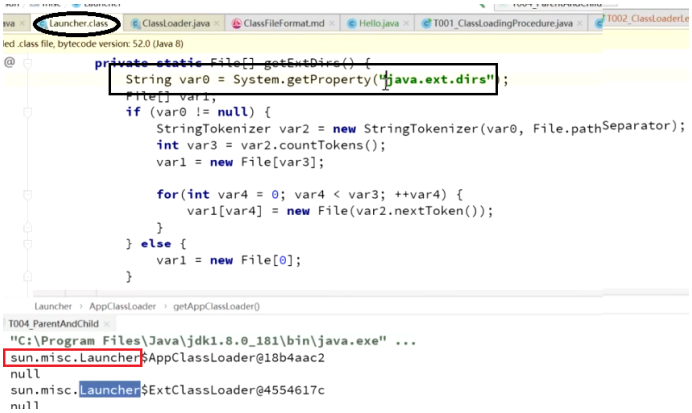
ClassLoader原理和自定义类加载器



不同类加载器 中加载的路径是在Launcher中 已经设置好的

```
/**
 *  可以通过该程序找到 不同加载器对应的目录
 */
public class T003_ClassLoaderScope {
    public static void main(String[] args) {
        String pathBoot = System.getProperty("sun.boot.class.path");
        System.out.println(pathBoot.replaceAll(";", System.lineSeparator()));

        System.out.println("--------------------");
        String pathExt = System.getProperty("java.ext.dirs");
        System.out.println(pathExt.replaceAll(";", System.lineSeparator()));

        System.out.println("--------------------");
        String pathApp = System.getProperty("java.class.path");
        System.out.println(pathApp.replaceAll(";", System.lineSeparator()));
    }
}
```

手动load一个类

```
public class T005_LoadClassByHand {
    public static void main(String[] args) throws ClassNotFoundException {
        // 拿到当前类的ClassLoader  即 APP ClassLoader    然后调用 loadClass ('类的全名')
        Class clazz = T005_LoadClassByHand.class.getClassLoader().loadClass("com.chase.jvm.c2_classloader.T002_ClassLoaderLevel");
        System.out.println(clazz.getName());

        //利用类加载器加载资源
        //T005_LoadClassByHand.class.getClassLoader().getResourceAsStream("");
    }
}
```

ClassLoader loadClass 源码

```
public abstract class ClassLoader {
  private final ClassLoader parent;
  protected Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException{
        synchronized (getClassLoadingLock(name)) {
            // First, check if the class has already been loaded
```

```
            Class<?> c = findLoadedClass(name);
            if (c == null) {    // 当前类没有load 到
                long t0 = System.nanoTime();
                try {
                    if (parent != null) {
                        c = parent.loadClass(name, false);   // 如果当前没有load 到 则从父加载器load  递归过程
                    } else {
                        c = findBootstrapClassOrNull(name);
                    }
                } catch (ClassNotFoundException e) {
                }

                if (c == null) {
                    long t1 = System.nanoTime();
                    c = findClass(name);         // 如果所有的加载器都没有缓存到，则自己加载    递归  找到直接返回
                    // this is the defining class loader; record the stats
                    sun.misc.PerfCounter.getParentDelegationTime().addTime(t1 - t0);
                    sun.misc.PerfCounter.getFindClassTime().addElapsedTimeFrom(t1);
                    sun.misc.PerfCounter.getFindClasses().increment();
                }
            }
            if (resolve) {
                resolveClass(c);
            }
            return c;
        }
    }
}

//   findClass 的源码
protected Class<?> findClass(String name) throws ClassNotFoundException {
    throw new ClassNotFoundException(name);
}

// 默认没有加载到 class 时会报错
// 所以自定义类加载器的时候只需要重写这个方法就好
//  这个模式  就是模板方法模式----------设计模式（模板方法模式）---逻辑都定义好了  部分需要自己实现

// 面试 模板方法的应用场景    ClassLoader 的loadClass 过程 --- 自定义加载器的时候
```

自定义类加载器

```
// 一般用于加载指定位置的Class的情况   是class文件  不是java文件
// 继承 ClassLoader
public class T006_MSBClassLoader extends ClassLoader {
    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        File f = new File("c:/test/", name.replace(".", "/").concat(".class"));  // 读到Class 文件
        try {
            FileInputStream fis = new FileInputStream(f);
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            int b = 0;

            while ((b=fis.read()) !=0) {
                baos.write(b);
            }
            byte[] bytes = baos.toByteArray();
            baos.close();
            fis.close();//可以写的更加严谨

            // 将 字节数组转换成 class 的类对象   并返回该对象
            return defineClass(name, bytes, 0, bytes.length);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return super.findClass(name); //throws ClassNotFoundException   抛异常
    }
    public static void main(String[] args) throws Exception {
        ClassLoader l = new T006_MSBClassLoader();
        Class clazz = l.loadClass("com.mashibing.jvm.Hello");
        Class clazz1 = l.loadClass("com.mashibing.jvm.Hello");
        System.out.println(clazz == clazz1);
```

```
        Hello h = (Hello)clazz.newInstance();
        h.m();
        System.out.println(l.getClass().getClassLoader());
        System.out.println(l.getParent());
        System.out.println(getSystemClassLoader());
    }
}
```

自定义类加载器的时候，parent 怎么指定？

```
public class T010_Parent {
    private static T006_MSBClassLoader parent = new T006_MSBClassLoader();
    private static class MyLoader extends ClassLoader {
        // 重写构造方法，传入指定的parent
        public MyLoader() {
            super(parent);
        }
    }
}
```

自定义的ClassLoader 默认的classloader是那个？

```
// 调用父类的无参构造
protected ClassLoader() {
    this(checkCreateClassLoader(), getSystemClassLoader());
}

// getSystemClassLoader()   通过该行代码 可以获得系统的ClassLoader
```

如果要打破双亲委派机制怎么办？—-不用系统的委派机制

由于委派机制是在loadClass中的 super.loadClass 中进行的

所以删除 super这个逻辑就行，即重写 loadClass方法

自定义加载器：重写findClass ——只是修改了加载class的逻辑

打破双亲委派机制：重写loadClass——不进行委派机制 删除super

打破双亲委派的示例

```
public class T012_ClassReloading2 {
    private static class MyLoader extends ClassLoader {
        @Override
        public Class<?> loadClass(String name) throws ClassNotFoundException {
            File f = new File("C:/work/ijprojects/JVM/out/production/JVM/" + name.replace(".", "/").concat(".class"));
            if(!f.exists()) return super.loadClass(name);
            try {
                InputStream is = new FileInputStream(f);
                byte[] b = new byte[is.available()];
                is.read(b);
                return defineClass(name, b, 0, b.length);
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            return super.loadClass(name);
        }
    }

    public static void main(String[] args) throws Exception {
        MyLoader m = new MyLoader();
        Class clazz = m.loadClass("com.chase.jvm.Hello");
        m = new MyLoader();
        Class clazzNew = m.loadClass("com.chase.jvm.Hello");
        System.out.println(clazz == clazzNew);
    }
}
```