Byte-Sized

low-level programming with Vapor

Brett R. Toomey

Vapor and iOS Developer @Nodes

@BrettRToomey

Agenda

- Address `String` performance
- Demystify bytes
- Build a simple parser that works on bytes

Strings

Pros

- Easy to debug
- Flexible with 3rd-party packages
- Lots of preexisting conveniences and snippets

Strings

Cons

- Performance
- Maintainability
- High level approach for a low level task

Bytes

Pros

- Super fast
- Rapid parser development
- Easy to maintain

Bytes

Cons

- Harder to debug
- Less builtin functionality

The Almighty Scanner

```
struct Scanner<Element> {
    var pointer: UnsafePointer<Element>
    let endAddress: UnsafePointer<Element>
    var elements: UnsafeBufferPointer<Element>
    // assuming you don't mutate no copy _should_ occur
    let elementsCopy: [Element]
}
```

```
 func peek(aheadBy n: Int = 0) -> Element? {
     guard pointer.advanced(by: n) < endAddress else { return nil }
     return pointer.advanced(by: n).pointee
 }

func pop() -> Element {
     assert(pointer != endAddress)
     defer { pointer = pointer.advanced(by: 1) }
     return pointer.pointee
 }

func pop(_ n: Int) {
     assert(pointer.advanced(by: n) <= endAddress)
     pointer = pointer.advanced(by: n)
 }
```

Example Usage

```
import Bits // vapor/bits

let bytes = "Hello, world!".makeBytes()
var scanner = Scanner(bytes)

scanner.peek() // 0x48 (H)
scanner.peek(aheadBy: 1) // 0x65 (e)

let byte = scanner.pop() // 0x48 (H)
```

The Goal

Make a parser for the following "JSON"

```
{
    "name": "Brett",
    "age": 90
}
```

How?

Recursive Descent Parsing

A top-down algorithm that breaks each grammatical structure into its own, mutually-recursive, function.

Algorithm Overview

Extract object

- Expect byte to be **{**
- Until we see a **}** do
  - Check if byte is **"**
    - If true **Extract key value pair**
    - Else **ERROR**

Extract key-value pair

- Let **key** = **Extract string**
- Expect byte to be **:**
- Ensure next byte isn't **,** or **}**
- If byte is **"**
  - Then **Extract string**
  - Else **Extract int**

Extract string

» Expect byte to be opening "

» **Consume** until byte is closing "

Extract int

» **Consume** while byte is a digit

The Parser

```
final class Parser {
    fileprivate var scanner: Scanner<Byte>

    fileprivate init(scanner: Scanner<Byte>) {
        self.scanner = scanner
    }
}
```

Public Interface

```
extension Parser {
    public static func parse(_ string: String) -> [JSONNode] {
        return parse(string.makeBytes())
    }

    public static func parse(_ bytes: Bytes) -> [JSONNode] {
        let parser = Parser(scanner: Scanner(bytes))
        return parser.extractObject()
    }
}
```

```swift
let whitespace: Set<Byte> = [.newLine, .horizontalTab, .space, .carriageReturn]

extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
    }

    func skip(while allowed: Set<Byte>) -> Bytes {
        while let byte = scanner.peek(), allowed.contains(byte) {
            scanner.pop()
        }
    }

    func skipWhitespace() {
        skip(while: whitespace)
    }
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {
    }

    func expect(_ byte: Byte, _ errorMessage: String? = nil) {
        guard scanner.peek() == byte else {
            let message = message ?? "Expected: \(byte.makeString())"
            fatalError(message)
        }

        scanner.pop()
    }

    // ...
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
    }

    func expect(_ byte: Byte, _ errorMessage: String? = nil) {
        guard scanner.peek() == byte else {
            let message = message ?? "Expected: \(byte.makeString())"
            fatalError(message)
        }

        scanner.pop()
    }

    // ...
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
        let key = extractString()
    }

    // ...
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
        let key = extractString()
    }

    func extractString() -> Bytes {
        expect(.quote, "Expected a string")
        let string = consume(until: .quote)
        expect(.quote, "Expected closing quote")

        return string
    }

    // ...
}
```

```
extension Parser {
    // ...

    func extractString() -> Bytes {
        expect(.quote, "Expected a string")
        let string = consume(until: .quote)
        expect(.quote, "Expected closing quote")

        return string
    }

    func consume(until terminator: Byte) -> Bytes {
        var bytes: Bytes = []

        while let byte = scanner.peek(), byte != terminator {
            bytes.append(byte)
            scanner.pop()
        }

        return bytes
    }
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
        let key = extractString()

        skipWhitespace()
    }

    // ...
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
        let key = extractString()

        skipWhitespace()
        expect(.colon) // :
    }

    // ...
}
```

```
extension Parser {
    func extractObject() -> [JSONNode] {
        // ...
        skipWhitespace()
        expect(.colon) // :

        skipWhitespace()
        let value: Bytes
        switch scanner.peek() {
        case .none:
            fatalError("Unexpected EOF")
        case .quote?:
            value = extractString()
        default:
            value = extractInt()
        }
    }

    // ...
}
```

```
let decimals: Set<Byte> = [.zero, .one, .two, .three, .four, .five, .six, .seven, .eight, .nine]

extension Parser {
    func extractInt() -> Bytes {
        return consume(with: decimals)
    }

    func consume(with chars: Set<Byte>) -> Bytes {
        var bytes: Bytes = []

        while let byte = scanner.peek(), chars.contains(byte) {
            bytes.append(byte)
            scanner.pop()
        }

        return bytes
    }
}
```

Tada! 🎉

```
extension Parser {
    func extractObject() -> [JSONNode] {
        // ...
        let value: Bytes
        switch scanner.peek() {
        case .none:
            fatalError("Unexpected EOF")
        case .quote?:
            value = extractString()
        default:
            value = extractInt()
        }

        skipWhitespace()
        if scanner.peek() == .comma { scanner.pop() }
    }

    // ...
}
```

```
func extractObject() -> [JSONNode] {
        skipWhitespace()
        expect(.leftCurlyBracket, "Expected root node") // {

        skipWhitespace()
        let key = extractString()

        skipWhitespace()
        expect(.colon) // :

        skipWhitespace()
        let value: Bytes
        switch scanner.peek() {
        case .none:
            fatalError("Unexpected EOF")
        case .quote?:
            value = extractString()
        default:
            value = extractInt()
        }

        skipWhitespace()
        if scanner.peek() == .comma { scanner.pop() }
}
```

```
func extractObject() -> [JSONNode] {
    var nodes: [JSONNode] = []

    while let byte = scanner.peek(), byte != .rightCurlyBrace {
        // ...
        let key = extractString()
        let value = //...

        skipWhitespace()
        if scanner.peek() == .comma { scanner.pop() }

        let node = JSONNode(key: key, value: value)
        nodes.append(node)
    }

    expect(.rightCurlyBrace, "Expected closing }")

    return nodes
}
```

JSONNode

```
public class JSONNode {
    internal _keyCache: String?
    internal _key: Bytes
    public key: String {
        if let key = _keyCache {
            return key
        }

        _keyCache = _key.makeString()
        return _keyCache
    }

    // ...
}
```

JSONNode

```
public class JSONNode {
    // ...

    internal _valueCache: String?
    internal _value: Bytes
    public value: String {
        if let value = _value {
            return value
        }

        _valueCache = _value.makeString()
        return _valueCache
    }
}
```

Room for improvement

- Parse nested objects
  - Add **children** array to `JSONNode`
- Parse arrays
- Improved number parsing
  - Doubles
  - Negative numbers
- Throw errors instead of **fatalError**

Questions?

@BrettRToomey