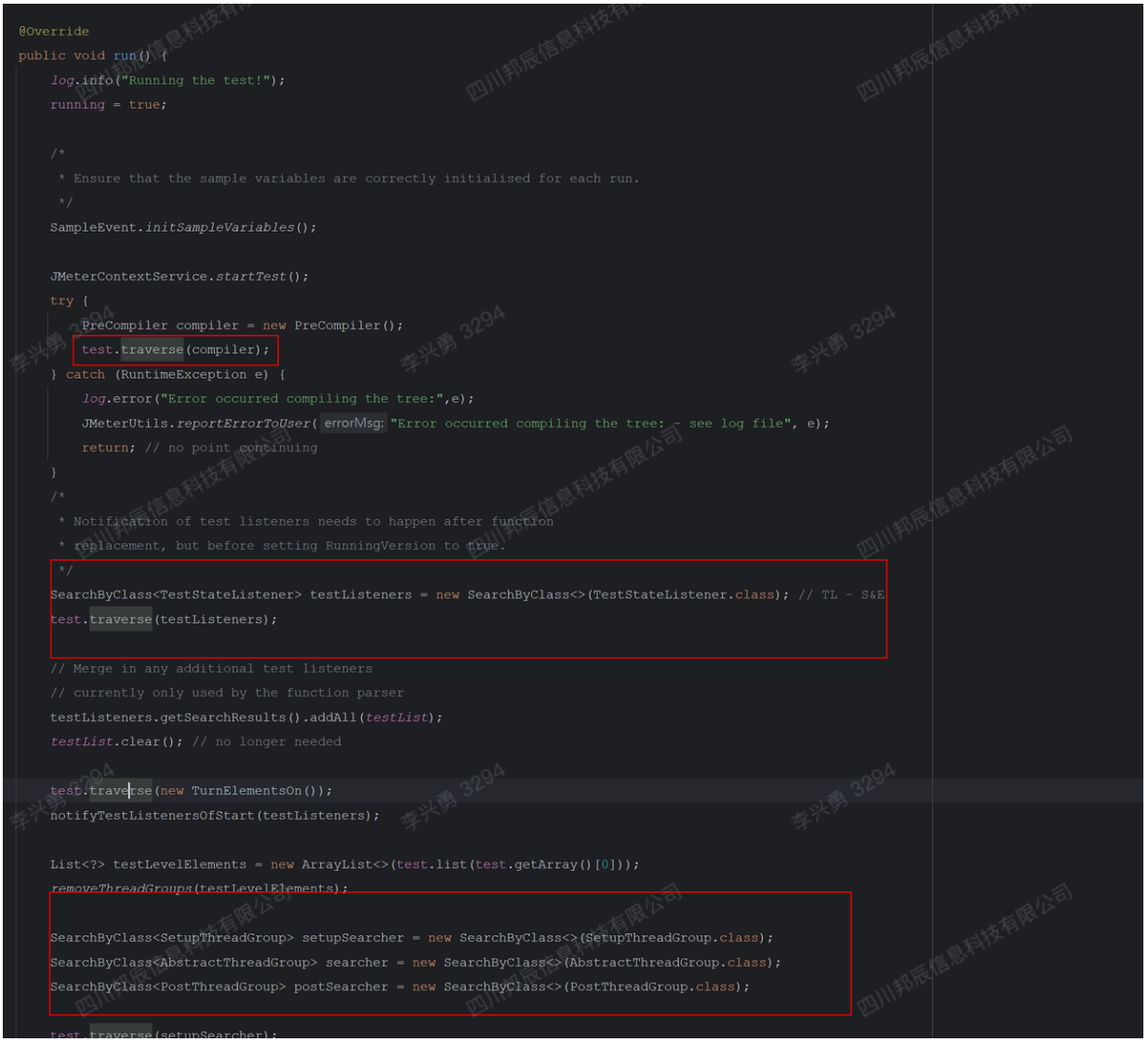
```
private void executeSamplePackage(Sampler current,  1 usage
        TransactionSampler transactionSampler,
        SamplePackage transactionPack,
        JMeterContext threadContext) {

    threadContext.setCurrentSampler(current);
    // Get the sampler ready to sample
    SamplePackage pack = compiler.configureSampler(current);
    runPreProcessors(pack.getPreProcessors());

    // Hack: save the package for any transaction controllers
    threadVars.putObject(PACKAGE_OBJECT, pack);

    delay(pack.getTimers());
    SampleResult result = null;
    if (running) {
        Sampler sampler = pack.getSampler();
        result = doSampling(threadContext, sampler);
    }
    // If we got any results, then perform processing on the result
    if (result != null) {
        if (!result.isIgnore()) {
            int nbActiveThreadsInThreadGroup = threadGroup.getNumberOfThreads();
            int nbTotalActiveThreads = JMeterContextService.getNumberOfThreads();
            fillThreadInformation(result, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
            SampleResult[] subResults = result.getSubResults();
            if (subResults != null) {
                for (SampleResult subResult : subResults) {
                    fillThreadInformation(subResult, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
                }
            }
            threadContext.setPreviousResult(result);
            runPostProcessors(pack.getPostProcessors());
            checkAssertions(pack.getAssertions(), result, threadContext);
            // PostProcessors can call setIgnore, so reevaluate here
            if (!result.isIgnore()) {
                // Do not send subsamples to listeners which receive the transaction sample
                List<SampleListener> sampleListeners = getSampleListeners(pack, transactionPack, transactionSampler);
                notifyListeners(sampleListeners, result);
            }
            compiler.done(pack);
            // Add the result as subsample of transaction if we are in a transaction
            if (transactionSampler != null && !result.isIgnore()) {
                transactionSampler.addSubSamplerResult(result);
```

4.2.5 启动流程

启动流程分为 gui 启动和 console 启动，调用关系可参考 jmeter 引擎启动链路图的内容。

我们主要讲一下JMeterEngine具体执行的过程，这里我们主要说一下StandardJMeterEngine ，远程引擎实现加了远程调用，这里就不去展开了。

我们直接进入到 JmeterEngine 的 run 方法



```java
@Override
public void run() {
    log.info("Running the test!");
    running = true;

    /*
     * Ensure that the sample variables are correctly initialised for each run.
     */
    SampleEvent.initSampleVariables();

    JMeterContextService.startTest();
    try {
        PreCompiler compiler = new PreCompiler();
        test.traverse(compiler);
    } catch (RuntimeException e) {
        log.error("Error occurred compiling the tree:",e);
        JMeterUtils.reportErrorToUser( errorMsg: "Error occurred compiling the tree: - see log file", e);
        return; // no point continuing
    }
    /*
     * Notification of test listeners needs to happen after function
     * replacement, but before setting RunningVersion to true.
     */
    SearchByClass<TestStateListener> testListeners = new SearchByClass<>(TestStateListener.class); // TL - S&E
    test.traverse(testListeners);

    // Merge in any additional test listeners
    // currently only used by the function parser
    testListeners.getSearchResults().addAll(testList);
    testList.clear(); // no longer needed

    test.traverse(new TurnElementsOn());
    notifyTestListenersOfStart(testListeners);

    List<?> testLevelElements = new ArrayList<>(test.list(test.getArray()[0]));
    removeThreadGroups(testLevelElements);

    SearchByClass<SetupThreadGroup> setupSearcher = new SearchByClass<>(SetupThreadGroup.class);
    SearchByClass<AbstractThreadGroup> searcher = new SearchByClass<>(AbstractThreadGroup.class);
    SearchByClass<PostThreadGroup> postSearcher = new SearchByClass<>(PostThreadGroup.class);

    test.traverse(setupSearcher);
```

里边大量的使用了 hashtree 的 traverse 方法，传入一个HashTreeTraverser 的实现类，通过 addNode 遍历整个 hashtree 访问需要的元素，相当于是遍历 hashtree，准备数据用于后续流程使用

```java
public interface HashTreeTraverser {  17 implementations
    /**
     * The tree traverses itself depth-first, calling addNode for each object it
     * encounters as it goes. This is a callback method, and should not be
     * called except by a HashTree during traversal.
     *
     * @param node
     *            the node currently encountered
     * @param subTree
     *            the HashTree under the node encountered
     */
    void addNode(Object node, HashTree subTree);  14 implementations

    /**
     * Indicates traversal has moved up a step, and the visitor should remove
     * the top node from its stack structure. This is a callback method, and
     * should not be called except by a HashTree during traversal.
     */
    void subtractNode();  14 implementations

    /**
     * Process path is called when a leaf is reached. If a visitor wishes to
     * generate Lists of path elements to each leaf, it should keep a Stack data
     * structure of nodes passed to it with addNode, and removing top items for
     * every {@link #subtractNode()} call. This is a callback method, and should
     * not be called except by a HashTree during traversal.
     */
```

整个数据准备工作做完之后调用 startThreadGroup 方法，按 threadgroup 运行整个 hashtree

```java
JMeterContextService.getContext().setSamplingStarted(true);
boolean mainGroups = running; // still running at this point, i.e. setUp was not cancelled
while (running && iter.hasNext()) {// for each thread group
    AbstractThreadGroup group = iter.next();
    //ignore Setup and Post here.  We could have filtered the searcher. but then
    //future Thread Group objects wouldn't execute.
    if (group instanceof SetupThreadGroup ||
            group instanceof PostThreadGroup) {
        continue;
    }
    groupCount++;
    String groupName = group.getName();
    log.info("Starting ThreadGroup: {} : {}", groupCount, groupName);
    startThreadGroup(group, groupCount, searcher, testLevelElements, notifier);
    if (serialized && iter.hasNext()) {
        log.info("Waiting for thread group: {} to finish before starting next group", groupName);
        group.waitThreadsStopped();
    }
} // end of thread groups
if (groupCount == 0){ // No TGs found
    log.info("No enabled thread groups found");
```

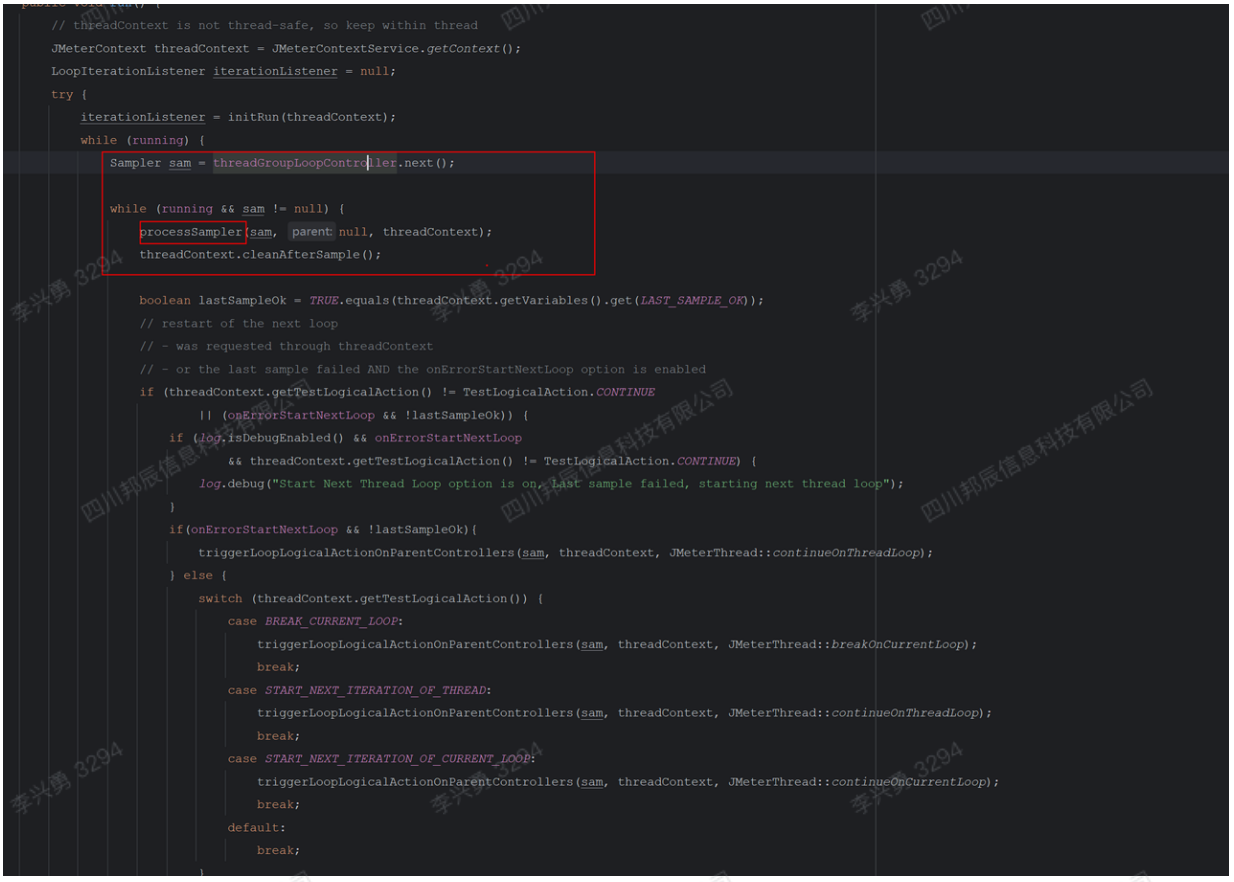
```
private void startThreadGroup(AbstractThreadGroup group, int groupCount, SearchByClass<?> searcher, List<?> testLevelElements, ListenerNotifier notifier)  3 usages
{
    try {
        int numThreads = group.getNumThreads();
        JMeterContextService.addTotalThreads(numThreads);
        boolean onErrorStopTest = group.getOnErrorStopTest();
        boolean onErrorStopTestNow = group.getOnErrorStopTestNow();
        boolean onErrorStopThread = group.getOnErrorStopThread();
        boolean onErrorStartNextLoop = group.getOnErrorStartNextLoop();
        String groupName = group.getName();
        log.info("Starting {} threads for group {}.", numThreads, groupName);
        if (onErrorStopTest) {
            log.info("Test will stop on error");
        } else if (onErrorStopTestNow) {
            log.info("Test will stop abruptly on error");
        } else if (onErrorStopThread) {
            log.info("Thread will stop on error");
        } else if (onErrorStartNextLoop) {
            log.info("Thread will start next loop on error");
        } else {
            log.info("Thread will continue on error");
        }
        ListedHashTree threadGroupTree = (ListedHashTree) searcher.getSubTree(group);
        threadGroupTree.add(group, testLevelElements);

        groups.add(group);
        group.start(groupCount, notifier, threadGroupTree,  engine: this);
    } catch (JMeterStopTestException ex) { // NOSONAR Reported by log
        JMeterUtils.reportErrorToUser( errorMsg: "Error occurred starting thread group :" + group.getName()+ ", error message:"+ex.getMessage()
            +", \r\nsee log file for more details", ex);
        return; // no point continuing
    }
```

```
public void start(int groupNum, ListenerNotifier notifier, ListedHashTree threadGroupTree, StandardJMeterEngine engine) {
    this.running = true;
    this.groupNumber = groupNum;
    this.notifier = notifier;
    this.threadGroupTree = threadGroupTree;
    int numThreads = getNumThreads();
    int rampUpPeriodInSeconds = getRampUp();
    delayedStartup = isDelayedStartup(); // Fetch once; needs to stay constant
    log.info("Starting thread group... number={} threads={} ramp-up={} delayedStart={}", groupNumber,
            numThreads, rampUpPeriodInSeconds, delayedStartup);
    if (delayedStartup) {
        threadStarter = new Thread(new ThreadStarter(notifier, threadGroupTree, engine),  name: getName()+"-ThreadStarter");
        threadStarter.setDaemon(true);
        threadStarter.start();
        // N.B. we don't wait for the thread to complete, as that would prevent parallel TGs
    } else {
        final JMeterVariables variables = JMeterContextService.getContext().getVariables();
        long lastThreadStartInMillis = 0;
        int delayForNextThreadInMillis = 0;
        final int perThreadDelayInMillis = Math.round((float) rampUpPeriodInSeconds * 1000 / numThreads);
        for (int threadNum = 0; running && threadNum < numThreads; threadNum++) {
            long nowInMillis = System.currentTimeMillis();
            if(threadNum > 0) {
                long timeElapsedToStartLastThread = nowInMillis - lastThreadStartInMillis;
                // Note: `int += long` assignment hides lossy cast to int
                delayForNextThreadInMillis = (int) (delayForNextThreadInMillis +
                        (perThreadDelayInMillis - timeElapsedToStartLastThread));
            }
            if (log.isDebugEnabled()) {
                log.debug("Computed delayForNextThreadInMillis:{} for thread:{}", delayForNextThreadInMillis, Thread.currentThread().getId());
            }
            lastThreadStartInMillis = nowInMillis;
            startNewThread(notifier, threadGroupTree, engine, threadNum, variables, nowInMillis, Math.max(0, delayForNextThreadInMillis));
        }
    }
    log.info("Started thread group number {}", groupNumber);
}
```

```
*/
private JMeterThread startNewThread(ListenerNotifier notifier, ListedHashTree threadGroupTree, StandardJMeterEngine engine,  2 usages
        int threadNum, JMeterVariables variables, long now, int delay) {
    JMeterThread jmThread = makeThread(engine,  monitor: this, notifier, groupNumber, threadNum, cloneTree(threadGroupTree), variables);
    scheduleThread(jmThread, now); // set start and end time
    jmThread.setInitialDelay(delay);
    Thread newThread = new Thread(jmThread, jmThread.getThreadName());
    registerStartedThread(jmThread, newThread);
    newThread.start();
    return jmThread;
}

/*
```

```
     * @return {@link JMeterThread}
     */
@API(status = API.Status.EXPERIMENTAL, since = "5.5")  3 usages
protected JMeterThread makeThread(
        StandardJMeterEngine engine,
        JMeterThreadMonitor monitor, ListenerNotifier notifier,
        int groupNumber, int threadNumber,
        ListedHashTree threadGroupTree,
        JMeterVariables variables) {
    boolean onErrorStopTest = getOnErrorStopTest();
    boolean onErrorStopTestNow = getOnErrorStopTestNow();
    boolean onErrorStopThread = getOnErrorStopThread();
    boolean onErrorStartNextLoop = getOnErrorStartNextLoop();
    String groupName = getName();
    final JMeterThread jmeterThread = new JMeterThread(threadGroupTree, monitor, notifier, isSameUserOnNextIteration());
    jmeterThread.setThreadNum(threadNumber);
    jmeterThread.setThreadGroup(this);
    jmeterThread.putVariables(variables);
    String distributedPrefix =
            JMeterUtils.getPropDefault(JMeterUtils.THREAD_GROUP_DISTRIBUTED_PREFIX_PROPERTY_NAME,  defaultVal: "");
    final String threadName = distributedPrefix + (distributedPrefix.isEmpty() ? "":"-") +groupName + " " + groupNumber + "-" + (threadNumber +
    jmeterThread.setThreadName(threadName);
    jmeterThread.setEngine(engine);
    jmeterThread.setOnErrorStopTest(onErrorStopTest);
    jmeterThread.setOnErrorStopTestNow(onErrorStopTestNow);
    jmeterThread.setOnErrorStopThread(onErrorStopThread);
    jmeterThread.setOnErrorStartNextLoop(onErrorStartNextLoop);
    return jmeterThread;
}
```

下边我们就直接进入到 JmeterThread 类里边看具体的运行流程，这里我们直接看 run 方法



```
    // threadContext is not thread-safe, so keep within thread
    JMeterContext threadContext = JMeterContextService.getContext();
    LoopIterationListener iterationListener = null;
    try {
        iterationListener = initRun(threadContext);
        while (running) {
            Sampler sam = threadGroupLoopController.next();

            while (running && sam != null) {
                processSampler(sam,  parent: null, threadContext);
                threadContext.cleanAfterSample();

                boolean lastSampleOk = TRUE.equals(threadContext.getVariables().get(LAST_SAMPLE_OK));
                // restart of the next loop
                // - was requested through threadContext
                // - or the last sample failed AND the onErrorStartNextLoop option is enabled
                if (threadContext.getTestLogicalAction() != TestLogicalAction.CONTINUE
                        || (onErrorStartNextLoop && !lastSampleOk)) {
                    if (log.isDebugEnabled() && onErrorStartNextLoop
                            && threadContext.getTestLogicalAction() != TestLogicalAction.CONTINUE) {
                        log.debug("Start Next Thread Loop option is on, last sample failed, starting next thread loop");
                    }
                    if(onErrorStartNextLoop && !lastSampleOk){
                        triggerLoopLogicalActionOnParentControllers(sam, threadContext, JMeterThread::continueOnThreadLoop);
                    } else {
                        switch (threadContext.getTestLogicalAction()) {
                            case BREAK_CURRENT_LOOP:
                                triggerLoopLogicalActionOnParentControllers(sam, threadContext, JMeterThread::breakOnCurrentLoop);
                                break;
                            case START_NEXT_ITERATION_OF_THREAD:
                                triggerLoopLogicalActionOnParentControllers(sam, threadContext, JMeterThread::continueOnThreadLoop);
                                break;
                            case START_NEXT_ITERATION_OF_CURRENT_LOOP:
                                triggerLoopLogicalActionOnParentControllers(sam, threadContext, JMeterThread::continueOnCurrentLoop);
                                break;
                            default:
                                break;
                        }
```

这里看到主要是通过 controller 的 next 方法深度优先去遍历 samper 执行，具体执行过程在 sampler 和 controller 里边已经提到，这里就不再过多描述

```java
SampleResult transactionResult = null;
// Check if we are running a transaction
TransactionSampler transactionSampler = null;
// Find the package for the transaction
SamplePackage transactionPack = null;
try {
    if (current instanceof TransactionSampler) {
        transactionSampler = (TransactionSampler) current;
        transactionPack = compiler.configureTransactionSampler(transactionSampler);

        // Check if the transaction is done
        if (transactionSampler.isTransactionDone()) {
            transactionResult = doEndTransactionSampler(transactionSampler,
                    parent,
                    transactionPack,
                    threadContext);
            // Transaction is done, we do not have a sampler to sample
            current = null;
        } else {
            Sampler prev = current;
            // It is the sub sampler of the transaction that will be sampled
            current = transactionSampler.getSubSampler();
            if (current instanceof TransactionSampler) {
                SampleResult res = processSampler(current, prev, threadContext);// recursive call
                threadContext.setCurrentSampler(prev);
                current = null;
                if (res != null) {
                    transactionSampler.addSubSamplerResult(res);
                }
            }
        }
    }

    // Check if we have a sampler to sample
    if (current != null) {
        executeSamplePackage(current, transactionSampler, transactionPack, threadContext);
    }
```

```java
private void executeSamplePackage(Sampler current, 1 usage
        TransactionSampler transactionSampler,
        SamplePackage transactionPack,
        JMeterContext threadContext) {

    threadContext.setCurrentSampler(current);
    // Get the sampler ready to sample
    SamplePackage pack = compiler.configureSampler(current);
    runPreProcessors(pack.getPreProcessors());

    // Hack: save the package for any transaction controllers
    threadVars.putObject(PACKAGE_OBJECT, pack);

    delay(pack.getTimers());
    SampleResult result = null;
    if (running) {
        Sampler sampler = pack.getSampler();
        result = doSampling(threadContext, sampler);
    }
    // If we got any results, then perform processing on the result
    if (result != null) {
        if (!result.isIgnore()) {
            int nbActiveThreadsInThreadGroup = threadGroup.getNumberOfThreads();
            int nbTotalActiveThreads = JMeterContextService.getNumberOfThreads();
            fillThreadInformation(result, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
            SampleResult[] subResults = result.getSubResults();
            if (subResults != null) {
                for (SampleResult subResult : subResults) {
                    fillThreadInformation(subResult, nbActiveThreadsInThreadGroup, nbTotalActiveThreads);
                }
            }
            threadContext.setPreviousResult(result);
            runPostProcessors(pack.getPostProcessors());
            checkAssertions(pack.getAssertions(), result, threadContext);
            // PostProcessors can call setIgnore, so reevaluate here
            if (!result.isIgnore()) {
                // Do not send subsamples to listeners which receive the transaction sample
                List<SampleListener> sampleListeners = getSampleListeners(pack, transactionPack, transactionSampler);
                notifyListeners(sampleListeners, result);
            }
            compiler.done(pack);
            // Add the result as subsample of transaction if we are in a transaction
            if (transactionSampler != null && !result.isIgnore()) {
```