CRT 库功能

2020/09/03 •      +2

本文内容

C 运行时库 (CRT)

C++ 标准库

如果应用程序使用多个 CRT 版本，将存在什么问题?

另请参阅

本主题将介绍各种 .lib 文件，包括 C 运行时库及其关联的编译器选项和预处理器指令。

C 运行时库 (CRT)

C 运行时库 (CRT) 是集成了 ISO C99 标准库的 C++ 标准库。 实现 CRT 的 Visual C++ 库支持用于 .NET 开发的本机代码开发以及本机和托管混合代码。 所有版本的 CRT 都支持多线程开发。 大多数的库都支持通过静态链接将库直接链接到代码中，或通过动态链接让代码使用常用 DLL 文件。

从 Visual Studio 2015 开始，CRT 已被重构为新的二进制文件。 通用 CRT (UCRT) 包含通过标准 C99 CRT 库导出的函数和全局函数。 UCRT 现为 Windows 组件，并作为 Windows 10 的一部分提供。 静态库、DLL 导入库和 UCRT 的头文件现在 Windows 10 SDK 中提供。 安装 Visual C++ 时，Visual Studio 安装程序将安装使用 UCRT 所需 Windows 10 SDK 的子集。 可以在 Visual Studio 2015 及更高版本支持的任何 Windows 版本上使用 UCRT。 可以使用 vcredist 重新分发它，以便支持 Windows 10 以外的 Windows 版本。 有关详细信息，请参阅重新分发 Visual C++ 文件。

下表列出了实现 UCRT 的库。

库	关联的 DLL	特征	选项	预处理器指令
libucrt.lib	无	将 UCRT 静态链接到你的代码。	**/MT**	_MT
libucrtd.lib	无	用于静态链接的 UCRT 调试版本。 不可再发行。	**/MTd**	_DEBUG, _MT
ucrt.lib	*ucrtbase.dll*	UCRT 的 DLL 导入库。	**/MD**	_MT, _DLL
ucrtd.lib	*ucrtbased.dll*	UCRT 调试版本的 DLL 导入库。 不可再发行。	**/MDd**	_DEBUG, _MT, _DLL

vcruntime 库包含 Visual C++ CRT 实现特定的代码，例如异常处理和调试支持、运行时

检查和类型信息、实现的详细信息和某些扩展的库函数。 此库特定于所用编译器的版本。

此表列出了实现 vcruntime 库的库。

库	关联的 DLL	特征	选项	预处理器指令
libvcruntime.lib	无	静态链接到你的代码。	**/MT**	_MT
libvcruntimed.lib	无	用于静态链接的调试版本。 不可再发行。	**/MTd**	_MT, _DEBUG
vcruntime.lib	*vcruntime<version>.dll*	vcruntime 的 DLL 导入库。	**/MD**	_MT, _DLL
vcruntimed.lib	*vcruntime<version>d.dll*	调试 vcruntime 的 DLL 导入库。 不可再发行。	**/MDd**	_DEBUG, _MT, _DLL

> **备注**
>
> 当发生 UCRT 重构时，并发运行时的函数将移入 *concrt140.dll* ，这已添加到 c + + 可再发行组件包。 此 DLL 是 C++ 并行容器和算法（如 `concurrency::parallel_for`）所必需的。 此外，C++ 标准库需要 Windows XP 版的此 DLL 来支持同步基元，因为 Windows XP 不具有条件变量。

初始化 CRT 的代码是几个库中的一个，根据 CRT 库是采用静态或动态链接还是本机、托管或混合代码而定。 此代码处理 CRT 启动、内部逐线程数据初始化和终止。 它特定于所用编译器的版本。 此库始终采用动态链接，即使使用动态链接的 UCRT 也是如此。

此表列出了实现 CRT 初始化和终止的库。

库	特征	选项	预处理器指令
libcmt.lib	将本机 CRT 启动静态链接到你的代码。	**/MT**	_MT
libcmtd.lib	静态链接本机 CRT 启动的调试版本。 不可再发行。	**/MTd**	_DEBUG, _MT

库	特征	选项	预处理器指令

msvcrt.lib	与 DLL UCRT 和 vcruntime 一起使用的本机 CRT 启动的静态库。	**/MD**	_MT, _DLL
msvcrtd.lib	与 DLL UCRT 和 vcruntime 一起使用的本机 CRT 启动调试版本的静态库。 不可再发行。	**/MDd**	_DEBUG, _MT, _DLL
msvcmrt.lib	与 DLL UCRT 和 vcruntime 一起使用的本机和托管混合 CRT 启动的静态库。	**/clr**	
msvcmrtd.lib	与 DLL UCRT 和 vcruntime 一起使用的本机和托管混合 CRT 启动调试版本的静态库。 不可再发行。	**/clr**	
msvcurt.lib	纯托管 CRT 的**已弃用**静态库。	**/clr:pure**	
msvcurtd.lib	纯托管 CRT 调试版本的**已弃用**静态库。 不可再发行。	**/clr:pure**	

如果在没有指定 C 运行时库的编译器选项的情况下从命令行链接程序，则链接器将使用静态链接的 CRT 库： *libcmt.lib* 、 *libvcruntime.lib* 和 *libucrt.lib* 。

使用静态链接的 CRT 意味着由 C 运行时库保存的任何状态信息对于 CRT 的该实例而言是本地的。 例如，如果在 strtok 使用静态链接的 CRT 时使用，则分析器的位置与 `strtok` `strtok` 同一进程中的代码中使用的状态无关 (但在不同的 DLL 或 EXE) 中，后者链接到静态 CRT 的另一个实例。 相反，动态链接的 CRT 可共享动态链接到 CRT 的进程中的所有代码的状态。 如果使用这些函数新的更安全版本，这一问题则不适用；例如， `strtok_s` 就不存在此问题。

由于通过链接到静态 CRT 构建的 DLL 将具有其自己的 CRT 状态，因此不建议以静态方式链接到 DLL 中的 CRT，除非特别需要和需了解这一后果。 例如，如果 _set_se_translator 在加载 dll （链接到其自己的静态 CRT）的可执行文件中调用，则该转换器将不会捕获由 dll 中的代码生成的任何硬件异常，但会捕获由主可执行文件中的代码生成的硬件异常。

如果使用 **/clr** 编译器开关，则代码将与静态库 msvcmrt.lib 链接。 静态库将提供托管的代码和本机 CRT 之间的代理。 不能使用静态链接的 CRT (**/MT** 或 **/MTd** 与) 的选项 **/clr** 。 改用动态链接的库 (**/MD** 或 **/MDd**)。 纯托管的 CRT 库在 Visual Studio 2015 中已弃用并在 Visual Studio 2017 中不受支持。

有关将 CRT 与结合使用的详细信息 **/clr** ， 请参阅 Mixed (本机和托管) 程序集。

若要生成应用程序的调试版本， _DEBUG 必须定义该标志，并且该应用程序必须与其中

一个库的调试版本链接。 有关使用调试版本的库文件的详细信息，请参阅 CRT 调试技术。

此版本的 CRT 不完全符合 C99 标准。 在 Visual Studio 2019 版本16.8 之前的版本中，<tgmath.h> 不支持标头。 在所有版本中， `CX_LIMITED_RANGE` `FP_CONTRACT` 不支持和 pragma 宏。 某些元素（如标准 IO 函数中参数说明符的含义）默认采用旧的解释。 您可以使用 `/Zc` 编译器一致性选项并指定链接器选项来控制库一致性的某些方面。

C++ 标准库

C++ 标准库	特征	选项	预处理器指令
libcpmt.lib	多线程, 静态链接	**/MT**	_MT
msvcprt.lib	多线程、动态链接 () 的导入库 *msvcp<version>.dll*	**/MD**	_MT, _DLL
libcpmtd.lib	多线程, 静态链接	**/MTd**	_DEBUG, _MT
msvcprtd.lib	多线程、动态链接 () 的导入库 *msvcp<version>d.dll*	**/MDd**	_DEBUG, _MT, _DLL

当你生成项目的发行版时，默认情况下， (会链接) 的一个基本 C 运行时库 *libcmt.lib* *msvcmrt.lib* *msvcrt.lib* ，具体取决于你选择的编译器选项 (多线程、DLL、 **/clr**) 。 如果在代码中包含其中一个 C++ 标准库标头文件，则将在编译时通过 Visual C++ 自动链接 C++ 标准库。 例如：

C++ 复制

```
#include <ios>
```

对于二进制兼容性，可通过单个导入库指定多个 DLL 文件。 版本更新可能会引入点库，点库是引入新的库功能的 单独的 Dll**。 例如，Visual Studio 2017 版本15.6 引入 *msvcp140_1.dll* 了支持附加的标准库功能，而不会破坏支持的 ABI *msvcp140.dll* 。 *msvcprt.lib* Visual Studio 2017 版本15.6 中包含的导入库支持这两个 dll，此版本的 vcredist 安装这两个 dll。 传送后，点库具有固定 ABI，且不会在以后的点库中包含依赖项。

如果应用程序使用多个 CRT 版本，将存在什么问题?

每个可执行映像（EXE 或 DLL）可以具有其自己静态链接的 CRT，或可以动态链接到

CRT。 静态包括在某个映像中或某个映像动态加载的 CRT 版本取决于构建 该 CRT 时采用的工具和库。 单个进程可能会加载多个 EXE 和 DLL 映像，每个都有其自己的 CRT。 每个 CRT 可能使用不同的分配器，可能具有不同的内部结构布局，可能使用不同的存储排列方式。 这意味着，分配的内存、CRT 资源或跨 DLL 边界传递的类可能会导致内存管理、内部静态使用情况或布局解释方面的问题。 例如，如果在一个 DLL 中分配类，但将其传递给另一个 DLL 或由另一个 DLL 删除，那么使用了哪个 CRT 释放器？ 导致的错误程度可以从微小到立即致命，因此强烈建议不要直接传输此类资源。

你可以使用应用程序二进制接口 (ABI) 技术避免这些问题，因为此技术被设计成稳定且版本可控。 设计 DLL 导出接口以按值传递信息，或致力于调用方传入而非本地分配并返回给调用方的内存。 使用封送处理技术在可执行映像之间复制结构化数据。 本地封装资源并仅允许通过向客户端公开的句柄或函数操作。

如果进程中的所有映像全都使用相同的 CRT 动态加载版本，则也有可能避免这些问题。 若要确保所有组件都使用同一 DLL 版本的 CRT，请使用选项生成它们， `/MD` 并使用相同的编译器工具集和属性设置。

如果程序跨 DLL 边界传递某些 CRT 资源，请务必小心。 即使使用相同版本的 CRT，文件句柄、区域设置和环境变量等资源也可能导致问题。 有关所涉及问题以及如何解决这些问题的详细信息，请参阅跨 DLL 边界传递 CRT 对象时可能的错误。

另请参阅

- C 运行时库参考

此页面有帮助吗?