# Goahead的认识



Goahead有不少介绍，其官网是

<http://embedthis.com/products/goahead/>

另一个是AppWeb，以及BOA等，不过从KSFT\_API来讲，我们对这类技术组件的技术要求是：

少量的内存消耗；

代码跨平台；

C/C++的编程接口；

可支持安全通信SSL；

代码规模小；

不要求高并发性能；

从目前看，GoAhead更适合些。

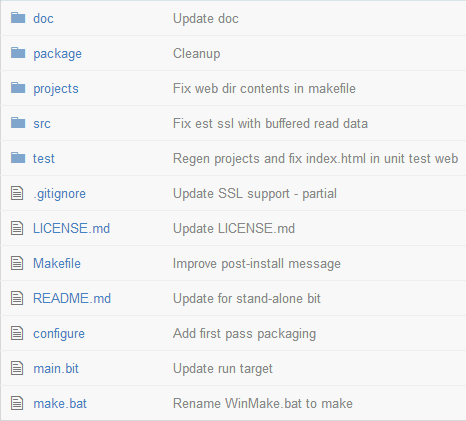


以上是我考虑的使用情况，这里嵌入式WebServer就是该类技术组件。

# Goahead的windows平台构建使用

1 下载源代码包到本地（也可以通过github， <https://github.com/embedthis/goahead>），如

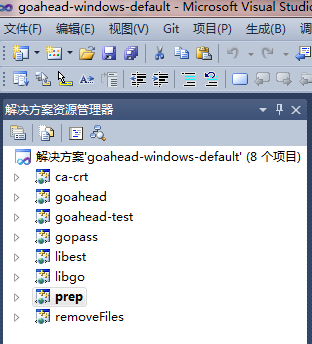
C:\github\goahead，不要加压在待中文的复杂目录下，可能存在批处理失败的可能。



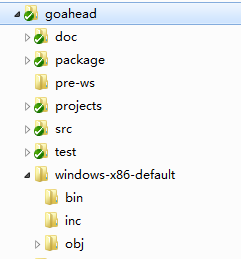
2 编译工程选择vs2010，解决方案文件是

C:\github\goahead\projects\goahead-windows-default.sln

打开后，



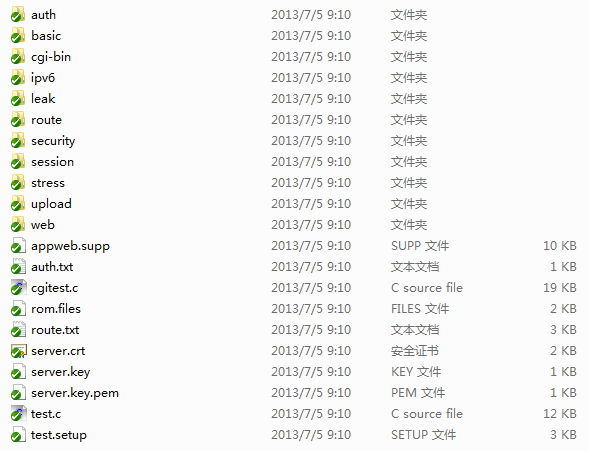
3 工程基本没问题，直接生成解决方案即可。会生成输出windows-x86-default目录，

 其中bin/下文件有，注意其中的goaheah-test.exe



4 运行测试goahead-test

测试项目goahead-test是一个很典型的测试工程，其提供默认或指定端口的web server，其web样例应用的目录是C:\github\goahead\test，



注意其中关键文件是

web应用的根下的route.txt，auth.txt等；

在根下的web/为默认的web page目录（我猜测）。

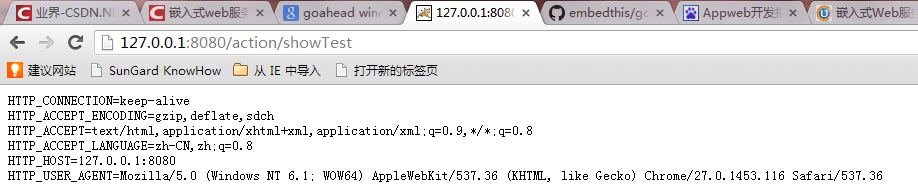
通过IDE方式

指定工程goahead-test，重新设置工作目录为”..\..\test”，这样就设置C:\github\goahead\test为当前程序的执行目录，与内部代码的默认匹配，就需要指定home了。

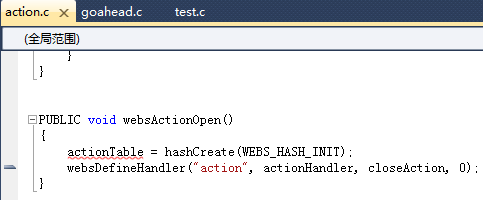


编译执行即可，console里没有任何输出。

这时可以在浏览器输入http://127.0.0.1:8080/action/showTest



注意：URL里的/action/，在代码里有反映



通过命令行方式，这是需要指定HOME和web目录，还能指定端口，如下

C:\github\goahead\windows-x86-default\bin>goahead-test.exe --home C:\github\goah

ead\test web 127.0.0.1:7889



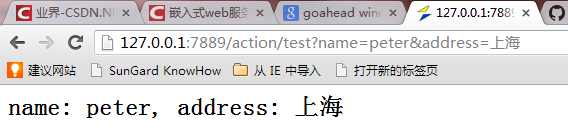
这里重新指定了端口，改为7889，浏览器效果如下



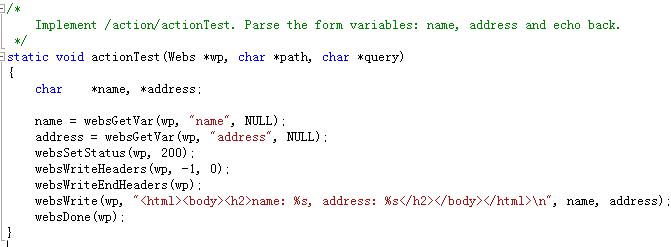
再测试一个，输入URL

http://127.0.0.1:7889/action/test?name=peter&address=上海

注意如果出现乱码，就是字符集的问题，修改浏览器编码为”UTF-8”，效果如下。



我们看一下其后台的代码



可见其支持URL的格式规范还是不错的，两个参数被正确的获取并进入页面显示出来。

这些功能都是在嵌入式web server中十分重要的。

# Goahead的linux平台构建使用

1. 安装构建bit

<http://embedthis.com/products/bit/index.html>

从该处下载bit源代码包，下载到linux，用root登录linux，依次执行

tar xzvf bit-0.8.7-0-src.gz

cd bit-0.8.7-0

./configure

make boot

make install

安装完毕

1. 构建goahead

从网站下载源代码包，解压

tar xzvf goahead-3.1.1-0-src.gz

cd goahead-3.1.1-0

./configure --prefix=XXXX

其中XXXX为新创建的目标目录，然后

bit

bit install

完成后，目标目录下大致为

[node@irdgc3 goahead]$ du .

12 ./var/www/goahead-default

16 ./var/www

20 ./var

4 ./usr/local/share/man/man1

8 ./usr/local/share/man

12 ./usr/local/share

16 ./usr/local/lib/goahead/3.1.1/doc/man/man1

20 ./usr/local/lib/goahead/3.1.1/doc/man

24 ./usr/local/lib/goahead/3.1.1/doc

900 ./usr/local/lib/goahead/3.1.1/bin

932 ./usr/local/lib/goahead/3.1.1

936 ./usr/local/lib/goahead

940 ./usr/local/lib

4 ./usr/local/bin

960 ./usr/local

964 ./usr

20 ./etc/goahead

24 ./etc

1012 .

目标二进制文件，包括动态库在

[node@irdgc3 goahead]$ ll ./usr/local/lib/goahead/3.1.1/bin

总用量 896

-rw-r--r--. 1 node nodejs 211602 7月 5 17:16 ca.crt

-rwxr-xr-x. 1 node nodejs 19214 7月 5 17:16 goahead

-rwxr-xr-x. 1 node nodejs 305103 7月 5 17:16 libest.so

-rwxr-xr-x. 1 node nodejs 374233 7月 5 17:16 libgo.so

目标配置文件在

[node@irdgc3 goahead]$ ll ./etc/goahead

总用量 16

-rw-r--r--. 1 node nodejs 444 7月 5 17:16 auth.txt

-rw-r--r--. 1 node nodejs 1903 7月 5 17:16 route.txt

-rw-r--r--. 1 node nodejs 1679 7月 5 17:16 self.crt

-rw-r--r--. 1 node nodejs 1679 7月 5 17:16 self.key

1. 测试验证

cd linux-x64-debug

cd bin

./goahead-test --home /home/node/goahead-3.1.1-0/test/ web 127.0.0.1:7889

这里/home/node/goahead-3.1.1-0/test/ 为源代码包中的test项目根目录

执行时如出现涉及当前主机名的错误信息，需要观察/etc/hosts下，有无主机名配置，注意其中的红色，如果没有则需要添加（用root用户）。

[root@irdgc3 etc]# more hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

10.253.46.217 irdgc3

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

运行成功后，测试效果如下：



# Goahead的代码问题临时处理

#ifdef \_\_cplusplus

extern "C" {

#endif

#include "goahead.h"

#include "js.h"

#ifdef \_\_cplusplus

}

#endif

一般对goahead的动态库使用方式，在C++工程里需要这样引用，但可能goahead.h编译时存在错误，如下

1> KS\_Spy.cpp

1>c:\svn\peripheral\_products\第三方接口\新接口\extsdk\ksft\_api4spy2\dev2\ks\_spy\ks\_spy.cpp(18): error C2059: 语法错误:“}”

1>c:\svn\peripheral\_products\第三方接口\新接口\extsdk\ksft\_api4spy2\dev2\ks\_spy\ks\_spy.cpp(18): error C2143: 语法错误 : 缺少“;”(在“}”的前面)

1>c:\svn\peripheral\_products\第三方接口\新接口\extsdk\ksft\_api4spy2\dev2\ks\_spy\ks\_spy.cpp(18): error C2059: 语法错误:“}”

临时修改为将goahead.h的代码段

#ifdef \_\_cplusplus

}

#endif

#endif /\* \_h\_GOAHEAD \*/

改为

#ifdef \_\_cplusplus

//}

#endif

#endif /\* \_h\_GOAHEAD \*/

这样才能通过，为何原代码有问题还不清楚。这是在3.1.3版本使用时发现的。

# Linux版本中其他编译说明

**Boost**

1. 解压后，先执行bootstrap.sh；
2. 执行 ./b2 ;
3. 编译结果在stage/ 下；

**Jsoncpp**

安装JsonCpp需要下载JsonCpp和Scon还需要机器上有python

下载JsonCpp

下载scons-2.1.0

scons又是一个牛叉的工具，功能和GNU make一样，又比make简单多了。scons是python工具，需要先安装好python。

下载scons 在这里http://www.scons.org/

解压 tar -zxvf scons-2.0.1.tar.gz

设定环境变量

# export MYSCONS=解压的路径

# export SCONS\_LIB\_DIR=$MYSCONS/engine

开始编译jsoncpp

我的jsoncpp目录在/root/software/jsoncpp-src-0.5.0/

# cd jsoncpp-src-0.5.0

# python $MYSCONS/script/scons platform=linux-gcc

【注释】：网上有一些错误的 也可能是早期的版本 # python $MYSCONS/script/scons.py platform=linux-gcc

经过测试 发现报错 查看路径下 没有scons.py文件 故改为scons

之后编译

生成静态和动态库文件

在jsoncpp-src-0.5.0/libs/linux-gcc-4.1.2目录下

libjson\_linux-gcc-4.4.4\_libmt.a

libjson\_linux-gcc-4.4.4\_libmt.so