



C++ 基础

第 7 章：深入 IO

主讲人 李伟

微软高级工程师

《C++ 模板元编程实战》作者





目录



1. IOStream 概述



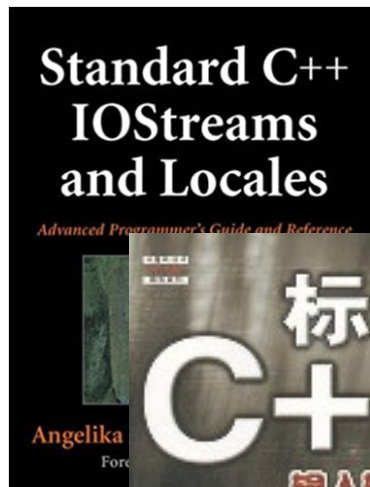
2. 输入与输出



3. 文件与内存操作



4. 流的状态、定位与同步



标准 C++ IOStreams 与 Locales

Angelika Langer & Klaus Kreft 著
李伟译 (V2.0)

2015 年 6 月 20 日



IOStream 概述

- IOStream 采用流式 I/O 而非记录 I/O ，但可以在此基础上引入结构信息
- 所处理的两个主要问题
 - 表示形式的变化：使用格式化 / 解析在数据的内部表示与字符序列间转换
 - 与外部设备的通信：针对不同的外部设备（终端、文件、内存）引入不同的处理逻辑
- 所涉及到的操作
 - 格式化 / 解析
 - 缓存
 - 编码转换
 - 传输
- 采用模板来封装字符特性，采用继承来封装设备特性
 - 常用的类型实际上是类模板实例化的结果



输入与输出

- 输入与输出分为格式化与非格式化两类
- 非格式化 I/O：不涉及数据表示形式的变化
 - 常用输入函数：get / read / getline / gcount
 - 常用输出函数：put / write
- 格式化 I/O：使用移位操作符来进行的输入 (>>) 与输出 (<<)
 - C++ 通过操作符重载以支持内建数据类型的格式化 I/O
 - 可以通过重载操作符以支持自定义类型的格式化 I/O
- 格式控制
 - 可接收位掩码类型（showpos）、字符类型（fill）与取值相对随意（width）的格式化参数
 - 注意 width 方法的特殊性：触发后被重置



输入与输出（续）

- 操纵符
 - 简化格式化参数的设置
 - 触发实际的插入与提取操作
- 提取会放松对格式的限制
- 提取 C 风格字符串时要小心内存越界



文件与内存操作

- 文件操作
 - `basic_ifstream` / `basic_ofstream` / `basic_fstream`
 - 文件流可以处于打开 / 关闭两种状态，处于打开状态时无法再次打开，只有打开时才能 I/O
- 文件流的打开模式（图引自 C++ IOSTream 一书）
 - 每种文件流都有缺省的打开方式
 - 注意 `ate` 与 `app` 的异同
 - `binary` 能禁止系统特定的转换
 - 避免意义不明确的流使用方式（如 `ifstream + out`）

标记名	作用
<code>in</code>	打开以供读取
<code>out</code>	打开以供写入
<code>ate</code>	表示起始位置位于文件末尾
<code>app</code>	附加文件，即总是向文件尾写入
<code>trunc</code>	截断文件，即删除文件中的内容
<code>binary</code>	二进制模式



文件与内存操作

- 合法的打开方式组合（引自 C++ IOStream 一书）

打开方式	效果	加结尾模式标记	加二进制模式标记
<code>in</code>	只读方式打开文本文件	初始文件位置位于文件末尾	禁止系统转换
<code>out trunc</code> <code>out</code>	如果文件存在，将长度截断为 0； 否则建立文件供写入	初始文件位置位于文件末尾	禁止系统转换
<code>out app</code>	附加；打开或建立文本文件， 仅供文件末尾写入	初始文件位置位于文件末尾	禁止系统转换
<code>in out</code>	打开文本文件供更新使用（支持读写）	初始文件位置位于文件末尾	禁止系统转换
<code>in out trunc</code>	如果文件存在，将长度截断为 0； 否则建立文件供更新使用	初始文件位置位于文件末尾	禁止系统转换



文件与内存操作

- 内存流： `basic_istream` / `basic_ostream` / `basic_stringstream`
- 也会受打开模式： `in` / `out` / `ate` / `app` 的影响
- 使用 `str()` 方法获取底层所对应的字符串
 - 小心避免使用 `str().c_str()` 的形式获取 C 风格字符串
- 基于字符串流的字符串拼接优化操作



流的状态

- iostate
 - failbit / badbit / eofbit / goodbit
- 检测流的状态
 - good() / fail() / bad() / eof() 方法
 - 转换为 bool 值（参考cppreference）
- 注意
 - 转换为 bool 值时不会考虑 eof
 - fail 与 eof 可能会被同时设置，但二者含意不同
- 通常来说，只要流处于某种错误状态时，插入 / 提取操作就不会生效



流的状态（续）

- 设置流状态
 - clear：设置流的状态为具体的值（缺省为 goodbit）
 - setstate：将某个状态附加到现有的流状态上
- 捕获流异常：exceptions方法F



流的定位

- 获取流位置
 - `tellg()` / `tellp()` 可以用于获取输入 / 输出流位置 (`pos_type` 类型)
 - 两个方法可能会失败, 此时返回 `pos_type(-1)`
- 设置流位置
 - `seekg()` / `seekp()` 用于设置输入 / 输出流的位置
 - 这两个方法分别有两个重载版本:
 - 设置绝对位置: 传入 `pos_type` 进行设置
 - 设置相对位置: 通过偏移量 (字符个数 `ios_base::beg`) + 流位置符号的方式设置
 - `ios_base::beg`
 - `ios_base::cur`
 - `ios_base::end`



流的同步

- 基于 `flush()` / `sync()` / `unitbuf` 的同步
 - `flush()` 用于输出流同步，刷新缓冲区
 - `sync()` 用于输入流同步，其实现逻辑是编译器所定义的
 - 输出流可以通过设置 `unitbuf` 来保证每次输出后自动同步
- 基于绑定 (tie) 的同步
 - 流可以绑定到一个输出流上，这样在每次输入 / 输出前可以刷新输出流的缓冲区
 - 比如： `cin` 绑定到了 `cout` 上
- 与 C 语言标准 IO 库的同步
 - 缺省情况下，C++ 的输入输出操作会与 C 的输入输出函数同步
 - 可以通过 `sync_with_stdio` 关闭该同步

感谢聆听 !
Thanks for Listening

