FastParse

Fast, Modern Parser Combinators

Li Haoyi, SF Scala 10 Oct 2015

http://tinyurl.com/fastparse

Agenda

15min: Parsing Text

10min: FastParse

15min: Performance, Debugging, Internals

10min: Live coding demo

10min: Q&A

Total: 60min

Who Am I

Li Haoyi

Dropbox Dev-Tools, Web-Infra

Worked on Scala.js, Ammonite-REPL in free time

Parsing Text

Parsing Text is Hard!

String.split/String.replace	Extremely convenient! Totally inflexible
Regexes	Crazy terse Syntax, Non-recursive
Hand-rolled Recursive-descent	Fast, Tedious & repetitive, Error-prone
lex/yacc, ANTLR	Fast! Complex, confusing code generation

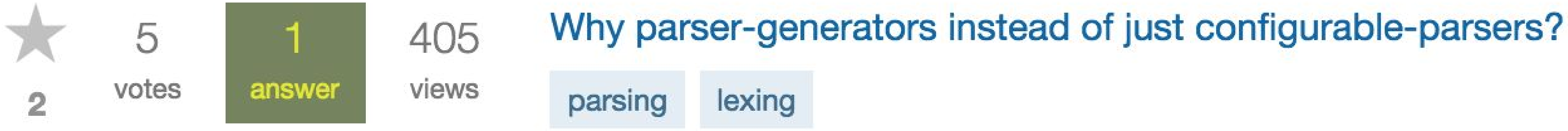
scala/tools/nsc/ast/parser/Parsers.scala

```scala
def enumerators(): List[Tree] = {
  val enums = new ListBuffer[Tree]
  enums ++= enumerator(isFirst = true)
  while (isStatSep) {
    in.nextToken()
    enums ++= enumerator(isFirst = false)
  }
  enums.toList
}
def enumerator(isFirst: Boolean, allowNestedIf: Boolean = true): List[Tree] =
  if (in.token == IF && !isFirst) makeFilter(in.offset, guard()) :: Nil
  else generator(!isFirst, allowNestedIf)
```

https://github.com/ruby/ruby/blob/trunk/parse.y

```
| mlhs '=' command_call
   {
   /*%%%*/
     value_expr($3);
     $1->nd_value = $3;
     $$ = $1;
   /*%
     $$ = dispatch2(massign, $1, $3);
   %*/
   }
| var_lhs tOP_ASGN command_call
   {
     value_expr($3);
     $$ = new_op_assign($1, $2, $3);
   }
```

```
| primary_value '[' opt_call_args rbracket tOP_ASGN
command_call
   {
   /*%%%*/
     NODE *args;
     value_expr($6);
     if (!$3) $3 = NEW_ZARRAY();
     args = arg_concat($3, $6);
     if ($5 == tOROP) {
         $5 = 0;
     }
     else if ($5 == tANDOP) {
         $5 = 1;
     }
     $$ = NEW_OP_ASGN1($1, $5, args);
     fixpos($$, $1);
```


★ 2

5 votes

1 answer

405 views

Why parser-generators instead of just configurable-parsers?

parsing lexing

Parser Combinators!

```
import scala.util.parsing.combinator._
object P extends RegexParsers{
  val plus = "+"
  val num = rep("[0-9]".r)
  val expr = num ~ plus ~ num
}
X.parseAll(X.expr, "123+123")
// [1.8] parsed: ((List(1, 2, 3)~+)~List(1, 2, 3))
X.parseAll(X.expr, "123123")
// [1.7] failure: `+' expected but end of source found
```

Parser Combinators!

```
import scala.util.parsing.combinator._
object P extends RegexParsers{
  val plus: Parser[String] = "+"
  val num: Parser[List[String]] = rep("[0-9]".r)
  val expr:Parser[List[String] ~ String ~ List[String]] = num ~ plus ~ num
}
X.parseAll(X.expr, "123+123")
// [1.8] parsed: ((List(1, 2, 3)~+)~List(1, 2, 3))
X.parseAll(X.expr, "123123")
// [1.7] failure: `+' expected but end of source found
```

Extracting Results

```
import scala.util.parsing.combinator._
object P extends RegexParsers{
  val plus = "+"
  val num = rep("[0-9]".r) map {_.mkString.toInt}
  val expr = num ~ plus ~ num map {case l ~ _ ~ r => l + r }
}

X.parseAll(X.expr, "123123+123123")
// [1.14] parsed: 246246
```

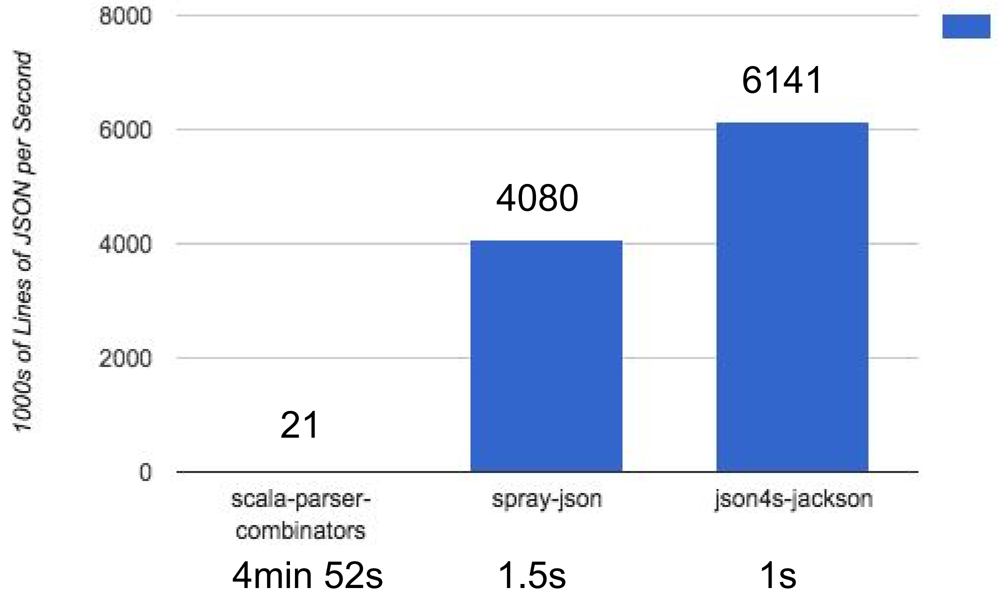
Extracting Results

```
import scala.util.parsing.combinator._
object P extends RegexParsers{
  val plus: Parser[String] = "+"
  val num: Parser[Int] = rep("[0-9]".r) map {_.mkString.toInt}
  val expr: Parser[Int] = num ~ plus ~ num map { case l ~ _ ~ r => l + r }
}

X.parseAll(X.expr, "123123+123123")
// [1.14] parsed: 246246
```

Recursion

```
import scala.util.parsing.combinator._
object P extends RegexParsers{
  val plus = "+"
  val num = rep1("[0-9]".r) map {_.mkString.toInt}
  val side = "(" ~> expr <~ ")" | num
  val expr: Parser[Int] = (side ~ plus ~ side) map {case l~_~r => l + r}
}
P.parseAll(P.expr, "1+(3+4)")
// [1.8] parsed: 8
P.parseAll(P.expr, "((1+2)+(3+4))+5")
// [1.16] parsed: 15
```

Performance



Parboiled2

https://github.com/sirthias/parboiled2

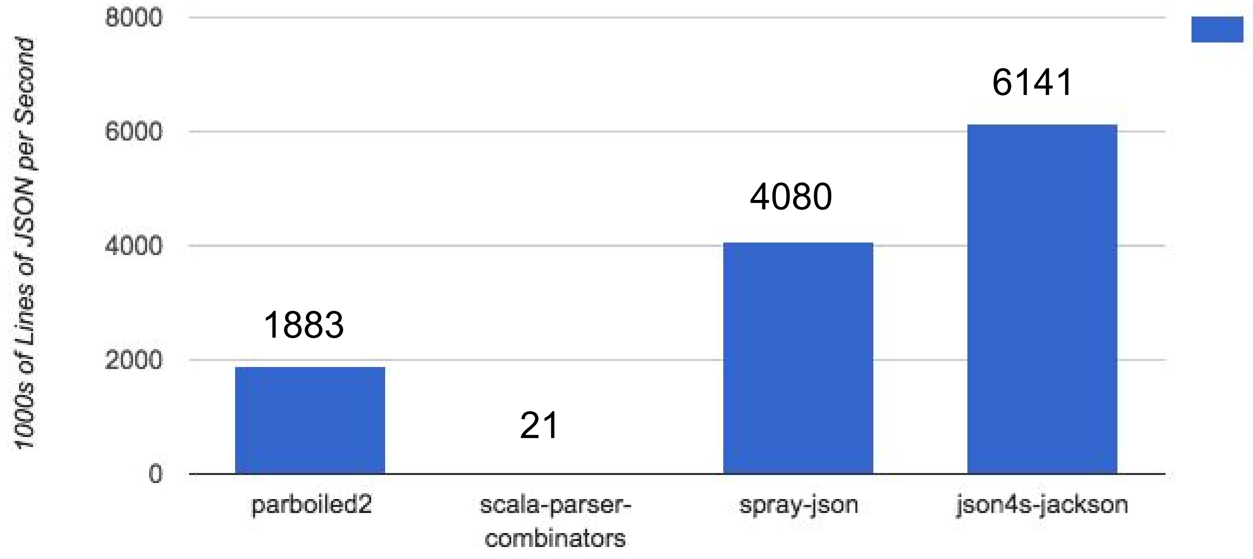
Fast!

Used in Akka, other places

Has some problems…

https://groups.google.com/forum/#!msg/scala-internals/4N-uK5YOtKI/9vAdsH1VhqAJ

Performance



Parboiled2 Error 1

```
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:16: type mismatch;
[error]  found   : shapeless.::[Int,shapeless.::[scalatex.stages.Ast.Block,shapeless.HNil]]
[error]  required: scalatex.stages.Ast.Block
[error]     new Parser(input, offset).Body.run().get
[error]                                           ^
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:60: overloaded method value apply with alternatives:
[error]   [I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[I,shapeless.::[J,shapeless.::[K,shapeless.::[L,shapeless.::[M,shapeless.::[N,shapeless.::[O,shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[J,shapeless.::[K,shapeless.::[L,shapeless.::[M,shapeless.::[N,shapeless.::[O,shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless
```

Parboiled2 Error 2

```
.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(J, K, L, M, N, O, P, Q, R, S,
T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.
In,j.Out] <and>
[error]   [K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text,
scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[K,
shapeless.::[L,shapeless.::[M,shapeless.::[N,shapeless.::[O,shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,
shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]]]]],shapeless.HNil,RR],
implicit c: org.parboiled2.support.FCapture[(K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.
Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.
stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[L,shapeless.::[M,
shapeless.::[N,shapeless.::[O,shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,
shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.
support.FCapture[(L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.
Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.
stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[M,shapeless.::[N,
shapeless.::[O,shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,
shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]]],
```

Parboiled2 Error 3

```
shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text,
scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [N, O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (N, O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.
Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[N,shapeless.::[O,
shapeless.::[P,shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,
shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(N, O, P, Q, R, S, T,
U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,
j.Out] <and>
[error]   [O, P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain,
Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[O,shapeless.::[P,shapeless.::[Q,
shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,
shapeless.HNil]]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(O, P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.
Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [P, Q, R, S, T, U, V, W, X, Y, Z, RR](f: (P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int,
scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[P,shapeless.::[Q,shapeless.::[R,
shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.
HNil]]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(P, Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text,
scalatex.stages.Ast.Chain, Int,
```

Parboiled2 Error 4

```
scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [Q, R, S, T, U, V, W, X, Y, Z, RR](f: (Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[Q,shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(Q, R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [R, S, T, U, V, W, X, Y, Z, RR](f: (R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[R,shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(R, S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [S, T, U, V, W, X, Y, Z, RR](f: (S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[S,shapeless.::[T,shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(S, T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [T, U, V, W, X, Y, Z, RR](f: (T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int,
```

Parboiled2 Error 5

```
scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[T,shapeless.::[U,shapeless.::[V,
shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.
FCapture[(T, U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.
parboiled2.Rule[j.In,j.Out] <and>
[error]   [U, V, W, X, Y, Z, RR](f: (U, V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block)
=> RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[U,shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,
shapeless.::[Z,shapeless.HNil]]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(U, V, W, X, Y, Z, scalatex.stages.Ast.
Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [V, W, X, Y, Z, RR](f: (V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) =>
RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[V,shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,
shapeless.HNil]]]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(V, W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.
stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [W, X, Y, Z, RR](f: (W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)
(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[W,shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]]],
shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(W, X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int,
scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
```

Parboiled2 Error 6

```
[error]   [X, Y, Z, RR](f: (X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[X,shapeless.::[Y,shapeless.::[Z,shapeless.HNil]]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(X, Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [Y, Z, RR](f: (Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[Y,shapeless.::[Z,shapeless.HNil]],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(Y, Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [Z, RR](f: (Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.::[Z,shapeless.HNil],shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(Z, scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [RR](f: (scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.HNil,shapeless.HNil,RR], implicit c: org.parboiled2.support.FCapture[(scalatex.stages.Ast.Block.Text, scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [RR](f: (scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.HNil,shapeless.::[scalatex.stages.Ast.Block.Text,shapeless.HNil],RR], implicit c: org.parboiled2.support.FCapture[(scalatex.stages.Ast.Chain, Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
```

Parboiled2 Error 7

```
[error]   [RR](f: (Int, scalatex.stages.Ast.Block) => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.HNil,shapeless.::
[scalatex.stages.Ast.Block.Text,shapeless.::[scalatex.stages.Ast.Chain,shapeless.HNil]],RR], implicit c: org.parboiled2.support.FCapture
[(Int, scalatex.stages.Ast.Block) => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [RR](f: scalatex.stages.Ast.Block => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.HNil,shapeless.::[scalatex.
stages.Ast.Block.Text,shapeless.::[scalatex.stages.Ast.Chain,shapeless.::[Int,shapeless.HNil]]],RR], implicit c: org.parboiled2.support.
FCapture[scalatex.stages.Ast.Block => RR])org.parboiled2.Rule[j.In,j.Out] <and>
[error]   [RR](f: () => RR)(implicit j: org.parboiled2.support.ActionOps.SJoin[shapeless.HNil,shapeless.::[scalatex.stages.Ast.Block.Text,
shapeless.::[scalatex.stages.Ast.Chain,shapeless.::[Int,shapeless.::[scalatex.stages.Ast.Block,shapeless.HNil]]]],RR], implicit c: org.
parboiled2.support.FCapture[() => RR])org.parboiled2.Rule[j.In,j.Out]
[error]  cannot be applied to ((scalatex.stages.Ast.Chain, scalatex.stages.Ast.Block) => scalatex.stages.Ast.Chain)
[error]     IndentBlock ~> {
[error]                 ^
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:71: The
`optional`, `zeroOrMore`, `oneOrMore` and `times` modifiers can only be used on rules of type `Rule0`, `Rule1[T]` and `Rule[I, O <: I]`!
[error]     push(offsetCursor) ~ IfHead ~ BraceBlock ~ optional("else" ~ (BraceBlock | IndentBlock))
[error]                                              ^
```

Parboiled2 Error 8

```
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:74: The `optional`, `zeroOrMore`, `oneOrMore` and `times` modifiers can only be used on rules of type `Rule0`, `Rule1[T]` and `Rule[I, O <: I]`!
[error]     Indent ~ push(offsetCursor) ~ IfHead ~ IndentBlock ~ optional(Indent ~ "@else" ~ (BraceBlock | IndentBlock))
[error]                                                        ^
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:91: type mismatch;
[error]  found   : Int
[error]  required: String
[error]     ((a, b, c) => Ast.Block.For(b, c, a))
[error]                               ^
[error] /Users/haoyi/Dropbox (Personal)/Workspace/scala-js-book/scalatexApi/src/main/scala/scalatex/stages/Parser.scala:112: type mismatch;
[error]  found   : org.parboiled2.Rule[shapeless.HNil,shapeless.::[Int,shapeless.::[scalatex.stages.Ast.Block,shapeless.HNil]]]
[error]  required: org.parboiled2.Rule[shapeless.HNil,shapeless.::[scalatex.stages.Ast.Block,shapeless.HNil]]
[error]   def BraceBlock: Rule1[Ast.Block] = rule{ '{' ~ BodyNoBrace ~ '}' }
[error]                                          ^
[error] 6 errors found
[error] (scalatexApi/compile:compile) Compilation failed
[error] Total time: 9 s, completed Nov 10, 2014 7:57:23 AM
```

Parboiled2 Original Error

```diff
  def BodyEx(exclusions: String = "") = rule{
-     push(offsetCursor) ~ oneOrMore(BodyItem(exclusions)) ~> {(i, x) =>
-       Ast.Block(x.flatten, i)
+     push(offsetCursor) ~ oneOrMore(BodyItem(exclusions)) ~> {(x) =>
+       Ast.Block(x.flatten)
    }
  }
```

Parsing Text is Hard!

String.split	Extremely convenient! Totally inflexible
Regexes	Crazy terse Syntax, Non-recursive
Hand-rolled Recursive-descent	Ridiculously tedious & repetitive, Error-prone
lex/yacc, ANTLR,	Fast! Complex, confusing code generation
scala-parser-combinators	Convenient! Flexible! Super slow
Parboiled2	Fast! Flexible! Crazy errors, awkward API

Simplified Overview

```scala
trait Parser[+T]{
  def parse(input: String, index: Int = 0): Result[T]
}
sealed trait Result[+T]{
  def index: Int
}
object Result{
  case class Success[+T](value: T, index: Int) extends Result[T]
  case class Failure(lastParser: Parser[_], index: Int)
  extends Result[Nothing]
}
```

Usage

```
object Foo{
  import fastparse.all._
  val plus = P( "+" )
  val num = P( CharIn('0' to '9').rep(1) ).!.map(_.toInt)
  val side = P( "(" ~ expr ~ ")" | num )
  val expr: P[Int] = P( side ~ plus ~ side ).map{case (l, r) => l + r}
}
Foo.expr.parse("123+123") // Success(246,7)
Foo.expr.parse("(1+2)+(3+4)") // Success(10,11)
Foo.expr.parse("(1+2") // Failure(("(" ~ expr ~ ")" | num):0 ..."(1+2")
```

Usage

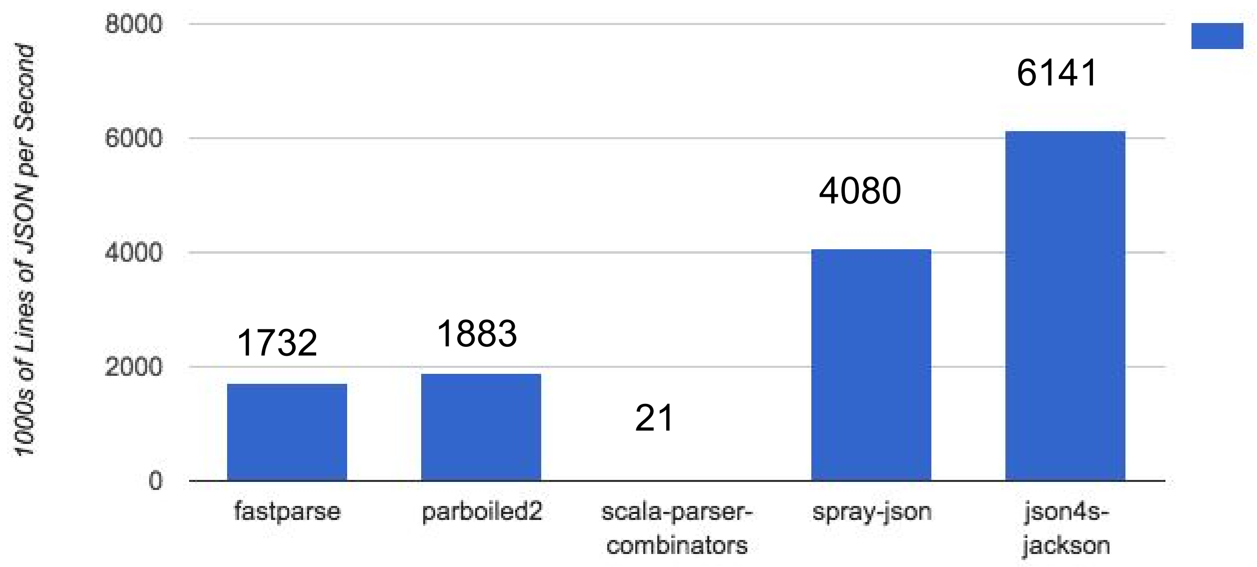
```
object Foo{
  import fastparse.all._
  val plus: P[Unit] = P( "+" )
  val num: P[Int] = P( CharIn('0' to '9').rep(1) ).!.map(_.toInt)
  val side: P[Int] = P( "(" ~ expr ~ ")" | num )
  val expr: P[Int] = P( side ~ plus ~ side ).map{case (l, r) => l + r}
}
Foo.expr.parse("123+123") // Success(246,7)
Foo.expr.parse("(1+2)+(3+4)") // Success(10,11)
Foo.expr.parse("(1+2") // Failure(("(" ~ expr ~ ")" | num):0 ..."(1+2")
```

Components

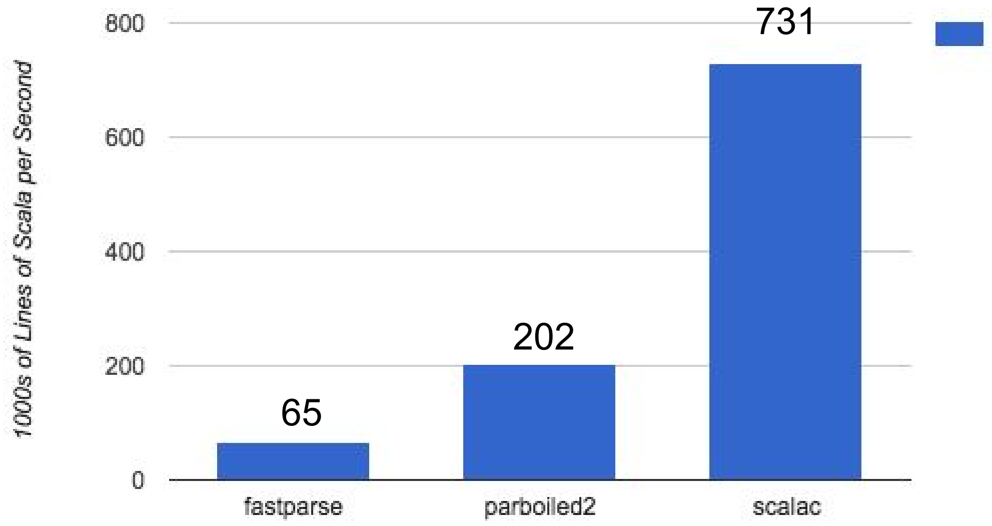
```
"hello" : P[Unit]

a ~ b   : P[(A, B)]

a | b   : P[T >: A >: B]

a ~! b  : P[(A, B)] // Cut

a.rep() : P[Seq[A]]

a.?     : P[Option[A]]

a.!     : P[String] // Capture

!(a), &(a) // Pos/Neg Lookahead
```

```
a.map(f: A => B): P[B]

a.flatMap(f: A => P[B]): P[B]

a.filter(f: A => Boolean): P[A]

a.log(s: String): P[A]

CharPred(f: Char => Boolean)

CharIn(s: Seq[Char]*)

CharsWhile(f: Char => Boolean, min: Int = 1)

StringIn(strings: String*)
```

Performance



Performance



Scala-Parser-Combinator Internals

```
def ~! [U](p: => Parser[U]) = OnceParser{
 (
   for(a <- this; b <- commit(p))
   yield new ~(a,b)
 ).named("~!")
}
```

Lambda w/ 2 captures: p & this

By-name lambda captures p

Allocation with at
least 1 fields

Lambda w/ 3 captures: p & a & this

Allocation with at
least 2 fields

FastParse Internals

```
def parseRec(cfg: ParseCtx, index: Int) = p1.parseRec(cfg, index) match{
  case f: Mutable.Failure => failMore(f, index, cfg.logDepth, traceParsers = if(cfg.traceIndex ==
-1) Nil else List(p1), cut = f.cut)
  case Mutable.Success(value0, index0, traceParsers0, cut0)  =>
    p2.parseRec(cfg, index0) match{
      case f: Mutable.Failure => failMore(
        f, index, cfg.logDepth,
        traceParsers = traceParsers0 ::: f.traceParsers,
        cut = cut | f.cut | cut0
      )
      case Mutable.Success(value1, index1, traceParsers1, cut1)  =>
       success(cfg.success, ev(value0, value1), index1, traceParsers1 ::: traceParsers0, cut1 |
cut0 | cut)
    }
}
```

All in one method

Zero allocations

Basic Error Handling

```
object Foo{
  import fastparse.all._
  val plus = P( "+" )
  val num = P( CharIn('0' to '9').rep(1) ).!.map(_.toInt)
  val side = P( "(" ~ expr ~ ")" | num )
  val expr: P[Int] = P( side ~ plus ~ side ).map{case (l, r) => l + r}
}

Foo.expr.parse("(1+(2+3x))+4")
// Failure(("(" ~ expr ~ ")" | num):0 ..."(1+(2+3x))")
```

Cuts

```scala
object Foo{
  import fastparse.all._
  val plus = P( "+" )
  val num = P( CharIn('0' to '9').rep(1) ).!.map(_.toInt)
  val side = P( "(" ~! expr ~ ")" | num )
  val expr: P[Int] = P( side ~ plus ~ side ).map{case (l, r) => l + r}
}

Foo.expr.parse("(1+(2+3x))+4")
// Failure(")":7 ..."x))+4")
```

Advanced Error Handling

~~`case class Failure(lastParser: Parser[_], index: Int)`~~

~~`extends Result[Nothing]`~~

```scala
case class Failure(input: String, index: Int,
                   lastParser: Parser[_], traceData: (Int, Parser[_]))
                   extends Result[Nothing]{
 lazy val traced: TracedFailure
 def msg: String
}
```

Parses a second time
to collect more data!

Advanced Error Handling

```
val fail = Foo.expr.parse("(1+(2+3x))+4").asInstanceOf[fastparse.core.Result.Failure]
> fail.traced.trace // The named parsers in the stack when it failed
expr:0 / side:0 / expr:1 / side:3 / (")" | CharIn("0123456789")):7 ..."x))+4"
> fail.traced.stack // Same as .trace, but as a List[Frame] rather than String
List(
  Frame(0,expr), // (1+(2+3x))+4
  Frame(0,side), // (1+(2+3x))+4
  Frame(1,expr), //  1+(2+3x))+4
  Frame(3,side)  //    (2+3x))+4
)
> (fail.index, fail.lastParser) // Last index and last parser at which it failed
(7, ")")         //         x))+4
```

Advanced Error Handling

```
> fail.traced.traceParsers // Every parser that could have succeeded at Failure#index
List(")", CharIn("0123456789"))

> fail.traced.fullStack // Every single parser in the stack when it failed
List(
  Frame(0,expr),      Frame(0,expr), Frame(0,side ~ plus ~ side),
  Frame(0,side),      Frame(0,"(" ~! expr ~! ")" | num), Frame(1,"(" ~! expr ~! " )"),
  Frame(1,expr),      Frame(1,expr), Frame(3,side ~ plus ~ side),
  Frame(3,side),      Frame(3,"(" ~! expr ~! ")" | num), Frame(7,"(" ~! expr ~! ")")
)
> (fail.index, fail.lastParser) // Last index and last parser at which it failed
(7, ")")          //         x))+4
```

Use cases

Debug your parser when it is wrong (e.g. you’re still working on it)

Providing errors to your users so they can debug why their input is wrong

Customizing errors, e.g. “parser X is in stack, user probably made mistake Y”

- `fail.traced.stack.contains(_.parser == Foo.side)`

ScalaParse Syntax Errors

```
trait Basic {
  b match {
    case C case _ => false
  }
}
// Scalac
'=>' expected but 'case' found.
// FastParse
expected "|" | `=>` | `⇒`
found "case"
```

```
var = 2

// Scalac
illegal start of simple pattern

// FastParse
expected Binding ~ InfixPattern |
         InfixPattern |
         VarId
found = "= 2"
```

Debugging

```
object Foo{
  import fastparse.all._
  val plus = P( "+" )
  val num = P( CharIn('0' to '9').rep(1) ).!.map(_.toInt)
  val side = P( "(" ~! expr ~ ")" | num ).log()
  val expr:P[Int] = P( side ~ plus ~ side ).map{case (l, r) => l+r}.log()
}
```

```
Foo.expr.parse("(1+(2+3x))+4")
```

```
(1+(2+3x))+4    +expr:0
(1+(2+3x))+4     +side:0
 1+(2+3x))+4       +expr:1
 1+(2+3x))+4         +side:1
 1+(2+3x))+4         -side:1:Success(2)
   (2+3x))+4         +side:3
    2+3x))+4           +expr:4
    2+3x))+4             +side:4
    2+3x))+4             -side:4:Success(5)
      3x))+4             +side:6
      3x))+4             -side:6:Success(7)
    2+3x))+4           -expr:4:Success(7)
   (2+3x))+4         -side:3:Failure(side:3 / ")":3 ..."(2+3x))+4", cut)
 1+(2+3x))+4       -expr:1:Failure(expr:1 / side:3 / ")":1 ..."1+(2+3x))+", cut)
(1+(2+3x))+4     -side:0:Failure(side:0 / expr:1 / side:3 / ")":0 ..."(1+(2+3x))", cut)
(1+(2+3x))+4    -expr:0:Failure(expr:0 / side:0 / expr:1 / side:3 / ")":0 ..."(1+(2+3x))"
```

Why all the talk of debugging?

As a *developer*, most of your time spent interacting with your parser is when your parser is incorrect and throwing errors at you.

As an end *user*, most of your time spent interacting with the parser is when your input is incorrect and it is throwing errors at you.

Implementation Details

Straightforward recursive-descent PEG

- No fancy parsing algorithms, disambiguation, async/push-parsing, ...
- No fancy macro-optimizations or parser-transformations; WYWIWYG

Object Oriented Design

- Build your own components! Just implement `Parser[+T]`

Externally immutable, but...

- Built-in `Parser[+T]`s are optimized & fast: while-loops, bitsets, etc.
- Internally uses `Mutable.{Success[T], Failure}` to save allocations

Example Usages

Examples: Math, Whitespace-handling, indentation-blocks, JSON

- http://lihaoyi.github.io/fastparse/#ExampleParsers

PythonParse: parsing a full python AST from source, including indentation-blocks

- https://github.com/lihaoyi/fastparse/tree/master/pythonparse

ScalaParse: parses Scala without generating an AST, heavily used in Ammonite

- https://github.com/lihaoyi/fastparse/tree/master/scalaparse

Scalatex: Programmable documents; uses ScalaParse & adds indentation-blocks

- https://github.com/lihaoyi/Scalatex

Parsing Text is ~~Hard~~ Easy!

String.split	Extremely convenient! Totally inflexible
Regexes	Crazy terse Syntax, Non-recursive
Hand-rolled Recursive-descent	Ridiculously tedious & repetitive, Error-prone
lex/yacc, ANTLR,	Fast! Complex, confusing code generation
scala-parser-combinators	Convenient! Flexible! Super slow, funky API
Parboiled2	Fast! Flexible! Crazy errors, awkward API
FastParse	Convenient! Fast! Flexible! Nice Errors/API!

FastParse Demo!

JSON

Questions?

Code & Issues: https://github.com/lihaoyi/fastparse

Docs: https://lihaoyi.github.io/fastparse

Chat Room: https://gitter.im/lihaoyi/fastparse

Ask me about

- Hack-free indentation-parsing
- Higher-order parsers
- Monadic Parser Combinators