《31 讲带你搞懂 SkyWalking》

徐郡明　前搜狗资深技术专家、源码剖析系列畅销书作者

第04讲：实战掌握 Byte Buddy 体验代码生成的顺畅

为什么需要运行时代码生成？

Java 是一种强类型的编程语言

即要求所有变量和对象都有一个确定的类型

如果在赋值操作中出现类型不兼容的情况，就会抛出异常

强类型检查在大多数情况下是可行的

然而在某些特殊场景下

强类型检查则成了巨大的障碍

为什么需要运行时代码生成？

Java 反射 API 两个明显的缺点：

- 在早期 JDK 版本中，反射 API 性能很差
- 反射 API 能绕过类型安全检查，反射 API 自身并不是类型安全的

为什么选择Byte Buddy？

以下代码生成库在 Java 中也很流行：

- Java Proxy

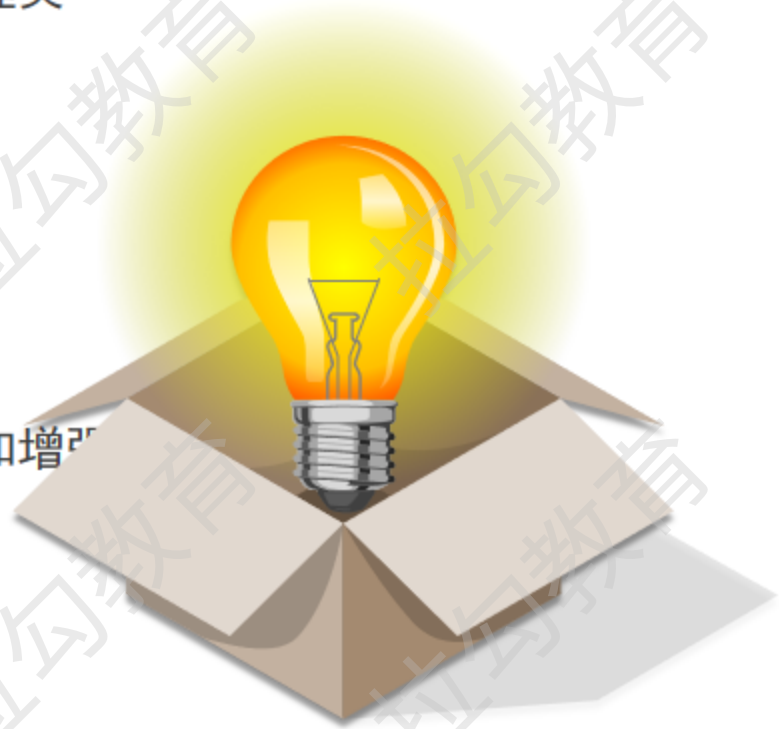
是 JDK 自带的一个代理工具，它允许为实现了一系列接口的类生成代理类

要求目标类必须实现接口是一个非常大限制

例如在某些场景中

目标类没有实现任何接口且无法修改目标类的代码实现

Java Proxy 就无法对其进行扩展和增



为什么选择Byte Buddy？

拉勾教育

以下代码生成库在 Java 中也很流行：

- CGLIB

CGLIB 诞生于 Java 初期，但不幸的是没有跟上 Java 平台的发展

虽然 CGLIB 本身是一个相当强大的库，但也变得越来越复杂

导致许多用户放弃了 CGLI

为什么选择Byte Buddy？

以下代码生成库在 Java 中也很流行：

- Javassist

对Java 开发者非常友好

它使用Java 源代码字符串和 Javassist 提供的一些简单 API ，共同拼凑出用户想要的 Java 类

自带编译器，拼凑好的 Java 类在程序运行时会被编译成为字节码并加载到 JVM 中

Javassist 库简单易用，而且使用 Java 语法构建类与平时写 Java 代码类似

Javassist 编译器在性能上比不了 Javac 编译器

在动态组合字符串以实现比较复杂的逻辑时容易出错

为什么选择Byte Buddy？

以下代码生成库在 Java 中也很流行：

- Byte Buddy

Byte Buddy 提供了一种非常灵活且强大的领域特定语言

通过编写简单的 Java 代码即可创建自定义的运行时类

与此同时

Byte Buddy 还具有非常开放的定制性

能够应付不同复杂度的需求

Byte Buddy 基础入门

拉勾教育

	基线	Byte Buddy	cglib	Javassist	Java proxy
简单的类创建	0.003 (0.001)	142.772 (1.390)	515.174 (26.753)	193.733 (4.430)	70.712 (0.645)
接口实现	0.004 (0.001)	1'126.364 (10.328)	960.527 (11.788)	1'070.766 (59.865)	1'060.766 (12.231)
方法调用	0.002 (0.001)	0.002 (0.001)	0.003 (0.001)	0.011 (0.001)	0.008 (0.001)
类型扩展	0.004 (0.001)	885.983 (7.901)	1'632.730 (52.737)	683.478 (6.735)	5'408.329 (52.437)
父类方法调用	0.004 (0.001)	0.004 (0.001)	0.021 (0.001)	0.025 (0.001)	0.004 (0.001)

Byte Buddy 基础入门

拉勾教育

```
DynamicType.Unloaded<?> dynamicType = new ByteBuddy()
        .subclass(Object.class) // 生成 Object的子类
        .name("com.xxx.Type")    // 生成类的名称为
"com.xxx.Type"
        .make();
```

Byte Buddy 基础入门

Byte Buddy 动态增强代码总共有三种方式：

- subclass

  对应 ByteBuddy.subclass() 方法

  就是为目标类（即被增强的类）生成一个子类

  在子类方法中插入动态代码

Byte Buddy 基础入门

拉勾教育

Byte Buddy 动态增强代码总共有三种方式：

- rebasing

  对应 ByteBuddy.rebasing() 方法

当使用 rebasing 方式增强一个类时

Byte Buddy 保存目标类中所有[illegible]现

Byte Buddy 基础入门

```
class Foo { // Foo的原始定义
  String bar() { return "bar"; }
}

class Foo { // 增强后的Foo定义
  String bar() { return "foo" +
bar$original(); }
  private String bar$original() { return
"bar"; }
}
```

Byte Buddy 基础入门

Byte Buddy 动态增强代码总共有三种方式：

- redefinition

  对应 ByteBuddy.redefine() 方法

  当重定义一个类时，Byte Buddy 可以对一个已有的类添加属性和方法

  或者删除已经存在的方法实现

  如果使用其他的方法实现替换已经存在的方法实现，则原来存在的方法实现就会消失

增强 Foo 类的 bar() 方法使其直接返回 "unknow" 字符

串增强结果如下

```
class Foo { // 增强后的Foo定义
  String bar() { return "unknow"; }
}
```

Byte Buddy 提供了几种类加载策略

这些策略定义在 ClassLoadingStrategy.Default 中：

- WRAPPER 策略：创建一个新的 ClassLoader 来加载动态生成的类型
- CHILD_FIRST 策略：创建一个子类优先加载的 ClassLoader，即打破了双亲委派模型
- INJECTION 策略：使用反射将动态生成的类型直接注入到当前 ClassLoader 中

Byte Buddy 基础入门

```
Class<?> loaded = new ByteBuddy()
        .subclass(Object.class)
        .name("com.xxx.Type")
        .make()
        // 使用 WRAPPER 策略加载生成的动态类型
        .load(Main2.class.getClassLoader(),
                ClassLoadingStrategy.Default.WRAPPER)
        .getLoaded();
```

实例演示

拉勾教育

后续内容为录屏演示

请根据视频内容进行练习

实例演示

Byte Buddy 基础入门


```
String hello = new ByteBuddy()
    .subclass(DB.class)
    .method(named("hello"))
    .intercept(MethodDelegation.withDefaultConfiguration()
        .withBinders(
            // 要用@Morph注解之前，需要通过  Morph.Binder 告诉 Byte
Buddy
            // 要注入的参数是什么类型
            Morph.Binder.install(OverrideCallable.class)
        )
        .to(new Interceptor()))
    .make()
    .load(Main.class.getClassLoader(), INJECTION)
    .getLoaded()
    .newInstance()
    .hello("World");
```

Byte Buddy 基础入门

通过一个示例展示 Byte Buddy 除修改方法实现之外的其他三个功能：

- defineMethod() 方法：新增方法
- defineField() 方法：新增字段
- Implement() 方法：实现一个接口

Next：第05讲《OpenTracing 简介，先有标准后有天》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U