How compiler frontend is different from what IDE needs?

Ilya Biryukov

JetBrains
ReSharper C++

November 3, 2016

Compilers and IDEs

Compilers and IDEs are similar, but have different design goals

Compilers and IDEs

Compilers and IDEs are similar, but have different design goals

- Compilers mostly run on correct code, IDEs often run on broken code

Compilers and IDEs

Compilers and IDEs are similar, but have different design goals

- Compilers mostly run on correct code, IDEs often run on broken code
- Compilers need to store fewer source locations than IDEs

Compilers and IDEs

Compilers and IDEs are similar, but have different design goals

- Compilers mostly run on correct code, IDEs often run on broken code
- Compilers need to store fewer source locations than IDEs
- Compiler works on a single TU, IDE is aware of the whole project

Deferred resolve

- Compilers resolve names as they are encountered

Deferred resolve

- Compilers resolve names as they are encountered
- This work is often redundant for an IDE

Deferred resolve

- Compilers resolve names as they are encountered
- This work is often redundant for an IDE
- E.g., do we really need to resolve everything inside `<iostream>`?

Example

```
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
}
```

Global includes

A typical C++ file

```
//// 1. Global includes
#include <vector>
#include <boost/something.hpp>
/* ... */

//// 2. Declarations
struct my_struct {
  /* ... */
};
my_struct create_struct();
/* ... */
```

Global includes

- Most includes are well-formed

Global includes

- Most includes are well-formed
  - Included in global scope

Global includes

- Most includes are well-formed
  - Included in global scope
  - Parse into a list of well-formed declarations

Global includes

- Most includes are well-formed
  - Included in global scope
  - Parse into a list of well-formed declarations
- Result of parsing the header may be reused between TUs
  - Some trickery involved to ensure preprocessor context matches

Reparse inside function body

How to process a change inside a function body?

Reparse inside function body

How to process a change inside a function body?

- Parse only the function body
- Replace old function body with the new one

Reparse inside function body

How to process a change inside a function body?

- Parse only the function body
- Replace old function body with the new one
- Very fast

Global reparse

What if a change is global?

Global reparse

A typical C++ file

```
//// 1. Global includes
#include <vector>
#include <boost/something.hpp>
/* ... */

//// 2. Declarations
struct my_struct {
  /* ... */
};
my_struct create_struct();
/* ... */
```

Global reparse

What if a change is global?

- Log all modifications after includes

Global reparse

What if a change is global?

- Log all modifications after includes
- When reparse is required
  - Rollback logged modifications
  - Parse only the rest of the file

Global reparse

What if a change is global?

- Log all modifications after includes
- When reparse is required
  - Rollback logged modifications
  - Parse only the rest of the file
- Much faster than processing the whole file

Conclusion

- ReSharper C++ has its own C++ frontend

Conclusion

- ReSharper C++ has its own C++ frontend
- With some optimizations on top of it
  - Deferred resolve
  - Optimized processing of global includes
  - Incremental reparse inside function bodies
  - Incremental reparse after global includes

Conclusion

- ReSharper C++ has its own C++ frontend
- With some optimizations on top of it
  - Deferred resolve
  - Optimized processing of global includes
  - Incremental reparse inside function bodies
  - Incremental reparse after global includes
- Still not fast enough
  - Running under managed runtime(.NET)
  - Slow preprocessing