

Elasticsearch 插件管理

插件注册

```
public class Bootstrap {  
  
    private Node node;  
  
    private static volatile Thread keepAliveThread;  
    private static volatile CountDownLatch keepAliveLatch;  
  
    private static Bootstrap bootstrap;  
  
    private void setup(boolean addShutdownHook, Tuple<Settings, Environment> tuple) throws Exception {  
        Settings settings = tuple.v1();  
        initializeNatives(settings.getAsBoolean("bootstrap.mlockall", false),  
            settings.getAsBoolean("bootstrap.ctrlhandler", true));  
  
        // We do not need to reload system properties here as we have already applied them in building the settings and  
        // reloading could cause multiple prompts to the user for values if a system property was specified with a prompt  
        // placeholder  
        Settings nodeSettings = ImmutableSettings.settingsBuilder()  
            .put(settings)  
            .put(InternalSettingsPreparer.IGNORE_SYSTEM_PROPERTIES_SETTING, true)  
            .build();  
        NodeBuilder nodeBuilder = NodeBuilder.nodeBuilder().settings(nodeSettings).loadConfigSettings(false);  
        node = nodeBuilder.build();  
        if (addShutdownHook) {  
            Runtime.getRuntime().addShutdownHook(new Thread() {  
                @Override public void run() {  
                    node.close();  
                }  
            });  
        }  
    }  
}
```

```
m es-plugin-framework x Bootstrap.java x C:\java\...\NativeScriptEngineService.java x NodeBuilder.java x m elasticsearch x In  
142 }  
143  
144 /**  
145  * The cluster name this node is part of (maps to the <tt>cluster.name</tt> setting). Defaults  
146  * to <tt>elasticsearch</tt>.  
147  *  
148  * @param clusterName The cluster name this node is part of.  
149  */  
150 public NodeBuilder clusterName(String clusterName) {  
151     settings.put("cluster.name", clusterName);  
152     return this;  
153 }  
154  
155 /**  
156  * Builds the node without starting it.  
157  */  
158 public Node build() {  
159     return new InternalNode(settings.build(), loadConfigSettings);  
160 }  
161
```

```

134
135 public InternalNode(Settings preparedSettings, boolean loadConfigSettings) throws Elasticse
136     final Settings pSettings = settingsBuilder().put(preparedSettings)
137         .put(Client.CLIENT_TYPE_SETTING, CLIENT_TYPE).build();
138     Tuple<Settings, Environment> tuple = InternalSettingsPreparer.prepareSettings(pSettings);
139     tuple = new Tuple<>(TribeService.processSettings(tuple.v1()), tuple.v2());
140
141     // The only place we can actually fake the version a node is running on:
142     Version version = pSettings.getAsVersion("tests.mock.version", Version.CURRENT);
143
144     ESLogger logger = Loggers.getLogger(Node.class, tuple.v1().get("name"));
145     logger.info("version[{}], pid[{}], build[{}]/{}", version, JvmInfo.jvmInfo().pid(), Bui
146
147     logger.info("initializing ...");
148
149     if (logger.isDebugEnabled()) {
150         Environment env = tuple.v2();
151         logger.debug("using home [{}], config [{}], data [{}], logs [{}], work [{}], plugin
152             env.homeFile(), env.configFile(), Arrays.toString(env.dataFiles()), env.log
153             env.workFile(), env.pluginsFile());
154     }
155
156     // workaround for LUCENE-6482
157     Codec.availableCodecs();
158
159     this.pluginsService = new PluginsService(tuple.v1(), tuple.v2());
160     this.settings = pluginsService.updatedSettings();
161     // create the environment based on the finalized (processed) view of the settings
162     this.environment = new Environment(this.settings());
163

```

```

public PluginsService(Settings settings, Environment environment) {
    super(settings);
    this.environment = environment;
    this.checkLucene = componentSettings.getAsBoolean("check_lucene", true);
    this.esPluginPropertiesFile = componentSettings.get(ES_PLUGIN_PROPERTIES_FILE_KEY, ES_PLUGIN_PROPERTIES);
    this.loadClasspathPlugins = componentSettings.getAsBoolean(LOAD_PLUGIN_FROM_CLASSPATH, true);

    ImmutableList.Builder<Tuple<PluginInfo, Plugin>> tupleBuilder = ImmutableList.builder();

    // First we load all the default plugins from the settings
    String[] defaultPluginsClasses = settings.getAsArray("plugin.types");
    for (String pluginClass : defaultPluginsClasses) {
        Plugin plugin = loadPlugin(pluginClass, settings);
        PluginInfo pluginInfo = new PluginInfo(plugin.name(), plugin.description(), hasSite(plugin.name()), true, PluginInfo.VERSION_NOT_AVAILABLE);
        if (logger.isTraceEnabled()) {
            logger.trace("plugin loaded from settings [{}]", pluginInfo);
        }
        tupleBuilder.add(new Tuple<>(pluginInfo, plugin));
    }

    // now, find all the ones that are in the classpath
    loadPluginsIntoClassLoader();
    if (loadClasspathPlugins) {
        tupleBuilder.addAll(loadPluginsFromClasspath(settings));
    }
    this.plugins = tupleBuilder.build();

    // We need to build a List of jvm and site plugins for checking mandatory plugins
    Map<String, Plugin> jvmPlugins = Maps.newHashMap();

```

es-
plugin.properties

```

plugin.properties ×  ESPluginFramework.java ×  Bootstrap.java ×  java\...\NativeScriptEngineService.java ×  NodeBuilder.java ×  M Elasticsearch ×  InternalNode.java ×  PluginsService.java ×  Immut
114     loadPluginsIntoClassLoader();
115     if (loadClasspathPlugins) {
116         tupleBuilder.addAll(loadPluginsFromClasspath(settings));
117     }
118     this.plugins = tupleBuilder.build();
119
120     // We need to build a List of jvm and site plugins for checking mandatory plugins
121     Map<String, Plugin> jvmPlugins = Maps.newHashMap();
122     List<String> sitePlugins = Lists.newArrayList();
123
124     for (Tuple<PluginInfo, Plugin> tuple : this.plugins) {
125         jvmPlugins.put(tuple.v2().name(), tuple.v2());
126         if (tuple.v1().isSite()) {
127             sitePlugins.add(tuple.v1().getName());
128         }
129     }
130
131     // we load site plugins
132     ImmutableList<Tuple<PluginInfo, Plugin>> tuples = loadSitePlugins();
133     for (Tuple<PluginInfo, Plugin> tuple : tuples) {
134         sitePlugins.add(tuple.v1().getName());
135     }
136
137     // Checking expected plugins
138     String[] mandatoryPlugins = settings.getAsArray("plugin.mandatory", null);
139     if (mandatoryPlugins != null) {
140         Set<String> missingPlugins = Sets.newHashSet();
141         for (String mandatoryPlugin : mandatoryPlugins) {
142             if (!jvmPlugins.containsKey(mandatoryPlugin) && !sitePlugins.contains(mandatoryPlugin) && !missingPlugins.contains(mandatoryPlugin)) {
143                 missingPlugins.add(mandatoryPlugin);
144             }
145         }
146         if (!missingPlugins.isEmpty()) {
147             throw new ElasticsearchException("Missing mandatory plugins [" + Strings.collectionToDelimitedString(missingPlugins, ", ") + "]");
148         }
149     }
150
151     logger.info("loaded {}, sites {}", jvmPlugins.keySet(), sitePlugins);
152

```

加载plugin实例

我们的plugin一般都是jvmPlugins

各种校验功能

```

bootstrap.java ×  C:\java\...\NativeScriptEngineService.java ×  NodeBuilder.java ×  M Elasticsearch ×  InternalNode.java ×  PluginsService.java ×  InjectorBuilder.java
334     private void loadPluginsIntoClassLoader() {
335         File pluginsDirectory = environment.pluginsFile();
336         if (!isAccessibleDirectory(pluginsDirectory, logger)) {
337             return;
338         }
339
340         ClassLoader classLoader = settings.getClassLoader();
341         Class classLoaderClass = classLoader.getClass();
342         Method addURL = null;
343         while (!classLoaderClass.equals(Object.class)) {
344             try {
345                 addURL = classLoaderClass.getDeclaredMethod("addURL", URL.class);
346                 addURL.setAccessible(true);
347                 break;
348             } catch (NoSuchMethodException e) {
349                 // no method, try the parent
350                 classLoaderClass = classLoaderClass.getSuperclass();
351             }
352         }
353         if (addURL == null) {
354             logger.debug("failed to find addURL method on classLoader [" + classLoader + "] to add methods");
355             return;
356         }
357
358         for (File plugin : pluginsDirectory.listFiles()) {
359             // We check that subdirs are directories and readable
360             if (!isAccessibleDirectory(plugin, logger)) {
361                 continue;
362             }
363
364             logger.trace("--- adding plugin {}", plugin.getAbsolutePath());
365

```

获取系统配置的插件目录

遍历目录获取子目录

跳过子目录

```

try {
    // add the root
    addURL.invoke(classLoader, plugin.toURI().toURL());
    // gather files to add
    List<File> libFiles = Lists.newArrayList();
    if (plugin.listFiles() != null) {
        libFiles.addAll(Arrays.asList(plugin.listFiles()));
    }
    File libLocation = new File(plugin, "lib");
    File listing[] = libLocation.listFiles();
    if (listing != null) {
        libFiles.addAll(Arrays.asList(listing));
    }

    // if there are jars in it, add it as well
    for (File libFile : libFiles) {
        if (!(libFile.getName().endsWith(".jar") || libFile.getName().endsWith(".zip"))) {
            continue;
        }
        addURL.invoke(classLoader, libFile.toURI().toURL());
    }
} catch (Throwable e) {
    logger.warn("failed to add plugin [" + plugin + "]", e);
}

```

加入classloader备用

```

private ImmutableList<Tuple<PluginInfo, Plugin>> loadPluginsFromClasspath(Settings settings) {
    ImmutableList.Builder<Tuple<PluginInfo, Plugin>> plugins = ImmutableList.builder();

    // Trying JVM plugins: looking for es-plugin.properties files
    try {
        Enumeration<URL> pluginUrls = settings.getClassLoader().getResources(esPluginPropertiesFile);

        // use a set for uniqueness as some classloaders such as groovy's can return the same URL multiple times and
        // these plugins should only be loaded once
        HashSet<URL> uniqueUrls = new HashSet<>(Collections.list(pluginUrls));
        for (URL pluginUrl : uniqueUrls) {
            Properties pluginProps = new Properties();
            InputStream is = null;
            try {
                is = pluginUrl.openStream();
                pluginProps.load(is);
                String pluginClassName = pluginProps.getProperty("plugin");
                String pluginVersion = pluginProps.getProperty("version", PluginInfo.VERSION_NOT_AVAILABLE);
                Plugin plugin = loadPlugin(pluginClassName, settings);

                // Is it a site plugin as well? Does it have also an embedded _site structure
                File siteFile = new File(new File(environment.pluginsFile(), plugin.name()), "_site");
                boolean isSite = isAccessibleDirectory(siteFile, logger);
                if (logger.isTraceEnabled()) {
                    logger.trace("found a jvm plugin [{}], [{}]",
                        plugin.name(), plugin.description(), isSite ? ": with _site structure" : "");
                }

                PluginInfo pluginInfo = new PluginInfo(plugin.name(), plugin.description(), isSite, true, pluginVersion);

                plugins.add(new Tuple<PluginInfo, Plugin>(pluginInfo, plugin));
            } catch (Throwable e) {
                logger.warn("failed to load plugin from [" + pluginUrl + "]", e);
            } finally {
                IOUtils.closeWhileHandlingException(is);
            }
        }
    }
}

```

遍历所有包含es-plugin.properties的文件

读取内容

```

114 loadPluginsIntoClassLoader();
115 if (loadClasspathPlugins) {
116     tupleBuilder.addAll(loadPluginsFromClasspath(settings));
117 }
118 this.plugins = tupleBuilder.build();
119
120 // We need to build a List of jvm and site plugins for checking mandatory plugins
121 Map<String, Plugin> jvmPlugins = Maps.newHashMap();
122 List<String> sitePlugins = Lists.newArrayList();
123
124 for (Tuple<PluginInfo, Plugin> tuple : this.plugins) {
125     jvmPlugins.put(tuple.v2().name(), tuple.v2());
126     if (tuple.v1().isSite()) {
127         sitePlugins.add(tuple.v1().getName());
128     }
129 }
130
131 // we load site plugins
132 ImmutableList<Tuple<PluginInfo, Plugin>> tuples = loadSitePlugins();
133 for (Tuple<PluginInfo, Plugin> tuple : tuples) {
134     sitePlugins.add(tuple.v1().getName());
135 }
136
137 // Checking expected plugins
138 String[] mandatoryPlugins = settings.getAsArray("plugin.mandatory", null);
139 if (mandatoryPlugins != null) {
140     Set<String> missingPlugins = Sets.newHashSet();
141     for (String mandatoryPlugin : mandatoryPlugins) {
142         if (!jvmPlugins.containsKey(mandatoryPlugin) && !sitePlugins.contains(mandatoryPlugin) && !missingPlugins.contains(mandatoryPlugin)) {
143             missingPlugins.add(mandatoryPlugin);
144         }
145     }
146     if (!missingPlugins.isEmpty()) {
147         throw new ElasticsearchException("Missing mandatory plugins [" + Strings.collectionToDelimitedString(missingPlugins, ", ") + "]");
148     }
149 }
150
151 logger.info("loaded {}, sites {}", jvmPlugins.keySet(), sitePlugins);
152

```

加载plugin实例

我们的plugin一般都是jvmPlugins

各种校验功能

```

153 MapBuilder<Plugin, List<OnModuleReference>> onModuleReferences = MapBuilder.newMapBuilder();
154 for (Plugin plugin : jvmPlugins.values()) {
155     List<OnModuleReference> list = Lists.newArrayList();
156     for (Method method : plugin.getClass().getDeclaredMethods()) {
157         if (!method.getName().equals("onModule")) {
158             continue;
159         }
160         if (method.getParameterTypes().length == 0 || method.getParameterTypes().length > 1) {
161             logger.warn("Plugin: {} implementing onModule with no parameters or more than one parameter", plugin.name());
162             continue;
163         }
164         Class moduleClass = method.getParameterTypes()[0];
165         if (!Module.class.isAssignableFrom(moduleClass)) {
166             logger.warn("Plugin: {} implementing onModule by the type is not of Module type {}", plugin.name(), moduleClass);
167             continue;
168         }
169         method.setAccessible(true);
170         list.add(new OnModuleReference(moduleClass, method));
171     }
172     if (!list.isEmpty()) {
173         onModuleReferences.put(plugin, list);
174     }
175 }
176 this.onModuleReferences = onModuleReferences.immutableMap();
177
178 this.refreshInterval = componentSettings.getAsTime("info_refresh_interval", TimeValue.timeValueSeconds(10));
179
180

```

各种校验

形成最终数据

```
java\...\InternalNode.java x elasticsearch.yml x C:\elasticsearch-1.6.0-sources.jar\...\ModulesBuilder.java x java\...\ModulesBuilder.java x C
189 modules.add(new ClusterNameModule(settings));
190 modules.add(new ThreadPoolModule(threadPool));
191 modules.add(new DiscoveryModule(settings));
192 modules.add(new ClusterModule(settings));
193 modules.add(new RestModule(settings));
194 modules.add(new TransportModule(settings));
195 if (settings.getAsBoolean(HTTP_ENABLED, true)) {
196     modules.add(new HttpServerModule(settings));
197 }
198 modules.add(new RiversModule(settings));
199 modules.add(new IndicesModule(settings));
200 modules.add(new SearchModule());
201 modules.add(new ActionModule(false));
202 modules.add(new MonitorModule(settings));
203 modules.add(new GatewayModule(settings));
204 modules.add(new NodeClientModule());
205 modules.add(new BulkUdpModule());
206 modules.add(new ShapeModule());
207 modules.add(new PercolatorModule());
208 modules.add(new ResourceWatcherModule());
209 modules.add(new RepositoriesModule());
210 modules.add(new TribeModule());
211
212 injector = modules.createInjector();
213
214 client = injector.getInstance(Client.class);
215 threadPool.setNodeSettingsService(injector.getInstance(NodeSettingsService.class));
216 success = true;
217 } finally {
218     if (!success) {
219         nodeEnvironment.close();
220         ThreadPool.terminate(threadPool, 10, TimeUnit.SECONDS);
221     }
222 }
```

```
ModulesBuilder.java x TransportClient.java x java\...\Modules.java x java\...\PluginsModule.java x IndicesService.java x IndexService.java x IndicesClus
41 public ModulesBuilder add(Module module) {
42     modules.add(module);
43     if (module instanceof SpawnModules) {
44         Iterable<? extends Module> spawned = ((SpawnModules) module).spawnModules();
45         for (Module spawn : spawned) {
46             add(spawn);
47         }
48     }
49     return this;
50 }
51
52 @Override
53 public Iterator<Module> iterator() { return modules.iterator(); }
54
55
56 public Injector createInjector() {
57     Modules.processModules(modules);
58     Injector injector = Guice.createInjector(modules);
59     Injectors.cleanCaches(injector);
60     // in ES, we always create all instances as if they are eager singletons
61     // this allows for considerable memory savings (no need to store construction info) as well as cycles
62     ((InjectorImpl) injector).readOnlyAllSingletons();
63     return injector;
64 }
65
66 }
```

```

C ModulesBuilder.java × C TransportClient.java × C java\...\Modules.java × C java\...\PluginsModule.java × C IndicesSer
36
37 @
38 public static Module createModule(Class<? extends Module> moduleClass, @Nullable Setting
39     Constructor<? extends Module> constructor;
40     try {
41         constructor = moduleClass.getConstructor(Settings.class);
42         try {
43             return constructor.newInstance(settings);
44         } catch (Exception e) {
45             throw new ElasticsearchException("Failed to create module [" + moduleClass +
46         }
47     } catch (NoSuchMethodException e) {
48         try {
49             constructor = moduleClass.getConstructor();
50             try {
51                 return constructor.newInstance();
52             } catch (Exception e1) {
53                 throw new ElasticsearchException("Failed to create module [" + moduleCla
54             }
55         } catch (NoSuchMethodException e1) {
56             throw new ElasticsearchException("No constructor for [" + moduleClass + "]")
57         }
58     }
59
60 public static void processModules(Iterable<Module> modules) {
61     for (Module module : modules) {
62         if (module instanceof PreProcessModule) {
63             for (Module module1 : modules) {
64                 ((PreProcessModule) module).processModule(module1);
65             }
66         }
67     }
68 }
69 }
70

```

```
ModulesBuilder.java x TransportClient.java x java\...\Modules.java x java\...\PluginsModule.java x IndicesService.java x IndexService.java
34  /**
35  *
36  */
37  public class PluginsModule extends AbstractModule implements SpawnModules, PreProcessModule {
38
39      private final Settings settings;
40
41      private final PluginsService pluginsService;
42
43      public PluginsModule(Settings settings, PluginsService pluginsService) {
44          this.settings = settings;
45          this.pluginsService = pluginsService;
46      }
47
48      @Override
49      public Iterable<? extends Module> spawnModules() {
50          List<Module> modules = Lists.newArrayList();
51          Collection<Class<? extends Module>> modulesClasses = pluginsService.modules();
52          for (Class<? extends Module> moduleClass : modulesClasses) {
53              modules.add(createModule(moduleClass, settings));
54          }
55          modules.addAll(pluginsService.modules(settings));
56          return modules;
57      }
58
59      @Override
60      public void processModule(Module module) {
61          pluginsService.processModule(module);
62      }
63
64      @Override
65      protected void configure() { bind(PluginsService.class).toInstance(pluginsService); }
66
67      }
68
69  }
```

```
TransportClient.java x java\...\Modules.java x java\...\PluginsModule.java x Plugin.java x PluginsService.java x ESPluginFramework.java x elasticsearch
184
185  public void processModules(Iterable<Module> modules) {
186      for (Module module : modules) {
187          processModule(module);
188      }
189  }
190
191  public void processModule(Module module) {
192      for (Tuple<PluginInfo, Plugin> plugin : plugins()) {
193          plugin.v2().processModule(module);
194          // see if there are onModule references
195          List<OnModuleReference> references = onModuleReferences.get(plugin.v2());
196          if (references != null) {
197              for (OnModuleReference reference : references) {
198                  if (reference.moduleClass.isAssignableFrom(module.getClass())) {
199                      try {
200                          reference.onModuleMethod.invoke(plugin.v2(), module);
201                      } catch (Exception e) {
202                          logger.warn("plugin {}, failed to invoke custom onModule method", e, plugin.v2().name());
203                      }
204                  }
205              }
206          }
207      }
208  }
```

注册script脚本

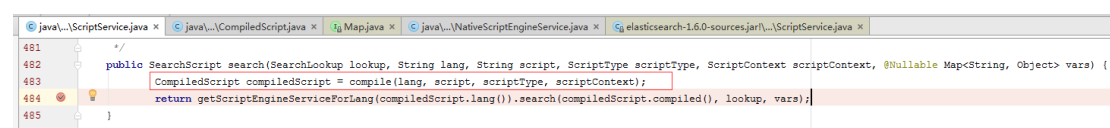
插件调用过程

at org.elasticsearch.script.NativeScriptEngineService.search(NativeScriptEngineService.java:80)


```

    at org.elasticsearch.script.ScriptService.search(ScriptService.java:484)
    at
org.elasticsearch.index.query.functionscore.script.ScriptScoreFunctionParser.parse(ScriptScoreFu
nctionParser.java:90)
    at
org.elasticsearch.index.query.functionscore.FunctionScoreQueryParser.parseFiltersAndFunctions(
FunctionScoreQueryParser.java:217)
    at
org.elasticsearch.index.query.functionscore.FunctionScoreQueryParser.parse(FunctionScoreQuer
yParser.java:122)
    at
org.elasticsearch.index.query.QueryParseContext.parseInnerQuery(QueryParseContext.java:302)
    at
org.elasticsearch.index.query.IndexQueryParserService.innerParse(IndexQueryParserService.java:
382)
    at
org.elasticsearch.index.query.IndexQueryParserService.parse(IndexQueryParserService.java:281)
    at
org.elasticsearch.index.query.IndexQueryParserService.parse(IndexQueryParserService.java:276)
    at org.elasticsearch.search.query.QueryParseElement.parse(QueryParseElement.java:33)
    at org.elasticsearch.search.SearchService.parseSource(SearchService.java:719)
    at org.elasticsearch.search.SearchService.createContext(SearchService.java:560)
    at org.elasticsearch.search.SearchService.createAndPutContext(SearchService.java:532)
    at org.elasticsearch.search.SearchService.executeQueryPhase(SearchService.java:294)
    at
org.elasticsearch.search.action.SearchServiceTransportAction$5.call(SearchServiceTransportActio
n.java:231)
    at
org.elasticsearch.search.action.SearchServiceTransportAction$5.call(SearchServiceTransportActio
n.java:228)
    at
org.elasticsearch.search.action.SearchServiceTransportAction$23.run(SearchServiceTransportActi
on.java:559)

```



```

481  */
482  public SearchScript search(SearchLookup lookup, String lang, String script, ScriptType scriptType, ScriptContext scriptContext, @Nullable Map<String, Object> vars) {
483      CompiledScript compiledScript = compile(lang, script, scriptType, scriptContext);
484      return getScriptEngineServiceForLang(compiledScript.lang()).search(compiledScript.compiled(), lookup, vars);
485  }
486

```

```

258 */
259 public CompiledScript compile(String lang, String script, ScriptType scriptType, ScriptContext scriptContext) {
260     assert scriptContext != null;
261
262     //scriptType might not get serialized depending on the version of the node we talk to, if null treat as inline
263     if (scriptType == null) {
264         scriptType = ScriptType.INLINE;
265     }
266     if (lang == null) {
267         lang = defaultLang;
268     }
269
270     ScriptEngineService scriptEngineService = getScriptEngineServiceForLang(lang);
271     //For backwards compat attempt to load from disk
272     if (scriptType == ScriptType.INLINE) {
273         CacheKey cacheKey = newCacheKey(scriptEngineService, script);
274         CompiledScript compiled = staticCache.get(cacheKey); //On disk scripts will be loaded into the staticCache by the listener
275         if (compiled != null) {
276             scriptType = ScriptType.FILE;
277             if (canExecuteScript(lang, scriptEngineService, scriptType, scriptContext) == false) {
278                 throw new ScriptException("scripts of type [" + scriptType + "], operation [" + scriptContext.getKey() + "] and lang [" + lang + "] are disabled");
279             }
280             return compiled;
281         }
282     }
283
284     if (canExecuteScript(lang, scriptEngineService, scriptType, scriptContext) == false) {
285         throw new ScriptException("scripts of type [" + scriptType + "], operation [" + scriptContext.getKey() + "] and lang [" + lang + "] are disabled");
286     }
287     return compileInternal(lang, script, scriptType);
288 }
289

```

```

292 */
293 public CompiledScript compileInternal(String lang, String scriptOrId, ScriptType scriptType) {
294     assert scriptType != null;
295     if (lang == null) {
296         lang = defaultLang;
297     }
298     if (logger.isDebugEnabled()) {
299         logger.trace("Compiling lang: {} type: {}", lang, scriptType, scriptOrId);
300     }
301
302     ScriptEngineService scriptEngineService = getScriptEngineServiceForLang(lang);
303     CacheKey cacheKey = newCacheKey(scriptEngineService, scriptOrId);
304
305     if (scriptType == ScriptType.FILE) {
306         CompiledScript compiled = staticCache.get(cacheKey); //On disk scripts will be loaded into the staticCache by the listener
307         if (compiled == null) {
308             throw new ElasticsearchIllegalArgumentException("Unable to find on disk script " + scriptOrId);
309         }
310         return compiled;
311     }
312
313     String script = scriptOrId;
314     if (scriptType == ScriptType.INDEXED) {
315         final IndexedScript indexedScript = new IndexedScript(lang, scriptOrId);
316         script = getScriptFromIndex(indexedScript.lang, indexedScript.id);
317         cacheKey = newCacheKey(scriptEngineService, script);
318     }
319
320     CompiledScript compiled = cache.getIfPresent(cacheKey);
321     if (compiled == null) {
322         //Either an un-cached inline script or an indexed script
323         compiled = new CompiledScript(lang, scriptEngineService.compile(script));
324         //Since the cache key is the script content itself we don't need to
325         //invalidate/check the cache if an indexed script changes.
326         cache.put(cacheKey, compiled);
327     }
328     return compiled;
329 }
330

```

```

44     this.scripts = ImmutableMap.copyOf(scripts);
45 }
46
47 @Override
48 public String[] types() { return new String[]{NAME}; }
49
50 @Override
51 public String[] extensions() { return new String[0]; }
52
53 @Override
54 public boolean sandboxed() { return false; }
55
56 @Override
57 public Object compile(String script) {
58     NativeScriptFactory scriptFactory = scripts.get(script);
59     if (scriptFactory != null) {
60         return scriptFactory;
61     }
62     throw new ElasticsearchIllegalArgumentException("Native script [" + script + "] not found");
63 }
64

```

```

    public SearchScript search(SearchLookup lookup, String lang, String script, ScriptType scriptType, ScriptContext scriptContext, @Nullable Map<String, Object> vars) {
        CompiledScript compiledScript = compile(lang, script, scriptType, scriptContext);
        return getScriptEngineServiceForLang(compiledScript.lang()).search(compiledScript.compiled(), lookup, vars);
    }

```

```

71  @Override
72  public ExecutableScript executable(Object compiledScript, @Nullable Map<String, Object> vars) {
73      NativeScriptFactory scriptFactory = (NativeScriptFactory) compiledScript;
74      return scriptFactory.newScript(vars);
75  }
76
77  @Override
78  public SearchScript search(Object compiledScript, SearchLookup lookup, @Nullable Map<String, Object> vars) {
79      NativeScriptFactory scriptFactory = (NativeScriptFactory) compiledScript;
80      AbstractSearchScript script = (AbstractSearchScript) scriptFactory.newScript(vars);
81      script.setLookup(lookup);
82      return script;
83  }
84

```

```

77  }
78
79  ScriptParameterValue scriptValue = scriptParameterParser.getDefaultScriptParameterValue(); scriptValue: ScriptParameterValue$ScriptParameterValue$5399
80  if (scriptValue != null) {
81      script = scriptValue.script();
82      scriptType = scriptValue.scriptType(); scriptValue: ScriptParameterValue$ScriptParameterValue$5399
83  }
84  if (script == null) {
85      throw new QueryParsingException(parseContext.index(), NAMES[0] + " requires 'script' field");
86  }
87
88  SearchScript searchScript;
89  try {
90      searchScript = parseContext.scriptService().search(parseContext.lookup(), scriptParameterParser.lang(), script, scriptType, ScriptContext.Standard.SEARCH, vars);
91      return new ScriptScoreFunction(script, vars, searchScript);
92  } catch (Exception e) {
93      throw new QueryParsingException(parseContext.index(), NAMES[0] + " the script could not be loaded", e);
94  }
95  }
96

```

```

561  parseSource(context, request.extrasource()); request: shardsearchtransportrequest@53548
562
563  // if the from and size are still not set, default them
564  if (context.from() == -1) {
565      context.from(0);
566  }
567  if (context.size() == -1) {
568      context.size(10);
569  }
570
571  // pre process
572  dfsPhase.preProcess(context); context: DefaultSearchContext@55520
573  queryPhase.preProcess(context);
574  fetchPhase.preProcess(context);
575
576  // compute the context keep alive
577  long keepAlive = defaultKeepAlive;
578  if (request.scroll() != null && request.scroll().keepAlive() != null) {
579      keepAlive = request.scroll().keepAlive().millis();
580  }
581  context.keepAlive(keepAlive);
582  } catch (Throwable e) {
583      context.close();
584      throw ExceptionsHelper.convertToRuntime(e);
585  }
586

```

```

elasticsearch-1.6.0-sources.jar\...\SearchService.java x
291     }
292
293     public QuerySearchResultProvider executeQueryPhase(ShardSearchRequest request) throws ElasticsearchException { request: ShardSearchTransportRequest$
294         final SearchContext context = createAndPutContext(request); context: DefaultSearchContext@5520 request: ShardSearchTransportRequest$5548
295         try {
296             context.indexShard().searchService().onPreQueryPhase(context); context: DefaultSearchContext$5520
297             long time = System.nanoTime();
298             contextProcessing(context);
299
300             loadOrExecuteQueryPhase(request, context, queryPhase);
301
302             if (context.searchType() == SearchType.COUNT) {
303                 freeContext(context.id());
304             } else {
305                 contextProcessedSuccessfully(context);
306             }
307             context.indexShard().searchService().onQueryPhase(context, System.nanoTime() - time);
308
309             return context.queryResult();
310         } catch (Throwable e) {
311             // execution exception can happen while loading the cache, strip it
312             if (e instanceof ExecutionException) {
313                 e = e.getCause();
314             }
315             context.indexShard().searchService().onFailedQueryPhase(context);
316             logger.trace("Query phase failed", e);

```