Cheatography

Erlang Binaries Cheat Sheet

by Magnus (Fylke) via cheatography.com/93081/cs/20609/

Overview

Erlang can match binaries just as any list of things. `<<E1, E2, E3>> = Bin` divides the binary `Bin` into three elements of type `integer` of one byte each. This means that `Bin` has to be 24 bits long, or we get a `badmatch`. You can also make partial matches, in a `[Head | Tail]` fashion, by putting `/bitstring` on the last element, like so: `<<E1, E2, E3/bit str ing >> = Bin`. This is a **type modifier** and tells Erlang that there are two 8-bit elements, in `E1` and `E2` respectively, and then an undetermined amount of bits stored in `E3`..

Type Modifiers

Type	Size in bits	Remarks
`integer`	As many as it takes	Default size is 8 bits
`float`	**64**|32|16	Need to specify length if other than default: `<<A :16 /fl oat >>`
`binary|bytes`	8 per chunk	Anything matched must be of size evenly divisible by 8 (this is default)
`bitstring|bits`	1 per chunk	Will always match, use as Tail for a list
`utf8|utf16|utf32`	8-32, 16-32, and 32	`<<"a bc"/ utf 8>>` is the same as `<<$ a/utf8, $b/utf8, $c/utf 8 >>`
`signed|`**`unsigned`**	N/A	Default is unsigned
`big``|little|native`	N/A	Endianness - `native` is resolved at load time to whatever the CPU uses
`unit:IntLiteral`	N/A	Define a custom unit of length 1..256

Examples

Expression	Result
`<<97, 98, 99>>`	`<<"a bc">>` (turn off with `shell: str ing s(f a lse)`)
`<<A :2/ unit:6, B:1/un it: 4>> = <<7, 42>>`	`A = 114 B = 10`
`<<A :16 /fl oat >> = <<1, 17>>`	`1.6272 068 023 681 6 4e-5`
`<<A /si gne d>> = <<2 55>>`	`-1`
`<<A :16 /bi g>> = <<255, 0>>`	`65280`
`<<A :16 /li ttl e>> = <<255, 0>>`	`255`
`<<"p öpc örn " /ut f8>>`	How Erlang handles unicode

When constructing a binary, if the size of an integer `N` is too large to fit inside the given segment, the most significant bits are silently discarded and only the `N` least significant bits kept.

Segments

Binary Comprehension Example

Just like with lists, there is a notation for binary comprehension. Below i convert a 32 bit integer into a hex representation:

```
int_as _he x(Int) ->
        Int AsBin = <<I nt: 32> >,
        " 0x" ++ lists: fla tte n([ byt e_t o_h ex( <
> <= IntAsB in]).
byte_t o_h ex( <<N ibb le1:4, Nibble 2:4 >>) ->
        [in teg er_ to_ lis t(N ibble1, 16), intege r
)].
```

You can mix list- and binary comprehension: if the generator is a list, use `<-`, if it's a binary, use `<=`. If you want the result to be a binary, use `<<>>`, if you want a list, use `[]` around the expression.

Troubleshooting

Use the Erlang shell to trial and error you way to a correct expression. A understanding why your binaries are badmatching is `bit_size`:

```
bit_si ze( <<1 /in teg er> >). => 8 bit_si ze( <<< <1
str ing >>). => 2
bit_si ze( <<1.0/ flo at> >). => 64 bit_si ze( <<< <1
>>). => 16
```

A related one is `byte_size`:

```
MinByt esT oEn cod eNumber = byte_s ize (bi nar y:e r
ed(Num ber)).
```

Each segment in a binary has the following general syntax: `Value:Size/TypeSpecifierList`. The `Size` and `TypeSpecifier` can be omitted.
`Value` is either a literal or a variable, `Size` is multiplied by the unit in `TypeSp eci fie rList`, and can be any expression that evaluates to an `integer`[1]. Think of 'Size' as the number of items of the type in the 'TypeSpecifierList'
Contrived example: `<<X :4/ lit tle -si gne d-i nte ger --un it: 8>>` has a total size of 4*8 = 32 bits, and it contains a signed integer in little endian byte order.

[1] Mostly true, see Bit Syntax Expressions in Erlang documentation for complete picture.

By **Magnus** (Fylke)
cheatography.com/fylke/

Published 25th September, 2019.
Last updated 8th March, 2023.