

1.类加载器

1.1类加载【理解】

- 类加载的描述
 - 当程序要使用某个类时，如果该类还未被加载到内存中，则系统会通过类的加载，类的连接，类的初始化这三个步骤来对类进行初始化。如果不出现意外情况，JVM将会连续完成这三个步骤，所以有时也把这三个步骤统称为类加载或者类初始化
- 类的加载
 - 就是指将class文件读入内存，并为之创建一个 java.lang.Class 对象
 - 任何类被使用时，系统都会为之建立一个 java.lang.Class 对象
- 类的连接
 - 验证阶段：用于检验被加载的类是否有正确的内部结构，并和其他类协调一致
 - 准备阶段：负责为类的类变量分配内存，并设置默认初始化值
 - 解析阶段：将类的二进制数据中的符号引用替换为直接引用
- 类的初始化
 - 在该阶段，主要就是对类变量进行初始化
- 类的初始化步骤
 - 假如类还未被加载和连接，则程序先加载并连接该类
 - 假如该类的直接父类还未被初始化，则先初始化其直接父类
 - 假如类中有初始化语句，则系统依次执行这些初始化语句
 - 注意：在执行第2个步骤的时候，系统对直接父类的初始化步骤也遵循初始化步骤1-3
- 类的初始化时机
 - 创建类的实例
 - 调用类的类方法
 - 访问类或者接口的类变量，或者为该变量赋值
 - 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
 - 初始化某个类的子类
 - 直接使用java.exe命令来运行某个主类

1.2类加载器【理解】

1.2.1类加载器的作用

- 负责将.class文件加载到内存中，并为之生成对应的 java.lang.Class 对象。虽然我们不用过分关心类加载机制，但是了解这个机制我们就能更好的理解程序的运行！

1.2.2JVM的类加载机制

- 全盘负责：就是当一个类加载器负责加载某个Class时，该Class所依赖的和引用的其他Class也将由该类加载器负责载入，除非显示使用另外一个类加载器来载入
- 父类委托：就是当一个类加载器负责加载某个Class时，先让父类加载器试图加载该Class，只有在父类加载器无法加载该类时才尝试从自己的类路径中加载该类
- 缓存机制：保证所有加载过的Class都会被缓存，当程序需要使用某个Class对象时，类加载器先从缓存区中搜索该Class，只有当缓存区中不存在该Class对象时，系统才会读取该类对应的二进制数据，并将其转换成

Class对象，存储到缓存区

1.2.3Java中的内置类加载器

- Bootstrap class loader：它是虚拟机的内置类加载器，通常表示为null，并且没有父null
- Platform class loader：平台类加载器可以看到所有平台类，平台类包括由平台类加载器或其祖先定义的Java SE平台API，其实现类和JDK特定的运行时类
- System class loader：它也被称为应用程序类加载器，与平台类加载器不同。系统类加载器通常用于定义应用程序类路径，模块路径和JDK特定工具上的类
- 类加载器的继承关系：System的父加载器为Platform，而Platform的父加载器为Bootstrap

1.2.4ClassLoader 中的两个方法

- 方法分类

方法名	说明
static ClassLoader getSystemClassLoader()	返回用于委派的系统类加载器
ClassLoader getParent()	返回父类加载器进行委派

- 示例代码

```
1 public class ClassLoaderDemo {
2     public static void main(String[] args) {
3         //static ClassLoader getSystemClassLoader() : 返回用于委派的系统类加载器
4         ClassLoader c = ClassLoader.getSystemClassLoader();
5         System.out.println(c); //AppClassLoader
6
7         //ClassLoader getParent() : 返回父类加载器进行委派
8         ClassLoader c2 = c.getParent();
9         System.out.println(c2); //PlatformClassLoader
10
11         ClassLoader c3 = c2.getParent();
12         System.out.println(c3); //null
13     }
14 }
```

2.反射

2.1反射的概述【理解】

- 是指在运行时去获取一个类的变量和方法信息。然后通过获取到的信息来创建对象，调用方法的一种机制。由于这种动态性，可以极大的增强程序的灵活性，程序不用在编译期就完成确定，在运行期仍然可以扩展

2.2获取Class类对象的三种方式【应用】

2.2.1三种方式分类

- 类名.class属性
- 对象名.getClass()方法

- Class.forName(全类名)方法

2.2.2示例代码

```

1 public class ReflectDemo {
2     public static void main(String[] args) throws ClassNotFoundException {
3         //使用类的class属性来获取该类对应的Class对象
4         Class<Student> c1 = Student.class;
5         System.out.println(c1);
6
7         Class<Student> c2 = Student.class;
8         System.out.println(c1 == c2);
9         System.out.println("-----");
10
11        //调用对象的getClass()方法，返回该对象所属类对应的Class对象
12        Student s = new Student();
13        Class<? extends Student> c3 = s.getClass();
14        System.out.println(c1 == c3);
15        System.out.println("-----");
16
17        //使用Class类中的静态方法forName(String className)
18        Class<?> c4 = Class.forName("com.itheima_02.Student");
19        System.out.println(c1 == c4);
20    }
21 }

```

2.3反射获取构造方法并使用【应用】

2.3.1Class类获取构造方法对象的方法

- 方法分类

方法名	说明
Constructor<?>[] getConstructors()	返回所有公共构造方法对象的数组
Constructor<?>[] getDeclaredConstructors()	返回所有构造方法对象的数组
Constructor getConstructor(Class<?>... parameterTypes)	返回单个公共构造方法对象
Constructor getDeclaredConstructor(Class<?>... parameterTypes)	返回单个构造方法对象

- 示例代码

```

1 public class ReflectDemo01 {
2     public static void main(String[] args) throws ClassNotFoundException,
3         NoSuchMethodException, IllegalAccessException, InvocationTargetException,
4         InstantiationException {
5         //获取Class对象
6         Class<?> c = Class.forName("com.itheima_02.Student");
7     }
8 }

```

```
5
6      //Constructor<?>[] getConstructors() 返回一个包含 Constructor对象的数组，
   Constructor对象反映了由该 Class对象表示的类的所有公共构造函数
7  //      Constructor<?>[] cons = c.getConstructors();
8      //Constructor<?>[] getDeclaredConstructors() 返回反映由该 Class对象表示的类
   声明的所有构造函数的 Constructor对象的数组
9      Constructor<?>[] cons = c.getDeclaredConstructors();
10     for(Constructor con : cons) {
11         System.out.println(con);
12     }
13     System.out.println("-----");
14
15     //Constructor<T> getConstructor(Class<?>... parameterTypes) 返回一个
   Constructor对象，该对象反映由该 Class对象表示的类的指定公共构造函数
16     //Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes) 返回
   一个 Constructor对象，该对象反映由此 Class对象表示的类或接口的指定构造函数
17     //参数：你要获取的构造方法的参数的个数和数据类型对应的字节码文件对象
18
19     Constructor<?> con = c.getConstructor();
20
21     //Constructor提供了一个类的单个构造函数的信息和访问权限
22     //T newInstance(Object... initargs) 使用由此 Constructor对象表示的构造函数，
   使用指定的初始化参数来创建和初始化构造函数的声明类的新实例
23     Object obj = con.newInstance();
24     System.out.println(obj);
25
26     //      Student s = new Student();
27     //      System.out.println(s);
28 }
29 }
```

2.3.2Constructor类用于创建对象的方法

方法名	说明
T newInstance(Object...initargs)	根据指定的构造方法创建对象

2.4反射获取构造方法并使用练习1【应用】

- 案例需求
 - 通过反射获取公共的构造方法并创建对象
- 代码实现
 - 学生类

```
1 public class Student {
2     //成员变量：一个私有，一个默认，一个公共
3     private String name;
4     int age;
5     public String address;
6
7     //构造方法：一个私有，一个默认，两个公共
```

```

8      public Student() {
9      }
10
11     private Student(String name) {
12         this.name = name;
13     }
14
15     Student(String name, int age) {
16         this.name = name;
17         this.age = age;
18     }
19
20     public Student(String name, int age, String address) {
21         this.name = name;
22         this.age = age;
23         this.address = address;
24     }
25
26     //成员方法：一个私有，四个公共
27     private void function() {
28         System.out.println("function");
29     }
30
31     public void method1() {
32         System.out.println("method");
33     }
34
35     public void method2(String s) {
36         System.out.println("method:" + s);
37     }
38
39     public String method3(String s, int i) {
40         return s + "," + i;
41     }
42
43     @Override
44     public String toString() {
45         return "Student{" +
46             "name='" + name + '\'' +
47             ", age=" + age +
48             ", address='" + address + '\'' +
49             '}';
50     }
51 }

```

o 测试类

```

1  public class ReflectDemo02 {
2      public static void main(String[] args) throws ClassNotFoundException,
        NoSuchMethodException, IllegalAccessException, InvocationTargetException,
        InstantiationException {
3          //获取Class对象
4          Class<?> c = Class.forName("com.itheima_02.Student");

```

```

5
6      //public Student(String name, int age, String address)
7      //Constructor<T> getConstructor(Class<?>... parameterTypes)
8      Constructor<?> con = c.getConstructor(String.class, int.class,
String.class);
9      //基本数据类型也可以通过.class得到对应的Class类型
10
11     //T newInstance(Object... initargs)
12     Object obj = con.newInstance("林青霞", 30, "西安");
13     System.out.println(obj);
14 }
15 }

```

2.5反射获取构造方法并使用练习2【应用】

- 案例需求
 - 通过反射获取私有构造方法并创建对象
- 代码实现
 - 学生类：参见上方学生类
 - 测试类

```

1 public class ReflectDemo03 {
2     public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, IllegalAccessException, InvocationTargetException,
InstantiationException {
3         //获取Class对象
4         Class<?> c = Class.forName("com.itheima_02.Student");
5
6         //private Student(String name)
7         //Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)
8         Constructor<?> con = c.getDeclaredConstructor(String.class);
9
10        //暴力反射
11        //public void setAccessible(boolean flag):值为true, 取消访问检查
12        con.setAccessible(true);
13
14        Object obj = con.newInstance("林青霞");
15        System.out.println(obj);
16    }
17 }

```

2.6反射获取成员变量并使用【应用】

2.6.1 Class类获取成员变量对象的方法

- 方法分类

方法名	说明
Field[] getFields()	返回所有公共成员变量对象的数组
Field[] getDeclaredFields()	返回所有成员变量对象的数组
Field getField(String name)	返回单个公共成员变量对象
Field getDeclaredField(String name)	返回单个成员变量对象

- 示例代码

```

1 public class ReflectDemo01 {
2     public static void main(String[] args) throws ClassNotFoundException,
NoSuchFieldException, NoSuchMethodException, IllegalAccessException,
InvocationTargetException, InstantiationException {
3         //获取Class对象
4         Class<?> c = Class.forName("com.itheima_02.Student");
5
6         //Field[] getFields() 返回一个包含 Field对象的数组，Field对象反映由该 Class对
象表示的类或接口的所有可访问的公共字段
7         //Field[] getDeclaredFields() 返回一个 Field对象的数组，反映了由该 Class对象
表示的类或接口声明的所有字段
8         //      Field[] fields = c.getFields();
9         Field[] fields = c.getDeclaredFields();
10        for(Field field : fields) {
11            System.out.println(field);
12        }
13        System.out.println("-----");
14
15        //Field getField(String name) 返回一个 Field对象，该对象反映由该 Class对象表
示的类或接口的指定公共成员字段
16        //Field getDeclaredField(String name) 返回一个 Field对象，该对象反映由该
Class对象表示的类或接口的指定声明字段
17        Field addressField = c.getField("address");
18
19        //获取无参构造方法创建对象
20        Constructor<?> con = c.getConstructor();
21        Object obj = con.newInstance();
22
23        //      obj.addressField = "西安";
24
25        //Field提供有关类或接口的单个字段的信息和动态访问
26        //void set(Object obj, Object value) 将指定的对象参数中由此 Field对象表示的字
段设置为指定的新值
27        addressField.set(obj, "西安"); //给obj的成员变量addressField赋值为西安
28
29        System.out.println(obj);
30
31
32
33        //      Student s = new Student();
34        //      s.address = "西安";

```

```

35 //      System.out.println(s);
36     }
37 }

```

2.6.2 Field类用于给成员变量赋值的方法

方法名	说明
void set(Object obj, Object value)	给obj对象的成员变量赋值为value

2.7 反射获取成员变量并使用练习【应用】

- 案例需求
 - 通过反射获取成员变量并赋值
- 代码实现
 - 学生类：参见上方学生类
 - 测试类

```

1  public class ReflectDemo02 {
2      public static void main(String[] args) throws Exception {
3          //获取Class对象
4          Class<?> c = Class.forName("com.itheima_02.Student");
5
6          //Student s = new Student();
7          Constructor<?> con = c.getConstructor();
8          Object obj = con.newInstance();
9          System.out.println(obj);
10
11         //s.name = "林青霞";
12         //      Field nameField = c.getField("name"); //NoSuchFieldException:
name
13         Field nameField = c.getDeclaredField("name");
14         nameField.setAccessible(true);
15         nameField.set(obj, "林青霞");
16         System.out.println(obj);
17
18         //s.age = 30;
19         Field ageField = c.getDeclaredField("age");
20         ageField.setAccessible(true);
21         ageField.set(obj, 30);
22         System.out.println(obj);
23
24         //s.address = "西安";
25         Field addressField = c.getDeclaredField("address");
26         addressField.setAccessible(true);
27         addressField.set(obj, "西安");
28         System.out.println(obj);
29     }
30 }

```


2.8反射获取成员方法并使用【应用】

2.8.1Class类获取成员方法对象的方法

- 方法分类

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

- 示例代码

```
1 public class ReflectDemo01 {
2     public static void main(String[] args) throws Exception {
3         //获取Class对象
4         Class<?> c = Class.forName("com.itheima_02.Student");
5
6         //Method[] getMethods() 返回一个包含 方法对象的数组， 方法对象反映由该 Class对
        象表示的类或接口的所有公共方法，包括由类或接口声明的对象以及从超类和超级接口继承的类
7         //Method[] getDeclaredMethods() 返回一个包含 方法对象的数组， 方法对象反映由
        Class对象表示的类或接口的所有声明方法，包括public, protected, default ( package ) 访问和私
        有方法，但不包括继承方法
8         //      Method[] methods = c.getMethods();
9         Method[] methods = c.getDeclaredMethods();
10        for(Method method : methods) {
11            System.out.println(method);
12        }
13        System.out.println("-----");
14
15        //Method getMethod(String name, Class<?>... parameterTypes) 返回一个 方法
        对象，该对象反映由该 Class对象表示的类或接口的指定公共成员方法
16        //Method getDeclaredMethod(String name, Class<?>... parameterTypes) 返回
        一个 方法对象，它反映此表示的类或接口的指定声明的方法 Class对象
17        //public void method1()
18        Method m = c.getMethod("method1");
19
20        //获取无参构造方法创建对象
21        Constructor<?> con = c.getConstructor();
22        Object obj = con.newInstance();
23
24        //      obj.m();
25    }
```

```

26         //在类或接口上提供有关单一方法的信息和访问权限
27         //Object invoke(Object obj, Object... args) 在具有指定参数的指定对象上调用此
方法对象表示的基础方法
28         //Object : 返回值类型
29         //obj : 调用方法的对象
30         //args : 方法需要的参数
31         m.invoke(obj);
32
33         //         Student s = new Student();
34         //         s.method1();
35     }
36 }

```

2.8.2 Method类用于执行方法的方法

方法名	说明
Object invoke(Object obj, Object... args)	调用obj对象的成员方法，参数是args,返回值是Object类型

2.9 反射获取成员方法并使用练习【应用】

- 案例需求
 - 通过反射获取成员方法并调用
- 代码实现
 - 学生类：参见上方学生类
 - 测试类

```

1  public class ReflectDemo02 {
2      public static void main(String[] args) throws Exception {
3          //获取Class对象
4          Class<?> c = Class.forName("com.itheima_02.Student");
5
6          //Student s = new Student();
7          Constructor<?> con = c.getConstructor();
8          Object obj = con.newInstance();
9
10         //s.method1();
11         Method m1 = c.getMethod("method1");
12         m1.invoke(obj);
13
14         //s.method2("林青霞");
15         Method m2 = c.getMethod("method2", String.class);
16         m2.invoke(obj, "林青霞");
17
18         //         String ss = s.method3("林青霞",30);
19         //         System.out.println(ss);
20         Method m3 = c.getMethod("method3", String.class, int.class);
21         Object o = m3.invoke(obj, "林青霞", 30);
22         System.out.println(o);
23     }

```

```

24         //s.function();
25         //      Method m4 = c.getMethod("function"); //NoSuchMethodException:
com.itheima_02.Student.function()
26         Method m4 = c.getDeclaredMethod("function");
27         m4.setAccessible(true);
28         m4.invoke(obj);
29     }
30 }

```

2.10反射的案例【应用】

2.10.1反射练习之越过泛型检查

- 案例需求
 - 通过反射技术，向一个泛型为Integer的集合中添加一些字符串数据
- 代码实现

```

1  public class ReflectTest01 {
2      public static void main(String[] args) throws Exception {
3          //创建集合
4          ArrayList<Integer> array = new ArrayList<Integer>();
5
6          //      array.add(10);
7          //      array.add(20);
8          //      array.add("hello");
9
10         Class<? extends ArrayList> c = array.getClass();
11         Method m = c.getMethod("add", Object.class);
12
13         m.invoke(array, "hello");
14         m.invoke(array, "world");
15         m.invoke(array, "java");
16
17         System.out.println(array);
18     }
19 }

```

2.10.2运行配置文件中指定类的指定方法

- 案例需求
 - 通过反射运行配置文件中指定类的指定方法
- 代码实现

```

1  public class ReflectTest02 {
2      public static void main(String[] args) throws Exception {
3          //加载数据
4          Properties prop = new Properties();
5          FileReader fr = new FileReader("myReflect\\class.txt");
6          prop.load(fr);
7          fr.close();
8

```

```

9      /*
10         className=com.itheima_06.Student
11         methodName=study
12     */
13     String className = prop.getProperty("className");
14     String methodName = prop.getProperty("methodName");
15
16     //通过反射来使用
17     Class<?> c = Class.forName(className);//com.itheima_06.Student
18
19     Constructor<?> con = c.getConstructor();
20     Object obj = con.newInstance();
21
22     Method m = c.getMethod(methodName);//study
23     m.invoke(obj);
24 }
25 }

```

3.模块化

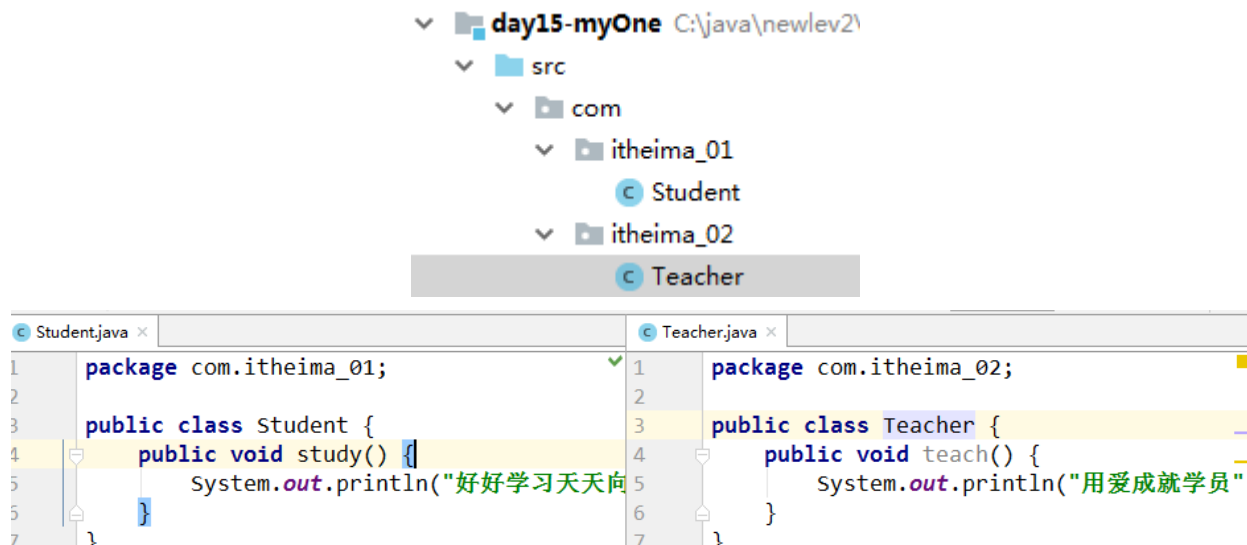
3.1模块化概述【理解】

Java语言随着这些年的发展已经成为了一门影响深远的编程语言，无数平台，系统都采用Java语言编写。但是，伴随着发展，Java也越来越庞大，逐渐发展成为一门“臃肿”的语言。而且，无论是运行一个大型的软件系统，还是运行一个小的程序，即使程序只需要使用Java的部分核心功能，JVM也要加载整个JRE环境。为了给Java“瘦身”，让Java实现轻量化，Java 9正式的推出了模块化系统。Java被拆分为N多个模块，并允许Java程序可以根据需要选择加载程序必须的Java模块，这样就可以让Java以轻量化的方式来运行

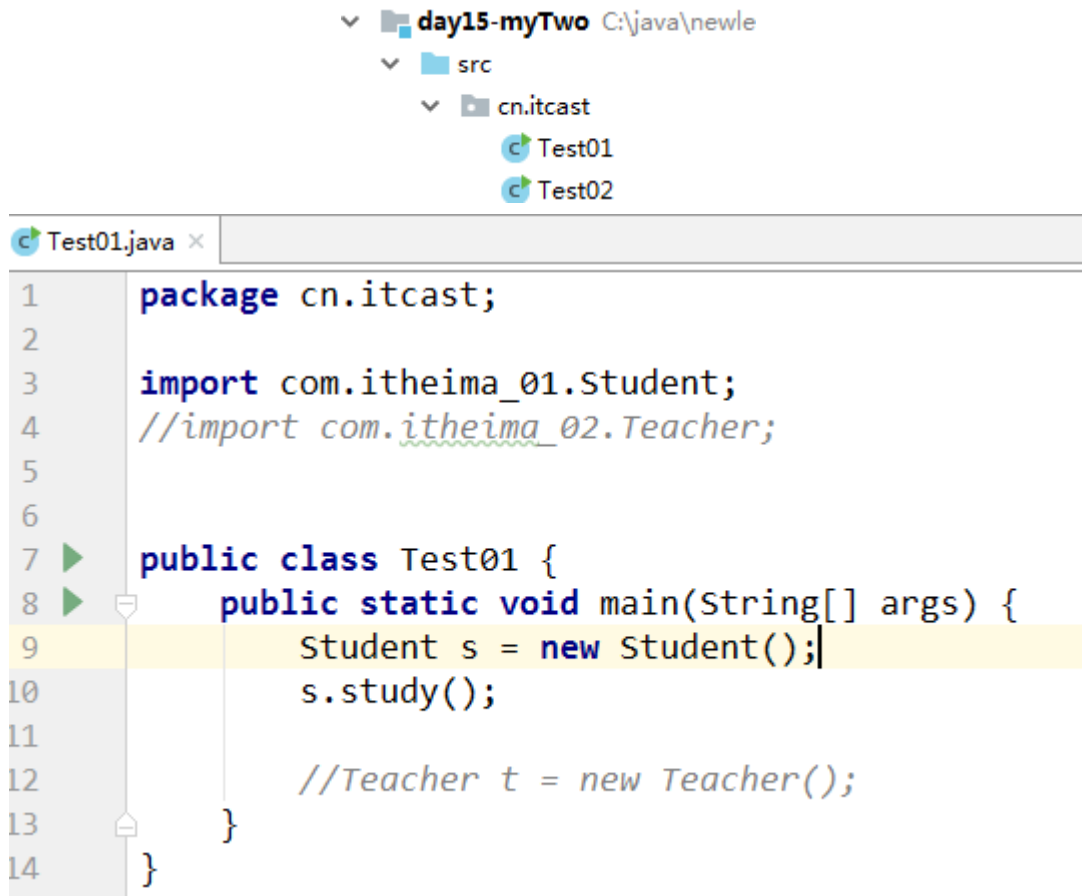
其实，Java 7的时候已经提出了模块化的概念，但由于其过于复杂，Java 7，Java 8都一直未能真正推出，直到Java 9才真正成熟起来。对于Java语言来说，模块化系统是一次真正的自我革新，这种革新使得“古老而庞大”的Java语言重新焕发年轻的活力

3.2模块的基本使用【应用】

1. 在项目中创建两个模块。一个是myOne,一个是myTwo
2. 在myOne模块中创建以下包和以下类，并在类中添加方法



3. 在myTwo模块中创建以下包和以下类，并在类中创建对象并使用



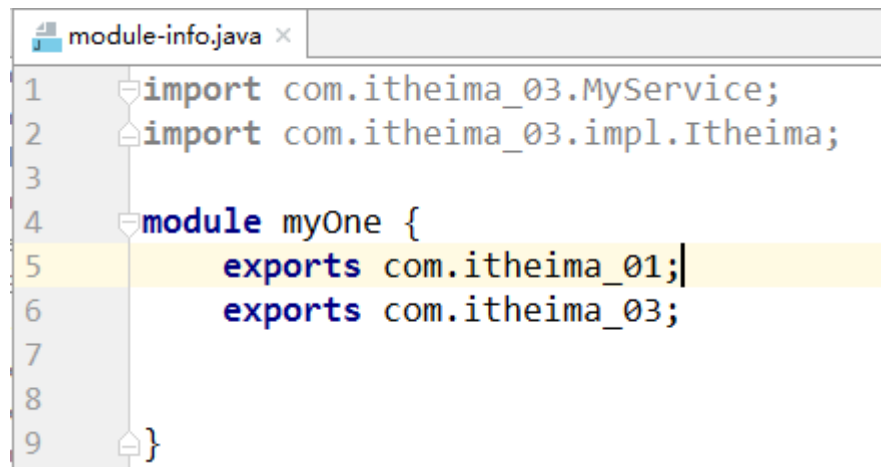
```
package cn.itcast;

import com.itheima_01.Student;
//import com.itheima_02.Teacher;

public class Test01 {
    public static void main(String[] args) {
        Student s = new Student();
        s.study();

        //Teacher t = new Teacher();
    }
}
```

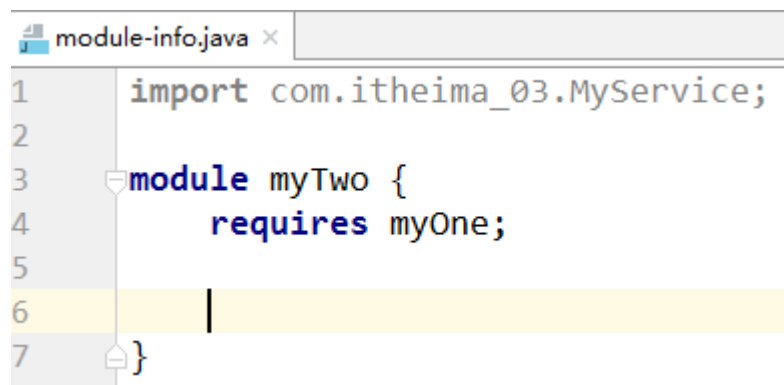
4. 在myOne模块中src目录下，创建module-info.java，并写入以下内容



```
import com.itheima_03.MyService;
import com.itheima_03.impl.Itheima;

module myOne {
    exports com.itheima_01;
    exports com.itheima_03;
}
```

5. 在myTwo模块中src目录下，创建module-info.java，并写入以下内容



```
import com.itheima_03.MyService;

module myTwo {
    requires myOne;
}
```

3.3模块服务的基本使用【应用】

1. 在myOne模块中新建一个包，提供一个接口和两个实现类

The screenshot shows an IDE with the following project structure:

- day15-myOne C:\java\newlev2\day
 - src
 - com
 - itheima_01
 - itheima_02
 - itheima_03
 - impl
 - Czxy
 - Itheima
 - MyService

The code editor shows the following files:

MyService.java

```
1 package com.itheima_03;
2
3 public interface MyService {
4     void service();
5 }
6
```

Czxy.java

```
1 package com.itheima_03.impl;
2
3 import com.itheima_03.MyService;
4
5 public class Czxy implements MyService {
6     @Override
7     public void service() {
8         System.out.println("上大学，来传智学院，一所不一样的大学，收获不一样的你");
9     }
10 }
```

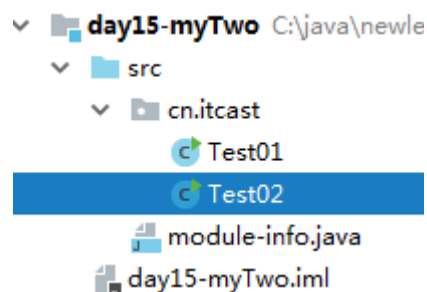
Itheima.java

```
1 package com.itheima_03.impl;
2
3 import com.itheima_03.MyService;
4
5 public class Itheima implements MyService {
6     @Override
7     public void service() { System.out.println("学IT，来黑马"); }
8 }
9
```

2. 在myOne模块中修改module-info.java文件，添加以下内容

```
module-info.java x
1  import com.itheima_03.MyService;
2  import com.itheima_03.impl.Itheima;
3
4  module myOne {
5      exports com.itheima_01;
6      exports com.itheima_03;
7
8
9      //provides MyService with Czxy;
10     provides MyService with Itheima;
11 }
```

3. 在myTwo模块中新建一个测试类



```
Test02.java x
1  package cn.itcast;
2
3  import ...
6
7  public class Test02 {
8      public static void main(String[] args) {
9          // 加载服务
10         ServiceLoader<MyService> myServices = ServiceLoader.load(MyService.class);
11
12         // 遍历服务
13         for(MyService my : myServices) {
14             // 调用接口中的方法
15             my.service();
16         }
17     }
18 }
19
```

4. 在myTwo模块中修改module-info.java文件，添加以下内容

module-info.java ×

```
1  import com.itheima_03.MyService;
2
3  module myTwo {
4      requires myOne;
5
6      uses MyService;
7  }
```