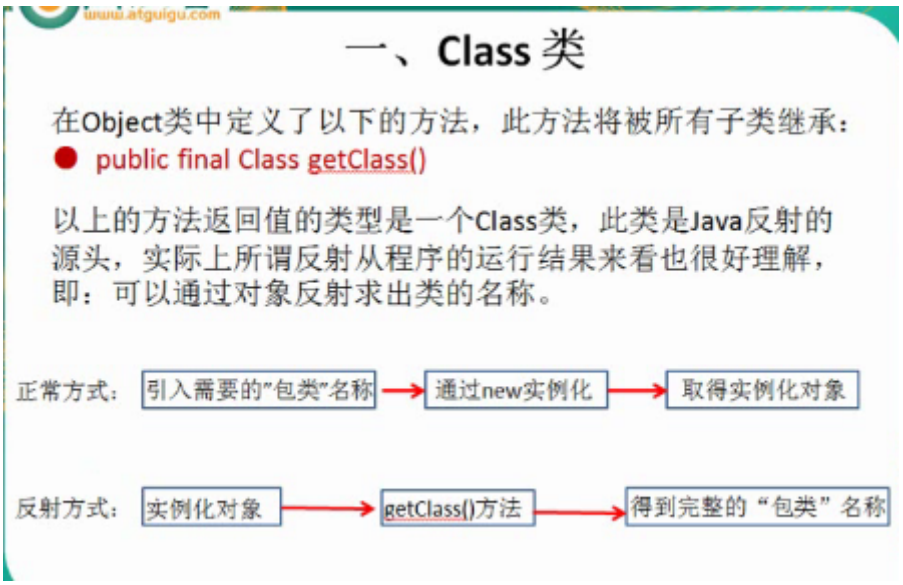


＊ 1，java.lang.Class是反射的源头

＊ 2，我们创建了一个类，通过编译javac.exe，先生成对应的.class文件

＊ 之后我们使用java.exe加载（jvm的类加载器）此.class文件，此.class文件

＊ 加载到内存以后，就是一个运行时类，存在在缓存区，那么这个运行时类本身就是一个

＊ Class的实例

＊ 3，每一个运行时类只加载一次

＊ 4，有了Class的实例以后，我们才可以进行如下的操作

1）创建对应的运行时类的对象

2）获取对应的运行时的完整结构（属性，方法，构造器，内部类，父类，异常，注解、、、、）

3）调用对应的运行时类的指定的结构（属性，方法，构造器）

4）反射的应用，动态代理

如何获取Class的实例（3种）

1，调用运行时类本身的.class属性

```
Class class2 = fanshe.class;
System.out.println(class2.getName());
Class class1 = String.class;
```

```
System.out.println(class1.getName());
```

2,通过运行时类的对象获取

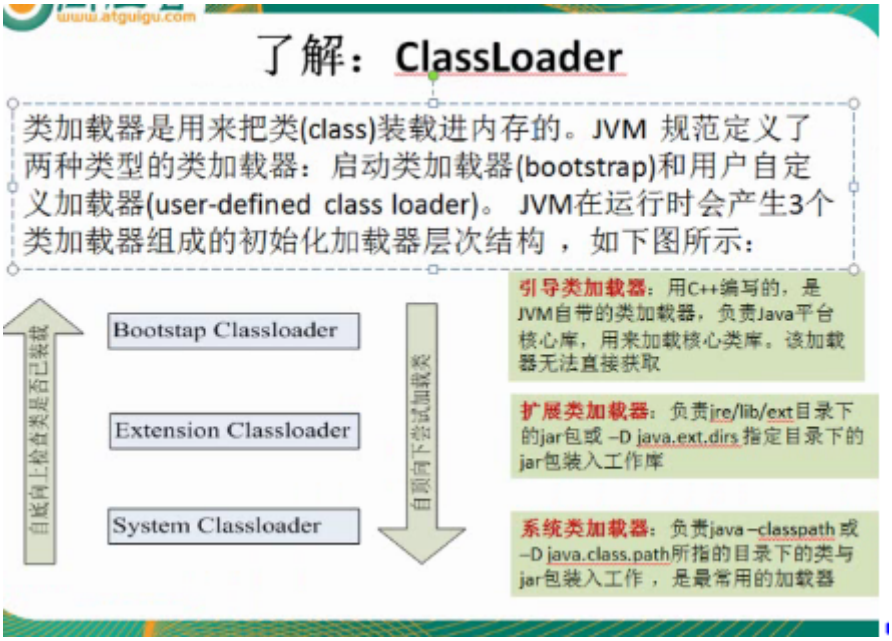
```
fanshe f1 = new fanshe();
Class class3 = f1.getClass();
System.out.println(class3.getName());
```

3，通过Class的静态方法获取，通过此方法时，反射的动态性

```
String claString = "thread.fanshe";
Class class4 = Class.forName(claString);
System.out.println(class4.getName());
```

4，通过类的加载器

```
ClassLoader classLoader = this.getClass().getClassLoader();
Class class5 = classLoader.loadClass(claString);
System.out.println(class5.getName());
```



关于类的加载器：ClassLoader

掌握如下：

```
ClassLoader lClassLoader = this.getClass().getClassLoader();
InputStream  iStream = lClassLoader.getResourceAsStream("thread\\anqili.properties");
Properties pros = new Properties();
pros.load(iStream);
String nameString = pros.getProperty("user");
```

```
                                    System.out.println(nameString);
                                    String pasString =
pros.getProperty("passworld");
                                    System.out.println(pasString);
```


```
        String claString = "thread.fanshe";
                Class class1 = Class.forName(claString);
                //创建运行时对象,newInstance调用的是对应的运行时类空参的构造器
                //创建类的时候尽量保存一个空参的构造器
                Object object  = class1.newInstance();
                fanshe f1 = (fanshe)object;
                System.out.println(f1);
```

通过反射获取类的完整结构：

一，获取属性：

1，getField（）只能获取到运行时类及其父类中声明为public的属性

2，getDeclaredFields（）获取运行时类本身声明的所有的属性

3，getModifiers（）获取每个属性的权限修饰符

```
for(Field f : fields1){
    //1.获取每个属性的权限修饰符
    int i = f.getModifiers();
    String str1 = Modifier.toString(i);
    System.out.print(str1 + " ");
```

4，getType（）获取属性的类型

```
//2.获取属性的类型
Class type = f.getType();
System.out.print(type.getName() + " ");
```

5，getName（）获取属性名

```
//3.获取属性名
System.out.print(f.getName());
```

二，获取方法

1，getmethods（）获取运行时类及其父类所有的声明为public的方法

```
Method[] m1 = clazz.getMethods();
for(Method m : m1){
    System.out.println(m);
}
```

2，getDeclaredMethods（）获取运行时类本身声明的所有的方法

```
Method[] m2 = clazz.getDeclaredMethods();
for(Method m : m2){
    System.out.println(m);
}
```

3，getAnnotations（）获取方法的注解

```
//1.注解
Annotation[] ann = m.getAnnotations();
System.out.println(ann);
```

4，getModifiers（）获取方法的权限修饰符

```
//2.权限修饰符
String str = Modifier.toString(m.getModifiers());
System.out.println(str + " ");
```

5，getReturnType（）获取方法的返回值类型

```
//3.返回值类型
Class returnType = m.getReturnType();
System.out.print(returnType.getName() + " ");
```

6，getName（）获取方法名

```
//4.方法名
System.out.print(m.getName() + " ");
```

7，getParameterTypes（）获取形参列表

```
//5.形参列表
System.out.print("(");
Class[] params = m.getParameterTypes();
for(int i = 0;i < params.length;i++){
    System.out.print(params[i].getName() + " args-" + i + " ");
}
System.out.print(")");
```

8，getExceptionType（）获取异常类型

```java
//6.异常类型
Class[] exps = m.getExceptionTypes();
if(exps.length != 0){
    System.out.print("throws ");
}
for(int i = 0;i < exps.length;i++){
    System.out.print(exps[i].getName() + " ");
}
System.out.println();
```