补充：Java反射

浙江大学城市学院
彭 彬
pengb@zucc.edu.cn

反射

Reflection **is a feature in the Java programming language**. It allows an **executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program**. For example, it's possible for a Java class to obtain the names of all its members and display them.

The ability to examine and manipulate a Java class from within itself may not sound like very much, but in other programming languages this feature simply doesn't exist. For example, there is no way in a Pascal, C, or C++ program to obtain information about the functions defined within that program.

One tangible use of reflection is in JavaBeans, where software components can be manipulated visually via a builder tool. The tool uses reflection to obtain the properties of Java components (classes) as they are dynamically loaded.

反射~Example

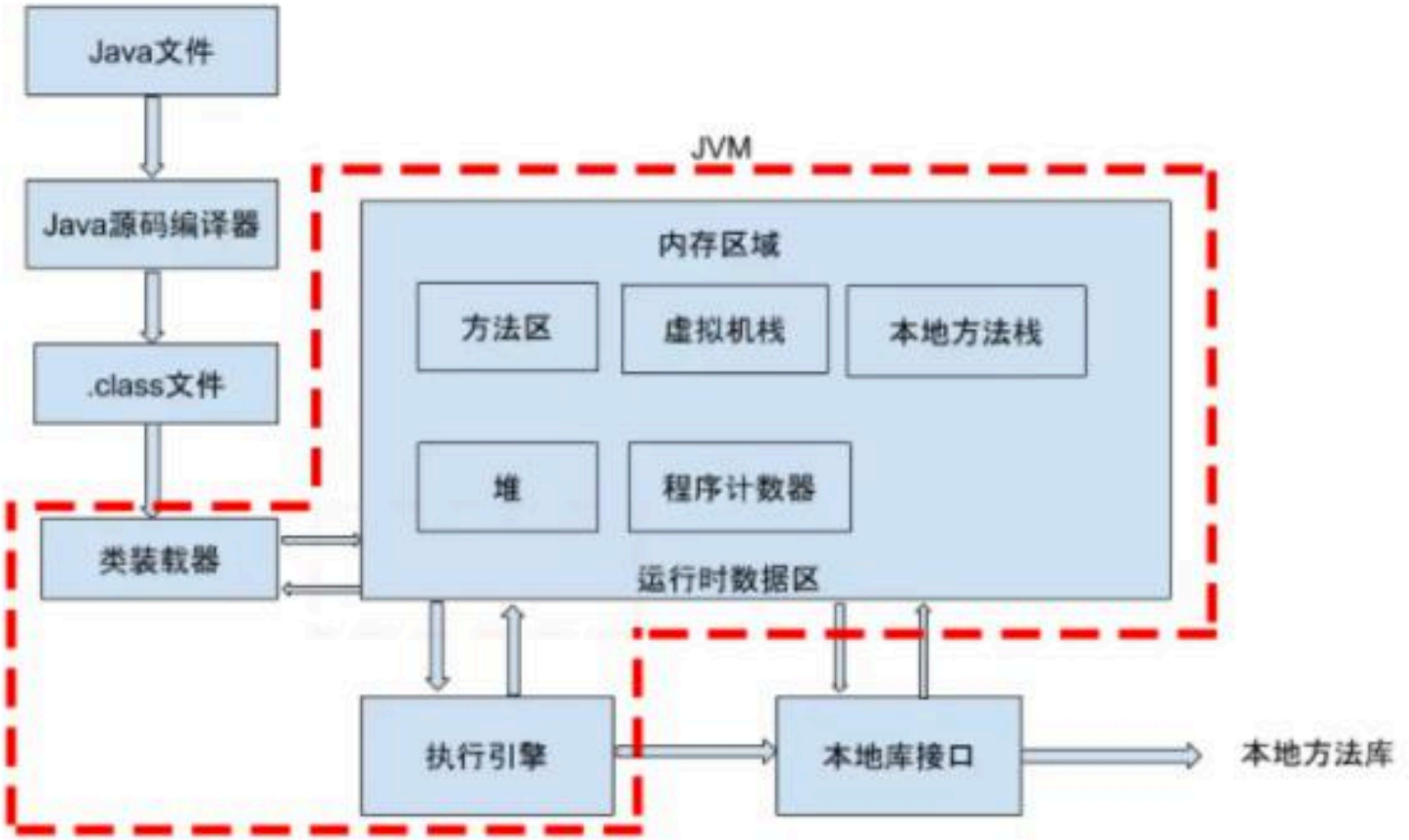
下面的例子通过创建一个Class实例，然后通过反射获取对应类具有的方法及其定义(代码BasicMain)

```
import java.lang.reflect.*;

   public class DumpMethods {
      public static void main(String args[])
      {
         try {
            Class c = Class.forName(args[0]);
            Method m[] = c.getDeclaredMethods();
            for (int i = 0; i < m.length; i++)
            System.out.println(m[i].toString());
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

类加载

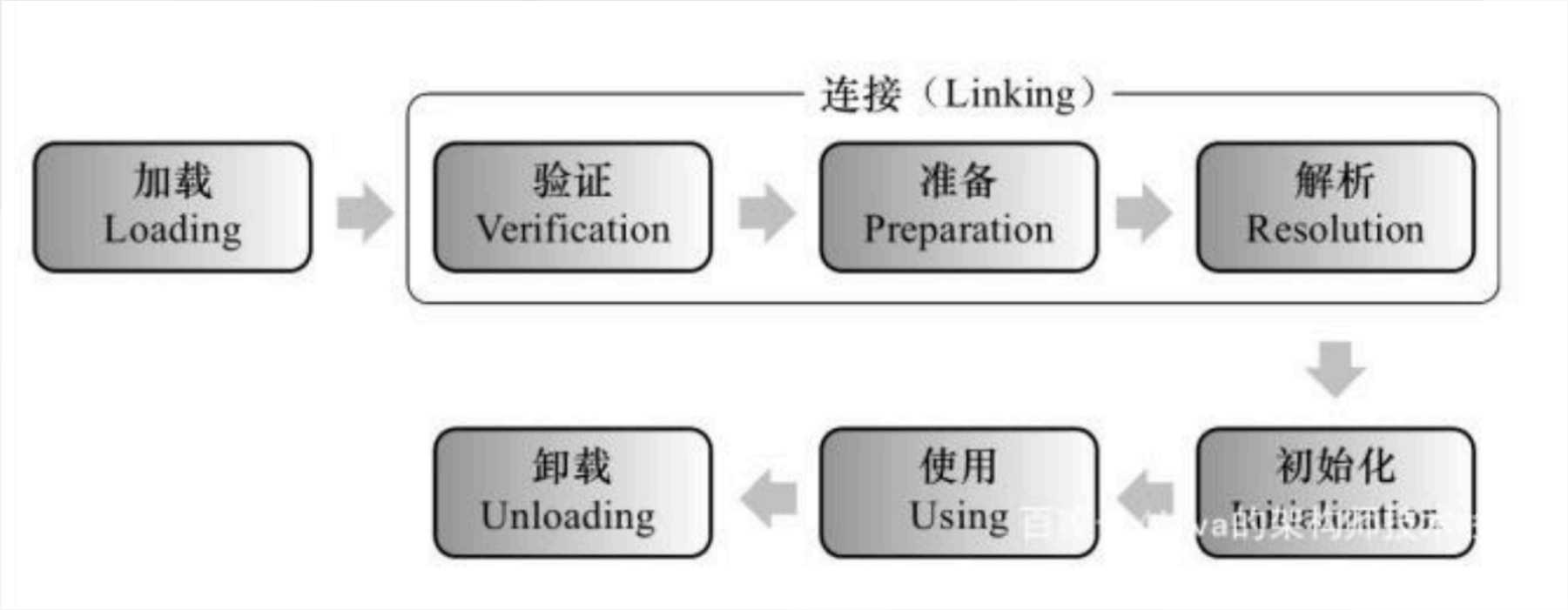
java文件通过编译器变成了.class文件，接下来类加载器又将这些.class文件加载到JVM中。其中**类装载器**的**作用**其实就是类的加载。



类加载将类的.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区创建一个 java.lang.Class对象，用来描述类定义的数据结构

类加载

类从被加载到虚拟机内存中开始，到卸载出内存为止，它的整个生命周期包括：加载、验证、准备、解析、初始化、使用和卸载七个阶段。

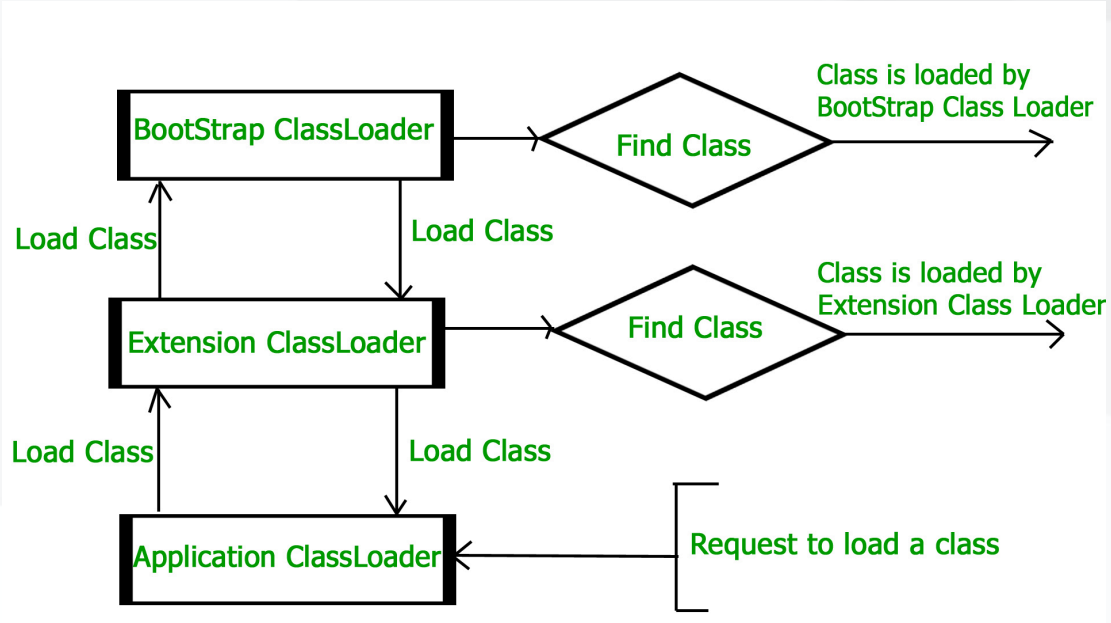


其中类加载的过程包括了加载、验证、准备、解析、初始化五个阶段。

类加载器

Java语言系统自带有三个类加载器：

- **Bootstrap ClassLoader** 、：最顶层的加载类，主要加载核心类库，也就是我们环境变量下面%JRE_HOME%\lib下的rt.jar。
- **Extention ClassLoader** ：扩展类加载器，加载JDK的一些library，包括jre/lib/ext目录中的库。
- **System ClassLoader:** ：也称为SystemAppClass。 加载应用的classpath的所有类。



类加载器~使用特定的类加载器

比如你可以使用URLClassLoader从互联网上加载java类

```
import java.net.*;
import java.io.*;
public class MyLoader {
   public static void main (String argv[]) throws Exception {

      URLClassLoader loader = new URLClassLoader(new URL[] { new URL("http://www.javacourses.com/classes/") });

      // Load class from class loader. argv[0] is the name of the class to be loaded
      Class c = loader.loadClass (argv[0]);

      // Create an instance of the class just loaded
      Object o = c.newInstance();

  }
}
```

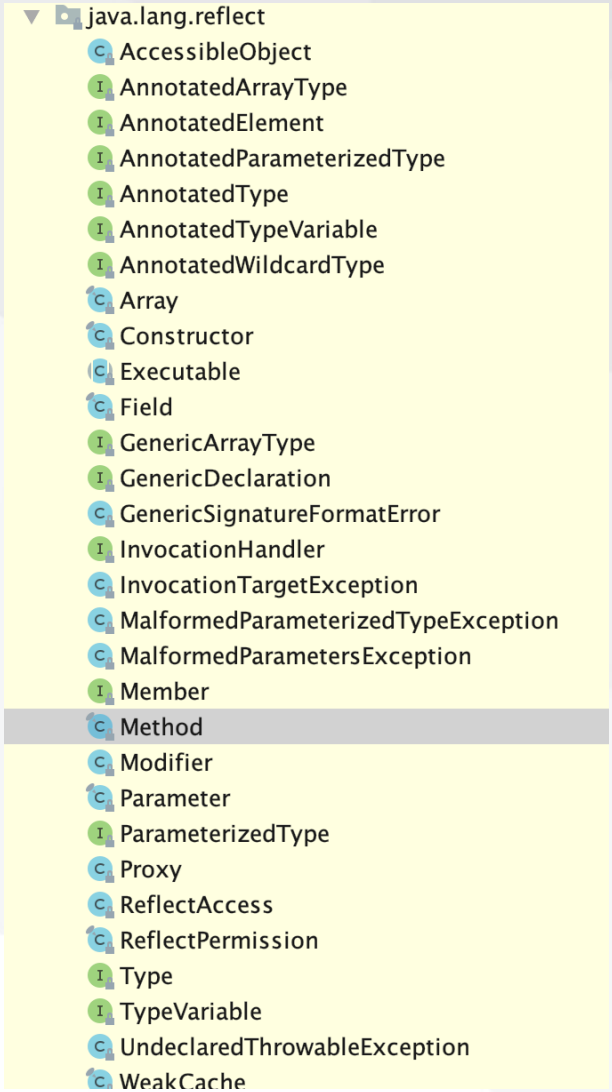
再次认识反射

反射就是一种工作机制，它可以通过检视类加载器加载的类的描述信息获取类的方法、属性等，并且提供了通过这些信息进行创建对象，运行方法的机制。所以基本的核心过程可以描述如下：

1. 类加载器加载.class字节码定义
2. 创建类的描述对象（Class类的实例）
3. 通过反射可以基于Class类实例获取一个特定类的定义信息
4. 通过反射可以创建Class类实例对应类的对象
5. 通过反射可以调用一个Class类实例对应类的对象方法

反射的使用

反射机制的类基本都定义在java.lang.reflect包中



而作为类“蓝图”的Class类定义在java.lang中

```
public final class Class<T> implements java.io.Serializable,
                              GenericDeclaration,
                              Type,
                              AnnotatedElement {
    private static final int ANNOTATION= 0x00002000;
    private static final int ENUM      = 0x00004000;
    private static final int SYNTHETIC = 0x00001000;

    private static native void registerNatives();
    static {
        registerNatives();
    }

    /*
     * Private constructor. Only the Java Virtual Machine creates Class objects.
     * This constructor is not used and prevents the default constructor being
     * generated.
     */
    private Class(ClassLoader loader) {
        // Initialize final field for classLoader.  The initialization value of non-null
        // prevents future JIT optimizations from assuming this final field is null.
        classLoader = loader;
    }
```

反射的使用

反射使用上，最基本的就是获取Class的实例定义，然后通过此示例创建对象，调用对象方法等。获取Class实例的三种途径：

1.通过getClass()方法

```
User user = new User();
Class clazz = user.getClass();
```

2.通过 类名.class 。

```
Class clazz = User.class;
```

3. 通过Class.forName方法

```
Class clazz = Class.forName("com.yourapp.User");
```

方法3比方法1，2更具有动态性，无需import具体的类

反射的使用

获取Class的实例后，就可以调用Class类的方法继续获取方法，调用方法等。见示例工程中的ReflectionMain，比如：

```
// Creating class object from the object using
// getclass method
Class cls = obj.getClass();
System.out.println("The name of class is " +
        cls.getName());

// Getting the constructor of the class through the
// object of the class
Constructor constructor = cls.getConstructor();
System.out.println("The name of constructor is " +
        constructor.getName());

System.out.println("The public methods of class are : ");

// Getting methods of the class through the object
// of the class by using getMethods
Method[] methods = cls.getMethods();

// Printing method names
for (Method method:methods) {
    System.out.println(method.getName());
}
```

参考内容

下面是参考内容

https://docs.oracle.com/javase/tutorial/reflect/

https://www.oracle.com/technical-resources/articles/java/javareflection.html

https://www.geeksforgeeks.org/reflection-in-java/

https://zhuanlan.zhihu.com/p/80519709

https://www.jianshu.com/p/62f46357afcc

https://www.oracle.com/technical-resources/articles/javase/classloaders.html

https://baijiahao.baidu.com/s?id=1636309817155065432&wfr=spider&for=pc

https://www.geeksforgeeks.org/classloader-in-java/

END

Pb&Lois