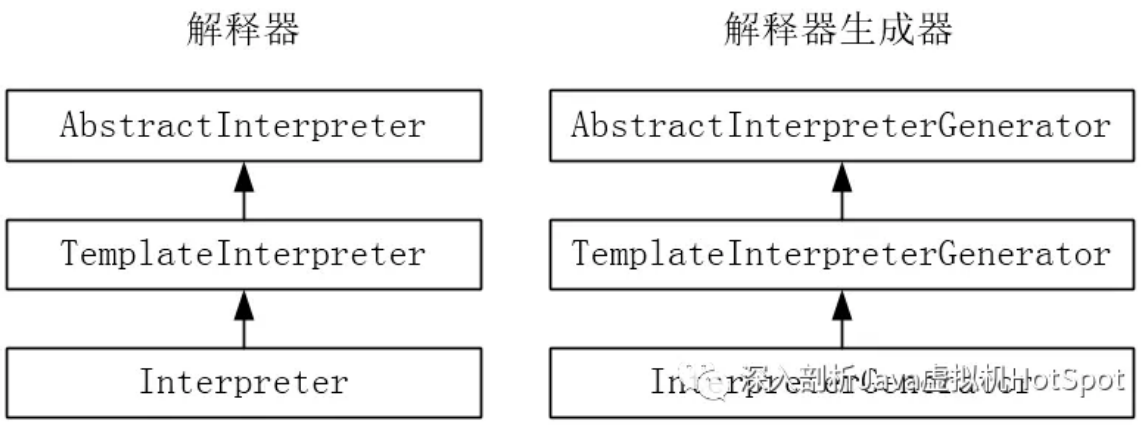
第15章-解释器及解释器生成器

Original 马智 深入剖析Java虚拟机HotSpot Yesterday

方法解释执行时需要解释器与解释器生成器的支持。解释器与解释器生成器的继承体系如下：



下面详细介绍解释器与解释器生成器。

1、解释器

解释器是一堆本地代码例程构造的，这些例程会在虚拟机启动的时候写入到StubQueue中，以后解释执行时就只需要进入指定例程即可。

解释器的继承体系如下：

```
AbstractInterpreter        /interpreter/abstractInterpreter.hpp
     CppInterpreter
     TemplateInterpreter   /interpreter/templateInterpreter.hpp
           Interpreter     /interpreter/templateInterpreter.hpp
```


Interpreter通过宏可以继承自CppInterpreter或者TemplateInterpreter，前者称为C++解释器，每个字节码指令都对应一段C++代码，通过switch的方式处理字节码，后者称为模板解释器，每个指令对应一段机器指令片段，通过指令模板的方式处理字节码，HotSpot VM默认使用模板解释器。

（1）抽象解释器AbstractInterpreter

所有的解释器都继承自抽象解释器，类及重要属性的定义如下：

```
class AbstractInterpreter{
   StubQueue* _code
   address    _entry_table[n];
   // ...
};
```


_code属性在之前已经介绍过，这是一个队列，队列中的InterpreterCodelet表示一个例程，比如iconst_1对应的代码，invokedynamic对应的代码，异常处理对应的代码，方法入口点对应的代码，这些代码都是一个个InterpreterCodelet。整个解释器都是由这些例程组成的，每个例程完成解释器的部分功能，以此实现整个解释器。

_entry_table数组会会保存方法入口点，例如普通方法的入口点为_entry_table[0]、同步的普通方法的入口点为_entry_table[1]，这些_entry_table[0]，_entry_table[1]指向的就是之前_code队列里面的例程。这些逻辑都在是generate_all()函数中完成的，如下：

```
void TemplateInterpreterGenerator::generate_all() {
  // ...

  method_entry(zerolocals)
  method_entry(zerolocals_synchronized)
  method_entry(empty)
  method_entry(accessor)
  method_entry(abstract)
  method_entry(java_lang_math_sin  )
  method_entry(java_lang_math_cos  )
  method_entry(java_lang_math_tan  )
  method_entry(java_lang_math_abs  )
  method_entry(java_lang_math_sqrt )
  method_entry(java_lang_math_log  )
  method_entry(java_lang_math_log10)
  method_entry(java_lang_math_exp  )
  method_entry(java_lang_math_pow  )
  method_entry(java_lang_ref_reference_get)

  // ...
}
```

method_entry宏的定义如下：

```
#define method_entry(kind)                                              \
{                                                                       \
    CodeletMark cm(_masm, "method entry point (kind = " #kind ")");     \
    Interpreter::_entry_table[Interpreter::kind] = generate_method_entry(); \
}
```


可以看到，调用generate_method_entry()函数会返回例程对应的入口地址，然后保存到AbstractInterpreter类中定义的_entry_table数组中。调用generate_method_entry()函数传入的参数是枚举常量，表示一些特殊的方法和一些常见的方法类型。

（2）模板解释器TemplateInterpreter

模板解释器类的定义如下：

```
class TemplateInterpreter: public AbstractInterpreter {
  protected:
    // 数组越界异常例程
    static address    _throw_ArrayIndexOutOfBoundsException_entry;
    // 数组存储异常例程
    static address    _throw_ArrayStoreException_entry;
    // 算术异常例程
    static address    _throw_ArithmeticException_entry;
    // 类型转换异常例程
    static address    _throw_ClassCastException_entry;
    // 空指针异常例程
    static address    _throw_NullPointerException_entry;
    // 抛异常公共例程
    static address    _throw_exception_entry;

    // ...
}
```


抽象解释器定义了必要的例程，具体的解释器在这之上还有自己的特设的例程。模板解释器就是一个例子，它继承自抽象解释器，在那些例程之上还有自己的特设例程，例如上面定义的一些属性，保存了程序异常时的入口例程，其实还有许多为保存例程入口而定义的字段或数组，这里就不一一介绍了。

（3）解释器Interpreter

类的定义如下：

```
class Interpreter: public CC_INTERP_ONLY(CppInterpreter) NOT_CC_INTERP(TemplateInterpreter) {
  // ...
}
```


没有定义新的属性，只有几个函数。Interpreter默认通过宏扩展的方式继承TemplateInterpreter。

2、解释器生成器

要想得到可运行的解释器还需要解释器生成器。解释器生成器本来可以独自完成填充工作，可能为了解耦，也可能是为了结构清晰，HotSpot VM将字节码的例程抽了出来放到了TemplateTable模板表中，它辅助模板解释器生成器templateInterpreterGenerator生成各种例程。

解释器生成器的继承体系如下：

```
AbstractInterpreterGenerator        /interpreter/abstractInterpreter.hpp
     TemplateInterpreterGenerator   /interpreter/templateInterpreter.hpp
               InterpreterGenerator  /interpreter/interpreter.hpp
```


模板解释器生成器扩展了抽象解释器生成器。解释器生成器与解释器其实有某种意义上的对应关系，如抽象解释器生成器中定义了一些函数，调用这些函数会初始化抽象解释器中的属性，如保存例程的_entry_table数组等，在模板解释器生成器中定义的函数会初始化模板解释器中定义的一些属性，如_throw_ArrayIndexOutOfBoundsException_entry等。之前介绍过空指针的例程就是在这个TemplateInterpreterGenerator类的generate_all()函数中生成的。如下：

```
{
    CodeletMark cm(_masm, "throw exception entrypoints");
    // ...
    Interpreter::_throw_NullPointerException_entry = generate_exception_handler("java/lang/NullPointerException",NULL);
    // ...
}
```


关于解释器生成器不再过多介绍。

这里我们需要提醒的是，解释器和解释器生成器中定义的函数在实现过程中，有些和平台是无关的，所以会在/interpreter文件夹下的文件中实现。例如Interpreter和InterpreterGenerator类定义在/interpreter文件夹下，其中定义的函数会在/interpreter文件夹下的interpreter.cpp文件中实现，但是有些函数是针对特定平台，我们只讨论linux在x86架构下的64位实现，所以cpu/x86/vm文件夹下也有interpreter_x86.hpp和interpreter_x86_64.cpp等文件，只需要在定义Interpreter类时包含interpreter_x86.hpp文件即可。

关注公众号 “深入剖析Java虚拟机HotSpot”

深入剖析Java虚拟机HotSpot

对HotSpot VM进行深度源码剖析，如果要系统的学习相关内容，推荐作者的《深入剖析J...

18篇原创内容

Official Account

People who liked this content also liked

第8篇-dispatch_next()函数分派字节码

深入剖析Java虚拟机HotSpot

第7篇-为Java方法创建栈帧

深入剖析Java虚拟机HotSpot

第9篇-字节码指令的定义

深入剖析Java虚拟机HotSpot