java.lang.Class
java.lang.reflect.Method
java.lang.reflect.Field
java.lang.reflect.Constructor

.实例化 Class 类对象（四种方法）
① Class a = String.class;
② Class a = 对象.getClass();
③ class a = class.forName(包.类)
④ 略

通过反射获取类的完整结构.
① 获取父类
② 获取接口
③ 获取构造方法
④ 获取修饰符
⑤ 获取泛型类型

```
package com.test.clazz;


import java.lang.reflect.Constructor;

public class Test {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.test.clazz.Student");
            Class superClazz = clazz.getSuperclass();  // 获取父类
            System.out.println(superClazz.getName());
            Class[] interfaces = clazz.getInterfaces();  // 获取所有实现的接口
            for(Class c:interfaces){
                System.out.println(c.getName());
```

```
        }
        Constructor[] cons = clazz.getConstructors();  // 获取公有的构造方法
        for(Constructor c:cons){
            System.out.println(c.getName()+","+c.getModifiers()); // 打印构造方法的名称和修饰符，1代表public、2代表private
            Class[] paramClazz = c.getParameterTypes();  // 获取参数类型
            for(Class pc:paramClazz){
                System.out.println(pc.getName());
            }
        }
        Constructor[] cons1 = clazz.getDeclaredConstructors();  // 获取所有构造方法，包括公有和私有的（私有的在单例设计模式中就有）
        for(Constructor c:cons){
            System.out.println(c.getName()+","+c.getModifiers()); // 打印构造方法的名称和修饰符
        }
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
  }
}
/*
Class clazz = Class.forName();
        = "类的实例".getClass();
        = String.class;
 */
```

通过反射的构造方法创建对象

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

public class Test {
   public static void main(String[] args) {
      try {
         Class clazz = Class.forName("com.test.clazz.Student");
         try {
            Object obj =  clazz.newInstance();  // 这里就调用构造函数了
            System.out.println("还没强转");
            Student student = (Student) obj;

            Constructor c = clazz.getConstructor(String.class);  // 少了个s
            Student stu1 = (Student) c.newInstance("湖北大学");

            // 通过反射机制，可以强制调用私有方法
            Constructor d = clazz.getDeclaredConstructor(String.class,String.class,int.class);
            d.setAccessible(true);  // 解除封装
            Student stu2 = (Student) d.newInstance("湖北大学", "李德敖", 21);
         } catch (InstantiationException e) {
            e.printStackTrace();
```

```
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

通过反射机制获取全部方法

```
import java.lang.reflect.Method;

public class Test {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.test.clazz.Student");
            Method[] ms = clazz.getMethods();
            for(Method m:ms){
                System.out.println("方法名:"+m.getName());
                System.out.println("返回值类型:"+m.getReturnType());
                System.out.println("修饰符:"+m.getModifiers());
                Class[] pcs = m.getParameterTypes();  // 获取方法参数类型
                if(pcs!=null&&pcs.length>0){
                    for(Class pc:pcs){
                        System.out.println("参数类型" + pc.getName());
                        System.out.println("--------------------");
                    }
                }
            }
            System.out.println("**********************************\n**********************************");
            Method[] ams = clazz.getDeclaredMethods();  // 包含私有方法
            for(Method m:ams){
                System.out.println("方法名:"+m.getName());
                System.out.println("返回值类型:"+m.getReturnType());
                System.out.println("修饰符:"+m.getModifiers());
                Class[] pcs = m.getParameterTypes();  // 获取方法参数类型
                if(pcs!=null&&pcs.length>0){
                    for(Class pc:pcs){
                        System.out.println("参数类型" + pc.getName());
                        System.out.println("--------------------");
                    }
                }
            }
  } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

通过反射获取类的全部属性和包名

```
import java.lang.reflect.Field;
public class Test {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.test.clazz.Student");
            Field[] fs = clazz.getFields();
            for(Field f:fs){
                System.out.println("属性名:"+f.getName());
                System.out.println("修饰符:"+f.getModifiers());
                System.out.println("类型"+f.getType());
                System.out.println("--------------------------------");
            }
            System.out.println("================================");
            Field[] fss = clazz.getDeclaredFields();
            for(Field f:fss){
                System.out.println("属性名:"+f.getName());
                System.out.println("修饰符:"+f.getModifiers());
                System.out.println("类型"+f.getType());
                System.out.println("--------------------------------");
            }

            Package p = clazz.getPackage();
            System.out.println("包名:"+p.getName());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

反射获取构造方法

```
public class Test {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.test.clazz.Student");
            Constructor con = clazz.getConstructor(); // 获取无参构造
            Student stu = (Student)con.newInstance();
            Method m = clazz.getMethod("setInfo", String.class, String.class);  // 获取指定方法
            m.invoke(stu, "李德敖", "第一中学");  // 调用指定方法:参数1是需要实例化的对象，后面的是调用
方法的实际参数

            Method m1 = clazz.getDeclaredMethod("test");
            m1.setAccessible(true);
            m1.invoke(stu);
 } catch (ClassNotFoundException | NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
```

```
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

反射获取私有属性

```
public class Test {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.test.clazz.Student");
            Constructor con = clazz.getConstructor();
            Student stu = (Student) con.newInstance();

            Field f = clazz.getField("school");
            f.set(stu, "第三中学");
            String school = (String) f.get(stu);
            System.out.println(school);

            Field f1 = clazz.getDeclaredField("address");
            f1.setAccessible(true);
            f1.set(stu, "湖北大学");
            String address = (String) f1.get(stu);
            System.out.println(address);
    } catch (ClassNotFoundException | NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        }
    }
}
```

动态代理

```
package com.test.clazz;

public interface TestDemo {
    void test1();
    void test2();
}
```

```
package com.test.clazz;

public class TestDemoImp implements TestDemo{

    @Override
    public void test1() {
        System.out.println("执行test1方法");
    }

    @Override
    public void test2() {
        System.out.println("执行test2方法");
    }
}
```

```
package com.test.clazz;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

/**
 * 动态代理类
 * @author lda
 */
public class ProxyDemo implements InvocationHandler {
    Object obj;  // 被代理对象
    public ProxyDemo(Object obj){
        this.obj = obj;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println(method.getName()+"开始执行");
        Object result = method.invoke(this.obj, args);
        System.out.println(method.getName()+"结束执行");
        return result;
    }
```

```
package com.test.clazz;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;

public class Test2 {
    public static void main(String[] args) {
        TestDemo test = new TestDemoImp();
        /**
         * 如果一个对象想要被Proxy.newProxyInstance代理，他必须有相应的接口
         */
```

```
        test.test1();
        test.test2();
        /**
         *需求:
         * 在执行方法前打印test1、test2开始执行
         * 执行方法后打印test1、test2执行完毕
         */
        InvocationHandler handler = new ProxyDemo(test);

        /**
         * 参数1是代理对象的类加载器
         * 参数2是被代理对象的的接口
         * 参数3是代理对象
         *
         * 返回值是代理成功后的对象
         */
        // 注意这里的定义类型和强转必须用TestDemo接口类型
        TestDemo t = (TestDemo)
Proxy.newProxyInstance(handler.getClass().getClassLoader(),test.getClass().getInterfaces(),handler);
        t.test1();
        t.test2();
    }
}
```