

Please note that GitHub no longer supports your web browser.
We recommend upgrading to the latest [Google Chrome](#) or [Firefox](#).


Learn more

Ignore

programthink / **opensource**

Branch: master opensource / libs / **cpp.wiki**

Find fileCopy path

 **programthink** 统一示例风格 32983f6 Feb 25, 2017

1 contributor

3541 lines (2253 sloc) 96.4 KB

C/C++ 开源库及示例代码

Table of Contents

- 说明
- 1** 综合性的库
- 2** 数据结构 & 算法
 - 2.1 容器
 - 2.1.1 标准容器
 - 2.1.2 Lockfree 的容器
 - 2.1.3 环形缓冲
 - 2.1.4 多维数组
 - 2.1.5 图
 - 2.2 对容器的操作
 - 2.3 字符串处理
 - 2.3.1 字符集
 - 2.3.2 字符串格式化
 - 2.3.3 正则表达式
 - 2.3.4 （其它）
 - 2.4 内存相关
 - 2.4.1 智能指针
 - 2.4.2 内存池
 - 2.5 时间 & 日期
 - 2.6 编码 & 解码
 - 2.6.1 序列化
 - 2.6.2 Base64
 - 2.7 （其它）
 - 2.7.1 随机数
 - 2.7.2 UUID
- 3** 编程范式
 - 3.1 面向对象编程（OOP）
 - 3.2 泛型编程（GP）
 - 3.3 函数式编程（FP）
 - 3.4 元编程（Metaprogramming）
- 4** 调试 & 测试
 - 4.1 调试日志
 - 4.2 单元测试
 - 4.3 健壮性测试
 - 4.4 性能测试

5 操作系统

- └ 5.1 跨操作系统
 - └ 5.1.1 文件系统
 - └ 5.1.2 线程
 - └ 5.1.3 进程
 - └ 5.1.4 本地进程间通信 (IPC)
- └ 5.2 特定于 Windows 系统
 - └ 5.2.1 注册表
 - └ 5.2.2 Windows 服务 (Service)
- └ 5.3 特定于 Linux & Unix 系统

6 图形用户界面 (GUI)

- └ 6.1 GUI 框架
 - └ 6.1.1 跨平台的 GUI 框架
 - └ 6.1.2 特定于 Windows 的 GUI 框架
- └ 6.2 图表 (Chart)

7 文本用户界面 (TUI)

- └ 7.1 命令行参数
- └ 7.2 文本终端

8 网络

- └ 8.1 链路层 & 网络层
- └ 8.2 传输层
- └ 8.3 标准的应用层
 - └ 8.3.1 综合性的库
 - └ 8.3.2 HTTP
 - └ 8.3.3 DNS
 - └ 8.3.4 XMPP
- └ 8.4 自定义的应用层
- └ 8.5 网络库、框架、中间件

9 数据库

- └ 9.1 开源数据库
- └ 9.2 数据库 API 的封装库
 - └ 9.2.1 综合性的封装库
 - └ 9.2.2 MySQL 封装库
 - └ 9.2.3 PostgreSQL 封装库
 - └ 9.2.4 Oracle 封装库
 - └ 9.2.5 DB2 封装库
 - └ 9.2.6 SQLite 封装库
 - └ 9.2.7 Redis 封装库
 - └ 9.2.8 MongoDB 封装库
- └ 9.3 ODBC 相关
- └ 9.4 ORM (Object-Relational Mapping)

10 Web

- └ 10.1 HTTP Server
- └ 10.2 HTTP Client
- └ 10.3 浏览器引擎
- └ 10.4 浏览器整合
- └ 10.5 (其它)

11 信息安全

- └ 11.1 密码学

12 处理文件格式

- └ 12.1 结构化数据格式
 - └ 12.1.1 CSV
 - └ 12.1.2 JSON
 - └ 12.1.3 YAML
- └ 12.2 压缩文件 & 打包文件

- └ 12.2.1 综合性的库
- └ 12.2.2 zip
- └ 12.2.3 bzip2 (bz2)
- └ 12.2.4 gzip (gz)
- └ 12.2.5 tar
- └ 12.2.6 rar
- └ 12.2.7 snappy
- └ 12.2.8 Brotli
- └ 12.2.9 LZFSE
- └ 12.3 标记语言
 - └ 12.3.1 XML
 - └ 12.3.2 HTML
- └ 12.4 PDF
- └ 12.5 MS Office 文档
- └ 12.6 RTF
- └ 12.7 CHM
- 13 图像**
 - └ 13.1 图像处理
 - └ 13.2 图像格式转换
 - └ 13.3 图像渲染
 - └ 13.4 计算机视觉
- 14 多媒体**
 - └ 14.1 多媒体框架
 - └ 14.2 视频库
 - └ 14.3 音频库
- 15 游戏**
 - └ 15.1 综合性的游戏引擎
 - └ 15.2 3D 渲染引擎
 - └ 15.3 物理引擎
- 16 数值运算 & 科学计算**
 - └ 16.1 综合性的库
 - └ 16.2 有理数
 - └ 16.3 高精度数值运算
 - └ 16.4 矩阵
 - └ 16.5 线性代数
- 17 跨语言编程**
 - └ 17.1 整合多种语言的库
 - └ 17.2 整合单一语言的库
 - └ 17.2.1 整合 Python 语言
- 18 (其它)**
 - └ 18.1 词法分析 & 语法分析

说明

本页面汇总俺收集的各种 C 和 C++ 的开源代码库，不定期更新。

如果你发现本页面的开源库有错漏之处，非常欢迎给俺提供反馈——有 GitHub 帐号的同学，可以[给俺发 issue](#)；没帐号的同学，可以去[俺博客](#)留言。

1 综合性的库

Boost

Home: <http://boost.org/>

Links: [Wikipedia 维基百科](#)

Boost 大概是最重要的第三方 C++ 库。其作者有很多是 C++ 标准委员会的成员。Boost 的很多子库后来都成为 C++ 的标准库。

本页面的其它章节还会继续提及 Boost 在各种领域的应用。

wxWidgets

Home: <https://wxwidgets.org/>

Links: [Wikipedia 维基百科](#)

这是一个非常老牌的 C++ 开源 GUI 框架，诞生于1992年。原先叫做 wxWindows，后来因为微软的法律纠纷，改为现名。

它支持的操作系统平台很多（包括嵌入式系统）。

很多开源项目用到它，比如：BitTorrent、aMule、FileZilla、Code::Blocks、Dolphin.....

虽然它以 GUI 为主，但是也提供了其它一些辅助功能（比如：进程间通讯、网络、数据库、多媒体.....）

Qt

Home: <http://www.qt-project.org/>

Links: [Wikipedia 维基百科](#)

这是一个非常老牌的 C++ 开源 GUI 框架，于1995年发布 1.0 版本。原先由 Trolltech 公司维护，后来该公司被 Nokia 收购。

它支持的操作系统平台很多（包括嵌入式系统）。

虽然它以 GUI 为主，但是也提供了其它一些辅助功能（比如：网络、数据库、多媒体、3D引擎.....）

APR (Apache Portable Runtime)

Home: <https://apr.apache.org/>

Links: [Wikipedia 维基百科](#)

这是由 Apache 社区维护的 C 开源库，主要提供操作系统相关的功能（文件系统、进程、线程、用户、IPC）。此外还提供了一些网络相关的功能。

APR 原先是 Apache Web 服务器的一个组成部分，后来独立出来，成为一个单独的开源项目。

ACE (Adaptive Communication Environment)

Home: <http://www.cse.wustl.edu/~schmidt/ACE.html>

Links: [Wikipedia 维基百科](#)

这是一个跨平台的 C++ 库，提供了一套网络通讯的框架；另外还支持线程、进程和 IPC。

POCO

Home: <http://pocoproject.org/>

Links: [Wikipedia](#)

它的名称源自“POrtable COmponents”，是一个基于 C++ 的开源库。

它的功能以网络通讯为主，同时也提供一些其它功能（比如：多线程、进程间通讯、数据库、XML、JSON.....）

Dlib

Home: <http://dlib.net/>

Links: [Wikipedia](#)

诞生于2002年的 C++ 开源库，提供了非常多的功能（网络、多线程、GUI、数值计算、图像处理、数据挖掘.....）。

它还有一个特色是：同时提供了 Python API :)

Crypto++

Home: <http://www.cryptopp.com/>

Links: [Wikipedia](#)

它也叫“CryptoPP”或“libcrypto++”，是非常著名的开源加密库，诞生于1995年。基于 C++ 开发，大量用到模板语法。

虽然它以加密为主，但是也提供了其它一些辅助功能（比如：数据压缩、编码解码、计时器...）

2 数据结构 & 算法

2.1 容器

2.1.1 标准容器

std

C++ 98 标准内置的 STL 提供了如下容器：

- 数组：vector（动态数组）、valarray（针对数值类型特化的 vector）、bitset（储存比特的【固定】数组）
- 链表：list（双向）
- 队列：queue、deque（双端队列）
- 栈：stack
- 映射：map（键值无重复）、multimap（键值可重复）
- 集合：set（元素无重复）、multiset（元素可重复）

C++ 11 标准新增了如下容器：

- 数组：array（相比 vector，它的 size 是编译时【固定】的）
- 链表：forward_list（相比 list，它是【单向】的）
- 映射：unordered_map、unordered_multimap（相比 map 和 multimap，这俩采用 hash 实现）
- 集合：unordered_set、unordered_multiset（相比 set 和 multiset，这俩采用 hash 实现）

下面几个容器，C++ 标准【没有】包含，但包含在某些知名的 STL 第三方库中（比如 SGI 的 STL）：

- 映射：hash_map、hash_multimap（与 unordered_map、unordered_multimap 相同）
- 集合：hash_set、hash_multiset（与 unordered_set、unordered_multiset 相同）

2.1.2 Lockfree 的容器

（“lock-free”翻译成“锁无关”会引发歧义，所以俺直接用洋文）

Boost.Lockfree

Docs: <http://boost.org/libs/lockfree>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了三种 lock-free 的容器（queue、stack、spsc_queue）。最后这种是“环形缓冲”。

libcds（Concurrent Data Structures）

Home: <http://libcds.sourceforge.net/>

这是一个跨平台的 C++ 开源库，提供了若干 lock-free 的容器。它的 2.0.0 版本，代码重写以支持 C++ 11 标准。

2.1.3 环形缓冲

Boost.CircularBuffer

Docs: http://boost.org/libs/circular_buffer

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“环形缓冲区”的模板。

“环形缓冲区”可以降低内存分配的开销。俺曾经写过一篇博文推荐环形缓冲区（在“[这里](#)”）。

代码示例

```
#include <boost/circular_buffer.hpp>

boost::circular_buffer<int> cb(3); // Create with capacity for 3 integers
cb.push_back(1);
cb.push_back(2);
cb.push_back(3);

// The buffer is full now
// pushing subsequent elements will overwrite front-most elements.

cb.push_back(4); // Overwrite 1 with 4.
cb.push_back(5); // Overwrite 2 with 5.
// The buffer now contains 3, 4 and 5.

// Elements can be popped from either the front or the back.
cb.pop_back(); // 5 is removed.
cb.pop_front(); // 3 is removed.

// Leaving only one element with value = 4.
assert(cb[0] == 4);
```

2.1.4 多维数组

Boost.MultiArray

Docs: http://boost.org/libs/multi_array

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了任意维的数组。

代码示例——3维数组

```
#include <boost/multi_array.hpp>

typedef boost::multi_array<double, 3> array_type;
typedef array_type::index index;
array_type A(boost::extents[3][4][2]);

int values = 0;
for(index i=0; i!=3; ++i)
    for(index j=0; j!=4; ++j)
        for(index k=0; k!=2; ++k)
            A[i][j][k] = values++;
```

2.1.5 图

Boost.Graph

Docs: <http://boost.org/libs/graph>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，简称 BGL（Boost Graph Library），封装了“图”这种数据结构。

它提供了与 STL 类似的泛型编程风格。

Dlib

Docs: http://dlib.net/graph_tools.html

Dlib 前面已经介绍过。它提供了2个类（graph & directed_graph）封装“图”。

2.2 对容器的操作

(STL 标准库里面已经实现了很多算法用来操作容器，考虑到本页面已经很长，这里只列举第三方库实现的算法)

Boost.Foreach

Docs: <http://boost.org/libs/foreach>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了通用的遍历，其效果类似于 Python 的 for 循环语法。

有了它，你在遍历时无需声明迭代器变量，也无需关心遍历的容器是何种类型。

代码示例——遍历字符串

```
#include <string>
#include <iostream>
#include <boost/foreach.hpp>

std::string test("Hello, world!"); // string 可以视为 char 的容器
BOOST_FOREACH(char ch, test)
{
    std::cout << ch;
}
```

2.3 字符串处理

2.3.1 字符集

Boost.Locale

Docs: <http://boost.org/libs/locale>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了各种“本地化”的功能。其中就包括字符集编码转换。

代码示例

```
#include <fstream>
#include <boost/locale.hpp>

using namespace boost::locale;
std::locale loc = generator().generate("he_IL.UTF-8");
std::wofstream file;
file.imbue(loc);
file.open("hello.txt");
file << L"הי!";
```

POCO.Text

Docs: <http://pocoproject.org/docs/package-Foundation.Text.html>

POCO 前面已经介绍过。它提供了 UTF8/UTF16/UTF32 的转换。

2.3.2 字符串格式化

Boost.Format

Docs: <http://boost.org/libs/format>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“格式化字符串”的功能。相比 ANSI C 的 sprintf() 和 snprintf()，它的格式化功能更强并且更安全。

代码示例

```
#include <iostream>
#include <boost/format.hpp>

using std::cout;
using boost::format;
```

```
// 基于“位置指示符”的格式串语法
cout << format("%1% %2% %3% %2% %1%") % "11" % "22" % "333";
// 输出如下:
// 11 22 333 22 11

// 兼容 printf() 的格式串语法
cout << format("%s %d") % "price" % 1234;
// 输出如下:
// price 1234
```

fmt

Home: <https://github.com/fmtlib/fmt>

这是一个轻量级、类型安全、高性能的字符串格式化库。它也可以用来替代 C++ 标准库中的 `IOStreams`。

代码示例

```
#include <string>
#include <fmt/format.h>

// 使用 Python 的格式化语法
fmt::print("Hello, {}!", "world");
// 使用 printf 的格式化语法
fmt::printf("Hello, %s!", "world");
// 使用序号参数,
std::string s = fmt::format("{} {1} {}", "Hello", "world");
// 使用命名参数
fmt::print("Hello, {name}! The answer is {number}. Goodbye, {name}.",
    fmt::arg("name", "World"), fmt::arg("number", 42));
```

2.3.3 正则表达式

PCRE (Perl Compatible Regular Expressions)

Home: <http://www.pcre.org/>

Links: [Wikipedia](#)

这是一个很老牌的正则表达式的库，诞生于1997年。很多知名的开源项目（Apache、PHP、KDE）用到了它。

Boost.Regex

Docs: <http://boost.org/libs/regex>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“正则式”的功能。

注：Boost 的这个子库已经进入 C++ 11 标准。

代码示例——基于正则式进行匹配和替换

```
#include <boost/regex.hpp>

using std::string;
using namespace boost;

// 正则式匹配的例子
bool validate_card_format(const string& s)
{
    const regex e("(\\d{4}[- ]){3}\\d{4}");
    return regex_match(s, e);
}

// 正则式替换的例子
const regex e("(\\A(\\d{3,4})[- ]?(\\d{4})[- ]?(\\d{4})[- ]?(\\d{4})\\z)");
const string machine_format("\\1\\2\\3\\4");
const string human_format("\\1-\\2-\\3-\\4");

string machine_readable_card_number(const string& s)
{

```



```

    return regex_replace(s, e, machine_format, match_default|format_sed);
}

string human_readable_card_number(const string& s)
{
    return regex_replace(s, e, human_format, match_default|format_sed);
}

```

re2

Home: <https://github.com/google/re2>

这是 Google 提供的正则式库，基于 C++。其 API 使用起来很简洁。

有多种脚本语言（Python、Ruby、Perl、Nodejs、Erlang、OCaml）提供了对它的封装。

代码示例——基于正则式进行匹配

```

#include <re2/re2.h>

int i;
string s;
assert(RE2::FullMatch("test:1234", "(\\w+):(\\d+)", &s, &i));
assert(s == "test");
assert(i == 1234);

```

Oniguruma（鬼车）

Home: <http://www.geocities.jp/kosako3/oniguruma/>

Links: [Wikipedia](#)

来自日本的正则式库，基于 C 语言。据说性能很高。

它被用在 Ruby、TextMate、Sublime Text、SubEthaEdit 等软件上。

POCO.RegExp

Docs: <http://pocoproject.org/docs/package-Foundation.RegExp.html>

POCO 前面已经介绍过。它提供了正则表达式的封装类。

Qt.QRegExp

Docs: <http://doc.qt.io/qt-4.8/qregexp.html>

Qt 前面已经介绍过。这是 Qt 中的一个类，提供了“正则式”的功能。

2.3.4（其它）

Boost.StringAlgorithms

Docs: <http://boost.org/libs/algorithm/string>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了各种字符串的算法（替换、合并、拆分、大小写转换、trim.....）。

代码示例——字符串合并

```

#include <string>
#include <vector>
#include <iostream>
#include <boost/algorithm/string.hpp>

std::vector<std::string> v;
// 此处填充 v
std::cout << boost::algorithm::join(v, " ") << '\n';

```

Boost.Lexical_Cast

Docs: http://boost.org/libs/lexical_cast

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了各种字符串与其它类型的转换。

注：Boost 的这个子库已经进入 C++ 11 标准。

代码示例

```
#include <string>
#include <iostream>
#include <boost/lexical_cast.hpp>

std::string s = boost::lexical_cast<std::string>(123);
std::cout << s << '\n';

double d = boost::lexical_cast<double>(s);
std::cout << d << '\n';
```

2.4 内存相关

2.4.1 智能指针

Boost.SmartPointers

Docs: http://boost.org/libs/smart_ptr

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了几种智能指针。最常用的是“shared_ptr”。

有了智能指针，你就无需操心 new 之后的 delete 了。

注：Boost 的这个子库已经进入 C++ 11 标准。

2.4.2 内存池

Boost.Pool

Docs: <http://boost.org/libs/pool>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“内存池”的功能。

Dlib

Docs: <http://dlib.net/other.html>

Dlib 前面已经介绍过。它提供了内存池（参见文档中以“memory_manager”开头的类）。

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了内存池的功能。

2.5 时间 & 日期

Boost.Date_Time

Docs: http://boost.org/libs/date_time

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了针对“日期 和 时间”的各种处理。

POCO.DateTime

Docs: <http://pocoproject.org/docs/package-Foundation.DateTime.html>

POCO 前面已经介绍过。它提供了若干个日期和时间的封装类（时区转换、格式化字符串、解析时间字符串）。

2.6 编码 & 解码

2.6.1 序列化

Boost.Serialization

Docs: <http://boost.org/libs/serialization>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了【可定制的】“序列化”功能。

2.6.2 Base64

Base64 是一组编码算法的总称。用于把二进制数据编码为文本。

Boost.Serialization

Docs: <http://boost.org/libs/serialization>

Boost 前面已经介绍过。使用前面提到的“Boost.Serialization”，你可以定制采用 Base64 方式进行编码和解码。

Crypto++

Docs: <http://www.cryptopp.com/docs/ref/annotated.html>

Crypto++ 前面已经介绍过。它提供了6个类，分别用于 Base64、Base32、Base16 的编码/解码。

POCO.Streams

Docs: <http://pocoproject.org/docs/package-Foundation.Streams.html>

POCO 前面已经介绍过。它提供了 Base64 和 Base32 的编码/解码。

2.7 （其它）

2.7.1 随机数

std

ANSI C 在 stdlib.h 中提供了随机数生成函数 rand()。使用前记得先用 srand() 函数播种，否则就傻了。

Boost.Random

Docs: <http://boost.org/libs/random>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“随机数生成”的功能。

相比 ANSI C 的随机数函数，它提供的功能更丰富。

代码示例

```
#include <ctime>
#include <boost/random.hpp>

double SampleNormal(double mean, double sigma)
{
    using namespace boost;

    // 建立一个 Mersenne twister 随机数产生器，使用当前时间播种
    static mt19937 rng(static_cast<unsigned>(std::time(NULL)));

    // 选择高斯机率分布
    normal_distribution<double> norm_dist(mean, sigma);

    // 使用 function 的形式，生成随机数据产生器
    variate_generator<mt19937&, normal_distribution<double> > normal_sampler(rng, norm_dist);

    // 传回样本分布结果
    return normal_sampler();
}
```

```
}
```

Crypto++

Docs: http://www.cryptopp.com/docs/ref/class_random_number_generator.html

Crypto++ 前面已经介绍过。它提供了好几个类，用于随机数生成。

2.7.2 UUID

Boost.UUID

Docs: <http://boost.org/libs/uuid>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了 UUID 的生成。

代码示例——生成 UUID

```
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>

// uuid 类以 POD 方式实现，可以直接用在 memcpy()
unsigned char uuid_data[16];
boost::uuids::uuid u;
memcpy(&u, uuid_data, 16);
```

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了 UUID 的生成、格式化成字符串、解析 UUID 字符串。

POCO.UUID

Docs: <http://pocoproject.org/docs/package-Foundation.UUID.html>

POCO 前面已经介绍过。它提供了 UUID 的生成、格式转换。

3 编程范式

（这一章节主要针对 C++——“支持多范式”是 C++ 的一大特色）

3.1 面向对象编程（OOP）

fruit

Home: <https://github.com/google/fruit>

它是 Google 开发的 C++ 库，提供了“依赖注入”（[dependency injection](#)）的框架。

代码示例——一个简单的例子

```
#include <iostream>
#include <fruit/fruit.h>

using fruit::Component;
using fruit::Injector;

class Writer
{
public:
    virtual void write(std::string str) = 0;
};

class StdoutWriter : public Writer
```

```

{
public:
    // Like "StdoutWriter() = default;" but also marks this constructor as the
    // one to use for injection.
    INJECT(StdoutWriter()) = default;

    virtual void write(std::string str) override
    {
        std::cout << str;
    }
};

class Greeter
{
public:
    virtual void greet() = 0;
};

class GreeterImpl : public Greeter
{
private:
    Writer* writer;

public:
    // Like "GreeterImpl(Writer* writer) {...}"
    // but also marks this constructor as the one to use for injection.
    INJECT(GreeterImpl(Writer* writer))
        : writer(writer)
    {
    }

    virtual void greet() override
    {
        writer->write("Hello world!\n");
    }
};

Component<Greeter> getGreeterComponent()
{
    return fruit::createComponent()
        .bind<Writer, StdoutWriter>()
        .bind<Greeter, GreeterImpl>();
}

int main()
{
    Injector<Greeter> injector(getGreeterComponent());
    Greeter* greeter = injector.get<Greeter*>();
    greeter->greet();
    return 0;
}

```

3.2 泛型编程（GP）

Boost.TypeTraits

Docs: http://boost.org/libs/type_traits

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“类型特化”相关的辅助功能。

3.3 函数式编程（FP）

（不了解“函数式编程”的同学，可以先看[维基百科](#)）

Boost.Function

Docs: <http://boost.org/libs/function>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，用来辅助封装函数对象（仿函式）。

注：Boost 的这个子库已经进入 C++ 11 标准。

代码示例——封装标准 C 的函数

```
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <boost/function.hpp>

using namespace std;

boost::function<int(const char*)> f = atoi;
cout << f("42") << '\n';

f = strlen;
cout << f("42") << '\n';
```

Boost.Lambda

Docs: <http://boost.org/libs/lambda>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“匿名函数/无名函数”的功能。

注：Boost 的这个子库已经进入 C++ 11 标准。

代码示例

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <boost/lambda/lambda.hpp>

using namespace std;

vector<int> v;
// 此处填充 v
for_each(v.begin(), v.end(), cout << boost::lambda::_1 << "\n");
```

3.4 元编程（Metaprogramming）

（不知道何为“元编程”，可以先看[维基百科](#)）

Boost.MPL

Docs: <http://boost.org/libs/mpl>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“模板元编程”的框架。

Dlib

Docs: <http://dlib.net/metaprogramming.html>

Dlib 前面已经介绍过。它提供了“模板元编程”的辅助类。

4 调试 & 测试

4.1 调试日志

Boost.Log

Docs: <http://boost.org/libs/log>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了记录日志的机制。

下面给出的示例是最简单的版本，其实它还提供了很丰富的扩展机制。

代码示例

```
#include <boost/log/trivial.hpp>

BOOST_LOG_TRIVIAL(debug) << "A debug severity message";
BOOST_LOG_TRIVIAL(info) << "An informational severity message";
```

POCO.Logging

Docs: <http://pocoproject.org/docs/package-Foundation.Logging.html>

POCO 前面已经介绍过。它提供了好几个用于调试日志的封装类（包括特定于 Windows 平台和特定于 POSIX 平台的类）。

它还支持日志文件的循环存储。

Dlib

Docs: <http://dlib.net/other.html#logger>

Dlib 前面已经介绍过。它提供了风格类似 log4j 的日志记录机制。

代码示例

```
#include <dlib/logger.h>
#include <dlib/misc_api.h>

using namespace dlib;

logger dlog("example");
dlog.set_level(LALL);

dlog << LINFO << "This is an informational message.";

int variable = 8;
dlog << LDEBUG << "The integer variable is set to " << variable;
```

wxWidgets

Docs: <http://docs.wxwidgets.org/trunk/grouplog.html>

wxWidgets 前面已经介绍过。它提供了记录日志的函数和宏。

log4cpp

Home: <http://log4cpp.sourceforge.net/>

如其名，这是一个模仿 log4j 的 C++ 库。支持多种操作系统（包括 Windows）。

4.2 单元测试

Boost.Test

Docs: <http://boost.org/libs/test>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了与测试相关的各种辅助工具（包括单元测试）。

Google Test

Home: <https://github.com/google/googletest>

Links: [Wikipedia](#)

这是 Google 提供的单元测试框架。从 Google Code 迁移到 GitHub 之后，又整合了 GoogleMock 项目。

一些知名的开源项目（Chromium、LLVM、OpenCV）用到了它。

CppUnit

Home: <http://freedesktop.org/wiki/Software/cppunit/>

Links: [Wikipedia](#)

如其名，这是一个 C++ 的单元测试框架。该项目起先是作为 JUnit 的 C++ 移植而创建的。

Check

Home: <http://check.sourceforge.net/>

Links: [Wikipedia](#)

这是针对 C 的单元测试框架。

代码示例

```
#include <check.h>

/* The basic unit test looks as follows: */
START_TEST (test_name)
{
    /* unit test code */
}
END_TEST

/* The "START_TEST/END_TEST" pair are macros that setup basic structures to permit testing.
   It is a mistake to leave off the END_TEST marker;
   doing so produces all sorts of strange errors when the check is compiled. */
```

4.3 健壮性测试

Boost.Test.ExecutionMonitor

Docs: <http://boost.org/libs/test/doc/html/execution-monitor.html>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，它除了提供“单元测试”，还提供“内存泄漏的检测”。

4.4 性能测试

benchmark

Home: <https://github.com/google/benchmark>

这是 Google 提供的性能测试辅助工具，用来测试指定函数的执行时间。

它可以把测试结果导出为 CSV 或 JSON 格式。

5 操作系统

5.1 跨操作系统

5.1.1 文件系统

Boost.Filesystem

Docs: <http://boost.org/libs/filesystem>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了对“文件系统”的操作。

代码示例——获取文件大小

```
#include <iostream>
#include <boost/filesystem.hpp>

int main(int argc, char* argv[])
```



```
{
    using namespace boost::filesystem;
    if(argc != 2)
    {
        std::cout << "Usage: \n" << argv[0] << " path\n";
        return 1;
    }
    std::cout << argv[1] << " " << file_size(argv[1]) << '\n';
    return 0;
}
```

POCO.Filesystem

Docs: <http://pocoproject.org/docs/package-Foundation.Filesystem.html>

POCO 前面已经介绍过。它提供了文件系统相关的封装类（遍历目录和文件、通配符匹配、临时文件、文件变化通知...）。

wxWidgets

Docs: <http://docs.wxwidgets.org/trunk/groupfile.html>

wxWidgets 前面已经介绍过。它提供了文件系统相关的封装类（遍历目录和文件、临时文件、文件变化通知...）。

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了“文件信息、文件名匹配”等功能。

5.1.2 线程

Boost.Thread

Docs: <http://boost.org/libs/thread>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“多线程”的功能。

代码示例

```
#include <iostream>
#include <boost/thread/thread.hpp>

void hello_world()
{
    std::cout << "Hello world, I'm a thread!\n";
}

int main()
{
    boost::thread my_thread(&hello_world); // 启动一个线程
    my_thread.join(); // 等待该线程结束
    return 0;
}
```

ACE

Docs: <http://www.dre.vanderbilt.edu/Doxygen/Stable/libace-doc/annotated.html>

ACE 前面已经介绍过。它提供了“多线程”的功能（参见文档中以“ACE_Thread”开头的类）

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了“线程池、线程同步/互斥”等功能，以及一些线程安全的数据结构。

POCO.Threading

Docs: <http://pocoproject.org/docs/package-Foundation.Threading.html>

POCO 前面已经介绍过。它提供了线程、线程池以及线程同步/互斥的封装类。

wxWidgets

Docs: <http://docs.wxwidgets.org/trunk/grouphreading.html>

wxWidgets 前面已经介绍过。它提供了线程以及线程同步/互斥的封装类。

GNU Common C++

Home: <http://www.gnu.org/software/commoncpp/>

由 GNU 提供的一套跨平台的线程并发框架。

5.1.3 进程

ACE

Docs: <http://www.dre.vanderbilt.edu/Doxygen/Stable/libace-doc/annotated.html>

ACE 前面已经介绍过。它提供了“进程管理”的功能（参见文档中以“ACE_Process”开头的类）。

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了“进程管理”的功能。

POCO.Processes

Docs: <http://pocoproject.org/docs/Poco.Process.html>

POCO 前面已经介绍过。它提供了“进程”的封装类。

5.1.4 本地进程间通信（IPC）

（本章节列举的是【本地】IPC，跨主机的网络通讯，参见本页面后续的章节）

Boost.Interprocess

Docs: <http://boost.org/libs/interprocess>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了共享内存和几种同步机制（Mutexes、Condition variables、Semaphores、Upgradable mutexes、File locks）。

ACE

Docs: <http://www.dre.vanderbilt.edu/Doxygen/Stable/libace-doc/annotated.html>

ACE 前面已经介绍过。它提供了许多种 IPC 机制（有些不是跨平台的）。

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了“进程同步、共享内存、信号处理”等功能。

POCO.Processes

Docs: <http://pocoproject.org/docs/package-Foundation.Processes.html>

POCO 前面已经介绍过。它提供了 IPC 相关的封装类（“共享内存”和“管道”）。

5.2 特定于 Windows 系统

5.2.1 注册表

wxWidgets

Docs: http://docs.wxwidgets.org/trunk/classwx_reg_key.html

wxWidgets 前面已经介绍过。它提供了操作 Windows 注册表的封装类。

POCO::Util

Docs: <http://pocoproject.org/docs/Poco.Util.html>

POCO 前面已经介绍过。它提供了操作 Windows 注册表的封装类（WinRegistryKey）。

5.2.2 Windows 服务（Service）

POCO::Util

Docs: <http://pocoproject.org/docs/Poco.Util.html>

POCO 前面已经介绍过。它提供了相应的封装类（WinService），可以用来操作 Service（注册、删除、启动、停止）。

5.3 特定于 Linux & Unix 系统

6 图形用户界面（GUI）

6.1 GUI 框架

6.1.1 跨平台的 GUI 框架

wxWidgets

Docs: <http://docs.wxwidgets.org/trunk/modules.html>

wxWidgets 前面已经介绍过。用 wxWidgets 开发 GUI 应用，其代码结构类似 MFC。熟悉 MFC 的程序员应该很容易上手。

代码示例——Hello world

```
#include <wx/wxprec.h>
#ifdef WX_PRECOMP
#    include <wx/wx.h>
#endif

class MyApp: public wxApp
{
public:
    virtual bool OnInit();
};
wxIMPLEMENT_APP(MyApp);

class MyFrame: public wxFrame
{
public:
    MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size);
private:
    void OnExit(wxCommandEvent& event);
    wxDECLARE_EVENT_TABLE();
};
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_EXIT, MyFrame::OnExit)
wxEND_EVENT_TABLE()

bool MyApp::OnInit()
{
    MyFrame* frame = new MyFrame("Hello, World", wxPoint(50, 50), wxSize(450, 340));
    frame->Show(true);
    return true;
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
```

```

        : wxFrame(NULL, wxID_ANY, title, pos, size)
    {
        wxMenu* menuFile = new wxMenu();
        menuFile->Append(wxID_EXIT);
        wxMenuBar* menuBar = new wxMenuBar();
        menuBar->Append(menuFile, "&File");
        SetMenuBar(menuBar);
    }

    void MyFrame::OnExit(wxCommandEvent& event)
    {
        Close(true);
    }

```

Qt

Qt 前面已经介绍过。下面给出一个 Hello world 的示例，让你看看 Qt 的风格。

代码示例——Hello world

```

#include <QtWidgets/QApplication>
#include <QtWidgets/QLabel>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QLabel label("Hello, world!");
    label.show();
    return app.exec();
}

```

GTK+

Home: <http://www.gtk.org/>

Links: [Wikipedia 维基百科](#)

老牌的 GUI 框架，诞生于1998年。原先叫做“GIMP Toolkit”，是基于 C 开发的跨平台界面组件库。

代码示例——Hello world

```

#include <gtk/gtk.h>

int main(int argc, char* argv[])
{
    gtk_init(&argc, &argv);

    GtkWidget* window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Hello, world!");
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(window), 200, 100);

    /*
    ** Map the destroy signal of the window to gtk_main_quit;
    ** When the window is about to be destroyed, we get a notification and
    ** stop the main GTK+ loop by returning 0
    */
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

    gtk_widget_show_all(window);

    /* Start the main loop, do nothing until the application is closed */
    gtk_main();

    return 0;
}

```

FLTK

Home: <http://www.fltk.org/>

Links: [Wikipedia 维基百科](#)

它的全称是“Fast, Light Toolkit”。如其名，它非常轻量级。用它写一个“Hello World 应用”，静态链接后大约才 100KB

代码示例——Hello world

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>

int main(int argc, char **argv)
{
    Fl_Window* window = new Fl_Window(300, 180);
    Fl_Box* box = new Fl_Box(20, 40, 260, 100, "Hello, World!");
    box->box(FL_UP_BOX);
    box->labelsize(36);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labeltype(FL_SHADOW_LABEL);

    window->end();
    window->show(argc, argv);
    return Fl::run();
}
```

6.1.2 特定于 Windows 的 GUI 框架

WTL (Windows Template Library)

Home: <http://sourceforge.net/projects/wtl/>

Links: [Wikipedia 维基百科](#)

这是微软雇员 Nenad Stefanovic 开发的开源 GUI 框架。开发 WTL 是为了提供一个轻量级的 MFC 替代品。

6.2 图表 (Chart)

Qt

Docs: <http://doc.qt.io/QtCharts/>

Qt 前面已经介绍过。它内置了一套 Chart 的封装类。

代码示例——饼图

```
#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>
#include <QtCharts/QChartView>
#include <QtCharts/QPieSeries>
#include <QtCharts/QPieSlice>

QT_CHARTS_USE_NAMESPACE

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);

    QPieSeries* series = new QPieSeries();
    series->append("Jane", 1);
    series->append("Joe", 2);
    series->append("Andy", 3);
    series->append("Barbara", 4);
    series->append("Axel", 5);

    QPieSlice* slice = series->slices().at(1);
    slice->setExploded();
    slice->setLabelVisible();
    slice->setPen(QPen(Qt::darkGreen, 2));
    slice->setBrush(Qt::green);

    QChart* chart = new QChart();
```

```

chart->addSeries(series);
chart->setTitle("Simple piechart example");
chart->legend()->hide();

QChartView* chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QMainWindow window;
window.setCentralWidget(chartView);
window.resize(400, 300);
window.show();

return app.exec();
}

```

wxCode

Home: <http://wxcode.sourceforge.net/>

该项目专门提供组件来扩展 wxWidgets 的功能。它里面提供了好几种图表的组件（wxChart、wxFreeChart、wxPlotCtrl）。

wxMathPlot

Home: <http://wxmathplot.sourceforge.net/>

看名称就知道它是跟 wxWidgets 搭配的。效果图参见“[这里](#)”

7 文本用户界面（TUI）

7.1 命令行参数

getopt

Home: https://www.gnu.org/software/libc/manual/html_node/Getopt.html

Links: [Wikipedia](#)

这是标准C用来处理命令行参数的老牌函数，诞生于上世纪80年代初期。

它有很多种不同的实现，如今用得最多的是 GNU C Library 的实现。GNU 还实现了一个增强版 getopt_long。

代码示例

```

#include <stdio.h>    /* for printf */
#include <stdlib.h>   /* for exit */
#include <unistd.h>   /* for getopt */

int main(int argc, char* argv[])
{
    int digit_optind = 0;
    int aopt = 0, bopt = 0;
    char* copt = NULL;
    char* dopt = NULL;
    int c;
    while( (c = getopt(argc, argv, "abc:d:012")) != -1)
    {
        int this_option_optind = optind ? optind : 1;
        switch(c)
        {
            case '0':
            case '1':
            case '2':
                if(digit_optind != 0 && digit_optind != this_option_optind)
                {
                    printf("digits occur in two different argv-elements.\n");
                }
                digit_optind = this_option_optind;
                printf("option %c\n", c);

```

```
        break;
    case 'a':
        printf("option a\n");
        aopt = 1;
        break;
    case 'b':
        printf("option b\n");
        bopt = 1;
        break;
    case 'c':
        printf("option c with value '%s'\n", optarg);
        copt = optarg;
        break;
    case 'd':
        printf("option d with value '%s'\n", optarg);
        dopt = optarg;
        break;
    case '?':
        break;
    default:
        printf("?? getopt returned character code 0%o ??\n", c);
    }
}
if(optind < argc)
{
    printf("non-option ARGV-elements: ");
    while(optind < argc)
    {
        printf("%s ", argv[optind++]);
    }
    printf("\n");
}
exit (0);
}
```

Boost.Program_options

Docs: http://boost.org/libs/program_options

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“处理命令行参数”的功能。

它的功能很丰富，但是比较重型。

7.2 文本终端

ncurses

Home: <https://www.gnu.org/software/ncurses/>

Links: [Wikipedia 维基百科](#)

ncurses 是“new curses”的缩略词，它是 [curses](#) 库的自由软件克隆，诞生于1993年。

大名鼎鼎的 [Eric S. Raymond](#) 曾参与早期版本的开发。

8 网络

8.1 链路层 & 网络层

libpcap

Home: <http://www.tcpdump.org/>

Links: [Wikipedia](#)

很著名的 Sniffer 抓包库，基于 C 语言开发。

代码示例——一个简单的抓包示例

```

#include <stdio.h>
#include <pcap.h>

int main()
{
    pcap_t* handle;          /* Session handle */
    char* dev;               /* The device to sniff on */
    char errbuf[PCAP_ERRBUF_SIZE]; /* Error string */
    struct bpf_program fp;    /* The compiled filter */
    char filter_exp[] = "port 23"; /* The filter expression */
    bpf_u_int32 mask;        /* Our netmask */
    bpf_u_int32 net;         /* Our IP */
    struct pcap_pkthdr header; /* The header that pcap gives us */
    const u_char* packet;     /* The actual packet */

    /* Define the device */
    dev = pcap_lookupdev(errbuf);
    if(dev == NULL)
    {
        fprintf(stderr, "Couldn't find default device: %s\n", errbuf);
        return 2;
    }
    /* Find the properties for the device */
    if(pcap_lookupnet(dev, &net, &mask, errbuf) == -1)
    {
        fprintf(stderr, "Couldn't get netmask for device %s: %s\n", dev, errbuf);
        net = 0;
        mask = 0;
    }
    /* Open the session in promiscuous mode */
    handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
    if(handle == NULL)
    {
        fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
        return 2;
    }
    /* Compile and apply the filter */
    if(pcap_compile(handle, &fp, filter_exp, 0, net) == -1)
    {
        fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp, pcap_geterr(handle));
        return 2;
    }
    if(pcap_setfilter(handle, &fp) == -1)
    {
        fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp, pcap_geterr(handle));
        return 2;
    }

    packet = pcap_next(handle, &header); /* Grab a packet */
    printf("Jacked a packet with length of [%d]\n", header.len);
    pcap_close(handle); /* Close the session */
    return 0;
}

```

WinPcap

Home: <http://www.winpcap.org/>

Links: [Wikipedia](#)

它是 libpcap 在 Windows 系统下的移植。

8.2 传输层

socket

socket 最早源自 BSD 系统，有时候也称“伯克利套接字”。

它已成了传输层网络编程的标准，主流的操作系统平台都支持，主流的 C/C++ 编译器也都内置了相关的库文件。

ACE

Docs: <http://www.dre.vanderbilt.edu/Doxygen/Stable/libace-doc/annotated.html>

ACE 前面已经介绍过。它提供了针对 socket 的更高层封装。

APR

Docs: <https://apr.apache.org/docs/apr/trunk/modules.html>

APR 前面已经介绍过。它提供了对 socket 的封装和增强。

POCO::Net

Docs: <http://pocoproject.org/docs/Poco.Net.html>

POCO 前面已经介绍过。它提供了针对 TCP 服务端的封装类。

8.3 标准的应用层

8.3.1 综合性的库

cURL & libcurl

Home: <http://curl.haxx.se/libcurl/>

Links: [Wikipedia 维基百科](#)

cURL 是一个功能很强的网络库/网络工具，支持 N 多应用层协议。下面是支持协议的列表（从它官网抄袭的）

DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMTP, SMTPS, Telnet and TFTP. curl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, HTTP/2

它采用 C 语言开发，开发很活跃，支持非常多的操作系统平台。

关于 cURL，俺前几年写过一篇博文推荐它（在“[这里](#)”）。

代码示例——IMAP 协议（邮件）

```
#include <stdio.h>
#include <curl/curl.h>

int main()
{
    curl_global_init(CURL_GLOBAL_ALL);

    CURL* curl = curl_easy_init();
    if(curl)
    {
        curl_easy_setopt(curl, CURLOPT_USERNAME, "user");
        curl_easy_setopt(curl, CURLOPT_PASSWORD, "password");

        // This will list the folders within the user's mailbox. If you want to
        // list the folders within a specific folder, for example the inbox,
        // then specify the folder as a path in the URL such as /INBOX
        curl_easy_setopt(curl, CURLOPT_URL, "imap://imap.example.com");

        CURLcode res = curl_easy_perform(curl);
        if(res != CURLE_OK) // Check for errors
        {
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
                curl_easy_strerror(res));
        }

        curl_easy_cleanup(curl); // Always cleanup
    }

    curl_global_cleanup();
    return 0;
}
```

cURLpp

Home: <http://www.curlpp.org/>

看名字就知道这是 cURL 的 C++ 封装。

POCO::Net

Docs: <http://pocoproject.org/docs/Poco.Net.html>

POCO 前面已经介绍过。它提供了几种常见应用层协议（HTTP、SMTP、POP3、FTP、NTP ...）的封装类。

8.3.2 HTTP

（关于“HTTP 协议”，请参见另一个大类：“Web 相关”）

8.3.3 DNS

c-ares

Home: <http://c-ares.haxx.se/>

这是一个 C 语言开发的 DNS 封装库，支持异步 DNS 请求，跨多种操作系统。

对比官网域名可知，它跟 cURL 是一家子。除了 cURL/libcurl 用到它，还有一些知名开源项目（比如：Wireshark、node.js ...）用到它。

8.3.4 XMPP

（XMPP 的洋文全称是“Extensible Messaging and Presence Protocol”。这是一个标准化的 IM 交互协议）

Swiften

Home: <http://swift.im/swiften.html>

这是一个 C++ 语言开发的 XMPP 封装库。它同时也是 Swift 聊天客户端所用的后端。

它大量使用了 Boost 的子库（Signal、Bind、Optional、Smart Pointers ...）。

QXmpp

Home: <https://github.com/qxmpp-project/qxmpp>

这是一个 C++ 语言开发的 XMPP 封装库。从它的名称可以看出——依赖了 Qt 框架（需要 Qt 4.5 或更高版本）。

8.4 自定义的应用层

Protocol Buffers

Home: <https://developers.google.com/protocol-buffers/>

Links: [Wikipedia](#)

它是 Google 开发的一个跨语言的库，用于传输业务数据时的“编码/解码”。其优点是：跨多种语言、高性能、向前兼容、向后兼容。

具体的使用，可以参考俺前几年写过的一篇博文（在“[这里](#)”）。

作为 Protocol Buffers 的发明者，Google 默认实现了三种编程语言（C++、Java、Python）对它的支持。除了 Google 官方提供的这三种语言，它还支持很多其它的编程语言（由第三方提供）。

Apache Thrift

Home: <https://thrift.apache.org/>

Links: [Wikipedia](#)

来自于 Apache 社区，提供了一种跨语言的通讯机制。

程序员通过 Thrift 的“接口定义语言”定义通讯协议格式，然后 Thrift 根据协议格式自动帮你生成服务端和客户端代码。

（在这个方面，它有点类似于 Google 的 Protocol Buffers）

8.5 网络库、框架、中间件

Boost.Asio

Docs: <http://boost.org/libs/asio>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了异步网络通讯和异步 I/O。

代码示例——TCP Server

```
#include <string>
#include <iostream>
#include <boost/asio.hpp>

int main()
{
    using boost::asio::ip::tcp;
    try
    {
        boost::asio::io_service io_service;
        tcp::acceptor acceptor(io_service, tcp::endpoint(tcp::v4(), 13));
        while(true)
        {
            tcp::socket socket(io_service);
            acceptor.accept(socket);

            std::string msg = "Hello, world";
            boost::system::error_code ignored_err;
            boost::asio::write(socket, boost::asio::buffer(msg), ignored_err);
        }
    }
    catch(std::exception& err)
    {
        std::cerr << err.what() << std::endl;
    }
    return 0;
}
```

ACE

Docs: <http://www.dre.vanderbilt.edu/Doxygen/Stable/libace-doc/annotated.html>

ACE 前面已经介绍过。它提供了很多种用于网络通讯的设计模式。

ZeroMQ (ØMQ)

Home: <http://www.zeromq.org/>

Links: [Wikipedia 维基百科](#)

ZeroMQ 是一个轻量级、跨平台的开源库，提供了高性能、异步的消息队列。采用 C++ 开发，提供了多种语言的绑定。

与传统的消息中间件不同，使用 ZeroMQ 不需要额外的“消息代理（message broker）”。

俺曾经写过一篇博文推荐它（在[“这里”](#)）。

代码示例——TCP Server

```
#include <zhelpers.hpp>

int main()
{
    zmq::context_t context(1);

    zmq::socket_t responder(context, ZMQ_REP);
    responder.connect("tcp://localhost:5560");
```

```

while(true)
{
    // Wait for next request from client
    std::string request = s_recv(responder);
    std::cout << "Received request: " << request << std::endl;

    // Do some 'work'
    sleep(1);

    // Send reply back to client
    s_send(responder, "Hello, world");
}
}

```

nanomsg

Home: <http://nanomsg.org/>

很类似 ZeroMQ 的库，比 ZMQ 更加轻量级。采用 C 开发，提供了多种语言的绑定。

API 完全参照 BSD socket 的风格和语义。

代码示例——Request/Reply

```

#include <assert.h>
#include <libc.h>
#include <stdio.h>
#include <nanomsg/nm.h>
#include <nanomsg/pipeline.h>

int reply(const char* url)
{
    int sock = nn_socket(AF_SP, NN_PULL);
    assert(sock >= 0);
    assert(nn_bind(sock, url) >= 0);
    while(1)
    {
        char* msg = NULL;
        int bytes = nn_recv(sock, &msg, NN_MSG, 0);
        assert(bytes >= 0);
        printf("RECEIVED:\n%s\n", msg);
        nn_freemsg(msg);
    }
}

int request(const char* url, const char* msg)
{
    int sz_msg = strlen(msg) + 1; // '\0'
    int sock = nn_socket(AF_SP, NN_PUSH);
    assert(sock >= 0);
    assert(nn_connect(sock, url) >= 0);
    printf("SENDING:\n%s\n", msg);
    int bytes = nn_send(sock, msg, sz_msg, 0);
    assert(bytes == sz_msg);
    return nn_shutdown(sock, 0);
}

```

ICE (Internet Communications Engine)

Home: <https://zeroc.com/>

Links: [Wikipedia 维基百科](#)

这是一个面向对象的通讯中间件，诞生于2002年。支持不同编程语言的通讯。

它的设计借鉴了 CORBA，好在没有 CORBA 那么复杂。

libevent

Home: <http://libevent.org/>

Links: [Wikipedia 维基百科](#)

它提供了异步事件处理机制。在网络开发中，可以用它替代传统的“event loop”，有助于简化代码。

它被一些知名的开源项目使用（比如：[Tor](#)、[memcached](#)）。

代码示例——HTTP Server（本示例基于 ANSI C）

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <event.h>
#include <evhttp.h>

void generic_request_handler(struct evhttp_request* req, void* arg)
{
    struct evbuffer* return_buffer = evbuffer_new();

    evbuffer_add_printf(return_buffer, "Thanks for the request!");
    evhttp_send_reply(req, HTTP_OK, "Client", return_buffer);
    evbuffer_free(return_buffer);
}

int main()
{
    short      http_port = 8080;
    char*      http_addr = "127.0.0.1";
    struct evhttp* http_server = NULL;

    event_init();
    http_server = evhttp_start(http_addr, http_port);
    evhttp_set_genb(http_server, generic_request_handler, NULL);

    fprintf(stderr, "Server started on port %d\n", http_port);
    event_dispatch();

    return 0;
}
```

代码示例——HTTP Server（本示例基于 C++ 11 标准）

```
#include <memory>
#include <cstdint>
#include <iostream>
#include <evhttp.h>

int main()
{
    if(!event_init())
    {
        std::cerr << "Failed to init libevent." << std::endl;
        return -1;
    }
    char const SrvAddress[] = "127.0.0.1";
    std::uint16_t SrvPort = 8080;
    std::unique_ptr<evhttp, decltype(&evhttp_free)> Server(evhttp_start(SrvAddress, SrvPort), &evhttp_free);
    if(!Server)
    {
        std::cerr << "Failed to init http server." << std::endl;
        return -1;
    }

    void (*OnReq)(evhttp_request* req, void*) = [] (evhttp_request* req, void*)
    {
        auto* OutBuf = evhttp_request_get_output_buffer(req);
        if(!OutBuf)
        {
            return;
        }
        evbuffer_add_printf(OutBuf, "<html><body><h1>Hello, World!</h1></body></html>");
        evhttp_send_reply(req, HTTP_OK, "", OutBuf);
    };
}
```

```
};

evhttp_set_gencb(Server.get(), OnReq, nullptr);
if(event_dispatch() == -1)
{
    std::cerr << "Failed to run messsage loop." << std::endl;
    return -1;
}
return 0;
}
```

libev

Home: <http://libev.schmorp.de/>

看名称就能猜到它跟 libevent 很像。实际上，你可以把 libev 视为 libevent 的竞争性替代品。

[gevent](#) 官方博客的一篇文章对比了这两库的优缺点。

libuv

Home: <https://github.com/libuv/libuv>

Links: [Wikipedia](#)

它提供了跨平台的异步 I/O 机制。主要是为了给 [Node.js](#) 使用。

除了支持网络通讯，还支持：线程池、Windows 命名管道、Unix domain sockets、文件系统事件通知

Dlib

Docs: <http://dlib.net/network.html>

Dlib 前面已经介绍过。它针对网络通讯，提供了比较高的抽象层。

9 数据库

9.1 开源数据库

MySQL

Home: <https://www.mysql.com/>

Links: [Wikipedia](#) 维基百科

名气最大的开源数据库，诞生于1995年，采用 C 和 C++ 语言开发。如今隶属于 Oracle 公司。

PostgreSQL

Home: <http://postgresql.org/>

Links: [Wikipedia](#) 维基百科

名气仅次于 MySQL 的开源数据库，诞生于1996年。采用 C 语言开发。

SQLite

Home: <http://sqlite.org/>

Links: [Wikipedia](#) 维基百科

它是一个很优秀的嵌入式（进程内）数据库，非常轻量级，支持各种作系统平台。采用 C 语言开发。

俺前几年写过一篇博文推荐它（在“[这里](#)”）。

MongoDB

Home: <https://www.mongodb.org/>

Links: [Wikipedia](#)

这是近几年兴起的 NoSQL 数据库的一员。它本身是基于 C++ 和 C 开发的。

Redis

Home: <http://redis.io/>

Links: [维基百科](#) [Wikipedia](#)

诞生于2009年，是目前（2014~2015）最流行的键值存储数据库，基于 C 语言开发。

以性能高而著称，很多大型网站用到它（Twitter、GitHub、Flickr、Instagram、百度、新浪、腾讯、搜狐）

Berkeley DB（BDB）

Home: <http://www.oracle.com/us/products/database/berkeley-db/>

Links: [Wikipedia](#) [维基百科](#)

诞生于1994年，是一个很老牌的嵌入式（进程内）数据库，提供“键值存储”的功能，基于 C 语言开发。

开发 BDB 的公司于2006年被 Oracle 收购。

很多开源项目用到它。甚至 MySQL 也包含了一个基于 BDB 的存储后端。

LevelDB

Home: <https://github.com/google/leveldb>

Links: [Wikipedia](#) [维基百科](#)

它是 Google 基于 C++ 开发的 NoSQL 数据库，提供“键值存储”的功能。

号称速度很快，内置数据压缩（基于 [Snappy](#) 库）。

比特币项目用到它。Facebook 基于它开发出 RocksDB 数据库。

Firebird

Home: <http://www.firebirdsql.org/>

Links: [Wikipedia](#) [维基百科](#)

它是2000年的时候，从 [Borland](#) 公司的 InterBase 数据库派生出来的。

基于 C++ 开发，支持多种操作系统平台。

关于它有个插曲：Firefox 浏览器曾经用过“Firebird”这个名称，因为跟 Firebird 数据库同名，后来才改用 Firefox 这个名称。

ScyllaDB

Home: <http://www.scylladb.com/>

这是2015年新兴的 NoSQL 数据库，相当于是用 C++ 重写了（Java 开发的）Cassandra。

号称性能提高10倍，并且延迟极低。

9.2 数据库 API 的封装库

9.2.1 综合性的封装库

OTL

Home: <http://otl.sourceforge.net/>

Links: [Wikipedia](#)

原生支持的数据库：Oracle、SQL Server、DB2、Informix、TimesTen、MAX/DB；另支持 ODBC。

它的特色是：全部代码都在一个头文件中。

代码示例——操作 Oracle 数据库

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

#define OTL_ORA8 // Compile OTL 4.0/OCI8
#include <otlv4.h> // include the OTL 4.0 header file

int main()
{
    otl_connect::otl_initialize();
    try
    {
        otl_connect db;
        db.rlogin("scott/tiger"); // connect to Oracle
        otl_cursor::direct_exec(
            db,
            "drop table test_tab",
            otl_exception::disabled // disable OTL exceptions
        );

        otl_cursor::direct_exec(
            db,
            "create table test_tab(f1 number, f2 varchar2(30))"
        );
    }
    catch(otl_exception& err)
    {
        using namespace std;
        // intercept OTL exceptions
        cerr << err.msg << endl; // print error message
        cerr << err.stm_text << endl; // print SQL that caused the error
        cerr << err.var_info << endl; // print variable that caused the error
    }

    db.logoff(); // disconnect from Oracle
    return 0;
}
```

9.2.2 MySQL 封装库

MySQL Connector C++

Home: <http://dev.mysql.com/doc/connector-cpp/en/>

这是 MySQL 官方提供的 C++ 封装。

代码示例——执行 SQL 语句

```
sql::mysql::MySQL_Driver* driver = sql::mysql::MySQL_Driver::get_mysql_driver_instance();
sql::Connection* conn = driver->connect("tcp://127.0.0.1:3306", "user", "password");
sql::Statement* stmt = conn->createStatement();

stmt->execute("USE EXAMPLE_DB");
stmt->execute("DROP TABLE IF EXISTS test");
stmt->execute("CREATE TABLE test(id INT, label CHAR(1))");
stmt->execute("INSERT INTO test(id, label) VALUES (1, 'a')");

delete stmt;
stmt = NULL;
delete conn;
conn = NULL;
```

MySQL++

Home: <http://www.tangentsoft.net/mysql%2B%2B/>

这是个老牌的库，诞生于1998年，提供了 MySQL API 的 C++ 封装。

代码示例——执行 SQL 语句

```
#include <iostream>
#include <mysql++.h>

void query(db_name, server_name, user, password)
{
    using namespace std;

    mysqlpp::Connection conn(false);
    if(!conn.connect(db_name, server_name, user, password))
    {
        cerr << "DB connection failed: " << conn.error() << endl;
        return;
    }

    mysqlpp::Query query = conn.query("select item from table1");
    mysqlpp::StoreQueryResult sqr = query.store()
    if(!sqr)
    {
        cerr << "Failed to get item list: " << query.error() << endl;
        return;
    }

    mysqlpp::StoreQueryResult::const_iterator iter;
    for(iter=sqr.begin(); iter!=sqr.end(); ++iter)
    {
        mysqlpp::Row row = *iter;
        cout << '\t' << row[0] << endl;
    }
}
```

POCO::Data::MySQL

Docs: <http://pocoproject.org/docs/package-MySQL.MySQL.html>

POCO 前面已经介绍过。它提供了 MySQL 的封装类

9.2.3 PostgreSQL 封装库

libpq

Home: <http://www.postgresql.org/docs/9.4/static/libpq.html>

这是由 PostgreSQL 官方提供的 C 封装类库。

libpqxx

Home: <http://pqxx.org/development/libpqxx/>

这是由 PostgreSQL 官方提供的 C++ 封装类库。

代码示例

```
#include <iostream>
#include <pqxx/pqxx>

void update(const std::string& name)
{
    pqxx::connection conn("dbname=company user=accounting");
    pqxx::work txn(conn);

    pqxx::result r = txn.exec("SELECT id FROM Employee WHERE name =" + name);
    if(r.size() != 1)
    {
        std::cerr << "Expected 1 employee with name " << name << ", "
                  << "but found " << r.size() << std::endl;
        return 1;
    }

    int employee_id = r[0][0].as<int>();
```

```

std::cout << "Updating employee #" << employee_id << std::endl;

txn.exec(
    "UPDATE EMPLOYEE SET salary = salary + 1 "
    "WHERE id = " + txn.quote(employee_id)
);

txn.commit();
}

```

9.2.4 Oracle 封装库

OCILIB

Home: <http://vrogier.github.io/ocilib/>

这是一个跨平台的 C 开源库。如其名，它封装了 Oracle 官方的 OCI（Oracle Call Interface）。

它同时还提供了 C++ 的 API。

代码示例

```

#include <ocilib.hpp>
using namespace ocilib;

int main()
{
    try
    {
        Environment::Initialize();
        Connection conn("db", "user", "password");
        Statement stmt(conn);
        stmt.Execute("select intcol, strcol from table");

        Resultset rs = stmt.GetResultset();
        while(rs.Next())
        {
            std::cout << rs.Get<int>(1) << " - " << rs.Get<ostring>(2) << std::endl;
        }
    }
    catch(std::exception& err)
    {
        std::cout << err.what() << std::endl;
    }

    Environment::Cleanup();
    return 0;
}

```

9.2.5 DB2 封装库

9.2.6 SQLite 封装库

官方的 C API

Docs: <http://sqlite.org/c3ref/intro.html>

SQLite 前面已经介绍过。由于 SQLite 本身就是用 C 语言开发的，因此它直接提供了基于 C 的 API 接口。

代码示例

```

#include <stdio.h>
#include <sqlite3.h>

static int callback(void* NotUsed, int argc, char** argv, char** azColName)
{
    for(int i=0; i<argc; i++)
    {
        printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
    }
}

```

```

    }
    printf("\n");
    return 0;
}

int main(int argc, char* argv[])
{
    if(argc != 3)
    {
        fprintf(stderr, "Usage: %s DATABASE SQL-STATEMENT\n", argv[0]);
        return 1;
    }

    sqlite3* db = NULL;
    int rc = sqlite3_open(argv[1], &db);
    if(rc)
    {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
    }

    char* szErrMsg = NULL;
    rc = sqlite3_exec(db, argv[2], callback, 0, &szErrMsg);
    if(rc != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", szErrMsg);
        sqlite3_free(szErrMsg);
    }

    sqlite3_close(db);
    return 0;
}

```

POCO::Data::SQLite

Docs: <http://pocoproject.org/docs/package-SQLite.SQLite.html>

POCO 前面已经介绍过。它提供了 sqlite 的封装类

9.2.7 Redis 封装库

Hiredis

Home: <https://github.com/redis/hiredis>

这是 Redis 官方提供的 C 客户端，很轻量级，支持“异步 API”。

代码示例——结合 libev 进行异步调用

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>

#include <hiredis.h>
#include <async.h>
#include <adapters/libev.h>

void getCallback(redisAsyncContext* context, void* r, void* privdata)
{
    redisReply* reply = r;
    if(reply == NULL)
    {
        return;
    }
    printf("argv[%s]: %s\n", (char*)privdata, reply->str);
    redisAsyncDisconnect(context); /* Disconnect after receiv reply */
}

void connectCallback(const redisAsyncContext* context, int status)
{

```

```

    if(status != REDIS_OK)
    {
        printf("Error: %s\n", context->errstr);
        return;
    }
    printf("Connected...\n");
}

void disconnectCallback(const redisAsyncContext* context, int status)
{
    if(status != REDIS_OK)
    {
        printf("Error: %s\n", context->errstr);
        return;
    }
    printf("Disconnected...\n");
}

int main(int argc, char* argv[])
{
    signal(SIGPIPE, SIG_IGN);

    redisAsyncContext* context = redisAsyncConnect("127.0.0.1", 6379);
    if(context->err)
    {
        printf("Error: %s\n", context->errstr);
        return 1; /* Let *context leak for now... */
    }

    redisLibevAttach(EV_DEFAULT_ context);
    redisAsyncSetConnectCallback(context, connectCallback);
    redisAsyncSetDisconnectCallback(context, disconnectCallback);
    redisAsyncCommand(context, NULL, NULL,
        "SET key %b", argv[argc-1], strlen(argv[argc-1]));
    redisAsyncCommand(context, getCallback, (char*)"end-1", "GET key");
    ev_loop(EV_DEFAULT_ 0);
    return 0;
}

```

9.2.8 MongoDB 封装库

官方的 **C API**

Docs: <https://api.mongodb.org/c/current/>

MongoDB 前面已经介绍过。这是其官方提供的 API。

代码示例

```

#include <stdio.h>
#include <bson.h>
#include <mongoc.h>

int main()
{
    mongoc_init();

    mongoc_client_t* client = mongoc_client_new("mongodb://localhost:27017/");
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "test", "test");

    bson_t* doc = bson_new();

    bson_oid_t oid;
    bson_oid_init(&oid, NULL);
    BSON_APPEND_OID(doc, "_id", &oid);
    BSON_APPEND_UTF8(doc, "hello", "world");

    bson_error_t error;
    if(!mongoc_collection_insert(collection, MONGOC_INSERT_NONE, doc, NULL, &error))
    {
        printf("%s\n", error.message);
    }
}

```

```
bson_destroy(doc);
mongoc_collection_destroy(collection);
mongoc_client_destroy(client);

return 0;
}
```

POCO::MongoDB

Docs: <http://pocoproject.org/docs/package-MongoDB.MongoDB.html>

POCO 前面已经介绍过。它提供了 MongoDB 的封装类

9.3 ODBC 相关

unixODBC

Home: <http://www.unixodbc.org/>

Links: [Wikipedia](#)

诞生于1999年，实现了全套的 ODBC 架构，包括：驱动管理器、相关的 GUI 界面和命令行界面。支持多种操作系统。

Libodbc++

Home: <http://libodbcxx.sourceforge.net/>

如其名，这是专门封装 ODBC 的 C++ 类库，支持多种操作系统。它提供的 API 类似于 JDBC 的 API

POCO::Data::ODBC

Docs: <http://pocoproject.org/docs/package-ODBC.ODBC.html>

POCO 前面已经介绍过。它提供了操作 ODBC 的封装类。

9.4 ORM (Object-Relational Mapping)

ODB

Home: <http://www.codesynthesis.com/products/odb>

Links: [Wikipedia](#)

它的特色是：可以根据 C++ 类定义自动生成数据库的表结构。

为了获得高性能，它直接调用具体数据库的原生 API。支持的数据库包括：MySQL、PostgreSQL、Oracle、SQL Server、SQLite

代码示例——声明一个可持久化的类

```
// 通过预处理语句"#pragma"来进行某些定制

#pragma db object table("people")
class person
{
public:
    // .....

private:
    friend class odb::access;
    person();

    #pragma db id auto
    unsigned long id_;

    string first_;
    string last_;
```

```
#pragma type("INT UNSIGNED")
unsigned short age_;

};
```

代码示例——查询

```
typedef odb::query<person> query;
typedef odb::result<person> result;

transaction trans(db.begin());
result r(db.query<person>(query::last == "Doe" && query::age < 30));

for(result::iterator i(r.begin()); i!=r.end(); ++i)
{
    cout << "Hello, " << i->first() << endl;
}

trans.commit();
```

hiberlite

Home: <https://github.com/paulftw/hiberlite>

专门提供给 Sqlite 的 ORM 封装库。基于 C++ 开发，其 API 采用类似 Boost.Serialization 的风格。

10 Web

10.1 HTTP Server

Apache HTTP Server

Home: <https://httpd.apache.org/>

Links: [Wikipedia 维基百科](#)

大名鼎鼎的 Apache，诞生于1995年，采用 C 和 C++ 开发。长期作为 Web Server 市场份额的老大。

Nginx

Home: <http://nginx.org/>

Links: [Wikipedia 维基百科](#)

Web Server 的后起之秀，诞生于2002年，采用 C 语言开发。其市场份额如今排名第二。

POCO::Net

Docs: <http://pocoproject.org/docs/package-Net.HTTPServer.html>

POCO 前面已经介绍过。它提供了 HTTP Server 的封装类

Dlib::server_http

Docs: http://dlib.net/network.html#server_http

Dlib 前面已经介绍过。它提供了一个简单的 HTTP Server 的类（server_http）。

10.2 HTTP Client

cURL & libcurl

Docs: <http://curl.haxx.se/libcurl/c/>

libcurl 前面已经介绍过。它提供了【完整的】HTTP 协议支持。另，HTTP 2.0 标准刚出来不久，它就已经支持了。

代码示例——HTTP POST

```
#include <stdio.h>
#include <curl/curl.h>

int main()
{
    curl_global_init(CURL_GLOBAL_ALL);

    CURL* curl = curl_easy_init();
    if(curl)
    {
        curl_easy_setopt(curl, CURLOPT_URL, "http://post.example.com/foo.cgi");
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "name=daniel&project=curl");

        CURLcode res = curl_easy_perform(curl);
        if(res != CURLE_OK) // Check for errors
        {
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
                curl_easy_strerror(res));
        }
        curl_easy_cleanup(curl); // always cleanup
    }

    curl_global_cleanup();
    return 0;
}
```

POCO::Net

Docs: <http://pocoproject.org/docs/package-Net.HTTPClient.html>

POCO 前面已经介绍过。它提供了 HTTP Client 的封装类。

10.3 浏览器引擎

WebKit

Home: <https://www.webkit.org/>

Links: [Wikipedia 维基百科](#)

它是很多浏览器使用的渲染引擎，基于 C++ 开发。

Gecko

Home: <https://developer.mozilla.org/>

Links: [Wikipedia 维基百科](#)

它是 Firefox 的渲染引擎，基于 C++ 开发，由 Mozilla 社区维护。

10.4 浏览器整合

CEF (Chromium Embedded Framework)

Home: <https://bitbucket.org/chromiumembedded/cef>

Links: [Wikipedia](#)

如其名，它提供了嵌入 Chrome 浏览器的框架。采用 C++ 开发。好几个商业公司（Google、Adobe、Facebook、Evernote...）的产品用到它。

以下是其它开源项目针对 CEF 的扩展，提供了其它编程语言的绑定。

- dotNet - <https://github.com/chillitom/CefSharp>
- dotNet (CEF1) - <https://bitbucket.org/fddima/cefglue>
- dotNet/Mono (CEF3) - <https://bitbucket.org/xilium/xilium.cefglue>

- dotNet (CEF3) - <https://bitbucket.org/chromiumfx/chromiumfx>
- Java - <https://bitbucket.org/chromiumembedded/java-cef>
- Go - <https://github.com/CzarekTomczak/cef2go>
- Delphi (CEF3) - <https://github.com/hgourvest/dcef3>

PhantomJS

Home: <http://phantomjs.org/>

Links: [Wikipedia](#)

2011年才诞生的。基于 C++ 开发，整合了 WebKit。

它本身没有提供 GUI 界面。但是提供了 JavaScript 的 API，让你可以操纵 WebKit 引擎。可以利用它进行 Web 界面的单元测试。

10.5 （其它）

WebSocket++

Home: <http://www.zaphoyd.com/websocketpp/>

顾名思义，它提供了 [WebSocket](#) 的 C++ 封装，基于 Boost Asio 构建。

支持多种操作系统平台，支持 TLS、proxy、IPv6。

11 信息安全

11.1 密码学

Crypto++

Docs: <http://www.cryptopp.com/docs/ref/annotated.html>

Crypto++ 前面已经介绍过。它提供了常见的对称加密算法（DES、AES、IDEA 等）、公钥加密算法（RSA、DSA 等）、散列算法（MD5、SHA1、RIPEMD 等）。

代码示例——计算 SHA256 散列值

```
#include <cryptopp/sha.h>
using namespace CryptoPP;

const byte* pbData = ...; // 要计算的数据的地址
unsigned int nDataLen = ...; // 字节数
byte abDigest[SHA256::DIGESTSIZE];
SHA256().CalculateDigest(abDigest, pbData, nDataLen);
// 注: abDigest 中通常会包含不可见字符，要输出为可见字符串需要做一下 Hex 编码
```

OpenSSL

Home: <https://www.openssl.org/>

Links: [Wikipedia](#) 维基百科

OpenSSL 基于 C 语言开发，在加密领域那可是大名鼎鼎。大部分常用的加密算法（对称、非对称）和散列算法，它都支持。

很多知名的软件（包括 Web Server）用到它，所以2014年的“[心脏滴血漏洞](#)”让好多网站中招。

LibreSSL

Home: <http://www.libressl.org/>

Links: [Wikipedia](#) 维基百科

OpenSSL 爆出“心脏滴血漏洞”之后，OpenBSD 社区的程序员复制了 OpenSSL 版本 1.0.1g 的代码，然后另起炉灶。

LibreSSL 的主要目标是“安全性”，其维护人员删除了原 OpenSSL 中大量过时的代码，替换了相关的内存管理函数（规避缓冲区溢出），增强了随机数生成算法....

GnuTLS

Home: <http://gnutls.org/>

Links: [Wikipedia](#)

采用 C 语言开发，名气也挺大。如其名，主要提供 TLS/SSL 的相关功能。

NaCl

Home: <http://nacl.cr.yp.to/>

Links: [Wikipedia](#)

这个名称是“Networking and Cryptography library”的缩写。

它采用 C 语言开发，另有其它编程语言（Python、Ruby、PHP）的 API 绑定。

它的作者同时也是 [qmail](#) 和 [Curve25519](#) 的作者。

libsodium

Home: <https://github.com/jedisct1/libsodium>

它派生自 NaCl，提供了跟 NaCl 兼容的 API。支持的操作系统平台更多。

shadowsocks 和 dnscrypt-proxy 用到它。

Keyczar

Home: <https://github.com/google/keyczar>

这是 Google 提供的加密库，同时提供 C++、Java、Python 三种语言的实现。

它提供了比较高层的 API，使用者无需关心太多的细节。

代码示例——加密/解密文本

```
#include <cassert>
#include <iostream>
#include <string>
#include <keyczar/keyczar.h>

void test(const std::string& key_location)
{
    keyczar::Keyczar* crypter = keyczar::Crypter::Read(key_location);
    if(!crypter)
    {
        return;
    }

    std::string plain = "Secret message";
    std::cout << "Plain text: " << plain << std::endl;

    std::string cipher;
    if(crypter->Encrypt(plain, &cipher))
    {
        std::cout << "Cipher text (Base64w): " << cipher << std::endl;
        std::string decrypted;
        if(crypter->Decrypt(cipher, &decrypted))
        {
            assert(plain == decrypted);
        }
    }
    delete crypter;
}
```

POCO::Crypto

Docs: <http://pocoproject.org/docs/Poco.Crypto.html>

POCO 前面已经介绍过。它提供了常见的加密算法和哈希算法。

12 处理文件格式

12.1 结构化数据格式

12.1.1 CSV

CSV 是一种历史悠久的结构化数据存储格式。其效果类似于一张数据库二维表。

Boost.Tokenizer

Docs: <http://boost.org/libs/tokenizer>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，用来灵活地切割字符串。使用它，可以帮你提取 CSV 的行和列。

12.1.2 JSON

JSON 格式源自 JavaScript，如今在 Web 开发中广为应用。

Boost.PropertyTree

Docs: http://boost.org/libs/property_tree

Boost 前面已经介绍过。这是 Boost 的其中一个子库，封装了某种特殊的“树”结构（property_tree）。它支持对 JSON 的读写。

代码示例——读写 JSON 字符串

```
#include <sstream>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>

using boost::property_tree::ptree;
using boost::property_tree::read_json;
using boost::property_tree::write_json;

// Write json
ptree pt1;
pt1.put("foo", "bar");
std::ostringstream oss;
write_json(oss, pt1, false);
std::string json = oss.str(); // {"foo":"bar"}
```

```
// Read json
std::istringstream iss(json);
ptree pt2;
read_json(iss, pt2);
std::string value = pt2.get<std::string>("foo");
```

POCO::JSON

Docs: <http://pocoproject.org/docs/package-JSON.JSON.html>

POCO 前面已经介绍过。它提供了 JSON 的封装类

rapidjson

Home: <https://github.com/miloyip/rapidjson>

这是一个 C++ 的 JSON 库。提供了 SAX 和 DOM 风格的 API。

（另，作者是香港同胞）

jsoncpp

Home: <https://github.com/open-source-parsers/jsoncpp>

如其名，这是个 C++ 的 JSON 封装库。

12.1.3 YAML

YAML 是一种类似于 json 的结构化数据格式。它在确保可读性的基础上，提供了超越 json 的灵活性和扩展性。

yaml-cpp

Home: <https://github.com/jbeder/yaml-cpp>

C++ 实现的 YAML 解析器。

LibYAML

Home: <http://pyyaml.org/wiki/LibYAML>

C 语言实现的 YAML 解析器。

12.2 压缩文件 & 打包文件

12.2.1 综合性的库

libarchive

Home: <http://www.libarchive.org/>

C 语言实现，支持的格式：

可读写的格式：zip、gzip、bzip2、xz、lzma、tar、ISO、cpio、ar、pax、mtree；

只读的格式：7z、rar、cab、rpm、lzh、lzop、raw、xar

LZMA SDK

Home: <http://www.7-zip.org/sdk.html>

这是由 7-zip 官方提供的。7-zip 就是用它进行压缩/解压缩。

支持的格式：7z、LZMA、LZMA2、XZ

PhysicsFS

Home: <http://icculus.org/physfs/>

Links: [Wikipedia](#)

针对不同的压缩/归档格式，提供了类似 VFS 的抽象封装层。主要用于游戏开发中。

支持的格式：zip、7z、GRP、PAK、HOG、MVL、WAD...

zopfli

Home: <https://github.com/google/zopfli>

这是由 Google 开发的 C 库，提供对 zip 和 gzip 格式的压缩（不提供解压）。

压缩的速度比较慢，但是可以得到更高的压缩率。

12.2.2 zip

[格式说明](#)

libzip

Home: <http://www.nih.at/libzip/>

Links: [Wikipedia](#)

用 C 语言开发的库，基于 zlib 库。

Poco::Zip

Docs: <http://pocoproject.org/docs/Poco.Zip.html>

POCO 前面已经介绍过。它提供了若干封装类，用于 zip 格式的压缩和解压。

12.2.3 bzip2 (bz2)

[格式说明](#)

libbzip2

Home: <http://bzip.org/>

这是 bzip2 官方提供的库，C 语言实现。

12.2.4 gzip (gz)

zlib

Home: <http://zlib.net/>

Links: [Wikipedia](#) [维基百科](#)

C 语言实现，诞生于1995年，被大量的开源项目使用（OpenSSL、OpenSSH、Apache、PostgreSQL、Git、libpng.....）。

12.2.5 tar

libtar

Home: <http://www.feep.net/libtar/>

基于 C 语言开发，可以对 tar 格式添加内容或读取内容。

12.2.6 rar

unrarlib

Home: <http://www.unrarlib.org/>

该项目的开发已经停止。只支持对 RAR2 格式的解压缩。

12.2.7 snappy

snappy

Home: <https://google.github.io/snappy/>

由 Google 开发的压缩格式，特点是非常快（不论是压缩还是解压）；但是压缩率不如 gzip。

起先被用于 Google 内部的 BigTable，如今被用于多种 NoSQL 数据库（比如：Cassandra、Hadoop、LevelDB、MongoDB、RocksDB...）

支持多种语言的绑定（C#、Common Lisp、Erlang、Go、Haskell、Lua、Java、Node.js、Perl、PHP、Python、R、Ruby、Smalltalk）

12.2.8 Brotli

Brotli

Home: <https://github.com/google/brotli>

Links: [Wikipedia](#)

由 Google 开发的压缩格式，压缩率很高（据说高于 LZMA 和 bz2）。

该算法很新，是2015年9月才发布的。

12.2.9 LZFSE

LZFSE

Home: <https://github.com/lzfse/lzfse>

Links: [维基百科](#)

由苹果开发的压缩格式。苹果称它的压缩率与“ZLib level 5”相似，但速度快2至3倍。

该算法是前不久（2016年7月）才开源出来的。

12.3 标记语言

12.3.1 XML

Expat

Home: <http://www.libexpat.org/>

Links: [Wikipedia](#)

基于 C 语言实现，诞生于1998年。很多知名的开源项目（Apache Server、Firefox、Python、PHP、Perl）用到它。

libxml2

Home: <http://xmlsoft.org/>

Links: [Wikipedia](#)

基于 C 语言实现，诞生于1999年。提供了多种语言（C++、Python、Ruby、Common Lisp、PHP、Perl）的 API 绑定。

wxWidgets

Docs: <http://docs.wxwidgets.org/trunk/groupxml.html>

wxWidgets 前面已经介绍过。它提供了 XML 的封装类，其内部是基于 Expat 进行解析。

POCO::XML

Docs: <http://pocoproject.org/docs/package-XML.XML.html>

POCO 前面已经介绍过。它提供了 XML 的封装类。

libxml++

Home: <http://libxmlplusplus.sourceforge.net/>

如其名，它是针对前面提到的 libxml2 的 C++ 封装。

12.3.2 HTML

htmlcxx

Home: <http://htmlcxx.sourceforge.net/>

如其名，是基于 C++ 开发的。支持 HTML 和 CSS 的解析。

12.4 PDF

PoDoFo

Home: <http://podofo.sourceforge.net/>

基于 C++ 开发的跨平台库，名称取自“Portable Document Format”每个单词的头两个字母：)

它既支持 PDF 文件的生成，也支持 PDF 内容的提取。它同时还提供一堆命令行的工具，用来操作 PDF 文件。

LibHaru

Home: <http://libharu.org/>

Links: [Wikipedia](#)

它是基于 C 语言开发的跨平台库，可以用来生成 PDF 文件格式。

代码示例

```
#include <hpdf.h>

// 创建文档对象
HPDF_Doc doc = HPDF_New(error_handler, NULL);
if(!doc)
{
    printf("ERROR: cannot create pdf object.\n");
    return 1;
}

// 设置文档属性
HPDF_SetCompressionMode(doc, HPDF_COMP_ALL);
HPDF_SetPageMode(doc, HPDF_PAGE_MODE_USE_OUTLINE);
HPDF_SetPassword(doc, "owner pwd", "user pwd");

// 添加一页
HPDF_Page page_1 = HPDF_AddPage(doc);
// 设置页属性
HPDF_Page_SetSize(page_1, HPDF_PAGE_SIZE_B5, HPDF_PAGE_LANDSCAPE);

// 保存到文件
HPDF_SaveToFile(doc, "test.pdf");

// 结束
HPDF_Free(doc);
```

12.5 MS Office 文档

wwWare

Home: <http://wwware.sourceforge.net/>

它能够读取 Word 文档的内容，支持的 Word 版本是（2000、97、95、6）。

AbiWord 和 KWord 用到它。

12.6 RTF

LibRTF

Home: <http://sourceforge.net/projects/librtf/>

C 语言实现的库，可以解析 RTF 文件格式。

12.7 CHM

CHMLIB

Home: <http://www.jedrea.com/chmlib/>

这是一个轻量级的库，基于 C 语言开发，可以用来提取 CHM 格式文件的内容。

它提供了多种编程语言（C++、Python、Perl、Common Lisp）的 API 绑定。

libCHMxx

Home: <http://www.mare.ee/indrek/libchmxx/>

它就是基于 CHMLIB 的 C++ 封装库。

13 图像

13.1 图像处理

ImageMagick

Home: <http://imagemagick.org/>

Links: [Wikipedia 维基百科](#)

ImageMagick 可说是最强大的开源图片处理工具集，采用 C 语言编写。诞生于1990年，其开发至今依然非常活跃。支持非常多的操作系统平台。

它提供许多编程语言的 API，对于 C++ 是 [Magick++](#)，对于 C 是 [MagickWand](#)

Boost.GIL (Generic Image Library)

Docs: <http://boost.org/libs/gil>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，实现了图像处理功能。

代码示例——调整图像尺寸

```
#include <boost/gil/image.hpp>
#include <boost/gil/typedefs.hpp>
#include <boost/gil/extension/io/jpeg_io.hpp>
#include <boost/gil/extension/numeric/sampler.hpp>
#include <boost/gil/extension/numeric/resample.hpp>

using namespace boost::gil;

rgb8_image_t img;
jpeg_read_image("input.jpg", img);

// Scale the image to 100x100 pixels using bilinear resampling
rgb8_image_t square(100, 100);
resize_view(const_view(img), view(square), bilinear_sampler());
jpeg_write_view("output.jpg", const_view(square));
```

Dlib

Docs: <http://dlib.net/imaging.html>

Dlib 前面已经介绍过。它提供了常见的图像处理功能（旋转、剪切、拉伸、过滤）。

13.2 图像格式转换

ImageMagick

ImageMagick 前面已经介绍过。它支持非常多的图片格式（[清单](#)），基本上你听说过的，它都支持。甚至包括 Postscript 和 PDF。

在支持的格式中，它可以实现其中几十种格式的相互转换。

13.3 图像渲染

Cairo

Home: <http://cairographics.org/>

Links: [Wikipedia 维基百科](#)

它提供了矢量图像的渲染功能。支持多种后端输出（Win32 GDI、OpenGL、Xlib、XCB、PDF、PNG、SVG）。

基于 C 语言开发，提供多种语言绑定（C++、Java、C#、Python、Ruby、Perl、Scheme、Smalltalk）。

cairomm

Home: <http://cairographics.org/cairomm/>

这是针对 Cairo 的 C++ 封装库。

Skia

Home: <https://github.com/google/skia>

Links: [Wikipedia 维基百科](#)

它是 Google 基于 C++ 开发的图像渲染库。支持多种后端输出（rasterization、OpenGL、PDF、SVG、SWF）。

原先由 Skia 公司开发，后来该公司被 Google 收购。被用于 Android、Chrome、Chrome OS、Firefox 等知名开源项目。

PBRT (Physically Based Rendering Toolkit)

Home: <http://pbrt.org/>

基于光线追踪的物理渲染系统，采用 C++ 开发。

13.4 计算机视觉

OpenCV

Home: <http://opencv.org/>

Links: [Wikipedia 维基百科](#)

它是一个跨平台的计算机视觉库，由 Intel 发起并参与开发。开发语言是 C 和 C++。

提供其它编程语言（Python、Java、MATLAB/OCTAVE ...）的 API 绑定。

14 多媒体

14.1 多媒体框架

FFmpeg

Home: <http://ffmpeg.org/>

Links: [Wikipedia 维基百科](#)

名气非常大的开源多媒体框架，基于 C 和汇编开发，支持多种操作系统。

另外，该开源项目还提供了若干命令行工具，包含了一些辅助功能。

- ffmpeg 格式转换工具
- ffplay 简化版的播放器
- ffserver 流媒体服务器
- ffprobe 显示多媒体文件信息

几个知名的开源播放器（VLC、MPC-HC、xine）用到它，Google Chrome 也用到它。

Libav

Home: <http://libav.org/>

Links: [Wikipedia](#)

它是2011年从 FFmpeg 派生出来的。基于 C 语言开发，支持多种操作系统。

14.2 视频库

libavcodec

Home: <http://ffmpeg.org/>

它来自于 FFmpeg 社区，基于 C 语言实现，提供了多种视频格式和音频格式的编码/解码功能。

由于 Libav 从 FFmpeg 分裂出来，Libav 下也带有一个同名的库。

14.3 音频库

PortMedia & PortAudio

Home: <http://www.portaudio.com/>

Links: [Wikipedia](#)

PortAudio 是 PortMedia 的组成部分，提供了音频的播放和录制功能。支持多种底层 API（ALSA、DirectSound、WASAPI、ASIO...）

OpenAL

Home: <http://www.openal.org/>

Links: [Wikipedia](#) 维基百科

C 语言开发的 3D 音效库，跨平台。最初由 Loki Software 开发。Loki 倒闭以后，这个项目由开源社区继续维护。

15 游戏

15.1 综合性的游戏引擎

id Tech 系列

Links: [Wikipedia](#)

这个系列来自于大名鼎鼎的 [id Software 公司](#)，由同样大名鼎鼎约翰·卡马克打造。

第一代诞生于1993年，是 DOS 时代的经典。

原先基于 C 和 汇编开发，从 id Tech 4 开始改用 C++ 开发。

- id Tech 1——俗称：Doom 引擎
- id Tech 2（Quake）——俗称：Quake 引擎
- id Tech 2（Quake II）——俗称：Quake II 引擎
- id Tech 3——俗称：Quake III 引擎
- id Tech 4——俗称：Doom 3 引擎

Crystal Space

Home: <http://www.crystalspace3d.org/>

Links: [Wikipedia](#)

以 C++ 编写，功能包括：2D 和 3D 渲染、音效、AI... 它的物理引擎基于 ODE 和 Bullet

Blender Game Engine

Home: <http://www.blender.org/>

Links: [Wikipedia](#) 维基百科

它是 **Blender** 的组成部分，以 C++ 编写，使用 Python 脚本扩展。功能包括：3D 渲染、碰撞检测、角色编辑器、音效、网络通讯、AI、...

Panda3D

Home: <http://www.panda3d.org/>

Links: [Wikipedia](#)

以 C++ 编写，用 Python 脚本扩展。虽然它的名字有“3D”，但它不仅仅是 3D 引擎，还包括了其它功能（碰撞检测、音效、关卡编辑器...）。

15.2 3D 渲染引擎

OGRE

Home: <http://www.ogre3d.org/>

Links: [Wikipedia](#) 维基百科

著名的 3D 渲染引擎，C++ 开发，诞生于2005年。支持很多操作系统（包括两大手机操作系统）。很多商业游戏用到它。

支持其它编程语言（Python、Ruby、Perl）的 API 绑定。支持 JVM 和 dotNet 平台。

Mesa 3D

Home: <http://mesa3d.org/>

Links: [Wikipedia](#) 维基百科

使用 C 语言开发，它是针对 OpenGL 规范的【纯软件】实现（大部分 OpenGL 的实现都用到了显卡硬件）。

15.3 物理引擎

Bullet

Home: <http://www.bulletphysics.org/>

Links: [Wikipedia](#) 维基百科

采用 C 和 C++ 开发。电影《2012》用到它，游戏“侠盗猎车手”、“荒野大镖客”用到它。

Box2D

Home: <http://www.box2d.org/>

Links: [Wikipedia](#) 维基百科

基于 C++ 开发的2维物理引擎。“愤怒的小鸟”用到它。

ODE (Open Dynamics Engine)

Home: <http://www.ode.org/>

Links: [Wikipedia](#) 维基百科

诞生于2001年，采用 C 和 C++ 开发。

Newton Game Dynamics

Home: <http://www.newtondynamics.com/>

Links: [Wikipedia](#)

基于 C++ 开发。

16 数值运算 & 科学计算

16.1 综合性的库

GSL (GNU Scientific Library)

Home: <https://www.gnu.org/software/gsl/>

Links: [Wikipedia](#)

由 GNU 官方提供, 包括: 复数、多项式、矩阵、线性代数、特征向量、快速傅里叶变换、统计、模拟退火.....

代码示例——贝塞尔函数

```
#include <stdio.h>
#include <gsl/gsl_sf_bessel.h>

double x = 5.0;
double y = gsl_sf_bessel_J0(x);
printf("J0(%g) = %.18e\n", x, y);
```

16.2 有理数

Boost.Rational

Docs: <http://boost.org/libs/rational>

Boost 前面已经介绍过。这是 Boost 的其中一个子库, 提供了“有理数”的功能。

16.3 高精度数值运算

GMP (GNU Multiple Precision)

Home: <https://gmplib.org/>

Links: [Wikipedia](#) [维基百科](#)

基于 C 语言的高精度数值运算库, 诞生于1991年, 非常老牌。

代码示例——大整数相乘

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

mpz_t x,y,result;

mpz_init_set_str(x, "7612058254738945", 10);
mpz_init_set_str(y, "9263591128439081", 10);
mpz_init(result);

mpz_mul(result, x, y);
gmp_printf("%Zd\n", result);

/* free used memory */
mpz_clear(x);
mpz_clear(y);
mpz_clear(result);
```

Boost.Multiprecision

Docs: <http://boost.org/libs/multiprecision>

Boost 前面已经介绍过。这是 Boost 的其中一个子库, 实现了高精度数值运算。它还提供了针对 GMP 的数据类型的封装。

16.4 矩阵

Boost.uBLAS.Matrix

Docs: <http://boost.org/libs/numeric/ublas/doc/matrix.html>

Boost 前面已经介绍过。这是 Boost 提供的矩阵模板类。

代码示例

```
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>

using namespace boost::numeric::ublas;

matrix<double> m(3, 3);
for(unsigned i=0; i<m.size1(); i++)
    for(unsigned j=0; j<m.size2(); j++)
        m(i, j) = 3 * i + j;

std::cout << m << std::endl;
```

Dlib

Docs: http://dlib.net/linear_algebra.html#matrix

Dlib 前面已经介绍过。它提供了一个矩阵类。

16.5 线性代数

Boost.uBLAS

Docs: <http://boost.org/libs/numeric/ublas>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，实现了 BLAS 的1、2、3级。

代码示例——计算矩阵与矢量的乘积

```
#include <iostream>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>

using namespace std;
using namespace boost::numeric::ublas;

vector<double> v(2);
v(0) = 1; v(1) = 2;

matrix<double> m(2,2);
m(0,0) = 0; m(0,1) = 1;
m(1,0) = 2; m(1,1) = 3;

vector<double> v2 = prod(m, v);
cout << v2 << endl;
```

Blitz++

Home: <http://blitz.sourceforge.net/>

Links: [Wikipedia](#)

它是基于 C++ 实现的。其特色是：采用“模板元编程”的技术进行编译时计算，从而优化了性能。

Armadillo

Home: <http://arma.sourceforge.net/>

Links: [Wikipedia](#)

类似 Blitz++，Armadillo 也用了“模板元编程”的技术。

代码示例

```
#include <iostream>
#include <armadillo>

using namespace std;
using namespace arma;

vec v;
v << 2.0 << 5.0 << 2.0;

// endr represents the end of a row in a matrix
mat m;
m << 1.0 << 2.0 << endr
  << 2.0 << 3.0 << endr
  << 1.0 << 3.0 << endr;

cout << "Least squares solution:" << endl << solve(m,v) << endl;
```

Dlib

Docs: http://dlib.net/linear_algebra.html

Dlib 前面已经介绍过。它提供了线性代数相关的封装类。

17 跨语言编程

17.1 整合多种语言的库

SWIG

Home: <http://swig.org/>

Links: [Wikipedia](#)

这是一个很老牌的、有名气的工具，它可以把多种语言（Java、Python、C#、Ruby、PHP、Perl、Lua、Go）整合到 C/C++ 中。

整合之后，你的 C/C++ 程序就可以享受到其它这些语言的特性啦，非常爽！

17.2 整合单一语言的库

17.2.1 整合 Python 语言

Boost.Python

Docs: <http://boost.org/libs/python>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，实现了 C++ 代码和 Python 代码的互操作。

代码示例——Hello world

```
// 这是一个标准的 C 函数
const char* greet()
{
    return "Hello, world";
}

// 使用如下代码对上述函数进行包装
#include <boost/python.hpp>
BOOST_PYTHON_MODULE(hello_ext)
{
    using namespace boost::python;
    def("greet", greet);
}
```

```
// 以下是调用该模块的 Python 代码
// import hello_ext
// print(hello_ext.greet())
```

18 （其它）

一些不方便归类的，暂时放到这里。

18.1 词法分析 & 语法分析

Boost.Spirit

Docs: <http://boost.org/libs/spirit>

Boost 前面已经介绍过。这是 Boost 的其中一个子库，提供了“基于 EBNF 的解析器框架”。