   在启动和运行期都可以加载agent代理，在启动的时候可通过-javaagent参数来执行agent代理，而在运行期就是通过attach这种机制动态load了。

      如果在vm启动过程中加载agent，那么会在vm初始化过程中先执行libinstrument.dylib里InvocationAdapter.c的Agent\_OnLoad方法，这里主要是实例化agent，解析agent的MF文件，将相关属性取出来，并注册jvmti的一些回调函数，在vm初始化完成之后，会通过回调函数去实例化Instrumentation实现对象，设置ClassFileLoadHook函数，并调用Pre-Main指定类的premain方法。

**C代码  [收藏代码](javascript:void())**

1. JNIEXPORT jint JNICALL
2. Agent\_OnLoad(JavaVM \*vm, **char** \*tail, **void** \* reserved) {
3. JPLISInitializationError initerror  = JPLIS\_INIT\_ERROR\_NONE;
4. jint                     result     = JNI\_OK;
5. JPLISAgent \*             agent      = NULL;
7. initerror = createNewJPLISAgent(vm, &agent);
8. **if** ( initerror == JPLIS\_INIT\_ERROR\_NONE ) {
9. **int**             oldLen, newLen;
10. **char** \*          jarfile;
11. **char** \*          options;
12. jarAttribute\*   attributes;
13. **char** \*          premainClass;
14. **char** \*          agentClass;
15. **char** \*          bootClassPath;
17. /\*
18. \* Parse <jarfile>[=options] into jarfile and options
19. \*/
20. **if** (parseArgumentTail(tail, &jarfile, &options) != 0) {
21. fprintf(stderr, "-javaagent: memory allocation failure.\n");
22. **return** JNI\_ERR;
23. }
25. /\*
26. \* Agent\_OnLoad is specified to provide the agent options
27. \* argument tail in modified UTF8. However for 1.5.0 this is
28. \* actually in the platform encoding - see 5049313.
29. \*
30. \* Open zip/jar file and parse archive. If can't be opened or
31. \* not a zip file return error. Also if Premain-Class attribute
32. \* isn't present we return an error.
33. \*/
34. attributes = readAttributes(jarfile);
35. **if** (attributes == NULL) {
36. fprintf(stderr, "Error opening zip file or JAR manifest missing : %s\n", jarfile);
37. free(jarfile);
38. **if** (options != NULL) free(options);
39. **return** JNI\_ERR;
40. }
42. premainClass = getAttribute(attributes, "Premain-Class");
43. **if** (premainClass == NULL) {
44. fprintf(stderr, "Failed to find Premain-Class manifest attribute in %s\n",
45. jarfile);
46. free(jarfile);
47. **if** (options != NULL) free(options);
48. freeAttributes(attributes);
49. **return** JNI\_ERR;
50. }
52. /\*
53. \* Add to the jarfile
54. \*/
55. appendClassPath(agent, jarfile);
57. /\*
58. \* The value of the Premain-Class attribute becomes the agent
59. \* class name. The manifest is in UTF8 so need to convert to
60. \* modified UTF8 (see JNI spec).
61. \*/
62. oldLen = (**int**)strlen(premainClass);
63. newLen = modifiedUtf8LengthOfUtf8(premainClass, oldLen);
64. **if** (newLen == oldLen) {
65. premainClass = strdup(premainClass);
66. } **else** {
67. **char**\* str = (**char**\*)malloc( newLen+1 );
68. **if** (str != NULL) {
69. convertUtf8ToModifiedUtf8(premainClass, oldLen, str, newLen);
70. }
71. premainClass = str;
72. }
73. **if** (premainClass == NULL) {
74. fprintf(stderr, "-javaagent: memory allocation failed\n");
75. free(jarfile);
76. **if** (options != NULL) free(options);
77. freeAttributes(attributes);
78. **return** JNI\_ERR;
79. }
81. /\*
82. \* If the Boot-Class-Path attribute is specified then we process
83. \* each relative URL and add it to the bootclasspath.
84. \*/
85. bootClassPath = getAttribute(attributes, "Boot-Class-Path");
86. **if** (bootClassPath != NULL) {
87. appendBootClassPath(agent, jarfile, bootClassPath);
88. }
90. /\*
91. \* Convert JAR attributes into agent capabilities
92. \*/
93. convertCapabilityAtrributes(attributes, agent);
95. /\*
96. \* Track (record) the agent class name and options data
97. \*/
98. initerror = recordCommandLineData(agent, premainClass, options);
100. /\*
101. \* Clean-up
102. \*/
103. free(jarfile);
104. **if** (options != NULL) free(options);
105. freeAttributes(attributes);
106. free(premainClass);
107. }
109. **switch** (initerror) {
110. **case** JPLIS\_INIT\_ERROR\_NONE:
111. result = JNI\_OK;
112. **break**;
113. **case** JPLIS\_INIT\_ERROR\_CANNOT\_CREATE\_NATIVE\_AGENT:
114. result = JNI\_ERR;
115. fprintf(stderr, "java.lang.instrument/-javaagent: cannot create native agent.\n");
116. **break**;
117. **case** JPLIS\_INIT\_ERROR\_FAILURE:
118. result = JNI\_ERR;
119. fprintf(stderr, "java.lang.instrument/-javaagent: initialization of native agent failed.\n");
120. **break**;
121. **case** JPLIS\_INIT\_ERROR\_ALLOCATION\_FAILURE:
122. result = JNI\_ERR;
123. fprintf(stderr, "java.lang.instrument/-javaagent: allocation failure.\n");
124. **break**;
125. **case** JPLIS\_INIT\_ERROR\_AGENT\_CLASS\_NOT\_SPECIFIED:
126. result = JNI\_ERR;
127. fprintf(stderr, "-javaagent: agent class not specified.\n");
128. **break**;
129. **default**:
130. result = JNI\_ERR;
131. fprintf(stderr, "java.lang.instrument/-javaagent: unknown error\n");
132. **break**;
133. }
134. **return** result;
135. }

      如果在运行期通过attach api来load agent，那么会在收到load指令之后，会调用InvocationAdapter.c的Agent\_OnAttach方法，其实现基本和Agent\_OnLoad一致，只是还会调用Agent-Class的agentmain方法，还有点不同就是对vmint事件没有再关注（都运行期了，关注也没用），而是直接对ClassFileLoad关注，也不会再调用Pre-Main指定的类的premain方法（顾名思义，是在执行main方法之前执行的，所以运行期搞执行Pre-Main的class也不妥）。

**C代码  [收藏代码](javascript:void())**

1. JNIEXPORT jint JNICALL
2. Agent\_OnAttach(JavaVM\* vm, **char** \*args, **void** \* reserved) {
3. JPLISInitializationError initerror  = JPLIS\_INIT\_ERROR\_NONE;
4. jint                     result     = JNI\_OK;
5. JPLISAgent \*             agent      = NULL;
6. JNIEnv \*                 jni\_env    = NULL;
8. /\*
9. \* Need JNIEnv - guaranteed to be called from thread that is already
10. \* attached to VM
11. \*/
12. result = (\*vm)->GetEnv(vm, (**void**\*\*)&jni\_env, JNI\_VERSION\_1\_2);
13. jplis\_assert(result==JNI\_OK);
15. initerror = createNewJPLISAgent(vm, &agent);
16. **if** ( initerror == JPLIS\_INIT\_ERROR\_NONE ) {
17. **int**             oldLen, newLen;
18. **char** \*          jarfile;
19. **char** \*          options;
20. jarAttribute\*   attributes;
21. **char** \*          agentClass;
22. **char** \*          bootClassPath;
23. jboolean        success;
25. /\*
26. \* Parse <jarfile>[=options] into jarfile and options
27. \*/
28. **if** (parseArgumentTail(args, &jarfile, &options) != 0) {
29. **return** JNI\_ENOMEM;
30. }
32. /\*
33. \* Open the JAR file and parse the manifest
34. \*/
35. attributes = readAttributes( jarfile );
36. **if** (attributes == NULL) {
37. fprintf(stderr, "Error opening zip file or JAR manifest missing: %s\n", jarfile);
38. free(jarfile);
39. **if** (options != NULL) free(options);
40. **return** AGENT\_ERROR\_BADJAR;
41. }
43. agentClass = getAttribute(attributes, "Agent-Class");
44. **if** (agentClass == NULL) {
45. fprintf(stderr, "Failed to find Agent-Class manifest attribute from %s\n",
46. jarfile);
47. free(jarfile);
48. **if** (options != NULL) free(options);
49. freeAttributes(attributes);
50. **return** AGENT\_ERROR\_BADJAR;
51. }
53. /\*
54. \* Add the jarfile to the system class path
55. \*/
56. **if** (appendClassPath(agent, jarfile)) {
57. fprintf(stderr, "Unable to add %s to system class path "
58. "- not supported by system class loader or configuration error!\n",
59. jarfile);
60. free(jarfile);
61. **if** (options != NULL) free(options);
62. freeAttributes(attributes);
63. **return** AGENT\_ERROR\_NOTONCP;
64. }
66. /\*
67. \* The value of the Agent-Class attribute becomes the agent
68. \* class name. The manifest is in UTF8 so need to convert to
69. \* modified UTF8 (see JNI spec).
70. \*/
71. oldLen = strlen(agentClass);
72. newLen = modifiedUtf8LengthOfUtf8(agentClass, oldLen);
73. **if** (newLen == oldLen) {
74. agentClass = strdup(agentClass);
75. } **else** {
76. **char**\* str = (**char**\*)malloc( newLen+1 );
77. **if** (str != NULL) {
78. convertUtf8ToModifiedUtf8(agentClass, oldLen, str, newLen);
79. }
80. agentClass = str;
81. }
82. **if** (agentClass == NULL) {
83. free(jarfile);
84. **if** (options != NULL) free(options);
85. freeAttributes(attributes);
86. **return** JNI\_ENOMEM;
87. }
89. /\*
90. \* If the Boot-Class-Path attribute is specified then we process
91. \* each URL - in the live phase only JAR files will be added.
92. \*/
93. bootClassPath = getAttribute(attributes, "Boot-Class-Path");
94. **if** (bootClassPath != NULL) {
95. appendBootClassPath(agent, jarfile, bootClassPath);
96. }
98. /\*
99. \* Convert JAR attributes into agent capabilities
100. \*/
101. convertCapabilityAtrributes(attributes, agent);
103. /\*
104. \* Create the java.lang.instrument.Instrumentation instance
105. \*/
106. success = createInstrumentationImpl(jni\_env, agent);
107. jplis\_assert(success);
109. /\*
110. \*  Turn on the ClassFileLoadHook.
111. \*/
112. **if** (success) {
113. success = setLivePhaseEventHandlers(agent);
114. jplis\_assert(success);
115. }
117. /\*
118. \* Start the agent
119. \*/
120. **if** (success) {
121. success = startJavaAgent(agent,
122. jni\_env,
123. agentClass,
124. options,
125. agent->mAgentmainCaller);
126. }
128. **if** (!success) {
129. fprintf(stderr, "Agent failed to start!\n");
130. result = AGENT\_ERROR\_STARTFAIL;
131. }
133. /\*
134. \* Clean-up
135. \*/
136. free(jarfile);
137. **if** (options != NULL) free(options);
138. free(agentClass);
139. freeAttributes(attributes);
140. }
142. **return** result;
143. }