**Notes on JVM Profiling**

Each method encapsulates method data object (MDO), during method parsing MDO (MethodData::initialize) skeleton is created which has a blank profile cell for each bytecode of method.

DataLayout holds an array (\_cell) of intptr where each entry points to a profile item. A profile item may be a Klass points or a counter or taken / not-taken count etc.

These blank cells are filled by Interpreter and C1 profile modes (2,3), C2 is only a profile consumer.

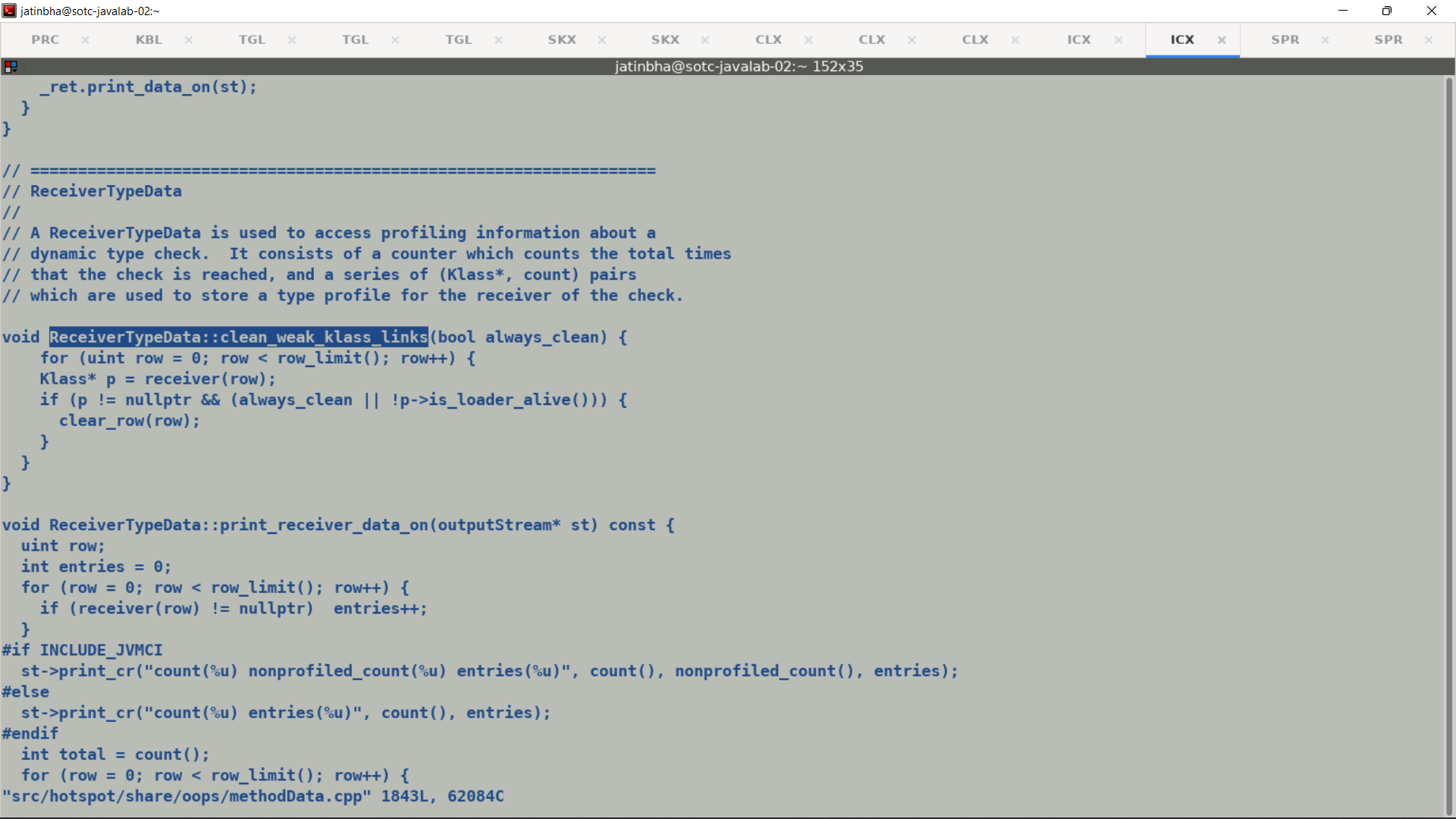
Profile data is not accurate and due to data races while updating the MDO can have inaccurate information, but again any optimization based on profile also accompanies emission of a constraint in JIT code and if those constraints are not met then compilation is made non-reentrant and method is up for re-compilation.

While MDO is a much granular profile, JVM also keeps a minimal profile information i.e., back edge taken count and method invocation count.

Following flags are used to tune JVM profile collection.

Profiling always uses saturating counters.

|  |
| --- |
|  |
| **TraceTypeProfile** | **OpenJDK6** |  | **bool** |  |  | **c2** | **False** | **product** | **Trace type profile** | **share/opto/c2\_globals.hpp** |
| **TypeProfileArgsLimit** | **OpenJDK8** |  | **intx** |  |  | **runtime** | **2 range(0, 16)** | **product** | **max number of call arguments to consider for type profiling** | **share/runtime/globals.hpp** |
| **TypeProfileCasts** | **OpenJDK6** |  | **bool** |  |  | **runtime** | **True** | **develop** | **treat casts like calls for purposes of type profiling** | **share/runtime/globals.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **aarch64** |  | **111** | **define\_pd\_global** |  | **cpu/aarch64/globals\_aarch64.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **arm** |  | **0** | **define\_pd\_global** |  | **cpu/arm/globals\_arm.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uint** |  |  | **runtime** |  | **product\_pd** | **=XYZ, with Z: Type profiling of arguments at call;**  **Y: Type profiling of return value at call;**  **X: Type profiling of parameters to methods;**  **X, Y and Z in 0=off ; 1=jsr292 only; 2=all methods** | **share/runtime/globals.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **ppc** |  | **111** | **define\_pd\_global** |  | **cpu/ppc/globals\_ppc.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **riscv** |  | **111** | **define\_pd\_global** |  | **cpu/riscv/globals\_riscv.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **s390** |  | **111** | **define\_pd\_global** |  | **cpu/s390/globals\_s390.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **x86** |  | **111** | **define\_pd\_global** |  | **cpu/x86/globals\_x86.hpp** |
| **TypeProfileLevel** | **OpenJDK8** |  | **uintx** |  | **zero** |  | **0** | **define\_pd\_global** |  | **cpu/zero/globals\_zero.hpp** |
| **TypeProfileMajorReceiverPercent** | **OpenJDK6** |  | **intx** |  |  | **c2** | **90 range(0, 100)** | **product** | **% of major receiver type to all profiled receivers** | **share/opto/c2\_globals.hpp** |
| **TypeProfileParmsLimit** | **OpenJDK8** |  | **intx** |  |  | **runtime** | **2 range(-1, 64)** | **product** | **max number of incoming parameters to consider for type profiling**  **, -1 for all** | **share/runtime/globals.hpp** |
| **TypeProfileWidth** | **OpenJDK6** |  | **intx** |  |  | **runtime** | **2 range(0, 8)** | **product** | **Number of receiver types to record in call/cast profile** | **share/runtime/globals.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **aarch64** |  | **false** | **define\_pd\_global** |  | **cpu/aarch64/c1\_globals\_aarch64.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **arm** |  | **false** | **define\_pd\_global** |  | **cpu/arm/c1\_globals\_arm.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  |  | **compiler** | **false** | **define\_pd\_global** |  | **share/compiler/compiler\_globals\_pd.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  |  | **runtime** | **True** | **product** | **Check interpreter profile for historically monomorphic calls** | **share/runtime/globals.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **ppc** |  | **false** | **define\_pd\_global** |  | **cpu/ppc/c1\_globals\_ppc.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **riscv** |  | **false** | **define\_pd\_global** |  | **cpu/riscv/c1\_globals\_riscv.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **s390** |  | **false** | **define\_pd\_global** |  | **cpu/s390/c1\_globals\_s390.hpp** |
| **UseTypeProfile** | **OpenJDK6** |  | **bool** |  | **x86** |  | **false** | **define\_pd\_global** |  |  |



When GC sweeps the Meta Data for Method alongside it also sweeps out its MDO, anyways MDO is part of method.

Example.

CPROMPT>cat test.java

abstract class BaseObj {}

class DerivedObj1 extends BaseObj {}

class DerivedObj2 extends BaseObj {}

class DerivedObj3 extends BaseObj {}

class DerivedObj4 extends BaseObj {}

public class test {

public static int micro1(BaseObj obj) {

if (obj.getClass() == DerivedObj1.class) {

return ((DerivedObj1)obj).hashCode();

} else if (obj.getClass() == DerivedObj2.class) {

return ((DerivedObj2)obj).hashCode();

} else if (obj.getClass() == DerivedObj3.class) {

return ((DerivedObj3)obj).hashCode();

} else if (obj.getClass() == DerivedObj4.class) {

return ((DerivedObj4)obj).hashCode();

} else {

return -1;

}

}

public static void main (String [] args) {

int res = 0;

BaseObj [] oarr = {new DerivedObj1(), new DerivedObj2(), new DerivedObj3(), new DerivedObj4()};

for (int i = 0; i < 10000; i++) {

res += micro1(oarr[i & 3]);

}

System.out.println(res);

}

}

A screenshot of a computer program

Description automatically generated