一、 反射机制

1. Java 获取反射的三种方法

1) 通过 new 对象实现反射机制

2) 通过路径实现反射机制

3) 通过类名实现反射机制

com.open.reflect/Student.java

```
package com.open.reflect;

public class Student {
    private int id;
    String name;
    protected boolean sex;
    public float score;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isSex() {
        return sex;
    }

    public void setSex(boolean sex) {
        this.sex = sex;
    }
    public float getScore() {
        return score;
    }

    public void setScore(float score) {
        this.score = score;
    }

}
```

com.open.reflect/Get.java

```
package com.open.reflect;

public class Get {
    //获取反射机制三种方式
    public static void main(String[]args)throws ClassNotFoundException{
        //方式一(通过建立对象)
        Student stu=new Student();
        Class classobj1=stu.getClass();
        System.out.println(classobj1.getName());
        //方式二（所在通过路径-相对路径）
        Class classobj2=Class.forName("com.open.reflect.Student");
        System.out.println(classobj2.getName());
        //方式三（通过类名）
        Class classobj3=Student.class;
        System.out.println(classobj3.getName());}
}
```

2. Java 反射机制

Java 反射机制是在运行状态中，对于任意一个类，都能够获得这个类的所有属性和方法，对于任意一个对象都能够调用它的任意一个属性和方法。这种在运行时动态的获取信息以及动态调用对象的方法的功能称为 Java 的反射机制。

Class 类与 java.lang.reflect 类库一起对反射的概念进行了支持，该类库包含了 Field,Method,Constructor 类(每个类都实现了 Member 接口)。这些类型的对象时由 JVM 在运行时创建的，用以表示未知类里对应的成员。

这样你就可以使用 Constructor 创建新的对象，用 get()和 set()方法读取和修改与 Field 对象关联的字段，用 invoke()方法调用与 Method 对象关联的方法。另外，还可以调用 getConstructors()等很便利的方法，以返回表示字段，方法，以及构造器的对象的数组。这样匿名对象的信息就能在运行时被完全确定下来，而在编译时不需要知道任何事情。

com.open.reflect/Fruit.java

```
package com.open.reflect;

public class Fruit {
    public Fruit() {
        System.out.println("无参构造器 Run..........");
    }

    public Fruit(String type) {
        System.out.println("有参构造器 Run.........." + type);
    }
}
```

com.open.reflect/Fruit.java

```
package com.open.reflect;
```

```
import java.lang.reflect.Constructor;

public class ReflectTest{
       public static void main(String[]args)throws Exception{
       Class clazz=null;
       clazz=Class.forName("com.open.reflect.Fruit");
       Constructor<Fruit> constructor1=clazz.getConstructor();

Constructor<Fruit>constructor2=clazz.getConstructor(String.class);
       Fruit fruit1=constructor1.newInstance();
       Fruit fruit2=constructor2.newInstance("Apple");
       }
}
```

3. **反射的优势和劣势**

反射机制实际上就是上帝模式，如果说方法的调用是 Java 正确的打开方式，那反射机制就是上帝偷偷开的后门，只要存在对应的 class，一切都能够被调用。

那上帝为什么要打开这个后门呢？这涉及到了静态和动态的概念

静态编译：在编译时确定类型，绑定对象

动态编译：运行时确定类型，绑定对象

两者的区别在于，动态编译可以最大程度地支持多态，而多态最大的意义在于降低类的耦合性，因此反射的优点就很明显了：解耦以及提高代码的灵活性。

优势

运行期类型的判断，动态类加载：提高代码灵活度

劣势

性能瓶颈：反射相当于一系列解释操作，通知 JVM 要做的事情，性能比直接的 java 代码要慢很多

4. **应用场景**

在我们平时的项目开发过程中，基本上很少会直接使用到反射机制，但这不能说明反射机制没有用，实际上有很多设计、开发都与反射机制有关，例如模块化的开发，通过反射去调用对应的字节码；动态代理设计模式也采用了反射机制，还有我们日常使用的 Spring / MyBatis 等框架，也是利用 CGLIB 反射机制才得以实现。

案例：Spring 通过 XML 配置模式装载 Bean 的过程

将程序内所有 XML 或 Properties 配置文件加载入内存中

Java 类里面解析 xml 或 properties 里面的内容，得到对应实体类的字节码字符串以及相关的属性信息

使用反射机制，根据这个字符串获得某个类的 Class 实例

动态配置实例的属性

二、 元编程

1. **概念**

用程序来生成程序

元编程是以模板为基础，准确的说应该是模板特化和递归。

2. **举例**

任务：从一个 CSV 文件中读取数据，形成 Java 对象，然后对外提供一个 API，让别人调用。

CSV 文件叫做 employee.csv，里边的内容如下：

```
name,age,level
Andy,25,B7
Joe, 22, B6
```

我们的 API 就需要返回一个 List<Employee>， Employee 类长这个样子：

```
public class Employee{
    private String name;
    private String age;
    private String level;
    }
```

class 中的每个字段和 csv 文件的“表头”的“列名”保持一致。

写一个 EmployeeParser，专门解析 CSV 文件，形成 Employee 对象。

但是如果需求变化了呢？

那个 CSV 文件新加了一个字段，叫做 salary ，程序也需要修改！

```
name,age,level,salary
Andy,25,B7,3000
Joe, 22, B6,2500
```

如果后期还会新加字段呢？

解决：使用模板用程序来生成程序

根据 CSV 的列名自动地生成 Employee 类。

CSV 的“列名”经过读取，可以变成一个 Java 的 List ，例如["name","age","level"]，

如何写一段代码，把这个 List 变成一个 Employee Class 呢？

可以用模板技术，比如 velocity 模板，定义一个 employee.vm

```
public class Employee{
    #foreach ($field in $headers)
    private String $field;
    #end
}
```

然后再写一个代码生成器，读取 employee.csv 的“表头”，形成 List，把 List 传递给这个 employee.vm 模板，就可以输出 Java 类。