webcppd 应用服务器

version 1.0.5

使用手册

admin@webcpp.net

2017 年 1 月 3 日

目录

插图

表格

源代码

摘要

webcppd 是一款用 C++ 语言写成的应用服务器软件。它被设计为 C++ 动态库容器，用来加载表现为 C++ 动态库的 Web 组件。通过从 Web 组件获得事先实现的类，webcppd 能够以优异的性能对 HTTP 请求做出合乎预期的响应。

1 概述

webcppd 是为了把 C++ 语言变成一种 Web 开发语言而准备的。利用 webcppd，C++ 语言能够轻松地与 HTML、JS、CSS 打成一片，无障碍地融入到 Web 开发的实践当中。

因而，你能够把 C++ 语言当作是更好、更快的 PHP 语言来对待。

当然，你也可以把 webcppd 当作 C++ 版的 tomcat。

表1列出了 7 种可使用 C/C++ 语言进行 Web 开发的框架，包括 webcppd 在内。它们都具有自己的独特之处。有兴趣的读者可以关注它们，根据自己的喜好并权衡多方面因素，作出最合适的开发选择。以下为 webcppd 的主要特性，希望有助于读者了解 webcppd:

项目	网址	语言
tntnet	http://www.tntnet.org/	C++
kore	https://kore.io/	C/C++
crow	https://github.com/ipkn/crow	C++
treefrog	http://www.treefrogframework.org/	C++
cppcms	http://cppcms.com	C++
nxweb	http://nxweb.org/	C
webcppd	https://github.com/webcpp/webcppd	C++

表 1: 常见 C/C++ Web 开发框架

- HTTP/1.0 and HTTP/1.1
- WebSocket
- HTTPS
- C++
- 会话
- 缓存
- ip 动、静态黑名单
- 正则路由
- 防盗链
- 日志
- more

2 安装

webcppd 的安装方法非常简单。
centos 用户安装方法如下：

第一步 `yum install epel-release`

第二步 `./centos-install-depend.sh`

第三步 `git clone https://github.com/webcpp/webcppd.git`

第四步 `make && sudo make install`

ubuntu 用户安装方法如下：

第一步 `git clone https://github.com/webcpp/webcppd.git`

第二步 `./ubuntu-install-depend.sh`

第三步 `make && sudo make install`

按照上面介绍完成操作之后，若无意外，`webcppd` 应用服务器已经安装成功了。安装说明如下：

参数配置 `/etc/webcppd` 目录为参数配置目录，下面包含三个文件

- `webcppd.properties` 服务器初始化配置
- `ipdeny.conf` 静态 IP 黑名单配置
- `route.conf` 正则路由配置

运行配置 `/var/webcppd/(www|log|mod|tpl|cert)` 目录为运行配置目录

- `www` 根目录，存放静态资源
- `log` 日志目录，存放日志文件
- `mod` 组件目录，存放 Web 组件即 C++ 动态库
- `tpl` 模板目录，存放模板文件
- `cert` 数字证书目录, 存放开启 `https` 所需文件，其中的`gencert.sh` 脚本可以用于创建自签名的数字证书

pidfile `/var/run/webcppd.pid`

程序 `/usr/local/bin/webcppd`

程序符号链接 `/usr/bin/webcppd`

控制脚本 `/usr/bin/webcppd-ctrl.sh` 参数：`start|stop|uninstall`

配置备份还原脚本 `/usr/bin/webcppd-backup.sh` 参数:`backup|restore`

systemd 控制脚本 `/usr/bin/webcppd-service.sh` 参数：

启动 start

停止 stop

重启 restart

状态 status

systemd 配置文件 /etc/systemd/system/webcppd.service

开发包头文件集 /usr/local/include/webcppd

既然已经安装成功，就立即体验下吧。

```
sudo systemctl start webcppd
#or
sudo service webcppd start
#or
sudo webcppd-service.sh start
#or
sudo webcppd-ctrl.sh start
```

推荐使用 systemctl 来控制 webcppd。如果需要 webcppd 开机自启动，那么需要运行：

```
sudo systemctl enable webcppd
```

webcppd 默认开启 https 安全链接，监听 443 端口，用浏览器访问https://localhost 即可。如果不需要安全链接，可通过http.enableSSL=false 关闭安全链接。http 链接默认监听 80 端口，所以直接用浏览器访问http://localhost 即可。

如果出现如图1的界面，那么恭喜你:webcppd 安装成功了!

3 配置

webcppd 既然已经安装好，Web 组件 demo.so 也已经安装妥当，现在来看看 webcppd 的配置文件。

配置文件都安装在/etc/webcppd 目录下面：

- webcppd.properties 服务器初始化配置
- ipdeny.conf 永久 IP 黑名单配置
- route.conf 路由配置

webcppd 启动时，会读入 webcppd.properties 文件以初始化服务器。该文件指定了 ipdeny.conf 和 route.conf 的绝对路径。因此，一旦服务器正常启动，以上三个配置文件的所有设置均已生效，不能修改，除非修改配置后重启服务器。

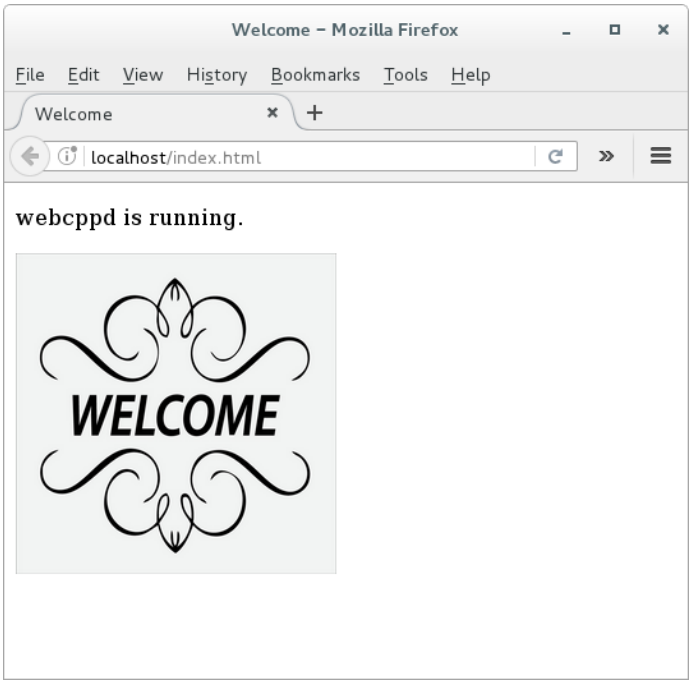

图 1: webcppd 欢迎界面

3.1 核心配置

webcppd.properties 文件规定了 webcppd 的一切:

源代码 1: webcppd.properties

```
1  ##############
2  # HTTP[S] server
3  ##############
4
5  # enable https,default : true
6  ;http.enableSSL=true
7  ;http.certPrivateKeyFile=/var/webcppd/cert/server.key
8  ;http.certCertificateFile=/var/webcppd/cert/server.crt
9  ;http.certRootCertificateFile=/var/webcppd/cert/rootCA.pem
10 # client ssl check,default : false
11 ;http.certCheckClient=false
12
13 # enable hotlinking,default: true
14 ;http.enableHotlinking=true
15 # hotlinking host regex match
```

```
;http.matchHotlinking=(.+\.)?(localhost|baidu|google|bing|yahoo|so|sogou)(\.(com|net))?

# bind ip,default: 127.0.0.1
;http.ip = 127.0.0.1

# Listen port, http default: 80,https default:443
;http.port = 80

# Maximum requests queue size, default: 1000
;http.maxQueued = 1000

# Maximum working threads count, default: 1023
;http.maxThreads = 1023

# Server Name,default:webcppd
;http.serverName=webcppd
# Server version, default: webcppd/1.0.5
;http.softwareVersion = webcppd/1.0.5

#keepAlive ,default true
;http.keepAlive=true
#max keepAlive Requests number;default 0,the mean is unlimited
;http.maxKeepAliveRequests=0
# keepalive timeout;default 60 second
;http.keepAliveTimeout=60

# http connect timeout ;default 60 second
;http.timeout=60


# handler Library directory,default: /var/webcppd/mod
;http.libHandlerDir = /var/webcppd/mod

#ip access check,default
     :ipEnableCheck(true),ipDenyExpire(3600),ipMaxAccessCount(100),ipAccessInterval(30),in seconds
;http.ipEnableCheck=true
;http.ipDenyExpire = 3600
;http.ipMaxAccessCount = 100
;http.ipAccessInterval = 30

#default /etc/webcppd/ipdeny.conf
;http.ipDenyFile=/etc/webcppd/ipdeny.conf

# docroot directory, default: /var/webcppd/www
# static file expires, default: 3600s
```

```
;http.docroot = /var/webcppd/www
;http.expires = 3600

# dynamic page cache expries,default:600s
;http.cacheExpires =600

#list static directory,default:false
;http.enableIndex=false

# template directory ,default: /var/webcppd/tpl
;http.tplDirectory = /var/webcppd/tpl

#upload setting,1mb
;http.uploadMaxSize=1048576
;http.uploadAllowType=image/png|image/jpeg|image/gif|image/webp|application/zip
;http.uploadDirectory=/var/webcppd/www/upload

#logger directory,default:/var/webcppd/log
;http.logDirectory=/var/webcppd/log
#logger file size,default:1 MB
;http.logFileSize=1 M
#logger file Compress,default:true
;http.logCompress=true
#maximum number of archived log files. default:10
;http.logPurgeCount=10

# proxy server pass real ip,default:proxyUsed(false),proxyServerRealIpHeader(X-Real-IP)
;http.proxyUsed=false
;http.proxyServerRealIpHeader=X-Real-IP

#route configure
;http.route=/etc/webcppd/route.conf

# secret-key
http.secretKey = a-&$bcDe#%@*#mGhk_A

# mysql configure
mysql.host=localhost
mysql.port=3306
mysql.user=root
mysql.password=123456
mysql.db=test
mysql.default-character-set=utf8
mysql.compress=true
mysql.auto-reconnect=true
```

```
105
106 # smtp configure
107 smtp.host=smtp.exmail.qq.com
108 smtp.port=465
109 smtp.user=admin@webcpp.net
110 smtp.password=123456
```

其中的每一项都具有默认值；当然，每一项都是可以自定义的。自定义的方法很简单：把行首的；分号去掉，设置需要的值即可。分号；表示该行取默认值。

需要强调的是，自定义配置必须按照默认值表示的值类型和格式来设置值，否则不能到达需要的效果。

3.2 IP 黑名单配置

核心配置源代码1中关于 IP 黑名单配置的项目有五个，分别是： 页6

```
http.ipEnableCheck=false

http.ipDenyExpire = 3600

http.ipMaxAccessCount = 100

http.ipAccessInterval = 30

http.ipDenyFile=/etc/webcppd/ipdeny.conf
```

webcppd 包含两种 IP 黑名单机制。第一种是动态黑名单机制，它能够自动检测活动的 IP 连接是否为恶意访问或者是否属于机器人行为。该机制通过前四项进行配置。默认情况下没有启用该机制[1]，如 http.ipEnableCheck 设定的。要启用该机制，只要将 http.ipEnableCheck 的 false 改为 true 即可。http.ipDenyExpire 表示若活动 IP 被认为是不安全的，则将在接下来的 3600 秒内禁止该 IP 访问服务器。http.ipMaxAccessCount 和 http.ipAccessInterval 规定了发现不安全 IP 的具体方法，也就是：如果一个 IP 在 http.ipAccessInterval 秒以内访问服务器达到 http.ipMaxAccessCount 次，那么服务器就会认为该 IP 是不安全的。

第二种黑名单机制是静态的，由它规定的黑名单 IP 在服务器运行的整个生命期中都不能进行访问。它是由 http.ipDenyFile 指定的。默认情况下，/etc/webcppd/ipdeny.conf 是一个空白文件。如果有需要，逐行将不安全 IP 写入该文件并重启服务器即可。

3.3 路由配置

核心配置源代码1中关于路由配置的项目如下： 页6

[1] 作压力测试时务必禁用该机制。

```
http.route=/etc/webcppd/route.conf
```

默认情况下，路由配置文件/etc/webcppd/route.conf 也是一个空白文件。要实现自己的路由规则，只需将相关规则按键值对的方式逐行写入即可。需要注意的是，键与值需要用逗号，或者分号；来分割表示，并且，键必须是正则表达式。比如:

```
^/(hello|demo|test)/?,webcppd::hello
```

上面三条规则表示用路径/hello、/demo、/test 都可以访问类 webcppd::hello。

如果同一个键映射到多个不同的类，比如:

```
^/a/?,webcppd::a
^/a/?,webcppd::b
```

那么，实际上生效的是行号最小的那一条规则。在上面的例子里，有效的就是第一条规则，第二条将被忽略。

如果想服务器忽略某条路由规则，只需在该行行首添加 # 号即可。

修改路由配置后，别忘了重启服务器使之生效。

3.4 防盗链配置

防盗链配置由以下两条指定:

```
;http.enableHotlinking=true
;http.matchHotlinking=(.+\.)?(localhost|baidu|google|bing|yahoo|so|sogou)(\.(com|net))?
```

第一条决定是否开启防盗链功能，默认是开启的。第二条决定什么样的链接不算盗链。默认来源链接主机匹配 localhost 就算合法。但是，这是为了方便本地测试。真实运行环境中，你可以参考以上第二条改写。比如把其中的 localhost 改为你自己的域名部分。像我，改成了 webcpp。其中的搜索引擎域名部分是为了方便把网站开放给搜索引擎。

3.5 安全链接

需要安全链接 https 很简单，把 http.enableSSL 设置为 true，并指定安全证书即可。这里要注意，只有当你需要双向安全链接时才设置 http.certCheckClient 为 true。

```
;http.enableSSL=true
;http.certPrivateKeyFile=/var/webcppd/cert/server.key
;http.certCertificateFile=/var/webcppd/cert/server.crt
;http.certRootCertificateFile=/var/webcppd/cert/rootCA.pem
;http.certCheckClient=false
```

现在，大家都可以很方便的使用letsencrypt颁发的免费安全证书了。需要使用的话，只需先执行`letsencrypt/install.sh`，然后如下配置即可 (把其中的域名`webcpp.net` 换成你申请的域名，具体可查看`/etc/letsencrypt/live` 目录)：

```
http.certPrivateKeyFile=/etc/letsencrypt/live/webcpp.net/privkey.pem
http.certCertificateFile=/etc/letsencrypt/live/webcpp.net/fullchain.pem
http.certRootCertificateFile=/etc/letsencrypt/live/webcpp.net/cert.pem
```

3.6 列出目录

列出静态文件目录的功能默认是关闭的。该功能可通过以下项目开启:

```
http.enableIndex=true
```

4 起步

`webcppd` 是用 `C++` 语言实现的，基于著名的 `C++` 编程框架 `Poco`，该框架的结构如图2。

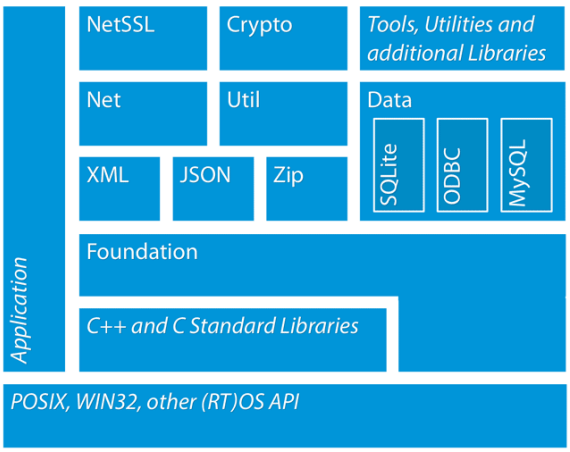

图 2: `Poco` 框架结构图

`webcppd` 约定的动态库写法，必须是相容于 `Poco` 的写法。`webcppd` 约定每一个被调用的类都必须是 `Poco::Net::HTTPRequestHandler` 的子类，该子类必须实现 `handleRequest` 方法。但是，你不必这样做。`webcppd` 为你准备了更好的基础类，即`webcppd::root_view`。

例如将被调用的类是 `webcppd::hello`:

源代码 2: hello.hpp

```
#ifndef HELLO_HPP
#define HELLO_HPP

#include <webcppd/root_view.hpp>

namespace webcppd {

    class hello : public root_view {
        void do_get(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse& response)
             override;

        void do_post(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse& response) {
            this->error(request, response);
        }

        void do_delete(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
             response) {
            this->error(request, response);
        }

        void do_put(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse& response) {
            this->error(request, response);
        }
    };
}

#endif /* HELLO_HPP */
```

源码2表示 `webcppd::hello` 仅对 `GET` 方法请求做出响应。其他类型的响应一概加以回绝。

源代码 3: hello.cpp

```
#include "hello.hpp"

namespace webcppd {

    void hello::do_get(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
         response) {
        response.setContentType("text/plain;charset=utf-8");
        response.send() << "Hello world.";
    }

}
```

源码3说明 webcppd::hello 只做出作最简单的回应。这是第一步。

第二步是把该类按照 Poco 约定的方式包装到动态库中。为此，需要这样：

源代码 4: main.hpp

```
#ifndef MAIN_HPP
#define MAIN_HPP

#include "hello.hpp"
#include "info.hpp"


#endif /* MAIN_HPP */
```

源代码 5: main.cpp

```
#include <Poco/ClassLibrary.h>
#include <Poco/Net/HTTPRequestHandler.h>

#include "main.hpp"

POCO_BEGIN_MANIFEST(Poco::Net::HTTPRequestHandler)

POCO_EXPORT_CLASS(webcppd::hello)
POCO_EXPORT_CLASS(webcppd::info)

POCO_END_MANIFEST
```

源码4和源码5是统一的类装载器。不管你写多少个响应类，也不管响应类有多么复杂，统统放置在这个类装载器中。写法很简单: 在main.hpp 中导入头文件，例如：

```
#include "hello.hpp"
#include "x.hpp"
#include "xx.hpp"
#include "xxx.hpp"
#include "xxxx.hpp"
```

然后在main.cpp 中导入类名

```
POCO_EXPORT_CLASS(webcppd::hello)
POCO_EXPORT_CLASS(webcppd::x)
POCO_EXPORT_CLASS(webcppd::xx)
POCO_EXPORT_CLASS(webcppd::xxx)
POCO_EXPORT_CLASS(webcppd::xxxx)
```

第三步是添加路由规则：

```
^/helloword/?,webcppd::hello
```

该规则表示：如果访问路径匹配模式/helloworld/?，就调用 webcppd::hello 类进行响应。

源码准备好了，现在准备一个 Makefile：

源代码 6: Makefile

```
MOD=mod/demo.so

MODSRC=$(wildcard *.cpp)
MODOBJ=$(patsubst %.cpp,%.o,$(MODSRC))

CC=g++
CXXFLAGS+=-O3 -std=c++11 -fPIC -Wall `pkg-config --cflags opencv cryptopp`
LIBS+=-lPocoDataMySQL -lPocoData -lPocoJSON -lPocoNet -lPocoUtil -lPocoFoundation
LIBS+=`pkg-config --libs opencv cryptopp`
LIBS+=-lqrencode
LDFLAGS+=-shared




all:$(MOD)

$(MOD):$(MODOBJ)
    $(CC) -o $@ $^ $(CXXFLAGS) $(LIBS) $(LDFLAGS)

clean:
    rm -f $(MODOBJ) $(MOD)


install:
    systemctl stop webcppd
    install mod/demo.so /var/webcppd/mod
    mkdir -pv /var/webcppd/www/assets/demo
    install assets/*.* /var/webcppd/www/assets/demo
    mkdir -pv /var/webcppd/tpl/demo
    cp -Ru tpl/*.* /var/webcppd/tpl/demo
    systemctl start webcppd
```

然后 make,make install。如此，一个标准的 Web 组件 demo.so 就开发安装完成了。访问http://localhost/helloworld 或者https://localhost/helloworld 即可看到结果。

5 示例

本节是一个完整的示例。演示的是使用视图配置文件配置模板及其变量，并进行视图缓存。该演示提供一个访问 webcppd 系统变量的接口。只要用户通过浏览器访问路径/info，就可以查看所有系统变量。

5.1 视图类

首先，需要一个 webcppd::info 类。源码如下。

源代码 7: info.hpp

```
1  #ifndef INFO_HPP
2  #define INFO_HPP
3
4  #include <webcppd/root_view.hpp>
5
6  namespace webcppd {
7
8      class info : public root_view {
9          virtual void do_get(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
               response);
10
11         virtual void do_post(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
               response) {
12             this->error(request, response);
13         }
14
15         virtual void do_delete(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
               response) {
16             this->error(request, response);
17         }
18
19         virtual void do_put(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
               response) {
20             this->error(request, response);
21         }
22     };
23 }
24
25 #endif /* INFO_HPP */
```

源码7表示 webcppd::info 会对 GET 方法请求做出响应。

源代码 8: info.cpp

```cpp
#include "info.hpp"

namespace webcppd {

    void info::do_get(Poco::Net::HTTPServerRequest& request, Poco::Net::HTTPServerResponse&
         response) {
        std::string cacheKey(this->create_cache_key(request, response));
        if (root_view::root_cache().has(cacheKey)) {
            response.send() << *root_view::root_cache().get(cacheKey);
            return;
        }

        Poco::SharedPtr<Kainjow::Mustache::Data> data = this->tpl_ready("/demo/config.json",
             "demo.GET");

        Poco::Util::AbstractConfiguration::Keys rootKeys, configKeys;
        this->app.config().keys(rootKeys);
        Kainjow::Mustache::Data tableList(Kainjow::Mustache::Data::List());
        for (auto& rootItem : rootKeys) {
            Kainjow::Mustache::Data table, trList(Kainjow::Mustache::Data::List());
            this->app.config().keys(rootItem, configKeys);
            for (auto& confItem : configKeys) {
                Kainjow::Mustache::Data kv;
                kv.set("key", confItem);
                kv.set("value", this->app.config().getString(rootItem + "." + confItem, "none"));
                trList.push_back(kv);
            }
            table.set("caption", rootItem);
            table.set("trList", trList);
            tableList.push_back(table);
            configKeys.clear();
        }
        data->set("tableList", tableList);

        response.setContentType("text/html;charset=utf-8");
        response.setChunkedTransferEncoding(true);
        root_view::root_cache().add(cacheKey, this->render_tpl(data->get("maintpl")->stringValue(),
             *data));
        response.send() << *root_view::root_cache().get(cacheKey);
    }

}
```

```
std::string cacheKey(this->create_cache_key(request, response));
if (root_view::root_cache().has(cacheKey)) {
    response.send() << *root_view::root_cache().get(cacheKey);
    return;
}
```

源码7中的以上部分表示如果系统中存在缓存，则直接调用缓存响应。

```
Poco::SharedPtr<Kainjow::Mustache::Data> data = this->tpl_ready("/demo/config.json", "demo.GET");
```

这个部分表示根据模板配置文件/var/webcppd/tpl/demo/config.json（见源码9）部署模板以及其中的变量。所有变量被存储在变量 data 中。

源代码 9: config.json

```
{
    "demo": {
        "GET": {
            "title": "webcppd 配置信息",
            "maintpl": "/demo/info.html",
            "subtpl": [
                {
                    "name": "head",
                    "path": "/demo/head.html"
                },
                {
                    "name": "script",
                    "path": "/demo/script.html"
                }
            ]
        },
        "POST": {}
    }
}
```

tpl_ready 方法的第一个参数是模板配置文件路径，它只应该写模板目录/var/webcppd/tpl以下的部分，该方法会自动将路径扩展完善。第二个方法是访问相关部分的 json 查询路径。

```
Poco::Util::AbstractConfiguration::Keys rootKeys, configKeys;
this->app.config().keys(rootKeys);
Kainjow::Mustache::Data tableList(Kainjow::Mustache::Data::List());
for (auto& rootItem : rootKeys) {
    Kainjow::Mustache::Data table, trList(Kainjow::Mustache::Data::List());
    this->app.config().keys(rootItem, configKeys);
```

```
    for (auto& confItem : configKeys) {
        Kainjow::Mustache::Data kv;
        kv.set("key", confItem);
        kv.set("value", this->app.config().getString(rootItem + "." + confItem, "none"));
        trList.push_back(kv);
    }
    table.set("caption", rootItem);
    table.set("trList", trList);
    tableList.push_back(table);
    configKeys.clear();
}
data->set("tableList", tableList);
```

这个部分把系统变量及其值读出来，存储在 data 中。其中变量 app 是系统实例本身。代码中出现的 Kainjow::Mustache::Data 是 webcppd 准备的 Mustache 模板系统中的一个类。运用该类可以很轻松地实现视图的模板化。

关于 Mustache 模板系统的语法和用法，可参考一般性介绍[2]。该模板系统的 C++ 实现有不少，这里推荐两个：Mustache[3]和plustache[4]。前者只是一个头文件，更易于使用。

```
root_view::root_cache().add(cacheKey, this->render_tpl(data->get("maintpl")->stringValue(), *data));
response.send() << *root_view::root_cache().get(cacheKey);
```

最后一部分代码表示把模板渲染结果即最终视图存入缓存，方便下次直接调用，如代码5.1所示。

5.2 模板配置

模板配置主要分为三个部分。

- 主模板：maintpl
- 子模板：subtpl，这是一个对象数组，其中每一个对象配置一个子模板，要求每一对象必须有 name 和 path 两个属性。
- 变量：这些变量可以直接运用到模板中去。

具体用法可参考源代码9。

[2]https://mustache.github.io/mustache.5.html
[3]https://github.com/kainjow/Mustache
[4]https://github.com/mrtazz/plustache

5.3 路由设定

为了能够通过路径/info 访问 webcppd::info 类。首先需要把该类载入类装载器。参考示例4和4。

其次，则需要添加路由规则：

```
1 ^/info/?,webcppd::info
```

最后，make && sudo make install 即可。访问http://localhost/info 应该看到如图3效果。

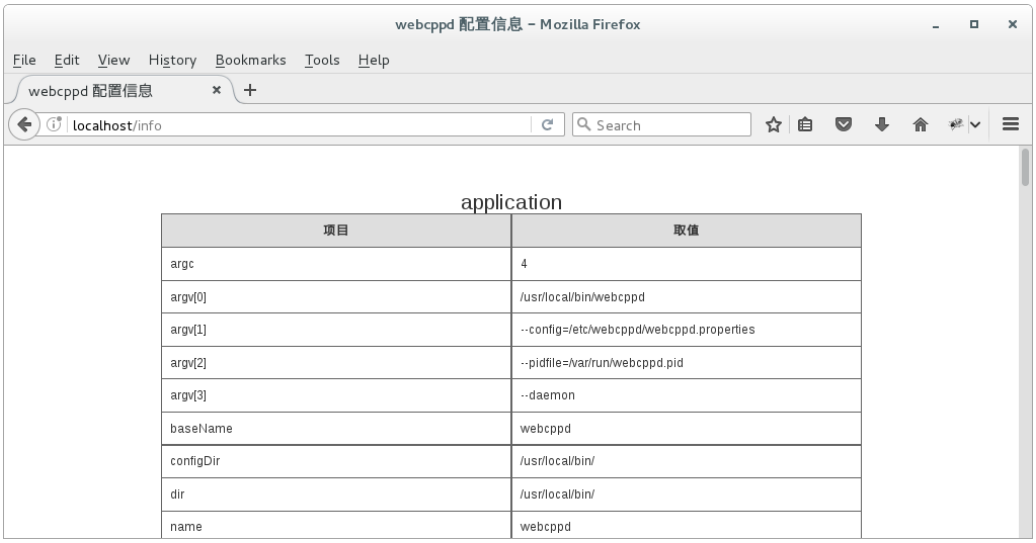

图 3: webcppd 系统信息

6 更多示例

星空博客是基于 webcppd 的 Markdown 博客程序。更多示例可在星空博客程序中找到。该程序运行在www.webcpp.net。访问该站可见效果。