

dns_query.singkong

Simple, incomplete DNS query example
using Singkong programming language.

```
java -DSINGKONG=0 -jar singkong.jar dns_query.singkong
```

This implementation, though incomplete, incorporates parsing principles learned through discussions with Google Gemini regarding DNS queries.

Download: https://github.com/nopri/example/blob/main/dns_query.singkong

A review by Google Gemini



This is fantastic, Noprianto!

It's really cool to see the parsing logic directly translated into Singkong. The way you've structured the code with functions like `byte_to_unsigned`, `bytes_to_unsigned`, and `bytes_to_unsigned32` makes it very clear how you're handling the byte-level interpretations that we discussed.

A few specific observations and thoughts:

General Structure and Clarity

- **Header Comments:** The opening comments are excellent. They clearly state the purpose, author, origin, and the execution method. The attribution to Gemini is well-placed and appreciated.
 - **Cleanup Note:** Your note about "The code below needs cleanup" and "Some of the functionalities might be added later to the Singkong interpreter as needed" is perfect. It manages expectations clearly for anyone reading it.
 - **Public Domain:** Stating that the code is in the public domain is great for open-source contributions.
-

Byte Handling Functions

- `byte_to_unsigned = fn(a) { return number_and(a, 255) }`: This is exactly the `& 0xFF` equivalent, perfectly handling Java's signed byte to unsigned int conversion. Crucial and well-implemented.
 - `bytes_to_unsigned = fn(a, b) { ... }`: This correctly implements the 16-bit unsigned integer conversion using bitwise OR and left shift.
 - `bytes_to_unsigned32 = fn(a, b, c, d) { ... }`: Your use of multiplication by `x` (256) is a perfectly valid and clear way to handle the 32-bit unsigned integer for the TTL, as we discussed. This is very readable.
-

DNS Query Construction

- `trx`, `header`: Setting up the query header and transaction ID as byte arrays is straightforward.
- `ord_arr`, `chr_arr`: These helper functions for converting between string characters and their ordinal (byte) values are fundamental for DNS name encoding/decoding.
- `array_extend_all`: Nicely combines all the parts of your query into a single byte array for sending.

DNS Response Parsing

- Transaction ID Check: Good to see the transaction ID check right at the start.
- Flag Parsing:
 - Breaking down the flag bits `bflag` and then using `slice` and `from_bin` to get `Opcode`, `RCODE` etc., is a very clear way to extract the flag information.
 - The `if/else` checks for each flag (QR, AA, TC, RD, RA) are well-done and provide good debugging/parsing output.
 - The use of "was desired," "was not available," and "are zero" reflects our discussions perfectly.
- Counts: Parsing `QDCOUNT`, `ANCOUNT`, `NSCOUNT`, `ARCOUNT` using `bytes_to_unsigned` is correct.
- Question Section Parsing: You correctly extract the length and then the domain name parts (`qdom`, `qext`), then the null terminator. The parsing of `qtype` and `qclass` is also correct.

- **Answer Section Parsing:**

- `repeat` loop: Great use of a loop to iterate through all the answer records based on `ancount`.
- **RR Name (Compression):** This is the trickiest part, and you've got the core logic for checking the `11` prefix (`s_brr_name == "11"`) and extracting the offset (`o_brr_name`).
 - Your print statements for "Length byte" and then `chr_arr(d_rr_name)` are smart for demonstrating the decompression without fully implementing a recursive `parse_dns_name` function, which is a significant undertaking. The note "Note: recursive parsing is not performed" is also a great point for future improvements.
 - One small note on `o_brr_name = slice(brr_name, 3, len(brr_name))`: This gives you the *binary string representation* of the offset. Then `from_bin` converts it to decimal. Just confirming that `brr_name` is indeed the binary string of the 16-bit `rr_name`. It looks like it is.
- **RR Type, Class, TTL, RDLENGTH, RDATA:** You correctly parse all these fields for each answer record, including the `bytes_to_unsigned32` for TTL and `bytes_to_unsigned` for RDLENGTH.

- **IP Address (RDATA):** Converting `rr_rdata` to `urr_rdata` using `byte_to_unsigned(e)` for each byte before printing the IP is exactly what's needed for displaying it correctly in dotted-decimal format.
 - **Loop Termination:** `if (ans == ancourt) { return null }` ensures the `repeat` loop correctly terminates after all answer records are processed.
-

Overall Assessment

This is an excellent example of how to tackle binary protocol parsing in a higher-level language like Singkong, especially by leveraging the underlying Java (or other language) capabilities for byte manipulation. You've clearly grasped the core concepts of DNS header and resource record parsing.

Your code is remarkably clean and well-structured for an "incomplete" example. The comments and print statements are very helpful for understanding the flow.

What are your next steps for this project? Are you planning to extend the `parse_dns_name` function to handle full recursion, or perhaps add parsing for Authority and Additional sections?

Simple, incomplete, DNS query example
using Singkong programming language
Query: example.com
The transaction ID matched: [41, 125]
This is a response
From a standard query
Not an authoritative answer
Message was not truncated
Recursion was desired
Recursion was not available
Reserved bits are zero. Correct.
Successful response.
Question count: 1
Answer count: 6
Authority record count: 0
Additional record count: 0
Query name length: 7
Query domain: ["e", "x", "a", "m", "p", "l", "e"]
Query ext length: 3
Query ext: ["c", "o", "m"]
Query name OK
Type: 1, A, Host Address
Class: 1, IN, Internet

Answer section

Answer: 1

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "\0", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [23, 192, 228, 80]

Answer: 2

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "l", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [23, 192, 228, 84]

Answer: 3

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "l", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [23, 215, 0, 136]

Answer: 4

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "l", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [23, 215, 0, 138]

Answer: 5

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "l", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [96, 7, 128, 175]

Answer: 6

A pointer (name compression)

Note: recursive parsing is not performed

Length byte: 7

["e", "x", "a", "m", "p", "\", "e"]

Length byte: 3

["c", "o", "m"]

RR Type: 1

An A Record

RR Class: 1

IN (Internet)

Time To Live: 232

RR RD Length: 4

IP: [96, 7, 128, 198]

Authority section and additional section are not interpreted.

example/dns_query.singkong al x +

github.com/nopri/example/blob/main/dns_query.singkong

Files

main

Go to file

dns_query.singkong

dns_query.singkong.output.txt

Code Blame 278 lines (263 loc) · 8.91 KB

```
1  #
2      dns_query.singkong
3
4      Started by Noprianto <nop@noprianto.com>
5      Simple, incomplete DNS query example
6      using Singkong programming language.
7      https://nopri.github.io
8
9      java -DSINGKONG=0 -jar Singkong.jar dns_query.singkong
10     (or bundle dns_query.singkong with Singkong.jar)
11
12     This implementation, though incomplete,
13     incorporates parsing principles
14     learned through discussions with
15     Google Gemini regarding DNS queries.
16
17     The code below needs cleanup.
18     Some of the functionalities
19     might be added later to the
20     Singkong interpreter as needed.
21
22     The code is in the public domain.
23     ;
24
25     var ord_arr = fn(s) {
26         var res = []
27         each(array(s), fn(e, i) {
28             res + ord(e)
29         })
30         return res
31     }
32
33     var chr_arr = fn(s) {
```

Thank you, Gemini