

第十三章 - 类加载、反射

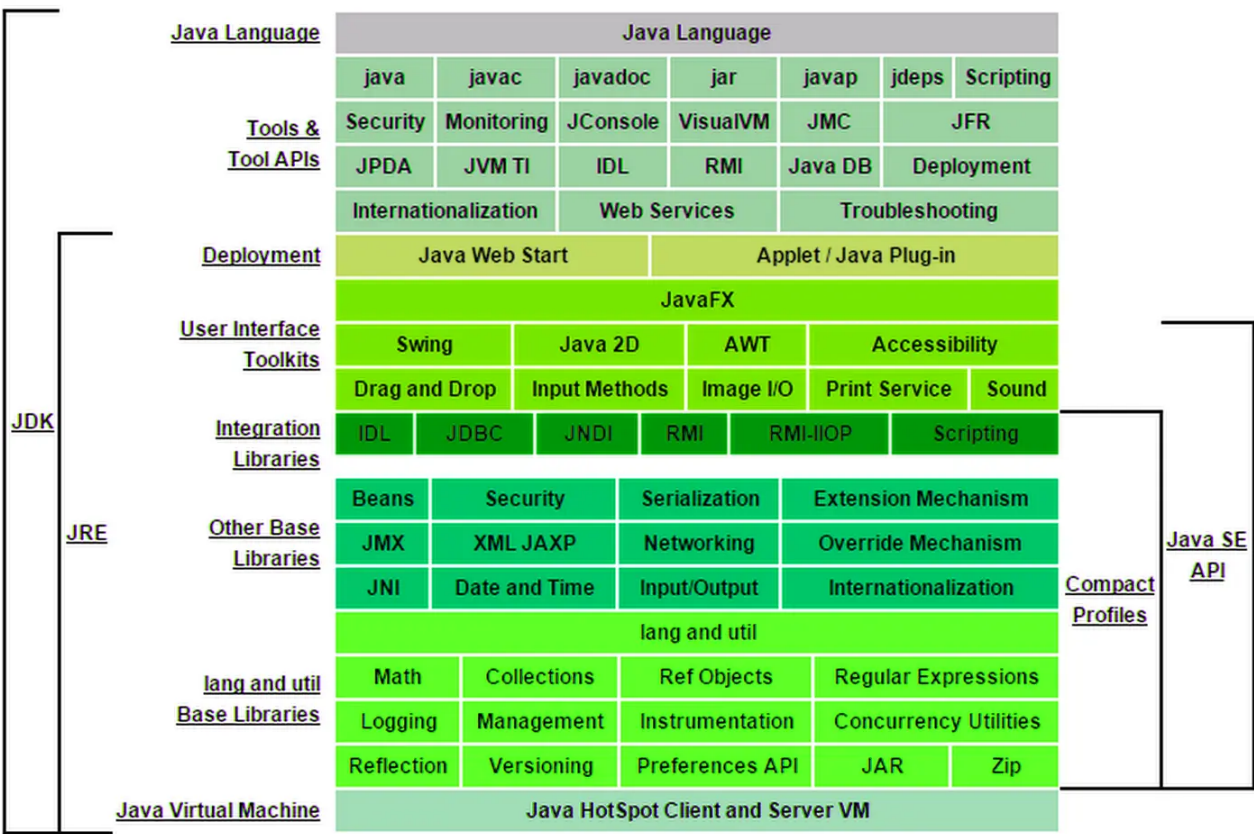
1 基础回顾

在Java课程的第一章，我们就讲过JVM虚拟机、类加载相关内容，现在简单回顾下之前学习的内容。

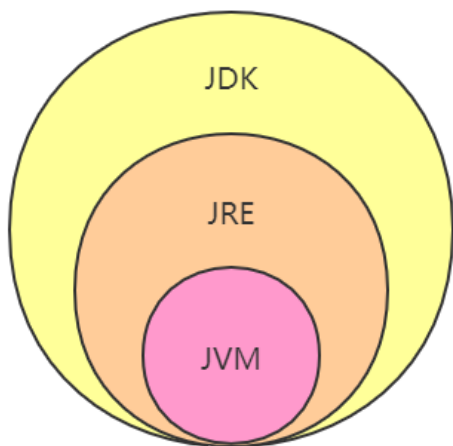
1) JDK、JRE、JVM关系

- JDK(Java Development Kit) Java开发工具包 Java开发必备
- JRE(Java Runtime Environment) Java程序运行时环境 Java程序运行必备
- JVM(Java Virtual Mechain) Java虚拟机

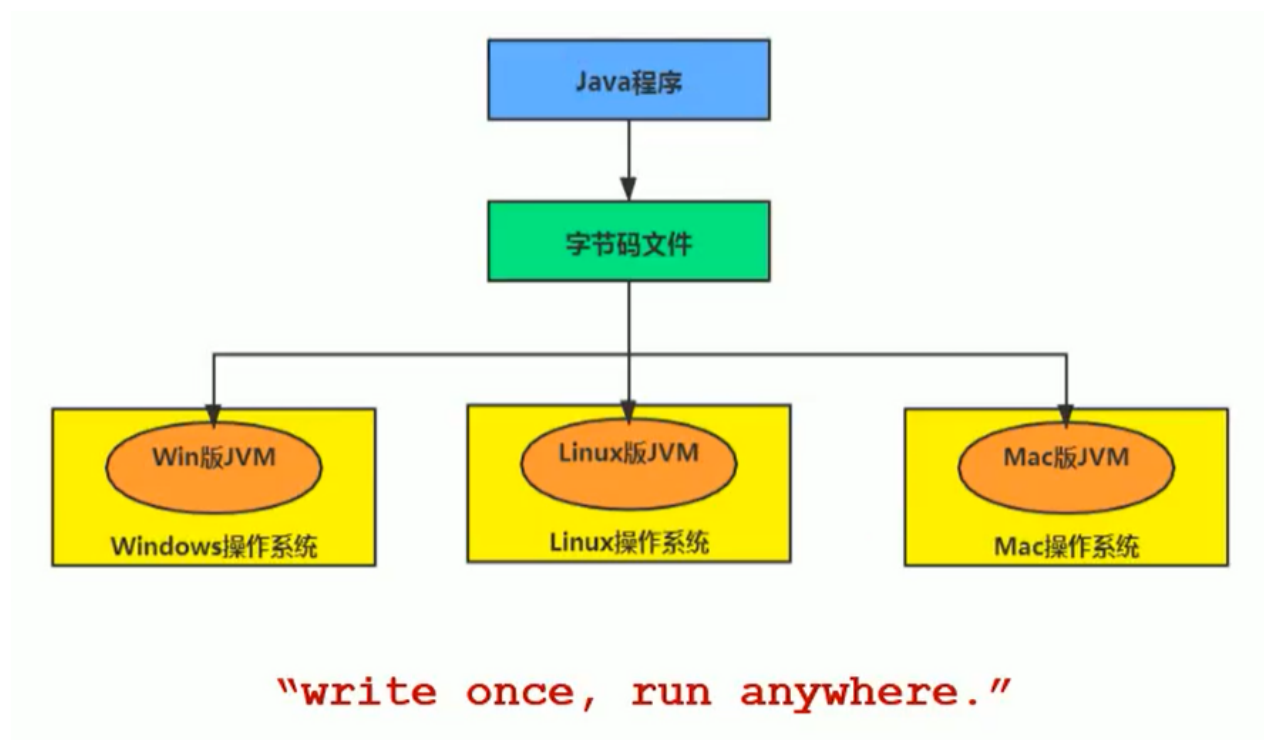
具体关系如下：



结论：JDK中包含JRE，JRE中包含JVM



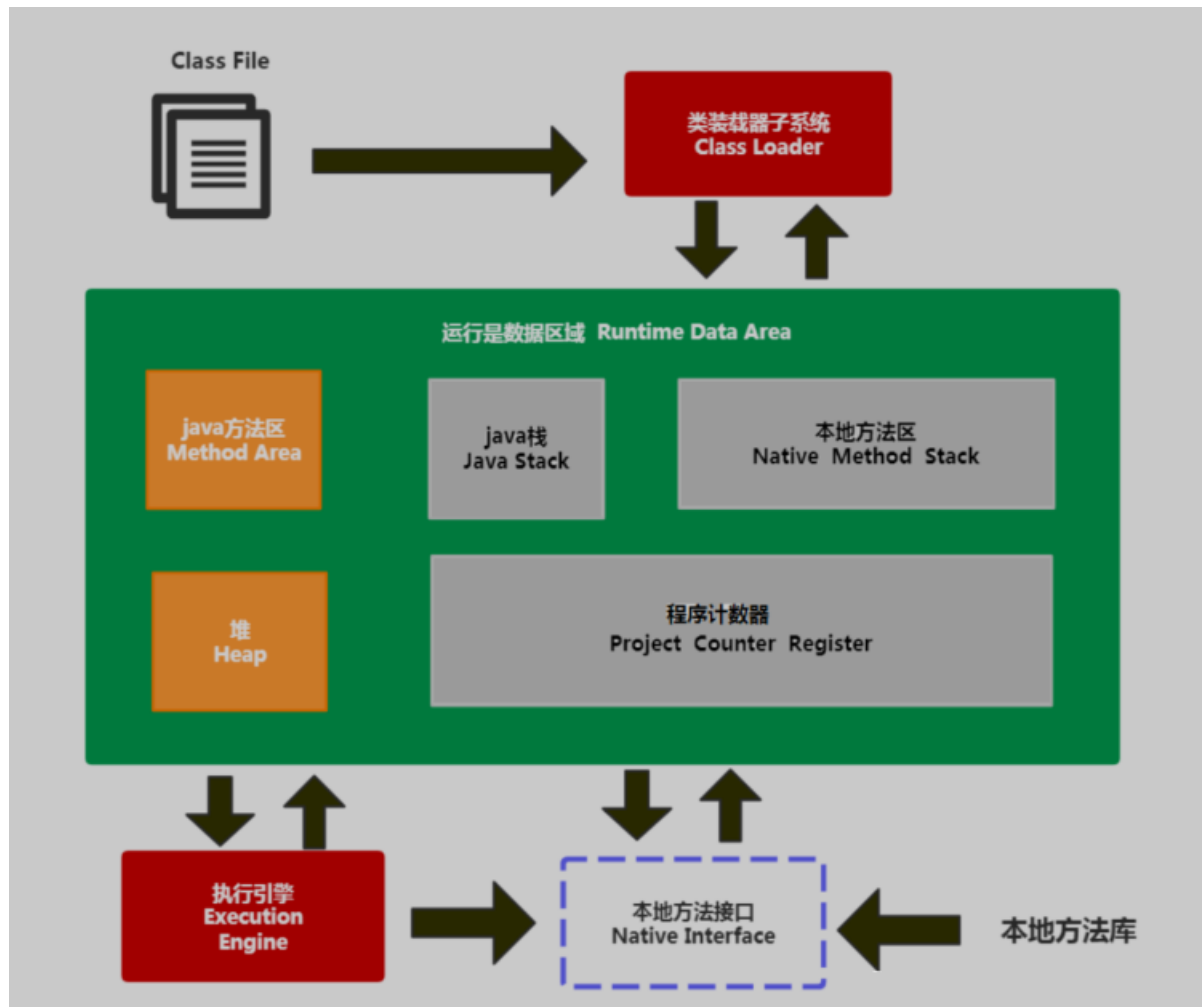
2) Java程序跨平台原理



2 JVM虚拟机

JVM (Java Virtual Machine) 是Java平台的核心组件，它提供了跨平台的能力，使得Java程序可以在不同的操作系统上运行。JDK中的JVM**负责解释和执行Java字节码文件**，同时还提供了内存管理、垃圾回收等功能，使得Java程序能够高效、安全地运行。

JVM内存结构



类加载器 (Class Loader)：类加载器负责加载Java字节码文件（.class文件），并将其转换为可执行的代码。它将类加载到JVM的运行时数据区域中，并解析类的依赖关系

运行时数据区 (Runtime Data Area)：运行时数据区域是JVM用于存储程序运行时数据的区域。它包括以下几个部分：

- 方法区 (Method Area) : 用于存储类的结构信息、常量池、静态变量等
- 堆 (Heap) : 用于存储对象实例和数组内存
- 栈 (Stack) : 也叫做虚拟机栈, 方法调用执行、局部变量所需内存由它提供
- 本地方法栈 (Native Method Stack) : 本地方法栈与虚拟机栈所发挥的作用非常相似, 其区别是虚拟机栈为虚拟机执行Java方法服务, 而本地方法栈则是为虚拟机使用到的本地 (Native) 方法服务
- 程序计数器 (Program Counter) : 用于存储当前线程执行的字节码指令的地址

执行引擎 (Execution Engine) : 执行引擎负责执行编译后的字节码指令, 将其转换为机器码并执行。它包括**解释器**和**即时编译器 (Just-In-Time Compiler, JIT)** 两个部分, 用于提高程序的执行效率 (其具体工作原理, 下一章节具体讨论)

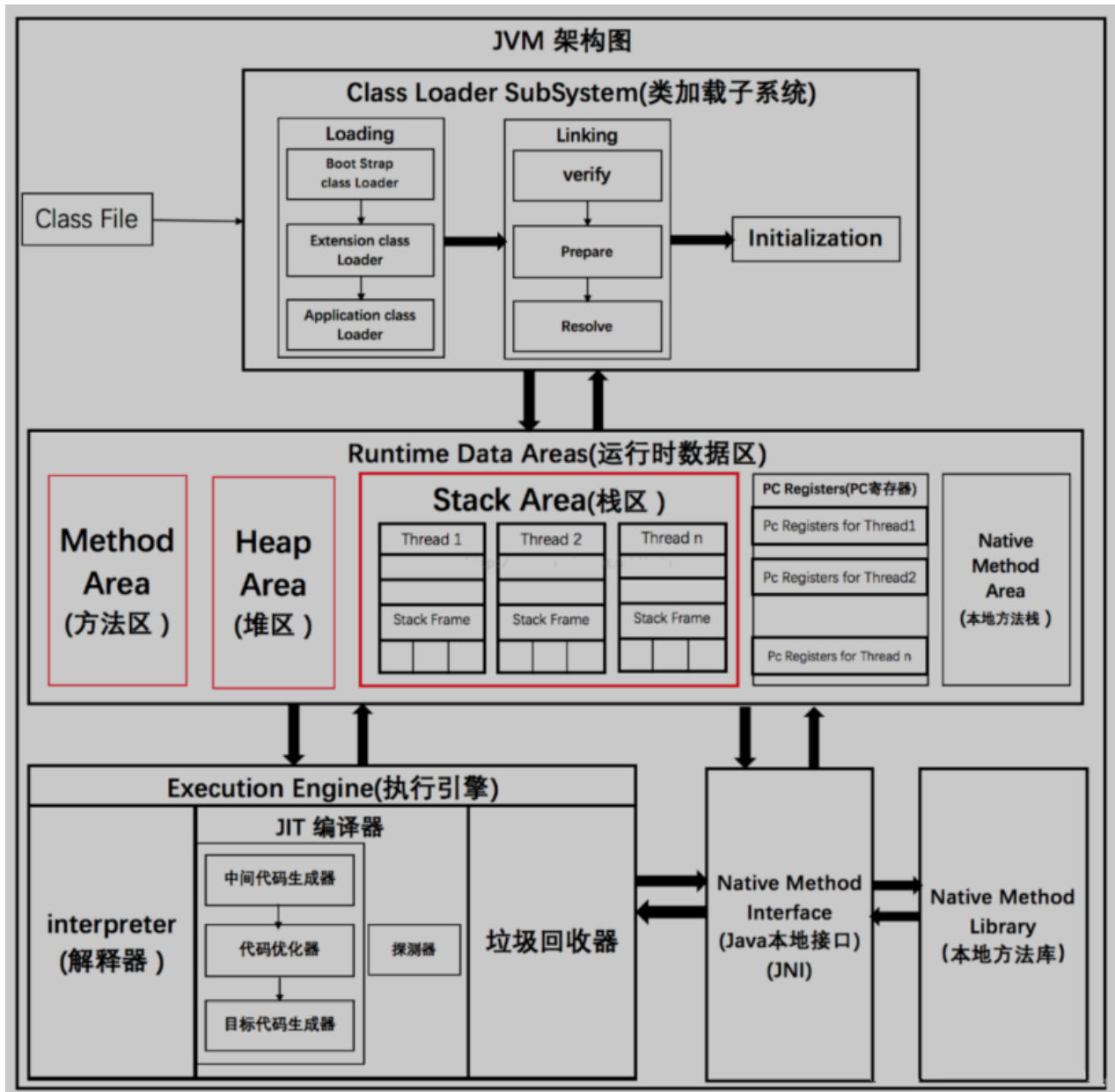
垃圾回收器 (Garbage Collector) : 垃圾回收器负责自动回收不再使用的对象和释放内存空间。它通过标记-清除、复制、标记-整理等算法来进行垃圾回收

本地方法接口 (Native Method Interface) : 本地方法接口允许Java程序调用本地方法, 即使用其他语言编写的代码

3 类加载

通过上文大家已大致了解JVM内部构成, 下面我们来讨论类加载具体细节及JVM详细构成。

JVM架构及执行流程如下:



- **解释执行**

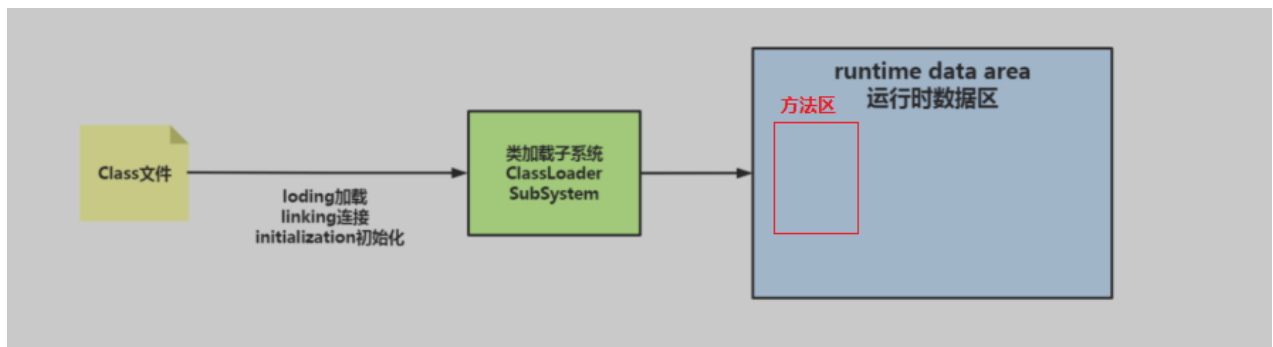
class文件内容，需要交给JVM进行解释执行，简单理解就是JVM解释一行就执行一行代码。所以如果Java代码全是这样的运行方式的话，效率会稍低一些。

- **JIT (Just In Time) 即时编译**

执行代码的另一种方式，JVM可以把Java中的 **热点代码** 直接编译成计算机可以运行的二进制指令，这样后续再调用这个热点代码的时候，就可以直接运行编译好的指令，大大提高运行效率。

3.1 类加载器

类加载器可以将编译得到的 **.class文件**（存储在磁盘上的物理文件）加载在到内存中。



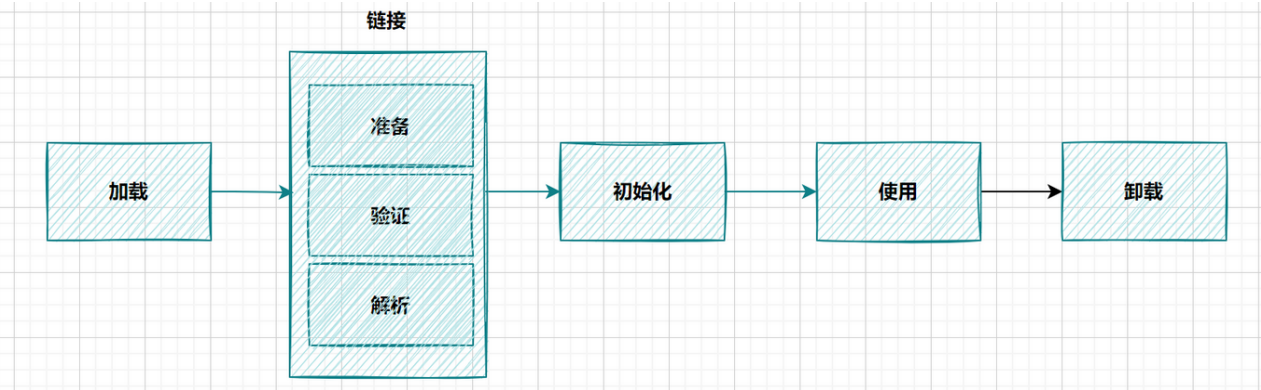
3.2 加载时机

当第一次使用到某个类时，该类的class文件会被加载到内存方法区。

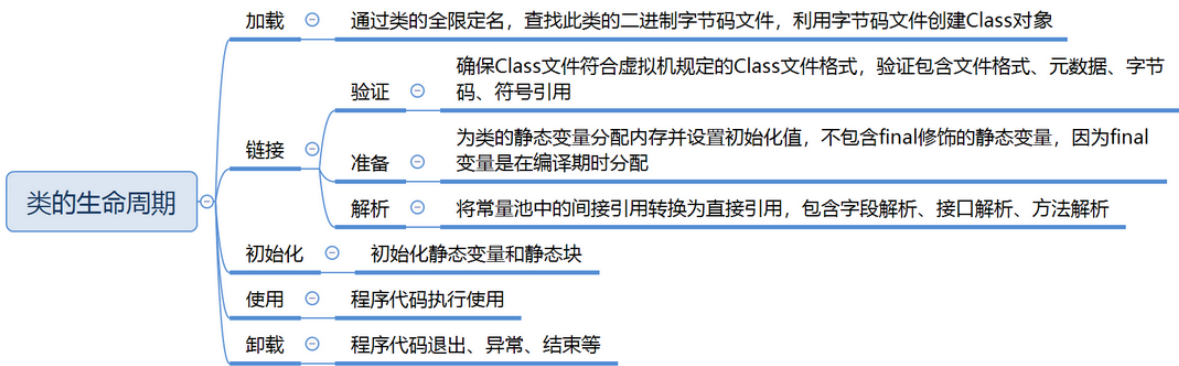
- 使用 **java** 命令来运行某个主类
- 创建类的实例（对象）
- 调用类的 **static方法**
- 访问类或接口的 **static成员**，或者为该类static成员赋值
- 初始化某个类时，其父类会被自动加载
- 使用反射方式（下一章节）来获取类的字节码对象时，会加载某个类或接口的 **class文件**

3.3 加载过程

类加载的过程：加载、验证、准备、解析、初始化



具体加载步骤：



1) 加载 (Loading)

将类的字节码文件加载到内存方法区中。这个阶段由类加载器完成，类加载器根据类的全限定名来定位并读取类的字节码文件，然后将字节码转换为JVM内部的数据结构。

2) 链接 (Linking)

- 验证 (Verification)

对加载的字节码进行验证，确保字节码的结构和语义是正确的（确保class文件中的信息符合虚拟机规范，有没有安全隐患）。

验证过程包括文件格式验证、元数据验证、字节码验证和符号引用验证等，以确保字节码的安全性和正确性。

- 准备 (Preparation)

为类的 **static静态变量** 分配内存，并设置默认初始值。

这个阶段会在方法区中为类的静态变量分配内存空间，并设置默认初始值（如0、0.0、null等），但不会执行静态变量的初始化代码。

- 解析 (Resolution)

将类的符号引用解析为直接引用。

在解析阶段，将**符号引用（如类名、方法名、字段名）**转换为**直接引用（如直接指向方法、字段的指针或偏移量）**，以便于后续的访问和调用。

3) 初始化 (Initialization)

对类进行初始化，包括执行静态变量的赋值和静态代码块的初始化。

在此阶段，会按照程序的顺序执行类的静态变量赋值和静态代码块中的初始化代码，完成类的初始化工作。

注意事项：

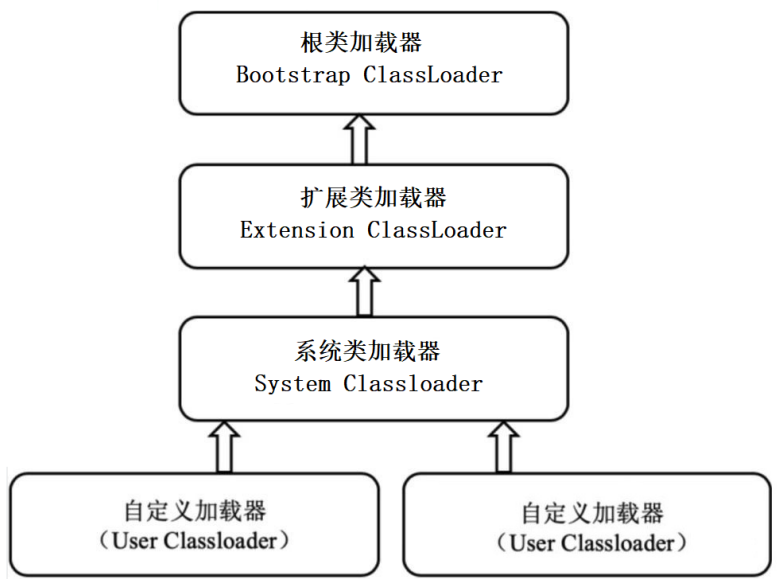
类加载过程是按需进行的，即在首次使用类时才会触发类的加载和初始化。此外，类加载过程是由Java虚拟机的类加载器负责完成的，不同的类加载器可能有不同的加载策略和行为。

类加载小结：

JVM的类加载过程包括加载、验证、准备、解析和初始化等阶段，它们共同完成将Java类加载到内存中，并为类的静态变量分配内存、解析符号引用、执行静态代码块等操作，最终使得类可以被正确地使用和执行。

3.4 加载器分类

JDK8类加载器可以分为以下四类：



- **Bootstrap ClassLoader 根类加载器**

也被称为引导类加载器，通常表示为null。

它是Java虚拟机的一部分，负责加载Java核心类库，比如 `rt.jar` 等，根加载器是所有类加载器的顶级加载器，它不是一个Java对象，而是由JVM实现的一部分。

类一般存在 `%JAVA_HOME%\jre\lib\rt.jar` 中



- **Extension ClassLoader 扩展类加载器**

负责加载Java的扩展类库，也可以通过 `java.ext.dirs` 系统属性来指定扩展类库的路径

这些类一般存在 `%JAVA_HOME%\jre\lib\ext\` 下的jar包中

电脑 > Windows (C:) > Program Files > Java > jdk1.8.0_74 > jre > lib > ext					搜索"e"
名称	修改日期	类型	大小		
 access-bridge-64.jar	2022/6/22 14:36	Executable Jar File	184 KB		
 cldrdata.jar	2022/6/22 14:36	Executable Jar File	3,771 KB		
 dnsns.jar	2022/6/22 14:36	Executable Jar File	9 KB		
 jaccess.jar	2022/6/22 14:36	Executable Jar File	44 KB		
 jfxrt.jar	2022/6/22 14:36	Executable Jar File	17,734 KB		
 localedata.jar	2022/6/22 14:36	Executable Jar File	2,195 KB		

- **System ClassLoader 系统类加载器**

它负责加载应用程序的类，包括**用户自定义的类和第三方库**等。它是大多数Java应用程序默认的分类加载器。

系统类加载器的**搜索路径**包括**当前工作目录**和**CLASSPATH环境变量**指定的路径

- **User ClassLoader 自定义类加载器**

自定义类加载器可以用于加载特定的类或实现类加载的特殊需求，目前应用很少，可忽略。

案例演示：

```
1 package com.briup.chap13.test;
2
3 public class Test034_ClassLoader {
4     public static void main(String[] args) {
5         //获取系统类加载器
6         ClassLoader systemClassLoader =
7             ClassLoader.getSystemClassLoader();
8         //获取系统类加载器的父加载器 --- 扩展类加载器
```

```

9      ClassLoader extClassLoader =
      systemClassLoader.getParent();
10
11      //获取平台类加载器的父加载器 --- 根类加载器
12      ClassLoader bootClassLoader =
      extClassLoader.getParent();
13
14      System.out.println("系统类加载器" + systemClassLoader);
15      System.out.println("扩展类加载器" + extClassLoader);
16      System.out.println("根类加载器" + bootClassLoader);
17  }
18  }

```

3.5 双亲委托

双亲委托机制是Java类加载器的一种工作机制，通过层级加载和委托父类加载器来加载类，确保类的唯一性、安全性和模块化。在学习这个知识点之前，大家看下面题目。

1) 问题引入

用户自定义类 `java.lang.String`，在测试类main方法中使用该类，思考：类加载器到底加载是哪个类，是JDK提供的String，还是用户自定义的String？

自定义String类：

```

1  package java.lang;
2
3  import java.util.Arrays;
4
5  public class String {
6      private char[] arr;
7
8      public String() {
9          System.out.println("in String() ...");

```

```

10         arr = new char[10];
11     }
12
13     public String(char[] array) {
14         System.out.println("in String(char[]) ...");
15         int len = array.length;
16         arr = new char[len];
17         System.arraycopy(array, 0, arr, 0, len);
18     }
19
20     @Override
21     public String toString() {
22         return "MyString: " + Arrays.toString(arr);
23     }
24 }

```

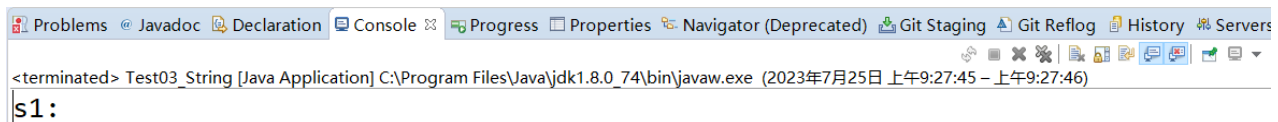
测试类:

```

1  package com.briup.chap13.test;
2
3  import java.lang.String;
4
5  public class Test035_String {
6      public static void main(String[] args) {
7          String s1 = new String();
8
9          System.out.println("s1: " + s1);
10     }
11 }

```

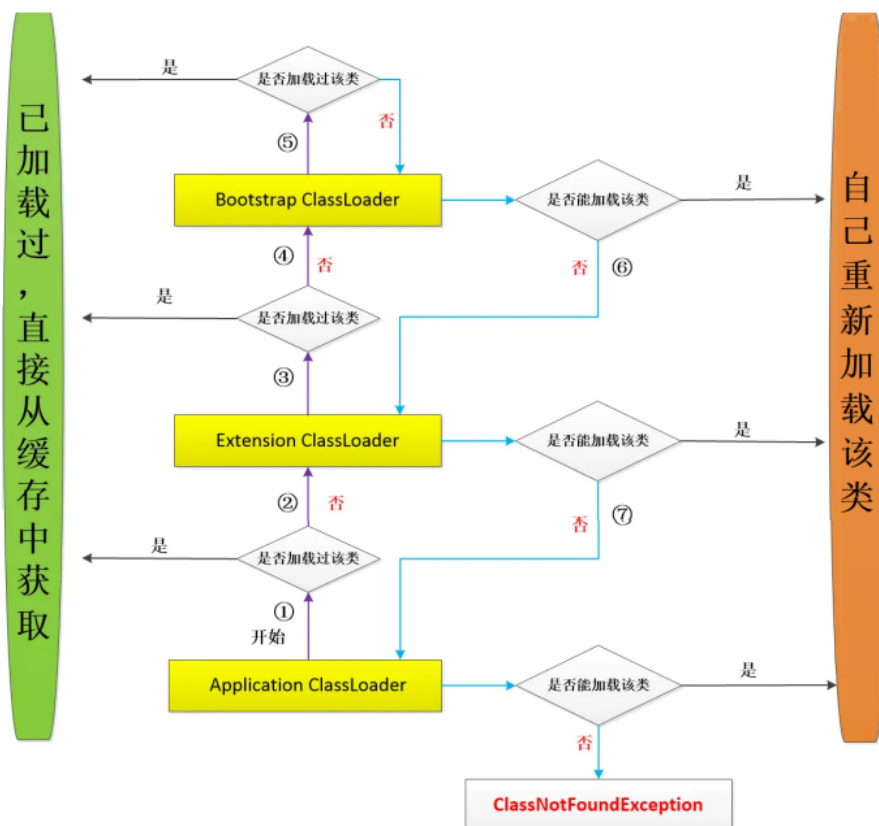
程序运行效果:



从运行效果可知: **最终加载的类是JDK提供的 `java.lang.String`**

为什么？答案是**双亲委托机制**！

2) 双亲委托机制



如果一个类加载器收到类加载请求，它并不会自己先去加载，而是把这个请求委托给父类的加载器去执行，如果父类加载器还存在其父类加载器，则进一步向上委托，最终加载请求会到达顶层的启动类加载器 **Bootstrap ClassLoader**。

如果顶层类加载器可以完成加载任务，则进行class文件类加载，加载成功后返回。如果当前类加载器无法加载，则向下委托给子类加载器，此时子类加载器才会尝试加载，成功则返回，失败则继续往下委托，如果所有的加载器都无法加载该类，则会抛出ClassNotFoundException，这就是双亲委托机制。

3.6 常用方法

方法名	说明
<code>public static ClassLoader getSystemClassLoader()</code>	获取系统类加载器
<code>public InputStream getResourceAsStream(String name)</code>	加载某一个资源文件，注意是相对路径

案例展示：

准备一个jdbc的配置文件 `jdbc.properties`，借助类加载器中方法解析，遍历输出其配置内容。

配置文件 `jdbc.properties`：

```
1  driverClass=com.mysql.jdbc.Driver
2  url=jdbc:mysql://localhost:3306/db01
3  username=root
4  password=briup
```

测试类：

```
1  package com.briup.chap13.test;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.util.Map.Entry;
6  import java.util.Properties;
7  import java.util.Set;
8
9  // static ClassLoader getSystemClassLoader()    获取系统类加载器
10 // InputStream getResourceAsStream(String name) 加载当前类class
    文件相同目录下资源文件
11 public class Test036_LoadFile {
```

```

12     public static void main(String[] args) throws IOException
13     {
14         //1.获取系统类加载器
15         ClassLoader systemClassLoader =
16         ClassLoader.getSystemClassLoader();
17
18         //2.利用加载器去加载一个指定的文件
19         // 参数：文件的路径（注意，该路径为相对路径，相对于当前类
20         class文件存在的目录）
21         // 返回值：字节流
22         InputStream is =
23         systemClassLoader.getResourceAsStream("jdbc.properties");
24         System.out.println("is: " + is);
25
26         //3.实例化Properties对象，解析配置文件内容并输出
27         Properties prop = new Properties();
28         prop.load(is);
29
30         //配置文件内容遍历
31         Set<Entry<Object, Object>> entrySet = prop.entrySet();
32         for (Entry<Object, Object> entry : entrySet) {
33             String key = (String) entry.getKey();
34             String value = (String) entry.getValue();
35
36             System.out.println(key + ": " + value);
37         }
38     }

```

运行效果：

```
Problems @ Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Serv
<terminated> Test05_LoadFile [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 上午11:18:09 - 上午11:18:09)
is: java.io.BufferedInputStream@7852e922
driverClass: com.mysql.jdbc.Driver
password: briup
url: jdbc:mysql://localhost:3306/db01
username: root
```

注意事项： `getResourceAsStream(String path)`，参数 `path` 是相对路径，相对当前测试类 `class` 文件所在的目录！

此电脑	新加卷 (F:)	develop	220726-WorkSpace	2023-CoreJava	bin > com > briup > chap13 > test
名称	修改日期	类型	大小		
jdbc.properties	2023/7/25 11:05	PROPERTIES 文件	1 KB		
Test01_EnumBasic.class	2023/7/24 16:58	CLASS 文件	1 KB		
Test02_Define01.class	2023/7/24 18:06	CLASS 文件	1 KB		

4 反射

4.1 反射概述

Java反射机制是指在Java程序在运行状态下，**动态地获取、检查和操作类的信息和对象的能力**。

反射机制作用：

- 对于任意一个类，都能够知道这个类的所有属性和方法
- 对于任意一个对象，都能够调用它的任意一个方法和属性

这种动态获取信息以及动态调用对象方法的功能称为Java语言的反射机制。

当一个类被使用的时候，类加载器会把该类的字节码文件装入内存（类加载），同时在堆空间创建一个 **字节码对象（Class类对象）**，这个对象是Java**反射机制的核心**，它包含了一个类运行时信息。

4.2 反射核心类

在Java中，**Class** 类是一个重要的核心类，它用于表示一个类或接口的运行时信息。每个类在Java虚拟机中都有一个对应的 **Class** 对象，可以通过该对象获取类的构造函数、方法、属性等信息，并且可以进行实例化对象、方法调用和数据成员访问等操作。

Class核心类JavaSE源码：

```
1  package java.lang;
2
3  //字节码类
4  public final class Class<T> implements java.io.Serializable,
5                                         GenericDeclaration,
6                                         Type,
7                                         AnnotatedElement {
8      //省略...
9
10     //获取类的所有构造方法
11     @CallerSensitive
12     public Constructor<?>[] getConstructors() throws
SecurityException {
13         checkMemberAccess(Member.PUBLIC,
Reflection.getCallerClass(), true);
14         return
copyConstructors(privateGetDeclaredConstructors(true));
15     }
16
```

```

17      //获取类的所有数据成员
18      @CallerSensitive
19      public Field[] getFields() throws SecurityException {
20          checkMemberAccess(Member.PUBLIC,
21              Reflection.getCallerClass(), true);
22          return copyFields(privateGetPublicFields(null));
23      }
24
25      //获取类的所有成员方法
26      @CallerSensitive
27      public Method[] getMethods() throws SecurityException {
28          checkMemberAccess(Member.PUBLIC,
29              Reflection.getCallerClass(), true);
30          return copyMethods(privateGetPublicMethods());
31      }
32  }

```

在Java反射中，`Class`、`Constructor`、`Method`和`Field`是表示类的不同部分的关键类。它们提供了访问和操作类的构造函数、方法和字段的方法。

- `Class` 类：表示一个类的运行时信息。通过 `Class` 类可以获取类的构造函数、方法和字段等信息。可以使用 `Class.forName()` 方法获取一个类的 `Class` 对象，也可以通过对象的 `getClass()` 方法获取其对应的 `Class` 对象。
- `Constructor` 类：表示一个类的构造函数。通过 `Constructor` 类可以创建类的实例。可以使用 `Class` 对象的 `getConstructors()` 或 `getConstructor()` 方法获取构造函数的对象。
- `Method` 类：表示一个类的方法。通过 `Method` 类可以调用类的方法。可以使用 `Class` 对象的 `getMethods()` 或 `getMethod()` 方法获取方法的对象。
- `Field` 类：表示一个类的字段。通过 `Field` 类可以访问和修改类的字段的值。可以使用 `Class` 对象的 `getFields()` 或 `getField()` 方法获取字段的对象。

4.3 字节码对象

JVM虚拟机对类进行加载时，会在堆空间创建一个 **字节码对象（Class类对象）**。

通过该字节码对象程序员可以获取类的构造函数、方法、属性等信息，并且可以进行实例化、方法调用和属性访问等操作。

简单来说：**如果要用反射机制，则必须先获取类的字节码对象**，那么如何获取呢？

获取Class对象方式：

- 使用类字面常量：`类名.class`
- Object类中方法：`对象.getClass()`

```
public final native Class<?> getClass();
```

- 借助Class类中方法：``Class.forName("类的全包名")``

```
public static Class<?> forName(String className);
```

案例展示1：

使用前两种方式获取同一个类的字节码对象，并验证一个类的字节码对象是否唯一。

```
1 package com.briup.chap13.test;
2
3 public class Test043_Class {
4     public static void main(String[] args) {
5         String s = "hello";
6         //1. 对象.getClass()
```

```

7      Class<? extends String> c1 = s.getClass();
8      //2. 类.class
9      Class c2 = String.class;
10
11      System.out.println("c1: " + c1);
12      System.out.println("c2: " + c2);
13
14      //验证同一个类的字节码对象是否唯一
15      System.out.println("c1 == c2 : " + (c1 == c2));
16  }
17 }
18
19 //结果为true

```

案例展示2:

自定义Student类，然后使用三种方式获取该类的字节码对象，并验证字节码对象是否唯一。

自定义Student类:

```

1  package com.briup.chap13.bean;
2
3  public class Student {
4      public String id;
5      private String name;
6      private int age;
7
8      public Student() {}
9
10     private Student(String id) {
11         System.out.println("in private Student(id) ...");
12         this.id = id;
13     }
14

```

```

15      //      String.class,String.class,int.class
16      public Student(String id, String name, int age) {
17          this.id = id;
18          this.name = name;
19          this.age = age;
20      }
21
22      public String getId() {
23          return id;
24      }
25      private void setId(String id) {
26          this.id = id;
27      }
28      public String getName() {
29          return name;
30      }
31      public int getAge() {
32          return age;
33      }
34
35      @Override
36      public String toString() {
37          return "Student [id=" + id + ", name=" + name + ",
38      age=" + age + "]";
39      }

```

测试类:

```

1      package com.briup.chap13.test;
2
3      import com.briup.chap13.bean.Student;
4
5      //使用3种不同方式，获取自定义类的字节码对象，并验证是否唯一
6      public class Test043_Class {
7          public static void main(String[] args) throws Exception {

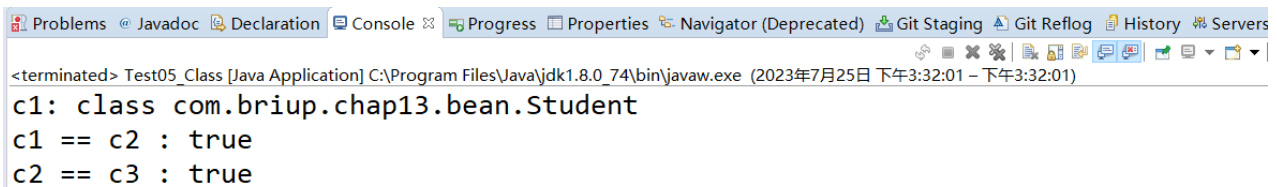
```

```

8      //第三种方式获取字节码对象，注意：参数为类的全包名
9      Class<?> c1 =
    Class.forName("com.briup.chap13.bean.Student");
10
11      Class c2 = Student.class;
12
13      Student s = new Student();
14      Class<? extends Student> c3 = s.getClass();
15
16      System.out.println("c1: " + c1);
17      System.out.println("c1 == c2 : " + (c1 == c2));
18      System.out.println("c2 == c3 : " + (c3 == c2));
19  }
20 }

```

运行效果：



```

<terminated> Test05_Class [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午3:32:01 - 下午3:32:01)
c1: class com.briup.chap13.bean.Student
c1 == c2 : true
c2 == c3 : true

```

注意事项：一个类的字节码对象，有且只有一个！

补充：其他类型字节码对象获取

- 基本数据类型获取字节码对象固定格式：数据类型.class
- 数组类型获取格式：数据类型[].class; 数组名.getClass();

案例展示：

```

1  //基本类型|数组 字节码对象
2  public static void main(String[] args) {
3      //基本类型字节码对象：
4      // 数据类型.class
5      System.out.println(int.class);

```

```
6      System.out.println(double.class);
7
8      //数组 字节码对象
9      // 数据类型[].class;
10     // 数组名.getClass();
11     int[] arr = {1,2,4,3,6};
12     System.out.println(arr.getClass());
13     System.out.println(int[].class);
14 }
15
16 //输出结果:
17 int
18 double
19 class [I
20 class [I
```

4.4 构造方法

通过反射可以获取类的构造方法（含private）对象，并借助其实例化对象。

1) 构造器相关方法

方法名	说明
Constructor<?>[] getConstructors()	返回所有public构造方法对象的数组
Constructor<?>[] getDeclaredConstructors()	返回所有构造方法（含private）对象的数组
Constructor getConstructor(Class<?>... parameterTypes)	返回指定public构造方法对象
Constructor getDeclaredConstructor(Class<?>... parameterTypes)	返回指定构造方法（含private）对象

案例展示1:

获取Student的字节码对象，进而获取该类的构造器对象，并遍历输出。

```

1  package com.briup.chap13.test;
2
3  import java.lang.reflect.Constructor;
4  import com.briup.chap13.bean.Student;
5
6  public class Test044_Constructor {
7      public static void main(String[] args) throws Exception {
8          //1.获取字节码对象
9          Class<?> clazz =
10             Class.forName("com.briup.chap13.bean.Student");
11             System.out.println(clazz);
12
13             //2.由字节码对象获取所有public构造方法
14             Constructor<?>[] carr = clazz.getConstructors();
15             for (Constructor<?> con : carr) {
16                 System.out.println(con);
17             }
18         }
19     }

```



```

17
18         System.out.println("-----");
19
20         //3.获取所有构造方法(含private)
21         Constructor<?>[] carr2 =
clazz.getDeclaredConstructors();
22         for (Constructor<?> con : carr2) {
23             System.out.println(con);
24         }
25     }
26 }

```

2) Constructor类创建对象方法

方法名	说明
T newInstance(Object...initargs)	根据指定的构造方法创建对象
setAccessible(boolean flag)	设置为true,表示取消访问检查

案例展示2:

获取Student的字节码对象，进而获取该类的无参构造器对象，并借助该对象实例化对象。

```

1    //无参构造器对象的使用
2    public static void main(String[] args) throws Exception {
3        //1.获取字节码对象
4        Class<?> clazz =
Class.forName("com.briup.chap13.bean.Student");
5        System.out.println(clazz);
6
7        //2.获取 无参构造器对象

```

```

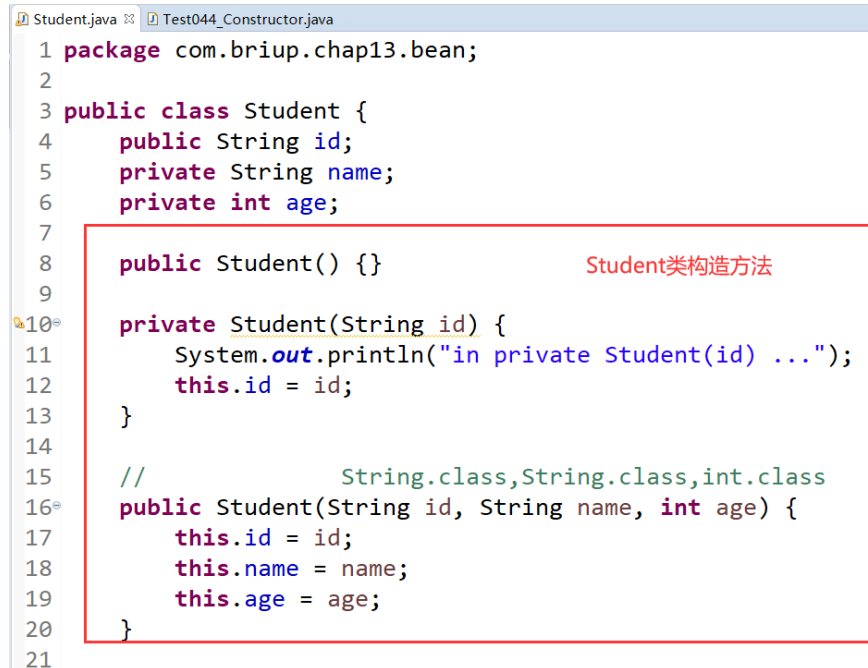
8      Constructor<?> con = clazz.getConstructor();
9
10     //3.使用无参构造器 实例化对象
11     Student s = (Student)con.newInstance();
12
13     System.out.println("s: " + s);
14 }

```

案例展示3:

获取Student的字节码对象，进而获取该类指定构造器对象(含private构造器)，并通过构造器对象实例化对象。

Student类（前面案例已经提供）构造方法如下：



```

Student.java  Test044_Constructor.java
1 package com.briup.chap13.bean;
2
3 public class Student {
4     public String id;
5     private String name;
6     private int age;
7
8     public Student() {}           Student类构造方法
9
10    private Student(String id) {
11        System.out.println("in private Student(id) ...");
12        this.id = id;
13    }
14
15    // String.class,String.class,int.class
16    public Student(String id, String name, int age) {
17        this.id = id;
18        this.name = name;
19        this.age = age;
20    }
21

```

具体测试代码如下：

```

1 public static void main(String[] args) throws Exception {
2     //1.获取Student类字节码对象
3     Class<?> clazz =
4         Class.forName("com.briup.chap13.bean.Student");
5     System.out.println(clazz);
6 }

```

```

5
6    //2.获取指定public构造器
7    Constructor<?> c1 =
8
9    clazz.getConstructor(String.class,String.class,int.class);
10   System.out.println(c1);
11
12   // error    private Student(String);
13   //Constructor<?> c2 = clazz.getConstructor(String.class);
14   //System.out.println(c2);
15
16   //3.获取指定任意构造器（含private）
17   Constructor<?> c2 =
18   clazz.getDeclaredConstructor(String.class);
19   System.out.println(c2);
20
21   System.out.println("-----");
22
23   //4.使用public构造器对象 直接创建Student对象
24   Student s1 = (Student) c1.newInstance("002", "zs", 20);
25   System.out.println("s1: " + s1);
26
27   System.out.println("-----");
28
29   //5.使用private构造器对象 直接创建Student对象
30   //设置c2具有访问权限【核心代码】
31   c2.setAccessible(true);
32   Student s2 = (Student) c2.newInstance("005");
33   System.out.println("s2: " + s2);
34 }

```

4.5 成员变量

通过反射可以获取类的所有数据成员（含private）对象，进而实现数据成员值的获取与设置。

1) Filed相关方法

方法名	说明
Field[] getFields()	返回所有公共成员变量对象的数组(含继承的)
Field[] getDeclaredFields()	返回所有成员变量对象的数组(含private，但不包括继承的)
Field getField(String name)	返回单个公共成员变量对象(含继承的)
Field getDeclaredField(String name)	返回单个成员变量对象(含private，不包括继承的)

案例展示1:

定义HighStudent类，继承Student类:

```
1 package com.briup.chap13.bean;
2
3 public class HighStudent extends Student {
4     private int eScore;
5     public int chinaScore;
6 }
```

定义测试类，获取HighStudent类的public属性对象:

```

1  package com.briup.chap13.test;
2
3  import java.lang.reflect.Field;
4  import com.briup.chap13.bean.Student;
5
6  public class Test045_Field {
7      // 获取public属性
8      public static void main(String[] args) throws Exception {
9          // 1.获取字节码对象
10         Class<?> clazz =
11         Class.forName("com.briup.chap13.bean.HighStudent");
12         System.out.println(clazz);
13
14         System.out.println("-----");
15
16         // 2.获取所有public属性(含继承的)
17         Field[] fields = clazz.getFields();
18         for (Field f : fields) {
19             System.out.println(f);
20         }
21
22         System.out.println("-----");
23
24         // 3.获取指定public属性(含继承的)
25         Field f1 = clazz.getField("id");
26         System.out.println("f1: " + f1);
27     }
28 }

```

运行效果：

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test045_Field [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:27:39 – 下午11:27:40)
class com.briup.chap13.bean.HighStudent
-----
public int com.briup.chap13.bean.HighStudent.chinaScore
public java.lang.String com.briup.chap13.bean.Student.id
-----
f1: public java.lang.String com.briup.chap13.bean.Student.id
```

案例展示2:

获取HighStudent类的所有属性对象（含private）。

```
1 // 获取所有属性(含private, 但不包括继承的)
2 public static void main(String[] args) throws Exception {
3     // 1.获取字节码对象
4     Class<?> clazz =
        Class.forName("com.briup.chap13.bean.HighStudent");
5     System.out.println(clazz);
6
7     System.out.println("-----");
8
9     // 2.获取所有属性(含private, 但不包括继承的)
10    Field[] fields = clazz.getDeclaredFields();
11    for (Field f : fields) {
12        System.out.println(f);
13    }
14
15    System.out.println("-----");
16
17    // 3.获取所有属性(当前类新增, 不包括继承)
18    Field f1 = clazz.getDeclaredField("eScore");
19    System.out.println("f1: " + f1);
20 }
```

运行效果:

```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Servers
<terminated> Test045_Field [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:23:48 - 下午11:23:49)
class com.briup.chap13.bean.HighStudent
-----
private int com.briup.chap13.bean.HighStudent.eScore
public int com.briup.chap13.bean.HighStudent.chinaScore
-----
f1: private int com.briup.chap13.bean.HighStudent.eScore
```

2) 属性获取及设置方法

方法名	说明
void set(Object obj, Object value)	赋值
Object get(Object obj)	获取值

案例展示3:

获取Student类的指定属性对象（含private），并进行属性值修改及获取。

Student类属性如下：

```
Student.java Test045_Field.java
1 package com.briup.chap13.bean;
2
3 public class Student {
4     public String id;
5     private String name;
6     private int age;
7 }
```

测试类代码如下：

```
1 // 使用Field进行 属性值获取|设置
2 public static void main(String[] args) throws Exception {
3     // 1.获取字节码对象
4     Class<?> clazz =
        Class.forName("com.briup.chap13.bean.Student");
```

```

5      System.out.println(clazz);
6
7      // 2.获取指定的public 属性
8      Field f1 = clazz.getField("id");
9
10     // 3.借助属性对象 获取属性值
11     // 属性是依赖对象而存在的，所以：
12     // 注意：通过反射里面属性 来获取 属性值，一定要 依赖普通对象
13     Student s = new Student("003", "tom", 23);
14     String id = (String) f1.get(s);
15     System.out.println("id: " + id);
16
17     // 借助属性对象 设置属性值
18     f1.set(s, "010");
19     System.out.println("after set, s: " + s);
20
21     System.out.println("-----");
22     System.out.println("操作private属性");
23
24     // 4.获取private属性
25     Field f2 = clazz.getDeclaredField("name");
26     // 设置可以访问
27     f2.setAccessible(true);
28
29     // 5.获取private属性值输出，然后修改
30     String name = (String) f2.get(s);
31     System.out.println("name: " + name);
32
33     f2.set(s, "rose");
34
35     // 6.输出测试
36     System.out.println("after set, s: " + s);
37 }

```

运行结果：


```
Problems Javadoc Declaration Console Progress Properties Navigator (Deprecated) Git Staging Git Reflog History Server
<terminated> Test045_Field [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:34:27 - 下午11:34:28)
class com.briup.chap13.bean.Student
id: 003
after set, s: Student [id=010, name=tom, age=23]
-----
操作private属性
name: tom
after set, s: Student [id=010, name=rose, age=23]
```

4.6 成员方法

通过反射可以获取类里面所有的成员（含私有）方法，并调用。

1) Method获取相关方法

方法名	说明
Method[] getMethods()	返回所有公共成员方法对象的数组(包含继承的)
Method[] getDeclaredMethods()	返回所有成员方法对象的数组(含private，不含继承)
Method getMethod(String name, Class<?>... parameterTypes)	返回单个公共成员方法对象(包含继承的)
Method getDeclaredMethod(String name, Class<?>... parameterTypes)	返回单个成员方法对象(含private，不含继承)

2) Method对象调用方法

方法名	说明
Object invoke(Object obj, Object... args)	运行方法

参数一：用obj对象调用该方法

参数二：调用方法的传递的参数(如果没有就不写)

返回值：方法的返回值(如果没有就不写)

案例展示1:

获取Student类的字节码对象，进而遍历输出其所有的Method对象，并调用其public方法。

```

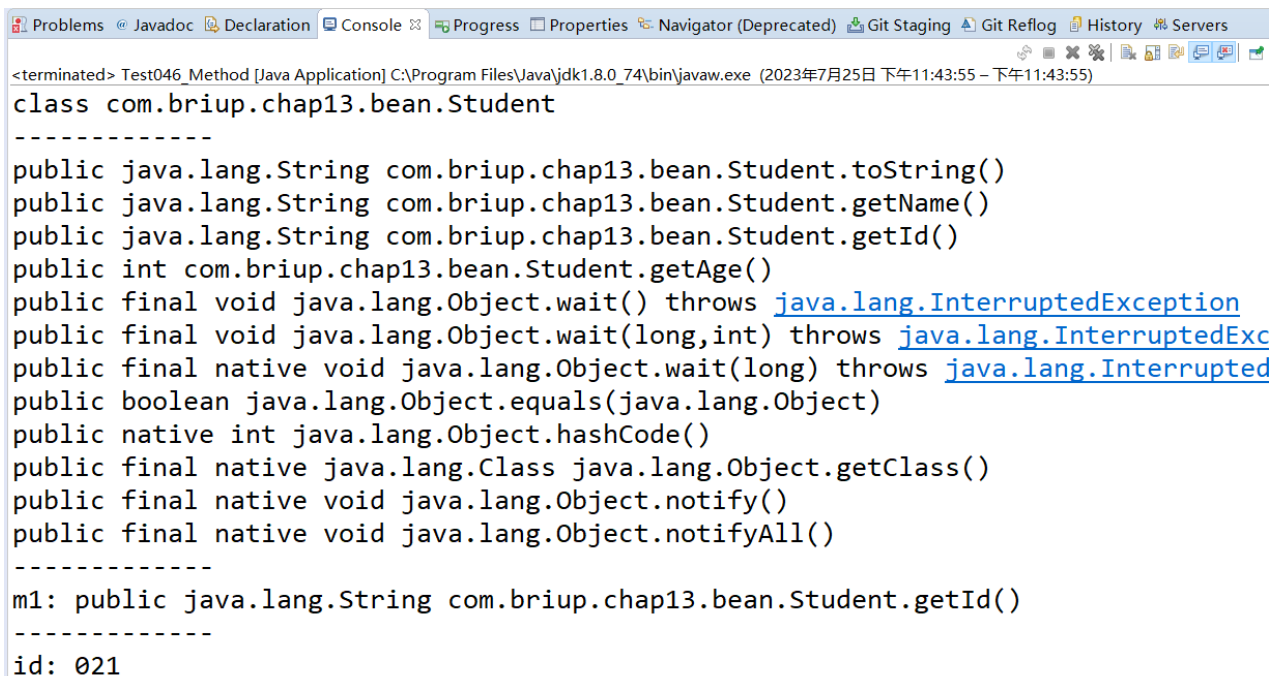
1  package com.briup.chap13.test;
2
3  import java.lang.reflect.Method;
4  import com.briup.chap13.bean.Student;
5
6  public class Test046_Method {
7      //1.获取method对象并输出 进而调用
8      public static void main(String[] args) throws Exception {
9          //1.获取字节码对象
10         Class<?> clazz =
11             Class.forName("com.briup.chap13.bean.Student");
12         System.out.println(clazz);
13
14         System.out.println("-----");
15
16         //2.获取所有public method对象(包含继承的)
17         Method[] methods = clazz.getMethods();
18         for (Method method : methods) {
19             System.out.println(method);
20         }
21     }
22 }
```

```

20
21         System.out.println("-----");
22
23         //3.获取指定public method对象
24         // public Method getMethod(String name, Class<?>...
parameterTypes)
25         Method m1 = clazz.getMethod("getId");
26         System.out.println("m1: " + m1);
27
28         System.out.println("-----");
29
30         //4.使用method对象调用方法
31         // 注意：普通method方法的调用，要依赖对象
32         Student s = new Student("021","tom",21);
33         // 借助 invoke方法执行
34         // public Object invoke(Object obj, Object... args)
35         String id = (String)m1.invoke(s);    //等同于 s.getId();
36         System.out.println("id: " + id);
37     }
38 }

```

运行效果：



```

<terminated> Test046_Method [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:43:55 - 下午11:43:55)
class com.briup.chap13.bean.Student
-----
public java.lang.String com.briup.chap13.bean.Student.toString()
public java.lang.String com.briup.chap13.bean.Student.getName()
public java.lang.String com.briup.chap13.bean.Student.getId()
public int com.briup.chap13.bean.Student.getAge()
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
-----
m1: public java.lang.String com.briup.chap13.bean.Student.getId()
-----
id: 021

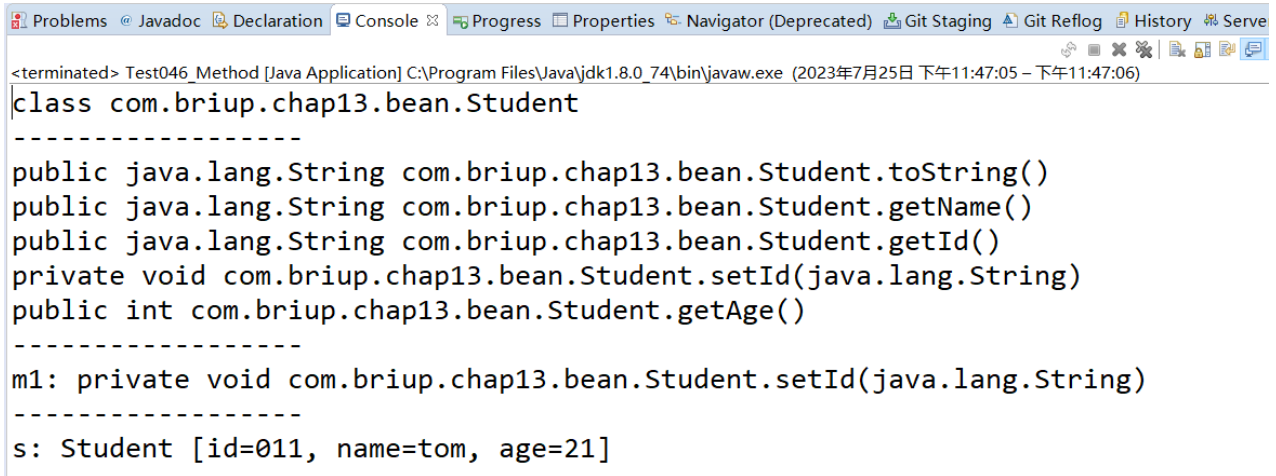
```

案例展示2:

获取Student类的字节码对象，进而获取其指定的Method对象（含private），并调用方法。

```
1  public static void main(String[] args) throws Exception {
2      //1.获取字节码对象
3      Class<?> clazz =
4      Class.forName("com.briup.chap13.bean.Student");
5      System.out.println(clazz);
6
7      System.out.println("-----");
8
9      //2.获取所有method并遍历(含private, 不含继承)
10     Method[] methods = clazz.getDeclaredMethods();
11     for (Method method : methods) {
12         System.out.println(method);
13     }
14
15     System.out.println("-----");
16
17     //3.获取指定private方法
18     Method m1 = clazz.getDeclaredMethod("setId",
19     String.class);
20     System.out.println("m1: " + m1);
21
22     System.out.println("-----");
23
24     //4.通过method调用private方法
25     Student s = new Student("021", "tom", 21);
26     //开放访问权限
27     m1.setAccessible(true);
28     m1.invoke(s, "011");    //等同于 s.setId("011");
29     System.out.println("s: " + s);
30 }
```

运行效果：



```

<terminated> Test046_Method [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:47:05 - 下午11:47:06)
class com.briup.chap13.bean.Student
-----
public java.lang.String com.briup.chap13.bean.Student.toString()
public java.lang.String com.briup.chap13.bean.Student.getName()
public java.lang.String com.briup.chap13.bean.Student.getId()
private void com.briup.chap13.bean.Student.setId(java.lang.String)
public int com.briup.chap13.bean.Student.getAge()
-----
m1: private void com.briup.chap13.bean.Student.setId(java.lang.String)
-----
s: Student [id=011, name=tom, age=21]
  
```

4.7 面试题

- 1 现有一个集合定义如下：
- 2 `List<Integer> list = new ArrayList<>();`
- 3 要求，往list集合中添加元素："hello"、123、3.14
- 4 请编码实现。

源码实现：

```

1 package com.briup.chap13.test;
2
3 import java.lang.reflect.Method;
4 import java.util.ArrayList;
5 import java.util.Iterator;
6 import java.util.List;
7
8 public class Test047_Question {
9     public static void main(String[] args) throws Exception {
10         List<Integer> list = new ArrayList<>();
  
```

```

11
12         //1.获取字节码对象
13         Class<? extends List> clazz = list.getClass();
14
15         //2.获取add方法
16         Method m = clazz.getDeclaredMethod("add",
Object.class);
17
18         //3.设置可以访问，添加元素
19         m.setAccessible(true);
20         m.invoke(list, "hello");
21         m.invoke(list, 123);
22         m.invoke(list, 3.14);
23
24         //4.遍历集合
25         Iterator<Integer> it = list.iterator();
26         while(it.hasNext())
27             System.out.println(it.next());
28     }
29 }

```

运行效果：

```

<terminated> Test047_Question [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:55:08 – 下午11:55:10)
hello
123
3.14

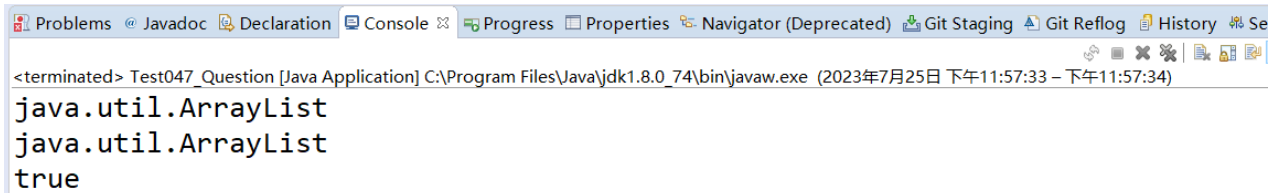
```

注意：泛型只在编译阶段做语法检查，运行期间会被自动忽略

测试案例：

```
1 public static void main(String[] args) {  
2     List<Integer> list1 = new ArrayList<>();  
3     List<String> list2 = new ArrayList<>();  
4  
5     Class<? extends List> clazz1 = list1.getClass();  
6     Class<? extends List> clazz2 = list2.getClass();  
7  
8     System.out.println(clazz1.getName());  
9     System.out.println(clazz2.getName());  
10    System.out.println(clazz1 == clazz2);  
11 }
```

运行效果：



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, Console, Progress, Properties, Navigator (Deprecated), Git Staging, Git Reflog, History, and Search. The console output is as follows:

```
<terminated> Test047_Question [Java Application] C:\Program Files\Java\jdk1.8.0_74\bin\javaw.exe (2023年7月25日 下午11:57:33 - 下午11:57:34)  
java.util.ArrayList  
java.util.ArrayList  
true
```