# 一.注解(Annotation)

## 1.什么是注解

```
1.Annotation的作用：
    不是程序本身，可以对程序做出解释(这一点跟注释comment没什么区别)。
    可以被其他程序(如：编辑器等)读取。
2.格式：
    "@+注释名"，可带参数，也可不带。
    如:@SuppressWanings(value="unckecked").
3.使用范围：
    可以附加在package,class,method,field等上面。
```

## 2.内置注解

```
1.@Override:表示一个方法声明打算重写父类中的另一个方法声明。
2.@Deprecated:可用于修辞方法，属性，类。表示不鼓励程序员使用这样的元素，通常是以为它很危险或者存在更好的选择。
3.@SuppressWarnings:镇压警告信息。需要一个参数。
    @SuppressWarnings("all")
    @SuppressWarnings("unchecked")
    @SuppressWarnings(value={"unckecked","deprecation"})
    等等.........
```

## 3.元注解

```
1.作用：
    负责注释其他注解，Java定义了4个meta-annotation，他们被用来提供对其他annotation类型做说明。

2.(@Target,@Retention,@Documented,@Inherited)
    1.@Target:表示注解的使用范围。
    2.@Retention:表示注解的生命周期(SOURCE<CLASS<RUNYIME)，一般是RUNTIME。
    3.@Documented:表示该注解将被写在Javadoc中。
    4.@Inherited:表示子类可以继承父类中的该注解。
```

## 4.自定义注解

```
1.使用@interface自定义注解时。自动继承Java.lang.annotation.Annotation接口

2.分析：
    1.@interface用来声明一个注解，格式：public @interface 注解名{自定义内容}。
    2.可以设置参数：类型+名+();
    3.返回值就是参数类型(基本类型)
    4.可以用default来声明参数的默认值。
    5.如果只有一个参数成员，一般参数名是"value".
    6.注解参数必须有值，一般设置默认值。
```

# 二.反射(Reflection)

## 1.什么是反射

```
1.reflection（反射）是Java被视为动态语言的关键，反射机制允许程序在执行期间借助于reflection API 取得任何类的内部信息，并且能操作任意对象的内部属性及方法。
    Class c = Class.forName("java.lang.String")

2.加载完类之后，在对内存的方法区中就产生了一个Class类型的对象(一个类只有一个Class对象),这个对象就包含了完整的类的结构信息。我们可以通过这个对象看到类的结构。这个对象就像一面镜子，透过这个镜子看到类的结构。称为:反射。
```


## 2.反射的优缺点

```
1.优点：
    可以实现动态创建对象和编译，体现出很大的灵活性。

2.缺点：
    对性能有影响。
```

# 3.Class类

```
1.Class本身也是一个类
    Class对象智能由系统简历对象
    Class类是Reflection的根源,针对任何你想动态加载、运行的类，唯有获得相应的lass对象。

2.获得Class对象：
    eg：
    Person person = new Teacher();
    //1.方式一：通过对象获得
        Class c1 = person.getClass();
        System.out.println(c1.hashCode());

        //2.方式二:通过forname()获得
        Class c2 = Class.forName("Reflection.Teacher");
        System.out.println(c2.hashCode());

        //3.方式三:通过类名.class获得
        Class c3 = Teacher.class;
        System.out.println(c3.hashCode());
```

## Class类的常用方法

| 方法名 | 功能说明 |
| --- | --- |
| static ClassforName(String name) | 返回指定类名name的Class对象 |
| Object newInstance() | 调用缺省构造函数，返回Class对象的一个实例 |
| getName() | 返回此Class对象所表示的实体（类，接口，数组类或void）的名称。 |
| Class getSuperClass() | 返回当前Class对象的父类的Class对象 |
| Class[] getinterfaces() | 获取当前Class对象的接口 |
| ClassLoader getClassLoader() | 返回该类的类加载器 |
| Constructor[] getConstructors() | 返回一个包含某些Constructor对象的数组 |
| Method getMothed(String name,Class.. T) | 返回一个Method对象，此对象的形参类型为paramType |
| Field[] getDeclaredFields() | 返回Field对象的一个数组 |

## 4.那些类型有Class对象

```
1.class:外不累,成员(成员内部类，静态内部类)，局部内部类。
2.interface：接口
3.[]:数组
4.enum：枚举
5.annotation：注解@interface
6.primitive type：基本数据类型
7.void
```

## 5.类的加载与ClassLoader的理解

```
1.加载：将class文件字节码内容你在到内存中，并把静态数据结构转换成方法区的运行时数据结构，然后生成一个代表这个类的java.lang.Class对象。
2.链接
    1.验证
    2.准备：为类变量(static)飞陪内存并设置默认值。
    3.解析
3.初始化
    执行类构造器<clinit>()方法
```

## 6.类的初始化

```
1.类的主动引用(会初始化)
    运行开始，先初始化main方法所在的类
    new 一个类的对象
    调用类的静态成员(除了dinal常量)和静态方法
    使用java.lang.reflect包的方法对类进行反射调用
    初始化时，先初始化其父类
2.类的被动引用(不会初始化)
    用过之类引用父类的静态变量，不会导致子类初始化。
    通过数组定义类引用，不会触发类的初始化
    引用常量不会出发此类的初始化

```

# 7.类加载器

```
1.类加载器：把类(class)装在进内存的
2.分类
    引导类加载器：c++便携，是JVM自带的加载器，负责java平台核心库。无法直接获取
    扩展类加载器：负责jre/lib/ext目录下的jar包等
    系统类加载器：最常用的
```

# 8.反射获取类的完整结构

```
1.反射获得类的:
    Field,Method,Constructor,Superclass,Interface,annotation.
2.方法:
    class.getXXX():      //获取类的public类型的XXX
    class.getDeclareXXX():       //获取类的所有类型的XXX

```

# 9.获得class对象之后

### 1.创建类的对象

```
调用Class对象的newInstance()方法
    1.类必须有一个无参构造器
    2.类的构造器的访问权限足够(public，private，之类的)
```

### 2.必须要无参吗?

```
可以没，步骤如下：
    1.通过Class类的getDeclaredConstructor(Class...ParameterTypes)取得本类的重载构造器
    2.传入对应的各个参数
    3.通过Constructor实例化对象
```

### 3.调用指定方法

```
通过反射，调用类的方法，一个Method完成
    1.通过Class类的getDeclaredMothod()方法取得一个Method对象，并设置各个参数
    2.之后使用Object invoke()方法进行调用，并传参。
```

### 4.invoke()函数

**Object invoke(Object obj, Object … args)**

- Object 对应原方法的返回值，若原方法无返回值，此时返回null
- 若原方法若为静态方法，此时形参Object obj可为null
- 若原方法形参列表为空，则Object[] args为null
- 若原方法声明为private,则需要在调用此invoke()方法前，显式调用方法对象的setAccessible(true)方法，将可访问private的方法。

### 5.setAccessible()函数

**setAccessible**

- Method和Field、Constructor对象都有setAccessible()方法。
- setAccessible作用是启动和禁用访问安全检查的开关。
- 参数值为true则指示反射的对象在使用时应该取消Java语言访问检查。
  - 提高反射的效率。如果代码中必须用反射，而该句代码需要频繁的被调用，那么请设置为true。
  - 使得原本无法访问的私有成员也可以访问
- 参数值为false则指示反射的对象应该实施Java语言访问检查

## 10.对象关系映射(ORM)

```
1.ORM:Object relationship Mapping -->>对象关系映射
    类和表结构对应
    属性和字段对应
    对象和记录对应
2.要求
    利用注解和反射完成类和表的映射关系
```