# 类实例域使用情况检测工具

## Why

1. 如果一个项目是**设计不合理**的项目，那么类的各个实例域的使用情况可以帮助设计者设计出更好的类结构。
2. 如果一个项目是**设计优秀**的项目，那么类的各个实例域的利用率应该会比较高，如果出现了某个**Field利用率低**的情况，项目中可能存在**bug**。
3. 如果一个类的某个域的被**频繁的使用**，这个Field可能涉及复杂的操作，相关部分可能是**脆弱**的。

## What

```
class  Student extends People {
  int id;
  String name;

  Student(){

  }

  void setId(int id){
    this.id = id;
  }

  Int getId(){
    return this.id;
  }

}
```

针对以上Student类，我们希望获得如下信息

```
{
    "className" : 该类的名称 => Student

    "instanceCount" :  在整个运行过程中，该类创建的实例数

    "fields" : [
        {
            "fieldName" : 该类的域的名称 => id

            "putCount" : 所有Student实例赋值id的总次数，

            "setCount" : 所有Student实例更改id的总次数，

            "putInstanceCount" : 所有Student实例中赋值过id的instance数量，
```

```
            "setInstanceCount" : 所有Student实例中更改过id的instance数量,
        },
        {
            "fieldName" : 该类的域的名称 => name

            "putCount" : 所有Student实例赋值name的总次数,

            "setCount" : 所有Student实例更改name的总次数,

            "putInstanceCount" : 所有Student实例中赋值过name的instance数量,

            "setInstanceCount" : 所有Student实例中更改过name的instance数量,
        }

    ]
}
```

**以上如同Student类一样用户关注的类称为关注类**

## How

**利用插桩的方法进行动态分析来收集**

主要进行三种插桩：插入新的域，插入新的方法、在方法中插入新的指令

**插入新的域**

- 插入脏位数组：用于防止`InstanceCount`的重复计数

**插入新的方法**

- 插入`putField`方法，用于收集对域的**更改**操作
- 插入`getField`方法，用于收集对域对**赋值**操作
- 插入`newInstance`方法，用于收集对象的**创建**操作
- 插入`descInstance`方法，用于解决对父类实例计数的**误增**

**插入新的指令**

- 在**构造函数**中，插入对本类的`newInstance`方法的调用，插入对父类的`descInstance`方法的调用，以及插入**对脏位数组的初始化**操作。

  **（对脏位数组的初始化操作一定要在`super`前面，也就是最开头，因为在子类构造函数中调用父类 构造函数时，父类构造函数的`this`指针指向的是子类实例，如果父类构造函数中修改了域，会调用子类的`putField`和`getField`方法，方法内用到的脏位数组还没有在子类构造函数中初始化的话就会出错）**。

- 在**原有的所有方法**中，在对操作数是**关注类**的域的`putField`指令后插入对该类的`putField`方法的调用，在对操作数是**关注类**的域的`getField`指令。

插桩后的Student

```
class Student  extends People {
  int id;
  String name;
  int[] getDirtyTag;
  int[] putDirtyTag;

  Student(){
    this.putDirtyTag = new int[2];
    this.getDirtyTag = new int[2];
    super();
    InfoCollection.incInstance(“Student”);
    InfoCollection.descInstance(“People”);
  }

  void setId(int id){
    this.id = id;
    this.putField(“id”); //增加field的赋值计数
  }

  int getId(){
    this.getField(“id”);//增加field的使用计数
    return this.id;
  }

  void getField(String fieldName) {
      if(fieldName == "id"){
          if (this.putDirtyTag[0] == 1) {
              InfoCollector.putField("Student", "id", false);
          } else {
              InfoCollector.putField("Student", "id", true);
              this.putDirtyTag[0] = 1;
          }
      }
      else if(fieldName == "name"){
          if (this.putDirtyTag[1] == 1) {
              InfoCollector.putField("Student", "name", false);
          } else {
              InfoCollector.putField("Student", "name", true);
              this.putDirtyTag[1] = 1;
          }
      }
  }

}
```

## Build

1. 将`tool project`编译成一个`tool.jar`包。（本质是一个JavaAgent，需要带上Maven生成的包）
2. 编写一个名称为`UserConfig.txt`的文本文件，文件的每行是**关注类.class**的路径。

3. 使用 `java -javaagent:JarPath=UserConfigPath:EnviorenmentConfigPath:Mode:StackSize -Xbootclasspath/a:JarPath Java项目`来执行，其中`JarPath`是`tool.jar`的绝对路径，`UserConfigPath`是`UserConfig.txt`的绝对路径,`EnviorenmentConfigPath`是`EnviorenmentConfig`的绝对路径，`Mode`是FieldDetection或者ConstructSite,`StackSize`是输出的callsite的层数。