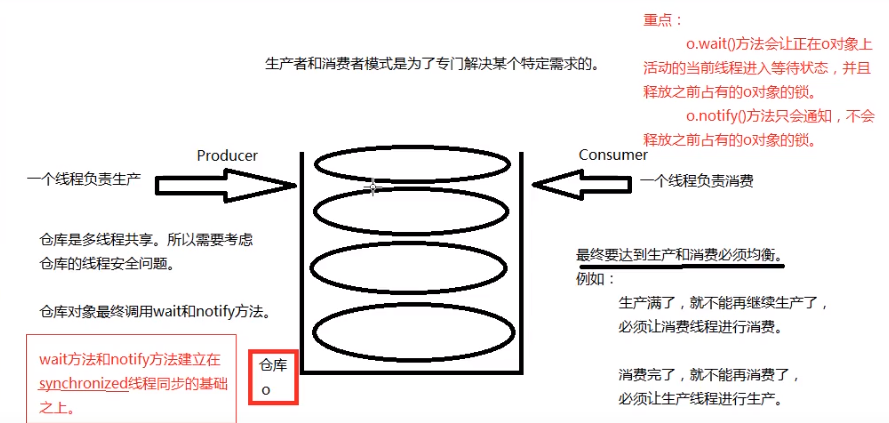
生产者与消费者模式：

1. 使用wait方法和notify方法实现生产者和消费者模式
2. wait方法和notify方法不是线程对象的方法，是普通Java对象都有的方法
3. wait方法和notify方法建立在线程同步的基础之上，因为多线程要同时操作一个仓库，有线程安全
4. wait方法的作用：让正在对象上活动的线程进入等待状态，并且释放线程之前占有的对象的锁
5. notify方法的作用：让正在对象上等待的线程唤醒，只是通知，不会释放对象上之前占有的锁



举例：

public class ThreadTest{

public static void main(String[] args){

}

}

Class Producer implements Runnable{

}

反射机制：可以操作字节码文件

1. 通过反射机制可以操作代码片段（class文件）
2. 反射机制相关的重要的类：

Java.lang.class：代表整个字节码，代表一个类型

Java.lang.reflect.Method:代表字节码中的方法字节码

Java.lang.reflect.Constructor:代表字节码中的构造方法字节码

Java.lang.reflect.Field:代表字节码中的属性字节码

1. 获取Class实例的三种方式：方式一：Class.forName()
2. 静态方法
3. 方法的参数是一个字符串
4. 字符串需要的是一个完整类名、必须带有包名。Java.lang句也不能省略

如：Class c1 = Class.forName(“java.lang.String”);

方式二：对象.getClass()，每个Java对都有getClass()

如：

String s = “aas”;

Class x = s.getClass();//x代表String.class字节码文件，x代表String类型

方式三：java语言中任何一种类型，包括基本数据类型，都有.class类型

如：Clas z = String.class;

注意：Class.forName(“java.lang.String”)运行时，加载类中的静态代码块

文件路径问题：🡪通用方式

1. 通用方式的条件是在类路径下（src下），src是类的根路径
2. String path=Thread.currentThread().getContextClassLoader().getResource(“com/aa/bb/\*\*”).getPath();

说明：Thread.currentThread()当前线程对象

1 getContextClassLoader()是线程对象的方法，可以获取到当前线程的类加载器对象

2 getResource(“”)获取资源，当前线程的类加载器默认从类的根路径下加载资源

资源绑定器：

1. java.util.ResourceBoulde

举例:ResourceBoundle rb = ResurceBoundle.getBoundle(“属性文件名”)；不能加后缀，必须在src下的（类路径下）。

ResourceBundle rb = ResourceBundle.*getBundle*("classinfo");  
String className = rb.getString("className");  
System.*out*.println(className);

类加载器：专门负责加载类的的命令或工具

JDK自带的三个加载器：启动类、扩展类、应用类加载器

Rt.jar中都是JDK最核心的类库

扩展知识：java中为了保证类加载的安全，使用了双亲委派机制。

反射机制之Java.lang.reflect.Field:代表字节码中的属性字节码

*//获取整个类* Class studentClass = Class.*forName*("com.pojo.Student");  
 *//获取类中所有的public修饰的Filed* Field[] fields = studentClass.getFields();  
 System.*out*.println(fields.length);  
 *//取出这个Filed* Field f = fields[0];  
 *//取出这个Field它的名字* String filedName = f.getName();  
 System.*out*.println(filedName);  
 *//获取所有的Filed* Field[] fs = studentClass.getDeclaredFields();  
 System.*out*.println(fs.length);  
 System.*out*.println("----------------------------");  
 *//遍历* for (Field field:fs) {  
 *//获取属性修饰符的列表，返回的是修饰符的代号* System.*out*.println(field.getModifiers().toString());  
 *//获取类型  
// System.out.println(field.getType().getName());* System.*out*.println(field.getType().getSimpleName());  
 *//获取属性的名字* System.*out*.println(field.getName());  
 }

*//实例对象*Class studentClass = Class.*forName*("com.pojo.Student");  
Object obj = studentClass.newInstance();  
*//获取num属性（根据属性的名称来获取Field）*Field noField = studentClass.getDeclaredField("num");  
*//给obj对象（student）对象的num属性赋值*noField.set(obj,12);  
*//获取对象的值*System.*out*.println(noField.get(obj));  
注意：setAccessible(true)-->打破封装，可以在外部访问private私有的属性

可变长度参数：

语法：类型… 参数名 如：int… args

1. 参数个数是0~N个
2. 在参数列表必须在最后一个位置上，而且可变长度参数只能有1个
3. 可变长度参数可以当做一个数组来看待

通过反射机制调用方法—>Java.lang.reflect.Method:代表字节码中的方法字节码

*//创建实例对象* Class studentService = Class.*forName*("com.service.StudentService");  
 Object obj = studentService.newInstance();  
 *//获取方法* Method loginMethod = studentService.getDeclaredMethod("login",String.class,String.class);  
 *//调用方法  
// Object reValue = loginMethod.invoke(obj, "admin", "111");* System.*out*.println(loginMethod.invoke(obj, "admin", "111"));

要记：调用方法四要素：1.对象 2.方法名 3.实参列表 4.返回值

重点：给你一个类，怎么获取他的父类和接口？

*//获取父类*Class stringClass = Class.*forName*("java.lang.String");  
Class superClass = stringClass.getSuperclass();  
System.*out*.println(superClass);  
*//获取String类实现的所有接口*Class[] interfaces = stringClass.getInterfaces();  
for (Class in : interfaces) {  
 System.*out*.println(in.getName());  
}

注解Annotation：是一种引用数据类型，编译之后也是XXX.class文件

1. 语法格式：@注解类型名
2. 注解可以出现在类、属性、方法、变量上

元注解：用来标注“注解类型”的注解

@Target（ElementType.\*\*）（元注解）🡪只能出现在\*上，\*可以是类、方法、属性

@Retention9（元注解）🡪可以被反射到

@Deprecated 🡪表示这个注解标注的元素已过时

@Override 🡪表示该方法继承父类的方法，重写

1. 在注解中可以定义属性

可以是各种类型及类型数组形式  
public static void main(String[] args) throws Exception{  
 *//获取这个类* Class c = Class.*forName*("com.study.annotation.MyAnnotationTest");  
 *//判断类上面是否有@MyAnnotation* System.*out*.println(c.isAnnotationPresent(MyAnnotation.class));  
 if (c.isAnnotationPresent(MyAnnotation.class)) {  
 *//获取该注解对象* MyAnnotation myAnnotation = (MyAnnotation) c.getAnnotation(MyAnnotation.class);  
 *//获取注解对象的属性，和调接口没区别* String name = myAnnotation.name();  
 System.*out*.println(myAnnotation.name());  
 }  
}

@Target({ElementType.*TYPE*, ElementType.*FIELD*})  
@Retention(RetentionPolicy.*RUNTIME*)*//runtime才能被反射*public @interface MyAnnotation {  
 String name() default "shenda";  
 *//String value();  
 //MyAnnot[] email; 属性可以是一个数组*}

@MyAnnotation  
public class MyAnnotationTest {  
}