3 objects / 3 threads

CMD thread - CobolProject / CobolFile

UI thread - CobolDocument / TextEditor / ErrorList / ProgramOutline

BG thread – CompilationUnit

CobolFile [create new] => CobolDocument

CobolFile [load existing] => CobolDocument

CobolFile [Save] <= CobolDocument

CobolFile [open] => TextEditor

CobolFile [close] => TextEditor

CobolDocument [close notification] => Cobol file : status closed

CobolDocument [global change notification] => Cobol file : status “modified” / “unchanged”

=> TextEditor

CobolDocument [text change] => CompilationUnit

CompilationUnit [token change] => TextEditor / ErrorList / ProgramOutline

AvalonEdit integration

* AvalonEdit => Compiler

IEnumerable<char> with newline chars

* Compiler => AvalonEdit

textLine.Indicator / textLine.Type

textLine.Text

Antlr integration

* Antlr => Compiler

1. ICharStream

Build a char Array interface on top of the Document interface

Copy source code of AntlrInputStream on top of this char array

1. IToken

Create a super class of the current Token class which implements IToken, use it internally, but never return it through a public API

1. ITokenSource

Create a Token enumerator on top of the CompilationUnit class (with TokensLines, from a starting point and until an ending point)

Implement ITokenSource on top of this enumerator

Build a CommonTokenStream from the token source

(Consider implementing BufferedTokenStream to avoid a duplicate storage)

ICharStream details

string SourceName { get; }

Gets the name of the underlying symbol source. This method returns a non-null,

non-empty string. If such a name is not known, this method returns Antlr4.Runtime.IntStreamConstants.UnknownSourceName

int Size { get; }

Returns the total number of symbols in the stream, including a single EOF symbol.

System.NotSupportedException: if the size of the stream is unknown.

int Index { get; }

Return the index into the stream of the input symbol referred to by LA(1). The behavior of this method is unspecified if no call to an Antlr4.Runtime.IIntStream

has occurred after this stream was constructed.

void Consume();

Consumes the current symbol in the stream. This method has the following effects:

Forward movement: The value of Antlr4.Runtime.IIntStream.Index before calling this method is less than the value of index() after calling this method.

Ordered lookahead: The value of LA(1) before calling this method becomes the value of LA(-1) after calling this method.

Note that calling this method does not guarantee that index() is incremented by exactly 1, as that would preclude the ability to implement filtering streams (e.g. Antlr4.Runtime.CommonTokenStream which distinguishes between "on-channel" and "off-channel" tokens).

System.InvalidOperationException: if an attempt is made to consume the the end of the stream (i.e. if LA(1)==Antlr4.Runtime.IntStreamConstants.Eof before calling consume ).

int La(int i);

Gets the value of the symbol at offset i from the current position.

When i==1 , this method returns the value of the current symbol in the stream

(which is the next symbol to be consumed).

When i==-1 , this method returns the value of the previously read symbol in the stream.

It is not valid to call this method with i==0 , but the specific behavior is unspecified because this method is frequently called from performance-critical code. This method is guaranteed to succeed if any of the following are true: i>0 i==-1 and Antlr4.Runtime.IIntStream.Index returns a value greater than the value of index() after the stream was constructed and LA(1) was called in that order.

Specifying the current index() relative to the index after the stream was created allows for filtering implementations that do not return every symbol from the underlying source.

Specifying the call to LA(1) allows for lazily initialized streams.

LA(i) refers to a symbol consumed within a marked region that has not yet been released.

If i represents a position at or beyond the end of the stream, this method returns Antlr4.Runtime.IntStreamConstants.Eof.

The return value is unspecified if i<0 and fewer than -i calls to Antlr4.Runtime.IIntStream.Consume() have occurred from the beginning of the stream before calling this method.

System.NotSupportedException: if the stream does not support retrieving the value of the specified symbol

int Mark();

A mark provides a guarantee that Antlr4.Runtime.IIntStream.Seek(System.Int32)

operations will be valid over a "marked range" extending from the index where

mark() was called to the current Antlr4.Runtime.IIntStream.Index . This allows

the use of streaming input sources by specifying the minimum buffering requirements

to support arbitrary lookahead during prediction.

The returned mark is an opaque handle (type int ) which is passed to Antlr4.Runtime.IIntStream.Release(System.Int32) when the guarantees provided by the marked range are no longer necessary.

When calls to mark() / release() are nested, the marks must be released in reverse order of which they were obtained.

Since marked regions are used during performance-critical sections of prediction, the specific behavior of invalid usage is unspecified (i.e. a mark is not released, or a mark is released twice, or marks are not released in reverse order from which they were created).

The behavior of this method is unspecified if no call to an Antlr4.Runtime.IIntStream has occurred after this stream was constructed.

This method does not change the current position in the input stream.

The following example shows the use of Antlr4.Runtime.IIntStream.Mark()as part of an operation to safely work within a marked region, then restore the stream position to its original value and release the mark.

IntStream stream = ...; int index = -1; int mark = stream.mark();

try { index = stream.index(); perform work here... }

finally { if (index != -1) { stream.seek(index); } stream.release(mark); }

Retourne : An opaque marker which should be passed to Antlr4.Runtime.IIntStream.Release(System.Int32) when the marked range is no longer required.

void Release(int marker);

This method releases a marked range created by a call to Antlr4.Runtime.IIntStream.Mark().

Calls to release() must appear in the reverse order of the corresponding calls to mark(). If a mark is released twice, or if marks are not released in reverse order of the corresponding calls to mark() , the behavior is unspecified.

marker: A marker returned by a call to mark() .

void Seek(int index);

Set the input cursor to the position indicated by index.

If the specified index lies past the end of the stream, the operation behaves as though index was the index of the EOF symbol.

After this method returns without throwing an exception, then at least one of the following will be true.

Antlr4.Runtime.IIntStream.Index will return the index of the first symbol appearing at or after the specified index. Specifically, implementations which filter their sources should automatically adjust index forward the minimum amount required for the operation to target a non-ignored symbol.

LA(1) returns Antlr4.Runtime.IntStreamConstants.Eof

This operation is guaranteed to not throw an exception if index lies within a marked region.

The behavior of this method is unspecified if no call to an Antlr4.Runtime.IIntStream has occurred after this stream was constructed.

index: The absolute index to seek to.

System.ArgumentException: if index is less than 0

System.NotSupportedException: if the stream does not support seeking to the specified index

string GetText(Interval interval);

This method returns the text for a range of characters within this input stream.

This method is guaranteed to not throw an exception if the specified interval lies entirely within a marked range.

interval: an interval within the stream

System.ArgumentNullException: if interval is null

System.ArgumentException: if interval.a < 0 , or if interval.b < interval.a - 1 , or if interval.b lies at or past the end of the stream

System.NotSupportedException: if the stream does not support getting the text of the specified interval

struct Interval

// An immutable inclusive interval a..b.

// The start of the interval.

public readonly int a;

// The end of the interval (inclusive).

public readonly int b;

IToken details

int Channel { get; }

Return the channel this token.

Each token can arrive at the parser on a different channel, but the parser only "tunes" to a single channel.

The parser ignores everything not on DEFAULT\_CHANNEL.

int Type { get; }

Get the token type of the token

int Column { get; }

The index of the first character of this token relative to the beginning of the line at which it occurs, 0..n-1

int Line { get; }

// The line number on which the 1st character of this token was matched, line=1..n

string Text { get; }

Get the text of the token

ICharStream InputStream { get; }

Gets the Antlr4.Runtime.ICharStream from which this token was derived.

int StartIndex { get; }

The starting character index of the token. This method is optional; return

-1 if not implemented.

int StopIndex { get; }

The last character index of the token. This method is optional; return -1 if not implemented.

ITokenSource TokenSource { get; }

Gets the Antlr4.Runtime.ITokenSource which created this token.

int TokenIndex { get; }

An index from 0..n-1 of the token object in the input stream. This must

be valid in order to print token streams and use TokenRewriteStream. Return

-1 to indicate that this token was conjured up since it doesn't have a valid index.

ITokenSource details

int Column { get; }

Get the index into the current line for the current position in the input stream. The first character on a line has position 0.

Retourne : The line number for the current position in the input stream, or -1 if the current token source does not track character positions.

int Line { get; }

Get the line number for the current position in the input stream. The first line in the input is line 1.

Retourne : The line number for the current position in the input stream, or 0 if the current token source does not track line numbers.

ICharStream InputStream { get; }

Get the Antlr4.Runtime.ICharStream from which this token source is currently providing tokens.

Retourne : The Antlr4.Runtime.ICharStream associated with the current position in the input, or null if no input stream is available for the token source.

string SourceName { get; }

Gets the name of the underlying input source. This method returns a non-null, non-empty string. If such a name is not known, this method returns Antlr4.Runtime.IntStreamConstants.UnknownSourceName

ITokenFactory TokenFactory { get; set; }

Set the Antlr4.Runtime.ITokenFactory this token source should use for creating Antlr4.Runtime.IToken objects from the input.

Retourne : The Antlr4.Runtime.ITokenFactory currently used by this token source.

IToken NextToken();

Return a Antlr4.Runtime.IToken object from your input stream (usually a Antlr4.Runtime.ICharStream). Do not fail/return upon lexing error; keep chewing on the characters until you get a good one; errors are not passed through to the parser.

Summary

CobolFile

Stream -> IEnumerable<char> with newline chars

IEnumerable<char> -> TextDocument

TextDocument / DocumentLine -> CompilationUnit / TokensLine

Note : code completion sample

goworks / goworks.editor / src / org / tvl / goworks / editor / go / completion / GoCompletionQuery.java

goworks / works.editor.antlr4 / src / org / antlr / works / editor / antlr4 / completion

ANTLRWorks 2 (AW2) performs a similar operation, which I'll describe here. If you reference the source code for AW2, keep in mind that it is only released under an LGPL license.

1.Create a special token which represents the location of interest for code completion.

•In some ways, this token behaves like the EOF. In particular, the ParserATNSimulator never consumes this token; a decision is always made at or before it is reached.

•In other ways, this token is very unique. In particular, if the token is located at an identifier or keyword, it is treated as though the token type was "fuzzy", and allowed to match any identifier or keyword for the language. For ANTLR 4 grammars, if the caret token is located at a location where the user has typed g, the parser will allow that token to match a rule name or the keyword grammar.

2.Create a specialized ATN interpreter that can return all possible parse trees which lead to the caret token, without looking past the caret for any decision, and without constraining the exact token type of the caret token.

3.For each possible parse tree, evaluate your code completion in the context of whatever the caret token matched in a parser rule.

4.The union of all the results found in step 3 is a superset of the complete set of valid code completion results, and can be presented in the IDE.

The following describes AW2's implementation of the above steps.

1.In AW2, this is the CaretToken, and it always has the token type CARET\_TOKEN\_TYPE.

2.In AW2, this specialized operation is represented by the ForestParser<TParser> interface, with most of the reusable implementation in AbstractForestParser<TParser> and specialized for parsing ANTLR 4 grammars for code completion in GrammarForestParser.

3.In AW2, this analysis is performed primarily by GrammarCompletionQuery.TaskImpl.runImpl(BaseDocument).

<http://stackoverflow.com/questions/19690802/getting-first-and-follow-metadata-from-an-antlr4-parser>

Files => CobolFile / ProjectFile

Compilation => CompilationUnit / CompilationProject

Text => TextDocument

Analysis => TokensDocument, ProcessedTokensDocument, SyntaxTree, CodeModel

CompilationProject / ProjectFile

IList<CompilationUnit> CompilationUnits (CompilationUnit = Program | Class)

CompilationDocument

CobolFile

TextDocument

TokensDocument

IList< CompilationDocument> CopyReferences  
 ProcessedTokensDocument

Ilist<CompilationError> Errors

SyntaxTree

CodeModel

Ilist<CompilationError> Errors  
 IList<CompilationUnit> CallReferences

IList< CompilationDocument> Copybooks

LanguageWorkspace / SolutionFile

IList<CompilationProject> CompilationProjects

IDictionary<CompilationDocument, ILanguageEditor> OpenedEditors

ILanguageEditor CurrentEditor

IErrorsList ErrorsList

Language services

* Colorizing / Outlining / Formatting
* Navigate To / Object Browser / Go To Definition
* Quick Info / Completion List / Signature Help / Find all references
* Rename / Extract Method

**Cycle de vie**

Etape 1 : input = CobolFile | (templateName, templateParams), result = IEnumerable<char>

Si chargement

TextFileReader(cobolFile)

Sinon

TextTemplateManager(string templateName, options)

Etape 2 : init TextDocument

Si éditeur

new TextEditor(IEnumerable<char>, options)

Sinon

new TextDocument()

Etape 3 : input = cobolFile? textDocument textFormat, result = CompilationDocument

CobolFile

TextDocument

TokensDocument  
 ProcessedTokensDocument

IList< CompilationDocument> CopyReferences

Ilist<CompilationError> Errors

CompilationDocument.LoadInMemory(… )

CompilationDocument.LoadInEditor(… name, textFormat, IEnumerable<char> => ITextDocument)

CompilationUnit : CompilationDocument

TextDocument :

initialized with a IEnumerable<char>

but should not start to send notifications before activation

CompilationProject

1. new CompilationProject(options) / new CompilationProject(fileName)
2. project.AddNewProgram(programName) / project.AddNewClass(className) / project.AddNewCopybook(copyName) / project.AddExistingFile(fileName)
3. project.Remove
4. project.Save(fileName)

CompilationUnit

CompilationDocument

1.