通过此介绍：<http://www.oschina.net/p/cat-dianping>

得知，运行cat-home项目里的‘com.dianping.cat.TestServer’可以启动CAT服务。so，我们就通过这个类来分析一下cat服务端的启动流程。

准备工作：需要把cat的代码导入IDE（我这里用的是eclipse），下载依赖（吐槽一下，太慢了），编译成功；

好，接着我们断点TestServer类跟踪整个启动流程。

----------------------------------------------分割线------------------------------------------------------

直接上TsetServer代码：

1. /\*\*
2. \* Junit4最大的改进是大量使用注解（元数据），很多实际执行过程都在Junit的后台做完了，
3. \* 而且写test case 的类不需要继承TestCase，只需要在所要做test case的方法前加@Test 注解即可。
4. \* @author admin
5. \*/
6. @RunWith(JUnit4.class)//测试运行于JUnit4测试环境
7. public class TestServer extends JettyServer {
8. public static void main(String[] args) throws Exception {
9. TestServer server = new TestServer();
10. System.setProperty("devMode", "true");//设置指定键对值的系统属性:开发模式（true）
11. server.startServer();//开启一个jetty服务
12. server.startWebApp();//在默认浏览器中打开一个页面
13. server.stopServer();//停止一个jetty服务
14. }
16. /\*\*
17. \* @Before, @After注解过的方法将在测试方法之前/之后执行。
18. \* @throws Exception
19. \*/
20. @Before
21. public void before() throws Exception {
22. System.setProperty("devMode", "true");
23. super.startServer();
24. }
25. @Override
26. protected String getContextPath() {
27. return "/cat";
28. }
29. /\*\*
30. \* 得到服务器端口
31. \*/
32. @Override
33. protected int getServerPort() {
34. return 2281;
35. }
36. @Override
37. protected void postConfigure(WebAppContext context) {
38. context.addFilter(GzipFilter.class, "/\*", Handler.ALL);
39. }
40. @Test
41. public void startWebApp() throws Exception {
42. // open the page in the default browser
43. // 在系统默认浏览器中打开一个页面，这个地方要注意一下，因为调用的是默认浏览器，而cat的对于别的浏览器页面不兼容，建议把默认浏览器设置为chrome
44. display("/cat/r");
45. waitForAnyKey();
46. }
47. }

从上面的代码可以看出重点在main方法，从中基本可以看出整个服务器的启动过程如下：

1，因为cat项目在开发模式下集成了jetty，所以第一步就是启动一个jetty服务（server.startServer();//开启一个jetty服务）；

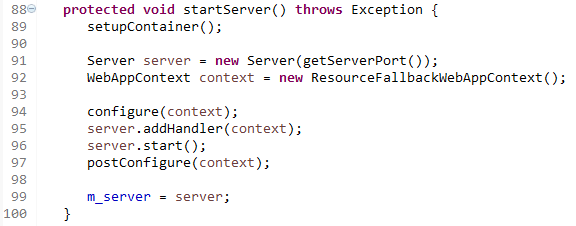
2，调用本地操作系统的默认浏览器打开一个页面（server.startWebApp();//在默认浏览器中打开一个页面）；

3，停止服务（server.stopServer();//停止一个jetty服务）；

所以，cat服务的启动过程可以通过以上三个过程步骤进行分析，接下来我们看看cat在startServer的时候都干了一些什么，而startServer整个过程又分为如下步骤：

JettyServer类

此类的代码点评封装在了一个jar里面（test-framework-2.2.0.jar）



1，安装Plexus IOC容器setupContainer()（没有接触过plexus ioc的请查看：<http://blog.csdn.net/songhuiqiao/article/details/49908165>）

     Plexus提供完整的软件栈，用于创建和执行软件项目。基于Plexus容器，应用程序可以利用面向组件的编程方式来构建模块化，容易集成和重复使用的可复用组件。

     虽然Plexus是一个类似控制反转（IoC）或依赖注入（DI）框架的框架 ，事实上它更是一个支持如下许多功能的全面的容器：

     ● 组件生命周期（Component lifecycles）

     ● 组件实例化战略（Component instantiation strategies）

     ● 嵌套容器（Nested containers ）

     ● 组件配置（Component configuration）

     ● 自动布线（Auto-wiring）

     ● 组件依赖关系以及各种依赖注入技术，包括构造函数注入，setter注入和private注入。（Component dependencies,

        and Various dependency injection techniques including constructor injection, setter injection, and private field injection.）

setupContainer()

protected void setupContainer() throws Exception {

      PlexusContainer container = ContainerLoader.getDefaultContainer();

      DefaultContext context = new DefaultContext();

      context.put("plexus", container);

      contextualize(context);

   }

//得到默认的Plexus容器：首先创建一个默认的容器配置对象，设置容器的配置文件为：/META-INF/plexus/plexus.xml，然后根据配置对象得到默认容器返回；

public static PlexusContainer getDefaultContainer() {

      DefaultContainerConfiguration configuration = new DefaultContainerConfiguration();

      configuration.setContainerConfiguration("/META-INF/plexus/plexus.xml"); ///D:/workspace/cat/trunk/cat-client/target/classes/META-INF/plexus/plexus.xml

      return getDefaultContainer(configuration); //得到默认ioc容器

   }

public static PlexusContainer getDefaultContainer(ContainerConfiguration configuration) {

      if (s\_container == null) {//判断容器是否为空

        。。。。。。

        preConstruction(configuration);//前置构造

        s\_container = new DefaultPlexusContainer(configuration);//创建默认Plexus容器

        postConstruction(s\_container);//后置构造

        。。。。。。

}

@SuppressWarnings("unchecked")

   private static void preConstruction(ContainerConfiguration configuration) throws Exception {

      LifecycleHandler plexus = configuration.getLifecycleHandlerManager().getLifecycleHandler("plexus"); //得到Plexus生命周期handler

      Field field = Reflects.forField().getDeclaredField(AbstractLifecycleHandler.class, "beginSegment");

      field.setAccessible(true);

      List<Phase> segment = (List<Phase>) field.get(plexus);

      segment.add(0, new org.unidal.lookup.extension.PostConstructionPhase());

      try {

         new ContainerConfigurationDecorator().process(configuration);

      } catch (Exception e) {

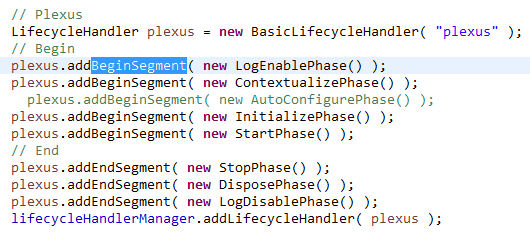
         e.printStackTrace();

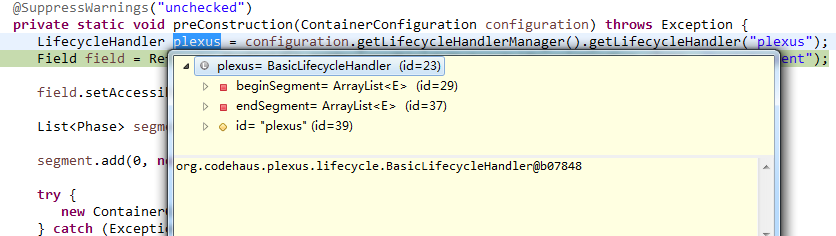
      }

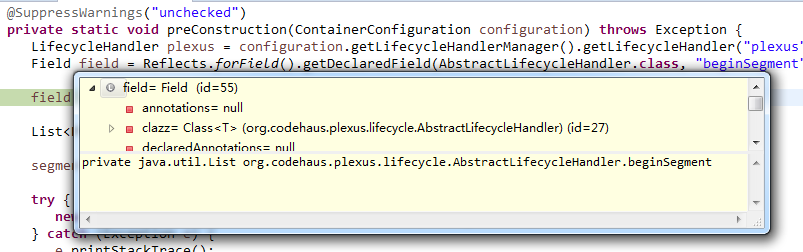
   }

在前置构造中调用getLifecycleHandlerManager()创建Plexus，Basic，Plexus configurable，passive，Bootstrap等生命周期handler；已Plexus为例，对应的BeginSegment和EndSegment分别如下：

DefaultContainerConfiguration.getLifecycleHandlerManager()







List<Phase> segment = (List<Phase>) field.get(plexus);下面是List中的BeginSegment的4个阶段

[[org.codehaus.plexus.personality.plexus.lifecycle.phase.LogEnablePhase@6f866002,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.LogEnablePhase@6f866002,)

[org.codehaus.plexus.personality.plexus.lifecycle.phase.ContextualizePhase@5f095c81,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.ContextualizePhase@5f095c81,)

[org.codehaus.plexus.personality.plexus.lifecycle.phase.InitializePhase@5f9849e5,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.InitializePhase@5f9849e5,)

[org.codehaus.plexus.personality.plexus.lifecycle.phase.StartPhase@71b8a6b](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.StartPhase@71b8a6b)]

然后接着把PostConstructionPhase阶段增加到List中的第0位；

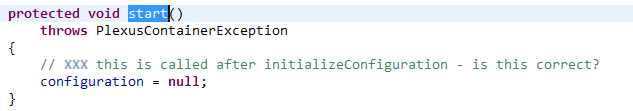
segment list变成如下：

[[org.unidal.lookup.extension.PostConstructionPhase@3ee3f8b9,](mailto:org.unidal.lookup.extension.PostConstructionPhase@3ee3f8b9,) [org.codehaus.plexus.personality.plexus.lifecycle.phase.LogEnablePhase@6f866002,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.LogEnablePhase@6f866002,) [org.codehaus.plexus.personality.plexus.lifecycle.phase.ContextualizePhase@5f095c81,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.ContextualizePhase@5f095c81,) [org.codehaus.plexus.personality.plexus.lifecycle.phase.InitializePhase@5f9849e5,](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.InitializePhase@5f9849e5,) [org.codehaus.plexus.personality.plexus.lifecycle.phase.StartPhase@71b8a6b](mailto:org.codehaus.plexus.personality.plexus.lifecycle.phase.StartPhase@71b8a6b)]

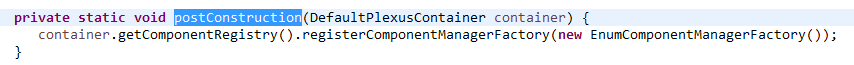
接着通过容器配置Decorator（new ContainerConfigurationDecorator()）进行处理，在process方法中获取容器配置/META-INF/plexus/plexus.xml，然后在生成一个临时文件：C:\Users\admin\AppData\Local\Temp\plexus-9216030128668808777.xml，最后把这个临时文件 configuration.setContainerConfigurationURL(tmp.toURI().toURL());

接着s\_container = new DefaultPlexusContainer(configuration);

进行改造，然后初始化（Plexus生命周期管理，其中发现组件discoverComponents方法执行比较慢），开始：

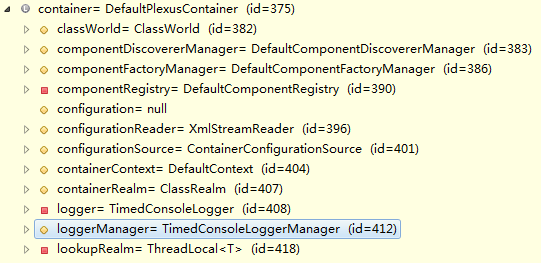


紧接着进行后置改造：



主要是注册一下组件的管理工厂类；

最终得到的PlexusContainer对象container的结构如下：



到此，安装Plexus IOC容器已经完毕。

2，根据服务器端口创建一个jetty服务，打开socket连接，创建服务对象；

3，创建一个WebAppContext对象，具体参考：<http://blog.csdn.net/kobejayandy/article/details/20165937>

4，安装应用；

@SuppressWarnings("unchecked")

   protected void configure(WebAppContext context) {

      File warRoot = getWarRoot(); //获取war包路径：src\main\webapp

      context.getInitParams().put("org.mortbay.jetty.servlet.Default.dirAllowed", "false");

      context.setContextPath(getContextPath()); ///cat

      context.setDescriptor(new File(warRoot, "WEB-INF/web.xml").getPath()); //设置描述符位置 src\main\webapp\WEB-INF\web.xml

      context.setResourceBase(warRoot.getPath()); //

   }

5，添加到处理器server.start();

6，启动jetty服务；

     当jetty容器其中的时候，会去读取src\main\webapp\WEB-INF\web.xml这个文件，下面接着分析一下web.xml这个文件

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="<http://java.sun.com/xml/ns/javaee>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"  
xsi:schemaLocation="<http://java.sun.com/xml/ns/javaee> <http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd>"  
version="2.5">  
<filter>  
<filter-name>cat-filter</filter-name>  
<filter-class>com.dianping.cat.servlet.CatFilter</filter-class>  
</filter>  
<filter>  
<filter-name>domain-filter</filter-name>  
<filter-class>com.dianping.cat.report.view.DomainFilter</filter-class>  
</filter>  
<servlet>  
<servlet-name>cat-servlet</servlet-name>  
<servlet-class>com.dianping.cat.servlet.CatServlet</servlet-class>  
<load-on-startup>1</load-on-startup>  
</servlet>  
<servlet>  
<servlet-name>mvc-servlet</servlet-name>  
<servlet-class>org.unidal.web.MVC</servlet-class>  
<init-param>  
<param-name>cat-client-xml</param-name>  
<param-value>client.xml</param-value>  
</init-param>  
<init-param>  
<param-name>init-modules</param-name>  
<param-value>false</param-value>  
</init-param>  
<load-on-startup>2</load-on-startup>  
</servlet>  
<filter-mapping>  
<filter-name>cat-filter</filter-name>  
<url-pattern>/r/\*</url-pattern>  
<dispatcher>REQUEST</dispatcher>  
</filter-mapping>  
<filter-mapping>  
<filter-name>domain-filter</filter-name>  
<url-pattern>/r/\*</url-pattern>  
<dispatcher>REQUEST</dispatcher>  
</filter-mapping>  
<filter-mapping>  
<filter-name>cat-filter</filter-name>  
<url-pattern>/s/\*</url-pattern>  
<dispatcher>REQUEST</dispatcher>  
</filter-mapping>  
<filter-mapping>  
<filter-name>cat-filter</filter-name>  
<url-pattern>/jsp/\*</url-pattern>  
<dispatcher>FORWARD</dispatcher>  
</filter-mapping>  
<servlet-mapping>  
<servlet-name>mvc-servlet</servlet-name>  
<url-pattern>/r/\*</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
<servlet-name>mvc-servlet</servlet-name>  
<url-pattern>/s/\*</url-pattern>  
</servlet-mapping>  
<jsp-config>  
<taglib>  
<taglib-uri>/WEB-INF/app.tld</taglib-uri>  
<taglib-location>/WEB-INF/app.tld</taglib-location>  
</taglib>  
</jsp-config>  
</web-app>

在web.xml文件中会先执行cat-servlet中的initComponents方法，然后再执行mvc-servlet的初始化方法，还会加载cat-filter，domain-filter等过滤器。

先看cat-servlet的initComponents方法：

/\*\*

  \* 初始化组件

  \*/

 @Override

 protected void initComponents(ServletConfig servletConfig) throws ServletException {

  try {

   ModuleContext ctx = new DefaultModuleContext(getContainer());

   ModuleInitializer initializer = ctx.lookup(ModuleInitializer.class);

   File clientXmlFile = getConfigFile(servletConfig, "cat-client-xml", "client.xml");

   File serverXmlFile = getConfigFile(servletConfig, "cat-server-xml", "server.xml");

   ctx.setAttribute("cat-client-config-file", clientXmlFile);//设置cat客户端配置文件：\data\appdatas\cat\client.xml

   ctx.setAttribute("cat-server-config-file", serverXmlFile);//设置cat服务端配置文件：\data\appdatas\cat\server.xml

   initializer.execute(ctx); //模块初始化容器执行

  } catch (Exception e) {

   m\_exception = e;

   System.err.println(e);

   throw new ServletException(e);

  }

 }

@Override

   public void execute(ModuleContext ctx) {

      Module[] modules = m\_manager.getTopLevelModules(); //得到顶级模块CatHomeModule

      execute(ctx, modules); //安装模块

   }

@Override

   public void execute(ModuleContext ctx, Module... modules) {

      Set<Module> all = new LinkedHashSet<Module>();

      info(ctx, "Initializing top level modules:");

      for (Module module : modules) {

         info(ctx, "   " + module.getClass().getName());

      }

      try {

         expandAll(ctx, modules, all); //全部展开所有模块

         for (Module module : all) {

            if (!module.isInitialized()) {

               executeModule(ctx, module, m\_index++);

            }

         }

      } catch (Exception e) {

         throw new RuntimeException("Error when initializing modules! Exception: " + e, e);

      }

   }

   private synchronized void executeModule(ModuleContext ctx, Module module, int index) throws Exception {

      long start = System.currentTimeMillis();

      // set flat to avoid re-entrance

      module.setInitialized(true);

      info(ctx, index + " ------ " + module.getClass().getName());

      // execute itself after its dependencies

      module.initialize(ctx);

      long end = System.currentTimeMillis();

      info(ctx, index + " ------ " + module.getClass().getName() + " DONE in " + (end - start) + " ms.");

   }

7，后置安装；

未完，待续。