Parsers Error Recovery for Practical Use

Alexander Azarov

azarov@osinka.ru / Osinka.ru

February 10, 2012

Osinka

- phpBB forum, > 8M posts
  - User generated content: BBCode markup
- Slowly migrating to Scala
  - Backend right now

Why parser combinators

One post source, many views

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

Why parser combinators

One post source, many views

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

Ability to retrieve information from posts

- 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 result: `Success | NoSuccess`

Parser error recovery

How it works

- Parser does not break
- It generates “error nodes” instead

Useful:

- for highlighting in editor
- to mark posts having failures in markup (for moderators/other users to see this)

“Catch-all” Parser

Approach

- Native Scala parsers library
- We use “catch-all” parser
    - A “catch-all” parser is always the last
    - It always returns a “success” `ParseResult` with special `FailNode`
    - `FailNode` contains the possible causes of the failure

AST

Example (Trivial "one tag" BBCode)

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

Corresponding AST

```
trait Node

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

Typical 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)
  }
}
```

Testing: passes successful parsing

Scalatest spec

```
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 font w/o arg") {
    parse("[font]text[/font]") must equal(Right(Font(None, Text
        ("text") :: Nil) :: Nil))
  }
```

Recovering parser

Special AST node

```
// extra node for AST
case class FailNode(reason: String, markup: String) extends Node
```

Recovering parser

Explicitly returning `FailNode`

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

Recovering parser

Explicitly returning `FailNode`

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

`recover` wrapper around the `Parser` enriches `FailNode` with markup

```
protected def recover(p: => Parser[Node]): Parser[Node] =
```

Recovering parser

Explicitly returning `FailNode`

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

`recover` wrapper around the `Parser` enriches `FailNode` with markup

```
protected def recover(p:  => Parser[Node]):  Parser[Node] =
```

Putting together, missing tags parser

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

Recovering parser

Checking content

```
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")
  }
}
```

Testing: passes markup errors

Scalatest spec

```
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
    ))
  }
```

Examples source code

- 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

- Caching to the rescue!

Thank you

- Email: azarov@osinka.ru
- Twitter: http://twitter.com/aazarov