Clank: Java-port of C/C++ Frontend

Sharing the NetBeans Team’s Experience

Petr Kudriavtsev
Vladimir Voskresensky
Oracle

March 27, 2017
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Speakers

Petr Kudriavtsev

Vladimir Voskresensky
Agenda

- Why porting?
- Known approaches
- Converter
- Porting C++ and Clang challenges
- Clank Demo
Why not binding?

- Why Emscripten?
  - LLVM IR to JavaScript 'assembler'?
- Why Lucene => CLucene?
  - Java ported to C++?
- Why Hibernate => NHibernate?
  - Java ported to .NET?
- Why people do porting?
  - It's fun!
C++ and 2*C == Java

The 10 most popular computer languages on GitHub

C++ and C == Java

TIOBE Programming Community Index

Source: www.tiobe.com
What is our favorite C++ Technology?
What is our favorite C++ Technology?
No religious wars! Let's share Clang Technology

Add One More Thread Holding Developers Together
Clang Technology evaluation

- Native Clang library requirements without functional regressions:
  - Full access to the strength of technology
  - All Java-aware platforms
  - Safety
  - Debug
  - Performance of native clang
  - JNI/JNA Bridging overhead
  - Upgrade to new Clang release
Clang Technology evaluation
(JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...

- Safety
- Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!

- Debug
- We hadn't have Mixed-dev in NetBeans yet...

- Performance of native clang
- Clang preprocessing itself is 2 times slower, parsing is 10x slower

- JNI/JNA Bridging overhead
- Need to expose whole AST API

- Upgrade to new Clang release
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
Clang Technology evaluation (JNI/JNA prototyping)

• Full access to the strength of technology
  – Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...

• All Java-aware platforms
  – MacOS, Linux, Windows, and Solaris
  – X86 and SPARC
  – 32 and 64bits

• Safety
  – Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
Clang Technology evaluation
(JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...

- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits

- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!

- Debug
  - We hadn't have Mixed-dev in NetBeans yet...

- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API
- Upgrade to new Clang release
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API

✔ Upgrade to new Clang release

Conclusion: Clang doesn't bring any extra value?
Clang Technology evaluation (JNI/JNA prototyping)

- Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...
- All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits
- Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!
- Debug
  - We hadn't have Mixed-dev in NetBeans yet...
- Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower
- JNI/JNA Bridging overhead
  - Need to expose whole AST API
- Upgrade to new Clang release

Wait! Let's try Clang in Java!
Clang Technology evaluation (JNI/JNA prototyping)

✅ Full access to the strength of technology
  - Including AST, ASTRecursiveVisitors, ASTMatchers, CFG ...

✅ All Java-aware platforms
  - MacOS, Linux, Windows, and Solaris
  - X86 and SPARC
  - 32 and 64bits

✅ Safety
  - Forgot QualType.isNull() check in your Java call? Welcome to JVM Core Dump!

✅ Debug
  - We hadn't have Mixed-dev in NetBeans yet...

❌ Performance of native clang
  - Clang preprocessing itself is 2 times slower, parsing is 10x slower

✅ JNI/JNA Bridging overhead
  - Need to expose whole AST API

❌ Upgrade to new Clang release

Wait! Let's try Clang in Java!
Agenda

• Why porting?
• **Known approaches**
• Converter
• Porting C++ and Clang challenges
• Clank Demo
• Inspired by ...
LLVM IR Based
LLVM IR Based

- Inspired by Emscripten
LLVM IR Based

- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode
LLVM IR Based

- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode
- Assembler Level Output
  - Difficult to understand
  - Difficult to debug by client

C++

```c
/*
 ** Return the SQL associated with a prepared statement
 */

SQLITE_API const char *sqlite3_sql(sqlite3_stmt *pStmt)
{
    Vdbe *p = (Vdbe *)pStmt;
    return (p && p->isPrepareV2) ? p->zSql : 0;
}
```

JavaScript

```javascript
function _sqlite3_sql($pStmt) {
    $pStmt = $pStmt|0;
    var $0 = 0, $1 = 0, $2 = 0, $3 = 0, $4 = 0, $5 = 0;
    sp = STACKTOP;
    STACKTOP = STACKTOP + 16|0;
    $0 = sp + 4|0;
    $p = sp;
    HEAP32[($0|0)]; = $pStmt;
    $1 = HEAP32[($0|0)].$0;
    HEAP32[($p|0)]; = $1;
    $2 = HEAP32[($p|0)];$0;
    $3 = ($2|0)!($0|0);
}"
```
LLVM IR Based

- Inspired by Emscripten
- Transform LLVM IR to Java Bytecode
- Assembler Level Output
  - Difficult to understand
  - Difficult to debug by client
- Java AST* APIs are needed to be generated from C-like IR back to Java Classes/methods
Existing C++ to Java Converters
Existing C++ to Java Converters

Low Accuracy on C++11 Codebases
Existing C++ to Java Converters

Low Accuracy on C++11 Codebases
Clang Based

- Inspired by ast-print
Clang Based

• Inspired by ast-print
  – Clang: C++ Source to Clang-AST

AST:

C++:

```cpp
int main(int argc, char** argv) {
  // Print description
  cout << "Support metric quote program" << endl;
```

```
FunctionDecl 0x554c360 <file:/home/petrk/devarea/sputnik-j>
  ParmVarDecl 0x554c210 <col:10, col:14> col:14 argc
  ParmVarDecl 0x554c288 <col:20, col:27> col:27 argv
  CompoundStmt 0x554dbf0 <col:33, line:23:1>
    CXOperatorCallExpr 0x554d360 <line:13:5, col:47>
      ImplicitCastExpr 0x554d348 <col:44> '__ostream__'
        DeclRefExpr 0x554d2c0 <col:44> '__ostream__'
        CXOperatorCallExpr 0x554c880 <col:5, col:13> '
          ImplicitCastExpr 0x554c868 <col:10> 'basic_os'
            DeclRefExpr 0x554c7e0 <col:10> 'basic_os'
              implicit_cast<
                ExplicitCast 0x554c418 <col:5> 'ostream':'clas'
                  ImplicitCastExpr 0x554c7c8 <col:13> 'const char'
                    StringLiteral 0x554c440 <col:13> 'const char'
                      ImplicitCastExpr 0x554da8 <col:47> 'basic_ostream'
```
Clang Based

• Inspired by ast-print
  – Clang: C++ Source to Clang-AST
  – ast-print: Clang-AST to C++ source

AST:

C++:

```cpp
FunctionDecl 0x554c360 <file-name>
  ParmVarDecl 0x554c210 <col:10, col:14> col:14 argc
  ParmVarDecl 0x554c288 <col:20, col:27> col:27 argv
  CompoundStmt 0x554dbf0 <col:33, line:23:1>
    CXXOperatorCallExpr 0x554d360 <line:13:5, col:47>
      ImplicitCastExpr 0x554d348 <col:44> '__ostream__'
        DeclRefExpr 0x554d2c0 <col:44> '__ostream_type'
  CXXOperatorCallExpr 0x554c808 <col:5, col:13>
    ImplicitCastExpr 0x554c868 <col:10> 'basic_os
      DeclRefExpr 0x554c7e0 <col:10> 'basic_ostream'
      DeclRefExpr 0x554c418 <col:5> 'ostream':'class
      ImplicitCastExpr 0x554c7c8 <col:13> 'const char
      StringLiteral 0x554c440 <col:13> 'const char
      ImplicitCastExpr 0x554d2a8 <col:47> 'basic_ostr
      DeclRefExpr 0x554d278 <col:47> 'basic_ostream

Printing main:
int main(int argc, char **argv) {
  cout << "Support metric quote program" << endl;
```
Clang Based

- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed

C++:

```cpp
int main(int argc, char** argv) {
    // Print description
    cout << "Support metric quote program" << endl;
```

Printed C++:

```
Printing main:
int main(int argc, char **argv) {
    cout << "Support metric quote program" << endl;
```
Clang Based

- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed
- But looks very promising!
Clang Based

- Inspired by ast-print
  - Clang: C++ Source to Clang-AST
  - ast-print: Clang-AST to C++ source
- Comments are missed
- But looks very promising!

Convert whole Clang-AST to Java Source!
Agenda

- Why porting?
- Known approaches
- **Converter**
- Porting C++ and Clang challenges
- Clank Demo
Prototype Converter

- Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy
Prototype Converter

• Within 1 day
  – Always print method bodies in class context to make Java happy
  – Replace arrow “→” by “.” to make Java happy

• Let's try to port!
Prototype Converter

• Within 1 day
  – Always print method bodies in class context to make Java happy
  – Replace arrow “→” by “.” to make Java happy

• Let's try to port!
  – And I'm going on vacation
Prototype Converter

• Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy

• Let's try to port!
  - After 2 weeks...
Prototype Converter

• Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy

• Let's try to port!
  - After 2 weeks...
Prototype Converter

• Within 1 day
  - Always print method bodies in class context to make Java happy
  - Replace arrow “→” by “.” to make Java happy

• Let's try to port!
  - After 2 weeks...

Team conclusion: Don't bother us with your crazy dreams! It is still manual!
Need a Plan...
Need a Plan...

- Bottom up approach
  - for API
Need a Plan...

- Bottom up approach
  - for API
Need a Plan...

- Bottom up approach
  - for API

Diagram:
- Clang/Lex
- Llvm Option
- Clang/Basic
- Llvm ADT/Support
Need a Plan...

- Followed by Top down approach
  - for implementations
Need a Plan…

- Followed by Top down approach
  - for implementations

- Clang/Lex
  - Llvm Option
  - Clang/Basic
  - Llvm ADT/Support
Need a Plan...

- Followed by Top down approach
  - for implementations

```
Clang/Lex

Llvm Option

Clang/Basic

Llvm ADT/Support
```
Need a Plan...

- **Bottom up approach**
  - Generate APIs without bodies
- **Followed by Top down approach**
  - Generate bodies starting from clients
    - Let's try Lex module
    - To build infrastructure
    - To evaluate ported Preprocessor
  - Adjusting APIs when better learn Clang/LLVM
    - Easy, fast, because bodies are absent
    - Add Java's “LibC++” and ADT/Support on demand
- **Use existing Clang tests to check semantic**
- **Annotate Java code to get help from IDE**
- **Release within NetBeans C++ support**
Same Time at Different World...

- Use Clang technology to parse C++
- Walk Clang AST to print Java code
During Short Nights...

- Use Clang technology to parse C++
- Walk Clang AST to print Java code
- 2 weeks to prototype JConvert
  - Port sample C++ project to Java
  - Keep semantic
  - Keep code as close as possible
  - Keep comments
And Long Weekends...

- Use Clang technology to parse C++
- Walk Clang AST to print Java code
- 2 weeks to prototype JConvert
  - Port sample C++ project to Java
  - Keep semantic
  - Keep code as close as possible
  - Keep comments
- Demo
JConvert 0.0.1

- C++ Quote vs Java Quote snippets

```cpp
int type = 0;
switch (response) {
    case 'Q':
        return 2; // default user requested termination
    case 'E':
        type = Cpu::HIGH.getValue();
        break;
    case 'M':
    default:
        type = Cpu::MEDIUM.getValue();
        break;
}

int amount = readNumberOf("CPUs", 1, 10);
MyCpu/*J*/= new Cpu(type, 0);
```
Agenda

- Why porting?
- Known approaches
- Converter
- **Porting C++ and Clang challenges**
- Clank Demo
Clang  - Pronunciation: /kläNG/  
A loud, resonant metallic sound or series of sounds  
- Oxford Dictionary

Clank  - Pronunciation: /kläNGk/  
A loud, sharp sound or series of sounds, typically made by pieces of metal meeting or being struck together  
- Oxford Dictionary
Clank: As close to origin as possible

- Convert Clang components for fully functional Preprocessor
  - Keeps comments
  - Semantically equivalent
  - Passes Clang tests

- Pure Java
  - Modular
  - Java “LibC++”

- Adopted by NetBeans

- The same License as LLVM
  - “Wanted the code to be used!” quoting Chris Lattner
“All hope abandon, ye who enter here.”
— Dante Alighieri, The Divine Comedy
C++ in Java Challenges

- Names collisions
  - Non-virtual methods in base and derived classes
    - In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors
- Diagnostics are not printed
  - Temporary objects lifecycle
- Multiple inheritance
- Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime
- Split by TUs vs Monolithic Java classes
- this+1 and TrailingObjects
- Custom new operators
- JAVA code Performance
Clank: All is solvable

- Names collisions
  - Non-virtual methods in base and derived classes
    - In Java all methods are virtual
    - 'unsigned int' vs 'int' overloaded methods and constructors

- Diagnostics are not printed
  - Temporary objects lifecycle

- Multiple inheritance

- Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime

- Split by TUs vs Monolithic Java classes

- this+1 and TrailingObjects

- Custom new operators

- JAVA Clank Preprocessor Performance
Clank: All is solvable

Complete and fast Clank Preprocessor, 1.1 MLoc, integrated into NetBeans

✓ Names collisions
  - Non-virtual methods in base and derived classes
    • In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors

✓ Diagnostics are not printed
  - Temporary objects lifecycle

✓ Multiple inheritance

✓ Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime

✓ Split by TUs vs Monolithic Java classes
✓ this+1 and TrailingObjects
✓ Custom new operators
✓ JAVA Clank Preprocessor Performance
Clank Memory Profiling

[Image of Clank Memory Profiling tool interface showing allocations, allocation pressure, and stack trace.]
Clank: Performance analysis and optimizations in Java code

• Use Performance Analyzer to compare with Clang
  – PerfAn profiles Java or C++ using sampling with 2% overhead
  – Compare Instructions and CPU Cycles and do local perf optimizations
• Use Java Flight Recorder to profile memory footprint
• Teach Converter to produce more optimal code
• Use specializations based on parametrized spec files
  – Change template file, all specializations are regenerated
  – Add mapping to generate specializations, regenerate code
Clank: All is solvable

✔ Names collisions
  - Non-virtual methods in base and derived classes
    • In Java all methods are virtual
  - 'unsigned int' vs 'int' overloaded methods and constructors

✔ Diagnostics are not printed
  - Temporary objects lifecycle

✔ Multiple inheritance

✔ Compile time preprocessor-conditional code in FileSystem
  - Changed #ifdef/#else/#endif to runtime

✔ Split by TUs vs Monolithic Java classes

✔ this+1 and TrailingObjects

✔ Custom new operators

✔ JAVA Clank Preprocessor Performance
Clank: Upgrade to Clang 3.9

- Tooling
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress
Clank: Upgrade to Clang 3.9

- Update view

**Builtin.java** (changed/total directives: 1/3, changed/total children: 4/43)
- **Generate with body**, **Generate without body**, **Generate with body in output Context** (changed/total directives: 3/31, changed/total children: 3/31)
- **Generate with body**, **Generate without body**, **Mark as updated**
  - **isPure** - **ADDED** (Insert after)
  - **Generate with body**, **Generate without body**, **Mark as updated**
  - **builtinIsSupported** - **CHANGED**
  - **Generate with body**, **Generate without body**, **Mark as updated**
  - **isTSBuiltin** - **COMMENT**
  - **Generate with body**, **Generate without body**, **Mark as updated**
  - **ID** - **INCLUDE**
  - **Generate with body**, **Generate without body**, **Mark as updated**

---
/export/devarea/LLVM38/llvm/tools/clang/lib/Basic/Builtins.cpp
+++ /export/devarea/LLVM39/llvm/tools/clang/lib/Basic/Builtins.cpp
@@ -72,1 +72,3 @@
- return !BuiltinsUnsupported && !MathBuiltinsUnsupported &&
+ bool OclCUnsupported = LangOpts.OpenCLVersion != 200 &&
+     BuiltinInfo.Langs == OCLC20_LANG;
+ return !BuiltinsUnsupported && !MathBuiltinsUnsupported && !OclCUnsupported &&
Clank: Upgrade to Clang 3.9

- Tooling
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress

- 1 person – 4 weeks for 1.1MLoc

- Improve Upgrade Tools based on feedback
Clank: Upgrade to Clang 3.9

- **Tooling**
  - Analyze diffs
  - Analyze dependencies
  - Detect Changed Entities
  - Prepare TODO actions
  - Process Moved and Renamed actions first
  - Drive upgrade
  - Mark progress
  - Track progress

- **1 person – 4 weeks for 1.1MLoc**

- Improve Upgrade Tools based on feedback

Let's move toward complete C++ Frontend!
Agenda

- Why porting?
- Known approaches
- Converter
- Porting C++ and Clang challenges
- **Clank Demo**
Clank: Modular Structure

Java “LibC++”

• Memory and Pointers abstraction
• Unsigned types support
• Bit fields support
• STL Templates / Specializations
• I/O
• Function pointers
• @Converted annotation
Clank: Modular Structure

Ported LLVM/Clang libraries

- Clang
  - Driver
  - Basic
  - Lex
  - AST
  - Analysis
  - Parse
  - Sema
  - Edit
  - Rewrite
  - Frontend
  - StaticAnalyzer
  - FrontendTool
  - Tools/Driver
- LLVM/Options
- ADT/Support On demand
  - with Templates and Specializations
- Tests: ADT/Support, Lexer, Preprocessor, Parser

2 Million lines of code
Clank: Modular Structure

In-progress LLVM/Clang libraries

- Tooling
- ASTMatchers
- Serialization
- LLVM/Bitcode
- LLVM/IR
Clank: Porting progress

18 November 2016 ... 15 March 2017
2 persons: 4 months with long Russian NY break
Clank: Porting progress

18 November 2016 ... 15 March 2017
2 persons: 4 months with long Russian NY break

Improve Converter based on commits with “MANUAL” keyword in subject
80% MANUALs are AUTO now
Clank

Clang

Thank you!

Unite Developers Together