

A brief history of Cxx.jl

What is it?

- Interop package to combine C++/Julia in the same program
- One of two ways to combine Julia & C++
- One of the oldest Julia packages (> 8 years old originally targeted Julia 0.1)
- A C++ REPL environment
- A great proof of concept, that never got fully realized
- Disclaimer: Though I wrote most of the code, I haven't been maintaining it for > 3 years and haven't added new feature for > 6 years.

Features - Basic Usage

```
cxx """  
    #include <iostream>  
    """
```

cxx string macro provides
C++ global scope

```
function cxx_hello_world()  
    icxx ""  
        std::cout << "Hello World" << std::endl;  
    ""
```

```
end
```

icxx string macro provides
C++ local scope

Features - Type System Mapping

cxxt string macro provides C++ type parsing scope

Julia type parameter may be interpolated into C++ type

```
function last(x::cxxt"std::vector<$T>" ) where {T}
    icxx"&$x.back();"
end
```

Values may be interpolated back into C++ scope

N.B.: Standard Julia specialization rules apply, i.e. specialized code will be generated for each std::vector instance

Features - Julia Type Mapping

```
using Colors
function cxx_colors()
    c = colorant"red"
    vecT = cxxt"std::vector<$(typeof(c))>"
    a = @cxx $vecT();
    @cxx $a.push_back(c)
    @cxx $a.push_back(colorant"green")
    collect(a)
end
```

Cxx.jl provides
C++ STL
integration with
Julia stdlib

C++ std::string template
instantiated on
complicated Julia type

@cxx macro
provides "pseudo
C++" syntax in Julia
for convenience

Features - Back and Forth

```
function cxx_loops()  
    icxx""  
        for (int i = 0; i < 10; i++) {  
            $:(println("Hello from julia $i"))  
        }  
    ""  
end
```

Julia Code may be
reverse-interpolated into C++
(this statement will run 10x)

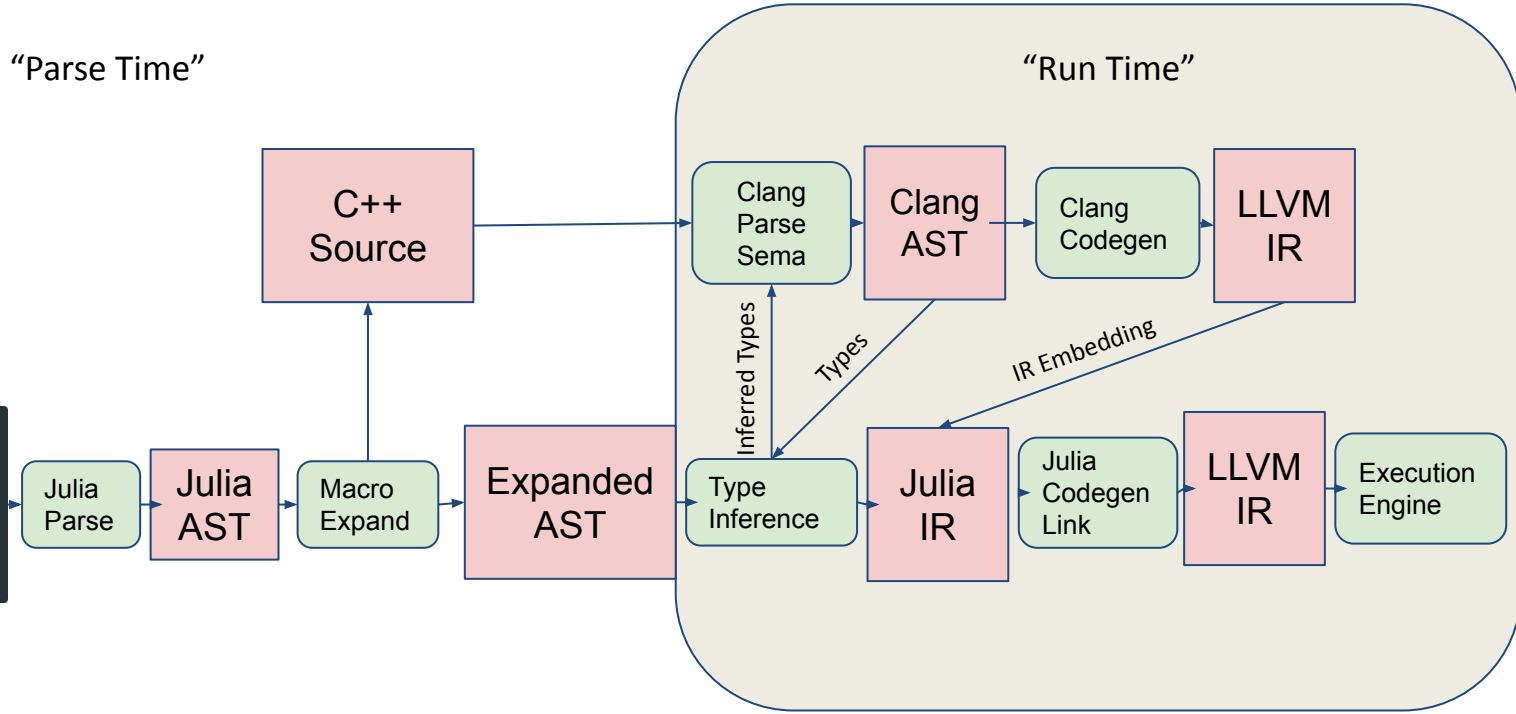
Nested interpolation works fine

Features - Interactive C++ REPL

```
Documentation: https://docs.julialang.org  
Type "?" for help, "]" for Pkg help.  
Version 1.1.0 (2019-01-21)  
Official https://julialang.org/ release  
  
[julia> using Cxx  
[C++ > // Press '<' to activate C++ mode  
[C++ > #include <iostream>  
true  
[C++ > std::cout << "Welcome to Cxx.jl" << std::endl;  
Welcome to Cxx.jl  
[C++ > std::string cxx = "C++";  
[julia> println("Combine Julia and ", String{icxx}cxx;")  
Combine Julia and C++  
julia> ]
```

How does it work? - Compiler

```
using Colors
function cxx_colors()
  c = colorant"red"
  vecT = cxxt"std::vector<${typeof(c)}>"
  a = @cxx $vecT{}
  @cxx $a.push_back(c)
  @cxx $a.push_back(colorant"green")
  collect(a)
end
```



How does it work? - Runtime

```
julia> using Cxx
```

```
julia> cxx"struct demo { int x; int y; };"  
true
```

```
julia> cxxt"demo"  
CppType{CxxQualType{CppType{:demo},(false, false, false)},N} where N
```

```
julia> a = icxx"demo{1, 2};"  
(struct demo) {  
  .x = (int &) 1  
  .y = (int &) 2  
}
```

```
julia> typeof(a)  
CppType{CxxQualType{CppType{:demo},(false, false, false)},8}
```

```
julia> typeof(icxx"&$a;")  
CppTypePtr{CxxQualType{CppType{:demo},(false, false, false)},(false, false, false)}
```

Embedding of C++ Types into Julia Type system (at Sema level)

CV Qualifiers

Value lives on Julia Heap

Default introspection code (N.B. Julia values carry type information unless erased by optimization)

Special representation for pointers/refs

How does it work? - Runtime

```
julia> cxx"""  
    struct nontrivial { ~nontrivial() { $:(println("I got deleted")); }; }  
    """  
true
```

```
julia> a = icxx"nontrivial{};"  
(struct nontrivial) {  
}
```

```
julia> a = nothing;
```

```
julia> # Some time later
```

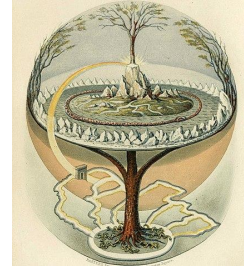
```
julia> GC.gc()  
I got deleted
```

Nesting works also
at global scope

Last reference
dropped here

Julia GC deletes
object => C++
destructor called

Challenges - Deployment



- Users likely got Julia as a binary package
 - Probably have no or an incompatible C++ stdlib/header environment
- Shipping Clang is a significant binary dependency
- Need story for shipping headers as well as binaries
- In recent versions, Julia community has taken control of binary dependencies - see github.com/JuliaPackaging/Yggdrasil
 - Likely easy to solve deployment challenges now, but additional work is required to plumb through all the required information

Challenges - Clang

- Clang not really designed for incremental compilation
 - Requires reaching into private internals to make it work
 - APIs generally unstable, but private APIs doubly so => significant maintenance burden with each version bump
 - Clang Occasionally gets into bad states
 - Particularly if the user makes C++ syntax errors => hard to unwind Clang state
 - Understand this is being worked on

Challenges - Julia (1)

- Clang dependency is big and slow to load
 - Ideally could load clang only if compilation is required, but currently no clear way to distinguish this case
 - Also an issue for Julia proper and being worked on, but no concrete timeline
- LLVM IR interop works, but somewhat half-hearted (e.g. not cacheable, incompatible with distributed computing)
- Julia has no scope-based lifetimes
 - Makes using C++ libraries that want RAII hard
 - May come in a future Julia version, but not yet designed
- Generic introspection utilities can violate C++ invariants (e.g. by not calling copy constructors)
 - No way to disallow this

Challenges - Julia (2)

- Julia Finalizers somewhat slow
 - Were design for small number of objects (e.g. files)
 - With Cxx, every C++ object with non-trivial destructor gets a finalizer
- Mapped C++ types are large => slow
 - Deeply nested C++ templates get mapped to deeply nested Julia type parameters
 - The Julia type system does not like this very much
- Julia LLVM pipeline not customizable
 - C++ and Julia need different passes => Cxx.jl can be slower than clang proper
 - Actively being worked on

Challenges - Fundamental (1)

- Lots of state that needs to be synchronized
 - LLVM IR
 - Julia IR
 - Clang AST
- For compilation speed Julia caches/reloads individual packages
 - How do we save/reload clang state?
 - PCMs may be an option, but when last investigated (~5-ish years ago were too complicated and required manual work for each C++ package)

Challenges - Fundamental (2)

- Different C++ packages require different compilation options
 - E.g. RTTI vs no-RTTI, exceptions vs no-exceptions
 - Cxx.jl supports multiple parallel Clang instances, but complicated
 - Need to be careful which compiler instance is used for every C++ statement
 - Type space is shared between instances, but options have ABI effects
- C++ compilation is slow (ok standalone, annoying at runtime)
- Julia users expect live-reload (hard to implement)

Cxx.jl consistently popular

Cxx.jl is the coolest thing ever! ✎

■ Offtopic



klaff

Sep 2019

I didn't find a `Place_to_Praise_Julia` category, so I'll post here.

I had a situation today in which an algorithm was supplied in the form of C snippets, and having a vague memory that one could compile C++ functions from a Julia REPL, I searched and found Cxx.jl. I tried the Cxx example, then a little copy/paste with the snippets and whammo!, instant answers at the REPL! So much fun!

Here's a big Thank You to the authors and contributors of Cxx.jl and all of Julia.

31 ❤️

created

Sep 2019

last reply

Sep 2019

2

replies

822

views

3

users

36

likes



cdsousa

13h



giordano:

`CxxWrap.jl` which is currently widely used to interface C++ libraries and isn't limited to Julia up to v1.3, so I don't understand the strong need for `Cxx.jl` (besides the fact it's cool)

`Cxx.jl` is not only cool, but it is also strongly helpful for doing quick and dirty experiments with C++ libs. It may not be robust to wrap libraries in the long term, but it enables the REPL driven development with C++ that we like so much in Julia!

I even used to rely a lot in `Cxx.jl` for quick checks when developing in C++ 😊

I miss being able to quickly trying using some C++-only algorithms within my Julia code 😞 (I'm currently trying using `cppyy` through `PyCall`)

A call to action

- Cxx.jl is currently only partially maintained
 - Works on Julia 1.3 / LLVM 6, but hasn't been keeping up
 - Several people interested in helping, but limited Clang knowledge
- Only 5-6 kloc, half of which on the C++ side
- Deeper integration with Cling/ROOT?
 - Regularly requested by HEP community
 - Quick prototype 6 years ago, but never went anywhere
- Reach out
 - keno@juliacomputing.com ; discourse.julialang.org ; julialang.org/slack/