链滴

# 关于对象和 this

作者：luofeng0603
原文链接：https://ld246.com/article/1681873293390
来源网站：链滴

# 前言

最新看到一篇关于对象和this的分析，挺有意思的，结合自己的经验，浅浅理解了一下，并做一下记。

# Self和this

灵感来自于python，python中定义方法，都会默认给个self，比如：

```
def __init__(self, name, age):
  self.__name = name
  self.__age = age
```

这个self其实是实例的引用，相当于是java中的this，不过java中通常是隐式的。

但是通过观察python的代码，我们可以发现，我们通过实例调用方法的时候，其实除了给需要的参外，我们还需要将当前实例的引用传过去。只不过在java中这个引用不需要我们手动传。

那么为什么一定要传这个实例的引用呢？能不能不传呢？

答案是不能，至于为什么？那跟java的设计有关系了。

我问了AI几个问题，一起来看一下：




对象是被创建到堆中的，但是好像没有方法？没有方法，那我们能调用对象的方法呢？

噢~，方法属于类，就是公共的，只有一份，jvm把方法存放在了方法区。

看第一句描述，每个对象都有一个指向它所属类的指针，这个指针指向该类的方法区的方法表。这就我们使用 对象.方法 能正确执行的原因~

简单总结一下：方法是属于类的，不是属于对象的，所以只有一份。jvm创建的对象只包含对象头、例数据，并不包含方法，类中定义的方法存放在方法区，每个对象都有一个指向它所属的类指针（就this呗~），这个指针指向了该类的方法区的方法表，当对象调用方法的时候，会去方法表中查找对的方法，然后把方法加载到内存中执行。

再来看上面的问题，因为类的方法是共用的，调用方法的时候是不是需要java将this（this代表实例的用）传递过去啊？你不传过去的话，jvm怎么知道这个方法是a对象调用还是b对象调用的？

# Static与this

static修饰的属性和方法是属于类的，是所有实例共享的。我们反过来想想，问什么要把属性或者方定义为static？

- static变量：大家共有的，大家都一样
- static方法：这个方法不处理差异化数据

也就是说static注定与差异化无关，即与具体对象的数据无关。

以静态方法为例，当我们确定一个方法只处理通用流程，而与实例无关时，我们就可以把它设置为静方法，比如我们经常写的工具类，里面的方法都是静态方法，因为跟具体的实例无关嘛~

# 关于this的一个小例子

```
public class Demo {
    public static void main(String[] args) {
        Son son = new Son();
        Daughter daughter = new Daughter();
    }
}

class Father {
    public Father() {
        System.out.println(this.getClass().getName());
    }
}

class Son extends Father {
}

class Daughter extends Father {
}
```

结果如下：

```
com.example.test.Son
com.example.test.Daughter
```

为什么？父类创建的时候，其实根据不知道有哪些子类，但是却在父类中得到了子类的名字。

我们知道子类实例化的时候会隐私调用父类构造器，所以上面的代码等同于如下：

```
class Son extends Father {
    public Son() {
        super();
    }
}

class Daughter extends Father {
    public Daughter() {
        super();
    }
}
```

所以其实就是调用方法调用罢了， 调用的时候也隐式传入了this，new Son() 的时候，调用了super()，其实传入了this，就是Son实例的引用，然后父类里面的this 其实就是 Son的实例，自然就能打印儿子的名称了，Daughter同理。

# 另一个小例子

```
public class ThisDemo {

    public static void main(String[] args) {
        // TODO
    }


    @Getter
    @Setter
    @Accessors(chain = true)
    static class Father {
        private String fatherName;
    }

    @Getter
    @Setter
    @Accessors(chain = true)
    static class Son extends Father {
        private String sonName;
    }
}
```

**问题一：可以new Son().setSonName("").setFatherName()，却不能new Son().setFatherNam("").setSonName()，为什么?**

**问题二：无论怎么调整setter顺序，返回值始终是Father类型，为什么?**

```
Father father = new Son("").setSonName("").setFatherName("");
```

Father father = new Son("").setFatherName("");

@Getter、@Setter、@Accessors 是lombook的注解， 其中@Accessors用来实现链式调用。

我们来看下IDEA编号的代码：

```
public class Demo2 {
    public Demo2() {
    }

    public static void main(String[] args) {
        Demo2.Father father = (new Demo2.Son()).setSonName("大头儿子").setFatherName("小
爸爸");
        System.out.println(father);
    }

    static class Son extends Demo2.Father {
        private String sonName;

        Son() {
        }

        public String getSonName() {
            return this.sonName;
        }

        public Demo2.Son setSonName(final String sonName) {
            this.sonName = sonName;
            return this;
        }
    }

    static class Father {
        private String fatherName;

        Father() {
        }

        public String getFatherName() {
            return this.fatherName;
        }

        public Demo2.Father setFatherName(final String fatherName) {
            this.fatherName = fatherName;
            return this;
        }
    }
}
```

看一下 setSon 和 setFather方法的返回值

回顾下我们上面的分析，调用方法的时候，其实会隐式传递this（就是当前实例的引用），所以：

new Son().setSonName("") 后是Son类型的，自然可以调用父类的方法

new Son().setFatherName("")后是Father类型的，自然调用不了子类的方法