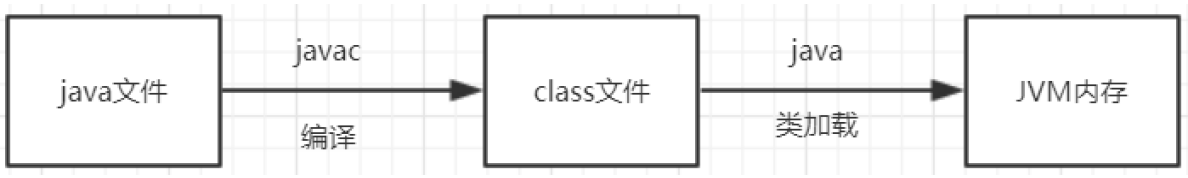
类加载

过程



- 加载
  - 通过类加载(ClassLoader)去加载字节码文件,转化为二进制数据
  - 在这个过程中生成java.lang.Class 对象(这个类所对应的)
- 链接
  - 验证: 对字节码文件正确性的校验(cafe babe)
  - 准备: 为类的静态成员分配内存,赋予默认初始值
    - ```
      static int a = 10;
      ```
  - 解析: 把符号引用(一组符号来描述引用的目标)用替换为直接引用(真实的内存地址)
- 初始化
  - 执行静态代码块的内容,给静态成员分配真实的值

类加载器

分类

Bootstrap ClassLoader 根类加载器
负责Java运行时核心类的加载，JDK中JRE的lib目录下rt.jar

Extension ClassLoader 扩展类加载器
负责JRE的扩展目录中jar包的加载，在JDK中JRE的lib目录下ext目录

Sysetm(App) ClassLoader 系统类加载器/应用加载器
负责加载自己定义的Java类

```
package _24reflect.com.cskaoyan._01introduction;

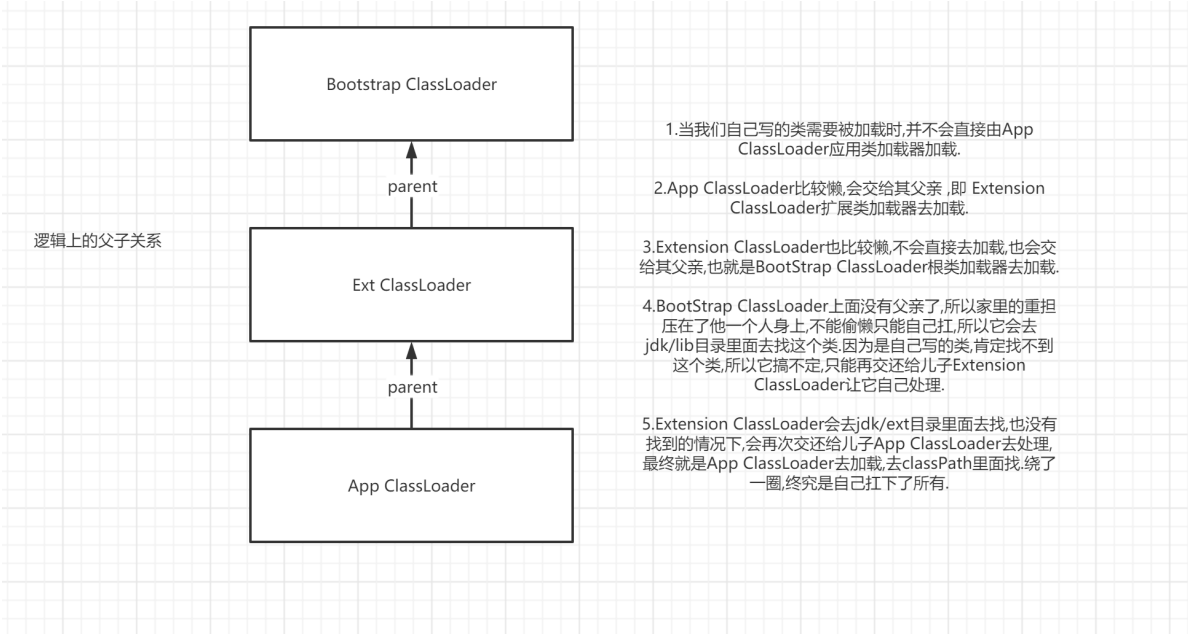
/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 9:55
 **/

public class ClassLoaderTest {
    public static void main(String[] args) {
        // 获取系统类加载器
        ClassLoader systemClassLoader = ClassLoader.getSystemClassLoader();
```
```

```
        System.out.println(systemClassLoader);
        // 查看加载的路径
        System.out.println(System.getProperty("java.class.path").
                replace(";", System.lineSeparator()));

        System.out.println("--------------------");
        // 获取扩展类加载器
        ClassLoader parent = systemClassLoader.getParent();
        System.out.println(parent);
        // java.ext.dirs
        System.out.println(System.getProperty("java.ext.dirs").
                replace(";", System.lineSeparator()));
        // 获取根类加载器 null 不是java写的
        ClassLoader parent1 = parent.getParent();
        System.out.println(parent1);
    }
}
```

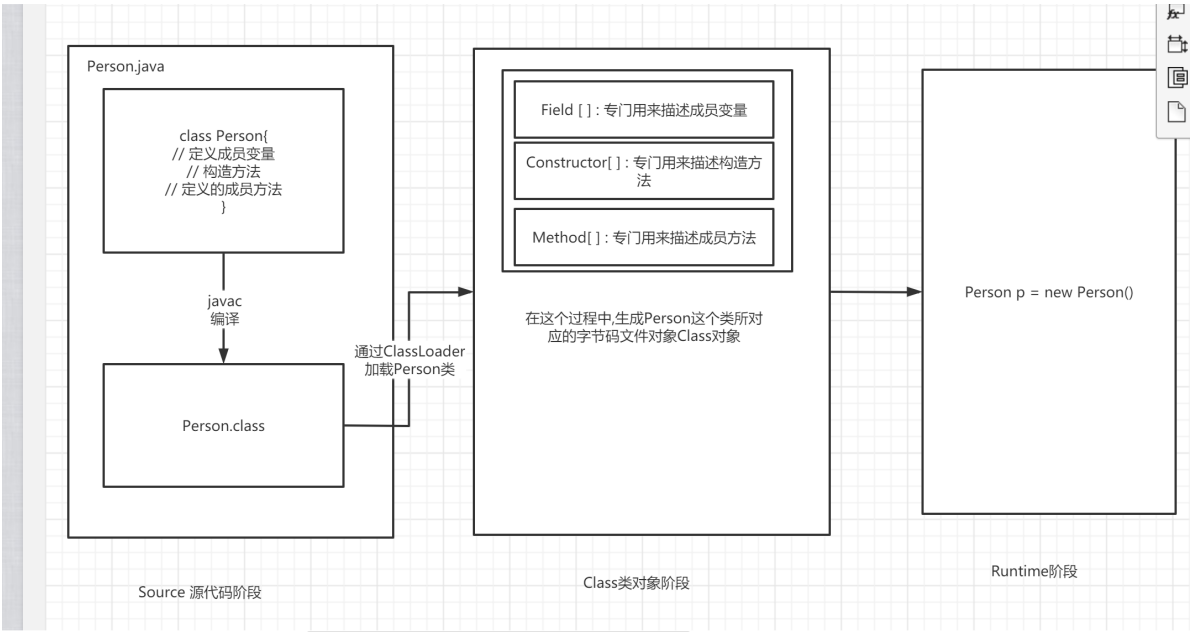
双亲委派模型



类加载时机

类加载时机 （类初始化时机）
创建类的实例(首次创建该类对象)
访问类的静态变量(首次)
调用类的静态方法(首次)
使用反射方式来强制创建某个类或接口对应的java.lang.Class对象
加载某个类的子类，会先触发父类的加载
直接使用java.exe命令来运行某个主类，也就是执行了某个类的main()方法

java代码的3个阶段



反射

什么是反射

获取运行时类信息的一种手段

反射技术的起点就是获取字节码文件对象

获取字节码文件对象的几种方式

- 对象.getClass()
- 类名.class
- Class.forName("全限定名")
- ClassLoader.loadClass(全类名)

Demo

```
package _24reflect.com.cskaoyan._02cls;

/**
 * @description: 获取字节码文件对象
 * @author: 景天
 * @date: 2022/8/1 10:56
 **/

public class Demo {
    public static void main(String[] args) throws ClassNotFoundException {
        // - 对象.getClass()
        Stu stu = new Stu();
        Class<? extends Stu> c1 = stu.getClass();
        //- 类名.class
        Class<Stu> c2 = Stu.class;
        System.out.println(c1==c2);

        //- Class.forName("全限定名")
        Class<?> c3 = Class.forName("_24reflect.com.cskaoyan._02cls.Stu");
        System.out.println(c1 == c3);
```

```
        //- ClassLoader.loadClass(全类名)
        ClassLoader systemClassLoader = ClassLoader.getSystemClassLoader();
        Class<?> c4 =
systemClassLoader.loadClass("_24reflect.com.cskaoyan._02cls.Stu");
        System.out.println(c1 == c4);
    }
}

class Stu{

}
```

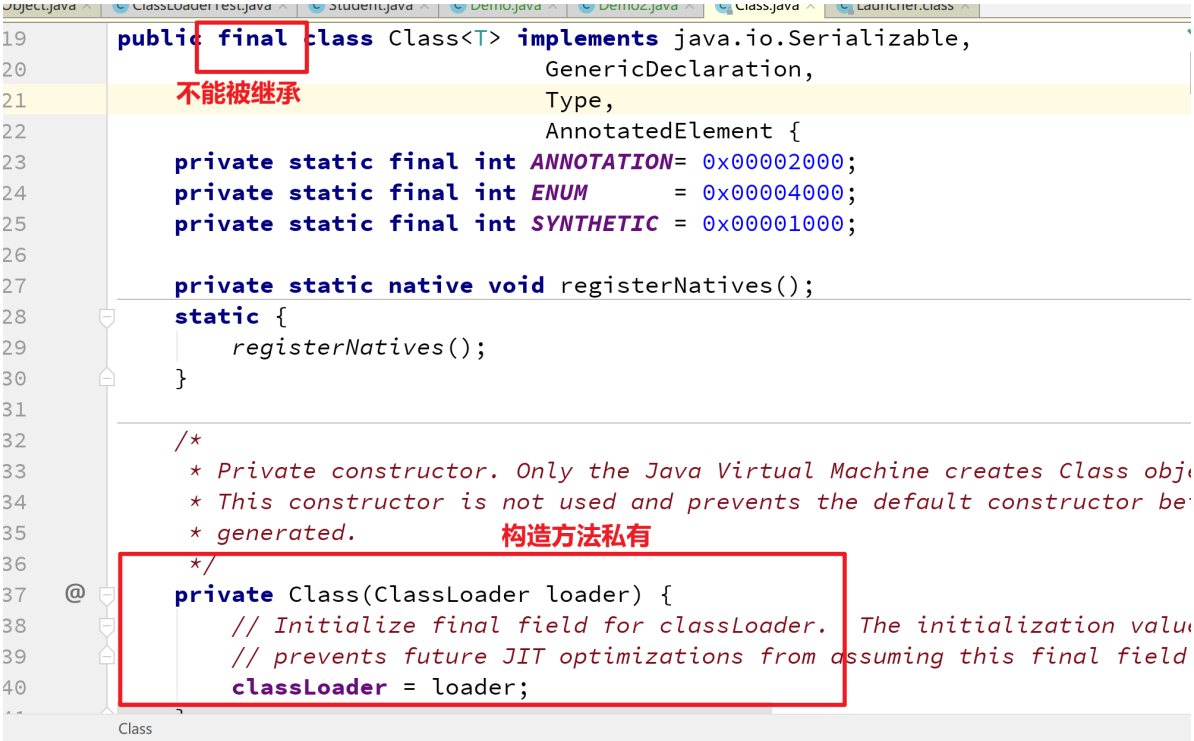
注意:

无论通过哪种方式获取到的字节码文件对象都是同一个

关于Class

`Class` 类的实例表示正在运行的 Java 应用程序中的类和接口

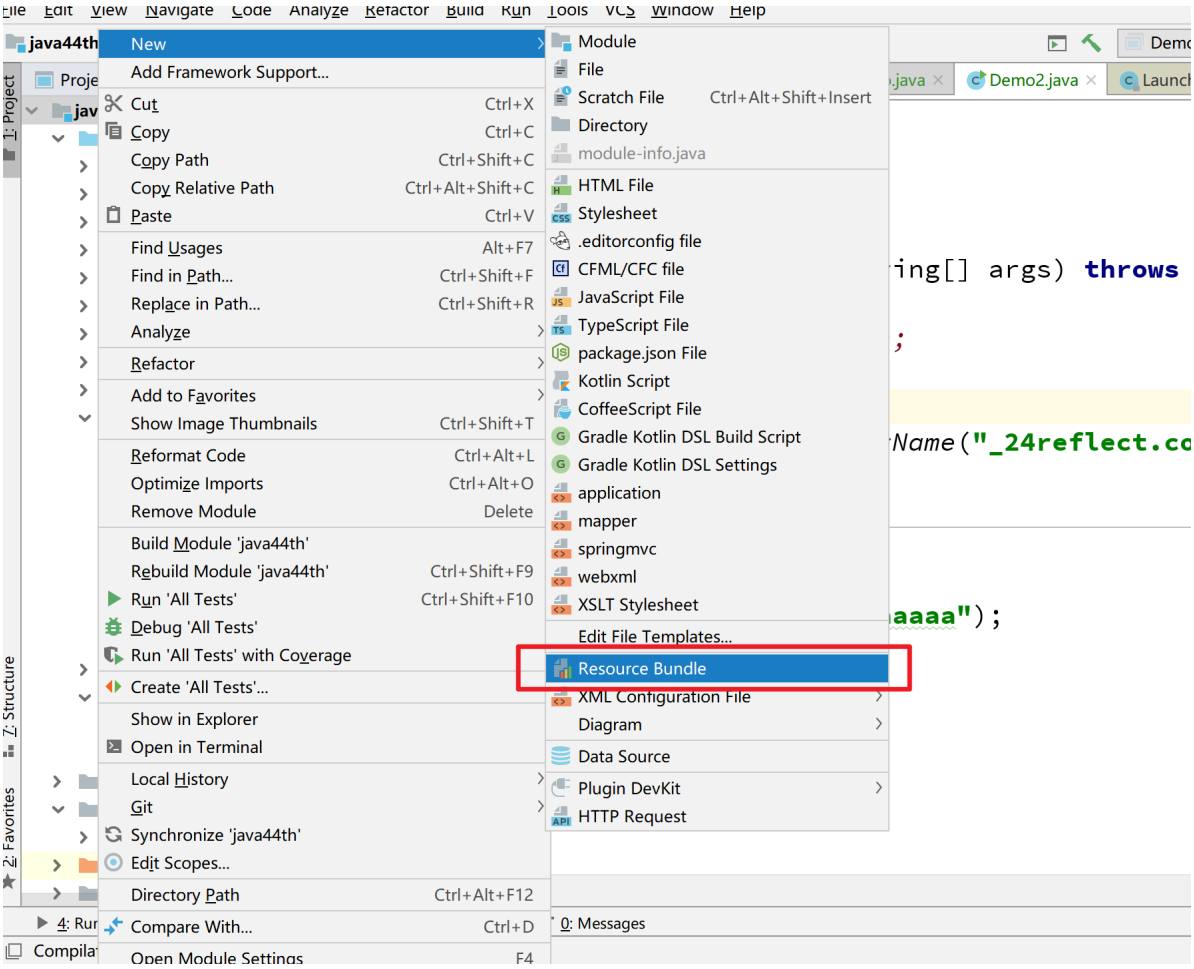
`Class` 没有公共构造方法。`Class` 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的 `defineClass` 方法自动构造的。



配置文件(.properties)

配置文件 .properties .xml .yml 一般放配置信息 比如数据库配置信息 比如第三方的服务信息

开发流程 : 本地 测试 线上



Properties

`Properties` 类表示了一个持久的属性集

构造方法

Properties()　　创建一个无默认值的空属性列表。

成员方法

void	load(InputStream inStream) 从输入流中读取属性列表（键和元素对）。
void	load(Reader reader) 按简单的面向行的格式从输入字符流中读取属性列表（键和元素对）。

String	getProperty(String key) 用指定的键在此属性列表中搜索属性。

Demo

```
package _24reflect.com.cskaoyan._03config;

import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.util.Properties;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 11:31
 **/

public class Demo {
    public static void main(String[] args) throws IOException {
        // 创建Properties对象
        Properties properties = new Properties();
        //load
        properties.load(new FileInputStream("config.properties"));

        // getProperty获取属性值
        String host = properties.getProperty("host");
        String username = properties.getProperty("username");
        System.out.println(host);
        System.out.println(username);
    }
}
```

通过类加载器加载

```
package _24reflect.com.cskaoyan._03config;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Properties;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 11:31
 **/

public class Demo2 {
    public static void main(String[] args) throws IOException {
        // 创建Properties对象
        Properties properties = new Properties();
        // 通过类加载器
        URL systemResource = ClassLoader.getSystemResource("");
        System.out.println(systemResource);
        InputStream in =
ClassLoader.getSystemResourceAsStream("config.properties");
        properties.load(in);


        //load
        //properties.load(new FileInputStream("config.properties"));
```

```
        // getProperty获取属性值
        String host = properties.getProperty("host");
        String username = properties.getProperty("username");
        System.out.println(host);
        System.out.println(username);
    }
}
```

通过反射获取构造方法(Constructor)

通过反射获取所有构造方法

```
Constructor[] getConstructors()
Constructor[] getDeclaredConstructors()
```

获取指定构造方法

```
Constructor<T> getConstructor(Class<?>... parameterTypes)
Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)
```

使用Constructor创建对象

```
newInstance(参数列表)
```

异常

java.lang.IllegalAccessException

```
package _24reflect.com.cskaoyan._04api;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 14:45
 **/

public class ConstructorTest {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, IllegalAccessException, InvocationTargetException,
InstantiationException {
        // 反射技术的起点
        // 获取字节码文件对象
        // 获取所有的构造方法
        Class<?> personCls =
Class.forName("_24reflect.com.cskaoyan.bean.Person");
        System.out.println("获取所有的public构造方法--------");
```

```
        // Constructor[] getConstructors()
        Constructor<?>[] constructors = personCls.getConstructors();
        for (Constructor constructor : constructors) {
            System.out.println(constructor);
        }

        System.out.println("获取所有的构造方法--------");

        //Constructor[] getDeclaredConstructors()
        Constructor<?>[] declaredConstructors =
personCls.getDeclaredConstructors();
        for (Constructor constructor : declaredConstructors) {
            System.out.println(constructor);
        }
        System.out.println("获取指定的public构造方法--------");

        // Constructor<T> getConstructor(Class<?>... parameterTypes)
        Constructor<?> constructor = personCls.getConstructor(String.class,
int.class, boolean.class);
        System.out.println(constructor);
        System.out.println("获取指定的构造方法--------");

        //Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)
        Constructor<?> declaredConstructor =
personCls.getDeclaredConstructor(String.class, int.class);
        System.out.println(declaredConstructor);

        // 利用构造方法对象 实例化对象  newInstance
        Object o = constructor.newInstance("zs", 20, true);
        System.out.println(o);

        // 暴力破解方式
        // 忽略java语法检查 setAccessible(true)
        declaredConstructor.setAccessible(true);
        Object o1 = declaredConstructor.newInstance("ls", 21);
        System.out.println(o1);

    }
}
```

通过反射获取成员变量(Field)

通过反射获取所有成员变量

```
Field[] getFields()
Field[] getDeclaredFields()
```

获取指定成员变量

```
Field getField(String name)
Field getDeclaredField(String name)
```

通过Field读写对象的成员变量(可暴力破解)

```
Object get(Object obj)：获取值，传入对象
void set(Object obj, Object value)：赋值，传入对象
```

```
package _24reflect.com.cskaoyan._04api;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 15:01
 **/

public class FieldTest {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchFieldException, NoSuchMethodException, IllegalAccessException,
InvocationTargetException, InstantiationException {
        // 获取字节码文件对象
        Class<?> personCls =
Class.forName("_24reflect.com.cskaoyan.bean.Person");
        // 获取所有的成员变量
        System.out.println("获取所有的public成员变量");
        // Field[] getFields()
        Field[] fields = personCls.getFields();
        for (Field field : fields) {
            System.out.println(field);
        }
        System.out.println("获取所有的成员变量");

        //Field[] getDeclaredFields()
        Field[] declaredFields = personCls.getDeclaredFields();
        for (Field field : declaredFields) {
            System.out.println(field);
        }
        System.out.println("获取指定的public的成员变量");

        // Field getField(String name)
        Field nameField = personCls.getField("name");
        System.out.println(nameField);
        System.out.println("获取指定的的成员变量");

        //Field getDeclaredField(String name)
        Field ageField = personCls.getDeclaredField("age");
        System.out.println(ageField);

        //void set(Object obj, Object value)：赋值，传入对象
        // 获取构造方法
        Constructor<?> declaredConstructor = personCls.getDeclaredConstructor();
        Object o = declaredConstructor.newInstance();
        nameField.set(o, "长风");
```

```
        System.out.println(o);
        // Object get(Object obj)：获取值，传入对象
        Object o1 = nameField.get(o);
        System.out.println(o1);
    }
}
```

通过反射获取成员方法(Method)

获取所有成员方法

```
Method[] getMethods()
Method[] getDeclaredMethods()
```

获取指定的成员方法

```
Method getMethod(String name, Class<?>... parameterTypes)
Method getDeclaredMethod(String name, Class<?>... parameterTypes)
```

利用Method调用对象的方法

```
Object invoke(Object obj, Object... args)
```

```
package _24reflect.com.cskaoyan._04api;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 15:14
 **/

public class MethodTest {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, IllegalAccessException, InvocationTargetException,
InstantiationException {
        // 获取字节码文件对象
        Class<?> personCls =
Class.forName("_24reflect.com.cskaoyan.bean.Person");

        // 获取所有的public的成员方法
        System.out.println("获取所有的public的成员方法");
        // Method[] getMethods() 会获取到父类的public的方法
        Method[] methods = personCls.getMethods();
        for (Method method : methods) {
            System.out.println(method);
        }
```

```
        System.out.println("获取所有的成员方法");

        //Method[] getDeclaredMethods()
        Method[] declaredMethods = personCls.getDeclaredMethods();
        for (Method method : declaredMethods) {
            System.out.println(method);
        }
        System.out.println("获取指定的public的成员方法");

        // Method getMethod(String name, Class<?>... parameterTypes)
        Method eatMethod1 = personCls.getMethod("eat");
        System.out.println(eatMethod1);
        //Method getDeclaredMethod(String name, Class<?>... parameterTypes)
        System.out.println("获取指定的成员方法");

        Method eatMethod2 = personCls.getDeclaredMethod("eat", String.class);
        System.out.println(eatMethod2);

        // 对象.方法名()
        // 调用方法
        // Object invoke(Object obj, Object... args)
        // 获取构造方法
        Constructor<?> declaredConstructor = personCls.getDeclaredConstructor();
        // 创建对象
        Object o = declaredConstructor.newInstance();
        eatMethod1.invoke(o);
        // 忽略java语法检查
        eatMethod2.setAccessible(true);
        eatMethod2.invoke(o, "apple");


    }
}
```

补充

其他API

通过字节码文件对象直接实例化

```
package _24reflect.com.cskaoyan._05add;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 15:57
 **/

public class Demo {
    public static void main(String[] args) throws Exception{
```

```
        // 获取字节码文件对象
        Class<?> c = Class.forName("_24reflect.com.cskaoyan._05add.A");
        // 直接实例化
        Object o = c.newInstance();
        System.out.println(o);
    }
}

class A{
    // 必须要有默认的无参构造方法
    int a;

    public A(int a) {
        this.a = a;
    }

    public A() {
    }
}
```

```
package _24reflect.com.cskaoyan._05add;

import _24reflect.com.cskaoyan.bean.Person;

import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.util.Arrays;

/**
 * @description: 补充API
 * @author: 景天
 * @date: 2022/8/1 16:00
 **/

public class Demo2 {
    public static void main(String[] args) throws Exception{
        // 获取字节码文件对象
        Class<?> c = Class.forName("_24reflect.com.cskaoyan.bean.Person");
        //Class<?> c = Class.forName("java.io.OutputStream");
        // 获取类名
        String name = c.getName();
        System.out.println(name);
        // 获取简单名
        String simpleName = c.getSimpleName();
        System.out.println(simpleName);
        // 获取父类
        Class<?> superclass = c.getSuperclass();
        System.out.println(superclass.getSimpleName());
        // 获取接口
        Class<?>[] interfaces = c.getInterfaces();
        for (Class i : interfaces) {
            System.out.println(i);
```

```
        }
        // 获取nameField
        Field nameField = c.getDeclaredField("name");
        // 获取权限修饰符
        int modifiers = nameField.getModifiers();
        System.out.println(modifiers);
        String s = Modifier.toString(modifiers);
        System.out.println(s);
        // 获取成员变量的类型
        Class<?> type = nameField.getType();
        System.out.println(type.getSimpleName());
        // 获取方法对象
        Method eatMethod = c.getDeclaredMethod("eat", String.class);
        Class<?> returnType = eatMethod.getReturnType();
        System.out.println(returnType);
        Class<?>[] parameterTypes = eatMethod.getParameterTypes();
        System.out.println(Arrays.toString(parameterTypes));

    }
}
```

自定义类加载器

步骤

- 继承ClassLoader
- 重写findClass方法

findClass的返回值就是要找的Class对象。

在findClass方法中调用defineClass方法，defineClass方法的参数

```
protected Class<?> loadClass(String name,boolean resolve){
    synchronized{
        Class<?> c=findLoadedClass(name);
        if(c==null){
            if(parent!=null){
               c = parent.loadClass(name,false);
            }else{
                c=findBootstrapClassOrNull(name);
            }

            if(c==null){
                c=findClass(name);
            }
        }

    }
}
```

```java
package _24reflect.com.cskaoyan._05add;

import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * @description: 自定义类加载器
 * @author: 景天
 * @date: 2022/8/1 16:18
 **/

/*
- 继承ClassLoader
- 重写findClass方法
 */
public class MyClassLoader extends ClassLoader{
    // 字节码文件路径
    String classPath;

    public MyClassLoader(String classPath) {
        this.classPath = classPath;
    }
    // 重写findClass方法

    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        // String name 类名字

        // 读取class文件,把数据填充到数组中
        Class<?> aClass = null;
        try {
            byte[] data = getData();
            // protected  Class<?> defineClass(String name, byte[] b, int off,
int len)
            // 将一个 byte 数组转换为 Class 类的实例。
            aClass = defineClass(name, data, 0, data.length);
            // 最终返回Class对象
        } catch (IOException e) {
            e.printStackTrace();
        }

        return aClass;
    }

    private byte[] getData() throws IOException {
        // 读取指定路径的class文件
        // 创建输入流
        FileInputStream in = new FileInputStream(classPath);
        // 需要借助于一个ByteArrayOutputStream
        // 创建一个输出流
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        // read
        byte[] bytes = new byte[1024];
```

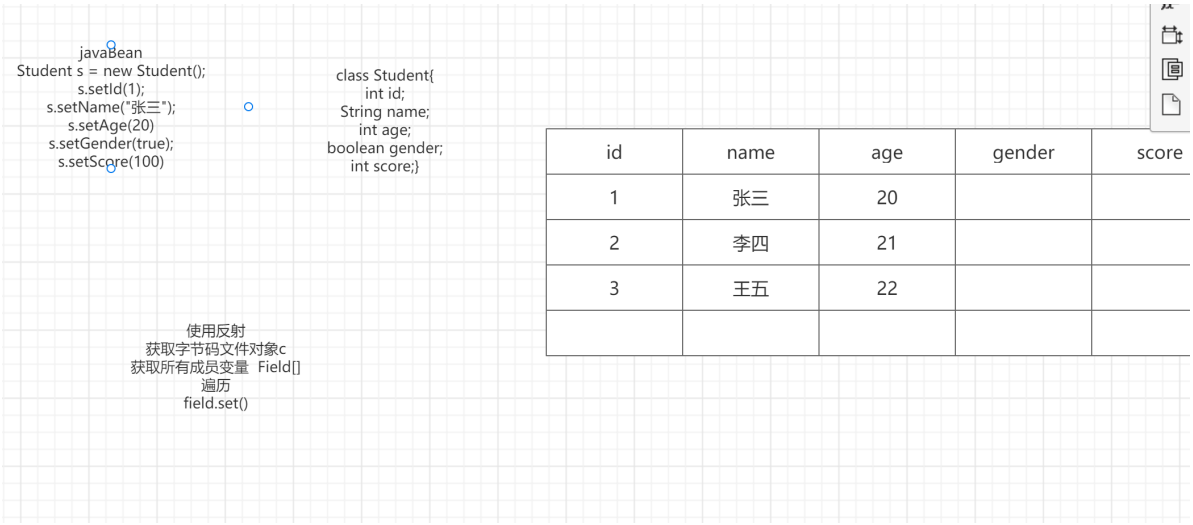
```
        // 边读边写
        int readCount;
        while ((readCount = in.read(bytes)) != -1) {
            out.write(bytes, 0, readCount);
        }
        // 获取字节数组
        //  byte[] toByteArray()
        // 创建一个新分配的 byte 数组。
        byte[] bytes1 = out.toByteArray();

        return bytes1;
    }
}


package _24reflect.com.cskaoyan._05add;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

/**
 * @description:
 * @author: 景天
 * @date: 2022/8/1 16:34
 **/

public class MyClassLoaderTest {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, IllegalAccessException, InstantiationException,
InvocationTargetException {
        // 定义加载路径
        String classPath = "D:\\Log.class";
        // 创建自定义的类加载器
        MyClassLoader myClassLoader = new MyClassLoader(classPath);
        // 加载Log类
        Class<?> logCls = myClassLoader.loadClass("Log");

        // 获取类加载器
        ClassLoader classLoader = logCls.getClassLoader();
        System.out.println(classLoader);

        // 获取方法对象
        Method method = logCls.getDeclaredMethod("func");

        // 执行里面的func方法
        Object o = logCls.newInstance();
        // invoke
        method.invoke(o);
    }
}
```

反射应用场景

通过反射获取注解信息

动态代理

ORM(Object Relational Mapping)框架



id	name	age	gender	score
1	张三	20		
2	李四	21		
3	王五	22		