xxbcoder

classloader加载的双亲委托模式

要深入了解ClassLoader，首先就要知道ClassLoader是用来干什么的，顾名思义，它就是用来加载Class文件到JVM，以供程序使用的。我们知道，java程序可以动态加载类定义，而这个动态加载的机制就是通过ClassLoader来实现的，所以可想而知ClassLoader的重要性如何。

看到这里，可能有的朋友会想到一个问题，那就是既然ClassLoader是用来加载类到JVM中的，那么ClassLoader又是如何被加载呢？难道它不是java的类?

没有错，在这里确实有一个ClassLoader不是用java语言所编写的，而是JVM实现的一部分，这个ClassLoader就是 bootstrap classloader（启动类加载器），这个ClassLoader在JVM运行的时候加载java核心的API以满足java程序最基本的需求，其中 就包括用户定义的ClassLoader，这里所谓的用户定义是指通过java程序实现的ClassLoader，一个是ExtClassLoader， 这个ClassLoader是用来加载java的扩展API的，也就是/lib/ext中的类，一个是AppClassLoader，这个 ClassLoader是用来加载用户机器上CLASSPATH设置目录中的Class的，通常在没有指定ClassLoader的情况下，程序员自定义 的类就由该ClassLoader进行加载。

当运行一个程序的时候，JVM启动，运行bootstrap classloader，该ClassLoader加载java核心API（ExtClassLoader和AppClassLoader也在此时被加 载），然后调用ExtClassLoader加载扩展API，最后AppClassLoader加载CLASSPATH目录下定义的Class，这就是一 个程序最基本的加载流程。

上面大概讲解了一下ClassLoader的作用以及一个最基本的加载流程，接下来将讲解一下ClassLoader加载的方式，这里就不得不讲一下ClassLoader在这里使用了双亲委托模式进行类加载。

每一个自定义ClassLoader都必须继承ClassLoader这个抽象类，而每个ClassLoader都会有一个parent ClassLoader，我们可以看一下ClassLoader这个抽象类中有一个getParent()方法，这个方法用来返回当前 ClassLoader的parent，注意，这个parent不是指的被继承的类，而是在实例化该ClassLoader时指定的一个 ClassLoader，如果这个parent为null，那么就默认该ClassLoader的parent是bootstrap classloader，这个parent有什么用呢?

我们可以考虑这样一种情况，假设我们自定义了一个ClientDefClassLoader，我们使用这个自定义的ClassLoader加载java.lang.String，那么这里String是否会被这个ClassLoader加载呢？事实上java.lang.String这个类并不 是被这个ClientDefClassLoader加载，而是由bootstrap classloader进行加载，为什么会这样？实际上这就是双亲委托模式的原因，因为在任何一个自定义ClassLoader加载一个类之前，它都会先 委托它的父亲ClassLoader进行加载，只有当父亲ClassLoader无法加载成功后，才会由自己加载，在上面这个例子里，因为 java.lang.String是属于java核心API的一个类，所以当使用ClientDefClassLoader加载它的时候，该 ClassLoader会先委托它的父亲ClassLoader进行加载，上面讲过，当ClassLoader的parent为null 时，ClassLoader的parent就是bootstrap classloader，所以在ClassLoader的最顶层就是bootstrap classloader，因此最终委托到bootstrap classloader的时候，bootstrap classloader就会返回String的Class。

我们来看一下ClassLoader中的一段源代码:

```
protected synchronized Class loadClass(String name, boolean resolve) throws ClassNotFoundException{
    // 首先检查该name指定的class是否有被加载
    Class c = findLoadedClass(name);
    if (c == null) {
        try {
            if (parent != null) {
                //如果parent不为null，则调用parent的loadClass进行加载
                c = parent.loadClass(name, false);
            }else{
```

```
                    //parent为null，则调用BootstrapClassLoader进行加载
                    c = findBootstrapClass0(name);
                }
            }catch(ClassNotFoundException e) {
                //如果仍然无法加载成功，则调用自身的findClass进行加载
                c = findClass(name);
            }
        }
        if (resolve) {
            resolveClass(c);
        }
        return c;
    }
```

从上面一段代码中，我们可以看出一个类加载的大概过程与之前我所举的例子是一样的，而我们要实现一个自定义类的时候，只需要实现findClass方法即可。

为什么要使用这种双亲委托模式呢?

第一个原因就是因为这样可以避免重复加载，当父亲已经加载了该类的时候，就没有必要子ClassLoader再加载一次。

第二个原因就是考虑到安全因素，我们试想一下，如果不使用这种委托模式，那我们就可以随时使用自定义的String来动态替代java核心api中定义类 型，这样会存在非常大的安全隐患，而双亲委托的方式，就可以避免这种情况，因为String已经在启动时被加载，所以用户自定义类是无法加载一个自定义的 ClassLoader。

上面对ClassLoader的加载机制进行了大概的介绍，接下来不得不在此讲解一下另外一个和ClassLoader相关的类，那就是Class类，每个被ClassLoader加载的class文件，最终都会以Class类的实例被程序员引用，我们可以把Class类当作是普通类的一个模板，JVM根 据这个模板生成对应的实例，最终被程序员所使用。

5. 时间工具类：获取指定年、季、月的开始和结束时间(36)

我们看到在Class类中有个静态方法forName，这个方法和ClassLoader中的loadClass方法的目的一样，都是用来加载class的，但是两者在作用上却有所区别。

xxbcoder
关注 - 0
粉丝 - 1
+加关注

« 上一篇：时间工具类：获取指定年、季、月的开始和结束时间
» 下一篇：newinstance()和new

posted @ 2016-01-08 11:32 xxbcoder 阅读(3059) 评论(0) 编辑 收藏