Ultra...

Smallest. Web server. Evar.

FOSDEM
31st January 2015

by Steven Goodwin

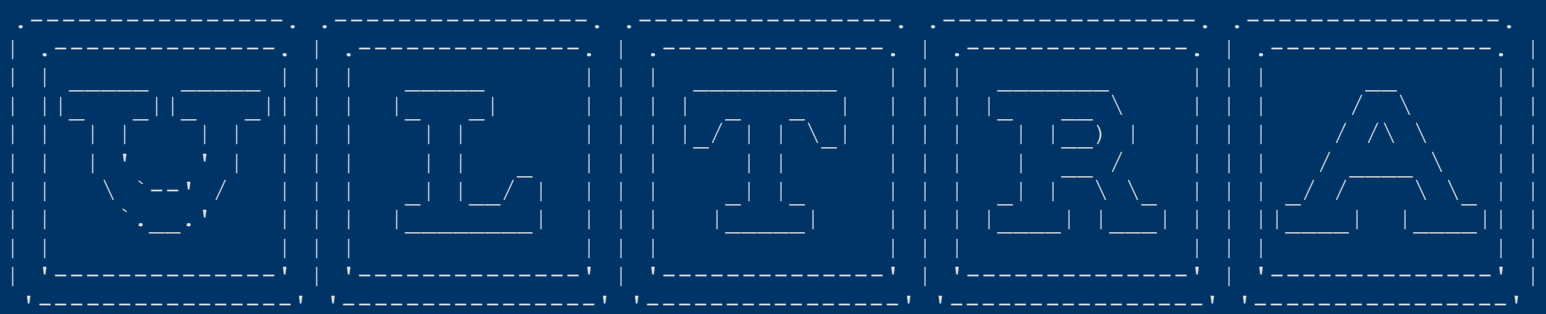
@marquis de geek

www.MarquisdeGeek.com

The Introduction Slide

- What it is
- Why I wrote it
- How I went about writing it
- If you'd like to follow along with the code:

https://github.com/MarquisdeGeek/ultra



The Ultra Conundrum?

- Web server – everything held in memory
- NoSQL data store – name:value pairs
- A data processing language
- SSI
- Multiple configurations
- Logging

- And all with a 51K binary

The Apache Comparison

The Apache Comparison

- The Ultra binary is **51K**.
- The README.html file is 36**K**

The nginx Comparison

The nginx Comparison

- The Ultra binary is **51K**.
- The nginx 'world' icon is **22K**

Why?

- Starting from scratch
- (Aka First principles)
- Something to learn from

Version 0.0

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
bzero((char *)&serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(port);

if (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))) {
    return -1;
}
listen(sockfd, 5);

while (1) {
    struct sockaddr_in clientaddr;
    socklen_t clientaddr_sz = sizeof(clientaddr);

    int cfd  = accept(sockfd, (struct sockaddr*)&clientaddr, &clientaddr_sz);

    char buffer[2048];
    read(cfd, buffer, sizeof(buffer)-1);

    // Generate page into char pageData[] based on contents of buffer

    send(cfd, pageData, strlen(pageData), MSG_MORE);
    close(cfd);

    waitpid(-1, NULL, 1/*WNOHANG*/);
}
```

The First Step Hypothesis

- Handle error codes
- Handle arbitrary data
- Handle configuration
- Logging
- Switch to C++

The Configuration Paradigm

Name value pairs

- string=string

Cut-price development:

- No whitespace padding around =
- Basic newline trim : *strchr(buffer,'\n') = '\0';
- Storage via STL : std:map<std::string,std::string>

Configuration - II

- How do I make the code more interesting?

1. Use it elsewhere – because it's elegant

Configuration - Elsewhere

```
$ more site/config/httpcodes.conf
200=OK
201=CREATED
202=Accepted
203=Partial Information
204=No Response
301=Moved
302=Found
303=Method
304=Not Modified
400=Bad request
401=Unauthorized
402=PaymentRequired
403=Forbidden
404=Not found
500=Internal Error
501=Not implemented
```

Configuration - Elsewhere

```
$ more site/config/mime.conf
css=text/css
ttf=font/ttf
otf=font/opentype
woff=application/x-font-woff
eot=application/vnd.ms-fontobject
svg=image/svg+xml
js=application/x-javascript
png=image/png
jpg=image/jpeg
gif=image/gif
```

Configuration - Elsewhere

```
requests_default.htm=1
requests_ultra.png=1
requests_style.css=1
requests_logo.png=1
requests_count=4
```

Configuration - Elsewhere

```
Example.com?arg1=Hello&arg2=FOSDEM

arg1=Hello
arg2=FOSDEM
```

The Reuse Experiment

2. Create a hierarchy – because I can

```
#configuration
develop.port=8088
live.port=80
```

I didn't explicitly handle the period any different to any other symbol.

“convention over configuration”

Reusing Code

- Do you call it 'live' or 'production'?

Reusing Code

- Do you call it 'live' or 'production'?

#configuration
production.port=80

ultra <site_dir> production

Reusing Code

- Do you call it 'live' or 'production'?

#configuration
production.port=80

ultra <site_dir> production

#configuration
live.port=80

ultra <site_dir> live

The meta data injection

- I wanting something like:

  <?php echo date(“Y”); ?>

- So I used:

  {(year)}

They're in settings.hpp if you're interested..

```
#define ULTRA_META_OPEN1    '{'
#define ULTRA_META_OPEN2    '('
#define ULTRA_META_CLOSE1   ')'
#define ULTRA_META_CLOSE2   '}'
```

Parsing meta data

```
unsigned char *
  UltraResponseText::parse(unsigned char *pData)
  {

  while(*pData) {

    if (*pData == ULTRA_META_OPEN1 &&
        *(pData+1) == ULTRA_META_OPEN2) {

    }
  }
}
```

This generates a list...

So...

```
The year is {(year)} !!111!!ZZ
```

- breaks down into a vector of 3 elements

```
The year is
{(year)}
!!111!!ZZ
```

Consequently...

```
The year is
{(year)}
!!111!!ZZ
```

- Each row above is an instance of `UltraLine`
- `UltraLine` has a `m_szLine` field
- `UltraLine` has an `m_bIsMeta` field
- `UltraLine` has a method called `process`

The Database Consideration

- I have a generic name=value store
- I have a way of rendering meta data into HTML

The Namespace Extension

- So to retrieve a field, and write it to the stream:

```
{(db:table.id.field)}
```

- I use a simple strchr() to find the colon
- Split the string
- Pass the RHS to the name=value code.
- Search for the LHS (e.g. 'db')
- Call the appropriate method to process 'db'

The Idleness Distraction

- If I can read a DB, can I write to it?
- Change the value
  - Increment
  - Decrement
  - Set to arbitrary value
  - Increase by an amount
  - Decrease by an amount
  - Multiple or divide by amount?
  - Should an amount be an integer or another DB value?

The YAGNI Reappearance

- Increment is simple an increase of 1
- Most operations operate on integers
- Who needs multiplication?
- So I started with three new basic constructs:

```
{(db:table.id.field)}   ; retrieve the field
{(db:table.id.field=n)} ; assign number 'n'
{(db:table.id.field+n)} ; add the number 'n'
{(db:table.id.field-n)} ; subtract the number
```

- Note the one character symbol for easy parsing
- Added a restriction, fields must be alphanumeric

The Variable Constant Paradox

- More conventions...
- ...by having a DB table called 'var'

- So to retrieve a variable, use

```
{(db:var.varname)}
```

- So change it with:

```
{(db:var.varname=12)}
```

The Virtualization Reduction

- `UltraLine` has a method called `process`

The Virtualization Reduction

- `UltraLine` has a method called `process`
- If we have a new class for each meta command, we just override the virtual method called `process`
- e.g.

```
void UltraRemapDatabaseFields::process(
  sgxString &resultPattern, const sgxString &source) {

     m_pData->getString(source, resultPattern)
}
```

- We can then map the LHS (e.g. “db”) to a class instance, and call `pLine->process`

The Hour-Long Feature Annihilation

- get:[argument name]
- config.dump:all
- db.dump:all
- stats.dump:all
- exec:[arbitrary command]
- redirect:[url]
- ssi:[filename]

(all setup in config.cpp)

The Server Initiation

- It was then I decided to serve files...

The Server Initiation

- It was then I decided to serve files...
- ...so I Googled 'Linux recursive file'
- ...found an ftw example
- ...typed the line `ftw(fileRoot.c_str(), buildCallback, 7);`
- ...wrote the callback to fopen ASCII files
- ...wrote the callback to fopen binary files
- ...then collapsed both into a utility ::slurp method
- ...and it was done!

The Convention Reappearance

- Convention over configuration is used in the directory used to server files.
- site
  - config
  - db
  - docs
    - assets
    - css
    - fonts
    - ssi

The List Deprecation

```
The year is {(year)} !!111!!ZZ
```

- breaks down into a list of

```
The year is
{(year)}
!!111!!ZZ
```

The Hierarchical Rationalisation

```
The year is {(year)} !!111!!ZZ
```

- breaks down into a hierarchy of

```
The year is
{(year)}
!!111!!ZZ
```

The Genius Re-normalization

- Using a hierarchy means I can do

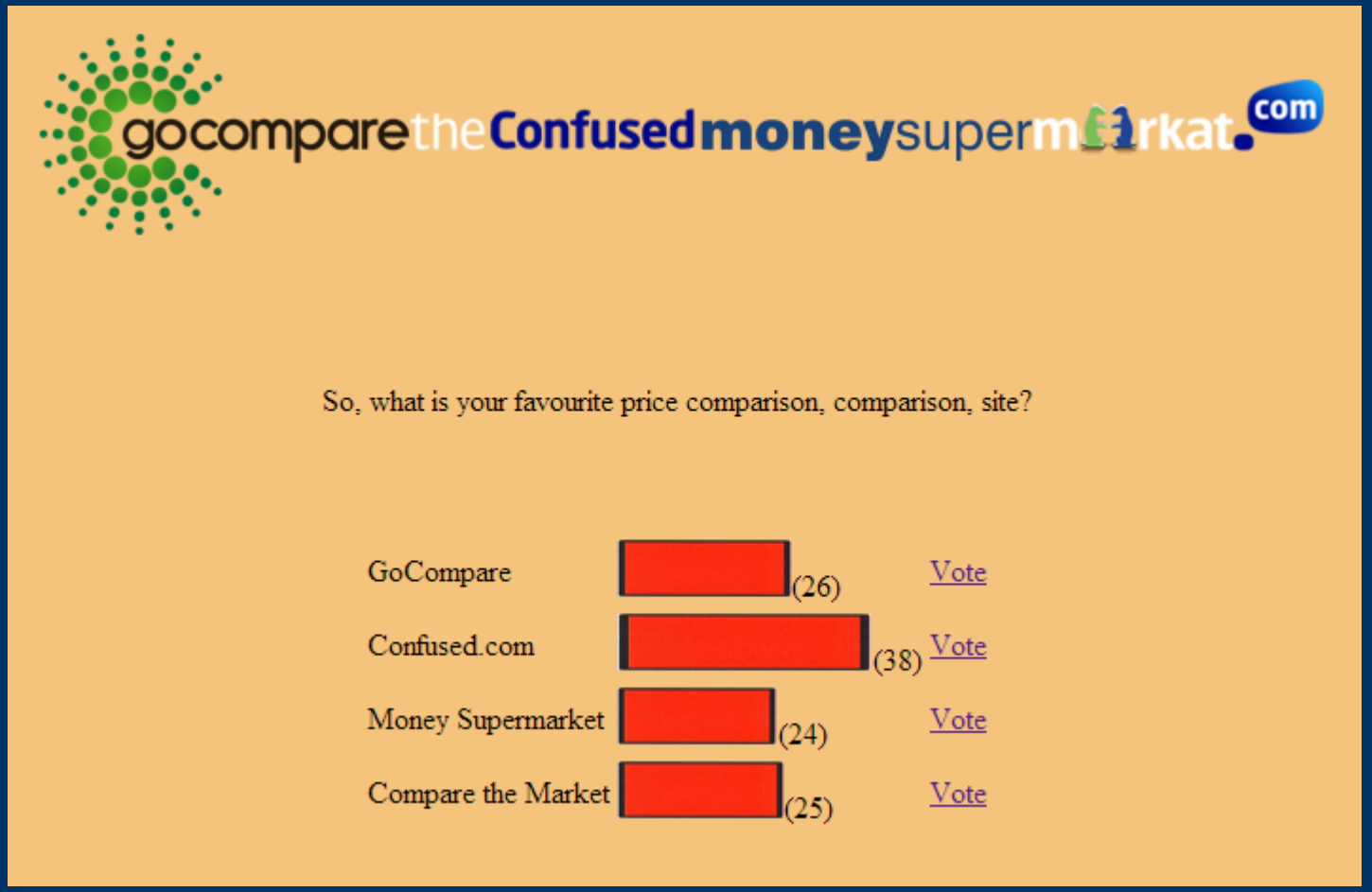
```
{(db:users.{(db:get.id)}.name)}
```

- And then, recursively, depth-first process each UltraLine

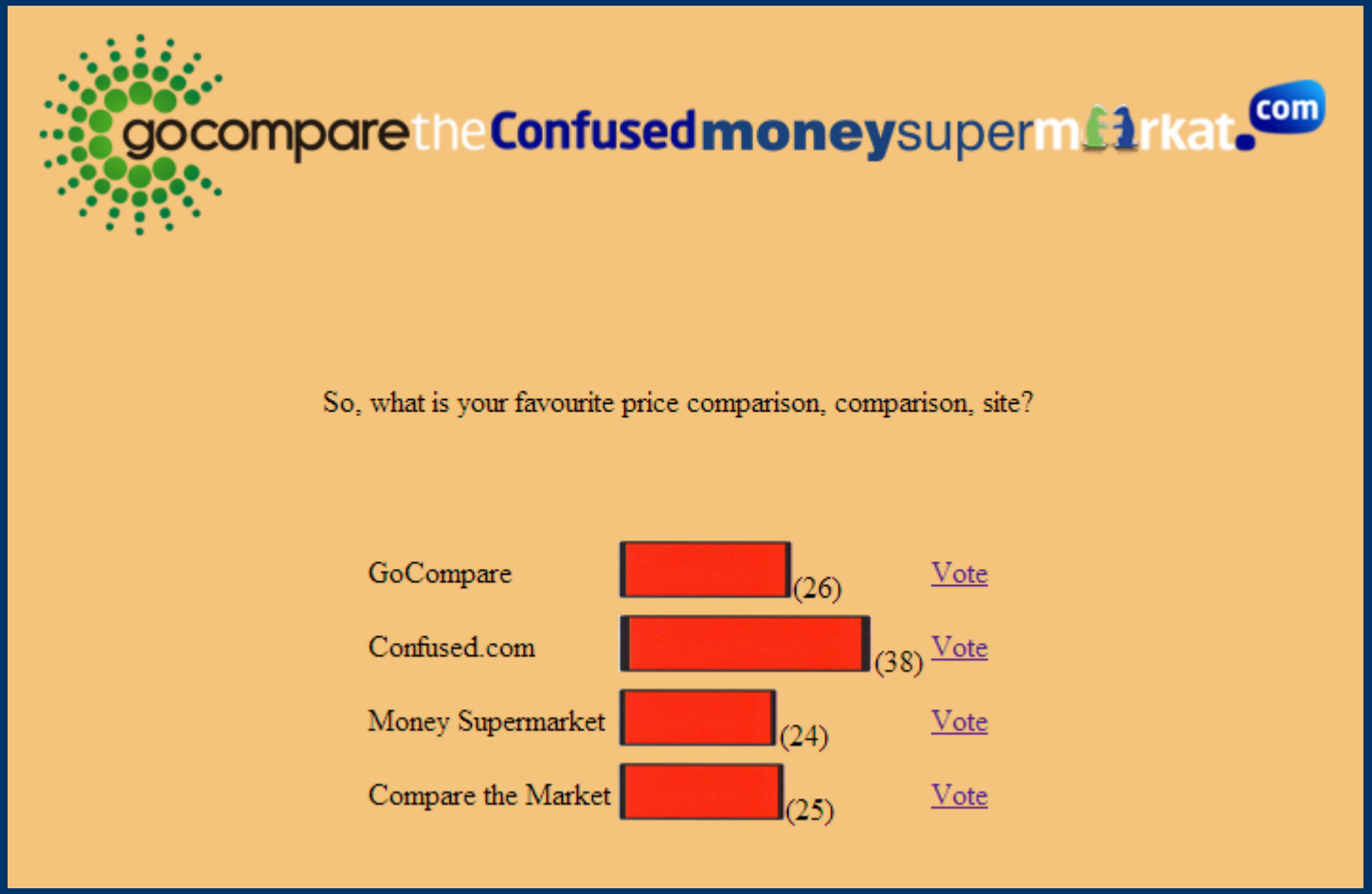
The Silliness Exemplification

gocomparetheconfusedmoneysupermeerkat.com

The Silliness Exemplification – pt II



The Silliness Exemplification – pt II



https://github.com/MarquisdeGeek/gocomparetheconfusedmoneysupermeerkat

The Show-off Amplification

- So we add conditional expressions

```
{(op.==:value1 value2 value3)}
{{op.if:condition if_true if_false_opt}}
```

- We have range checks

```
{(op.range:value min max)}
```

- We pretend it's a real DB, by allowing us to query number of “fields” in the DB

```
{(db.count:users)}
```

- Is it Turing-complete, yet?

The Example Example

- So we can do things like:

```
{(db!:var.id={(get:id)})}
{(db!:var.id={(op.range:{(db!:var.id)} 1
   {(db.count:users)})})}


Record : {(db:var.id)} of {(db.count:users)}

{(link:?id={(db:var.id-1)} Previous)}
{(link:?id={(db:var.id+1)} Next)}
```

The Example Additionment

- And navigation with:

```
{(db!:var.nav.page=1)}
{(ssi:header)}
```

(in and header)

```
<li {(op.if:{(op.==:{(db!:var.nav.page)} 0)}
  class="active")}>
  <a href="default.htm">About Ultra</a>
</li>
```

Conclusions

- Love is all you need

Conclusions

- A name=value store is all you need
- An example might still be running on:

  http://marquisdegeek.com:8088/

Any Questions?

@marquis de geek

www.MarquisdeGeek.com

Ultra Github:
https://github.com/MarquisdeGeek/ultra

The FOSDEM Diaries:
http://marquisdegeek.com/words_fosdem