在Cling上实现空指针解引用检测机制

丁保增

中科院软件研究所

baozeng@nfs.iscas.ac.cn

2013年11月16日

ISCAS

Overview

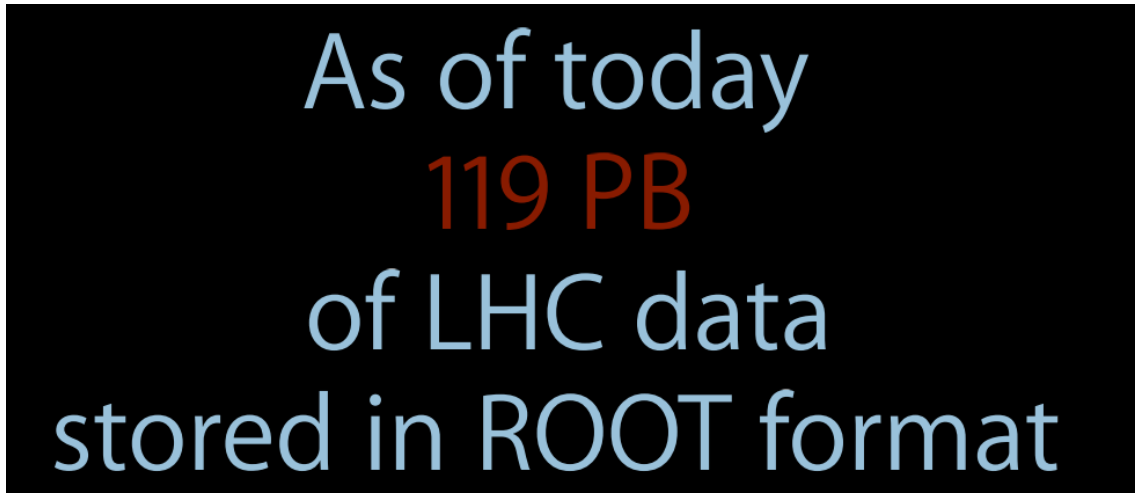
1. Cling介绍
   - Cling的来源
   - Cling的应用
2. Cling的原理
3. Cling上实现空指针解引用检测机制

CERN

欧洲核子研究组织CERN拥有世界上最大的粒子物理实验室，其大型强子对撞机LHC 也是当今最大的粒子加速设施。

ROOT

CERN使用自己开发的ROOT 软件来分析、处理和存储数据。ROOT使用C++作为开发语言，迄今为止CERN所使用和开发的C++的代码量约有5千万行。



CINT

ROOT的大多数用户并非是计算机专家，因此为了给他们一个快速开发应用的环境，ROOT 使用了一个C++解释器CINT，来解释执行C++代码. 用户可以使用CINT快速地开发应用(RAD)，由于不需要编译和链接，节省了不少的开发时间。

即时编译器Cling的出现

由于CINT设计时并没有考虑模块化，并且不支持新的C++标准C++11. 因此Cling 应运而生，取代了CINT.


```
Error: Can't call map<string, const  char*,less<string>, allocator<const
string,const char*> > >::operator[]((char*)0x255a9e8) in current scope
err.C:19:
Possible candidates are...
(in map<string,const char*,less<string>,allocator<const string,const
char*> > >)
Error: improper lvalue err.C:19
```

```
err.C:19:15: error: assigning to 'mapped_type' (aka 'const char*') from
incompatible type 'double'
    myMap["A"]  = 12.3;
                  ^ ~~~
```

Cling的使用方式(一)

Cling有下列使用方式：

- 直接输入C/C++语句

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <iostream>
[cling]$ using namespace std;
[cling]$ cout << "Hello world!" << endl;
Hello world!
```

Cling的使用方式(二)

- .L/.U library 加载/卸载库
- .I path 添加一个include 路径
- .printAST 打印抽象语法树AST
- .printIR 打印中间表示IR

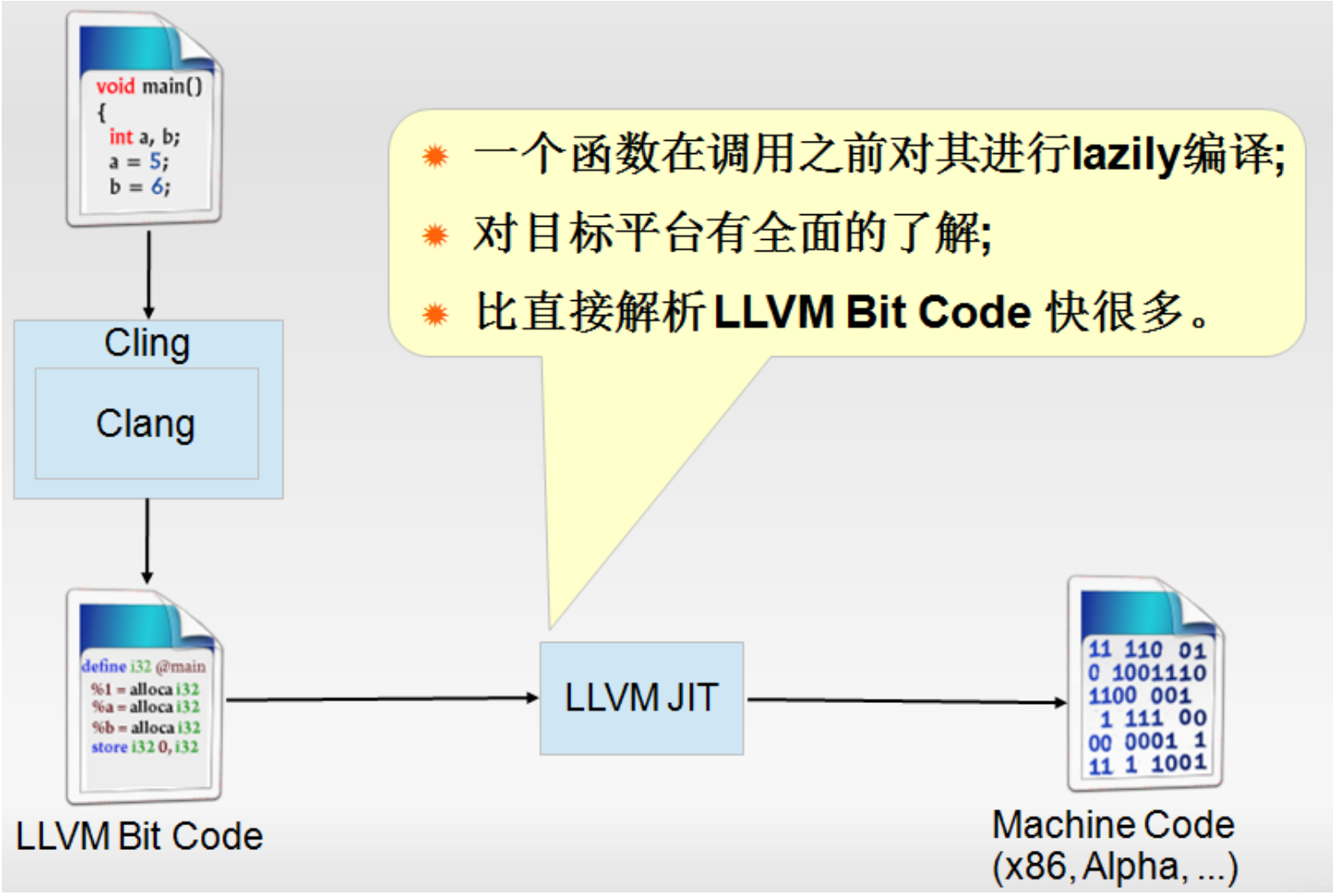
Cling的应用

Cling 用于下面几个方面：

- 用在快速应用开发中：Cling 与ROOT
- 让用户方便地调用C++库里面的函数，快速地学习和熟悉C++库的功能：Cling 与OpenGL (Demo), Cling 与Qt.

Cling的原理

Cling是基于LLVM和Clang来实现的，其在运行时将LLVM中间表示IR编译成可执行代码，实现了即时编译。



错误恢复

Cling将用户的一次输入称作一个transaction,当transaction中有错误时，那么整个输入是无效的。Cling将验证用户的输入，然后将其送到缓存队列中来生成代码。如果有个非法的，那么Cling将会丢掉该缓存并且修复AST。

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ int x; ERROR_HERE; int j;
In file included from -:1:
input_line_4:2:9: error: use of undeclared identifier 'ERROR_HERE'
 int x; ERROR_HERE; int j;
        ^

[cling]$ i
In file included from -:1:
input_line_5:2:2: error: use of undeclared identifier 'i'
 i
 ^
```

表达式处理

```
[cling]$ cout << “Hello\n”;
```

```
void wrapper() {
  cout << “Hello\n”;
}
```

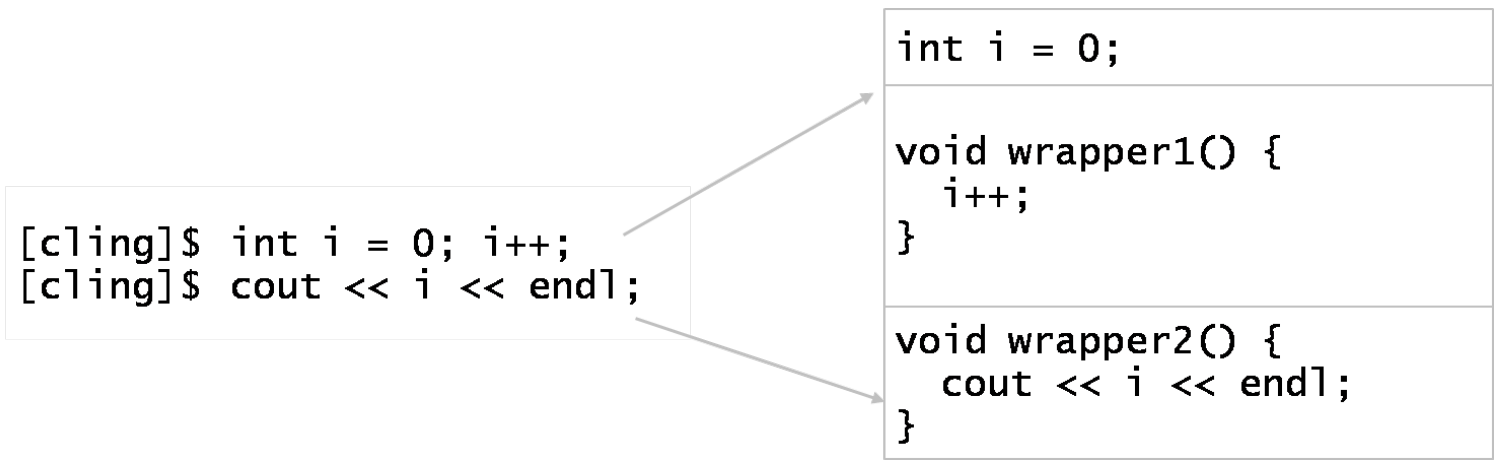
```
[cling]$ int i = 0; i++;
[cling]$ cout << i << endl;
```

```
void wrapper1() {
  int i = 0; i++;
}
```

```
void wrapper2() {
  cout << i <<endl;
}
```

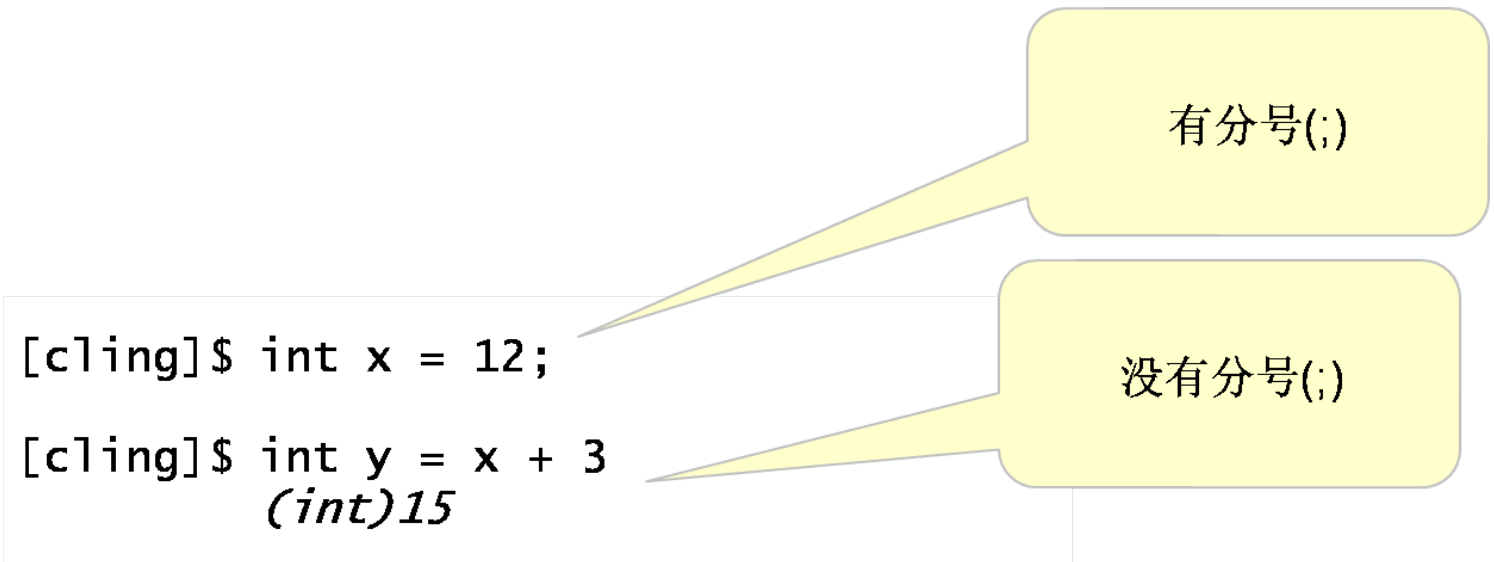
表达式处理

- 封装输入；
- 查找声明；
- 将声明向上提一个层次，作为全局的声明。



打印结果

用户输入一条语句后，有的希望看到执行后结果，有的则不希望看到该语句的执行结果。Cling通过用户在输入语句后面是否带有分号来判断是否需要输出结果。如果有分号,则不输出结果；如果没有则输出结果。



代码卸载

Cling支持代码卸载：当用户加载一个文件脚本，测试它的功能，发现和预期的不一致，那么可以卸载该文件，并且修复，然后重新加载。

```
[cling]$  .L Calculator.h
[cling]$  Calculator calc;
[cling]$  calc.Add(3, 1)
[cling]$  (int) 2  //???
[cling]$  .U Calculator.h
[cling]$  .L Calculator.h
[cling]$  Calculator calc;
[cling]$  calc.Add(3, 1)
[cling]$  (int) 4  //☺
```

```
// Calculator.h
class Calculator {
  int Add(int a, int b) {
    return a - b;
  }
...
};
```

```
// Calculator.h
class Calculator {
  int Add(int a, int b) {
    return a + b;
  }
...
};
```

空指针解引用

Cling上出现空指针解引用时，就会出现core dump现象，对用户不友好。

```
[cling]$ #include <cstring>
[cling]$ char *p = 0;
[cling]$ strcmp(p, “aa”);
Segmentation fault (core dumped)
```

方案

在一个指针解引用之前，插入一个检查语句，如果该指针是NULL,则抛出一个异常。期望的结果如下：

```
[cling]$ #include <cstring>
[cling]$ char *p = 0;
[cling]$ strcmp(p, “aa”);
 warning: null passed to a callee which requires a non-null
argument.
```

如何实现

两种方式：在AST或在IR上实现。由于在IR上插入异常的语句实现起来比较复杂，选择AST上实现。

检查下面的AST节点：

- ✓ UnaryOperator //*p
- ✓ MemberExpr //myclass->f
- ✓ BinaryOperator // *p + *q
- ✓ CastExpr() // int(*p)
- ✓ CallExpr // f() __attribute__((nonnull));

NonNull 参数

如果函数的参数有nonnull属性，那么该函数的这个参数就不能是NULL.比如：

```
1  extern void * my_memcpy(void *dest, const void *src,  size_t len )
                __attribute__ ((nonnull (1, 2)));
```

Cling将检查dest, src都必须不为空。

AST变换

在变换之前：

```
[cling]$ strcmp(p, "aa");

------------------Declaration--------------------
FunctionDecl 0x4bbc480 <input_line_8:1:1, line:4:1> __cling_Un1Qu33 'void (void)'
`-CompoundStmt 0x4bbc6a0 <line:1:24, line:4:1>
  |-CallExpr 0x4bbc610 <line:2:2, col:16> 'int'
  | |-ImplicitCastExpr 0x4bbc5f8 <col:2> 'int (*)(const char *, const char *) throw()' <FunctionT
oPointerDecay>
  | | `-DeclRefExpr 0x4bbc5d0 <col:2> 'int (const char *, const char *) throw()' lvalue Function
0x4baf660 'strcmp' 'int (const char *, const char *) throw()'
  | |-ImplicitCastExpr 0x4bbc660 <col:9> 'const char *' <NoOp>
  | | `-ImplicitCastExpr 0x4bbc648 <col:9> 'char *' <LValueToRValue>
  | |   `-DeclRefExpr 0x4bbc578 <col:9> 'char *' lvalue Var 0x4bbbc60 'p' 'char *'
  | `-ImplicitCastExpr 0x4bbc678 <col:12> 'const char *' <ArrayToPointerDecay>
  |   `-StringLiteral 0x4bbc5a0 <col:12> 'const char [3]' lvalue "aa"
  `-NullStmt 0x4bbc690 <line:3:1>
```

AST变换

在变换之后：

```
[cling]$ strcmp(p, "aa");
FunctionDecl 0x4bbbdc0 <input_line_7:1:1, line:4:1> __cling_Un1Qu32 'void (void)'
`-CompoundStmt 0x4bbc428 <line:1:24, line:4:1>
  |-IfStmt 0x4bbc2a0 <line:2:9, <invalid sloc>>
  | |-<<<NULL>>>
  | |-UnaryOperator 0x4bbc280 <col:9> '_Bool' prefix '!'
  | | `-ImplicitCastExpr 0x4bbc268 <col:9> '_Bool' <PointerToBoolean>
  | |   `-ImplicitCastExpr 0x4bbc070 <col:9> 'const char *' <NoOp>
  | |     `-ImplicitCastExpr 0x4bbc058 <col:9> 'char *' <LValueToRValue>
  | |       `-DeclRefExpr 0x4bbbeb8 <col:9> 'char *' lvalue Var 0x4bbbc60 'p' 'char *'
  | |-CallExpr 0x4bbc200 <<invalid sloc>> 'void'
  | | |-ImplicitCastExpr 0x4bbc1e8 <<invalid sloc>> 'void (*)(void *, void *)' <FunctionToPointer
Decay>
  | | | `-DeclRefExpr 0x4bbc198 <<invalid sloc>> 'void (void *, void *)' lvalue Function 0x474ae8
0 'cling__runtime__internal__throwNullDerefException' 'void (void *, void *)'
  | | |-ImplicitCastExpr 0x4bbc238 <<invalid sloc>> 'void *' <BitCast>
  | | | `-CStyleCastExpr 0x4bbc118 <<invalid sloc>> 'class clang::Sema *' <IntegralToPointer>
  | | |   `-IntegerLiteral 0x4bbc0e8 <<invalid sloc>> 'unsigned long' 74308016
  | | `-ImplicitCastExpr 0x4bbc250 <<invalid sloc>> 'void *' <BitCast>
  | |   `-CStyleCastExpr 0x4bbc170 <<invalid sloc>> 'class clang::Expr *' <IntegralToPointer>
  | |     `-IntegerLiteral 0x4bbc140 <<invalid sloc>> 'unsigned long' 79413360
  | `-<<<NULL>>>
  |-CallExpr 0x4bbc020 <col:2, col:16> 'int'
  | |-ImplicitCastExpr 0x4bbc008 <col:2> 'int (*)(const char *, const char *) throw()' <FunctionT
oPointerDecay>
  | | `-DeclRefExpr 0x4bbbf88 <col:2> 'int (const char *, const char *) throw()' lvalue Function
0x4baf660 'strcmp' 'int (const char *, const char *) throw()'
```

一些例子

```
[cling]$ int *p = 0;
[cling]$ *p = 5;
In file included from -:1:
input_line_5:2:3: warning: null passed to a callee which requires a non-null argument
 *p = 5;
  ^
[cling]$ int *q = 0;
[cling]$ int x;
[cling]$ x = *q;
In file included from -:1:
input_line_6:2:7: warning: null passed to a callee which requires a non-null argument
 x = *q;
      ^

[cling]$ .rawInput 1
[cling]! extern "C" int cannotCallWithNull(int* p) __attribute__((nonnull(1)));
[cling]! .rawInput 0
[cling]$ extern "C" int cannotCallWithNull(int* p) {
[cling]$ ?     if (!p)
[cling]$ ?       printf("Must not be called with p=0.\n");
[cling]$ ?     return 1;
[cling]$ ?   }
[cling]$ int *q = 0;
[cling]$ cannotCallWithNull(q);
In file included from -:1:
input_line_8:2:21: warning: null passed to a callee which requires a non-null argument
 cannotCallWithNull(q);
                    ^
```

2013-09-20-015658_686x99_scrot.png

提高效率

为了提高效率，尽量避免没有必要的运行时检测。

```
int *p = new int (1);
cannotCallWithNull(p);   // Do not need check!

int f(int* p = 0);     //  #include "MyLib.h"
int f(int* p)__attribute__((nonnull(1)))  //  #include MyLibProtected.h
f();                              // Do not need check, throw directly!
```

参考资料

V. Vassilev, P. Canal, A. Naumann, P. Russo. Cling – Past, Present and Future (slides)

V. Vassilev, P. Canal, A. Naumann, P. Russo, Cling – The New C++ Interpreter for ROOT 6 (slides)

V. Vassilev, P. Canal, A. Naumann, P. Russo Cling–The New Interactive Interpreter for ROOT 6, Journal of Physics, 2012

Thanks & Question?