ClassLoader in Java

http://duqing72.spaces.msn.com/blog/cns!a9248baf3813f999!144.entry

Java 的 ClassLoader

Java 中的 ClassLoader 采用 Delegation 机制。即每一个 ClassLoader 都有自己的 Parent Class Loader，当从一个 Class Loader 中加载一个 Class 时，会先到当前 ClassLoader 的 Parent 中寻找该 Class 是否已经被加载，如果是，则从其 Parent Class Loader 中得到 Class 的 Instance，如果没有，使用当前 ClassLoader 来加载 Class。请看如下程序：

```
public class Test {

  public static void main(String[] args) throws Exception {
    Test test = new Test();
    ClassLoader cl = test.getClass().getClassLoader();
    while (cl != null) {
        System.out.println("class loader is: " + cl.toString());
        cl = cl.getParent();
    }
  }

}
```

在 Windows 上使用 JDK1.5 输出如下：

class loader is: sun.misc.Launcher$AppClassLoader@82ba41

class loader is: sun.misc.Launcher$ExtClassLoader@923e30

对于每一个 Java 进程都有一个 System Class Loader，也叫 Application Class Loader，用于加载写在 CLASSPATH 中的 Class，并作为其它用户 Class Loader 的缺省的 Parent Class Loader。由以上输出可以看到，Test 类由 System Class Loader 加载，System Class Loader 的 Parent 是 Ext. Class Loader。Ext. Class Loader 还具有一个 Parent Class Loader，就是 BootstrapClassLoader。该 Class Loader 是 Java VM 内置的 Class Loader，用于加载 java.*。如果使用 ClassLoader.getParent()方法，当一个 Class Loader 的 Parent 是 Bootstrap 是，一般返回 null。(Windows, Linux, Solaris 均如此)

观察如下程序：

```
public class B {
  public void method1() {
    A a = new A();
    ...
  }
}


public class B {
  public void method1() {
    Class clazz = Class.forName("A");
    Object o = clazz.newInstance();
    ...
  }
}
```

这两段程序都会从“Current Class Loader”来加载 Class A。即从加载 B Class 的 Class Loader 来加载 Class A。

Thread 的 Context Class Loader

自从 JDK1.2 以后，Sun 为 java.lang.thread 加入了 setContextClassLoader 和 getContextClassLoader 两个方法，但是并没有指明应该如何使用。通常比较 Confuse 的是，当执行上述代码是，是从 Thread 的 Context Class Loader 中加载 Class A，还是从加载 Class B 的 Class Loader 中加载 A（如果两个 ClassLoader 不同的话）？答案是从加载 B 的 Class Loader 中加载 A，而不是从 Thread 的 Context Class Loader 中加载 A。那 Context Class Loader 是做什么用的呢？举例来说：在 JDK1.4 中，加入了 XML 的支持（JAXP），用户可以指定 DocumentBuilder 的 Concrete Class，该 Class 需实现 JAXP 接口。在 DocumentBuilderFactory 中是使用 Class.forName 加载该 Concrete Class 的。DocumentBuilderFactory 是使用 Bootstrap Class Loader 来加载的，但通常 JAXP 的实现都是位于 CLASSPATH 中，是由 System Class Loader 加载，而 Bootstrap Class Loader 又是 System Class Loader 的 Ancestor Class Loader，即 Bootstrap Class Loader 不能从 System Class Loader 中加载 Class。这时，就可以显式的使用 Thread.getContextClassLoader 来解决这一问题。而在 JDK1.4 中也正式这样实现的。

这里，我们再看一段程序：

```
    public class Test {
      public static void main(String[] args) throws Exception {
        Test test = new Test();
        ClassLoader cl = test.getClass().getClassLoader();
        while (cl != null) {
            System.out.println("class loader is: " + cl.toString());
            cl = cl.getParent();
        }

        System.out.println("thread context class loader is: " +
Thread.currentThread().getContextClassLoader());
        System.out.println("system class loader is: " +
ClassLoader.getSystemClassLoader());

        Class clazz = Class.forName("java.lang.String");
        System.out.println("bootstrap class loader is: " +
clazz.getClassLoader());
      }
    }
```

在 Windows 上使用 JDK1.5 输出如下：

```
    class loader is: sun.misc.Launcher$AppClassLoader@82ba41
    class loader is: sun.misc.Launcher$ExtClassLoader@923e30
    thread context class loader is: sun.misc.Launcher$AppClassLoader@82ba41
    system class loader is: sun.misc.Launcher$AppClassLoader@82ba41
    bootstrap class loader is: null
```

由此可见，在 Java 进程启动时，main thread 的 Context Class Loader 即为 System Class Loader。