Parsers Error Recovery for Practical Use

Alexander Azarov

azarov@osinka.ru / Osinka.ru

February 15, 2012

Osinka

- Forum of “BB” kind
    - (2.5M+ pages/day, ˜8M+ posts total)
        - User generated content: BBCode markup
- Migrating to Scala
    - Backend

BBCode

Few BBCode tags

[b]bold[/b]	<b>bold</b>
[i]italic[/i]	<i>italic</i>
[url=href]text[/url]	<a href="href">text</a>
[img]href[/img]	<img src="href"/>

BBCode example

- example of BBCode

Example

```
[quote="Nick"]
original   [b]text[/b]
[/quote]
Here it  is  the  reply  with
[url=http://www.google.com]link[/url]
```

Why parser combinators, 1

- Regexp maintenance is a headache
- Bugs extremely hard to find
- No markup errors

Why parser combinators, 2

One post source, many views

- HTML render for Web
- textual view for emails
- text-only short summary
- text-only for full-text search indexer

Why parser combinators, 3

Post analysis algorithms

- links (e.g. spam automated analysis)
- images
- whatever structure analysis we'd want

Universal AST

One AST

- different printers
- various traversal algorithms

Sounds great. But.

This all looks like a perfect world.
But what’s the catch??

Sounds great. But.

Humans.

They do mistakes.

Sounds great. But.

Humans.

They do mistakes.

Example

```
[quote]
[url=http://www.google.com]
[img]http://www.image.com
[/url[/img]
[/b]
```

User-Generated Content: Problem

Erroneous markup

- People do mistakes,
- But no one wants to see empty post,
- We have to show something meaningful in any case

Black or White World

- Scala parser combinators assume valid input
  - Parser result: `Success | NoSuccess`
    - no error recovery out of the box

Error recovery: our approach

- Our Parser never breaks
- It generates “error nodes” instead

Approach: Error nodes

- Part of AST, `FailNode` contains the possible causes of the failure
- They are meaningful
  - for highlighting in editor
  - to mark posts having failures in markup (for moderators/other users to see this)

Approach: input & unpaired tags

- Assume all input except tags as text
    - E.g. `[tag]text[/tag]` is a *text* node
- Unpaired tags as the last choice: markup errors

Example

Example

Trivial BBCode markup

Example (Trivial "one tag" BBCode)

Simplest [font=bold]BBCode [font=red]example[/font][/font]

- has only one tag, `font`
- though it may have an argument

Corresponding AST

AST

```
trait Node

case class Text(text: String)  extends Node
case class Font(arg: Option[String],  subnodes: List[Node])  extends
     Node
```

Parser

BBCode parser

```
lazy val nodes = rep(font | text)
lazy val text =
  rep1(not(fontOpen|fontClose) ~> "(?s).".r) ^^ {
    chars => Text(chars.mkString)
  }
lazy val font: Parser[Node] = {
  fontOpen ~ nodes <~ fontClose ^^ {
    case fontOpen(_, arg) ~ subnodes => Font(Option(arg),
        subnodes)
  }
}
```

Valid markup

Scalatest

```
describe("parser") {
  it("keeps spaces") {
    parse("  ")  must equal(Right(Text("  ")  ::  Nil))
    parse("  \n  ") must equal(Right(Text("  \n  ") :: Nil))
  }
  it("parses text") {
    parse("plain text") must equal(Right(Text("plain text") ::
         Nil))
  }
  it("parses bbcode-like text") {
    parse("plain [tag] [fonttext") must equal(Right(Text("
         plain [tag] [fonttext") :: Nil))
  }
```

Invalid markup

Scalatest

```
describe("error markup") {
  it("results in error") {
    parse("t[/font]") must be('left)
    parse("[font]t") must be('left)
  }
}
```

Recovery: Extra AST node

FailNode

```
case class FailNode(reason: String, markup: String) extends Node
```

Recovery: helper methods

Explicitly return `FailNode`

```
protected def failed(reason: String) = FailNode(reason, "")
```

Enrich `FailNode` with markup

```
protected def recover(p: => Parser[Node]): Parser[Node] =
  Parser { in =>
    val r = p(in)
    lazy val markup = in.source.subSequence(in.offset, r.next.offset
          ).toString
    r match {
      case Success(node: FailNode, next) =>
        Success(node.copy(markup = markup), next)
      case other =>
        other
```

Recovery: Parser rules

- never break (provide “alone tag” parsers)
- return `FailNode` explicitly if needed

nodes

```
lazy val nodes = rep(node | missingOpen)
lazy val node = font | text |  missingClose
```

“Missing open tag” parser

Catching alone `[/font]`

```
def missingOpen = recover {
  fontClose ^^^ { failed("missing open") }
}
```

Argument check

font may have limits on argument

```
lazy val font: Parser[Node] = recover {
  fontOpen ~ rep(node) <~ fontClose ^^ {
    case fontOpen(_, arg) ~ subnodes =>
      if (arg == null || allowedFontArgs.contains(arg))  Font(
            Option(arg), subnodes)
      else failed("arg incorrect")
  }
}
```

Passes markup error tests

Scalatest

```
describe("recovery") {
  it("reports incorrect arg") {
    parse("[font=b]t[/font]") must equal(Right(
      FailNode("arg incorrect", "[font=b]t[/font]") :: Nil
    ))
  }
  it("recovers extra ending tag") {
    parse("t[/font]") must equal(Right(
      Text("t") ::  FailNode("missing open", "[/font]") :: Nil
    ))
  }
```

Passes longer tests

Scalatest

```
it("recovers extra starting tag in a longer sequence") {
  parse("[font][font]t[/font]") must equal(Right(
    FailNode("missing close", "[font]") :: Font(None, Text("t
         ") :: Nil) :: Nil
  ))
}
it("recovers extra ending tag in a longer sequence") {
  parse("[font]t[/font][/font]") must equal(Right(
    Font(None, Text("t") :: Nil)  ::  FailNode("missing open", "
         [/font]") :: Nil
  ))
```

Examples source code

Fork me on GitHub

- Source code, specs: https://github.com/alaz/slides-err-recovery

Performance

- The biggest problem is performance.

  Scala parser combinators are very slow compared to the original **phpBB** BBCode parsing via regexp.

Benchmarks

	Scala	PHP
Typical 8k	51ms	5.3ms
Big w/err 76k	1245ms	136ms

- Workaround: caching

Thank you

- Email: azarov@osinka.ru
- Twitter: http://twitter.com/aazarov
- Source code, specs: https://github.com/alaz/slides-err-recovery