转载请注明链接:https://blog.csdn.net/feather_wch/article/details/51721514

讲解编程语言的分类(什么是静态类型，动态类型，强类型和弱类型预研)，介绍动态代理和反射机制，动态代理方面介绍JDK提供的动态代理，以及通过cglib等高效的字节码操作机制实现的动态代理。

# 动态代理

版本: 2018/09/07-1(18:00)



# 问题汇总

1. 编程语言分类所用到的动态类型和静态类型是指什么?
2. 编程语言分类所用到的强类型和弱类型是什么?
3. 运行时检查的语言是属于什么类型的?
4. 编译期检查的语言属于什么类型的?
5. 不同类型的变量进行赋值时，需要显式进行类型转换的语言，是什么类型的语言?
6. 不同类型的变量进行赋值时，不需显式进行类型转换的语言，是什么类型的语言?
7. 哪些语言是动态类型语言?
   > JavaScript、Python
8. 哪些语言是静态类型语言?
   > Java、C、C++、Kotlin
9. 哪些语言是弱类型语言?
   > C、C++、PHP、JavaScript
10. 哪些语言是强类型语言?
    > Java、Kotlin
11. 反射机制是什么?

12. 通过反射机制能干什么?
13. AccessibleObject.setAccessible(bolean flag)的作用
14. Java9中的setAccessible
15. 动态代理是什么？有什么作用?
16. 动态代理和静态代理的区别
17. 动态代理的三要素
18. 动态代理的应用场景
19. 动态代理解决了什么问题？在具体业务中有哪些应用场景?
20. 动态代理的实现方法?
21. JDK动态代理在设计和实现上与cglib等方式有什么不同?
22. JDK Proxy比cglib和Javassist慢几十倍?
23. JDK Proxy和cglib如何进行取舍?
24. JDK动态代理的实例
25. JDK Proxy的作用
26. JDK Proxy的缺点
27. JDK Prxoy的优势
28. cglib框架的优势
29. 面向切面编程AOP是什么?
30. 字节码操作机制是什么?
31. ASM是什么？Javasist是什么?

# 语言分类(3)

1、编程语言的分类?

> 1. 动态类型和静态类型
>    1. 动态类型：运行时检查
>    2. 静态类型：编译期检查
> 2. 强类型和弱类型。
>    1. 强类型: 不同类型变量赋值时，需要显式的强制地进行 类型转换
>    2. 弱类型：不同类型变量赋值时，不需要显式的 类型转换

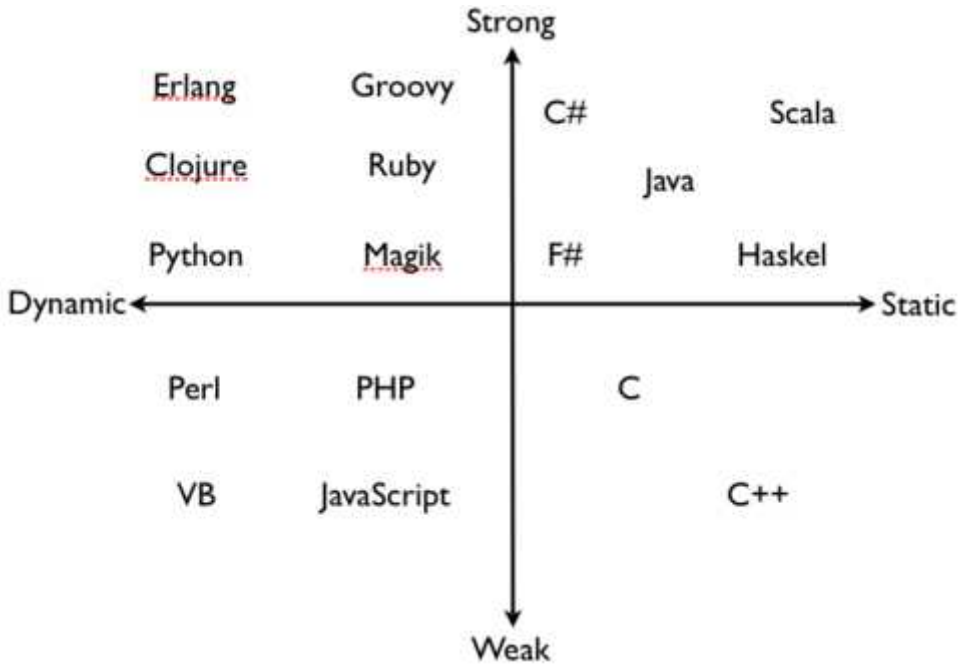
2、Java语言是什么类型的语言?

> 1. Java是静态的强类型的语言，但是因为具有反射机制，因此也具有部分动态类型语言的能力。

3、哪些语言是动态类型、静态类型、强类型、弱类型?

| | Java | Kotlin | C | C++ | JavaScript | Pyhton | Ruby | C# | PHP |
|---|---|---|---|---|---|---|---|---|---|
| 动态类型 | | | | | ★ | ★ | ★ | | ★ |

| | Java | Kotlin | C | C++ | JavaScript | Pyhton | Ruby | C# | PHP |
|---|---|---|---|---|---|---|---|---|---|
| 静态类型 | ★ | ★ | ★ | ★ | | | | ★ | |
| 强类型 | ★ | ★ | | | | ★ | ★ | ★ | |
| 弱类型 | | | ★ | ★ | ★ | | | | ★ |




# 反射机制()

1、反射机制是什么?

> 1. Java语言提供的基础功能
> 2. 赋予程序在运行时自省(introspect)的能力
> 3. 能够让Java灵活地操作运行时才能确定的信息。

2、通过反射机制能干什么?

> 可以直接操作类或者对象:
>
> 1. 获取对象的类
> 2. 获取类声明的属性和方法
> 3. 调用方法
> 4. 构造对象
> 5. 在运行时修改类定义。

3、AccessibleObject.setAccessible(bolean flag)的作用

> 1. 可以在运行时修改成员访问限制。
> 2. 比如去绕开一些API的限制。

4、Java9中的setAccessible

> 1. Java 9以后，Jigsaw项目新增的模块化系统，会对反射进行限制。并提出了Open的概念。
> 2. 被反射操作的模块 以及 指定的包 需要对反射调用者模块是Open的，才能调用 `setAccessible`
> 3. Java 9暂时保留和兼容了Java 8的行为，但以后可能会出现变化。
> 4. 可以同参数显式设置，setAccessible是否可以使用。

```
--illegal-access={permit | warn | deny}
```

# 动态代理

1、动态代理是什么？有什么作用?

> 1. 一种方便运行时动态构建代理、动态处理代理方法调用的机制
> 2. 一种广泛应用于产品开发中的技术。
> 3. 通过 动态代理 能优雅的解决繁琐的重复编程
> 4. 通过代理可以让 调用者 和 实现者 解耦。

2、动态代理和静态代理的区别

> 1. 静态代理：在编译阶段就知道需要代理哪一个对象。
> 2. 动态代理：运行时才知道要代理哪一个对象

3、动态代理的三要素

> 1. 抽象类接口
> 2. 被代理类：具体实现抽象接口
> 3. 动态代理类：实际调用被代理类的方法和属性的类。需要实现InvocationHandler

4、动态代理的应用场景

> 1. 包装RPC调用(Remote Programe Call---远程程序调用)
> 2. 面向切面的编程(AOP)
> 3. 框架内部的寻址、序列化、反序列化

5、动态代理解决了什么问题？在具体业务中有哪些应用场景?

> 1. 解决了调用者和实现者， 耦合度过高 的问题。
> 2. 进行RPC调用，Android中的AIDL、Binder
> 3. 框架内部的寻址、序列化、反序列化等内容，与调用者解耦。

6、动态代理的实现方法?

> 1. JDK自身提供的动态代理：主要就是利用了反射机制。 `JDK Proxy`

> 2. 利用字节码操作机制：性能更高，类似 `ASM`、`cglib(基于ASM)`、`Javasist` 等。

7、JDK动态代理在设计和实现上与cglib等方式有什么不同?

> 1. JDK Proxy: 使用反射和部分字节码操作来实现。
> 2. cglib使用字节码操作机制来实现。

8、JDK Proxy比cglib和Javassist慢几十倍?

> 并没有这么大的差距。
>
> 1. 主流JDK版本中，JDK Proxy在大部分场景可以提供对等的性能水平。
> 2. 反射机制性能在现代JDK中，已经进行了极大的改进和优化。已经没有过高的性能损耗。
> 3. 此外JDK很多功能也不全是反射，也采用了 `ASM` 进行字节码操作。

9、JDK Proxy和cglib如何进行取舍?

> 1. 性能并不是唯一的标准，
> 2. 可靠性、可维护性、工作量也是重要的因素。
> 3. 可靠性上： 推荐JDK Proxy
> 4. 编码工作量、性能上：推荐cglib

# JDK Proxy

10、JDK动态代理的实例

```
public class Main {
    // 抽象类接口
    public interface Animal {
        void cry();
    }
    // 被代理类：具体实现抽象接口
    public static class Dog implements Animal {
        @Override
        public void cry() {
            System.out.println("汪汪！");
        }
    }
    // 动态代理类：实际调用被代理类的方法和属性的类。需要实现InvocationHandler
    static class MyInvocationHandler implements InvocationHandler{
        private Object target;
        public MyInvocationHandler(Object target){
            this.target = target;
        }

        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            System.out.println("invoking cry()");
            Object result = method.invoke(target, args);
            return result;
        }
    }

    public static void main(String[] args) {
        // 1. 创建被代理者
        Dog dog = new Dog();
        // 2. 实例化Handler： 用于插入额外的逻辑
        MyInvocationHandler handler = new MyInvocationHandler(dog);
        // 3. 构造代理，实例化的是Proxy对象。
        Animal animal = (Animal) Proxy.newProxyInstance(Dog.class.getClassLoader(),
                Dog.class.getInterfaces(),
                handler);

        // 4. 调用代理方法
        animal.cry();
    }
}
```

## 11、JDK Proxy的作用

> 1. 使用代理模式，能通过代理去间接执行目标的逻辑。
> 2. 此外使用代理，能够在执行被代理者的逻辑之间，去添加额外的逻辑。

## 12、JDK Proxy的缺点

> 1. 被调用者(如Dog)必须要去实现被代理的接口(如Animal)
> 2. 如果被调用者，没有实现接口，就无法使用JDK Proxy了，这时候可以使用 `cglib`

13、JDK Prxoy的优势

> 1. 最小化依赖关系。减少依赖意味着简化开发和维护。
> 2. 兼容性强：会随着JDK版本进行升级。但是字节码类库通常需要更新，以保证在新版Java上能够使用。
> 3. 代码实现简单。

## cglib

14、cglib框架的优势

> 1. 侵入性低。cglib动态代理不需要调用者实现接口。
> 2. 更专注。只操作关心的类，不需要额外工作量。
> 3. 高性能

# 知识扩展

1、面向切面编程AOP是什么?

> 1. AOP是OOP的一种补充。
> 2. 比如在一种场景中：业务A，业务B，业务C都有日志、安全等功能。但是A、B、C的日志可以统一进行处理，安全部分可以采用通用的安全策略、
> 3. AOP通过动态代理机制，大幅度提高了代码的抽象程度和复用度。减少繁琐的工作量。
> 4. `Facade`、`Observer` 等设计模式都可以通过动态代理进行优雅的实现。

2、字节码操作机制是什么?

3、ASM是什么？Javasist是什么?

# 参考资料

1. Java 反射详解(68题)