toc: true title: 《从1到100深入学习Flink》—— Standalone Session Cluster 启动流程深度分析之 Job Manager 启动 date: 2019-02-02 tags:

- Flink
- 大数据
- 流式计算

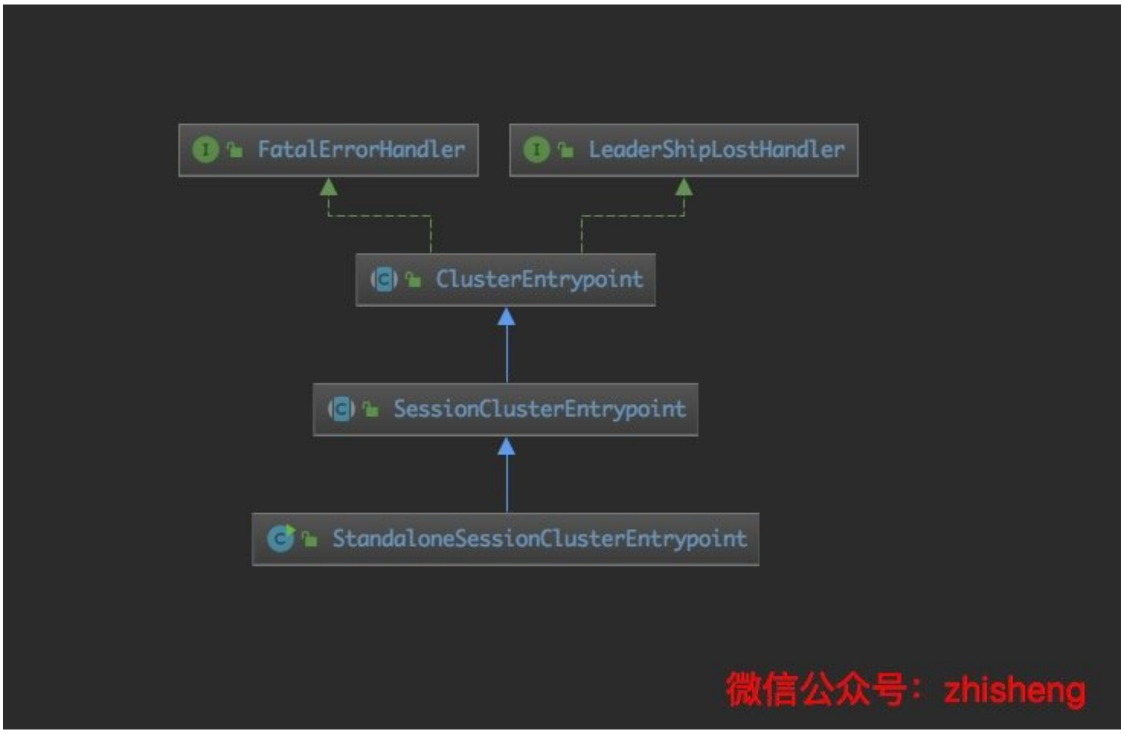
前言

上篇文章分析了 Blink 的脚本执行流程，并探究到单机集群模式下启动 JobManager 和 TaskManager 对应的 Java 启动类（入口点），这篇文章我们先深入分析下 jobmanager 的启动流程，下篇文章我们深入分析下 taskmanager 的启动流程。

我们先来看下上篇文章中揪出的关键入口类：StandaloneSessionClusterEntrypoint

StandaloneSessionClusterEntrypoint

它的类继承结构如下：



可以看见，StandaloneSessionClusterEntrypoint 继承自 SessionClusterEntrypoint，SessionClusterEntrypoint 又继承自 ClusterEntrypoint 类。ClusterEntrypoint 实现 FatalErrorHandler 和 LeaderShipLostHandler 两个 Handler 接口。

回到 StandaloneSessionClusterEntrypoint 类中，我们看到里面的 main 方法如下

```
public static void main(String[] args) {
    // startup checks and logging
    EnvironmentInformation.logEnvironmentInfo(LOG, 
StandaloneSessionClusterEntrypoint.class.getSimpleName(), args);
    SignalHandler.register(LOG);
    JvmShutdownSafeguard.installAsShutdownHook(LOG);

    Configuration configuration = 
loadConfiguration(parseArguments(args));

    StandaloneSessionClusterEntrypoint entrypoint = new 
StandaloneSessionClusterEntrypoint(configuration);

    entrypoint.startCluster();
}
```

第一行代码其实就是打印有关环境的信息，如代码修订、当前用户、Java 版本和一些 JVM 参数。 第二行就是注册一些信号处理。 第三行就是安装安全关闭的钩子。

第四行先将 main 方法传入的参数进行解析，来看下方法 parseArguments(args):

```
protected static ClusterConfiguration parseArguments(String[] args)
{
    ParameterTool parameterTool = ParameterTool.fromArgs(args);

    final String configDir = parameterTool.get("configDir", "");

    final int restPort;

    final String portKey = "webui-port";
    if (parameterTool.has(portKey)) {
        restPort = Integer.valueOf(parameterTool.get(portKey));
    } else {
        restPort = -1;
    }

    return new ClusterConfiguration(configDir, restPort);
}
```

先由 ParameterTool 接收参数，然后拿到 configDir(配置文件夹目录) 和 webui-port 端口，最后重新构造一个 ClusterConfiguration 对象。

再来看下 loadConfiguration(parseArguments(args) 方法：

```
protected static Configuration
loadConfiguration(ClusterConfiguration clusterConfiguration) {
    final Configuration configuration =
GlobalConfiguration.loadConfiguration(clusterConfiguration.getConfi
gDir());

    final int restPort = clusterConfiguration.getRestPort();

    if (restPort >= 0) {
        configuration.setInteger(RestOptions.PORT, restPort);
    }

    return configuration;
}
```

clusterConfiguration.getConfigDir() 把配置文件夹所在的路径传入到 GlobalConfiguration.loadConfiguration() 方法中去，然后解析配置文件生成 GlobalConfiguration 对象，configuration.setInteger(RestOptions.PORT, restPort) 设置 Rest Port 端口。

继续分析 GlobalConfiguration.loadConfiguration() 方法内部实现：

```
public static Configuration loadConfiguration(final String
configDir) {
    return loadConfiguration(configDir, null);
}

/**
 * 通过该方法根据给定的目录文件夹来加载配置文件，如果还有动态的配置，则追加
 */
public static Configuration loadConfiguration(final String
configDir, @Nullable final Configuration dynamicProperties) {

    if (configDir == null) {
        throw new IllegalArgumentException("Given configuration
directory is null, cannot load configuration");
    }

    final File confDirFile = new File(configDir);
    if (!(confDirFile.exists())) {
        throw new IllegalConfigurationException(
            "The given configuration directory name '" + configDir
+
                "' (" + confDirFile.getAbsolutePath() + ") does not
describe an existing directory.");
    }

    // 获取到 flink yaml 配置文件 flink-conf.yaml
    final File yamlConfigFile = new File(confDirFile,
FLINK_CONF_FILENAME);

    if (!yamlConfigFile.exists()) {
        throw new IllegalConfigurationException(
            "The Flink config file '" + yamlConfigFile +
                "' (" + confDirFile.getAbsolutePath() + ") does not
exist.");
    }

    //加载 yaml 文件
    Configuration configuration = loadYAMLResource(yamlConfigFile);

    if (dynamicProperties != null) {
        configuration.addAll(dynamicProperties);
    }

    return configuration;
}
```

这个方法先检查传入到文件夹目录是否存在，如果不存在则抛出异常，重要的是加载配置文件的方法，继续跟进 loadYAMLResource() 方法：

```java
public static Configuration loadYAMLResource(File file) {
    final Configuration config = new Configuration();

    try (BufferedReader reader = new BufferedReader(new
InputStreamReader(new FileInputStream(file)))){

        String line;
        int lineNo = 0;
        while ((line = reader.readLine()) != null) {
            lineNo++;
            // 1. check for comments
            String[] comments = line.split("#", 2);
            String conf = comments[0].trim();

            // 2. get key and value
            if (conf.length() > 0) {
                String[] kv = conf.split(": ", 2);

                // skip line with no valid key-value pair
                if (kv.length == 1) {
                    LOG.warn("Error while trying to split key and
value in configuration file " + file + ":" + lineNo + ": \"" + line
+ "\"");
                    continue;
                }

                String key = kv[0].trim();
                String value = kv[1].trim();

                // sanity check
                if (key.length() == 0 || value.length() == 0) {
                    LOG.warn("Error after splitting key and value
in configuration file " + file + ":" + lineNo + ": \"" + line +
"\"");
                    continue;
                }

                LOG.info("Loading configuration property: {}, {}",
key, isSensitive(key) ? HIDDEN_CONTENT : value);
                config.setString(key, value);
            }
        }
    } catch (IOException e) {
        throw new RuntimeException("Error parsing YAML
configuration.", e);
    }


    return config;
}
```

上面方法就是获取了 yaml 文件里面的 key/value 值，然后存入 Configuration（其实就是一个 Map） 中。

然后继续回到 StandaloneSessionClusterEntrypoint 类的 Main 方法中将返回的 Configuration 对象作为参数构造 StandaloneSessionClusterEntrypoint 对象。

```
StandaloneSessionClusterEntrypoint entrypoint = new
StandaloneSessionClusterEntrypoint(configuration);
```

我们看下这个构造方法如下：

```
public StandaloneSessionClusterEntrypoint(Configuration
configuration) {
    super(configuration);
}

public SessionClusterEntrypoint(Configuration configuration) {
    super(configuration);
}

protected ClusterEntrypoint(Configuration configuration) {
    this.configuration =
generateClusterConfiguration(configuration);
    this.terminationFuture = new CompletableFuture<>();

    shutDownHook =
ShutdownHookUtil.addShutdownHook(this::cleanupDirectories,
getClass().getSimpleName(), LOG);
}
```

发现最终其实还是在 ClusterEntrypoint 集群入口类里参与里对象的构造，然后运行 ClusterEntrypoint 中的 startCluster 方法：

```
entrypoint.startCluster();

protected void startCluster() {
    LOG.info("Starting {}.", getClass().getSimpleName());

    try {
        configureFileSystems(configuration);

        SecurityContext securityContext =
installSecurityContext(configuration);

        securityContext.runSecured((Callable<Void>) () -> {
            //主方法
            runCluster(configuration);

            return null;
        });
    } catch (Throwable t) {
        LOG.error("Cluster initialization failed.", t);

        shutDownAndTerminate(
            STARTUP_FAILURE_RETURN_CODE,
            ApplicationStatus.FAILED,
            t.getMessage(),
            false);
    }
}
```

1、先来看下 `configureFileSystems(configuration)` 方法:

```
protected void configureFileSystems(Configuration configuration)
throws Exception {
    LOG.info("Install default filesystem.");

    try {
        FileSystem.initialize(configuration);
    } catch (IOException e) {
        throw new IOException("Error while setting the default " +
            "filesystem scheme from configuration.", e);
    }
}
```

该方法内部调用 `FileSystem.initialize()` 初始化配置文件中的共享文件设置。

2、installSecurityContext

根据全局的配置文件配置安全相关的配置，zk、Hadoop、用户等安全配置

```
protected SecurityContext installSecurityContext(Configuration
configuration) throws Exception {
    LOG.info("Install security context.");

    SecurityUtils.install(new
SecurityConfiguration(configuration));

    return SecurityUtils.getInstalledContext();
}
```

3、runCluster

这个方法是最关键的，里面会先初始化服务、写入 Job Manager 的地址和端口进入配置、开启就去那所有的组件（RPC 服务、HA 服务、blob 服务、心跳检查服务、metric 服务）

```
protected void runCluster(Configuration configuration) throws
Exception {
    synchronized (lock) {
        //初始化服务，做好准备条件
        initializeServices(configuration);

        //将 jobmanager 地址写入配置
        configuration.setString(JobManagerOptions.ADDRESS,
commonRpcService.getAddress());
        configuration.setInteger(JobManagerOptions.PORT,
commonRpcService.getPort());

        //开启相关的组件服务
        startClusterComponents(
            configuration, commonRpcService, haServices,
            blobServer, heartbeatServices, metricRegistry);

        dispatcher.getTerminationFuture().whenComplete(
            (Void value, Throwable throwable) -> {
                if (throwable != null) {
                    LOG.info("Could not properly terminate the
Dispatcher.", throwable);
                }

                // This is the general shutdown path. If a separate
more specific shutdown was
                // already triggered, this will do nothing
                shutDownAndTerminate(
                    SUCCESS_RETURN_CODE,
                    ApplicationStatus.SUCCEEDED,
                    throwable != null ? throwable.getMessage() :
null,
                    true);
            });
    }
}
```

3.1、initializeServices

初始化服务，创建共有的 RPC 服务、创建 HA 服务、开启 BlobServer 服务、创建心跳检查服务、创建 metric

```
protected void initializeServices(Configuration configuration)
throws Exception {
    LOG.info("Initializing cluster services.");

    synchronized (lock) {
        final String bindAddress =
configuration.getString(JobManagerOptions.ADDRESS);
        final String portRange = getRPCPortRange(configuration);

        //创建 RPC 服务
        commonRpcService = createRpcService(configuration,
bindAddress, portRange);

        // 配置JobManager地址和端口
        configuration.setString(JobManagerOptions.ADDRESS,
commonRpcService.getAddress());
        configuration.setInteger(JobManagerOptions.PORT,
commonRpcService.getPort());
        //创建 HA 服务
        haServices = createHaServices(configuration,
commonRpcService.getExecutor());
        //创建 blobServer
        blobServer = new BlobServer(configuration,
haServices.createBlobStore());
        blobServer.start();
        //创建心跳服务
        heartbeatServices = createHeartbeatServices(configuration);
        //创建 metric 注册服务
        metricRegistry = createMetricRegistry(configuration);

        //开启 MetricQueryService
        final ActorSystem actorSystem = ((AkkaRpcService)
commonRpcService).getActorSystem();
        metricRegistry.startQueryService(actorSystem, null);

        archivedExecutionGraphStore =
createSerializableExecutionGraphStore(configuration,
commonRpcService.getScheduledExecutor());

        clusterInformation = new ClusterInformation(
            commonRpcService.getAddress(), blobServer.getPort());

        transientBlobCache = new TransientBlobCache(
            configuration,
            new InetSocketAddress(
                clusterInformation.getBlobServerHostname(),
                clusterInformation.getBlobServerPort()));
    }
}
```

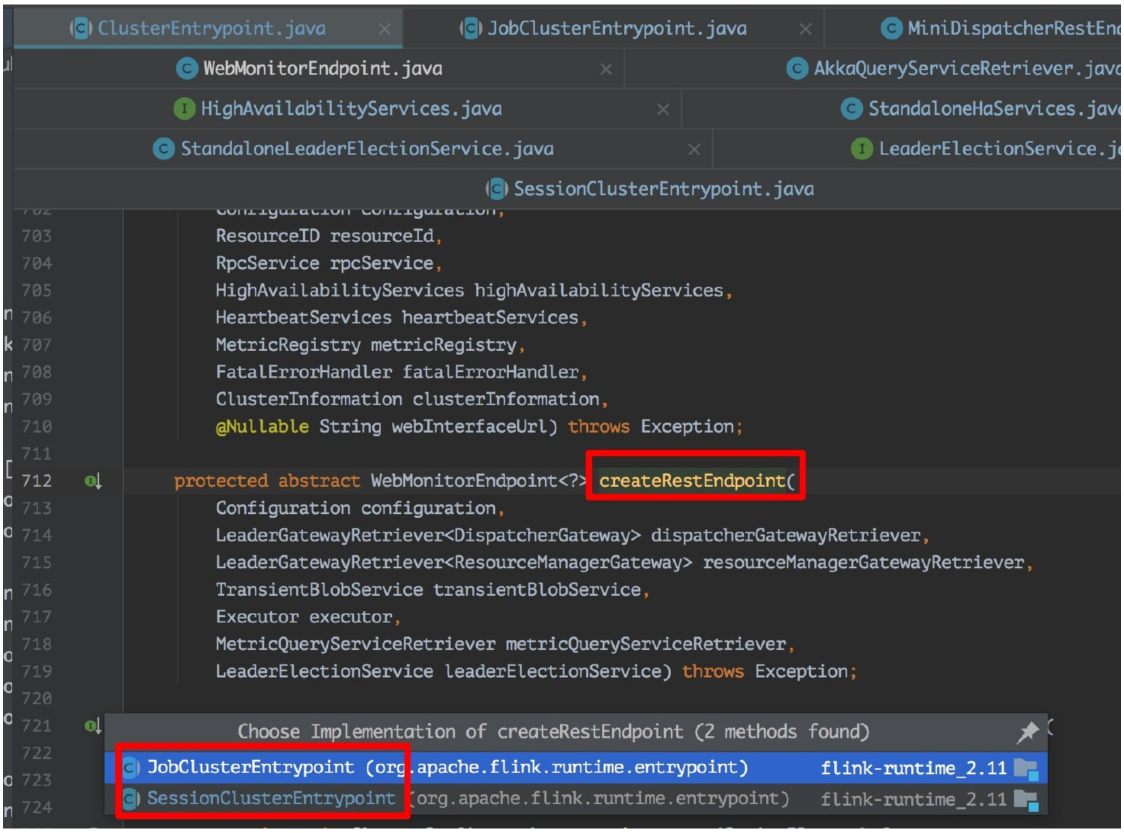
3.2、startClusterComponents

上一步初始化了服务，这步就是启动组件和服务：

```
startClusterComponents(configuration, commonRpcService, haServices,
                blobServer, heartbeatServices, metricRegistry);

protected void startClusterComponents(Configuration configuration,
            RpcService rpcService, HighAvailabilityServices
highAvailabilityServices,
            BlobServer blobServer, HeartbeatServices
heartbeatServices,
            MetricRegistry metricRegistry) throws Exception {
        synchronized (lock) {
            dispatcherLeaderRetrievalService =
highAvailabilityServices.getDispatcherLeaderRetriever();

            resourceManagerRetrievalService =
highAvailabilityServices.getResourceManagerLeaderRetriever();

            //调度程序网关
            LeaderGatewayRetriever<DispatcherGateway>
dispatcherGatewayRetriever = new RpcGatewayRetriever<>(
                rpcService, DispatcherGateway.class,
DispatcherId::fromUuid,
                10, Time.milliseconds(50L));
            //资源管理网关
            LeaderGatewayRetriever<ResourceManagerGateway>
resourceManagerGatewayRetriever = new RpcGatewayRetriever<>
(rpcService,ResourceManagerGateway.class,

ResourceManagerId::fromUuid,10,Time.milliseconds(50L));

            // TODO: Remove once we have ported the MetricFetcher
to the RpcEndpoint
            final ActorSystem actorSystem = ((AkkaRpcService)
rpcService).getActorSystem();
            final Time timeout =
Time.milliseconds(configuration.getLong(WebOptions.TIMEOUT));

            //创建 Rest Endpoint
            webMonitorEndpoint = createRestEndpoint(
                configuration,dispatcherGatewayRetriever,
resourceManagerGatewayRetriever,
                transientBlobCache,rpcService.getExecutor(),
                new AkkaQueryServiceRetriever(actorSystem,
timeout),
highAvailabilityServices.getWebMonitorLeaderElectionService());


            LOG.debug("Starting Dispatcher REST endpoint.");
            webMonitorEndpoint.start(); //开启

            //创建资源管理器
            resourceManager = createResourceManager(
                configuration,ResourceID.generate(),
                rpcService,highAvailabilityServices,
                heartbeatServices,metricRegistry,
                this,clusterInformation,
                webMonitorEndpoint.getRestBaseUrl());

            //job manager 的 metric 数据，主要是 JVM 的数据
（ClassLoader、GarbageCollector、Memory、Threads、CPU）
            jobManagerMetricGroup =
MetricUtils.instantiateJobManagerMetricGroup(metricRegistry,
rpcService.getAddress());

            //JobManager 中存储已完成的作业存档
            final HistoryServerArchivist historyServerArchivist =
HistoryServerArchivist.createHistoryServerArchivist(configuration,
webMonitorEndpoint);

            //创建调度器
            dispatcher = createDispatcher(
                configuration,rpcService,highAvailabilityServices,

resourceManager.getSelfGateway(ResourceManagerGateway.class),
                blobServer,heartbeatServices,
jobManagerMetricGroup,

metricRegistry.getMetricQueryServicePath(),archivedExecutionGraphSt
ore,

this,webMonitorEndpoint.getRestBaseUrl(),historyServerArchivist,thi
s);

            LOG.debug("Starting ResourceManager.");
            resourceManager.start();

resourceManagerRetrievalService.start(resourceManagerGatewayRetriev
er);

            LOG.debug("Starting Dispatcher.");
            //启动调度器
            dispatcher.start();
```

```
dispatcherLeaderRetrievalService.start(dispatcherGatewayRetriever);
        }
    }
```

1、上面中 `createRestEndpoint()` 中创建 Rest Endpoint，这个方法在 JobClusterEntrypoint 和 SessionClusterEntrypoint 中都有具体的实现


ClusterEntrypoint.java
JobClusterEntrypoint.java
MiniDispatcherRestEnd
WebMonitorEndpoint.java
AkkaQueryServiceRetriever.java
HighAvailabilityServices.java
StandaloneHaServices.java
StandaloneLeaderElectionService.java
LeaderElectionService.ja
SessionClusterEntrypoint.java

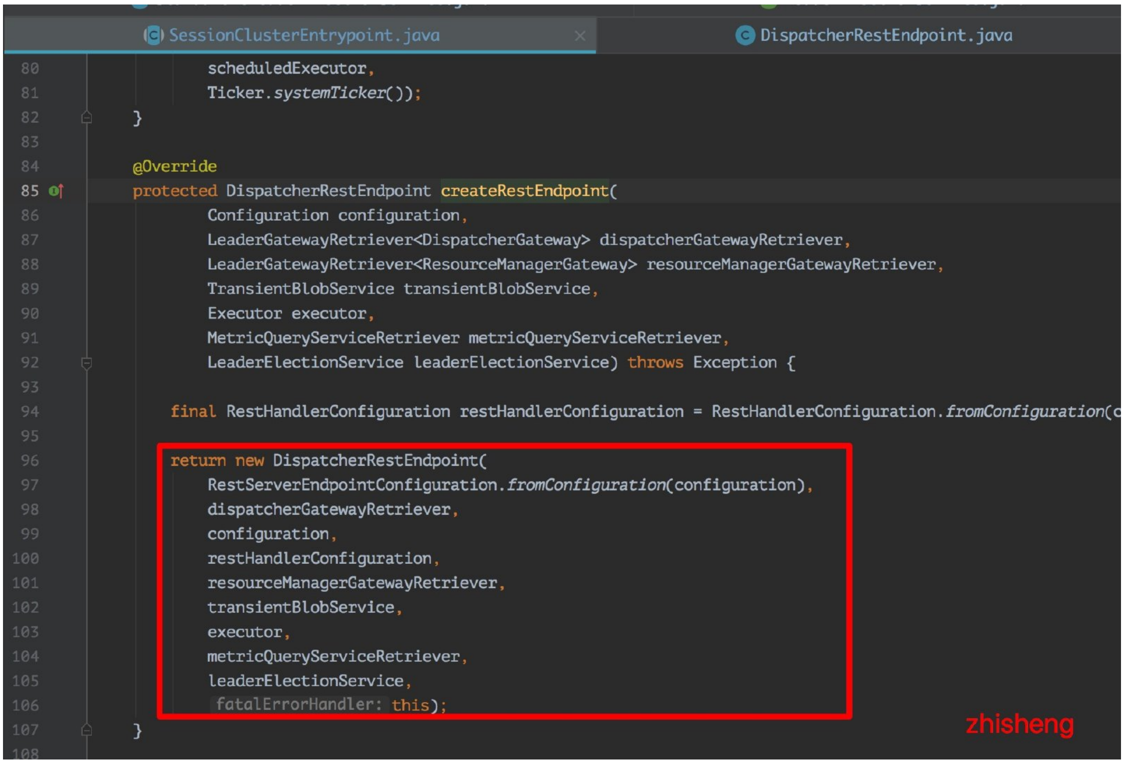
```
        Configuration configuration,
        ResourceID resourceId,
        RpcService rpcService,
        HighAvailabilityServices highAvailabilityServices,
        HeartbeatServices heartbeatServices,
        MetricRegistry metricRegistry,
        FatalErrorHandler fatalErrorHandler,
        ClusterInformation clusterInformation,
        @Nullable String webInterfaceUrl) throws Exception;

    protected abstract WebMonitorEndpoint<?> createRestEndpoint(
        Configuration configuration,
        LeaderGatewayRetriever<DispatcherGateway> dispatcherGatewayRetriever,
        LeaderGatewayRetriever<ResourceManagerGateway> resourceManagerGatewayRetriever,
        TransientBlobService transientBlobService,
        Executor executor,
        MetricQueryServiceRetriever metricQueryServiceRetriever,
        LeaderElectionService leaderElectionService) throws Exception;
```

Choose Implementation of createRestEndpoint (2 methods found)

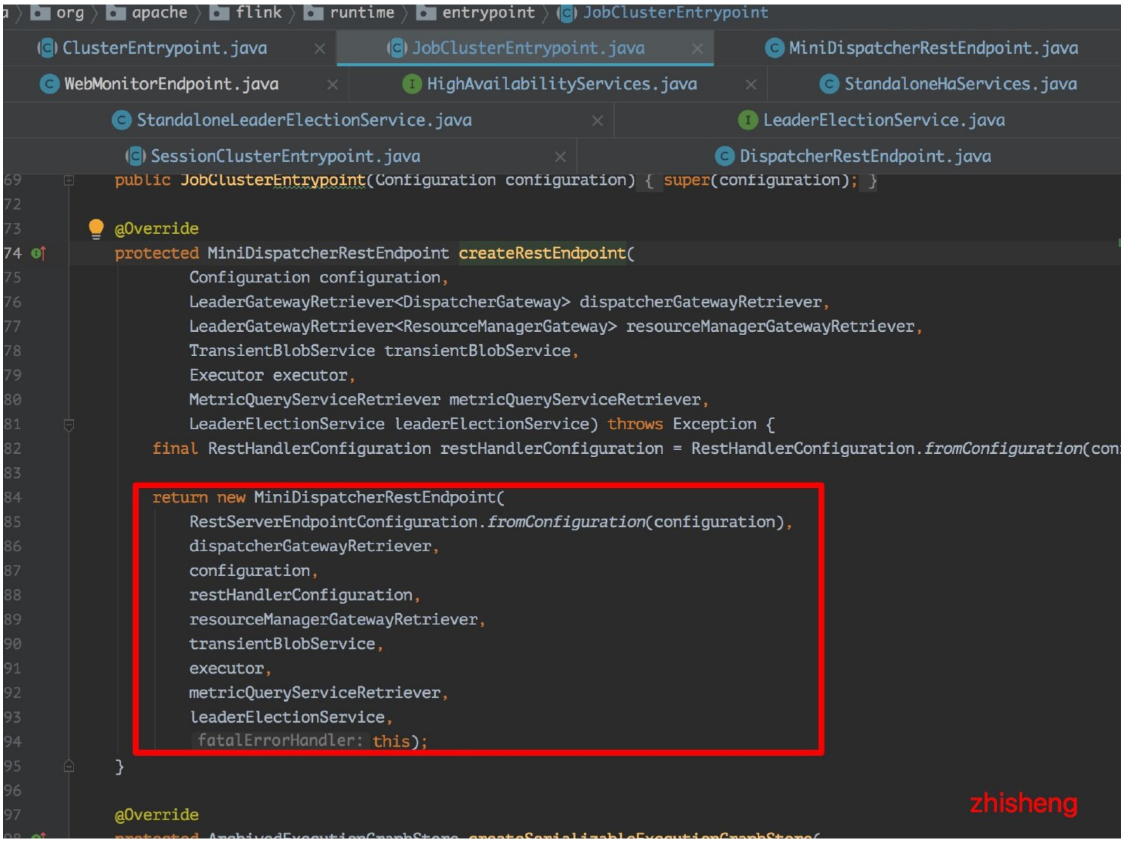
JobClusterEntrypoint (org.apache.flink.runtime.entrypoint)   flink-runtime_2.11

SessionClusterEntrypoint (org.apache.flink.runtime.entrypoint)   flink-runtime_2.11

实现如下：



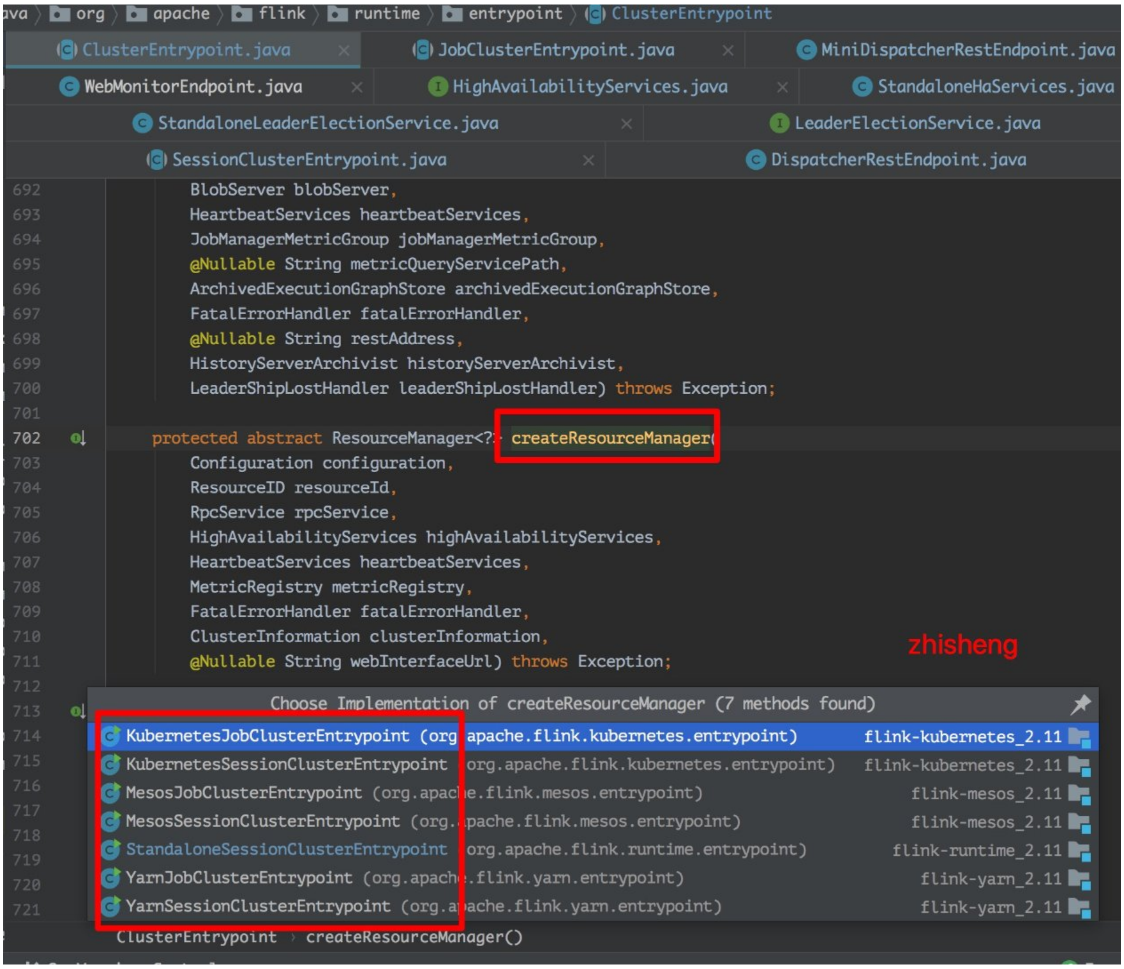
```java
            scheduledExecutor,
            Ticker.systemTicker());
    }

    @Override
    protected DispatcherRestEndpoint createRestEndpoint(
            Configuration configuration,
            LeaderGatewayRetriever<DispatcherGateway> dispatcherGatewayRetriever,
            LeaderGatewayRetriever<ResourceManagerGateway> resourceManagerGatewayRetriever,
            TransientBlobService transientBlobService,
            Executor executor,
            MetricQueryServiceRetriever metricQueryServiceRetriever,
            LeaderElectionService leaderElectionService) throws Exception {

        final RestHandlerConfiguration restHandlerConfiguration = RestHandlerConfiguration.fromConfiguration(c

        return new DispatcherRestEndpoint(
            RestServerEndpointConfiguration.fromConfiguration(configuration),
            dispatcherGatewayRetriever,
            configuration,
            restHandlerConfiguration,
            resourceManagerGatewayRetriever,
            transientBlobService,
            executor,
            metricQueryServiceRetriever,
            leaderElectionService,
             fatalErrorHandler: this);
    }
```


org
apache
flink
runtime
entrypoint
JobClusterEntrypoint
ClusterEntrypoint.java
JobClusterEntrypoint.java
MiniDispatcherRestEndpoint.java
WebMonitorEndpoint.java
HighAvailabilityServices.java
StandaloneHaServices.java
StandaloneLeaderElectionService.java
LeaderElectionService.java
SessionClusterEntrypoint.java
DispatcherRestEndpoint.java
zhisheng

```java
    public JobClusterEntrypoint(Configuration configuration) { super(configuration); }

    @Override
    protected MiniDispatcherRestEndpoint createRestEndpoint(
            Configuration configuration,
            LeaderGatewayRetriever<DispatcherGateway> dispatcherGatewayRetriever,
            LeaderGatewayRetriever<ResourceManagerGateway> resourceManagerGatewayRetriever,
            TransientBlobService transientBlobService,
            Executor executor,
            MetricQueryServiceRetriever metricQueryServiceRetriever,
            LeaderElectionService leaderElectionService) throws Exception {
        final RestHandlerConfiguration restHandlerConfiguration = RestHandlerConfiguration.fromConfiguration(con

        return new MiniDispatcherRestEndpoint(
            RestServerEndpointConfiguration.fromConfiguration(configuration),
            dispatcherGatewayRetriever,
            configuration,
            restHandlerConfiguration,
            resourceManagerGatewayRetriever,
            transientBlobService,
            executor,
            metricQueryServiceRetriever,
            leaderElectionService,
             fatalErrorHandler: this);
    }

    @Override
    protected ArchivedExecutionGraphStore createSerializableExecutionGraphStore(
```

后面再分析下里面的具体实现，这里不能跟的太深了，阅读源码要学会跳进去，更要学会跳出来！

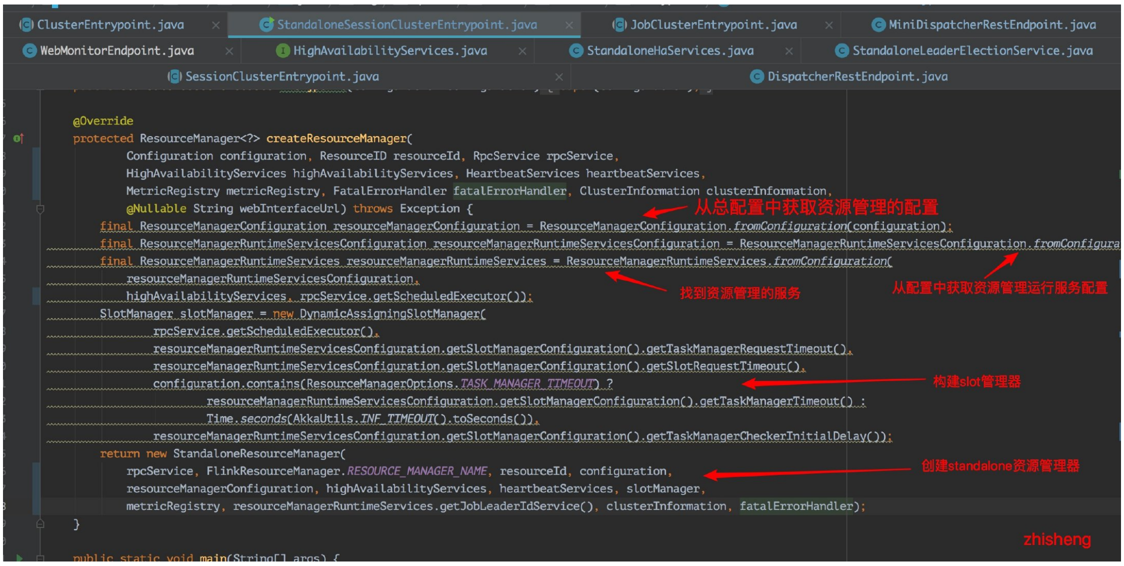
2、上面中 `createResourceManager()` 中创建资源管理器，这个方法在下面这些类中都有自己的实现：


ava 〉 org 〉 apache 〉 flink 〉 runtime 〉 entrypoint 〉 ClusterEntrypoint
ClusterEntrypoint.java
JobClusterEntrypoint.java
MiniDispatcherRestEndpoint.java
WebMonitorEndpoint.java
HighAvailabilityServices.java
StandaloneHaServices.java
StandaloneLeaderElectionService.java
LeaderElectionService.java
SessionClusterEntrypoint.java
DispatcherRestEndpoint.java
zhisheng
ClusterEntrypoint 〉 createResourceManager()

```
			BlobServer blobServer,
			HeartbeatServices heartbeatServices,
			JobManagerMetricGroup jobManagerMetricGroup,
			@Nullable String metricQueryServicePath,
			ArchivedExecutionGraphStore archivedExecutionGraphStore,
			FatalErrorHandler fatalErrorHandler,
			@Nullable String restAddress,
			HistoryServerArchivist historyServerArchivist,
			LeaderShipLostHandler leaderShipLostHandler) throws Exception;

	protected abstract ResourceManager<?> createResourceManager(
			Configuration configuration,
			ResourceID resourceId,
			RpcService rpcService,
			HighAvailabilityServices highAvailabilityServices,
			HeartbeatServices heartbeatServices,
			MetricRegistry metricRegistry,
			FatalErrorHandler fatalErrorHandler,
			ClusterInformation clusterInformation,
			@Nullable String webInterfaceUrl) throws Exception;
```

```
Choose Implementation of createResourceManager (7 methods found)
KubernetesJobClusterEntrypoint (org.apache.flink.kubernetes.entrypoint)      flink-kubernetes_2.11
KubernetesSessionClusterEntrypoint (org.apache.flink.kubernetes.entrypoint)  flink-kubernetes_2.11
MesosJobClusterEntrypoint (org.apache.flink.mesos.entrypoint)                    flink-mesos_2.11
MesosSessionClusterEntrypoint (org.apache.flink.mesos.entrypoint)                flink-mesos_2.11
StandaloneSessionClusterEntrypoint (org.apache.flink.runtime.entrypoint)       flink-runtime_2.11
YarnJobClusterEntrypoint (org.apache.flink.yarn.entrypoint)                       flink-yarn_2.11
YarnSessionClusterEntrypoint (org.apache.flink.yarn.entrypoint)                   flink-yarn_2.11
```

这里先在 `StandaloneSessionClusterEntrypoint` 中先看下它的创建资源管理器是咋实现的？


ClusterEntrypoint.java
StandaloneSessionClusterEntrypoint.java
JobClusterEntrypoint.java
MiniDispatcherRestEndpoint.java
WebMonitorEndpoint.java
HighAvailabilityServices.java
StandaloneHaServices.java
StandaloneLeaderElectionService.java
SessionClusterEntrypoint.java
DispatcherRestEndpoint.java
zhisheng

```java
@Override
protected ResourceManager<?> createResourceManager(
        Configuration configuration, ResourceID resourceId, RpcService rpcService,
        HighAvailabilityServices highAvailabilityServices, HeartbeatServices heartbeatServices,
        MetricRegistry metricRegistry, FatalErrorHandler fatalErrorHandler, ClusterInformation clusterInformation,
        @Nullable String webInterfaceUrl) throws Exception {
    final ResourceManagerConfiguration resourceManagerConfiguration = ResourceManagerConfiguration.fromConfiguration(configuration);
    final ResourceManagerRuntimeServicesConfiguration resourceManagerRuntimeServicesConfiguration = ResourceManagerRuntimeServicesConfiguration.fromConfigura
    final ResourceManagerRuntimeServices resourceManagerRuntimeServices = ResourceManagerRuntimeServices.fromConfiguration(
        resourceManagerRuntimeServicesConfiguration,
        highAvailabilityServices, rpcService.getScheduledExecutor());
    SlotManager slotManager = new DynamicAssigningSlotManager(
            rpcService.getScheduledExecutor(),
            resourceManagerRuntimeServicesConfiguration.getSlotManagerConfiguration().getTaskManagerRequestTimeout(),
            resourceManagerRuntimeServicesConfiguration.getSlotManagerConfiguration().getSlotRequestTimeout(),
            configuration.contains(ResourceManagerOptions.TASK_MANAGER_TIMEOUT) ?
                    resourceManagerRuntimeServicesConfiguration.getSlotManagerConfiguration().getTaskManagerTimeout() :
                    Time.seconds(AkkaUtils.INF_TIMEOUT().toSeconds()),
            resourceManagerRuntimeServicesConfiguration.getSlotManagerConfiguration().getTaskManagerCheckerInitialDelay());
    return new StandaloneResourceManager(
        rpcService, FlinkResourceManager.RESOURCE_MANAGER_NAME, resourceId, configuration,
        resourceManagerConfiguration, highAvailabilityServices, heartbeatServices, slotManager,
        metricRegistry, resourceManagerRuntimeServices.getJobLeaderIdService(), clusterInformation, fatalErrorHandler);
}

public static void main(String[] args) {
```

从总配置中获取资源管理的配置

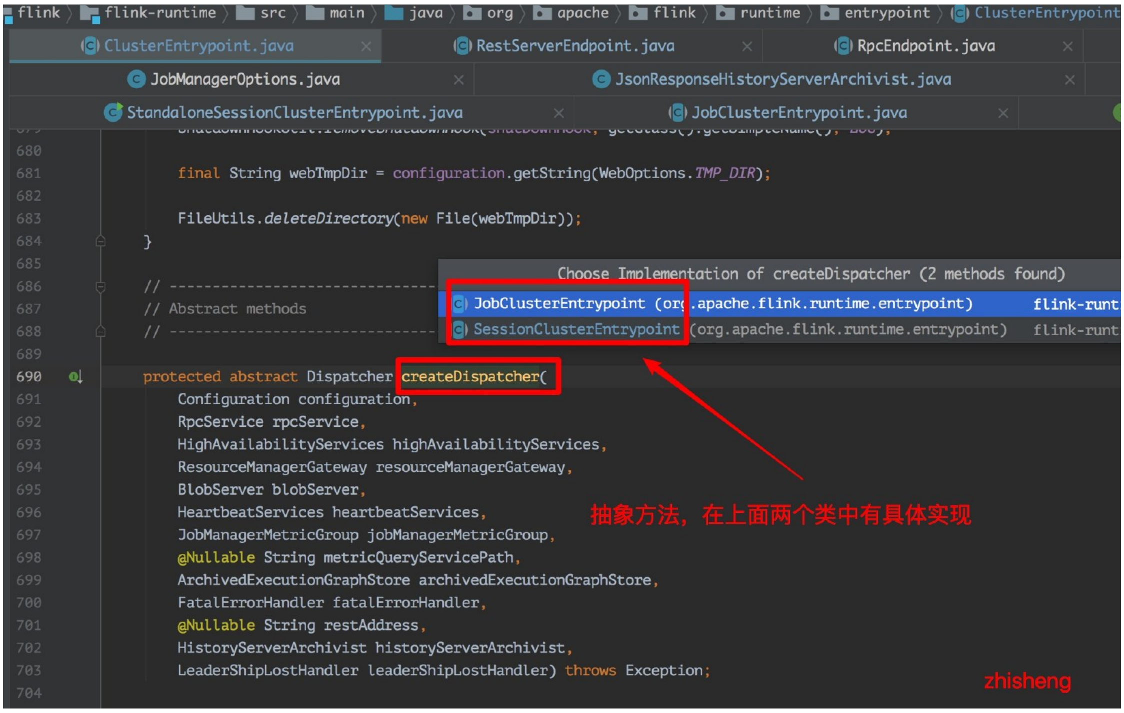
找到资源管理的服务

从配置中获取资源管理运行服务配置

构建slot管理器

创建standalone资源管理器

3、上面中 `createDispatcher()` 中创建调度器，这是个抽象方法，在 JobClusterEntrypoint 和 SessionClusterEntrypoint 中都有具体的实现。

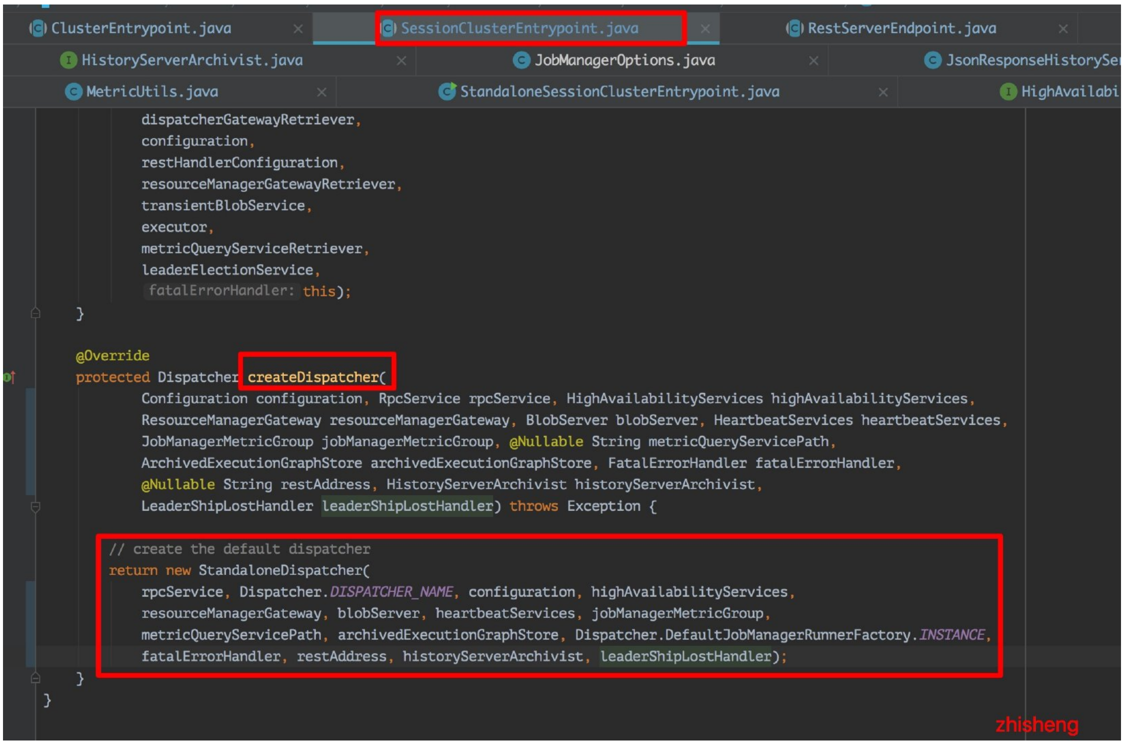


```java
        final String webTmpDir = configuration.getString(WebOptions.TMP_DIR);

        FileUtils.deleteDirectory(new File(webTmpDir));
    }

    // ------------------------------
    // Abstract methods
    // ------------------------------

    protected abstract Dispatcher createDispatcher(
        Configuration configuration,
        RpcService rpcService,
        HighAvailabilityServices highAvailabilityServices,
        ResourceManagerGateway resourceManagerGateway,
        BlobServer blobServer,
        HeartbeatServices heartbeatServices,
        JobManagerMetricGroup jobManagerMetricGroup,
        @Nullable String metricQueryServicePath,
        ArchivedExecutionGraphStore archivedExecutionGraphStore,
        FatalErrorHandler fatalErrorHandler,
        @Nullable String restAddress,
        HistoryServerArchivist historyServerArchivist,
        LeaderShipLostHandler leaderShipLostHandler) throws Exception;
```

```
Choose Implementation of createDispatcher (2 methods found)
JobClusterEntrypoint (org.apache.flink.runtime.entrypoint)       flink-runt
SessionClusterEntrypoint (org.apache.flink.runtime.entrypoint)   flink-runt
```

抽象方法，在上面两个类中有具体实现

比如 SessionClusterEntrypoint 中是这样的：

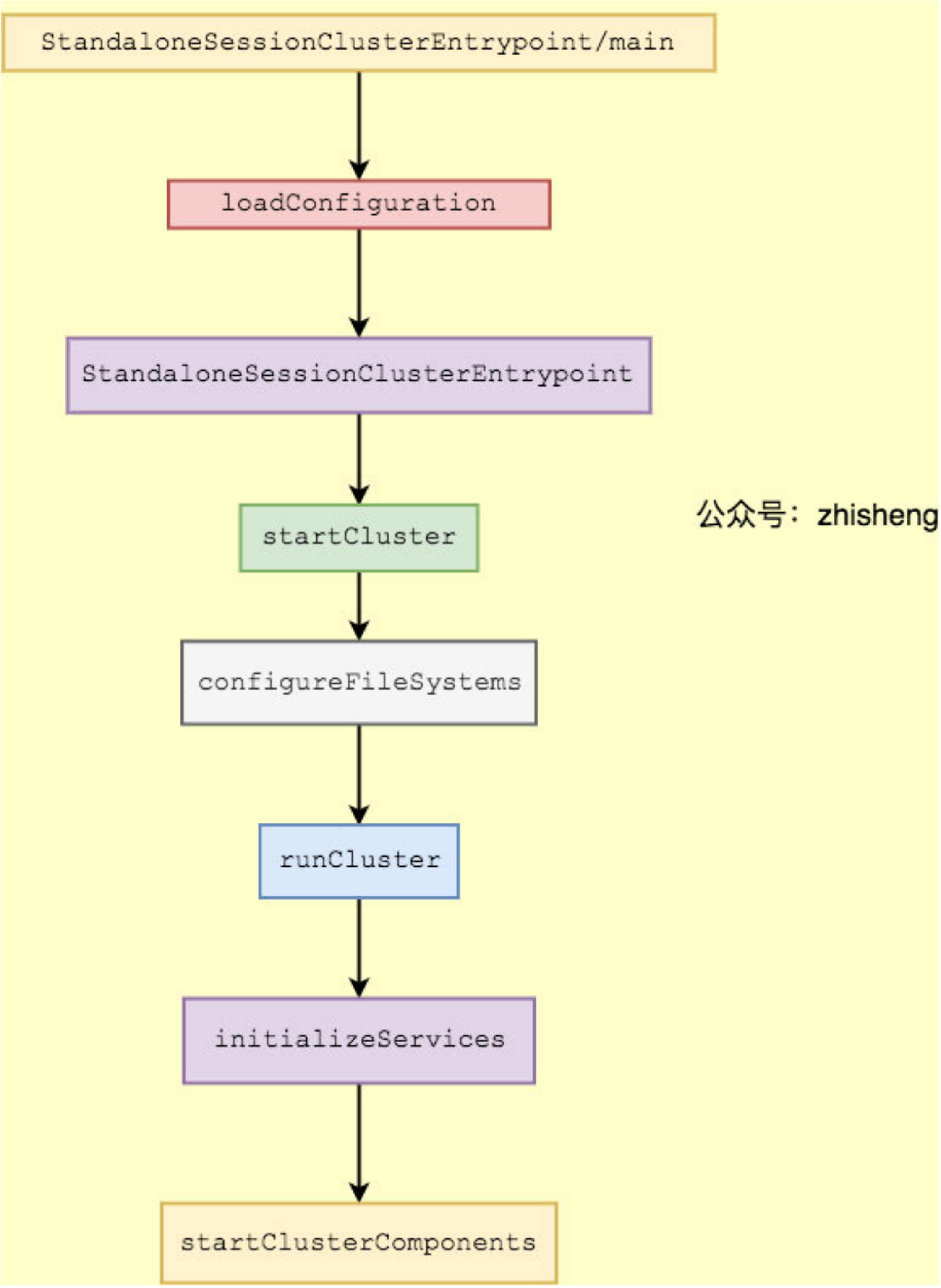


```java
				dispatcherGatewayRetriever,
				configuration,
				restHandlerConfiguration,
				resourceManagerGatewayRetriever,
				transientBlobService,
				executor,
				metricQueryServiceRetriever,
				leaderElectionService,
				fatalErrorHandler: this);
	}

	@Override
	protected Dispatcher createDispatcher(
			Configuration configuration, RpcService rpcService, HighAvailabilityServices highAvailabilityServices,
			ResourceManagerGateway resourceManagerGateway, BlobServer blobServer, HeartbeatServices heartbeatServices,
			JobManagerMetricGroup jobManagerMetricGroup, @Nullable String metricQueryServicePath,
			ArchivedExecutionGraphStore archivedExecutionGraphStore, FatalErrorHandler fatalErrorHandler,
			@Nullable String restAddress, HistoryServerArchivist historyServerArchivist,
			LeaderShipLostHandler leaderShipLostHandler) throws Exception {

		// create the default dispatcher
		return new StandaloneDispatcher(
			rpcService, Dispatcher.DISPATCHER_NAME, configuration, highAvailabilityServices,
			resourceManagerGateway, blobServer, heartbeatServices, jobManagerMetricGroup,
			metricQueryServicePath, archivedExecutionGraphStore, Dispatcher.DefaultJobManagerRunnerFactory.INSTANCE,
			fatalErrorHandler, restAddress, historyServerArchivist, leaderShipLostHandler);
	}
}
```

终于这个 `startClusterComponents()` 结束了，开心！继续回到 `runCluster()` 方法，这个方法里面其他的代码也没什么需要特别说明的，终于讲完了 `StandaloneSessionClusterEntrypoint` 类的整个流程！

总结

用图简单的画了一下流程，但是内部的启动流程还是很复杂的，需要我们好好跟代码跟下去，更要能跟出来！



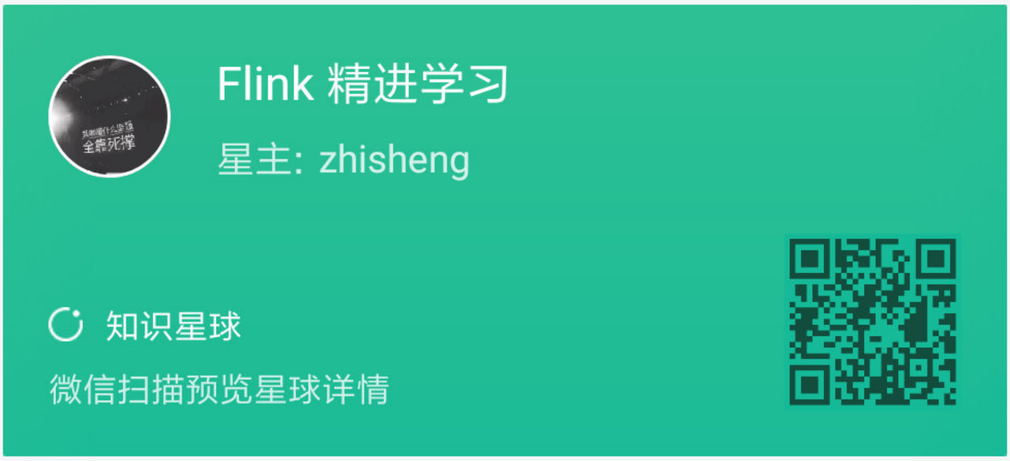
用图简单的画了一下流程，但是内部的启动流程还是很复杂的，需要我们好好跟代码跟下去，更要能跟出来！

中文源码分析

https://github.com/zhisheng17/flink

相关

更多私密文章和资料请加知识星球！



Flink 精进学习

星主: zhisheng

知识星球

微信扫描预览星球详情