
Web Application Library

WebAppLib是一系列主要用于类Unix操作系统环境下WEB开发的C++类库。设计目的是通过提供使用简单方便、相对独立的C++类和函数来简化CGI程序开发过程中的常见操作，提高开发效率，降低系统维护与改进的难度，适用于中等以上规模WEB系统开发

WebAppLib所有的类、函数、变量都声明于webapp命名空间内，由以下部分组成：

String : 继承并兼容与std::string的字符串类，增加了开发中常用的字符串处理函数；

Cgi : 支持文件上传的CGI参数读取类；

Cookie : HTTP Cookie设置与读取类；

MysqlClient : MySQL数据库连接类，MySQL连接处理C函数接口的C++封装；

MysqlData : MySQL查询结果数据集类，MySQL查询结果数据提取C函数接口的C++封装；

Template : 支持在模板中嵌入条件跳转、循环输出脚本的 HTML 模板类；

HttpClient : HTTP/1.1通信协议客户端类；

DateTime : 日期时间运算、格式化输出类；

TextFile : 固定分隔符文本文件读取解析类；

ConfigFile : INI格式配置文件解析类；

FileSystem : 文件系统操作函数库；

Encode : 字符串编码解码函数库；

Utility : 系统调用与工具函数库

类库详细使用说明可参见类库参考手册 [help.chm](#)

编译本类库要求使用g++编译器，版本不低于v3.4.0，目前支持的操作系统有Linux(CentOS v4.0以上版本)，Solaris(v10以上版本)，还可以通过Cygwin环境运行于Windows操作系统

[Readme](#)

pilot.cn@gmail.com

模块索引

模块

这里列出了所有模块:

waCgi相关数据类型与全局函数.....	3
waDateTime相关数据类型与全局函数.....	5
waEncode字符串加密编码函数库	10
waFileSystem文件操作函数库	12

waHttpClient相关全局函数	21
waMySQLClient相关数据类型与全局函数.....	23
waString相关全局函数.....	24
waUtility系统调用工具函数库	27

命名空间索引

命名空间列表

这里列出了所有文档化的命名空间定义,附带简要说明:

webapp (Web Application Library namespace)	30
--	----

类索引

类列表

这里列出了所有类、结构、联合以及接口定义等,并附带简要说明:

webapp::Cgi (CGI参数读取类)	35
webapp::ConfigFile (INI格式配置文件解析类)	37
webapp::Cookie (Cookie读取,设置类)	43
webapp::DateTime (DateTime日期时间运算类)	45
webapp::HttpClient (HTTP客户端类 使用说明文档及简单范例)	55
webapp::MySQLClient (MySQL数据库连接类)	62
webapp::MySQLData (MySQL数据集类)	68
webapp::String (继承自string的字符串类 基类string使用说明文档)	73
webapp::Template (支持条件、循环脚本的HTML模板处理类 使用说明文档及简单范例) ..	84
webapp::TextFile (固定分隔符文本文件读取解析类)	89

文件索引

文件列表

这里列出了所有文档化的文件,并附带简要说明:

example.cpp (代码示例文件,演示一个简单CGI流程)	92
Makefile (Web Application Library Makefile)	93
Makefile.example (WebAppLib example Makefile)	93
waCgi.cpp (Cgi,Cookie类实现文件)	93
waCgi.h (Webapp::Cgi,webapp::Cookie类头文件 依赖于 webapp::String , webapp::Encode) ..	94

waConfigFile.cpp (INI格式配置文件解析类实现文件)	95
waConfigFile.h (INI格式配置文件解析类头文件 依赖于 webapp::String , webapp::TextFile)	96
waDateTime.cpp (Webapp::DateTime类实现文件)	97
waDateTime.h (Webapp::DateTime类头文件 日期时间运算)	98
waEncode.cpp (字符串BASE64、URI、MD5编码函数实现文件)	100
waEncode.h (编码,加解密函数头文件 字符串BASE64、URI、MD5编码函数)	101
waFileSystem.cpp (文件操作函数实现文件)	102
waFileSystem.h (文件操作函数头文件 常用文件操作)	104
waHttpClient.cpp (HTTP客户端类实现文件)	106
waHttpClient.h (HTTP客户端类头文件 依赖于 webapp::String , webapp::Encode 使用说明文档及简单范例)	106
waMysqlClient.cpp (Webapp::MysqlClient,webapp::MysqlData类实现文件)	108
waMysqlClient.h (Webapp::MysqlClient,webapp::MysqlData类头文件 MySQL数据库C++接口)	108
waString.cpp (Webapp::String类实现文件)	110
waString.h (Webapp::String类头文件 继承自string的字符串类 基类string使用说明文档)	111
waTemplate.cpp (HTML模板处理类实现文件)	112
waTemplate.h (HTML模板处理类头文件 支持条件、循环脚本的HTML模板处理类 依赖于 waString 使用说明文档及简单范例)	113
waTextFile.cpp (固定分隔符文本文件读取解析类实现文件 读取解析固定分隔符文本文件)	114
waTextFile.h (固定分隔符文本文件读取解析类头文件 读取解析固定分隔符文本文件 依赖于 webapp::String)	114
waUtility.cpp (系统调用工具函数实现文件)	115
waUtility.h (系统调用工具函数头文件 常用系统调用和工具函数 依赖于 webapp::String , webapp::DateTime)	117
webapplib.h (开发库头文件集合)	118

模块说明

waCgi相关数据类型与全局函数

类型定义

- typedef map< string, string > [webapp::CgiList](#)
[Cgi](#) 参数值列表类型 (map<string,string>)
- typedef map< string, string > [webapp::CookieList](#)
[Cookie](#) 参数值列表类型 (map<string,string>)

函数

- void [webapp::http_head](#) ()
输出HTML Content-Type header

- string [webapp::get_env](#) (const string &envname)
取得环境变量

详细描述

类型定义说明

[webapp::CgiList](#)

[Cgi](#) 参数值列表类型 (map<string,string>)

[webapp::CookieList](#)

[Cookie](#) 参数值列表类型 (map<string,string>)

函数说明

void webapp::http_head ()

输出HTML Content-Type header

输出HTML Content-Type header,自动避免重复输出

string webapp::get_env (const string & envname)

取得环境变量

参数:

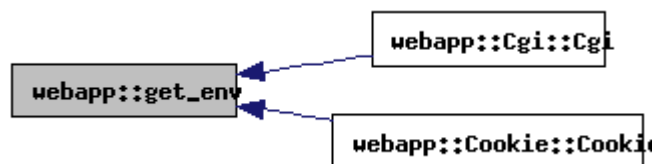
<i>envname</i>	环境变量名
----------------	-------

返回:

成功返回环境变量值,否则返回空字符串

参考自 [webapp::Cgi::Cgi\(\)](#) , 以及 [webapp::Cookie::Cookie\(\)](#).

这是这个函数的调用关系图:



waDateTime相关数据类型与全局函数

宏定义

- #define [TIME_ONE_SEC](#) 1
时长定义一秒钟
- #define [TIME_ONE_MIN](#) 60
时长定义一分钟
- #define [TIME_ONE_HOUR](#) 3600
时长定义一小时
- #define [TIME_ONE_DAY](#) 86400
时长定义一天
- #define [TIME_ONE_WEEK](#) 604800
时长定义一周

函数

- DateTime [webapp::operator+](#) (const DateTime &date1, const DateTime &date2)
相加操作
- DateTime [webapp::operator+](#) (const DateTime &date, const time_t &tt)
相加操作
- DateTime [webapp::operator-](#) (const DateTime &date1, const DateTime &date2)
相减操作
- DateTime [webapp::operator-](#) (const DateTime &date, const time_t &tt)
相减操作
- bool [webapp::operator==](#) (const DateTime &left, const DateTime &right)
时间相等比较
- bool [webapp::operator==](#) (const DateTime &left, const time_t &right)
时间相等比较
- bool [webapp::operator!=](#) (const DateTime &left, const DateTime &right)
时间不相等比较
- bool [webapp::operator!=](#) (const DateTime &left, const time_t &right)
时间不相等比较
- bool [webapp::operator>](#) (const DateTime &left, const DateTime &right)
时间大于比较
- bool [webapp::operator>](#) (const DateTime &left, const time_t &right)
时间大于比较
- bool [webapp::operator<](#) (const DateTime &left, const DateTime &right)
时间小于比较
- bool [webapp::operator<](#) (const DateTime &left, const time_t &right)
时间小于比较
- bool [webapp::operator>=](#) (const DateTime &left, const DateTime &right)
时间不小于比较
- bool [webapp::operator>=](#) (const DateTime &left, const time_t &right)

时间不小于比较

- bool [webapp::operator<=](#) (const DateTime &left, const DateTime &right)
时间不大于比较
- bool [webapp::operator<=](#) (const DateTime &left, const time_t &right)
时间不大于比较

详细描述

宏定义说明

#define TIME_ONE_SEC 1

时长定义一秒钟

#define TIME_ONE_MIN 60

时长定义一分钟

参考自 webapp::DateTime::mins().

#define TIME_ONE_HOUR 3600

时长定义一小时

参考自 webapp::DateTime::hours().

#define TIME_ONE_DAY 86400

时长定义一天

参考自 webapp::DateTime::days().

#define TIME_ONE_WEEK 604800

时长定义一周

参考自 webapp::DateTime::weeks().

函数说明

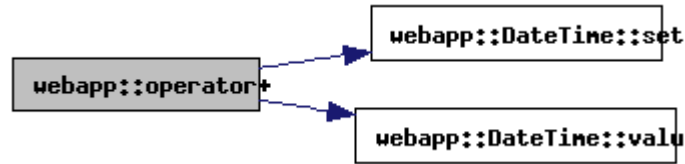
DateTime webapp::operator+ (const DateTime & date1, const DateTime & date2)

相加操作

时间相加

参考 `webapp::DateTime::set()` , 以及 `webapp::DateTime::value()`.

函数调用图:



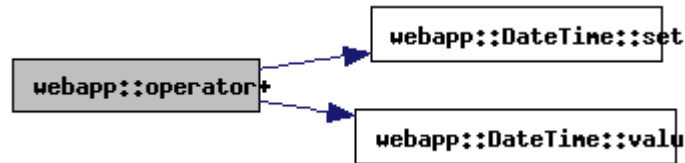
`DateTime webapp::operator+ (const DateTime & date, const time_t & tt)`

相加操作

时间相加

参考 `webapp::DateTime::set()` , 以及 `webapp::DateTime::value()`.

函数调用图:



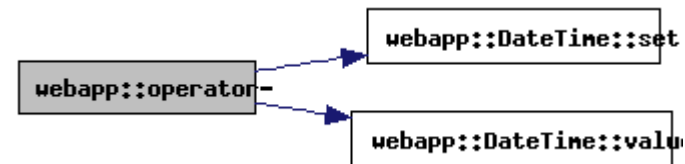
`DateTime webapp::operator- (const DateTime & date1, const DateTime & date2)`

相减操作

时间相减

参考 `webapp::DateTime::set()` , 以及 `webapp::DateTime::value()`.

函数调用图:



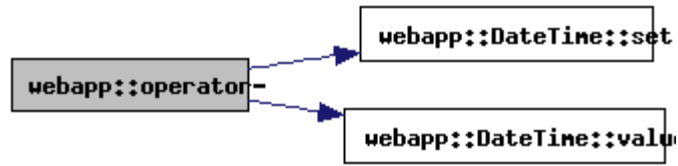
`DateTime webapp::operator- (const DateTime & date, const time_t & tt)`

相减操作

时间相减

参考 `webapp::DateTime::set()` , 以及 `webapp::DateTime::value()`.

函数调用图:



bool webapp::operator== (const DateTime & left, const DateTime & right) [inline]

时间相等比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator== (const DateTime & left, const time_t & right) [inline]

时间相等比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator!= (const DateTime & left, const DateTime & right) [inline]

时间不相等比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator!= (const DateTime & left, const time_t & right) [inline]

时间不相等比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator> (const DateTime & left, const DateTime & right) [inline]

时间大于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator> (const DateTime & left, const time_t & right) [inline]

时间大于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator< (const DateTime & left, const DateTime & right) [inline]

时间小于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator< (const DateTime & left, const time_t & right) [inline]

时间小于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator>= (const DateTime & left, const DateTime & right) [inline]

时间不小于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator>= (const DateTime & left, const time_t & right) [inline]

时间不小于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator<= (const DateTime & left, const DateTime & right) [inline]

时间不大于比较

参考 webapp::DateTime::value().

函数调用图:



bool webapp::operator<= (const DateTime & left, const time_t & right) [inline]

时间不大于比较

参考 webapp::DateTime::value().

函数调用图:



waEncode字符串加密编码函数库

函数

- string [webapp::uri_encode](#) (const string &source)
URI编码
- string [webapp::uri_decode](#) (const string &source)
URI解码
- string [webapp::base64_encode](#) (const string &source)
字符串MIME BASE64编码
- string [webapp::base64_decode](#) (const string &source)
字符串MIME BASE64解码
- string [webapp::md5_encode](#) (const string &source)

详细描述

函数说明

string webapp::uri_encode (const string & source)

URI 编码

参数:

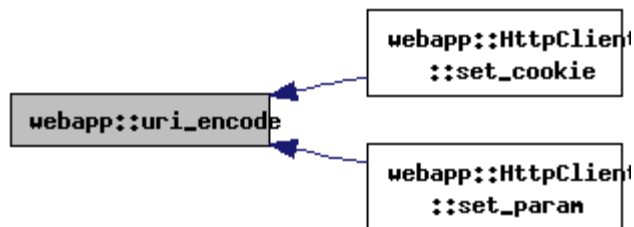
<i>source</i>	原字符串
---------------	------

返回:

编码结果字符串

参考自 webapp::HttpClient::set_cookie(), 以及 webapp::HttpClient::set_param().

这是这个函数的调用关系图:



string webapp::uri_decode (const string & source)

URI 解码

参数:

<i>source</i>	URI 编码字符串
---------------	-----------

返回:

解码结果

string webapp::base64_encode (const string & source)

字符串 MIME BASE64 编码

MIME Base64 编码

参数:

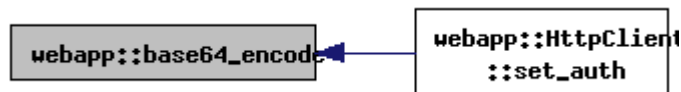
<code>source</code>	原字符串
---------------------	------

返回:

成功返回编码结果,否则返回空字符串

参考自 `webapp::HttpClient::set_auth()`.

这是这个函数的调用关系图:



string webapp::base64_decode (const string & source)

字符串MIME BASE64解码

MIME Base64解码

参数:

<code>source</code>	BASE64编码字符串
---------------------	-------------

返回:

成功返回解码结果,否则返回空字符串

string webapp::md5_encode (const string & source)

MD5编码

MD5解码

参数:

<code>source</code>	MD5编码字符串
---------------------	----------

返回:

解码结果

waFileSystem文件操作函数库

函数

- bool [webapp::file_exist](#) (const string &file)
文件或者目录是否存在
- bool [webapp::is_link](#) (const string &file)
文件是否为链接
- bool [webapp::is_dir](#) (const string &file)
是否为目录
- bool [webapp::make_link](#) (const string &srcfile, const string &destfile)
建立链接

- long [webapp::file_size](#) (const string &file)
取得文件大小
- long [webapp::file_time](#) (const string &file)
取得文件更改时间
- string [webapp::file_path](#) (const string &file)
取得文件路径
- string [webapp::file_name](#) (const string &file)
取得文件名称
- bool [webapp::rename_file](#) (const string &oldname, const string &newname)
文件或者目录改名
- bool [webapp::copy_file](#) (const string &srcfile, const string &destfile)
拷贝文件
- bool [webapp::delete_file](#) (const string &file)
删除文件
- bool [webapp::move_file](#) (const string &srcfile, const string &destfile)
移动文件
- vector< string > [webapp::dir_files](#) (const string &dir)
返回目录文件列表
- bool [webapp::copy_dir](#) (const string &srcdir, const string &destdir)
拷贝目录
- bool [webapp::delete_dir](#) (const string &dir)
删除目录
- bool [webapp::move_dir](#) (const string &srcdir, const string &destdir)
移动目录
- void [webapp::lock_file](#) (int fd, const int type)
文件句柄锁函数
- bool [webapp::is_locked](#) (int fd)
判断文件句柄锁
- FILE * [webapp::lock_open](#) (const string &file, const char *mode, const int type)
申请锁并打开文件

详细描述

函数说明

bool webapp::file_exist (const string & file)

文件或者目录是否存在

参数:

<i>file</i>	文件路径名
-------------	-------

返回值:

<i>true</i>	文件存在
<i>false</i>	不存在

参考自 `webapp::copy_dir()` , 以及 `webapp::make_dir()`.

这是这个函数的调用关系图:



`bool webapp::is_link (const string & file)`

文件是否为链接

参数:

<i>file</i>	文件路径名
-------------	-------

返回值:

<i>true</i>	文件存在且为符号链接
<i>false</i>	不存在或者不是符号链接

`bool webapp::is_dir (const string & file)`

是否为目录

参数:

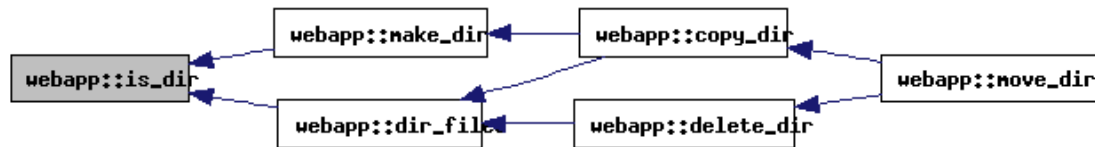
<i>dile</i>	目录路径名
-------------	-------

返回值:

<i>true</i>	目录存在
<i>false</i>	不存在或者不是目录

参考自 `webapp::dir_files()` , 以及 `webapp::make_dir()`.

这是这个函数的调用关系图:



`bool webapp::make_link (const string & srcfile, const string & destfile)`

建立链接

建立链接,新链接名文件必须不存在

参数:

<i>srcfile</i>	原文件名
<i>destfile</i>	新链接名

返回值:

<i>true</i>	操作成功
<i>false</i>	不成功

long webapp::file_size (const string & file)

取得文件大小

参数:

<i>file</i>	文件路径名
-------------	-------

返回:

若文件存在则返回大小,否则返回-1

long webapp::file_time (const string & file)

取得文件更改时间

参数:

<i>file</i>	文件路径名
-------------	-------

返回:

若文件存在则返回其最后更改时间,否则返回-1

string webapp::file_path (const string & file)

取得文件路径

参数:

<i>file</i>	文件路径名
-------------	-------

返回:

若能取得文件路径则返回,否则返回空字符串

string webapp::file_name (const string & file)

取得文件名称

参数:

<i>file</i>	文件路径名
-------------	-------

返回:

若能取得文件名称则返回,否则返回原文件路径名称

bool webapp::rename_file (const string & oldname, const string & newname)

文件或者目录改名

文件或者目录改名,新文件名必须与原文件名位于同一文件系统

参数:

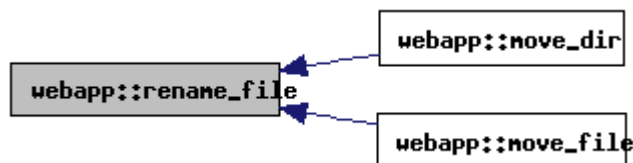
<i>oldname</i>	原文件名
<i>newname</i>	新文件名

返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考自 webapp::move_dir(), 以及 webapp::move_file().

这是这个函数的调用关系图:



bool webapp::copy_file (const string & srcfile, const string & destfile)

拷贝文件

参数:

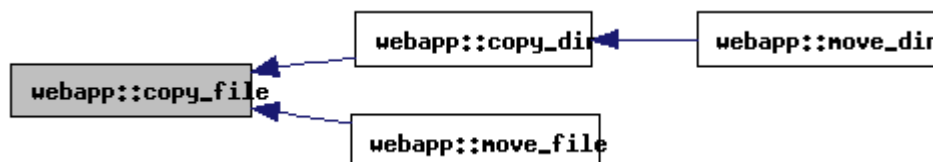
<i>srcfile</i>	原文件名
<i>destfile</i>	目的文件名,文件属性为0666

返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考自 webapp::copy_dir(), 以及 webapp::move_file().

这是这个函数的调用关系图:



bool webapp::delete_file (const string & file)

删除文件

参数:

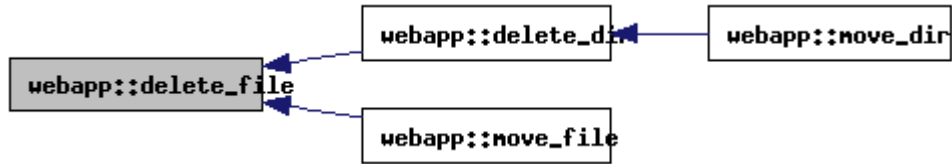
<i>file</i>	文件路径名
-------------	-------

返回值:

<i>true</i>	删除成功
<i>false</i>	文件不存在或者删除失败

参考自 `webapp::delete_dir()`, 以及 `webapp::move_file()`.

这是这个函数的调用关系图:



`bool webapp::move_file (const string & srcfile, const string & destfile)`

移动文件

参数:

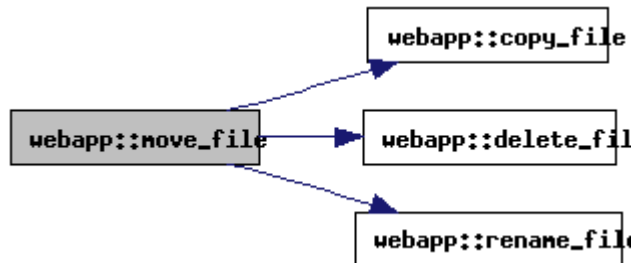
<i>srcfile</i>	原文件名
<i>destfile</i>	新文件名

返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考 `webapp::copy_file()`, `webapp::delete_file()`, 以及 `webapp::rename_file()`.

函数调用图:



`vector< string > webapp::dir_files (const string & dir)`

返回目录文件列表

参数:

<i>dir</i>	参数为目录路径名
------------	----------

返回:

返回结果为文件及子目录列表,子目录的第一个字符为 '/', 返回结果中不包括代表当前及上一级目录的 "./", "../"

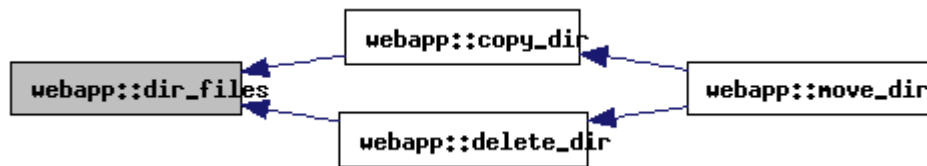
参考 webapp::is_dir().

参考自 webapp::copy_dir(), 以及 webapp::delete_dir().

函数调用图:



这是这个函数的调用关系图:



bool webapp::copy_dir (const string & srcdir, const string & destdir)

拷贝目录

拷贝目录,拷贝子目录时为递归调用

参数:

<i>srcdir</i>	原目录
<i>destdir</i>	目的目录

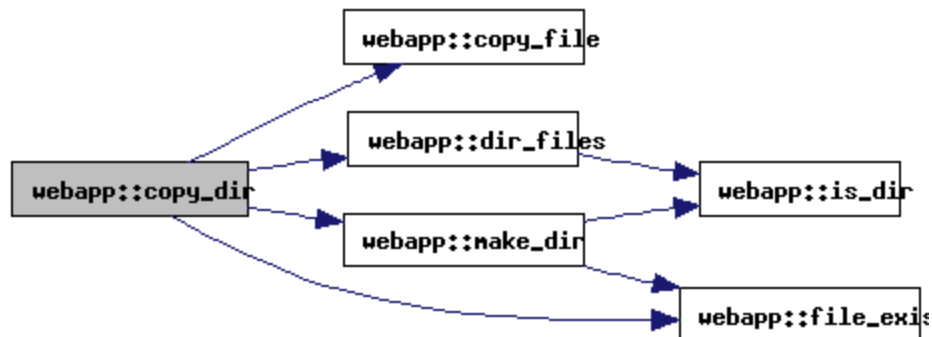
返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考 webapp::copy_file(),webapp::dir_files(),webapp::file_exist(), 以及 webapp::make_dir().

参考自 webapp::move_dir().

函数调用图:



这是这个函数的调用关系图:



bool webapp::delete_dir (const string & dir)

删除目录

删除目录,删除子目录时为递归调用

参数:

<i>dir</i>	要删除的目录
------------	--------

返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考 webapp::delete_file(), 以及 webapp::dir_files().

参考自 webapp::move_dir().

函数调用图:



这是这个函数的调用关系图:



bool webapp::move_dir (const string & srcdir, const string & destdir)

移动目录

参数:

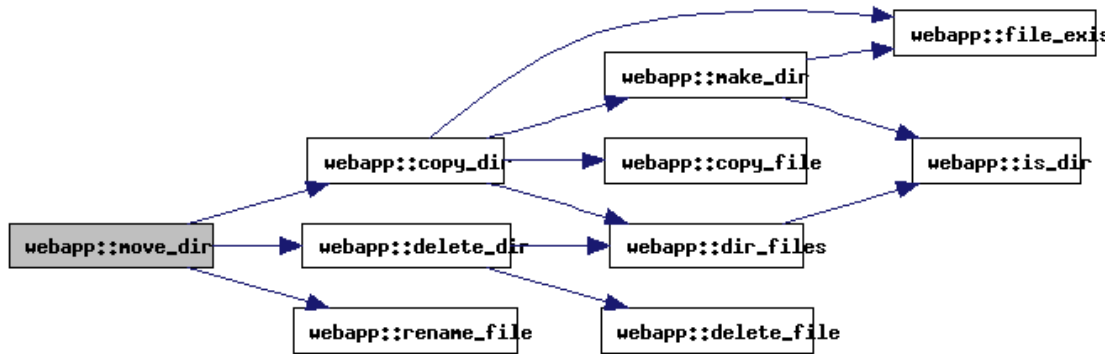
<i>srcdir</i>	原目录
<i>destdir</i>	目的目录

返回值:

<i>true</i>	操作成功
<i>false</i>	失败

参考 webapp::copy_dir(), webapp::delete_dir(), 以及 webapp::rename_file().

函数调用图:



void webapp::lock_file (int *fd*, const int *type*)

文件句柄锁函数

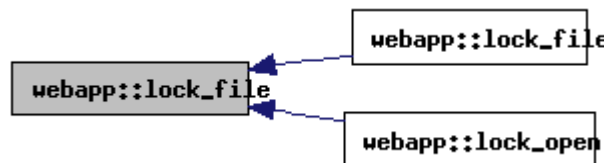
文件句柄锁函数，若文件已被锁则阻塞并等待

参数:

<i>fd</i>	文件句柄
<i>type</i>	锁模式，可选F_WRLCK、F_RDLCK、F_UNLCK

参考自 webapp::lock_file()，以及 webapp::lock_open()。

这是这个函数的调用关系图:



bool webapp::is_locked (int *fd*)

判断文件句柄锁

参数:

<i>fd</i>	文件句柄
-----------	------

返回值:

<i>true</i>	文件已被锁
<i>false</i>	文件未被锁

参考自 webapp::is_locked()。

这是这个函数的调用关系图:



FILE * webapp::lock_open (const string & file, const char * mode, const int type)

申请锁并打开文件

申请锁并打开文件，若文件已被锁则阻塞并等待

参数:

<i>file</i>	文件路径
<i>mode</i>	文件打开模式，与fopen()同参数意义相同
<i>type</i>	锁模式，可选F_WRLCK、F_RDLCK、F_UNLCK

返回:

文件句柄，失败返回NULL

参考 webapp::lock_file().

函数调用图:



waHttpClient相关全局函数

函数

- int [webapp::tcp_request](#) (const string &server, const int port, const string &request, string &response, const int timeout)
发送TCP请求并取得回应内容
- string [webapp::gethost_byname](#) (const string &domain)
根据服务器域名取得IP
- bool [webapp::isip](#) (const string &ipstr)
判断字符串是否为有效IP

详细描述

函数说明

int webapp::tcp_request (const string & server, const int port, const string & request, string & response, const int timeout)

发送TCP请求并取得回应内容

参数:

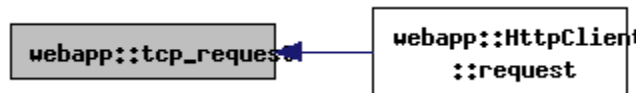
<i>server</i>	服务器IP
<i>port</i>	服务器端口
<i>request</i>	发送的TCP请求
<i>response</i>	服务器的回应内容
<i>timeout</i>	超时时长,单位为秒,为0不判断超时

返回值:

<i>0</i>	执行成功
<i>1</i>	创建socket失败
<i>2</i>	无法连接服务器
<i>3</i>	发送请求失败
<i>4</i>	设置定时器失败或者连接超时
<i>10</i>	未知错误

参考自 webapp::HttpClient::request().

这是这个函数的调用关系图:



string webapp::gethost_byname (const string & domain)

根据服务器域名取得IP

参数:

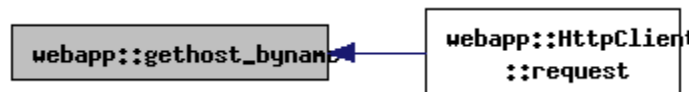
<i>domain</i>	服务器域名 (不包含"HTTP://"头及任何"/"字符)
---------------	-------------------------------

返回:

执行成功返回服务器IP,否则返回空字符串

参考自 webapp::HttpClient::request().

这是这个函数的调用关系图:



bool webapp::isip (const string & ipstr)

判断字符串是否为有效IP

参数:

<i>ipstr</i>	IP字符串
--------------	-------

返回值:

<i>true</i>	有效
-------------	----

<code>false</code>	无效
--------------------	----

参考自 `webapp::HttpClient::request()`.

这是这个函数的调用关系图:



waMysqlClient相关数据类型与全局函数

类型定义

- typedef map< string, string > [webapp::MysqlDataRow](#)
[MysqlData](#) 数据行类型 (map<string,string>)

函数

- string [webapp::escape_sql](#) (const string &str)
SQL 语句字符转义

详细描述

类型定义说明

[webapp::MysqlDataRow](#)

[MysqlData](#) 数据行类型 (map<string,string>)

函数说明

string webapp::escape_sql (const string & str)

SQL 语句字符转义

参数:

要转换的SQL字符串	
------------	--

返回:

转义过的字符串

waString相关全局函数

函数

- string [webapp::itos](#) (const long i, const ios::fmtflags base=ios::dec)
long int转换为string
- long [webapp::stoi](#) (const string &s, const ios::fmtflags base=ios::dec)
string转换为int
- double [webapp::stof](#) (const string &s)
string转换为double
- bool [webapp::isgbk](#) (const unsigned char c1, const unsigned char c2)
判断一个双字节字符是否是GBK编码汉字
- string [webapp::va_sprintf](#) (va_list ap, const string &format)
可变参数字符串格式化，与va_start()、va_end()宏配合使用
- string [webapp::va_str](#) (const char *format,...)
格式化字符串并返回

详细描述

函数说明

string webapp::itos (const long i, const ios::fmtflags base)

long int转换为string

参数:

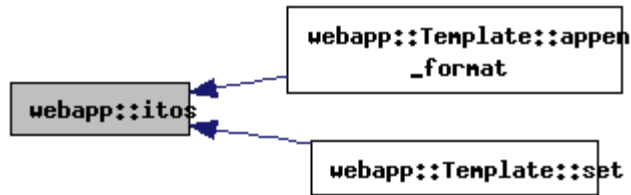
<i>i</i>	long int或者int
<i>base</i>	转换进制参数,可选 <ul style="list-style-type: none">● ios::dec 10进制● ios::oct 8进制● ios::hex 16进制● 默认为10进制

返回:

返回结果string,转换失败返回"0"

参考自 webapp::Template::append_format()，以及 webapp::Template::set().

这是这个函数的调用关系图:



long webapp::stoi (const string & s, const ios::fmtflags base)

string转换为int

string转换为long int

参数:

<i>s</i>	string
<i>base</i>	转换进制参数,可选 <ul style="list-style-type: none"> • ios::dec 10进制 • ios::oct 8进制 • ios::hex 16进制 • 默认为10进制

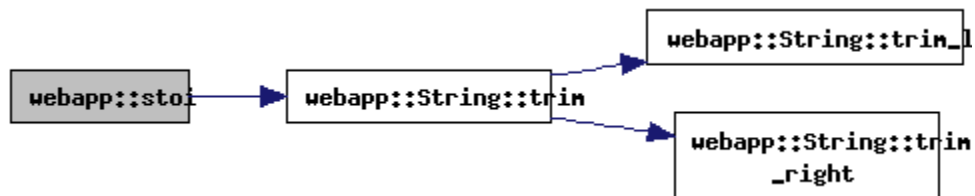
返回:

返回结果long int,转换失败返回0

参考 webapp::String::trim().

参考自 webapp::HttpClient::done().

函数调用图:



这是这个函数的调用关系图:



double webapp::stof (const string & s)

string转换为double

参数:

<i>s</i>	string
----------	--------

返回:

转换成功返回double,否则返回0

bool webapp::isgbk (const unsigned char c1, const unsigned char c2)

判断一个双字节字符是否是GBK编码汉字

参数:

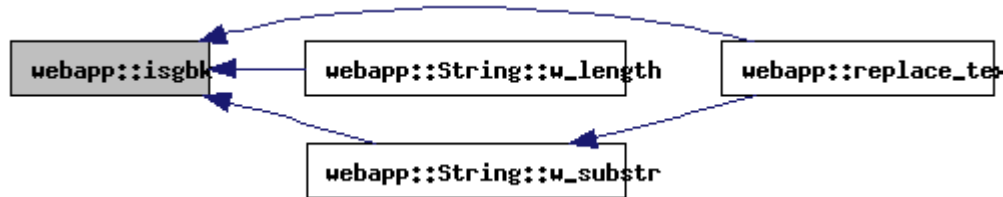
<i>c1</i>	双字节字符1
<i>c2</i>	双字节字符2

返回值:

<i>true</i>	是
<i>false</i>	否

参考自 webapp::replace_text(),webapp::String::w_length(), 以及 webapp::String::w_substr().

这是这个函数的调用关系图:



string webapp::va_sprintf (va_list ap, const string & format)

可变参数字符串格式化, 与va_start()、va_end()宏配合使用

参数:

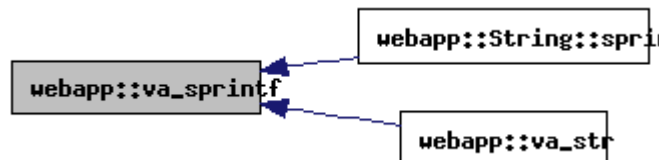
<i>format</i>	字符串格式
<i>ap</i>	可变参数列表

返回:

格式化字符串结果

参考自 webapp::String::sprintf(), 以及 webapp::va_str().

这是这个函数的调用关系图:



string webapp::va_str (const char * format, ...)

格式化字符串并返回

格式化字符串并返回，各参数定义与标准printf()函数完全相同

返回:

格式化字符串结果

参考 webapp::va_sprintf().

函数调用图:



waUtility系统调用工具函数库

宏定义

- #define [EXTRACT_ALL](#) (EXTRACT_ALPHA|EXTRACT_DIGIT|EXTRACT_PUNCT|EXTRACT_SPACE|EXTRACT_HTML)
提取正文函数过滤选项，过滤全部（字母、数字、标点、空白、HTML）

函数

- unsigned int [webapp::string_hash](#) (const string &str)
返回字符串HASH值，基于DJB HASH算法
- string [webapp::replace_text](#) (const string &text, const map< string, string > &replace)
全文词表替换，兼容GBK汉字
- string [webapp::extract_html](#) (const string &html)
提取HTML代码正文
- string [webapp::extract_text](#) (const string &text, const int option=[EXTRACT_ALL](#), const size_t len=0)
全角半角字符转换并提取正文
- void [webapp::file_logger](#) (const string &file, const char *format,...)
追加日志记录
- void [webapp::file_logger](#) (FILE *fp, const char *format,...)
追加日志记录
- string [webapp::system_command](#) (const string &cmd)
执行命令并返回命令输出结果
- string [webapp::host_addr](#) (const string &interface="eth0")
返回指定网卡设备绑定的IP地址

详细描述

宏定义说明

```
#define
EXTRACT_ALL (EXTRACT_ALPHA|EXTRACT_DIGIT|EXTRACT_PUNCT|EXTRACT_SPAC
E|EXTRACT_HTML)
```

提取正文函数过滤选项，过滤全部（字母、数字、标点、空白、HTML）

函数说明

```
unsigned int webapp::string_hash (const string & str)
```

返回字符串HASH值，基于DJB HASH算法
返回字符串HASH值，基于DJB HASH算法 Perl兼容实现版本 string_hash.pl JavaScript
兼容实现版本 string_hash.js

参数:

<i>str</i>	源字符串
------------	------

返回:

字符串HASH结果，无符号整数

```
string webapp::replace_text (const string & text, const map< string, string > & replace)
```

全文词表替换，兼容GBK汉字

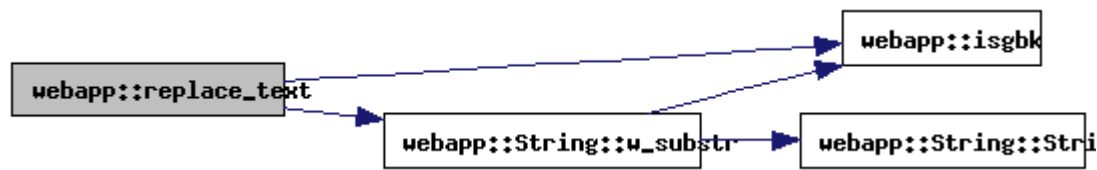
参数:

<i>text</i>	字符串原文
<i>replace</i>	替换对应词表

返回:

替换后结果
参考 webapp::isgbk()，以及 webapp::String::w_substr().

函数调用图:



```
string webapp::extract_html (const string & html)
```

提取HTML代码正文

参数:

<i>html</i>	HTML代码字符串
-------------	-----------

返回:

不含HTML代码的提取结果

参考自 `webapp::extract_text()`.

这是这个函数的调用关系图:



string webapp::extract_text (const string & *text*, const int *option*, const size_t *len*)

全角半角字符转换并提取正文

参数:

<i>text</i>	源字符串
<i>option</i>	过滤范围选项，可选值组合有 <ul style="list-style-type: none">● EXTRACT_ALPHA 过滤字母● EXTRACT_DIGIT 过滤数字● EXTRACT_PUNCT 过滤标点● EXTRACT_SPACE 过滤空白● EXTRACT_HTML 过滤HTML代码● 默认值为EXTRACT_ALL即以上全部
<i>len</i>	过滤长度，大于0时只截取前len个有效字符，默认为0

返回:

转换提取结果字符串，若源字符串内容被全部过滤则返回空

参

考

`webapp::EXTRACT_ALPHA`, `webapp::EXTRACT_DIGIT`, `webapp::EXTRACT_HTML`, `webapp::extract_html()`, `webapp::EXTRACT_PUNCT`，以及 `webapp::EXTRACT_SPACE`.

函数调用图:



void webapp::file_logger (const string & *file*, const char * *format*, ...)

追加日志记录

参数:

<i>file</i>	日志文件路径
<i>format</i>	日志行格式
...	日志数据参数列表

void webapp::file_logger (FILE * *fp*, const char * *format*, ...)

追加日志记录

参数:

<i>fp</i>	日志文件句柄, 或者stdout/stderr
<i>format</i>	日志行格式
...	日志数据参数列表

string webapp::system_command (const string & *cmd*)

执行命令并返回命令输出结果

参数:

<i>cmd</i>	命令字符串, 包括命令行参数
------------	----------------

返回:

命令执行输出结果

string webapp::host_addr (const string & *interface*)

返回指定网卡设备绑定的IP地址

参数:

<i>interface</i>	网卡设备名, 默认为"eth0"
------------------	------------------

返回:

指定网卡设备绑定的IP地址

命名空间文档

webapp 命名空间参考

Web Application Library namespace.

类

- class [Cgi](#)
- CGI参数读取类 class [Cookie](#)
- Cookie读取,设置类 class [ConfigFile](#)
- INI格式配置文件解析类 class [DateTime](#)
- DateTime日期时间运算类 class [HttpClient](#)
- HTTP客户端类 [使用说明文档及简单范例](#) class [MysqlData](#)
- MySQL数据集类 class [MysqlClient](#)

- MySQL数据库连接类 class [String](#)
- 继承自string的字符串类 [基类string 使用说明文档](#) class [Template](#)
- 支持条件、循环脚本的HTML模板处理类 [使用说明文档及简单范例](#) class [TextFile](#)

固定分隔符文本文件读取解析类 类型定义

- typedef map< string, string > [CgiList](#)
[Cgi](#) 参数值列表类型 (map<string,string>)
- typedef map< string, string > [CookieList](#)
[Cookie](#) 参数值列表类型 (map<string,string>)
- typedef map< string, string > [MysqlDataRow](#)
[MysqlData](#) 数据行类型 (map<string,string>)

枚举

- enum [extract_option](#) { [EXTRACT_ALPHA](#) = 2, [EXTRACT_DIGIT](#) = 4, [EXTRACT_PUNCT](#) = 8, [EXTRACT_SPACE](#) = 16, [EXTRACT_HTML](#) = 32 }

函数

- void [http_head](#) ()
输出HTML Content-Type header
- string [get_env](#) (const string &envname)
取得环境变量
- [DateTime operator+](#) (const [DateTime](#) &date1, const [DateTime](#) &date2)
相加操作
- [DateTime operator+](#) (const [DateTime](#) &date, const time_t &tt)
相加操作
- [DateTime operator-](#) (const [DateTime](#) &date1, const [DateTime](#) &date2)
相减操作
- [DateTime operator-](#) (const [DateTime](#) &date, const time_t &tt)
相减操作
- bool [operator==](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间相等比较
- bool [operator==](#) (const [DateTime](#) &left, const time_t &right)
时间相等比较
- bool [operator!=](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间不相等比较
- bool [operator!=](#) (const [DateTime](#) &left, const time_t &right)
时间不相等比较
- bool [operator>](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间大于比较
- bool [operator>](#) (const [DateTime](#) &left, const time_t &right)
时间大于比较
- bool [operator<](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间小于比较
- bool [operator<](#) (const [DateTime](#) &left, const time_t &right)
时间小于比较

- bool [operator>=](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间不小于比较
- bool [operator>=](#) (const [DateTime](#) &left, const time_t &right)
时间不小于比较
- bool [operator<=](#) (const [DateTime](#) &left, const [DateTime](#) &right)
时间不大于比较
- bool [operator<=](#) (const [DateTime](#) &left, const time_t &right)
时间不大于比较
- string [uri_encode](#) (const string &source)
URI编码
- string [uri_decode](#) (const string &source)
URI解码
- string [base64_encode](#) (const string &source)
字符串MIME BASE64 编码
- string [base64_decode](#) (const string &source)
字符串MIME BASE64 解码
- string [md5_encode](#) (const string &source)
MD5编码
- bool [file_exist](#) (const string &file)
文件或者目录是否存在
- bool [is_link](#) (const string &file)
文件是否为链接
- bool [is_dir](#) (const string &file)
是否为目录
- bool [make_link](#) (const string &srcfile, const string &destfile)
建立链接
- long [file_size](#) (const string &file)
取得文件大小
- long [file_time](#) (const string &file)
取得文件更改时间
- string [file_path](#) (const string &file)
取得文件路径
- string [file_name](#) (const string &file)
取得文件名称
- bool [rename_file](#) (const string &oldname, const string &newname)
文件或者目录改名
- bool [copy_file](#) (const string &srcfile, const string &destfile)
拷贝文件
- bool [delete_file](#) (const string &file)
删除文件
- bool [move_file](#) (const string &srcfile, const string &destfile)
移动文件
- vector< string > [dir_files](#) (const string &dir)
返回目录文件列表

- bool [make_dir](#) (const string &dir, const mode_t mode=S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH)
建立目录
- bool [copy_dir](#) (const string &srcdir, const string &destdir)
拷贝目录
- bool [delete_dir](#) (const string &dir)
删除目录
- bool [move_dir](#) (const string &srcdir, const string &destdir)
移动目录
- void [lock_file](#) (int fd, const int type)
文件句柄锁函数
- bool [is_locked](#) (int fd)
判断文件句柄锁
- FILE * [lock_open](#) (const string &file, const char *mode, const int type)
申请锁并打开文件
- void [lock_file](#) (FILE *fp, const int type)
文件句柄锁函数
- bool [is_locked](#) (FILE *fp)
判断文件句柄锁
- int [tcp_request](#) (const string &server, const int port, const string &request, string &response, const int timeout)
发送TCP请求并取得回应内容
- string [gethost_byname](#) (const string &domain)
根据服务器域名取得IP
- bool [isip](#) (const string &ipstr)
判断字符串是否为有效IP
- string [escape_sql](#) (const string &str)
SQL语句字符转义
- string [itos](#) (const long i, const ios::fmtflags base=ios::dec)
long int转换为string
- long [stoi](#) (const string &s, const ios::fmtflags base=ios::dec)
string转换为int
- string [ftos](#) (const double f, const int ndigit=2)
double转换为string
- double [stof](#) (const string &s)
string转换为double
- bool [isgbk](#) (const unsigned char c1, const unsigned char c2)
判断一个双字节字符是否是GBK编码汉字
- string [va_sprintf](#) (va_list ap, const string &format)
可变参数字符串格式化, 与va_start()、va_end()宏配合使用
- string [va_str](#) (const char *format,...)
格式化字符串并返回
- unsigned int [string_hash](#) (const string &str)
返回字符串HASH值, 基于DJB HASH算法
- string [replace_text](#) (const string &text, const map< string, string > &replace)

全文词表替换, 兼容GBK汉字

- string [extract_html](#) (const string &html)
提取HTML代码正文
- string [extract_text](#) (const string &text, const int option=[EXTRACT_ALL](#), const size_t len=0)
全角半角字符转换并提取正文
- void [file_logger](#) (const string &file, const char *format,...)
追加日志记录
- void [file_logger](#) (FILE *fp, const char *format,...)
追加日志记录
- string [system_command](#) (const string &cmd)
执行命令并返回命令输出结果
- string [host_addr](#) (const string &interface="eth0")
返回指定网卡设备绑定的IP地址

详细描述

Web Application Library namespace.

枚举类型说明

enum [webapp::extract_option](#)

枚举值:

EXTRACT_ALPHA 过滤英文字母
EXTRACT_DIGIT 过滤阿拉伯数字
EXTRACT_PUNCT 过滤半角标点
EXTRACT_SPACE 过滤空白
EXTRACT_HTML 过滤HTML代码

函数说明

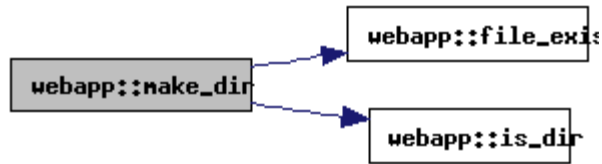
bool webapp::make_dir (const string & dir, const mode_t mode)

建立目录

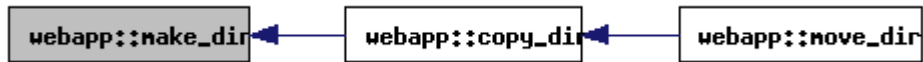
参考 [file_exist\(\)](#), 以及 [is_dir\(\)](#).

参考自 [copy_dir\(\)](#).

函数调用图:



这是这个函数的调用关系图:



void webapp::lock_file (FILE * *fp*, const int *type*)[inline]

文件句柄锁函数

参考 lock_file().

函数调用图:



bool webapp::is_locked (FILE * *fp*)[inline]

判断文件句柄锁

参考 is_locked().

函数调用图:



string webapp::ftos (const double *f*, const int *ndigit*)

double转换为string

类说明

webapp::Cgi类 参考

CGI参数读取类

#include <waCgi.h>

webapp::Cgi 的协作图:

webapp::Cgi
<ul style="list-style-type: none"> + Cgi() + ~Cgi() + get_cgi() + operator[] + is_trunc() + dump()

Public 成员函数

- [Cgi](#) (const long formdata_maxsize=0)
构造函数
- virtual [~Cgi](#) ()
析构函数
- string [get_cgi](#) (const string &name)
取得CGI参数
- string [operator\[\]](#) (const string &name)
取得CGI参数
- bool [is_trunc](#) () const
FORM数据大小是否超出限制
- [CgiList dump](#) () const
返回参数值列表

详细描述

CGI参数读取类

构造及析构函数说明

webapp::Cgi::Cgi (const long formdata_maxsize = 0)

构造函数
构造函数 读取并分析CGI内容

参数:

<i>formdata_maxsize</i>	参数是"multipart/form-data"方式POST时的最大FORM上传数据大小, 超过部分被截断不处理,单位为byte,默认为0即不限制数据大小
-------------------------	---

参考 webapp::get_env().

函数调用图:



virtual webapp::Cgi::~Cgi () [inline], [virtual]

析构函数

成员函数说明

string webapp::Cgi::get_cgi (const string & name)

取得CGI参数

参数:

<i>name</i>	CGI参数名,大小写敏感
-------------	--------------

返回:

成功返回CGI参数值,否则返回空字符串,多个同名CGI参数值之间分隔符为半角空格''

string webapp::Cgi::operator[] (const string & name) [inline]

取得CGI参数

bool webapp::Cgi::is_trunc () const [inline]

FORM数据大小是否超出限制

[CgiList](#) webapp::Cgi::dump () const [inline]

返回参数值列表

返回:

返回值类型为CgiList,即map<string,string>.

该类的文档由以下文件生成:

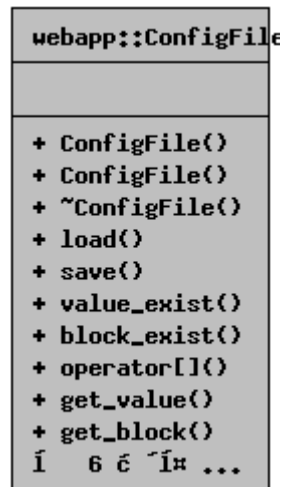
- [waCgi.h](#)
- [waCgi.cpp](#)

webapp::ConfigFile类 参考

INI格式配置文件解析类

```
#include <waConfigFile.h>
```

webapp::ConfigFile 的协作图:



Public 成员函数

- [ConfigFile \(\)](#)
默认构造函数
- [ConfigFile \(const string &file\)](#)
参数为配置文件名的构造函数
- [~ConfigFile \(\)](#)
析构函数
- bool [load](#) (const string &file)
读取解析配置文件
- bool [save](#) (const string &file="")
保存配置文件
- bool [value_exist](#) (const string &block, const string &name)
检查配置项是否存在
- bool [block_exist](#) (const string &block)
检查配置块是否存在
- string [operator\[\]](#) (const string &name)
读取配置项参数值
- string [get_value](#) (const string &block, const string &name, const string &default_value="")
读取配置项参数值
- map< string, string > [get_block](#) (const string &block)
读取指定配置块的全部配置项参数值
- vector< string > [block_list](#) ()
读取全部配置块列表
- bool [set_value](#) (const string &name, const string &value)
更新配置项
- bool [set_value](#) (const string &block, const string &name, const string &value)

更新配置项

- bool [set_block](#) (const string &block, const map< string, string > &valuelist)
更新指定配置块的配置项列表

- void [del_value](#) (const string &block, const string &name)
删除配置项

- void [del_block](#) (const string &block)
删除配置块

详细描述

INI格式配置文件解析类

构造及析构函数说明

webapp::ConfigFile::ConfigFile () [inline]

默认构造函数

webapp::ConfigFile::ConfigFile (const string & file) [inline]

参数为配置文件名的构造函数

webapp::ConfigFile::~~ConfigFile () [inline]

析构函数

成员函数说明

bool webapp::ConfigFile::load (const string & file)

读取解析配置文件

参数:

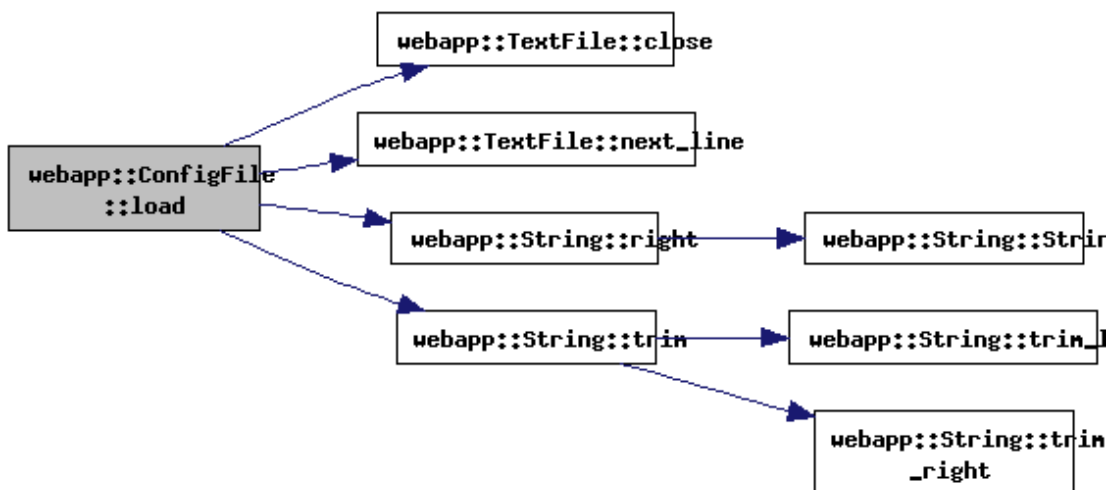
<i>file</i>	配置文件路径名
-------------	---------

返回值:

<i>true</i>	解析成功
<i>false</i>	解析失败

参 考 webapp::TextFile::close(),webapp::TextFile::next_line(),webapp::String::right() , 以 及 webapp::String::trim().

函数调用图:



bool webapp::ConfigFile::save (const string & file = "")

保存配置文件

参数:

<i>file</i>	配置文件路径名，默认为空则使用读取文件参数
-------------	-----------------------

返回值:

<i>true</i>	保存成功
<i>false</i>	保存失败

参考 webapp::String::save_file().

函数调用图:



bool webapp::ConfigFile::value_exist (const string & block, const string & name)

检查配置项是否存在

参数:

<i>block</i>	配置块名，为空表示全局配置块
<i>name</i>	配置项名

返回值:

<i>true</i>	存在
<i>false</i>	不存在

bool webapp::ConfigFile::block_exist (const string & *block*)

检查配置块是否存在

参数:

<i>block</i>	配置块名，为空表示全局配置块
--------------	----------------

返回值:

<i>true</i>	存在
<i>false</i>	不存在

string webapp::ConfigFile::operator[] (const string & *name*) [inline]

读取配置项参数值

string webapp::ConfigFile::get_value (const string & *block*, const string & *name*, const string & *default_value* = "")

读取配置项参数值

参数:

<i>block</i>	配置块名，为空表示全局配置块
<i>name</i>	配置项名
<i>default_value</i>	默认参数值，配置项不存在时返回

返回:

指定配置项参数值

map< string, string > webapp::ConfigFile::get_block (const string & *block*)

读取指定配置块的全部配置项参数值

参数:

<i>block</i>	配置块名，为空表示全局配置块
--------------	----------------

返回:

指定配置块的全部配置项参数值列表

vector< string > webapp::ConfigFile::block_list ()

读取全部配置块列表

返回:

全部配置块列表，包括全局配置块（*block*值为空）

bool webapp::ConfigFile::set_value (const string & *name*, const string & *value*) [inline]

更新配置项

bool webapp::ConfigFile::set_value (const string & *block*, const string & *name*, const string & *value*)

更新配置项

参数:

<i>block</i>	配置块名，为空表示全局配置块
<i>name</i>	配置项名
<i>value</i>	配置参数值

返回值:

<i>true</i>	更新成功
<i>false</i>	更新失败

bool webapp::ConfigFile::set_block (const string & *block*, const map< string, string > & *valuelist*)

更新指定配置块的配置项列表

参数:

<i>block</i>	配置块名，为空表示全局配置块
<i>valuelist</i>	配置参数值对列表

返回值:

<i>true</i>	更新成功
<i>false</i>	更新失败

void webapp::ConfigFile::del_value (const string & *block*, const string & *name*)

删除配置项

参数:

<i>block</i>	配置块名，为空表示全局配置块
<i>name</i>	配置项名

void webapp::ConfigFile::del_block (const string & *block*)

删除配置块

参数:

<code>block</code>	配置块名, 为空表示全局配置块
--------------------	-----------------

该类的文档由以下文件生成:

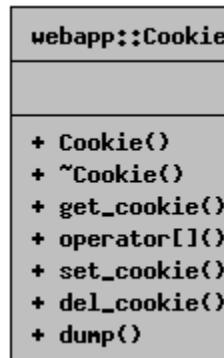
- [waConfigFile.h](#)
- [waConfigFile.cpp](#)

webapp::Cookie类 参考

Cookie读取,设置类

`#include <waCgi.h>`

webapp::Cookie 的协作图:



Public 成员函数

- [Cookie \(\)](#)
构造函数
- `virtual` [~Cookie \(\)](#)
析构函数
- `string` [get_cookie](#) (const string &name)
取得cookie内容
- `string` [operator\[\]](#) (const string &name)
取得cookie内容
- `void` [set_cookie](#) (const string &name, const string &value, const string &expires="", const string &path="/", const string &domain="") const
设置cookie内容
- `void` [del_cookie](#) (const string &name) const
清除指定的cookie内容
- [CookieList dump](#) () const
返回参数值列表

详细描述

Cookie读取,设置类

构造及析构函数说明

webapp::Cookie::Cookie ()

构造函数

构造函数 读取并分析Cookie环境变量

参考 webapp::get_env().

函数调用图:



virtual webapp::Cookie::~~Cookie () [inline], [virtual]

析构函数

成员函数说明

string webapp::Cookie::get_cookie (const string & name)

取得cookie内容

参数:

<i>name</i>	cookie参数名,大小写敏感
-------------	-----------------

返回:

成功返回cookie参数值,否则返回空字符串

string webapp::Cookie::operator[] (const string & name)[inline]

取得cookie内容

void webapp::Cookie::set_cookie (const string & name, const string & value, const string & expires = "", const string & path = "/", const string & domain = "") const

设置cookie内容

设置cookie内容 必须在输出content-type前调用

参数:

<i>name</i>	cookie名字
<i>value</i>	cookie值
<i>expires</i>	cookie有效期,GMT格式日期字符串,默认为空
<i>path</i>	cookie路径,默认为"/"
<i>domain</i>	cookie域,默认为""

void webapp::Cookie::del_cookie (const string & *name*) const [inline]

清除指定的cookie内容

参数:

<i>name</i>	cookie名字
-------------	----------

[CookieList](#) webapp::Cookie::dump () const [inline]

返回参数值列表

返回:

返回值类型为CookieList,即map<string,string>.

该类的文档由以下文件生成:

- [waCgi.h](#)
- [waCgi.cpp](#)

webapp::DateTime类 参考

DateTime日期时间运算类

```
#include <waDateTime.h>
```

webapp::DateTime 的协作图:

webapp::DateTime
+ DateTime() + DateTime() + DateTime() + DateTime() + DateTime() + DateTime() + ~DateTime() + operator=() + operator=() + operator+=() 1 29 6 11 ...

Public 成员函数

- [DateTime](#) ()
默认构造函数,以当前时间构造对象
- [DateTime](#) (const time_t &tt)
参数为 time_t 的构造函数
- [DateTime](#) (const int year, const int mon, const int mday, const int hour=0, const int min=0, const int sec=0)
参数为指定时间的构造函数
- [DateTime](#) (const tm &st)
参数为 tm 结构的构造函数
- [DateTime](#) (const string &datetime, const string &datemark="-", const string &dtmark=" ", const string &timemark=":")
参数为"YYYY-MM-DD HH:MM:SS" 格式字符串的构造函数
- [DateTime](#) (const [DateTime](#) &date)
拷贝构造函数
- virtual ~[DateTime](#) ()
析构函数
- [DateTime](#) & [operator=](#) (const [DateTime](#) &date)
赋值操作
- [DateTime](#) & [operator=](#) (const time_t &tt)
赋值操作
- [DateTime](#) & [operator+=](#) (const [DateTime](#) &date)
递增操作
- [DateTime](#) & [operator+=](#) (const time_t &tt)
递增操作
- [DateTime](#) & [operator-=](#) (const [DateTime](#) &date)
递减操作
- [DateTime](#) & [operator-=](#) (const time_t &tt)
递减操作
- int [year](#) () const

返回四位数年份

- `int month () const`
返回月份, 范围1~12
- `int m_day () const`
返回当月第几天, 范围1~31
- `int m_days () const`
返回当月天数, 范围1~31
- `int w_day () const`
返回当周第几天, 周一至周六返回1~6, 周日返回0
- `int y_day () const`
返回当年第几天, 范围0~365
- `int hour () const`
返回小时, 范围0~23
- `int min () const`
返回分钟, 范围0~59
- `int sec () const`
返回秒数, 范围0~59
- `long secs () const`
返回 1970-1-1 0:0:0 以来的秒数
- `long mins () const`
返回 1970-1-1 0:0:0 以来的分钟数
- `long hours () const`
返回 1970-1-1 0:0:0 以来的小时数
- `long days () const`
返回 1970-1-1 0:0:0 以来的天数
- `long weeks () const`
返回 1970-1-1 0:0:0 以来的周数
- `void set ()`
以当前时间设置对象
- `void set (const time_t &tt)`
以 `time_t` 参数设置对象
- `void set (const tm &st)`
以 `tm` 结构参数设置对象
- `void set (const int year, const int mon, const int mday, const int hour=0, const int min=0, const int sec=0)`
以指定时间设置对象
- `void set (const DateTime &date)`
以 `DateTime` 参数设置对象
- `void set (const string &datetime, const string &datemark="-", const string &dtmark="", const string &timemark=":")`
以"YYYY-MM-DD HH:MM:SS" 格式字符串设置对象
- `time_t value () const`
返回 `time_t` 类型的对象值
- `tm struct_tm () const`
返回 `struct tm` 类型的对象值

- string [date](#) (const string &datemark="-", const bool leadingzero=true) const
输出日期字符串
- string [time](#) (const string &timemark=":", const bool leadingzero=true) const
输出时间字符串
- string [datetime](#) (const string &datemark="-", const string &dtmark=" ", const string &timemark=":", const bool leadingzero=true) const
输出日期时间字符串
- string [gmt_datetime](#) () const
输出 GMT 格式日期时间字符串

详细描述

DateTime日期时间运算类

构造及析构函数说明

webapp::DateTime::DateTime () [inline]

默认构造函数,以当前时间构造对象

webapp::DateTime::DateTime (const time_t & tt) [inline]

参数为 time_t 的构造函数

webapp::DateTime::DateTime (const int year, const int mon, const int mday, const int hour = 0, const int min = 0, const int sec = 0) [inline]

参数为指定时间的构造函数

webapp::DateTime::DateTime (const tm & st) [inline]

参数为 tm 结构的构造函数

webapp::DateTime::DateTime (const string & datetime, const string & datemark = "-", const string & dtmark = " ", const string & timemark = ":") [inline]

参数为"YYYY-MM-DD HH:MM:SS"格式字符串的构造函数

webapp::DateTime::DateTime (const [DateTime](#) & date) [inline]

拷贝构造函数

virtual webapp::DateTime::~~DateTime () [inline], [virtual]

析构函数

成员函数说明

[DateTime](#) & webapp::DateTime::operator= (const [DateTime](#) & date)

赋值操作

[DateTime](#) & webapp::DateTime::operator= (const time_t & tt)

赋值操作

[DateTime](#) & webapp::DateTime::operator+= (const [DateTime](#) & date)

递增操作

参考 value().

函数调用图:



[DateTime](#) & webapp::DateTime::operator+= (const time_t & tt)

递增操作

[DateTime](#) & webapp::DateTime::operator-= (const [DateTime](#) & date)

递减操作

参考 value().

函数调用图:



[DateTime](#) & webapp::DateTime::operator-= (const time_t & tt)

递减操作

int webapp::DateTime::year () const [inline]

返回四位数年份

int webapp::DateTime::month () const [inline]

返回月份，范围1~12

int webapp::DateTime::m_day () const [inline]

返回当月第几天，范围1~31

int webapp::DateTime::m_days () const

返回当月天数，范围1~31

int webapp::DateTime::w_day () const [inline]

返回当周第几天，周一至周六返回1~6，周日返回0

int webapp::DateTime::y_day () const [inline]

返回当年第几天，范围0~365

int webapp::DateTime::hour () const [inline]

返回小时，范围0~23

int webapp::DateTime::min () const [inline]

返回分钟，范围0~59

int webapp::DateTime::sec () const [inline]

返回秒数，范围0~59

long webapp::DateTime::secs () const [inline]

返回 1970-1-1 0:0:0 以来的秒数

long webapp::DateTime::mins () const [inline]

返回 1970-1-1 0:0:0 以来的分钟数

参考 TIME_ONE_MIN.

long webapp::DateTime::hours () const [inline]

返回 1970-1-1 0:0:0 以来的小时数

参考 TIME_ONE_HOUR.

long webapp::DateTime::days () const [inline]

返回 1970-1-1 0:0:0 以来的天数

参考 TIME_ONE_DAY.

long webapp::DateTime::weeks () const [inline]

返回 1970-1-1 0:0:0 以来的周数

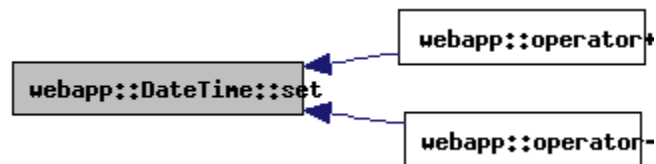
参考 TIME_ONE_WEEK.

void webapp::DateTime::set ()

以当前时间设置对象

参考自 webapp::operator+(), 以及 webapp::operator-().

这是这个函数的调用关系图:



void webapp::DateTime::set (const time_t & tt)

以 time_t 参数设置对象

参数:

<i>tt</i>	time_t类型参数
-----------	------------

void webapp::DateTime::set (const tm & st)

以 tm 结构参数设置对象

参数:

<i>st</i>	struct tm类型参数
-----------	---------------

void webapp::DateTime::set (const int year, const int mon, const int mday, const int hour = 0, const int min = 0, const int sec = 0)

以指定时间设置对象
以指定时间设置对象 若参数不是有效日期时间,则设置为系统初始时间（1970/1/1） 若参数日期时间不存在,则设置为顺延有效时间（非闰年2/29视为3/1）

参数:

year	年
mon	月
mday	日
hour	时,默认为0
min	分,默认为0
src	秒,默认为0

void webapp::DateTime::set (const [DateTime](#) & date)

以 [DateTime](#) 参数设置对象

参数:

date	Date类型参数
------	----------

参考 value().

函数调用图:



void webapp::DateTime::set (const string & datetime, const string & datemark = "-", const string & dtmark = " ", const string & timemark = ":")

以"YYYY-MM-DD HH:MM:SS"格式字符串设置对象
以"YYYY-MM-DD HH:MM:SS"格式字符串设置对象 若字符串格式错误或者时间值错误则设置为当前时间

参数:

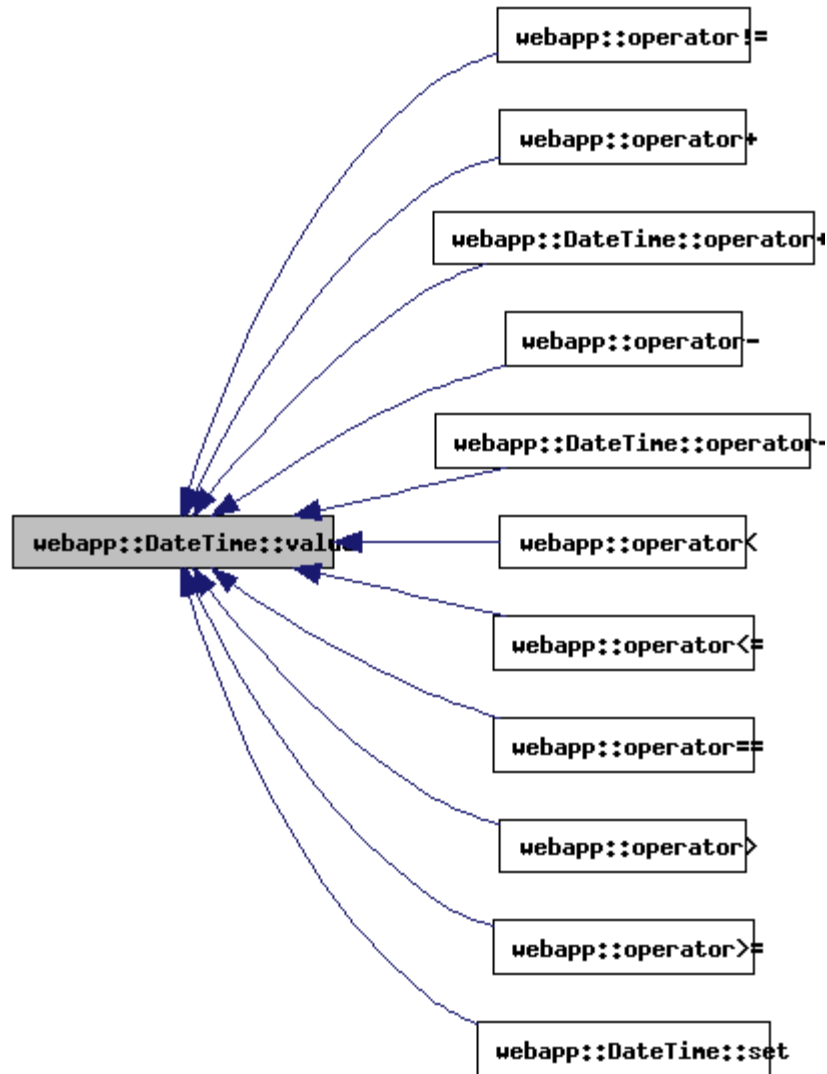
datetime	"YYYY-MM-DD HH:MM:SS"格式日期时间字符串
datemark	日期分隔字符,默认为"-"
dtmark	日期时间分隔字符,默认为" ",不能与datemark或timemark相同
timemark	时间分隔字符,默认为":"

time_t webapp::DateTime::value () const [inline]

返回 time_t 类型的对象值
参 考 自
webapp::operator!(),webapp::operator+(),operator+(),webapp::operator-(),operator-(),webapp::ope

rator<(),webapp::operator<=(),webapp::operator==(),webapp::operator>(),webapp::operator>=() , 以及 set().

这是这个函数的调用关系图:



```
tm webapp::DateTime::struct_tm () const[inline]
```

返回 struct tm 类型的对象值

```
string webapp::DateTime::date (const string & datemark = "-", const bool leadingzero = true) const
```

输出日期字符串

参数:

<i>datemark</i>	日期分隔字符,默认为"-"
<i>leadingzero</i>	是否补充前置零,默认为否

返回:

输出指定格式的日期字符串

string webapp::DateTime::time (const string & *timemark* = ":", const bool *leadingzero* = true) const

输出时间字符串

参数:

<i>timemark</i>	时间分隔字符,默认为":"
<i>leadingzero</i>	是否补充前置零,默认为否

返回:

输出指定格式的时间字符串

string webapp::DateTime::datetime (const string & *datemark* = "-", const string & *dtmark* = " ", const string & *timemark* = ":", const bool *leadingzero* = true) const

输出日期时间字符串

参数:

<i>datemark</i>	日期分隔字符,默认为"-"
<i>dtmark</i>	日期时间分隔字符,默认为" "
<i>timemark</i>	时间分隔字符,默认为":"
<i>leadingzero</i>	是否补充前置零,默认为否

返回:

输出指定格式的日期时间字符串

string webapp::DateTime::gmt_datetime () const

输出 GMT 格式日期时间字符串

输出 GMT 格式日期时间字符串 主要用于设置 cookie 有效期

返回:

GMT 格式日期时间字符串

该类的文档由以下文件生成:

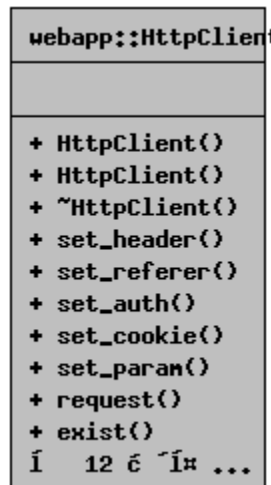
- [waDateTime.h](#)
- [waDateTime.cpp](#)

webapp::HttpClient类 参考

HTTP客户端类 [使用说明文档及简单范例](#)

#include <waHttpClient.h>

webapp::HttpClient 的协作图:



Public 类型

- enum `error_msg` { [ERROR_NULL](#) = 0, [ERROR_CREATE_SOCKET](#) = 1, [ERROR_CONNECT_SERVER](#) = 2, [ERROR_SEND_REQUEST](#) = 3, [ERROR_RESPONSE_TIMEDOUT](#) = 4, [ERROR_SERVERINFO_NULL](#) = 5, [ERROR_REQUEST_NULL](#) = 6, [ERROR_RESPONSE_NULL](#) = 7, [ERROR_RESPONSE_INVALID](#) = 8, [ERROR_HTTPSTATUS](#) = 9, [ERROR_UNKNOWN](#) = 10 }

Public 成员函数

- [HttpClient](#) ()
默认构造函数
- [HttpClient](#) (const string &url, const string &server="", const int port=80, const string &method="GET", const int timeout=5)
构造并执行HTTP请求
- virtual [~HttpClient](#) ()
析构函数
- void [set_header](#) (const string &name, const string &value)
设置指定的HTTP请求Header
- void [set_referer](#) (const string &referer)
设置HTTP请求Referer Header
- void [set_auth](#) (const string &username, const string &password)
设置HTTP请求Authorization Header
- void [set_cookie](#) (const string &name, const string &value)

设置HTTP请求Cookie Header

- void [set_param](#) (const string &name, const string &value)
设置HTTP请求CGI参数
- bool [request](#) (const string &url, const string &server="", const int port=80, const string &method="GET", const int timeout=5)
执行HTTP请求
- bool [exist](#) (const string &url, const string &server="", const int port=80)
URL 是否有效
- string [get_header](#) (const string &name)
获取指定的HTTP返回Header
- vector< [String](#) > [get_cookie](#) ()
获取HTTP返回Set-Cookie Header
- string [dump_header](#) ()
获取HTTP返回Header
- bool [done](#) () const
执行HTTP请求是否成功
- void [clear](#) ()
清空所有设置及状态值
- string [status](#) () const
获取HTTP返回Status
- string [content](#) () const
获取HTTP返回Content 正文
- unsigned int [content_length](#) () const
获取HTTP返回Content 正文长度(Content-Length)
- [error_msg](#) [errnum](#) () const
返回错误信息代码 代码信息定义参见 [Http::error_msg](#)
- string [error](#) () const
返回错误信息描述
- string [dump_request](#) () const
输出生成的HTTP请求全文
- string [dump_response](#) () const
输出获得的服务器返回全文

详细描述

HTTP客户端类 [使用说明文档及简单范例](#)

成员枚举类型说明

enum [webapp::HttpClient::error_msg](#)

枚举值:

ERROR_NULL 无错误
ERROR_CREATE_SOCKET 创建socket失败
ERROR_CONNECT_SERVER 无法连接服务器
ERROR_SEND_REQUEST 发送请求失败
ERROR_RESPONSE_TIMEDOUT 设置定时器失败或者连接超时
ERROR_SERVERINFO_NULL 服务器地址信息错误
ERROR_REQUEST_NULL HTTP请求格式错误
ERROR_RESPONSE_NULL 服务器回应为空
ERROR_RESPONSE_INVALID 服务器回应格式错误
ERROR_HTTPSTATUS 服务器回应HTTP状态错误
ERROR_UNKNOWN 未知错误

构造及析构函数说明

webapp::HttpClient::HttpClient () [inline]

默认构造函数

webapp::HttpClient::HttpClient (const string & url, const string & server = "", const int port = 80, const string & method = "GET", const int timeout = 5) [inline]

构造并执行HTTP请求

参数:

<i>url</i>	HTTP请求URL
<i>server</i>	服务器IP,为空字符串则根据参数1获得,默认为空字符串
<i>port</i>	服务器端口,默认为80
<i>method</i>	HTTP请求Method,默认为"GET"
<i>timeout</i>	HTTP请求超时时长,单位为秒,默认为5秒,为0不判断超时

virtual webapp::HttpClient::~~HttpClient () [inline], [virtual]

析构函数

成员函数说明

void webapp::HttpClient::set_header (const string & name, const string & value)

设置指定的HTTP请求Header

参数:

<i>name</i>	Header名称
<i>value</i>	Header值,

void webapp::HttpClient::set_referer (const string & *referer*)

设置HTTP请求Referer Header

参数:

<i>referer</i>	Referer Header值
----------------	-----------------

void webapp::HttpClient::set_auth (const string & *username*, const string & *password*)

设置HTTP请求Authorization Header

参数:

<i>username</i>	用户名
<i>password</i>	用户口令

参考 webapp::base64_encode().

函数调用图:



void webapp::HttpClient::set_cookie (const string & *name*, const string & *value*)

设置HTTP请求Cookie Header

参数:

<i>name</i>	Cookie名称
<i>value</i>	Cookie值

参考 webapp::uri_encode().

函数调用图:



void webapp::HttpClient::set_param (const string & *name*, const string & *value*)

设置HTTP请求CGI参数

参数:

<i>name</i>	CGI参数名称
<i>value</i>	CGI参数值

参考 webapp::uri_encode().

函数调用图:



bool webapp::HttpClient::request (const string & *url*, const string & *host* = "", const int *port* = 80, const string & *method* = "GET", const int *timeout* = 5)

执行HTTP请求

参数:

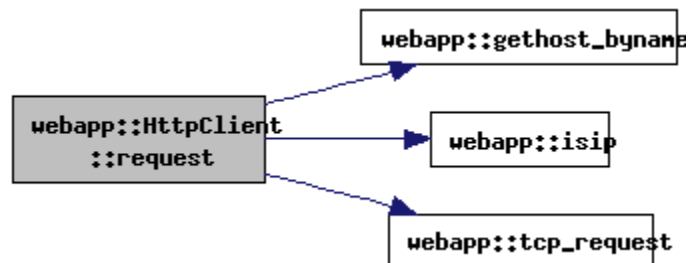
<i>url</i>	HTTP请求URL
<i>server</i>	服务器IP或者域名,为空字符串则根据参数1获得,默认为空字符串, 若参数 <i>url</i> , <i>server</i> 都不包含服务器地址信息,则函数返回失败
<i>port</i>	服务器端口,默认为80
<i>method</i>	HTTP请求Method,默认为"GET"
<i>timeout</i>	HTTP请求超时时长,单位为秒,默认为5秒

返回值:

<i>true</i>	执行成功
<i>false</i>	执行失败

参考 webapp::gethost_byname(),webapp::isip(), 以及 webapp::tcp_request().

函数调用图:



bool webapp::HttpClient::exist (const string & *url*, const string & *server* = "", const int *port* = 80)

URL 是否有效

参数:

<i>url</i>	HTTP请求URL
<i>server</i>	服务器IP,为空字符串则根据参数1获得,默认为空字符串,若参数url,server都不包含服务器地址信息,则函数返回失败
<i>port</i>	服务器端口,默认为80

返回值:

<i>true</i>	URL有效
<i>false</i>	URL已失效

string webapp::HttpClient::get_header (const string & name)

获取指定的HTTP返回Header

参数:

<i>name</i>	Header名称,
-------------	-----------

返回:

成功返回Header值,否则返回空字符串

vector< [String](#) > webapp::HttpClient::get_cookie ()

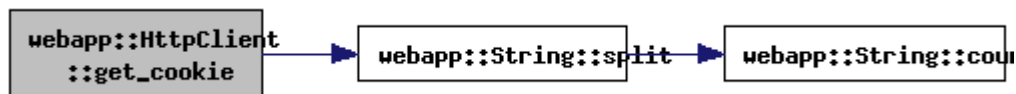
获取HTTP返回Set-Cookie Header

返回:

返回Cookie列表数组,每个元素为一个Cookie值

参考 webapp::String::split().

函数调用图:



string webapp::HttpClient::dump_header ()

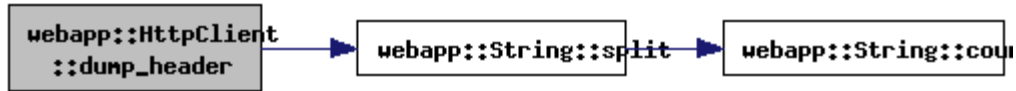
获取HTTP返回Header

返回:

HTTP返回Header字符串

参考 webapp::String::split().

函数调用图:



bool webapp::HttpClient::done () const

执行HTTP请求是否成功

返回值:

<i>true</i>	成功
<i>false</i>	失败

参考 webapp::stoi().

函数调用图:



void webapp::HttpClient::clear ()

清空所有设置及状态值

string webapp::HttpClient::status () const[inline]

获取HTTP返回Status

返回:

HTTP返回Status字符串

string webapp::HttpClient::content () const[inline]

获取HTTP返回Content正文

返回:

HTTP返回Content正文

unsigned int webapp::HttpClient::content_length () const[inline]

获取HTTP返回Content正文长度(Content-Length)

返回:

HTTP返回Content正文长度

[error_msg](#) **webapp::HttpClient::errnum () const [inline]**

返回错误信息代码 代码信息定义参见 Http::error_msg

string webapp::HttpClient::error () const

返回错误信息描述

返回:

返回错误信息描述

string webapp::HttpClient::dump_request () const [inline]

输出生成的HTTP请求全文

返回:

返回生成的HTTP请求全文

string webapp::HttpClient::dump_response () const [inline]

输出获得的服务器返回全文

返回:

返回获得的服务器返回全文

该类的文档由以下文件生成:

- [waHttpClient.h](#)
- [waHttpClient.cpp](#)

webapp::MysqlClient类 参考

MySQL数据库连接类

#include <waMysqlClient.h>

webapp::MysqlClient 的协作图:

webapp::MysqlClient
<ul style="list-style-type: none"> + MysqlClient() + MysqlClient() + ~MysqlClient() + connect() + disconnect() + is_connected() + select_db() + query() + query() + query_val() 1 6 6 1x ...

Public 成员函数

- [MysqlClient](#) ()
Mysql 默认构造函数
- [MysqlClient](#) (const string &host, const string &user, const string &pwd, const string &database, const int port=0, const char *socket=NULL)
Mysql 构造函数
- virtual [~MysqlClient](#) ()
Mysql 析构函数
- bool [connect](#) (const string &host, const string &user, const string &pwd, const string &database, const int port=0, const char *socket=NULL)
连接数据库
- void [disconnect](#) ()
断开数据库连接
- bool [is_connected](#) ()
判断是否连接数据库
- bool [select_db](#) (const string &database)
选择数据库
- bool [query](#) (const string &sqlstr, [MysqlData](#) &records)
执行SQL语句,取得查询结果
- bool [query](#) (const string &sqlstr)
执行SQL语句
- string [query_val](#) (const string &sqlstr, const unsigned long row=0, const unsigned int col=0)
返回查询结果中指定位置的字符串值
- [MysqlDataRow](#) [query_row](#) (const string &sqlstr, const unsigned long row=0)
返回查询结果中指定行
- unsigned long [affected](#) ()
上次查询动作所影响的记录条数
- unsigned long [last_id](#) ()
取得上次查询的一个AUTO_INCREMENT列生成的ID
- string [error](#) ()

取得Mysql错误信息

- unsigned int [errnum](#) ()
取得Mysql错误编号
- string [info](#) ()
取得更新信息

详细描述

MySQL数据库连接类

构造及析构函数说明

webapp::MysqlClient::MysqlClient () [inline]

Mysql默认构造函数

webapp::MysqlClient::MysqlClient (const string & *host*, const string & *user*, const string & *pwd*, const string & *database*, const int *port* = 0, const char * *socket* = NULL) [inline]

Mysql构造函数

参数:

<i>host</i>	MySQL主机IP
<i>user</i>	MySQL用户名
<i>pwd</i>	用户口令
<i>database</i>	要打开的数据库
<i>port</i>	数据库端口，默认为0
<i>socket</i>	UNIX_SOCKET，默认为NULL

virtual webapp::MysqlClient::~~MysqlClient () [inline], [virtual]

Mysql析构函数

成员函数说明

bool webapp::MysqlClient::connect (const string & *host*, const string & *user*, const string & *pwd*, const string & *database*, const int *port* = 0, const char * *socket* = NULL)

连接数据库

参数:

<i>host</i>	MySQL主机IP
<i>user</i>	MySQL用户名
<i>pwd</i>	用户口令
<i>database</i>	要打开的数据库
<i>port</i>	数据库端口, 默认为0
<i>socket</i>	UNIX_SOCKET, 默认为NULL

返回值:

<i>true</i>	成功
<i>false</i>	失败

void webapp::MysqlClient::disconnect ()

断开数据库连接

bool webapp::MysqlClient::is_connected ()

判断是否连接数据库

返回值:

<i>true</i>	连接
<i>false</i>	断开

bool webapp::MysqlClient::select_db (const string & database)

选择数据库

参数:

<i>database</i>	数据库名
-----------------	------

返回值:

<i>true</i>	成功
<i>false</i>	失败

bool webapp::MysqlClient::query (const string & sqlstr, [MysqlData](#) & records)

执行SQL语句,取得查询结果

参数:

<i>sqlstr</i>	要执行的SQL语句
<i>records</i>	保存数据结果的MysqlData对象

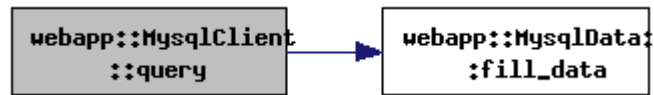
返回值:

<i>true</i>	成功
-------------	----

<i>false</i>	失败
--------------	----

参考 `webapp::MysqlData::fill_data()`.

函数调用图:



bool webapp::MysqlClient::query (const string & sqlstr)

执行SQL语句

参数:

<i>sqlstr</i>	要执行的SQL语句
---------------	-----------

返回值:

<i>true</i>	成功
<i>false</i>	失败

string webapp::MysqlClient::query_val (const string & sqlstr, const unsigned long row = 0, const unsigned int col = 0)

返回查询结果中指定位置的字符串值

参数:

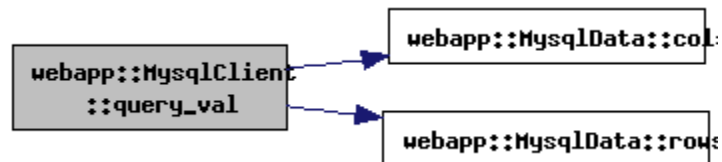
<i>sqlstr</i>	SQL查询字符串
<i>row</i>	数据行位置,默认为0
<i>col</i>	数据列位置,默认为0

返回:

查询成功返回字符串,否则返回空字符串

参考 `webapp::MysqlData::cols()`, 以及 `webapp::MysqlData::rows()`.

函数调用图:



[MysqlDataRow](#) webapp::MysqlClient::query_row (const string & sqlstr, const unsigned long row = 0)

返回查询结果中指定行

参数:

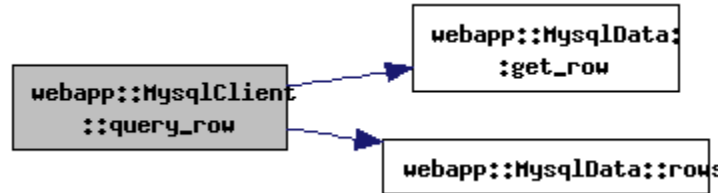
<i>sqlstr</i>	SQL查询字符串
<i>row</i>	数据行位置,默认为0

返回:

返回值类型为MysqlDataRow,即map<string,string>

参考 webapp::MysqlData::get_row(), 以及 webapp::MysqlData::rows().

函数调用图:



unsigned long webapp::MysqlClient::affected ()

上次查询动作所影响的记录条数

返回:

返回记录条数,类型unsigned long

unsigned long webapp::MysqlClient::last_id ()

取得上次查询的一个AUTO_INCREMENT列生成的ID

取得上次查询的一个AUTO_INCREMENT列生成的ID 一个Mysql表只能有一个AUTO_INCREMENT列,且必须为索引

返回:

返回生成的ID

string webapp::MysqlClient::error () [inline]

取得Mysql错误信息

返回:

返回错误信息字符串

unsigned int webapp::MysqlClient::errnum () [inline]

取得Mysql错误编号

返回:

返回错误信息编号

string webapp::MysqlClient::info ()

取得更新信息

返回:

返回更新信息

该类的文档由以下文件生成:

- [waMysqlClient.h](#)
- [waMysqlClient.cpp](#)

webapp::MysqlData类 参考

MySQL数据集类

```
#include <waMysqlClient.h>
```

webapp::MysqlData 的协作图:

webapp::MysqlData
<div><div># _rows</div><div># _cols</div><div># _curpos</div><div># _fetched</div><div># _mysqlres</div><div># _mysqlrow</div><div># _mysqlfields</div><div># _field_pos</div></div>
<div><div>+ MysqlData()</div><div>+ ~MysqlData()</div><div>+ operator()()</div><div>+ get_data()</div><div>+ operator()()</div><div>+ get_data()</div><div>+ get_row()</div><div>+ rows()</div><div>+ cols()</div><div>+ field_pos()</div><div>+ field_name()</div><div># fill_data()</div></div>

Public 成员函数

- [MysqlData](#) ()
MysqlData 构造函数
- virtual [~MysqlData](#) ()
MysqlData 析构函数
- string [operator\(\)](#) (const unsigned long row, const unsigned int col)
返回指定位置的MysqlData 数据
- string [get_data](#) (const unsigned long row, const unsigned int col)
返回指定位置的MysqlData 数据
- string [operator\(\)](#) (const unsigned long row, const string &field)
返回指定字段的MysqlData 数据
- string [get_data](#) (const unsigned long row, const string &field)
返回指定字段的MysqlData 数据
- [MysqlDataRow get_row](#) (const long row=-1)
返回指定位置的MysqlData 数据行
- unsigned long [rows](#) () const
返回MysqlData 数据行数
- unsigned int [cols](#) () const
返回MysqlData 数据列数
- int [field_pos](#) (const string &field)
返回字段位置
- string [field_name](#) (const unsigned int col) const
返回字段名称

Protected 成员函数

- bool [fill_data](#) (MYSQL *mysql)
填充MysqlData 数据

详细描述

MySQL数据集类

构造及析构函数说明

webapp::MysqlData::MysqlData () [inline]

MysqlData构造函数

webapp::MysqlData::~~MysqlData () [virtual]

MysqlData析构函数

成员函数说明

bool webapp::MysqlData::fill_data (MYSQL * *mysql*) [protected]

填充MysqlData数据

参数:

<i>mysql</i>	MYSQL*参数
--------------	----------

返回值:

<i>true</i>	成功
<i>false</i>	失败

参考自 webapp::MysqlClient::query().

这是这个函数的调用关系图:



string webapp::MysqlData::operator() (const unsigned long *row*, const unsigned int *col*) [inline]

返回指定位置的MysqlData数据

参数:

<i>row</i>	行位置
<i>col</i>	列位置

返回:

数据字符串

string webapp::MysqlData::get_data (const unsigned long *row*, const unsigned int *col*)

返回指定位置的MysqlData数据

参数:

<i>row</i>	数据行位置,默认为0
<i>col</i>	数据列位置,默认为0

返回:

返回数据,不存在则返回空字符串

string webapp::MysqlData::operator() (const unsigned long *row*, const string & *field*) [inline]

返回指定字段的MysqlData数据

参数:

<i>row</i>	行位置
<i>field</i>	字段名

返回:

数据字符串

string webapp::MysqlData::get_data (const unsigned long row, const string & field)

返回指定字段的MysqlData数据

参数:

<i>row</i>	行位置
<i>field</i>	字段名

返回:

数据字符串,不存在返回空字符串

[MysqlDataRow](#) **webapp::MysqlData::get_row (const long row = -1)**

返回指定位置的MysqlData数据行

参数:

<i>row</i>	数据行位置,默认为当前纪录位置, 当前纪录位置由 first(),prior(),next(),last(),find()函数操纵,默认为0
------------	--

返回:

返回值类型为MysqlDataRow,即map<string,string>

参考自 webapp::MysqlClient::query_row().

这是这个函数的调用关系图:

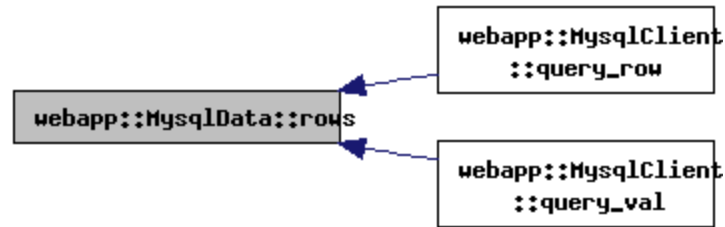


unsigned long webapp::MysqlData::rows () const [inline]

返回MysqlData数据行数

参考自 webapp::MysqlClient::query_row(), 以及 webapp::MysqlClient::query_val().

这是这个函数的调用关系图:

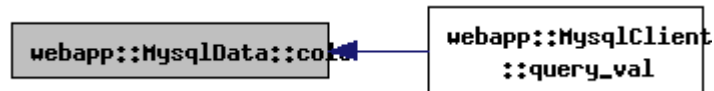


unsigned int webapp::MysqlData::cols () const [inline]

返回MysqlData数据列数

参考自 webapp::MysqlClient::query_val().

这是这个函数的调用关系图:



int webapp::MysqlData::field_pos (const string & field)

返回字段位置

参数:

<i>field</i>	字段名
--------------	-----

返回:

若数据结果中存在该字段则返回字段位置,否则返回-1

string webapp::MysqlData::field_name (const unsigned int col) const

返回字段名称

参数:

<i>col</i>	字段位置
------------	------

返回:

若数据结果中存在该字段则返回字段名称,否则返回空字符串

该类的文档由以下文件生成:

- [waMysqlClient.h](#)
- [waMysqlClient.cpp](#)

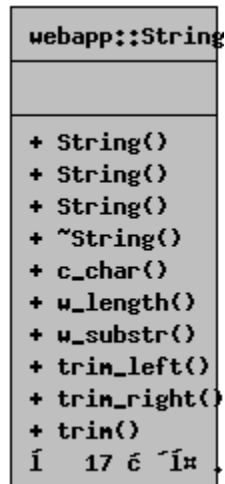
webapp::String类 参考

继承自string的字符串类 [基类string使用说明文档](#)

#include <waString.h>

继承自 std::string .

webapp::String 的协作图:



Public 类型

- enum [split_mode](#) { [SPLIT_IGNORE_BLANK](#), [SPLIT_KEEP_BLANK](#) }

Public 成员函数

- [String](#) ()
默认构造函数
- [String](#) (const char *s)
参数为char*的构造函数
- [String](#) (const string &s)
参数为string的构造函数
- virtual [~String](#) ()
析构函数
- char * [c_char](#) () const
返回 char* 型结果, 调用者必须调用 delete[] 释放所返回内存
- string::size_type [w_length](#) () const
返回字符数量, 支持全角字符
- [String w_substr](#) (const string::size_type pos=0, const string::size_type n=npos) const
截取子字符串, 支持全角字符
- void [trim_left](#) (const string &blank=BLANK_CHARS)
清除左侧空白字符
- void [trim_right](#) (const string &blank=BLANK_CHARS)
清除右侧空白字符
- void [trim](#) (const string &blank=BLANK_CHARS)

清除两侧空白字符

- [String left](#) (const string::size_type n) const
从左边截取指定长度子串
- [String mid](#) (const string::size_type pos, const string::size_type n=npos) const
从中间截取指定长度子串
- [String right](#) (const string::size_type n) const
从右边截取指定长度子串
- void [resize](#) (const string::size_type n)
调整字符串长度
- int [count](#) (const string &str) const
统计指定子串出现的次数
- vector< [String](#) > [split](#) (const string &tag, const int limit=0, const [split_mode](#) mode=[SPLIT_IGNORE_BLANK](#)) const
根据分割符分割字符串
- map< string, string > [tomap](#) (const string &itemtag="&", const string &exptag="=") const
转换字符串为MAP结构(map<string,string>)
- void [join](#) (const vector< string > &strings, const string &tag)
组合字符串
- bool [sprintf](#) (const char *format,...)
格式化赋值
- int [replace](#) (const string &oldstr, const string &newstr)
替换
- int [replace_all](#) (const string &oldstr, const string &newstr)
全文替换
- void [upper](#) ()
转换为大写字母
- void [lower](#) ()
转换为小写字母
- bool [isnum](#) () const
字符串是否完全由数字组成
- bool [load_file](#) (const string &filename)
读取文件到字符串
- bool [save_file](#) (const string &filename, const ios::openmode mode=ios::trunc|ios::out, const mode_t permission=S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) const
保存字符串到文件

详细描述

继承自string的字符串类 [基类string使用说明文档](#)

成员枚举类型说明

enum [webapp::String::split_mode](#)

枚举值:

SPLIT_IGNORE_BLANK 忽略连续多个分隔符, 返回结果不含空字段

SPLIT_KEEP_BLANK 不忽略连续多个分隔符, 返回结果包含空字段

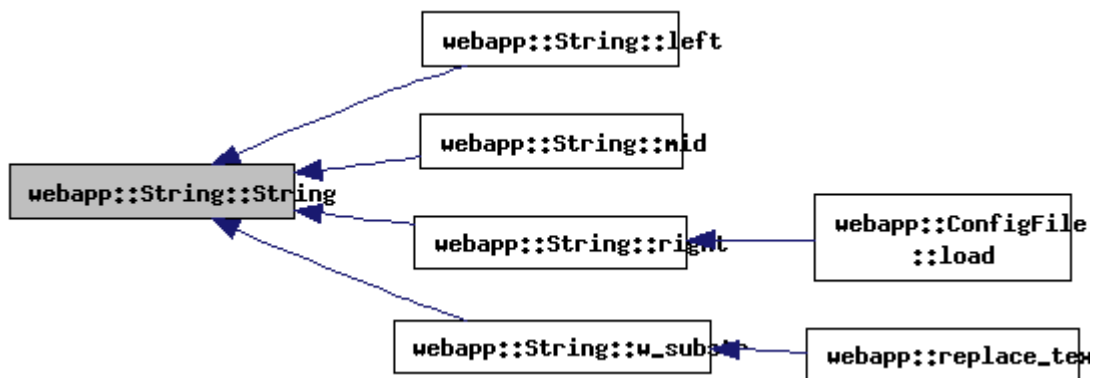
构造及析构造函数说明

webapp::String::String () [inline]

默认构造函数

参考自 left(),mid(),right(), 以及 w_substr().

这是这个函数的调用关系图:



webapp::String::String (const char * s) [inline]

参数为char*的构造函数

webapp::String::String (const string & s) [inline]

参数为string的构造函数

virtual webapp::String::~~String () [inline], [virtual]

析构造函数

成员函数说明

char * webapp::String::c_char () const

返回 char* 型结果，调用者必须调用 delete[] 释放所返回内存

返回:
char*类型数据结果

string::size_type webapp::String::w_length () const

返回字符数量，支持全角字符
返回字符数量，GBK汉字算作一个字符

返回:
字符数量
参考 webapp::isgbk().

函数调用图:



String webapp::String::w_substr (const string::size_type pos = 0, const string::size_type n = npos) const

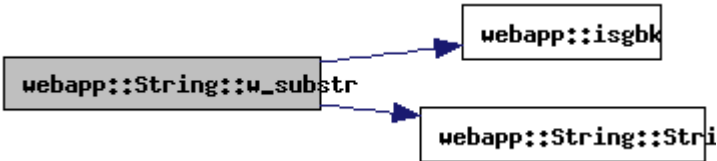
截取子字符串，支持全角字符
截取子字符串,避免出现半个汉字 若截取结果的首尾为半个汉字则删除,删除半个汉字后结果可能为空字符串，该函数避免在截取时将一个完整汉字分开,对字符串中原有的不完整汉字字符不作处理

参数:

<i>pos</i>	起始位置,默认为0,单字节计数方式
<i>n</i>	要截取的字符串长度,默认为到末尾,单字节计数方式

返回:
所截取的字符串
参考 webapp::isgbk() , 以及 String().
参考自 webapp::replace_text().

函数调用图:



这是这个函数的调用关系图:



void webapp::String::trim_left (const string & *blank* = BLANK_CHARS)

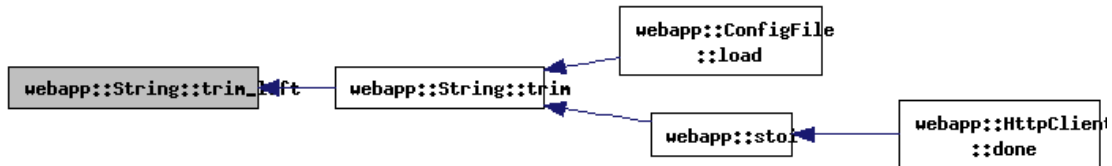
清除左侧空白字符

参数:

<i>blank</i>	要过滤掉的空白字符列表,默认为webapp::BLANK_CHARS
--------------	------------------------------------

参考自 trim().

这是这个函数的调用关系图:



void webapp::String::trim_right (const string & *blank* = BLANK_CHARS)

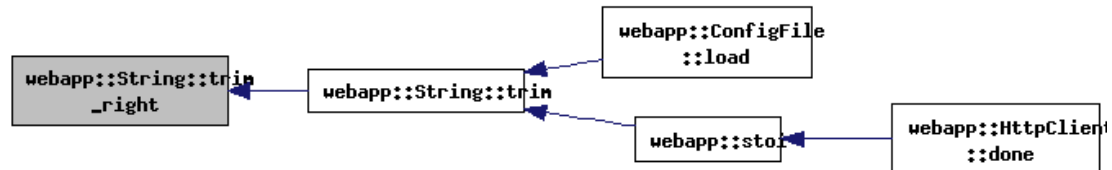
清除右侧空白字符

参数:

<i>blank</i>	要过滤掉的空白字符列表,默认为webapp::BLANK_CHARS
--------------	------------------------------------

参考自 trim().

这是这个函数的调用关系图:



void webapp::String::trim (const string & *blank* = BLANK_CHARS)

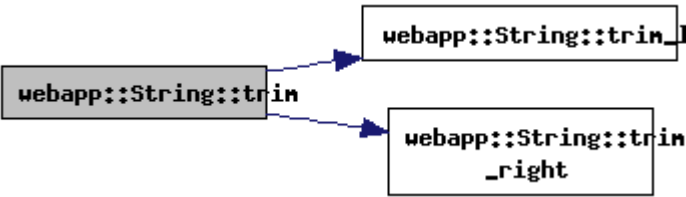
清除两侧空白字符

参数:

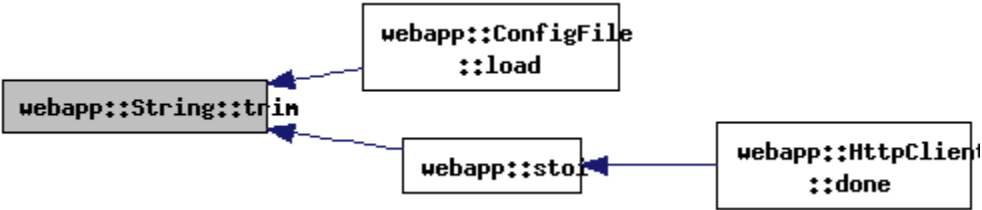
<i>blank</i>	要过滤掉的空白字符列表,默认为webapp::BLANK_CHARS
--------------	------------------------------------

参考 trim_left(), 以及 trim_right().

参考自 webapp::ConfigFile::load() , 以及 webapp::stoi().
函数调用图:



这是这个函数的调用关系图:



String webapp::String::left (const string::size_type *n*) const

从左边截取指定长度子串

参数:

<i>n</i>	要截取的字符串长度,若长度超出则返回原字符串
----------	------------------------

返回:

所截取的字符串

参考 String().

函数调用图:



String webapp::String::mid (const string::size_type *pos*, const string::size_type *n* = npos) const

从中间截取指定长度子串

参数:

<i>pos</i>	开始截取的位置
<i>n</i>	要截取的字符串长度,若长度超出则返回原字符串,默认为到末尾

返回:

所截取的字符串

参考 String().

函数调用图:



String webapp::String::right (const string::size_type n) const

从右边截取指定长度子串

参数:

<i>n</i>	要截取的字符串长度,若长度超出则返回原字符串
----------	------------------------

返回:

所截取的字符串

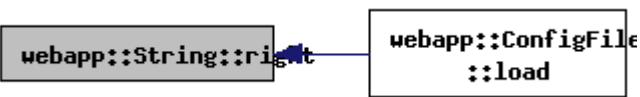
参考 String().

参考自 webapp::ConfigFile::load().

函数调用图:



这是这个函数的调用关系图:



void webapp::String::resize (const string::size_type n)

调整字符串长度

参数:

<i>n</i>	新字符串长度,若小于当前长度则截断,若大于当前长度则补充空白字符
----------	----------------------------------

int webapp::String::count (const string & str) const

统计指定子串出现的次数

参数:

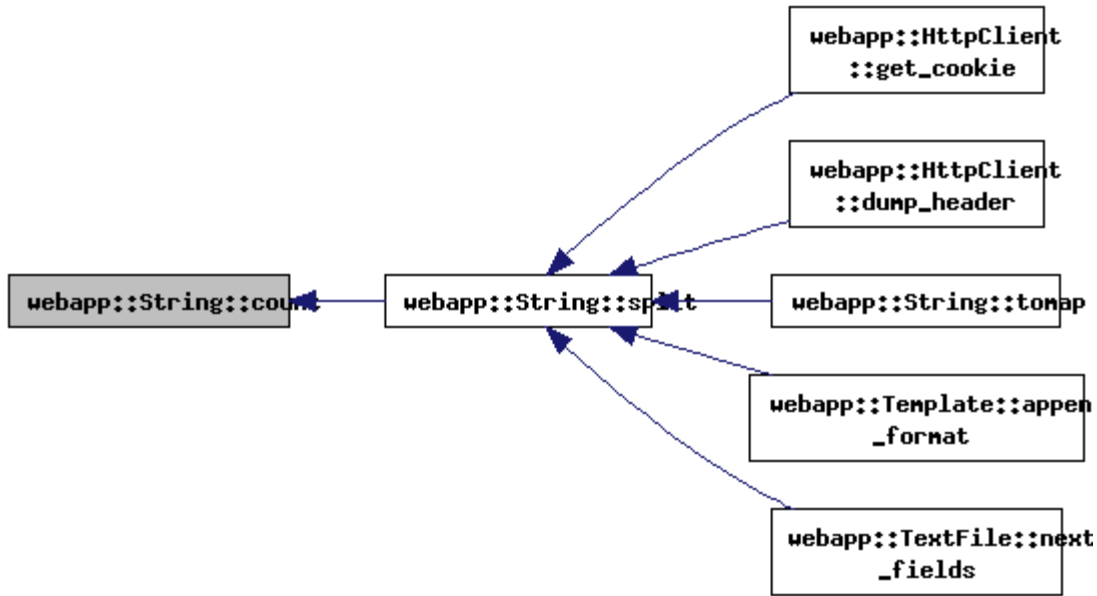
<i>str</i>	要查找的子串
------------	--------

返回:

子串不重复出现的次数

参考自 split().

这是这个函数的调用关系图:



```
vector< String > webapp::String::split (const string & tag, const int limit = 0, const
split\_mode mode = SPLIT\_IGNORE\_BLANK) const
```

根据分割符分割字符串

参数:

<i>tag</i>	分割标记字符串
<i>limit</i>	分割次数限制,默认为0即不限制
<i>mode</i>	结果返回模式,可选 <ul style="list-style-type: none"> ● String::SPLIT_IGNORE_BLANK 忽略连续多个分隔符, 返回结果不含空字段 ● String::SPLIT_KEEP_BLANK 不忽略连续多个分隔符, 返回结果包含空字段 ● 默认为String::SPLIT_IGNORE_BLANK

返回:

分割结果字符串数组 vector<String>

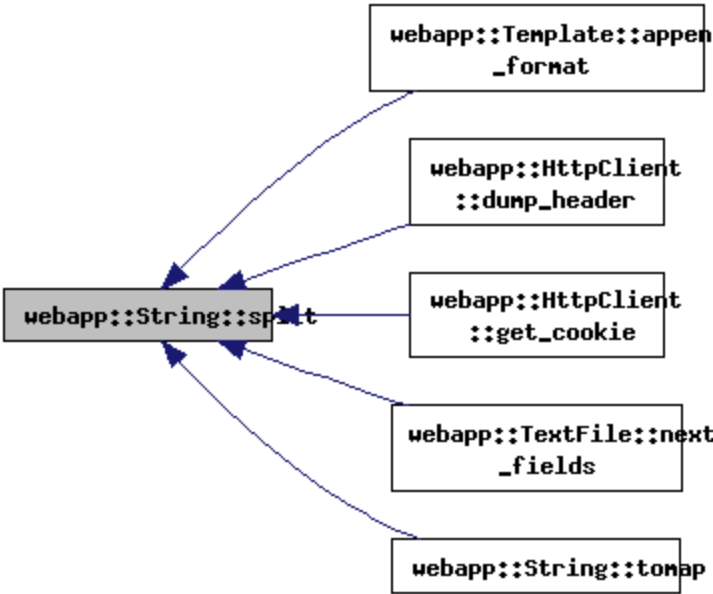
参考 count(), 以及 SPLIT_IGNORE_BLANK.

参 考 自
webapp::Template::append_format(),webapp::HttpClient::dump_header(),webapp::HttpClient::get_cookie(),webapp::TextFile::next_fields(), 以及 tomap().

函数调用图:



这是这个函数的调用关系图:



map< string, string > webapp::String::tomap (const string & *itemtag* = "&", const string & *exptag* = "=") const

转换字符串为MAP结构(map<string,string>)

参数:

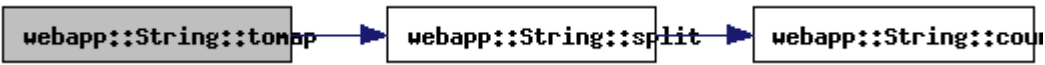
<i>itemtag</i>	表达式之间的分隔符,默认为"&"
<i>exptag</i>	表达式中变量名与变量值之间的分隔符,默认为"="

返回:

转换结果 map<string,string>

参考 split().

函数调用图:



void webapp::String::join (const vector< string > & *strings*, const string & *tag*)

组合字符串

组合字符串,与split()相反

参数:

<i>strings</i>	字符串数组
<i>tag</i>	组合分隔符

bool webapp::String::sprintf (const char * *format*, ...)

格式化赋值

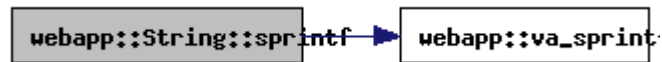
格式化赋值 各参数定义与标准sprintf()函数完全相同

返回值:

<i>true</i>	执行成功
<i>false</i>	失败

参考 webapp::va_sprintf().

函数调用图:



int webapp::String::replace (const string & *oldstr*, const string & *newstr*)

替换

替换 该函数重载了string::replace()

参数:

<i>oldstr</i>	被替换掉的字符串
<i>newstr</i>	用来替换旧字符串的新字符串

返回值:

<i>1</i>	替换成功
<i>0</i>	失败

参考自 replace_all().

这是这个函数的调用关系图:



int webapp::String::replace_all (const string & *oldstr*, const string & *newstr*)

全文替换

参数:

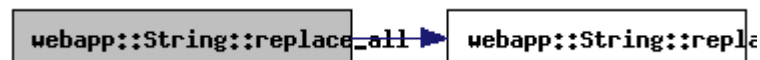
<i>oldstr</i>	被替换掉的字符串
<i>newstr</i>	用来替换旧字符串的新字符串

返回:

执行替换的次数

参考 replace().

函数调用图:



void webapp::String::upper ()

转换为大写字母

void webapp::String::lower ()

转换为小写字母

bool webapp::String::isnum () const

字符串是否完全由数字组成

返回值:

<i>true</i>	是
<i>false</i>	否

bool webapp::String::load_file (const string & filename)

读取文件到字符串

参数:

<i>filename</i>	要读取的文件完整路径名称
-----------------	--------------

返回值:

<i>true</i>	读取成功
<i>false</i>	失败

bool webapp::String::save_file (const string & filename, const ios::openmode mode = ios::trunc|ios::out, const mode_t permission = S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) const

保存字符串到文件

参数:

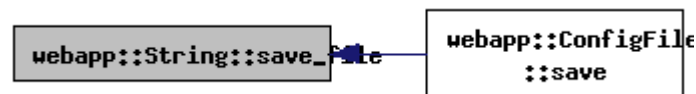
<i>filename</i>	要写入的文件路径名称
<i>mode</i>	写入方式,默认为ios::trunc ios::out
<i>permission</i>	文件属性参数, 默认为0666

返回值:

<i>true</i>	写入成功
<i>false</i>	失败

参考自 webapp::ConfigFile::save().

这是这个函数的调用关系图:



该类的文档由以下文件生成:

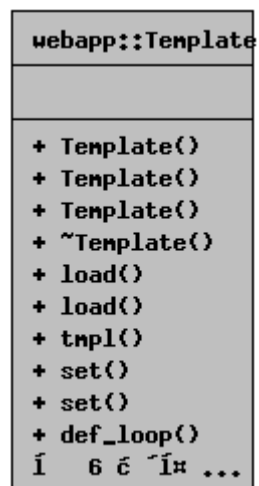
- [waString.h](#)
- [waString.cpp](#)

webapp::Template类 参考

支持条件、循环脚本的HTML模板处理类 [使用说明文档及简单范例](#)

#include <waTemplate.h>

webapp::Template 的协作图:



Public 类型

- enum [output_mode](#) { [TMPL_OUTPUT_DEBUG](#), [TMPL_OUTPUT_RELEASE](#) }

Public 成员函数

- [Template](#) ()
默认构造函数
- [Template](#) (const string tmpl_file)
构造函数
- [Template](#) (const string tmpl_dir, const string tmpl_file)
构造函数
- virtual [~Template](#) ()

析构函数

- bool [load](#) (const string &tmpl_file)
读取HTML模板文件
- bool [load](#) (const string &tmpl_dir, const string &tmpl_file)
读取模板
- void [tmpl](#) (const string &tmpl)
设置HTML模板内容
- void [set](#) (const string &name, const string &value)
设置替换规则
- void [set](#) (const string &name, const long value)
设置替换规则
- void [def_loop](#) (const string &loop, const char *field_0,...)
新建循环
- void [append_row](#) (const string &loop, const char *value_0,...)
添加一行数据到循环
- void [append_format](#) (const string &loop, const char *format,...)
添加一行指定格式的数据到循环
- void [clear_set](#) ()
清空所有替换规则
- string [html](#) ()
返回HTML字符串
- void [print](#) (const [output_mode](#) mode=[TMPL_OUTPUT_RELEASE](#))
输出HTML到stdout
- bool [print](#) (const string &file, const [output_mode](#) mode=[TMPL_OUTPUT_RELEASE](#), const mode_t permission=S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)
输出HTML到文件

详细描述

支持条件、循环脚本的HTML模板处理类 [使用说明文档及简单范例](#)

成员枚举类型说明

enum [webapp::Template::output_mode](#)

枚举值:

TMPL_OUTPUT_DEBUG 显示调试信息

TMPL_OUTPUT_RELEASE 不显示

构造及析构造函数说明

webapp::Template::Template () [inline]

默认构造函数

webapp::Template::Template (const string *tpl_file*) [inline]

构造函数

参数:

<i>tpl_file</i>	模板文件
-----------------	------

webapp::Template::Template (const string *tpl_dir*, const string *tpl_file*) [inline]

构造函数

参数:

<i>tpl_dir</i>	模板目录
<i>tpl_file</i>	模板文件

virtual webapp::Template::~~Template () [inline], [virtual]

析构造函数

成员函数说明

bool webapp::Template::load (const string & *tpl_file*)

读取HTML模板文件

参数:

<i>tpl_file</i>	模板路径文件名
-----------------	---------

返回值:

<i>true</i>	读取成功
<i>false</i>	失败

bool webapp::Template::load (const string & *tpl_dir*, const string & *tpl_file*) [inline]

读取模板

参数:

<i>tmpl_dir</i>	模板目录
<i>tmpl_file</i>	模板文件

返回值:

<i>true</i>	读取成功
<i>false</i>	失败

void webapp::Template::tmpl (const string & *tmpl*)

设置HTML模板内容

参数:

<i>tmpl</i>	模板内容字符串
-------------	---------

void webapp::Template::set (const string & *name*, const string & *value*)

设置替换规则

参数:

<i>name</i>	模板域名称
<i>value</i>	替换值

void webapp::Template::set (const string & *name*, const long *value*)[inline]

设置替换规则

参数:

<i>name</i>	模板域名称
<i>value</i>	替换值

参考 webapp::itos().

函数调用图:



void webapp::Template::def_loop (const string & *loop*, const char * *field_0*, ...)

新建循环

参数:

<i>loop</i>	循环名称
<i>field_0</i>	field_0及field_0之后为字段名称列表,最后一个参数必须是NULL
...	字段名称列表,最后一个参数必须是NULL

void webapp::Template::append_row (const string & *loop*, const char * *value_0*, ...)

添加一行数据到循环

添加一行数据到循环 必须先调用Template::def_loop()初始化循环字段定义,否则中止

参数:

<i>loop</i>	循环名称
<i>value_0</i>	value_0及value_0之后为字段名称列表,最后一个参数必须是NULL
...	字段名称列表,最后一个参数必须是NULL

void webapp::Template::append_format (const string & *loop*, const char * *format*, ...)

添加一行指定格式的数据到循环

添加一行指定格式的数据到循环 必须先调用Template::def_loop()初始化循环字段定义,否则中止

参数:

<i>loop</i>	循环名称
<i>format</i>	字段列循环式定义,"%d,%s,..."格式
...	第三个参数起为字段值列表, 字段值参数个数不能少于格式定义参数format中指定的个数

参考 webapp::itos(), 以及 webapp::String::split().

函数调用图:



void webapp::Template::clear_set ()

清空所有替换规则

清空所有替换规则 包括所有循环替换规则

string webapp::Template::html ()

返回HTML字符串

返回:

返回模板分析处理结果

void webapp::Template::print (const [output_mode](#) mode = [TMPL_OUTPUT_RELEASE](#))

输出HTML到stdout

参数:

<i>mode</i>	是否输出调试信息 <ul style="list-style-type: none">● Template::TMPL_OUTPUT_DEBUG 输出调试信息● Template::TMPL_OUTPUT_RELEASE 不输出调试信息● 默认为不输出调试信息
-------------	--

bool webapp::Template::print (const string & file, const [output_mode](#) mode = [TMPL_OUTPUT_RELEASE](#), const mode_t permission = S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

输出HTML到文件

参数:

<i>file</i>	输出文件名
<i>mode</i>	是否输出调试信息 <ul style="list-style-type: none">● Template::TMPL_OUTPUT_DEBUG 输出调试信息● Template::TMPL_OUTPUT_RELEASE 不输出调试信息● 默认为不输出调试信息
<i>permission</i>	文件属性参数，默认为0666

返回值:

<i>true</i>	文件输出成功
<i>false</i>	失败

该类的文档由以下文件生成:

- [waTemplate.h](#)
- [waTemplate.cpp](#)

webapp::TextFile类 参考

固定分隔符文本文件读取解析类

```
#include <waTextFile.h>
```

webapp::TextFile 的协作图:

webapp::TextFile
<ul style="list-style-type: none"> + TextFile() + TextFile() + ~TextFile() + open() + close() + next_line() + next_fields()

Public 成员函数

- [TextFile](#) ()
默认构造函数
- [TextFile](#) (const string &file)
参数为文本文件名的构造函数
- [~TextFile](#) ()
析构函数
- bool [open](#) (const string &file)
打开文本文件
- void [close](#) ()
关闭文本文件
- bool [next_line](#) (string &line)
读取下一行
- bool [next_fields](#) (vector< [String](#) > &fields, const string &split="\t", const int limit=0)
读取下一行并按分隔符拆分字段

详细描述

固定分隔符文本文件读取解析类

构造及析构函数说明

webapp::TextFile::TextFile () [inline]

默认构造函数

webapp::TextFile::TextFile (const string & file) [inline]

参数为文本文件名的构造函数

webapp::TextFile::~~TextFile () [inline]

析构函数

成员函数说明

bool webapp::TextFile::open (const string & file)

打开文本文件

参数:

<i>file</i>	文本文件路径名
-------------	---------

返回值:

<i>true</i>	打开文件成功
<i>false</i>	打开文件失败

void webapp::TextFile::close ()

关闭文本文件

参考自 webapp::ConfigFile::load().

这是这个函数的调用关系图:



bool webapp::TextFile::next_line (string & line)

读取下一行

参数:

<i>line</i>	读取到的文本行，不含末尾的'\n'
-------------	-------------------

返回值:

<i>true</i>	还未读到文件末尾
<i>false</i>	已读到文件末尾

参考自 webapp::ConfigFile::load().

这是这个函数的调用关系图:



`bool webapp::TextFile::next_fields (vector< String > & fields, const string & split = "\t", const int limit = 0)`

读取下一行并按分隔符拆分字段

参数:

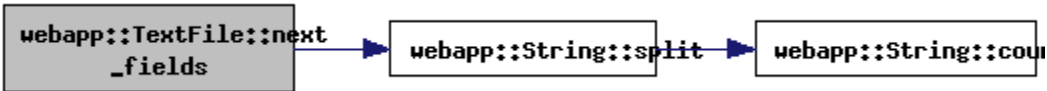
<i>fields</i>	读取到的字符串数组
<i>split</i>	字段切分分隔符，默认为'\t'
<i>limit</i>	字段切分次数限制，默认为0即不限制

返回值:

<i>true</i>	还未读到文件末尾
<i>false</i>	已读到文件末尾

参考 `webapp::String::split()`.

函数调用图:



该类的文档由以下文件生成:

- [waTextFile.h](#)
- [waTextFile.cpp](#)

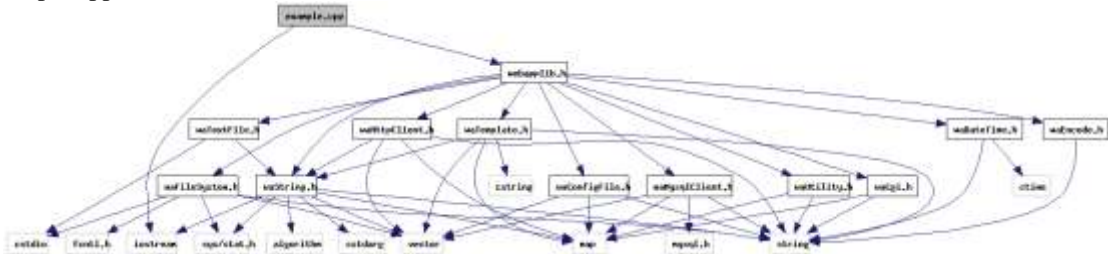
文件说明

example.cpp 文件参考

代码示例文件，演示一个简单CGI流程

```
#include <iostream>
#include "webapplib.h"
```

example.cpp 的引用(Include)关系图:



详细描述

代码示例文件，演示一个简单CGI流程

Makefile 文件参考

Web Application Library Makefile.

详细描述

Web Application Library Makefile.

Makefile.example 文件参考

WebAppLib example Makefile.

详细描述

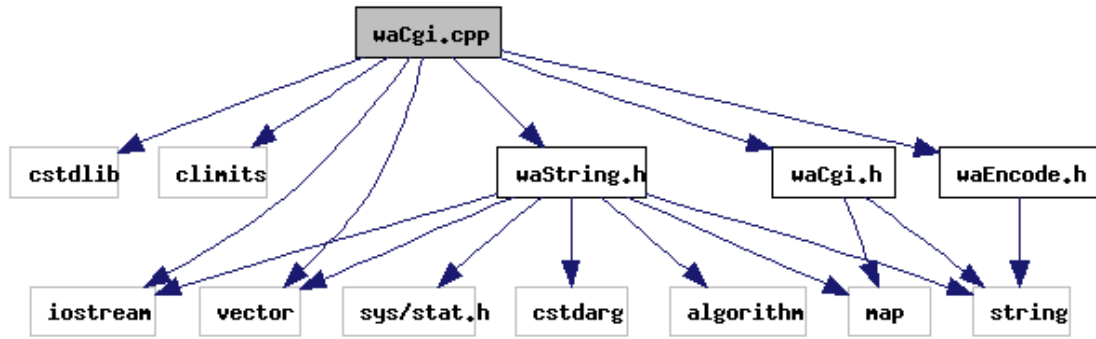
WebAppLib example Makefile.

waCgi.cpp 文件参考

Cgi, Cookie类实现文件

```
#include <cstdlib>
#include <climits>
#include <iostream>
#include <vector>
#include "waString.h"
#include "waEncode.h"
#include "waCgi.h"
```

waCgi.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- void [webapp::http_head\(\)](#)
输出HTML Content-Type header
- string [webapp::get_env\(\)](#) (const string &envname)
取得环境变量

详细描述

Cgi, Cookie类实现文件

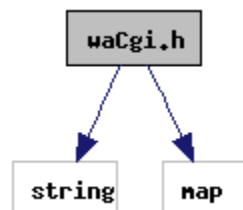
waCgi.h 文件参考

[webapp::Cgi](#), [webapp::Cookie](#)类头文件 依赖于 [webapp::String](#), [webapp::Encode](#)

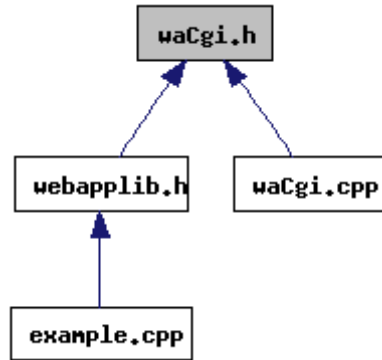
```
#include <string>
```

```
#include <map>
```

waCgi.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::Cgi](#)
- CGI参数读取类 class [webapp::Cookie](#)

Cookie读取,设置类 命名空间

- namespace [webapp](#)

Web Application Library namespace. 类型定义

- typedef map< string, string > [webapp::CgiList](#)
[Cgi](#) 参数值列表类型 (map<string,string>)
- typedef map< string, string > [webapp::CookieList](#)
[Cookie](#) 参数值列表类型 (map<string,string>)

函数

- void [webapp::http_head](#) ()
输出HTML Content-Type header
- string [webapp::get_env](#) (const string &envname)
取得环境变量

详细描述

[webapp::Cgi](#),webapp::Cookie类头文件 依赖于 [webapp::String](#), webapp::Encode

waConfigFile.cpp 文件参考

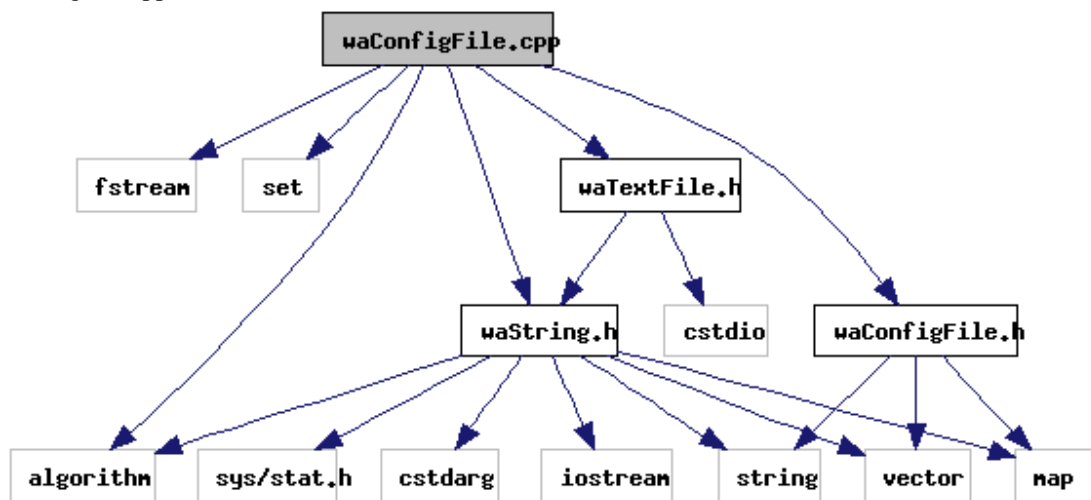
INI格式配置文件解析类实现文件

```

#include <fstream>
#include <set>
#include <algorithm>

```

```
#include "waString.h"
#include "waTextFile.h"
#include "waConfigFile.h"
waConfigFile.cpp 的引用(Include)关系图:
```



命名空间

- namespace [webapp](#)
Web Application Library namespace.

详细描述

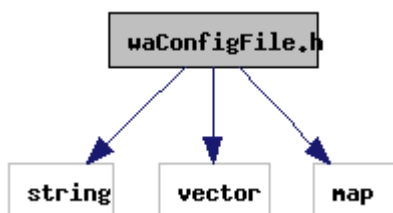
INI格式配置文件解析类实现文件

waConfigFile.h 文件参考

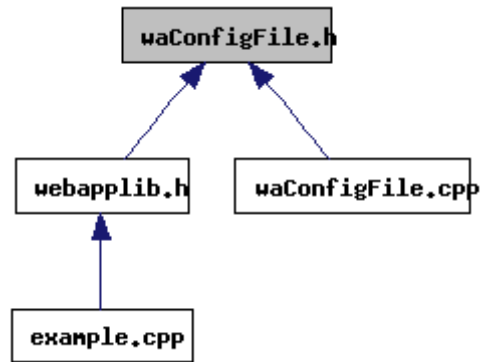
INI格式配置文件解析类头文件 依赖于 [webapp::String](#), [webapp::TextFile](#).

```
#include <string>
#include <vector>
#include <map>
```

waConfigFile.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::ConfigFile](#)

INI格式配置文件解析类 命名空间

- namespace [webapp](#)
Web Application Library namespace.

详细描述

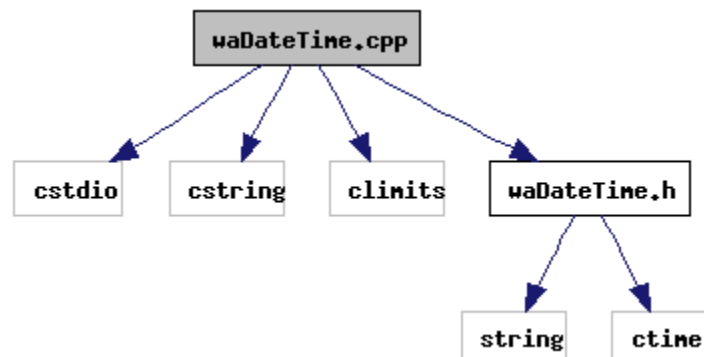
INI格式配置文件解析类头文件 依赖于 [webapp::String](#), [webapp::TextFile](#).

waDateTime.cpp 文件参考

`webapp::DateTime`类实现文件

```
#include <cstdio>
#include <cstring>
#include <climits>
#include "waDateTime.h"
```

`waDateTime.cpp` 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- DateTime [webapp::operator+](#) (const DateTime &date1, const DateTime &date2)
相加操作
- DateTime [webapp::operator+](#) (const DateTime &date, const time_t &tt)
相加操作
- DateTime [webapp::operator-](#) (const DateTime &date1, const DateTime &date2)
相减操作
- DateTime [webapp::operator-](#) (const DateTime &date, const time_t &tt)
相减操作

详细描述

webapp::DateTime类实现文件

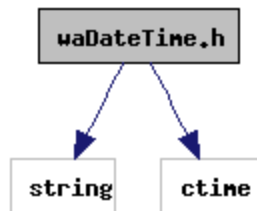
waDateTime.h 文件参考

webapp::DateTime类头文件 日期时间运算

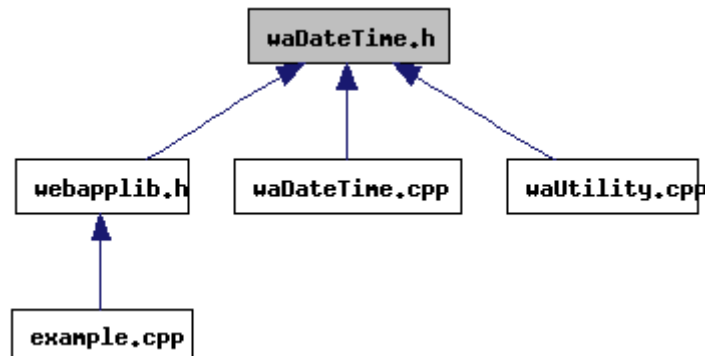
```
#include <string>
```

```
#include <ctime>
```

waDateTime.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::DateTime](#)

DateTime日期时间运算类 命名空间

- namespace [webapp](#)

Web Application Library namespace. 宏定义

- #define [TIME_ONE_SEC](#) 1
时长定义一秒钟
- #define [TIME_ONE_MIN](#) 60
时长定义一分钟
- #define [TIME_ONE_HOUR](#) 3600
时长定义一小时
- #define [TIME_ONE_DAY](#) 86400
时长定义一天
- #define [TIME_ONE_WEEK](#) 604800
时长定义一周

函数

- DateTime [webapp::operator+](#) (const DateTime &date1, const DateTime &date2)
相加操作
- DateTime [webapp::operator+](#) (const DateTime &date, const time_t &tt)
相加操作
- DateTime [webapp::operator-](#) (const DateTime &date1, const DateTime &date2)
相减操作
- DateTime [webapp::operator-](#) (const DateTime &date, const time_t &tt)
相减操作
- bool [webapp::operator==](#) (const DateTime &left, const DateTime &right)
时间相等比较
- bool [webapp::operator==](#) (const DateTime &left, const time_t &right)
时间相等比较
- bool [webapp::operator!=](#) (const DateTime &left, const DateTime &right)
时间不相等比较
- bool [webapp::operator!=](#) (const DateTime &left, const time_t &right)
时间不相等比较
- bool [webapp::operator>](#) (const DateTime &left, const DateTime &right)
时间大于比较
- bool [webapp::operator>](#) (const DateTime &left, const time_t &right)
时间大于比较
- bool [webapp::operator<](#) (const DateTime &left, const DateTime &right)
时间小于比较
- bool [webapp::operator<](#) (const DateTime &left, const time_t &right)

时间小于比较

- bool [webapp::operator>=](#) (const DateTime &left, const DateTime &right)
时间不小于比较
- bool [webapp::operator>=](#) (const DateTime &left, const time_t &right)
时间不小于比较
- bool [webapp::operator<=](#) (const DateTime &left, const DateTime &right)
时间不大于比较
- bool [webapp::operator<=](#) (const DateTime &left, const time_t &right)
时间不大于比较

详细描述

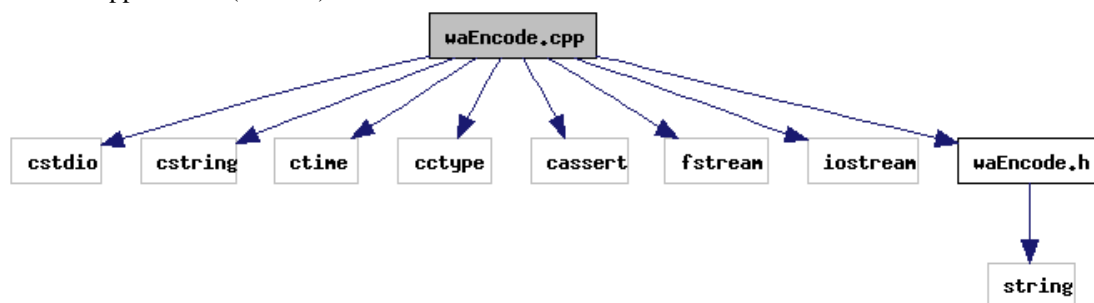
webapp::DateTime类头文件 日期时间运算

waEncode.cpp 文件参考

字符串BASE64、URI、MD5编码函数实现文件

```
#include <cstdio>
#include <cstring>
#include <ctime>
#include <cctype>
#include <cassert>
#include <fstream>
#include <iostream>
#include "waEncode.h"
```

waEncode.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- string [webapp::uri_encode](#) (const string &source)

URI编码

- string [webapp::uri_decode](#) (const string &source)

URI解码

- string [webapp::base64_encode](#) (const string &source)

字符串MIME BASE64 编码

- string [webapp::base64_decode](#) (const string &source)

字符串MIME BASE64 解码

- string [webapp::md5_encode](#) (const string &source)

MD5编码

详细描述

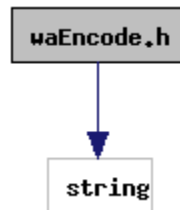
字符串BASE64、URI、MD5编码函数实现文件

waEncode.h 文件参考

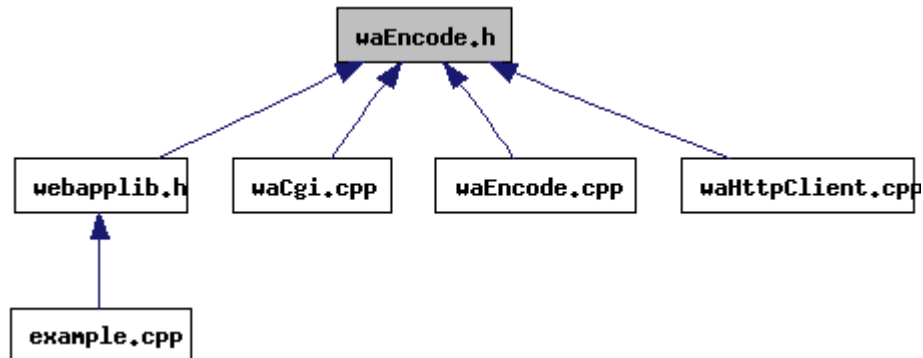
编码,加解密函数头文件 字符串BASE64、URI、MD5编码函数

```
#include <string>
```

waEncode.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- string [webapp::uri_encode](#) (const string &source)
URI 编码
- string [webapp::uri_decode](#) (const string &source)
URI 解码
- string [webapp::base64_encode](#) (const string &source)
字符串MIME BASE64 编码
- string [webapp::base64_decode](#) (const string &source)
字符串MIME BASE64 解码
- string [webapp::md5_encode](#) (const string &source)
MD5 编码

详细描述

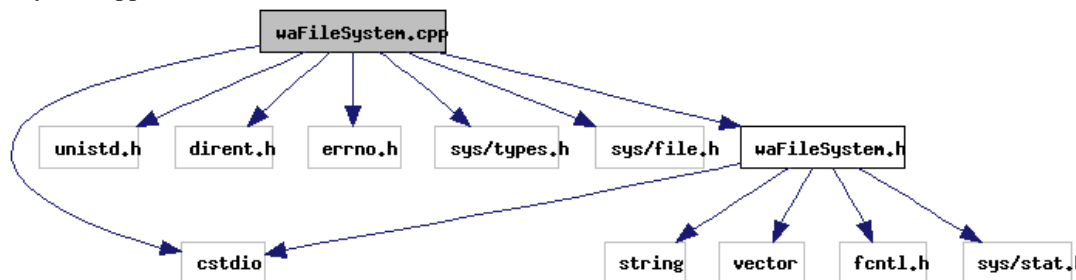
编码,加解密函数头文件 字符串BASE64、URI、MD5编码函数

waFileSystem.cpp 文件参考

文件操作函数实现文件

```
#include <cstdio>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/file.h>
#include "waFileSystem.h"
```

waFileSystem.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- bool [webapp::file_exist](#) (const string &file)

文件或者目录是否存在

- bool [webapp::is_link](#) (const string &file)
文件是否为链接
- bool [webapp::is_dir](#) (const string &file)
是否为目录
- bool [webapp::make_link](#) (const string &srcfile, const string &destfile)
建立链接
- long [webapp::file_size](#) (const string &file)
取得文件大小
- long [webapp::file_time](#) (const string &file)
取得文件更改时间
- string [webapp::file_path](#) (const string &file)
取得文件路径
- string [webapp::file_name](#) (const string &file)
取得文件名称
- bool [webapp::rename_file](#) (const string &oldname, const string &newname)
文件或者目录改名
- bool [webapp::copy_file](#) (const string &srcfile, const string &destfile)
拷贝文件
- bool [webapp::delete_file](#) (const string &file)
删除文件
- bool [webapp::move_file](#) (const string &srcfile, const string &destfile)
移动文件
- vector< string > [webapp::dir_files](#) (const string &dir)
返回目录文件列表
- bool [webapp::make_dir](#) (const string &dir, const mode_t mode=S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH)
建立目录
- bool [webapp::copy_dir](#) (const string &srcdir, const string &destdir)
拷贝目录
- bool [webapp::delete_dir](#) (const string &dir)
删除目录
- bool [webapp::move_dir](#) (const string &srcdir, const string &destdir)
移动目录
- void [webapp::lock_file](#) (int fd, const int type)
文件句柄锁函数
- bool [webapp::is_locked](#) (int fd)
判断文件句柄锁
- FILE * [webapp::lock_open](#) (const string &file, const char *mode, const int type)
申请锁并打开文件

详细描述

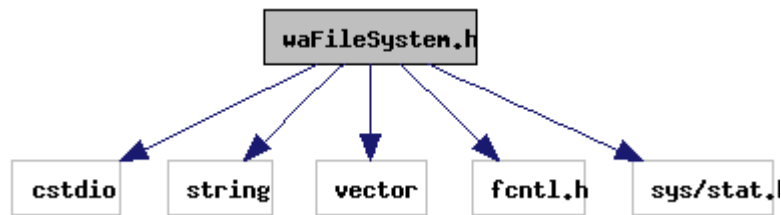
文件操作函数实现文件

waFileSystem.h 文件参考

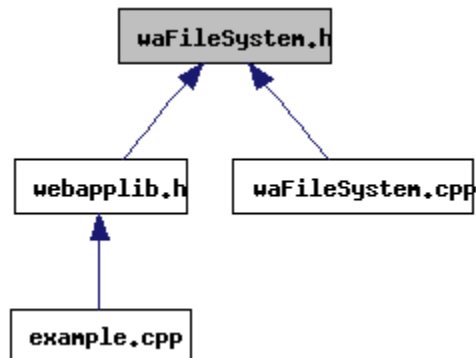
文件操作函数头文件 常用文件操作

```
#include <cstdio>
#include <string>
#include <vector>
#include <fcntl.h>
#include <sys/stat.h>
```

waFileSystem.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- bool [webapp::file_exist](#) (const string &file)
文件或者目录是否存在
- bool [webapp::is_link](#) (const string &file)
文件是否为链接
- bool [webapp::is_dir](#) (const string &file)
是否为目录
- bool [webapp::make_link](#) (const string &srcfile, const string &destfile)
建立链接

- long [webapp::file_size](#) (const string &file)
取得文件大小
- long [webapp::file_time](#) (const string &file)
取得文件更改时间
- string [webapp::file_path](#) (const string &file)
取得文件路径
- string [webapp::file_name](#) (const string &file)
取得文件名称
- bool [webapp::rename_file](#) (const string &oldname, const string &newname)
文件或者目录改名
- bool [webapp::copy_file](#) (const string &srcfile, const string &destfile)
拷贝文件
- bool [webapp::delete_file](#) (const string &file)
删除文件
- bool [webapp::move_file](#) (const string &srcfile, const string &destfile)
移动文件
- vector< string > [webapp::dir_files](#) (const string &dir)
返回目录文件列表
- bool [webapp::make_dir](#) (const string &dir, const mode_t mode=S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH)
建立目录
- bool [webapp::copy_dir](#) (const string &srcdir, const string &destdir)
拷贝目录
- bool [webapp::delete_dir](#) (const string &dir)
删除目录
- bool [webapp::move_dir](#) (const string &srcdir, const string &destdir)
移动目录
- void [webapp::lock_file](#) (int fd, const int type)
文件句柄锁函数
- void [webapp::lock_file](#) (FILE *fp, const int type)
文件句柄锁函数
- bool [webapp::is_locked](#) (int fd)
判断文件句柄锁
- bool [webapp::is_locked](#) (FILE *fp)
判断文件句柄锁
- FILE * [webapp::lock_open](#) (const string &file, const char *mode, const int type)
申请锁并打开文件

详细描述

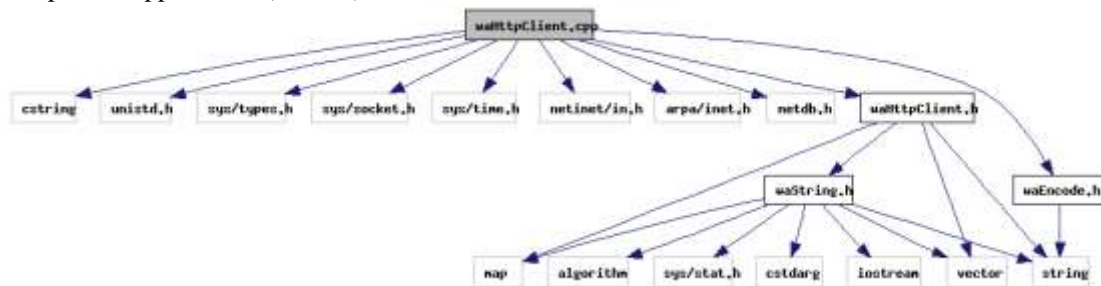
文件操作函数头文件 常用文件操作

waHttpClient.cpp 文件参考

HTTP客户端类实现文件

```
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include "waEncode.h"
#include "waHttpClient.h"
```

waHttpClient.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- int [webapp::tcp_request](#) (const string &server, const int port, const string &request, string &response, const int timeout)
发送TCP请求并取得回应内容
- string [webapp::gethost_byname](#) (const string &domain)
根据服务器域名取得IP
- bool [webapp::isip](#) (const string &ipstr)
判断字符串是否为有效IP

详细描述

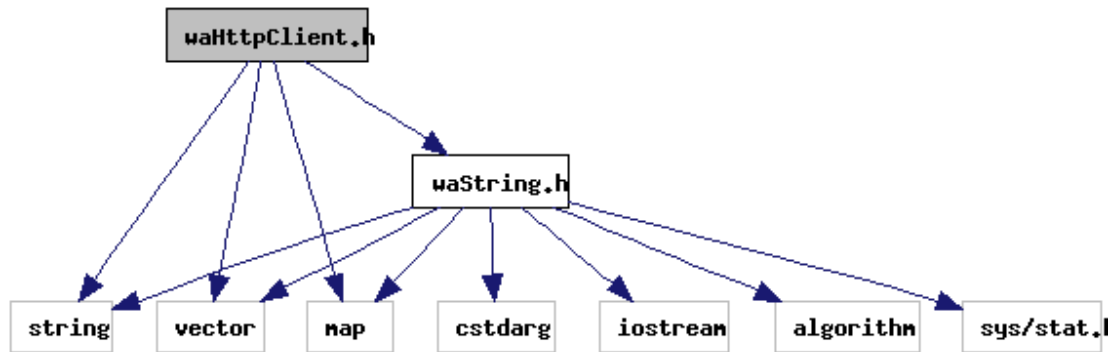
HTTP客户端类实现文件

waHttpClient.h 文件参考

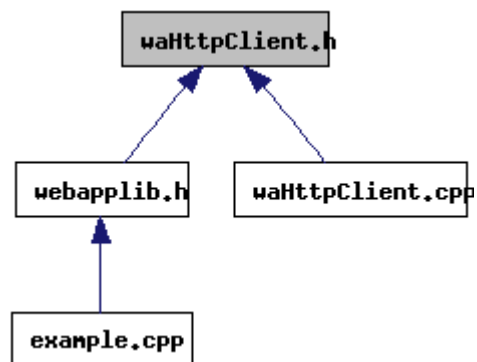
HTTP客户端类头文件 依赖于 [webapp::String](#), [webapp::Encode](#) [使用说明文档及简单范例](#)

```
#include <string>
#include <vector>
#include <map>
#include "waString.h"
```

waHttpClient.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::HttpClient](#)

HTTP客户端类 [使用说明文档及简单范例](#) 命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- int [webapp::tcp_request](#) (const string &server, const int port, const string &request, string &response, const int timeout)
发送TCP请求并取得回应内容
- string [webapp::gethost_byname](#) (const string &domain)
根据服务器域名取得IP
- bool [webapp::isip](#) (const string &ipstr)
判断字符串是否为有效IP

详细描述

HTTP客户端类头文件 依赖于 [webapp::String](#), [webapp::Encode](#) [使用说明文档及简单范例](#)

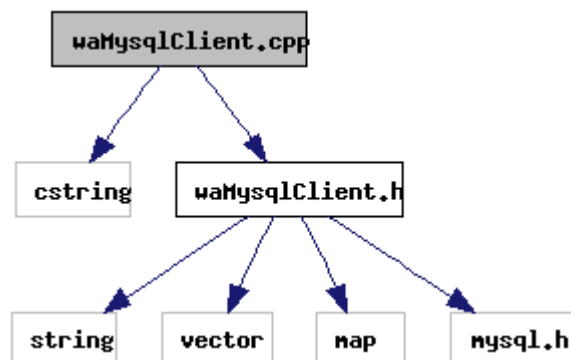
waMysqlClient.cpp 文件参考

[webapp::MysqlClient](#), [webapp::MysqlData](#)类实现文件

```
#include <cstring>
```

```
#include "waMysqlClient.h"
```

waMysqlClient.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- string [webapp::escape_sql](#) (const string &str)
SQL 语句字符转义

详细描述

[webapp::MysqlClient](#), [webapp::MysqlData](#)类实现文件

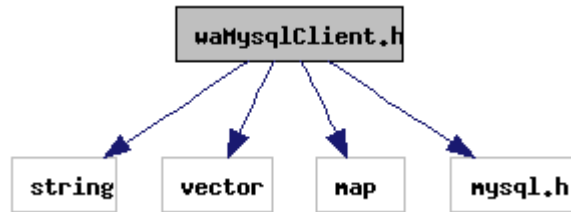
waMysqlClient.h 文件参考

[webapp::MysqlClient](#), [webapp::MysqlData](#)类头文件 MySQL数据库C++接口

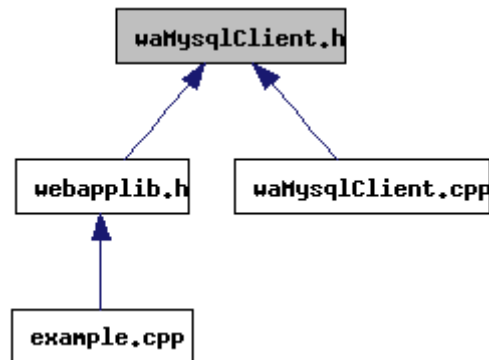
```
#include <string>
```

```
#include <vector>
```

```
#include <map>
#include <mysql.h>
waMysqlClient.h 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::MysqlData](#)
- MySQL数据集类 class [webapp::MysqlClient](#)

MySQL数据库连接类 命名空间

- namespace [webapp](#)

Web Application Library namespace. 类型定义

- typedef map< string, string > [webapp::MysqlDataRow](#)
[MysqlData](#) 数据行类型 (map<string,string>)

函数

- string [webapp::escape_sql](#) (const string &str)
SQL语句字符转义

详细描述

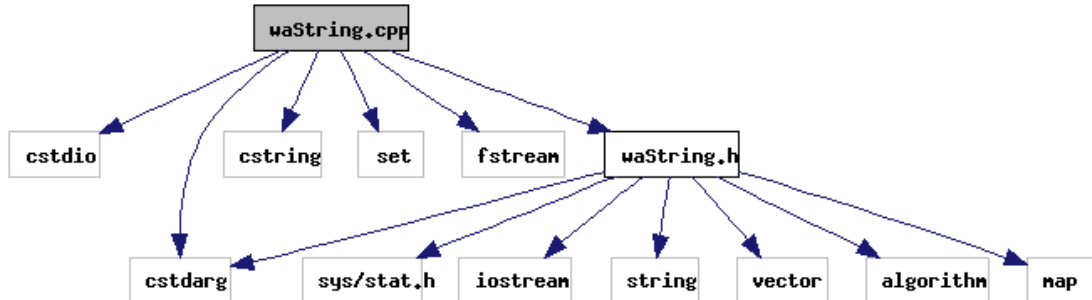
webapp::mysqlClient,webapp::MysqlData类头文件 MySQL数据库C++接口

waString.cpp 文件参考

webapp::String类实现文件

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <set>
#include <fstream>
#include "waString.h"
```

waString.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- string [webapp::itos](#) (const long i, const ios::fmtflags base=ios::dec)
long int 转换为string
- long [webapp::stoi](#) (const string &s, const ios::fmtflags base=ios::dec)
string 转换为int
- string [webapp::ftos](#) (const double f, const int ndigit=2)
double 转换为string
- double [webapp::stof](#) (const string &s)
string 转换为double
- bool [webapp::isgbk](#) (const unsigned char c1, const unsigned char c2)
判断一个双字节字符是否是GBK编码汉字
- string [webapp::va_sprintf](#) (va_list ap, const string &format)
可变参数字符串格式化, 与va_start()、va_end()宏配合使用
- string [webapp::va_str](#) (const char *format,...)
格式化字符串并返回

详细描述

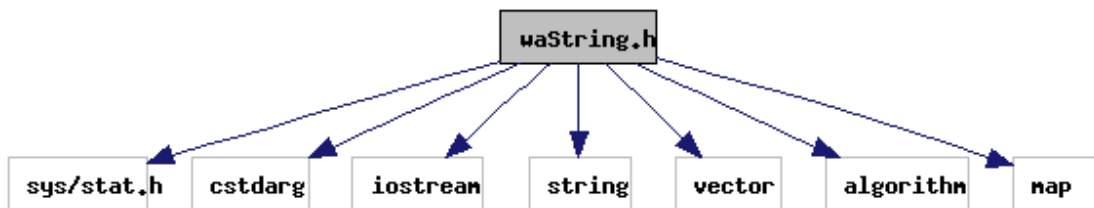
webapp::String类实现文件

waString.h 文件参考

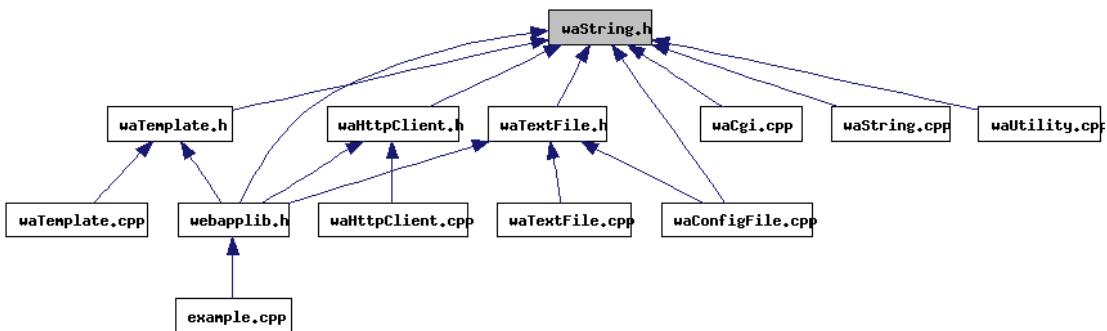
webapp::String类头文件 继承自string的字符串类 [基类string使用说明文档](#)

```
#include <sys/stat.h>
#include <cstdint>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <map>
```

waString.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::String](#)

继承自string的字符串类 [基类string使用说明文档](#) 命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- string [webapp::itos](#) (const long i, const ios::fmtflags base=ios::dec)
long int转换为string
- long [webapp::stoi](#) (const string &s, const ios::fmtflags base=ios::dec)
string转换为int
- string [webapp::ftos](#) (const double f, const int ndigit=2)
double转换为string
- double [webapp::stof](#) (const string &s)

string 转换为*double*

- bool [webapp::isgbk](#) (const unsigned char c1, const unsigned char c2)
判断一个双字节字符是否是GBK编码汉字
- string [webapp::va_sprintf](#) (va_list ap, const string &format)
可变参数字符串格式化, 与va_start(), va_end()宏配合使用
- string [webapp::va_str](#) (const char *format,...)
格式化字符串并返回

详细描述

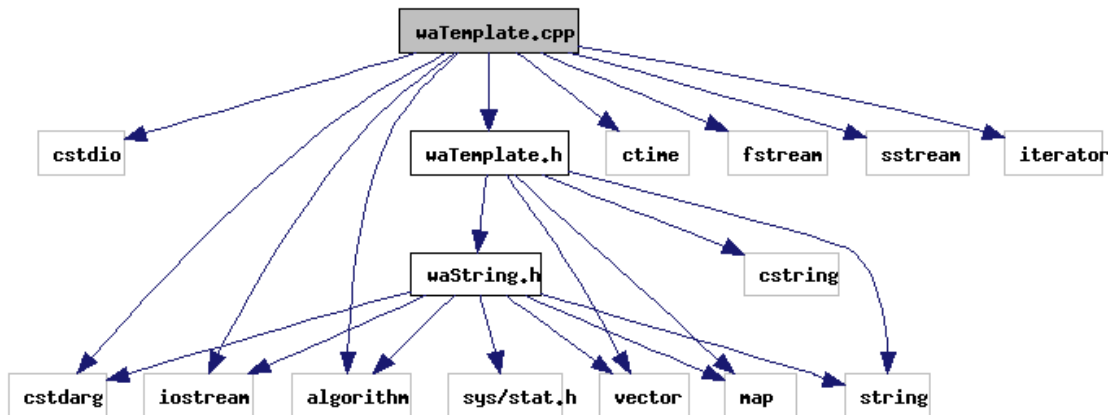
webapp::String类头文件 继承自string的字符串类 [基类string使用说明文档](#)

waTemplate.cpp 文件参考

HTML模板处理类实现文件

```
#include <cstdio>
#include <cstdarg>
#include <ctime>
#include <iostream>
#include <fstream>
#include <sstream>
#include <iterator>
#include <algorithm>
#include "waTemplate.h"
```

waTemplate.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)
Web Application Library namespace.

详细描述

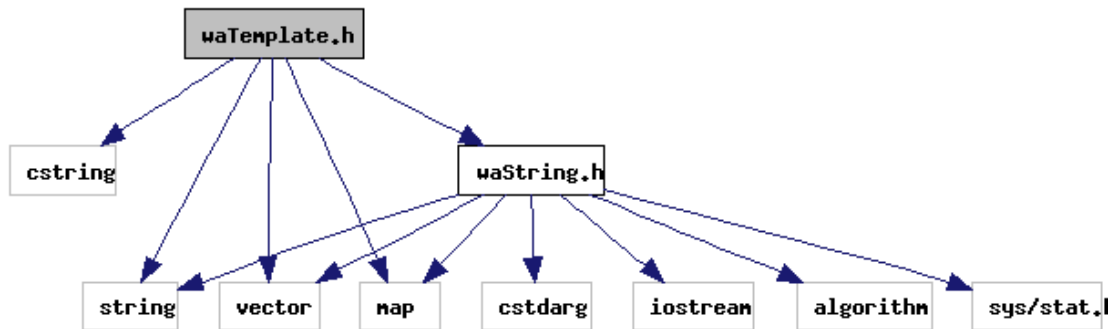
HTML模板处理类实现文件

waTemplate.h 文件参考

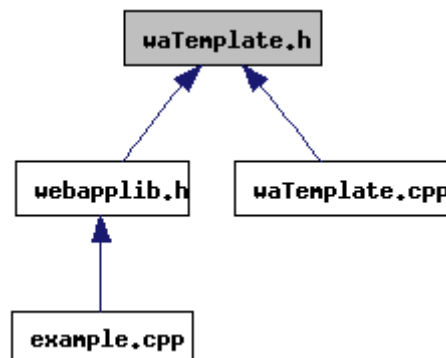
HTML模板处理类头文件 支持条件、循环脚本的HTML模板处理类 依赖于 waString [使用说明文档及简单范例](#)

```
#include <cstring>
#include <string>
#include <vector>
#include <map>
#include "waString.h"
```

waTemplate.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::Template](#)

支持条件、循环脚本的HTML模板处理类 [使用说明文档及简单范例](#) 命名空间

- namespace [webapp](#)

Web Application Library namespace.

详细描述

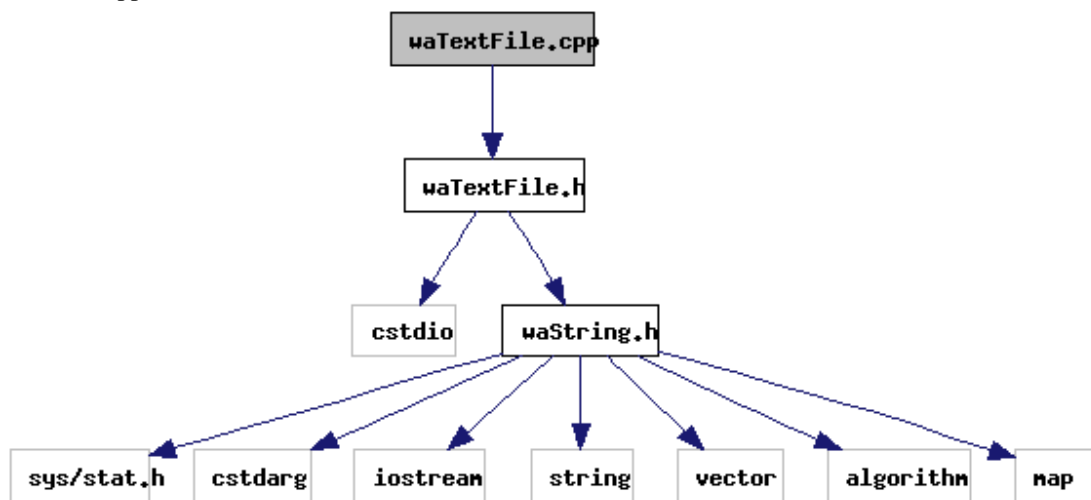
HTML模板处理类头文件 支持条件、循环脚本的HTML模板处理类 依赖于 `waString` [使用说明文档及简单范例](#)

waTextFile.cpp 文件参考

固定分隔符文本文件读取解析类实现文件 读取解析固定分隔符文本文件

```
#include "waTextFile.h"
```

waTextFile.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace.

详细描述

固定分隔符文本文件读取解析类实现文件 读取解析固定分隔符文本文件

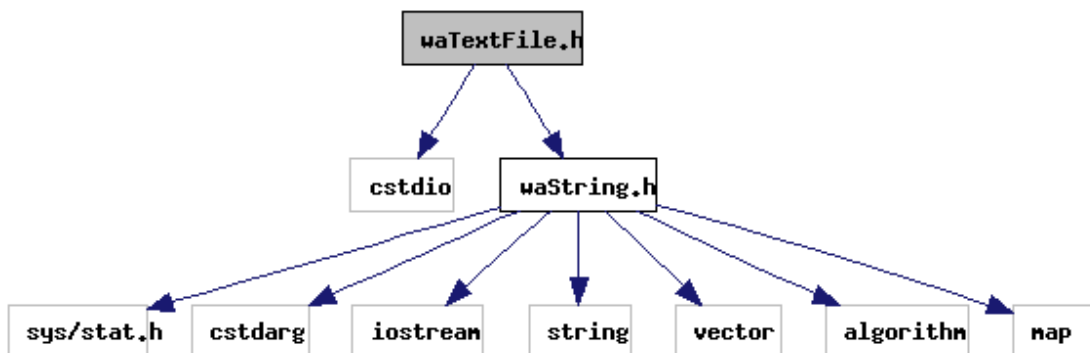
waTextFile.h 文件参考

固定分隔符文本文件读取解析类头文件 读取解析固定分隔符文本文件 依赖于 [webapp::String](#)

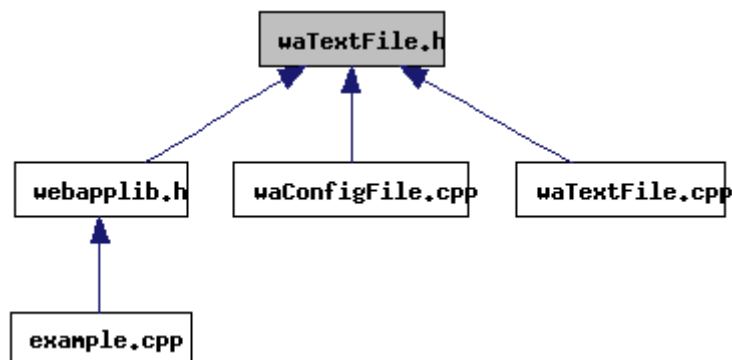
```
#include <cstdio>
```

```
#include "waString.h"
```

waTextFile.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class [webapp::TextFile](#)

固定分隔符文本文件读取解析类 命名空间

- namespace [webapp](#)
Web Application Library namespace.

详细描述

固定分隔符文本文件读取解析类头文件 读取解析固定分隔符文本文件 依赖于 [webapp::String](#)

waUtility.cpp 文件参考

系统调用工具函数实现文件

```

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cstdarg>

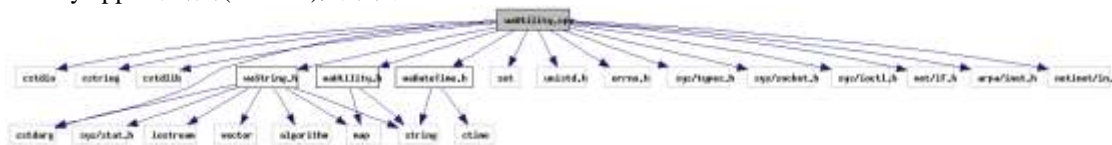
```

```

#include <set>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "waString.h"
#include "waDateTime.h"
#include "waUtility.h"

```

waUtility.cpp 的引用(Include)关系图:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 函数

- unsigned int [webapp::string_hash](#) (const string &str)
返回字符串HASH值, 基于DJB HASH算法
- string [webapp::replace_text](#) (const string &text, const map< string, string > &replace)
全文词表替换, 兼容GBK汉字
- string [webapp::extract_html](#) (const string &html)
提取HTML代码正文
- string [webapp::extract_text](#) (const string &text, const int option=[EXTRACT_ALL](#), const size_t len=0)
全角半角字符转换并提取正文
- void [webapp::file_logger](#) (const string &file, const char *format,...)
追加日志记录
- void [webapp::file_logger](#) (FILE *fp, const char *format,...)
追加日志记录
- string [webapp::system_command](#) (const string &cmd)
执行命令并返回命令输出结果
- string [webapp::host_addr](#) (const string &interface="eth0")
返回指定网卡设备绑定的IP地址

详细描述

系统调用工具函数实现文件

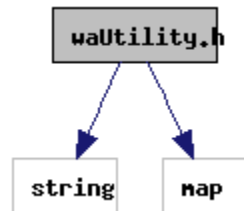
waUtility.h 文件参考

系统调用工具函数头文件 常用系统调用和工具函数 依赖于 [webapp::String](#), [webapp::DateTime](#)

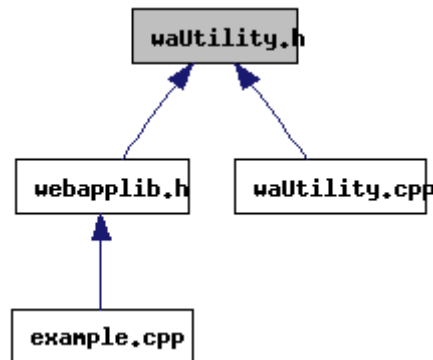
```
#include <string>
```

```
#include <map>
```

waUtility.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



命名空间

- namespace [webapp](#)

Web Application Library namespace. 宏定义

- #define [EXTRACT_ALL](#) (EXTRACT_ALPHA|EXTRACT_DIGIT|EXTRACT_PUNCT|EXTRACT_SPACE|EXTRACT_HTML)
提取正文函数过滤选项, 过滤全部 (字母、数字、标点、空白、HTML)

枚举

- enum [webapp::extract_option](#) { [webapp::EXTRACT_ALPHA](#) = 2, [webapp::EXTRACT_DIGIT](#) = 4, [webapp::EXTRACT_PUNCT](#) = 8, [webapp::EXTRACT_SPACE](#) = 16, [webapp::EXTRACT_HTML](#) = 32 }

函数

- unsigned int [webapp::string_hash](#) (const string &str)
返回字符串HASH值, 基于DJB HASH算法
- string [webapp::replace_text](#) (const string &text, const map< string, string > &replace)

全文词表替换，兼容GBK汉字

- string [webapp::extract_html](#) (const string &html)
提取HTML代码正文
- string [webapp::extract_text](#) (const string &text, const int option=[EXTRACT_ALL](#), const size_t len=0)
全角半角字符转换并提取正文
- void [webapp::file_logger](#) (const string &file, const char *format,...)
追加日志记录
- void [webapp::file_logger](#) (FILE *fp, const char *format,...)
追加日志记录
- string [webapp::system_command](#) (const string &cmd)
执行命令并返回命令输出结果
- string [webapp::host_addr](#) (const string &interface="eth0")
返回指定网卡设备绑定的IP地址

详细描述

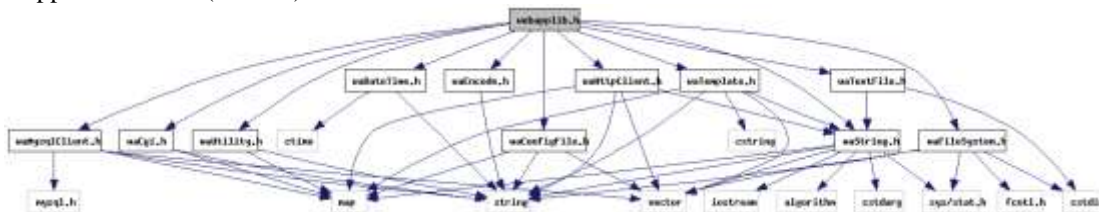
系统调用工具函数头文件 常用系统调用和工具函数 依赖于 [webapp::String](#), [webapp::DateTime](#)

webapplib.h 文件参考

开发库头文件集合

```
#include "waString.h"
#include "waCgi.h"
#include "waMySQLClient.h"
#include "waDateTime.h"
#include "waTemplate.h"
#include "waHttpClient.h"
#include "waEncode.h"
#include "waFileSystem.h"
#include "waUtility.h"
#include "waTextFile.h"
#include "waConfigFile.h"
```

webapplib.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:

