**erts的bif交互实现的基本分析**

**author:wangweihuai**

版本基于R15B03分析

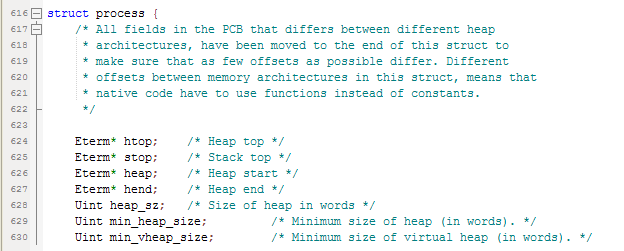
1， 基础数据类型

1.2 Process

erts/emulator/beam/erl\_process.h

C:\Documents and Settings\Administrator\Local Settings\Application Data\youdao\ynote\images\AB93E7C2DA8F4B959FCB7512FD2E5AFD\clipboard.png

616-783行定义一个完整的struct process，包括完整的pcb结构，下图只是一部分



1.2  Eterm

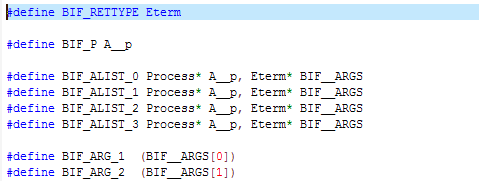
    erts/emulator/beam/sys.h

C:\Documents and Settings\Administrator\Local Settings\Application Data\youdao\ynote\images\2FB07531D5DC450486E30D03D89EEFEC\clipboard.png

在进程堆栈中的所有数据，都可以通过Eterm来表示

1.3 对BIF\_ALIST\_ 和 BIF\_ARG\_的定义

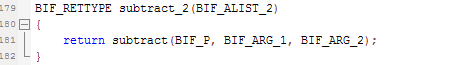
erts/emulator/beam/bif.h



BIF\_ALIST\_实际传递的是一个Process\* 和 Eterm\* 两个参数，跟在BIF\_ALIST\_后面表示的就是实际erlang函数传递给bif的变量的个数。

在函数体内部则是通过BIF\_ARG\_\*（34行-36行）的宏定义读取，通过对首位参数地址的偏移来获得参数的指针。

所有的参数对应都是一个指针，具体是那个怎么为需要的类型，可以跟踪list\_val验证，我现在并没有跟进，个人估计对应进程（Process）堆栈中的一个位置，通过某些位来标记类型和对应的长度来获得具体的数据。下面以erts/emulator/beam/erl\_bif\_lists.c中的substract\_2为例子



简单展开后为

Eterm subtract\_2(Process\* A\_\_P, Eterm\* BIF\_ARGS)

{

    return subtrace(A\_\_P,  BIF\_\_ARGS[0],  BIF\_ARGS[1]);

}

这里表示的是erlang代码中调用bif是传递两个参数，在bif内部通过BIF\_ARGS[0]和BIF\_ARGS[1]来索引读取

2. bif的c代码是怎么和erlang代码关联的

在erts/emulator/beam/bif.tab文件，里面定义了所有的导出的bif文件：

例如lists的bif：

#

# Bifs in lists module.

#

bif lists:member/2

bif 'erl.lang.list':is\_element/2 ebif\_list\_is\_element\_2 lists\_member\_2

bif lists:reverse/2

bif 'erl.lang.list':reverse/2 ebif\_list\_reverse\_2 lists\_reverse\_2

bif lists:keymember/3

bif 'erl.lang.list.keylist':is\_element/3 ebif\_keylist\_is\_element\_3 lists\_keymember\_3

bif lists:keysearch/3

bif 'erl.lang.list.keylist':search/3 ebif\_keylist\_search\_3 lists\_keysearch\_3

bif lists:keyfind/3

bif 'erl.lang.list':subtract/2 ebif\_list\_subtract\_2 ebif\_minusminus\_2

说明如下：

# <bif-decl> ::= "bif" <bif> <C-name>\* | "ubif" <bif> <C-name>\*

# <bif> ::= <module> ":" <name> "/" <arity>

#

# "ubif" is an unwrapped bif, i.e. a bif without a trace wrapper,

# or rather; the trace entry point in the export entry is the same

# as the normal entry point, and no trace wrapper is generated.

#

# Important: Use "ubif" for guard BIFs and operators; use "bif" for ordinary BIFs.

#

# Add new BIFs to the end of the file. Do not bother adding a "packaged BIF name"

# (such as 'erl.lang.number'); if/when packages will be supported we will add

# all those names.

#

# Note: Guards BIFs require special support in the compiler (to be able to actually

# call them from within a guard).

在makefile文件中（erts/emulator/Makefile.in）存在

BIFS = beam/bif.tab

这样一个编译项，使得编译后的bif与存在对应的模块中，可以和erlang实现的对应模块，可被虚拟机识别和调用，红色部分例子

erlang中erl.lang.list:subtract绑定的就是ebif\_list\_substrac\_2这个函数。