SOFTWARE ENGINEERING

Software Engineering Stack Exchange is a question and answer site for professionals, academics, and students working within the systems development life cycle. Join them; it only takes a minute:

Sign up

Here's how it works:

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

Can C++ be used as a server-side web development language? [closed]

Ask Question

I'd like to get into web development using C++ as the "scripting language" on the server-side. My server infrastructure is *nix based, so doing web development in C++ on Azure is not applicable and C++/CLI ASP.NET is also not applicable.

Separate from legacy CGI applications, can web development be done using C++ ?

web-development c++

edited Jan 3 '15 at 16:42
user40980

asked Mar 1 '11 at 22:39
Scott Davies
347 1 4 5

closed as too broad by gnat, GlenH7, user40980, Thomas Owens ♦ Jan 3 '15 at 18:00

Please edit the question to limit it to a specific problem with enough detail to identify an adequate answer. Avoid asking multiple distinct questions at once. See the How to Ask page for help clarifying this question.

If this question can be reworded to fit the rules in the help center, please edit the question.

locked by Thomas Owens ♦ Jan 3 '15 at 18:00

This question exists because it has historical significance, but **it is not considered a good, on-topic question for this site**, so please do not use it as evidence that you can ask similar questions here. This question and its answers are frozen and cannot be changed. More info: help center.

33 Of course it's *possible*, the question is; is it *practical*? – Ed S. Mar 1 '11 at 22:44

See this question on stackoverflow.com. – kevin cline Mar 1 '11 at 22:50

24 You could use assembly as a server-side language if you were so inclined. – Channel72 Mar 2 '11 at 1:00

8 Or even Brainf*ck if `,` are `.` are redirected to a socket. – dan04 Mar 2 '11 at 1:07

4 This brings back horrible memories of the first web project I was involved in. CGI gateways to C code. I still shudder when I think about it! :-) – Brian Knoblauch Mar 2 '11 at 17:42

9 Answers

Absolutely.

There are even several frameworks for developing them, including Wt, cppcms, CSP, and others.

FastCGI's mainline implementation is in C, and directly supports several languages, including C++.

Any programming language that can parse strings can be used in CGI or a servlet. Any language that can implement bindings with C libraries can also be used to develop modules for ISAPI- or Apache-compatible servers.

It's not particularly easy in C++, and good templating engines are few and far between, but it can be done.

Of course, the question of whether this is a good idea is another matter entirely. :)

Do note: Major websites like Amazon.com, eBay, and Google do use C++ for parts of their infrastructure. Realize, however, that Google only uses C++ for speed-critical systems, and Amazon.com only relatively recently switched away from Lisp (which angered some of their senior staff :).

Facebook formerly compiled PHP to C++, but their HipHop compiler (written partly in C++) has since been retooled as a bytecode virtual machine.

edited Jul 29 '13 at 3:00

answered Mar 1 '11 at 22:44

greyfade
10.4k 2 34 43

2

+1 For citing various frameworks. You should add that it's common for (very) big web apps to be powered by c++ (and other languages) : amazon.com, google.com, now facebook.com via hiphop, etc. – Klaim Mar 1 '11 at 23:15

7 @Klaim: It's common, but it is by no means the rule. Amazon's architecture was historically Lisp-based and only recently rewritten in C++. Google's architecture involves Java, Python, and others almost as often as C++, all for various reasons. Facebook only uses hiphop now because they found out PHP doesn't scale. :) – greyfade Mar 2 '11 at 0:22

4 I agree, but I meant that they still are well-known examples of use of C++ - to answer directly the original question title. – Klaim Mar 2 '11 at 8:00

1 @johannes Facebook's scaling problem stems from the fact that they have to maintain an order of magnitude more servers than are otherwise necessary, specifically because of the poor performance of an optimized PHP script. Linear scaling simply isn't good enough for such a large infrastructure. But remember that the "shared nothing" approach is not exclusive to PHP. C and C++ can do that, too. – greyfade May 17 '13 at 19:33

1 @amar The thing is there is little return except in the 0.1% of apps that need that raw performance. You could serve in 1/3 of the time in most other languages with good web stack support. Banks, web advertisers, etc all serve massive scale without resorting to C++. Even Facebook. Twitter. StackOverflow. All do it in higher level languages. Its here to stay but it isn't going to become the majority again. Probably ever. – Rig Dec 16 '13 at 15:26

Why not?

The OkCupid dating site is created with C++. There are probably other examples.

There's also a Qt-inspired toolkit for developing web applications with C++ called Wt.

answered Mar 1 '11 at 22:45

Vitor Py
4,628 1 23 31

11 *"Why not"*? Because it's much easier using a language that has more support for this sort of thing. – Ed S. Mar 1 '11 at 22:47

5 @Ed S. As I and greyfade pointed out there are frameworks for developing web applications with C++. – Vitor Py Mar 1 '11 at 22:49

2 Yes, but again, are they as easy to use as more commonly used frameworks? I'm honestly asking, I'm not a web developer and I've never used them, but something tells me they are likely not as mature or widely used as (for example) their ruby/python/PHP counterparts. – Ed S. Mar 1 '11 at 22:55

3 @EdS.: Neither Ruby nor Python started with web frameworks. In fact it took a decade for those to appear. The frameworks are the mere consequence of enough people wanting to use language X for

problem Y. The same could happen for C++. Main reasons why it didn't: C++ is not managed, takes ages to compile and has a higher entrance barrier in general. – back2dos Nov 21 '13 at 20:20

1 @back2dos: Who said either language was developed with the web in mind? I certainly did not. I used the term "support". – Ed S. Nov 21 '13 at 23:33

If you're planning to write your web application in C++, it would be total waste to then interface it as CGI.

My suggestion would be to build it asynchronous using ASIO (Asynchronous I/O). With that you can build blazing fast web service (combine with nginx as a reverse-proxy and statics server for best effects); Combine that with template library like Wt and you're ready to serve tens of thousands request per second from a single server.

Whether this is practical alternative to dynamic language web framework is another issue.

edited Nov 21 '13 at 20:13

answered May 1 '12 at 16:40

vartec
19.4k 1 43 93

The short answer is, ANYTHING can be used to write a webpage provide it can read the input, write interpretable output, and is executable by the webserver.

Technically, any language can be used as a CGI script provided it:

1. Interprets all the inputs and environment as presented by the server
2. Outputs in a known markup language (generally html)
3. Can be ran by the server

There are also other ways too. Perl has the ability to be built as a wrapper around c/c++ code, acting as an interpreting layer between the two (and this is not including perl modules that are flat out compiled as C).

answered Mar 2 '11 at 23:05

Avatar_Squadron
301 1 4

in the beginning, it was quite common - the first web sites I worked on in the late 1990s were ISAPI extensions written in C++, and they worked quite well.

edited May 1 '12 at 15:30

3 isapi.dll anyone? – red-dirt Mar 5 '11 at 23:10

or ATLServer - atlserver.codeplex.com – gbjbaanb Apr 27 '12 at 23:41

It appears Microsoft thinks it can too. Check out Casablanca which is a new set of tooling for (it appears) Azure using C++.

> Casablanca is a project to start exploring how to best support C++ developers who want to take advantage of the radical shift in software architecture that cloud computing represents.
>
> Here's what you get with Casablanca:
>
> - Support for accessing REST services from native code on Windows Vista, Windows 7, and Windows 8 Consumer Preview by providing asynchronous C++ bindings to HTTP, JSON, and URIs
> - A Visual Studio extension SDK to help you write C++ HTTP client side code in your Windows 8 Metro style app
> - Support for writing native-code REST for Azure, including Visual Studio integration
> - Convenient libraries for accessing Azure blob and queue storage from native clients as a

> first class Platform-as-a-Service (PaaS) feature
> - A consistent and powerful model for composing asynchronous operations based on C++ 11 features
> - A C++ implementation of the Erlang actor-based programming model
> - A set of samples and documentation

answered May 1 '12 at 15:43
gbjbaanb
45.1k 6 89 162

For PHP you can write your own C/C++ extensions and get good performance benefits that way. If I had a really CPU intensive part of my web application I would probably make a small C++ library that offloaded that processing to the extension and then returned the result back to the PHP and then the PHP outputs it to the browser.

The other thing people don't often consider is offloading certain CPU processing to the client side e.g. JavaScript/jQuery. If I've got a web server, I might need a a 3Ghz CPU to do CPU intensive processing for a particular function (maybe some data processing). My company is paying money for that server each month to keep it running. If I want to scale up operations for a 100 concurrent users running that CPU intensive task at the same time then maybe I need multiple CPUs and servers, increasing the cost to my business. If I offload that CPU intensive task to the client side, then each user that visits the website can do their own processing on the data and I don't have to increase my server capability therefore saving me money.

After all with all with the collective power of 100+ desktops/tablets/mobiles doing the processing for you that's a lot more power than your server sitting in a datacenter somewhere costing your business money each month to keep running. Potentially then all your server would be doing would be retrieving data from the database, serving content and a bit of pre/post processing and validation of the data before storing back in the database. Obviously you wouldn't make the client side code too CPU intensive which might block/freeze the web browser UI, you might fire off an AJAX request to the server, retrieve the data and then process the data asynchronously client-side, leaving the web-browser UI completely usable.

edited May 17 '13 at 0:18

answered May 16 '13 at 23:46

zuallauz

171 2 9

Yes, it can be used. The others have mentioned various approaches. Here is my own approach. The advantage is that it is totally portable and self-contained, all the picked libraries only depend on

ANSI C. Setting it up only requires the Linux Kernel and a C compiler (And the obvious stuff like Busybox, bash, etc) (or Windows and a compiler), no extra libraries are needed, no fancy huge installations.

The result is a single program which is both a web server and a dynamic page generator (Substitutes both "apache" and "php"), it will also have database access via sqlite.

Libraries used:

- Mongoose - Http server
- Sqlite - SQL Database
- MiniXML - Makes dynamic page generation easier. sort of like Javascript's `createElement`

The rest of this answer is a complete set-up guide for Linux. Both SQlite and MiniXML are optional, but the guide covers the full installation. It is up to you to comment out the non needed parts if you're interested in disabling either sqlite or MiniXML.

1. Download the 3 libraries

- Mongoose from Github (Hit the "download ZIP")
- Sqlite from sqlite.org (You will need Source code > sqlite-amalgamation-< version >.zip)
- MiniXML from msweet.org (mxml-< verison >.tar.gz)

2. Prepare your folder

- Create an empty folder (We'll call it the main folder)
- Put the following files in it:
    - From the sqlite tar.gz: `sqlite3.c` , `sqlite3.h`
    - From the Mongoose zip: `mongoose.c` , `mongoose.h`
    - From the mxml tar.gz: `mxml.h`

3. Compile mxml

You may have noticed mxml.c is missing, this is because we need to create a static mxml library. Go to the folder where the mxml tar.gz was downloaded and perform:

```
tar -xvf mxml-<version>.tar.gz #Extract the tar
cd mxml-<version> #Go to the newly extracted directory
./configure #prepare the compiler
make #compile, you may need to install "make" first.
```

Once the compilation is finished, many files will be generated, the only file of interest to us is `libmxml.a` , copy that file into the main folder.

3.1 Doublecheck

Check that the main folder has the following:

- For mongoose: `mongoose.c, mongoose.h`
- For mxml: `libmxml.a, mxml.h`
- for sqlite: `sqlite.c, sqlite.h`

4. main.c

Let's create the actual program, create a `main.c` file in the main folder, here is a skeleton for you to get started.

```
#include <string.h>
#include <stdio.h>

#include "mongoose.h"
#include "mxml.h"
#include "sqlite3.h"

/***Sqlite initialization stuff***/
//comment out everything sqlite related if you don't w
function and the include "sqlite3.h"
static int callback(void * custom, int argc, char **ar
char *zErrMsg = 0;
sqlite3 *db;
int rc;

/***Just some laziness shortcut functions I made***/
typedef mxml_node_t * dom; //type "dom" instead of "mx
#define c mxmlNewElement   //type "c" instead of "mxml
inline void t(dom parent,const char *string) //etc
{
    mxmlNewText(parent, 0, string);
}

//type "sa" instead of "mxmlElementSetAttr"
inline void sa(dom element,const char * attribute,cons
{
    mxmlElementSetAttr(element,attribute,value);
}




//The only non boilerplate code around in this progran
void serve_hello_page(struct mg_connection *conn)
{
    char output[1000];
    mg_send_header(conn, "Content-Type", "text/html; cha
    mg_printf_data(conn, "%s", "<!DOCTYPE html>");
    //This literally prints into the html document


    /*Let's generate some html, we could have avoided
     * xml parser and just spat out pure html with mg_
     * e.g. mg_printF_data(conn,"%s", "<html>hello</ht

    //...But xml is cleaner, here we go:
            dom html=mxmlNewElement(MXML_NO_PARENT,"ht
                dom head=c(html,"head");
                    dom meta=c(head,"meta");
                    sa(meta,"charset","utf-8");
                dom body=c(html,"body");
                    t(body,"Hello, world<<"); //The <
                    c(body,"br");
                    t(body,"Fred ate bred");
                dom table=c(body,"table");
                sa(table,"border","1");

                //populate the table via sqlite
                rc = sqlite3_exec(db, "SELECT * from m
&zErrMsg);
                if( rc!=SQLITE_OK )
                {
                    fprintf(stderr, "SQL error: %s\n",
                    sqlite3_free(zErrMsg);
                }

            mxmlSaveString (html,output,1000,  MXML_NC
            mg_printf_data(conn, "%s", output);
            mxmlDelete(html);
}

//sqlite callback
static int callback(void * custom, int argc, char **ar
{
    //this function is executed for each row
    dom table=(dom)custom;

    dom tr=c(table,"tr");
    dom td;
    int i;
    for(i=0; i<argc; i++)
    {
        td=c(tr,"td");
        if (argv[i])
            t(td, argv[i]);
        else
            t(td, "NULL");

        printf("%s == %s\n", azColName[i], argv[i] ? a
    }
     printf("\n");
     return 0;
}


static int event_handler(struct mg_connection *conn, e
{
    if (ev == MG_AUTH)
    {
        return MG_TRUE;   // Authorize all requests
    }
    else if (ev == MG_REQUEST)
    {
        if (!strcmp(conn->uri, "/hello"))
        {
            serve_hello_page(conn);
            return MG_TRUE;   // Mark as processed
        }
    }
    return MG_FALSE;  // Rest of the events are not pr

}

int main(void)
{
    struct mg_server *server = mg_create_server(NULL,
    //mg_set_option(server, "document_root", "."); //p
serving
    //TODO can I allow file listing without dir listin
    mg_set_option(server, "listening_port", "8080");


    rc = sqlite3_open("db.sqlite3", &db);
    if( rc )
    {
        fprintf(stderr, "Can't open database: %s\n"
        sqlite3_close(db);
        return(1);
    }

    printf("Server is running on port 8080!\n");
    for (;;)
    {
        mg_poll_server(server, 1000);  // Infinite
    }
    mg_destroy_server(&server);
    sqlite3_close(db);

    return 0;
}




/*
 * useful stuff:
 * mg_send_file(struct mg_connection *, const char
```

Finally, compiling!

Let's compile. `cd` to your main folder and execute these:

```
gcc -c main.c
gcc -c mongoose.c
gcc -c sqlite3.c
gcc -o server.out main.o mongoose.o sqlite3.o -ldl
```

Now, execute server.out with `/server.out` , and navigate to `localhost:8080/hello`

Done :)

edited Jan 3 '15 at 13:41

Hello World
176 4

1 github.com/bel2125/civetweb – Hey Jan 3 '15 at 16:13

@Hey: Thanks for pointing out this Mongoose alternative, I always prefer community-driven projects. I will probably replace Mongoose with Civetweb in my answer after I thoroughly test it. – Hello World Jan 3 '15 at 17:56

I guess that several embedded systems (e.g. routers, printers, ...) have some C++ driven web server.

In particular, you could use some HTTP server library like libonion to add some web capabilities to some C or C++ program, or to develop a light server with some web interface.

Some people are coding their Web server or their HTTP interface in Ocaml using Ocsigen. Not every web thing is PHP. And with FastCGI you could some dynamic web processing in/to your application.

edited Jan 3 '15 at 15:42

answered Jan 3 '15 at 15:36

Basile Starynkevitch
26.9k 5 59 98