

HCPP 说明

HCPP 是什么

如果你使用过 PHP 的话 HCPP 将会很好理解，下面是从 W3C 上面抄下来的一个简单的 PHP 示例。



The screenshot shows the W3School TIY (Try It Yourself) interface. It is divided into two main sections: '源代码:' (Source Code) and '运行结果:' (Run Result). The '源代码:' section contains the following PHP code:

```
<!DOCTYPE html>
<html>
<body>

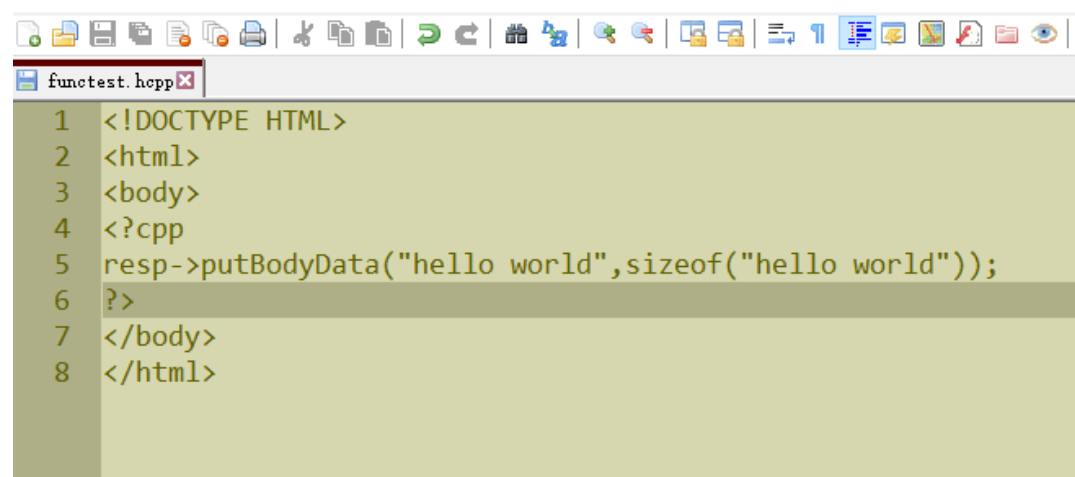
<?php
echo "我的第一段 PHP 脚本! ";
?>

</body>
</html>
```

A red arrow points to the PHP code block. The '运行结果:' section displays the output: '我的第一段 PHP 脚本!'.

图 1 PHP 脚本及其结果

HCPP 和 PHP 是一样的，就是把中间的 PHP 换成了 C++代码，标签由<?PHP...?>换成了<?cpp...?> 下面的代码运行后将在浏览器中看到"hello world"的页面。



The screenshot shows a web browser window with a single tab titled 'functest.hcpp'. The browser's address bar is empty. The main content area displays the following HCPP code:

```
1 <!DOCTYPE HTML>
2 <html>
3 <body>
4 <?cpp
5 resp->putBodyData("hello world",sizeof("hello world"));
6 ?>
7 </body>
8 </html>
```

图 2 HCPP 的"hello word"

HCPP 是怎样工作的

网络框架在收到一个 HTTP 请求后会根据请求的 URL 调用对应的服务函数(将该函数记为 ServerFunction)，每一个 ServerFunction 都有如下所示的公共部分。函数名不一定叫 ServerFunction，可自定义但不能重名。

```
1 extern "C"
2 bool ServerFunction(HttpSession* session)
3 {
4     HttpRequest* req = session->getHttpRequest();
5     HttpResponse* resp = session->getHttpResponse();
6     /*-----*/
7     //service code
8     /*-----*/
9     session->startResponse();
10    return true;
11 }
12
```

图 3 服务函数公共部分

如果在函数体内写入如下代码，浏览器发起请求后网络框架会调用该函数，在该函数被调用后将在浏览器中看到"hello world"的页面。(url 如何映射到该函数参见后面章节)

```
1 extern "C"
2 bool functest(HttpSession* session)
3 {
4     HttpRequest* req = session->getHttpRequest();
5     HttpResponse* resp = session->getHttpResponse();
6     /*-----*/
7     resp->putBodyData("<!DOCTYPE HTML><html><body>", sizeof("<!DOCT....."));
8     resp->putBodyData("hello word", sizeof("helloworld"));
9     resp->putBodyData("</body></html>", sizeof("</body></html>"));
10    /*-----*/
11    session->startResponse();
12    return true;
13 }
14
```

图 4 原始 c++ 实现的 http 响应服务

有了上述的函数模板后，只要将如图 2 所示的 hcpp 文件转化为类似图 4 所示的函数即可（具体过程见下图）。图 2 的 functest.hcpp 文件被专门的工具处理后将会生成一个名为 functest.cpp 的 c++ 文件，该文件内包含有一个名为 functest 的函数。最后 functest.cpp 会与其他通过 hcpp 生成的 cpp 文件被 g++ 编译器编译，生成一个包含 functest 函数和其他函数的动态链接库；网络框架在收到请求后会根据一定的规则调用动态链接库中的 functest 函数，完成一次 HTTP 响应。

hcpp 文件被转化为对应 ServerFunction 过程图示如下



图 5 hcpp 文件

上面的 hcpp 文件会被解析成三段，分别是 html 内容、c++代码、html 内容。让后会输出如下的 c++文件。



图 6 最终生成的 c++文件

最终生成的 functest 函数并不完全像图 4 所示的那样，因为 hcpp 被解析后不仅生成 cpp 文件，还生成了一个只保存 html 内容的 hinfo 文件；对于 functest 来说，hinfo 文件专门用一个对象来表示，该对象里面存储的就是 html_str1 和 html_str2。

HCPP 如何将 URL 映射到对应的服务函数

网络框架启动时会读取一个 URL 与函数的映射文件，该文件内容如下所示。文件中的每一行表示一个映射，格式为 `URL LIB:ServerFunc1 LIB:ServerFunc2 xxx:xxx...`。下面文件第一行表示当 url 为 `/test` 时调用一个名叫 `out` 的动态连接库中的 `functest` 函数，该 `functest` 函数就是上文 `functest.hcpp` 最终生成的函数。每个 url 最多可以对应 8 个服务函数，这些服务函数会被依次调用；如果这些函数调用过程中有一个函数返回 `true` 则后面的函数将不会被调用。下面的示例中为每一个 url 只安排了一个服务函数，所以这些函数应该返回 `true`

```
1  /test out:functest
2  /index out:vediolist
3  / out:vediolist
```

图 7 URL 映射文件

上述映射规则是在网络框架启动时建立的，这个时候会根据 url 映射文件加载动态链接库并导入其中对应的函数，因此就暂时的设计来说每添加一个新的服务需要重启网络服务。另外，在网络框架启动的时候还会读取一个配置文件，该文件记录了服务的根目录，就像 `apache` 设置网站根目录一样；上述的动态连接库就是通过该根目录来查找。

HCPP 如何使用（Linux）

1. 建立一个叫 `MyServer` 的目录
2. 将 `HttpServer` 和 `HttpTest` 拷贝到 `MyServer` 目录中
3. `MyServer` 下建立一个 `MyTest` 目录
4. 将 `HttpTest` 中的 `commonfile` 和 `Makefile` 拷贝到 `MyTest` 目录下
5. 在 `MyTest` 目录下建立一个 `testfunc.hcpp` 文件，参考 `HttpTest` 下 `functest.hcpp`
6. 在 `MyTest` 目录下使用 `make` 命令，如果没错的话会生成一个包含 `testfunc` 函数的 `out.so`
7. 进入 `HttpServer` 中将 `path.ini` 修改为 `MyTest` 所在目录，例如 `/home/.../MyTest/`
8. 进入 `HttpServer` 中将 `route.ini` 添加一行内容：`test1 out:testfunc`
9. 启动 `HttpServer` 目录下 `a.out` 后在浏览器访问 `127.0.0.1:4567/test1` 访问页面

HCPP 现状声明

HCPP 现在相当于是技术验证版本，因此功能简单且绝对不稳定，所以说它并不是一个解决方案，而是一个 idea 的参考或者说是一个玩具。如果需要使用它的话可以参考它的实现重新写一个。最后，由于此版本只是作为技术验证，所以代码很烂且没有注释！

HCPP 如何设计的

现目前 HCPP 的设计分为两个部分。第一个部分为网络框架，另一个部分为应用部件。应用部件的任务就是解析 hcpp 文件并最终生成动态连接库。

网络框架设计

- 网络框架分为三部分，分别是网络传输模块，协议模块，路由模块。

网络传输模块负责接受和发送一个协议数据包。该模块接收到一个数据包后会将该数据包提交到协议模块，协议模块对提交上来的数据解析，形成对应的协议对象。网络传输模块在提交数据后可以询问协议模块数据是否接受完毕，如果数据接受完毕网络传输模块将会停止接受数据并认为客户端请求发送完毕，然后调用 `recvAClientReq` 函数（虚函数）处理此次请求。本模块对应 `HttpServer` 目录下 `net` 目录。

协议模块负责解析请求数据与响应数据的封包。该模块解析请求数据并告知网络传输模块请求数据是否发送完毕。同时该模块也负责响应数据的管理，在服务程序决定发起响应后将协议数据封包并于网络传输模块合作将响应数据发送出去。本模块对应 `HttpServer` 目录下 `session` 目录。

路由模块负责将请求转发到对应的服务函数。该模块将请求过来的 url 作为查找对应服务的 key，在获取 url 对应的服务表后依次调用服务表中的服务函数。本模块对应 `HttpServer` 目录下 `route` 目录。

- 网络框架的组装

HCPP 服务程序的其实只是一个简单的测试程序，感兴趣的人可以参考 `HttpServer` 目录下 `main.cpp` 文件的代码来实现自己的网络服务程序。`main.cpp` 中将网络框架的三个模块组装到一起，最终实现了一个简单的网络服务程序。

应用部件设计

应用部件的任务为将 hcpp 解析成 cpp 文件，然后生成最终的动态连接库。一个 hcpp 文件最终会被生成 hinfo 和 cpp 共两个文件，其中 hinfo 文件中保存的是 hcpp 中的 html 内容，而 cpp 文件中则是 hcpp 中的代码部分。本模块对应 HttpServer 目录下的 hcpp 目录。生成 cpp 文件时需要 HttpTest 下的 commonfile 中的 cpp.template 文件作为模板，三行数字分别为头文件插入的位置、函数名插入的位置、代码插入的位置。生成的 hinfo 第一行数字表明共有 m 段 html 内容，接着 m 行为每段 html 内容的大小，后面紧跟着 html 内容。

目录及文件说明

HttpServer 目录：

app 目录： 无用，请忽略

hcpp 目录： 将 hcpp 文件生成 cpp 文件的实现代码以及 cpp 文件需要的代码

filemanager.hpp: 生成的 cpp 文件需要使用到的代码，用于模块管理文件

hcpp.hpp: 解析 hcpp 文件并生成 hinfo 和 cpp 文件

hcppfile.hpp: 生成的 cpp 文件需要使用到的代码，表示一个 hinfo 文件

main.cpp: 调用 hcpp.hpp 来解析 hcpp 文件及生成 hinfo 和 cpp 文件

net 目录： 负责请求数据的接受与发送

net.hpp: 底层的数据收发

session.hpp: 抽象一个协议或者会话

session 目录： http 协议与会话的实现

httpsession.hpp: 管理一次 http 会话

httprequest.hpp: http 请求的解析

httpresponse.hpp: http 响应数据的管理

include.hpp: 外部使用时直接包含该头文件

main.cpp: 网络服务程序的实现

path.ini: 服务根目录配置文件

route.ini: 路由信息文件

Makefile: 用于生成服务程序

HttpTest 目录:

commonfile 目录: 实现服务功能必须包含的文件夹

cpp.template: 用于生成 cpp 文件的模板

module.cpp: 动态连接库公有的函数实现

hcpp: hcpp 解析并生成 hinfo 和 cpp 文件夹的工具

*.hcpp: 一个具体的服务功能示例

*.hinfo: 生成的 html 内容文件

*.cpp: 生成的代码文件

Makefile: 记录了由 hcpp 文件生成动态链接库的过程