RTTI与反射机制

**RTTI：**

* 运行时识别一个对象的类型
* Java程序中，所有的类都是在对其第一次使用时，动态加载到JVM中的。当程序创建第一个对类的静态成员的引用时，就会加载这个类。
* 证明了构造器也是类的静态方法，即使构造器之前没有使用static关键字，因此，使用new操作符创建类的新对象也会被当做对类的静态成员的引用
* Java程序在它开始运行之前并非被完全加载，各个部分是在必需时才加载

**Class对象**

* 类加载器首先检查这个类的Class对象是否已经加载，如果尚未加载，默认的类加载器会根据类名查找.class文件，加载时会验证是否被破坏或者包含不良代码，一旦某个类的Class对象被载入内存，它就被用来创建这个类的所有对象
* Class.forName("");是取得Class对象的引用的一种方法，是用一个包含目标类的文本名的String作为输入参数，返回的是一个Class对象的引用，如果没有使用返回值，那么对forName的调用是为了产生作用（例如数据库驱动）:如果那个类没有被加载就加载他，加载的过程中static子句被执行
* ！！！而且不需要为了获得Class引用而持有该类型的对象
* ！！！forName()里面的字符串中必须使用全限定名（包含包名）
* Object.getClass()返回表示该对象的实际类型的Class引用
* getSimpleName()产生单独类名
* getCanonicalName()产生全限定名
* Class.getInterfaces()返回Class对象表示中所包含的接口
* getSuperclass()，返回其直接基类而不是接口
* newInstance()，返回Object类型实现虚拟构造器，虚拟构造器允许声明：不知道确切类型，但是你自己要正确创建你自己
* 类字面常量： 使用 .class的形式创建对Class对象的引用，不会自动初始化类中的对象，而Class.forName("")会立即进行初始化
* 为了使用类准备的工作

1. 加载：类加载器查找字节码，并从中创建一个Class对象
2. 链接：验证字节码，为静态域分配存储空间，如果必需，解析这个类创建的对其他类的所有引用
3. 初始化：如果该类有超类，则对其初始化，执行静态初始化器和静态初始化块

* Class引用表示的就是他所指向的对象的确切类型，而该对象便是一个Class类的一个对象
* 泛化的Class引用： Class<?> intClass = int.Class  好处是
* 还可以与将通配符与extends关键字相结合，创建一个范围 Class<? extends Number> bounded = int.class;

**反射：运行时的类信息**

* RTTI是类型在编译时必须已知，这样才能识别
* Class类与java.lang.reflect类库一起对反射的概念支持
* 反射提供了一种可以用来检查可用的方法，并返回方法名的机制
* 类库包含 Field Method Constructor类（每个类都实现了Member接口）这些类型的对象是由JVM在运行时创建的，用以表示未知类里对应的成员，
* Constructor: 创建新的对象
* get()  set() 方法读取和修改与Filed对象关联的字段
* invoke()方法调用与Method对象关联的方法
* getFields()  getMethods() getConstructors() 返回对象的数组

**动态代理：代理设计模式是为了提供额外的或不同的操作而插入的用来代替『实际』对象的对象，这些操作通常涉及与实际对象的通信**

接下来的，看这位仁兄的吧

<https://blog.csdn.net/yangaiyu/article/details/73827043>