

---

qpdf

版本 12.3.2

杰伊·伯肯比尔特

2026年1月24日



目录：

1	什么是 qpdf?	3
2	许可	5
3	下载 qpdf	7
4	构建与安装 qpdf	9
4.1	依赖关系	9
4.2	构建说明	10
4.3	构建选项	11
4.4	加密服务提供商	15
4.5	支持 zopfli 的建设	16
4.6	从 autoconf 转换为 cmake	16
5	包装员笔记	19
5.1	构建选项	19
5.2	软件包测试	19
5.3	包装文档	20
6	运行 qpdf	21
6.1	基本祈求	21
6.2	退出现状	22
6.3	壳体完备	22
6.4	求助/信息	23
6.5	一般期权	23
6.6	高级控制选项	25
6.7	PDF 转换	26
6.8	佩奇区	31
6.9	PDF 修改	31
6.10	加密	36
6.11	页面选择	40
6.12	叠加与底层	43
6.13	嵌入文件/附件	44
6.14	PDF 检查	46
6.15	JSON 选项	48
6.16	全球期权	49
6.17	测试或调试选项	50
6.18	Unicode 密码	51
6.19	优化文件大小	53
6.20	Zopfli 压缩算法	53

7 QDF 模式	55
8 使用 qpdf 库	57
8.1 使用 C 语言中的 qpdf	57
8.2 使用其他语言的 qpdf	57
8.3 关于 Unicode 文件名的说明	58
9 弱密码学	59
9.1 弱密码算法的定义	59
9.2 弱加密在 qpdf 中的应用	60
9.3 弱哈希在 qpdf 中的应用	60
9.4 qpdf 11.0 中的 API 破坏性变更	60
10 qpdf JSON	63
10.1 概述	63
10.2 JSON 术语	63
10.3 qpdf JSON 不是什么	64
10.4 qpdf JSON 格式	64
10.5 JSON 兼容性保证	71
10.6 JSON: 特殊考虑	72
10.7 从 JSON v1 到 v2 的变更	72
11 对 qpdf 的贡献	75
11.1 源代码库	75
11.2 代码格式化	75
11.3 自动化测试	75
11.4 个人评论	77
12 设计与图书馆注释	79
12.1 简介	79
12.2 检查模式	79
12.3 设计目标	80
12.4 辅助职业	81
12.5 实现说明	82
12.6 qpdf 对象内部结构	83
12.7 选角政策	84
12.8 加密	85
12.9 随机数生成	85
12.10 新增和删除页面	86
12.11 预留对象编号	86
12.12 从其他 PDF 文件复制对象	86
12.13 编写 PDF 文件	86
12.14 过滤流	87
12.15 对象访问器方法	88
12.16 智能指针	89
13 QPDFJob: 基于岗位的界面	95
13.1 QPDFJob 设计	97
14 线性化	99
14.1 线性化的基本策略	99
14.2 准备线性化	99
14.3 优化	99
14.4 写线性化文件	100
14.5 线性化数据的计算	100



14.6 线性化的已知问题	100
14.7 调试说明	101
15 客体与交叉引用流 103	
15.1 对象流	103
15.2 交叉引用流	104
15.3 线性化文件的启示	104
15.4 实现说明	105
16 PDF 加密	107
16.1 PDF 加密概念	107
16.2 PDF 加密详情	108
16.3 PDF 安全限制	109
16.4 qpdf 如何处理安全限制	110
16.5 用户密码和所有者密码	111
17 发布说明	113
18 致谢	163
19 个指数	165
qpdf 命令行选项	167 iii



欢迎来到 qpdf 文档！如需获取本文档的最新版本，请访问 <https://qpdf.readthedocs.io>。

I.



## 什么是 QPDF?

qpdf 是一个用于结构化、内容保留的 PDF 文件转换的程序和 C++ 库。QPDF 的网站位于 <https://qpdf.sourceforge.io/>。QPDF 的源代码托管在 GitHub at <https://github.com/qpdf/qpdf>。您可以在 <https://qpdf.readthedocs.io/> 找到最新版本的文档。

QPDF 为 PDF 生成软件的开发者，或仅仅想了解 PDF 文件内部结构的人提供了许多实用功能。通过 qpdf，可以将一个 PDF 文件中的对象复制到另一个 PDF 文件，并作 PDF 文件中的页面列表。这使得合并和拆分 PDF 文件成为可能。qpdf 库还让你可以从零开始创建 PDF 文件。在此模式下，你负责提供文件的所有内容，而 qpdf 库负责对象的所有语法表示、交叉引用表的创建，以及如果使用它们，还包括对象流、加密、线性化及其他语法细节。你仍然需要自己负责生成 PDF 内容。

QPDF 设计时外部依赖极少，并且有意非常轻量化。qpdf 不是 PDF 内容创建库、PDF 查看器，也不是能够将 PDF 转换成其他格式的程序。特别是，qpdf 对 PDF 内容流的语义一无所知。如果你在找能做到这一点的，应该考虑别的。不过，一旦你有了有效的 PDF 文件，qpdf 可以用来转换该文件，而这些转换可能是你原有的 PDF 制作工具无法处理的。例如，许多程序生成简单的 PDF 文件，但无法对它们进行密码保护、网页优化或执行类似的转换。

这份文档旨在全面，但也有维基专门收录不够完善的材料和正在进行的工作。



## 许可证

qpdf 采用 Apache 许可证 2.0 版本（“许可证”）授权。除非适用法律要求或书面协议，否则根据许可协议分发的软件均以“现状”方式分发，不作任何明示或暗示的保证或条件。有关许可和限制的具体条款，请参见许可证。





## 下载 QPDF

qpdf 被大多数 Linux 发行版包含。许多其他作系统也有原生包可用。

其他资源：

- [GitHub 发布页面](#)
- [GitHub 项目](#)
- [QPDF 项目网站](#)



## 构建与安装 QPDF

本章介绍如何构建和安装 qpdf。有关安装预构建的 qpdf 副本的信息，请参见[README.md] (<https://github.com/qpdf/qpdf/blob/main/README.md>)，这是仓库和源代码发行版中的顶层 README.md。

### 4.1 依赖关系

QPDF 几乎没有外部依赖。本节描述了在不同情况下构建 qpdf 所需的内容。

#### 4.1.1 基本依赖

- 一个支持 C++-20 的 C++ 编译器
- CMake 3.16 或更高版本
- ZLIB 或兼容的 ZLIB 实现
- 一个兼容 libjpeg 的库，如 jpeg 或 libjpeg-turbo
- 推荐但非必需：使用 gnutls 才能使用 gnutls 加密服务，和/或 openssl 才能使用 openssl 加密服务

- 如果指定了 ZOPFLI 构建选项（默认关闭），则使用 zopfli 库。

qpdf 源代码树包含一些自动生成的文件。代码生成器使用 Python 3。自动代码生成默认是关闭的。讨论内容请参考构建选项。

#### 4.1.2 测试依赖

QPDF 的测试套件由 Ctest 运行，Ctest 是 CMake 的一部分，但测试本身是通过 Qtest 的嵌入式副本实现的，QTEST 是用 Perl 实现的。在 Windows 上，MSYS2 的 Perl 是已知有效的。

qtest 需要 GNU diffutils 或其他支持 diff -u 的差分。默认的 diff 命令在 GNU/Linux 和 MacOS 上都能使用。

qpdf 测试套件的一部分通过将 PDF 文件转换成图片并进行比较，进行内容比较。图像比较测试默认是被禁用的。这些测试并非确定 qpdf 构建正确性的必要条件，因为测试套件还包含了实际比较的预期输出文件。图像比较测试提供了额外的检查，确保任何内容转换不会破坏页面渲染。影响内容流本身的转换默认关闭，仅提供帮助开发者查看 PDF 文件内容。如果你对库做了深度修改，导致 qpdf 生成的文件内容发生变化，那么你应该启用图像比较测试。在运行测试前，将 QPDF\_TEST\_COMPARE\_IMAGES 环境变量设置为 1 来启用它们。图像比较测试增加了以下额外要求：

- libtiff 命令行工具

- GhostScript 8.60 或更新版本

注意：在 qpdf 11 之前，qpdf.test 中启用了图像比较测试，必须将 QPDF\_SKIP\_TEST\_COMPARE\_IMAGES 设为 1 来禁用。这是 ./configure 自动完成的。现在你必须通过设置环境变量来启用图像比较测试。这次改动是因为开发者现在必须自己设置环境变量，而不是通过构建来设置。无论哪种情况，默认都是关闭的。

### 4.1.3 维护者依赖

- 作为维护者要运行 ABI 检查，你需要 castxml，check\_abi 用它生成所有公共类的大小。

### 4.1.4 Windows 上的额外需求

- 用 Visual Studio 构建 qpdf 时，使用默认的 cmake 选项时没有额外要求。  
你可以从 Visual C++ 命令行 shell 构建 qpdf。
- 使用 mingw 构建建议 MSYS2 搭配 mingw32 和/或 mingw64 工具链。你也可以在 MSYS2 环境下用 MSVC 构建。
- qpdf 的测试套件可以在 MSYS2 环境中运行，适用于基于 mingw 和 MSVC 的构建。

更多说明请参见源分布中的 README-windows.md。

### 4.1.5 建筑文档要求

qpdf 手册采用重构文本编写，使用 Sphinx 构建，使用 Read the Docs Sphinx 主题。4.3.2 版本之前的 Sphinx 可能无法正常工作。Sphinx 需要 Python 3。要从源码构建 HTML 文档，你需要安装 Sphinx 和主题，通常可以用 `pip install sphinx sphinx_rtd_theme` 完成。要构建 PDF 版本的文档，你需要 pdflatex、latexmk，以及相当完整的 LaTeX 安装。详细需求可在 Sphinx 文档中找到。要了解 qpdf 发行版的文档构建方式，请参阅 qpdf 源代码发行版中的 `build-scripts/build-doc` 文件。

## 4.2 构建说明

从 qpdf 11 开始，qpdf 是用 CMake 构建的。

### 4.2.1 基础构建调用

qpdf 以普通方式使用 cmake，有关如何运行 cmake 的详细信息，请参阅 CMake 文档。以下是简要总结。

你通常可以直接跑

```
cmake -S . -B 型
cmake --build build
```

如果你使用像 MSVC 这样的多配置生成器，应该将 `--config`（其中是 Release、Debug、RelWithDebInfo 或 MinSizeRel，如 CMake 文档中所述）传递给 build 命令。如果你运行的是单一配置生成器，比如 Linux 或 MSYS 默认的 Makefile 生成器，你可能需要将 `-DCMAKE_BUILD_TYPE=` 传递给原始的 cmake 命令。

运行 `ctest` 来运行测试套件。由于实际测试是用 `qtest` 实现的，你需要从 `--verbose` 转为 `cmake`，这样你才能看到各个测试的输出。否则，你会看到少量 `ctest` 命令运行时间非常长。如果你只想在特定测试套件中运行特定的测试文件，可以设置 TESTS 环境变量（`qtest-driver` 使用），并将 `-R` 参数传递给 `ctest`。例如：

```
TESTS=qutil ctest --verbose -R libtests
```

只会运行 Libtests 测试套件中的 qutil.test。

4.2.2 安装与包装

安装可以通过 `cmake --install` 或 `cpack` 来完成。对于大多数常规用例，`cmake --install` 或 `cpack` 可以按照 CMake 文档中描述的正常方式运行。qpdf 遵循所有常规安装规范，并使用 CMake 定义的变量来实现标准行为。

有几个组件可以单独安装：

表 1：安装组件

CLI 命令行工具库运行时库;如果你使用共享库构建开发者静态库、头文件及开发者文档所需的文件，则必须填写。文档，如果选择安装，则附带手册示例。
示例源文件

注意，该库组件只安装运行时库，不安装构建 qpdf 所需的头文件或其他文件/链接。为此，你需要开发。如果你用的是共享库，开发者会安装文件或创建符号链接，这些文件依赖于图书馆安装的文件，所以你需要两个都安装。如果你想基于 qpdf 库构建软件，只想安装所需的文件，以下是一些示例：

- 仅使用静态库安装开发文件：

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=RelWithDebInfo -DBUILD_SHARED_LIBS=OFF cmake --build build --parallel --target libqpdf cmake --install build --component dev
```

- 仅使用共享库安装开发文件：

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=RelWithDebInfo -DBUILD_STATIC_LIBS=OFF cmake --build build --parallel --target libqpdf cmake --install build --component lib cmake --install build --component lib cmake --install build --component dev
```

- 安装带有共享库和静态库的开发文件：

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=RelWithDebInfo cmake --build build --parallel --target libqpdf libqpdf_static cmake --install build --component lib cmake --install build --component lib cmake --install build --component dev
```

此外，还有一些独立选项，在构建选项中讨论，用于控制软件某些特定部分的安装方式。

4.3 构建选项

所有可用的构建选项都在顶层 CMakeLists.txt 文件中定义，并附有帮助文本。你可以通过任何标准的 cmake 前端（比如 `cmake-gui` 或 `ccmake`）看到它们。本节介绍适用于大多数人的选项

用户。如果你想将自动配置选项（从 qpdf 11 之前）映射到 cmake 选项，请参见“从自动配置转换为 cmake”。

如果你正在为发行版打包 qpdf，也应该阅读《打包员笔记》。

### 4.3.1 基础构建选项

#### BUILD\_DOC

是否用 Sphinx 构建文档。你必须安装所需的工具。

#### BUILD\_DOC\_HTML

选中 BUILD\_DOC 时可见。该选项控制将 HTML 文档与 PDF 文档分开构建，因为 sphinx 主题仅用于 HTML 文档。

#### BUILD\_DOC\_PDF

选中 BUILD\_DOC 时可见。该选项控制 PDF 文档的构建与 HTML 文档分开，因为构建 PDF 文档需要额外工具。

#### BUILD\_SHARED\_LIBS, BUILD\_STATIC\_LIBS

你可以配置是构建共享库、静态库，或者两者兼有。您必须至少选择其中一个选项。为了快速迭代，只选一个，这样构建时间减半。

在 Windows 上，如果你用共享库构建，必须在路径中包含 libqpdf 的输出目录（例如 libqpdf/ Release 或 libqpdf 在构建目录中），这样编译后的可执行文件才能找到 DLL。

如果你只用静态库构建，更新路径是不必要的。

#### 未来

该选项允许包含下一次主要版本计划中的更改。它们不是稳定 API 的一部分。这些更改违反了 ABI 规则，可能会被更改，这意味着基于该选项构建的 qpdf 链接代码可能与已安装的 qpdf 库不兼容二进制。如果你想测试拟议的 qpdf API 变更代码，并在这些变更加入发布前提供反馈，可以设置此选项。

打包器绝不应分发带有该选项的包。

#### QTEST\_COLOR

开启或关闭该功能以控制 qtest 输出是否使用颜色。

#### 佐弗利

使用 zopfli 库进行 zlib 兼容压缩。参见 Zopfli 压缩算法。

### 4.3.2 qpdf 工作选项

#### ENABLE\_COVERAGE

编译为 -coverage。有关生成覆盖报告的信息请参见 README-maintainer.md。

#### ENABLE\_QTC

默认情况下关闭，维护者模式下除外。关闭时，QTC: : TC 调用通过将 QTC: : TC 编译为空的内联函数来实现。底层的 QTC: : TC 仍然存在于库中，因此可以关闭 ENABLE\_QTC 构建和打包 qpdf 库，同时允许开发者代码如有需要使用 QTC: : TC。如果你正在修改 qpdf 代码，开启这个功能以实现更稳健的自动化测试是个好主意。否则，没必要开着它。

#### GENERATE\_AUTO\_JOB

部分 qpdf 源文件是从 job.yml 和 CLI 文档自动生成的。如果你在 qpdf CLI 中添加新的命令行参数，或者更新 qpdf 源代码中的 manual/cli.rst，应该开启这个功能。这个选项需要 Python 3。

#### 威尔

把编译器警告变成错误。我们希望 qpdf 尽可能编译时没有警告，但

总有可能因为编译器升级或工具变更而出现之前没有的警告。如果你用新编译器测试 qpdf，应该开启这个功能。

### 4.3.3 环境特定选项

#### SHOW\_FAILED\_TEST\_OUTPUT

通常，ctest（驱动 qpdf 测试套件）会将其输出的详细信息写入构建输出目录中的文件 `ctest.log`。如果你是在持续集成或自动化环境中运行构建，无法访问这些文件，你应该启用这个选项，同时运行 `ctest --verbose` 或 `ctest --output-on-failure`。这会导致详细的测试失败结果写入构建日志。

#### CI\_MODE

开启此功能后，qpdf 持续集成环境中使用的选项被广泛使用，确保我们尽可能多地发现问题。具体来说，这个选项支持 `SHOW_FAILED_TEST_OUTPUT` 和 `WERROR`，并强制构建本地加密提供商。

#### MAINTAINER\_MODE

如果你维护 QPDF，应该开启的设置选项。依次涉及以下内容：

- `BUILD_DOC`
- `ENABLE_QTC`
- `GENERATE_AUTO_JOB`
- 威尔
- `REQUIRE_NATIVE_CRYPTO`

在维护模式下可以关闭 `BUILD_DOC`，这样就不必满足建筑文档的额外需求。

### 4.3.4 构建期加密货币选择

自 9.1.0 版本起，qpdf 除了原生提供商外，还可使用外部加密提供商。有关一般讨论，请参见加密货币提供商。本节讨论如何配置哪些加密提供商被编译为 qpdf。

几乎所有情况下，外部加密提供商应优先选择而非本地供应商。不过，如果你不担心处理加密文件，想减少依赖数量，原生加密提供商是完全支持的。

默认情况下，qpdf 的构建支持所有依赖可用的外部加密提供商，只有在没有外部提供者可用时才启用本地加密提供商。你可以用这里描述的选项来改变这种行为。

#### USE\_IMPLICIT\_CRYPTO

默认情况下是开启的。关闭后，只会构建明确选定的加密服务提供商。您必须至少使用以下一项“必需”选项。

#### ALLOW\_CRYPTO\_NATIVE

该选项仅在选择 `USE_IMPLICIT_CRYPTO` 时可用，此时默认开启。关闭它可以防止 qpdf 在没有外部提供商时自动退回本地加密提供商。

#### REQUIRE\_CRYPTO\_NATIVE

即使有其他选择，也要构建原生加密提供商。

#### REQUIRE\_CRYPTO\_GNUTLS

需要 gnutls 加密服务提供商。如果没有 gnutls 库，开启这个会出错。

#### REQUIRE\_CRYPTO\_OPENSSL

要求使用 OpenSSL 加密服务提供商。如果 openssl 库不可用，开启这个会出错。

**DEFAULT\_CRYPTO**

明确选择默认使用的加密提供商。有关运行时加密提供商选择的信息，请参见运行时加密提供商选择。如果未指定，qpdf 会选择 gnutls（如果有的话），否则选择 openssl（如果有的话），最后优先选择原生。

举个例子：如果你只想用 gnutls 加密提供商来构建，你应该运行 cmake，设置为 `-DUSE_IMPLICIT_CRYPTO=0 -DREQUIRE_CRYPTO_GNUTLS=1`。

**4.3.5 高级构建选项**

这些选项仅用于特殊用途，对大多数用户无关紧要。

**AVOID\_WINDOWS\_HANDLE**

在 Windows 中禁用 HANDLE 类型。如果你在为某些嵌入式 Windows 环境构建，这会很有用。有些功能可能无法使用，但你仍然可以在这种配置中从内存中处理 PDF 文件。

**BUILD\_DOC\_DIST, INSTALL\_MANUAL**

默认情况下，安装 qpdf 不包含预编的手册副本。相反，它安装了一个 README 文件，告诉人们在哪里可以在线找到说明书。如果你想安装手册，必须启用 `INSTALL_MANUAL` 选项，并且必须在构建的手动目录里有 `doc-dist` 目录。如果选择了 `BUILD_DOC_DIST` 且 `BUILD_DOC_PDF` 和 `BUILD_DOC_HTML` 都开启，`doc-dist` 目录就会被创建。

由于构建文档所需的额外工具，`BUILD_DOC_DIST` 和 `INSTALL_MANUAL` 选项是独立且独立的。特别是，qpdf 的官方发布准备中，先在 Linux 中构建一个 `doc-dist` 目录，然后解压到 Windows 构建中，以便将其包含在 Windows 安装程序中。这避免了我们必须在 Windows 环境中构建文档。如需进一步讨论，请参见“文档包装理由”。

**INSTALL\_CMAKE\_PACKAGE**

控制是否安装 qpdf 的 cmake 配置文件（默认开启）。

**INSTALL\_EXAMPLES**

控制是否安装带文档的 qpdf 示例源文件（默认开启）。

**INSTALL\_PKGCONFIG**

控制是否安装 qpdf 的 pkg-config 配置文件（默认开启）。

**OSS\_FUZZ**

开启此选项后，fuzzer 的结构会根据 Google OSS-fuzz 项目的具体要求进行调整。没有理由因为其他原因开启这个功能。它由构建脚本支持，该脚本从该上下文构建 qpdf。

**SKIP\_OS\_SECURE\_RANDOM, USE\_INSECURE\_RANDOM**

本地加密实现在可用时使用作系统的安全随机数源。当外部加密提供商正在使用时，不使用该协议。如果你是在一个非常专业化的环境中构建，既没有外部加密提供商，也不能使用作系统提供的安全随机数生成器，你可以同时开启这两个选项。这会导致 qpdf 退回到不安全的随机数生成器，产生可猜测的随机数。最终的 qpdf 依然安全，但加密文件可能更容易遭受暴力破解攻击。除非你确定需要这些选项用于特殊用途，否则你不需要它们。这些选项是响应一位用户的特别请求而添加到 qpdf 中，该用户需要在没有安全随机数源的嵌入式环境中运行与 PDF 相关的特殊任务。



### 4.3.6 无 wchar\_t 建筑

在没有 wchar\_t 的系统上构建 qpdf 是可能的。最终的 qpdf 构建与普通 qpdf 构建不兼容 API，因此无法从 cmake 中选择该选项。该选项被添加到 qpdf 中，以支持在一个非常精简的嵌入式环境中安装，该环境仅包含部分标准 C++ 库的实现。

你可以通过在构建中定义 QPDF\_NO\_WCHAR\_T 预处理器符号（例如在 CFLAGS 和 CXXFLAGS 中包含 -DQPDF\_NO\_WCHAR\_T）来禁用 qpdf 代码中的 wchar\_t。

虽然 wchar\_t 是 C++ 标准库的一部分，几乎所有系统都应具备，但有些精简系统，比如针对某些嵌入式环境的系统，缺乏 wchar\_t。在内部，qpdf 所有内容都使用 UTF-8 编码，所以 qpdf 的 API 里没有什么重要内容需要用 wchar\_t。不过，有一些辅助方法可以从 wchar\_t\* 转换到 char\*。

如果你在不支持 wchar\_t 的环境中构建，可以在构建中定义预处理器符号 QPDF\_NO\_WCHAR\_T。无论你是在构建 qpdf，需要避免编译使用 wchar\_t QPDF 的代码，还是构建使用 qpdf 的客户端代码，这种方法都适用。

注意，当你用 libqpdf 构建代码时，只要你不调用任何使用 wchar\_t 的方法，构建中的 QPDF\_NO\_WCHAR\_T 定义不必与库构建时的定义一致。

## 4.4 加密服务提供商

从 qpdf 9.1.0 开始，qpdf 库可以通过多种密码函数提供者实现构建，我们称之为“加密提供者”。撰写本文时，加密实现必须提供 MD5 和 SHA2（256、384 和 512 位）哈希，以及带 CBC 加密和不加 CBC 加密的 RC4 和 AES256。未来如果将数字签名加入 qpdf，可能会有更多要求。其中一些是弱密码算法。关于为什么需要密码学，请参见弱密码学。

可用的加密服务提供商实现有 gnutls、openssl 和 native。OpenSSL 支持在 qpdf 10.0.0 中加入，OpenSSL 支持于 10.4.0。GnuTLS 支持在 qpdf 9.1.0 中引入。根据需要还可以添加额外的实现。开发者也有可能在不修改 qpdf 库的情况下提供自己的实现。

有关选择哪些加密提供商被编入 qpdf 的信息，请参见“构建时加密选择”。

### 4.4.1 运行时加密提供商选择

你可以使用 --show-crypto 选项来查询 qpdf，获取可用的加密提供商列表。默认提供者总是放在最前面，其余按词汇顺序排列。每个加密提供者单独列出，没有其他文本，便于在脚本中轻松使用该命令的输出。

你可以通过设置 QPDF\_CRYPTO\_PROVIDER 环境变量来覆盖使用的加密服务提供商。这样做的理由很少，但如果你明确想比较两个不同加密提供商的行为，测试性能或重现漏洞，可能会考虑这样做。对于那些正在实施自己加密服务提供商的人来说，这也可能很有用。

### 4.4.2 开发者加密提供商信息

如果你写的代码使用 libqpdf，并且想强制使用某个加密提供者，可以调用 QPDFCryptoProvider::setDefaultProvider 这个方法。该参数是内置或开发者提供的服务提供者名称。要添加自己的加密提供者，你需要创建一个从 QPDFCryptoImpl 衍生的类，并用 QPDFCryptoProvider 注册。更多信息请参见 include/qpdf/QPDFCryptoImpl 中的评论。

哈。

### 4.4.3 加密提供商设计说明

本节将介绍一些关于加密服务提供商接口设计方式的理由。你不需要知道这些信息，但这些信息是作为记录和参考，以防有趣。

一般来说，我想尽量避免包含那些有条件编译的大型代码块，在大多数构建中，有些代码根本不会被构建。这很危险，因为它很容易让无效代码悄无声息地潜入。因此，我希望能够在所有可用的加密提供商中构建 qpdf，这就是我为本地开发构建 qpdf 的方式。同时，如果某个打包器认为使用 qpdf 在该特定目的上受到严格审查的库以外的加密功能（如 gnutls、openssl 或 nettle）会带来安全风险，那么我想赋予该打包器完全禁用 qpdf 原生实现的能力。或者有人想避免对某个外部加密提供商产生依赖，我不希望该提供商的可用性在该环境中强加额外的外部依赖。这两种情况我都知道，对于一些 qpdf 用户来说是真实的。

我希望加密提供商的注册和选择是线程安全的，并且开发者能够确定性地提供自己的加密提供商，并将其设为默认。这也是要求使用 C++-11 的主要动机，因为这样我就能利用本地块静态初始化带来的线程安全保障。

QPDFCryptoProvider 类使用带有线程安全初始化的单例模式来创建 QPDFCryptoProvider 的单例实例，并在其公开接口中仅暴露静态方法。这样，如果开发者想调用任何 QPDFCryptoProvider 方法，库保证 QPDFCryptoProvider 已完全初始化，且所有内置加密提供者均已注册。让 QPDFCryptoProvider 真正知道所有内置服务，乍看之下可能有些遗憾，但这个选择让初始化行为非常清晰。提供者实现自动以非确定性顺序注册自己毫无疑问。这也意味着实现无需了解提供者接口，这使得单独测试更为容易。这种方法的另一个优点是，想要开发自己的加密提供商的开发者可以完全独立于 qpdf 库，只需两次调用，qpdf 就能在应用中使用他们的服务提供者。如果他们决定贡献代码，插入 qpdf 库只需对 qpdf 源代码做非常小的修改。

让加密提供商在运行时可选择决定让我有些挣扎，但出于各种原因我还是决定这么做了。允许终端用户轻松切换加密提供商，对于重现潜在漏洞非常有用。如果用户报告某个密码学问题，我可以让那个人用不同的值尝试

QPDF\_CRYPTO\_PROVIDER 变量。如果出现性能问题，情况也可能如此。这也使 qpdf 自有测试套件更容易与不同供应商进行代码测试，而无需让所有与 qpdf 关联的程序都知道可能存在多个提供者。在 qpdf 的持续集成环境中，每个支持的加密提供商都会运行整个测试套件。通过使用环境变量选择提供者，这变得更加简单。

最后，通过这种方式让加密服务提供商可选择，形成了一个我未来可能会为流过滤提供商效仿的模式。可以想象未来会有增强功能，有人能为像 /FlateDecode 这样的基础过滤器或其他 qpdf 不支持的其他过滤器提供自己的实现。使用 C++-11 的功能接口实现注册功能和注册提供者的内部存储也更为便捷，这也是当时需要 C++-11 的另一个原因。

## 4.5 支持 zopfli 的建设

如果你用 -DZOPFLI=ON 编译并有 zopfli 开发文件，qpdf 将支持佐普夫利。

有关使用 zopfli 与 qpdf 的信息，请参见 Zopfli 压缩算法。

## 4.6 从自动 conf 转换为 cmake

qpdf 11 之前的 qpdf 版本采用 autoconf 和自制的 GNU Make 构建系统。如果你用特殊的 ./configure 选项做了 qpdf，这部分可以帮助你切换到 cmake。

在大多数情况下，配置选项和 cmake 选项之间存在一对一的映射。但有一些例外：

- cmake 构建中有一些新选项，是自动构造构建中没有的。下表展示了 cmake 中的旧期权及其对应选项。

enable-avoid-windows-handle `AVOID_WINDOWS_HANDLE` enable-check-autofiles `non` – 与 `cmake` 无关  
 enable-crypto-gnutls `REQUIRE_CRYPTO_GNUTLS` enable-crypto-native `REQUIRE_CRYPTO_NATIVE` （但见上文）  
 enable-crypto-openssl `REQUIRE_CRYPTO_OPENSSL` enable-doc-maintenance `BUILD_DOC`  
 enable-external-libs `non` – detected automatic enable-html-doc `BUILD_DOC_HTML` enable-implicit-crypto `USE_IMPLICIT_CRYPTO`  
 enable-insecure-random `USE_INSECURE_RANDOM` enable-ld-version-script `nonone` – 检测到自动启用维护模式 `MAINTAINER_MODE` （略有差异）  
 enable-os-secure-random （默认开启）`SKIP_OS_SECURE_RANDOM` （默认关闭）  
 enable-oss-fuzz `OSS_FUZZ` enable-pdf-doc `BUILD_DOC_PDF`  
 enable-rpath `无` – `cmake` 正确处理 `rpath` 启用-显示-失败-测试输出 `SHOW_FAILED_TEST_OUTPUT`  
 QPDF\_TEST\_COMPARE\_IMAGES enable-test-compare-images 设置 环境变量  
 enable-werror `werror` with-buildrules `无` – 与 `cmake` with-default-crypto 无关 `DEFAULT_CRYPTO` 大型文件测试路径

设置 `QPDF_LARGE_FILE_TEST_PATH` 环境变量



## 打包员笔记

如果你正在为作系统发行版打包 qpdf，本章适合你。否则，可以跳过。

### 5.1 构建选项

关于构建选项的详细讨论，请参阅构建选项。本节特别关注对包装人员特别有用的选项。

- Perl 必须在构建时就存在。在 qpdf 9.1.1 版本之前，运行时依赖 Perl，但现在情况已不同。
- 确保你获得的加密货币服务商的行为是正确的。详情请阅读《构建期加密货币选择》。
- 建议在持续集成或其他自动化环境中使用 `SHOW_FAILED_TEST_OUTPUT`，因为它可以在构建日志中看到测试失败。这应与 `ctest --verbose` 或 `ctest --输出-failure` 结合使用。
- 打包者绝不应该定义未来构建选项。FUTURE 启用的 API 变更并不稳定，可能会破坏 API/ABI。该选项仅供那些用本地 qpdf 构建测试代码的人，以便提供早期反馈或为未来可能的 API 变更做准备。
- QPDF 的安装目标默认不安装完成文件，因为没有标准的完成文件位置。作为一个打包者，最好把它们安装在你发行版预期文件去的地方。你可以在完成目录中找到可以安装的完成文件。更多信息请参见完成文件/README.md 文件。
- 从 qpdf 11 开始，qpdf 默认安装会从 example 目录中安装带有文档的源文件。在 qpdf 11 之前，这只是对打包商的建议，但并非自动执行。
- 从 qpdf 11.10 开始，qpdf 可以构建支持 zopfli 的版本（参见 Building with zopfli 支持）。建议不要用 zopfli 构建 qpdf 作为发行版，因为它会增加 zopfli 作为依赖，而且该库的使用频率低于 qpdf 的其他依赖。想要这样的用户大概知道自己想要，而且他们可以从源代码编译。注意，根据 zopfli 自己的文档，zopfli 比 zlib 慢大约 100 倍，压缩输出也小了大约 5%。

### 5.2 包测试

pkg-test 目录包含非常小的测试壳脚本，旨在帮助对 qpdf 的安装进行烟雾测试。它们设计用于 Debian 的 autopkgtest 框架，但也可以被其他用户使用。详情请参见源代码发行版中的 pkg-test/README.md。

## 5.3 包装文档

从 qpdf 10.5 版本开始，预构建文档不再随 qpdf 源发行版一起分发。以下是你可能想考虑的几个套餐选项：

- 什么都别做

当你运行“安装”时，文件 README-doc.txt 会安装在文档目录中。该文件告诉读者在线哪里可以找到文档，以及去哪里下载离线文档副本。这是 Debian 包选择的选项。

- 嵌入预构建文档

你可以获取预构建的文档，并将其内容提取到你的发行版中。这就是 qpdf 发布网站上可用的 Windows 二进制发行版所做的事情。你可以在文件 qpdf-version-doc.zip 的发布区域找到预建文档。作为这种方法的一个例子，可以看看 qpdf 的 GitHub 动作构建脚本。build-scripts/build-doc 脚本的构建时，-DBUILD\_DOC\_DIST=1 来创建文档分发。build-scripts/build-windows 脚本会将其解压到构建树中，并以 -DINSTALL\_MANUAL=1 的设置将其包含在安装程序中。

- 自己构建文档

你可以在构建过程中构建文档。确保传递 -DBUILD\_DOC\_DIST=1 和 -DINSTALL\_MANUAL=1 来完成 cmake。这就是 ApplImage 构建的功能。在最初转换时，基于斯芬克斯的文档是最新版本的 Sphinx，是 4.3.2。旧版本并不保证能正常工作。

### 5.3.1 文档包装理由

本节描述了事情之所以会如此的原因。仅供参考;你不需要知道这些内容就能打包 QPDF。

这种变化的原因是什么？在 qpdf 10.5 之前，qpdf 手册是一个 docbook XML 文件。生成的文档是通过构建时样式表运行文件生成的，且不包含任何版权材料。从 10.5 版本开始，手册采用重构文本编写，并使用 Sphinx 构建。此更改旨在简化自动生成部分文档，并使文档更易于作。Sphinx 的 HTML 输出也比文档书样式表更易读、更适合在线使用。缺点是生成的 HTML 文档现在包含了 Javascript 代码和嵌入字体，PDF 版本的文档不再适合打印（至少在 10.5 版本发行时如此），因为外部链接目标不再显示，交叉引用也不再包含页码信息。即使内容采用了 MIT 和 BSD 许可，生成文档中存在受版权保护的内容，也会在多方面使打包者的工作变得复杂。首先，这意味着源仓库中的 NOTICE.md 文件必须跟上非仓库控制文件的版权信息。此外，一些发行版（尤其是 Debian/Ubuntu）不鼓励在包中包含 sphinx 生成的文档，更倾向于你在包构建过程中构建文档，并在运行时依赖包含代码的共享包。在将 qpdf 手册从 docbook 转换到 sphinx 时，需要比大多数 Debian/Ubuntu 版本中更新版本的 sphinx 和 html 主题。

由于始终在线的互联网连接比以前普遍得多，许多 qpdf 用户更愿意在线获取文档，而发行版中缺乏预建文档也不会成为大问题。然而，仍有一些人无法或选择不在线查看相关文件。对他们来说，预建文档仍然可用。

## 运行 QPDF

本章介绍如何从命令行运行 qpdf 程序。

### 6.1 基本祈祷

用法：qpdf [文件中] [选项] [文件外文件]

qpdf 命令读取 PDF 文件 (infile)，对内存中的文件进行各种变换或修改，并将结果写入 outfile。当无选项运行时，输出文件功能上与输入文件相同，但可以进行结构重组，且孤儿对象会从文件中移除。有许多选项可用于对文件进行变换或修改。

infile 可以是普通文件，也可以是 ---empty，从空的 PDF 文件开始。由于输入文件必须是可寻址的，因此无法使用标准输入。

outfile 可以是普通文件，表示标准输出，或者 replace-input 表示应覆盖输入文件。输出文件不必是可寻址的，即使生成线性化文件。你也可以用 --split-pages 为每个页面（或一组页面）创建独立的输出文件，而不是单一输出文件。

通过指定密码 --password 来打开受密码保护的文档。

除帮助选项（参见帮助/信息）外，所有选项都需要输入文件。如果提供了检查或 JSON 选项（参见 PDF 检查和 JSON 选项）或帮助选项，则不得提供输出文件。否则，需要输出文件。

如果 @filename 作为单词出现在命令行中，将逐行读取，每行都被视为命令行参数。前置和后置的空白部分故意不从行中移除，这使得处理以空格开头或结尾的参数成为可能。@- 选项允许从标准输入读取参数。这使得 qpdf 可以用任意数量的任意长度参数调用。它也非常方便地避免在命令行传递密码，但请参见 --password-file。注意，@filename 不能出现在参数中间，因此诸如 --arg=@filename 这样的构造不适用。相反，你必须在文件名文件中包含选项及其参数（例如 --option=parameter）作为一行，然后直接在命令行传递 @filename。

#### 6.1.1 相关选项

——空的

这个选项可以代替 “infile”。这会导致 qpdf 使用一个包含零页的虚拟输入文件。这个选项与 --pages 一起使用非常有用。详情请参见页面选择。

—替换输入

这个选项可以代替外文件。这会导致 qpdf 用输出文件替换输入文件。它通过写入 infilename.~qpdf-temp# 来实现这一点，完成后用临时文件覆盖输入文件

档案。如果有任何警告，原始输入会保存为 infilename。~qpdf-orig。如果有错误，输入文件保持不动。

--job-json-file=file

指定一个文件的名称，其内容预计包含 QPDFJob JSON 文件。该文件被读取并视为提供了等效的命令行参数。它可以重复使用，并与其他选项自由混合。运行带有 --job-json-help 的 qpdf，以获取 JSON 输入文件格式的描述。更多信息请参见 QPDFJob：基于职位的界面。注意这与 --json 无关，但可以与之结合使用。

6.2 退出状态

qpdf 的退出状态可以解释如下：

表 1：出口代码

0	未发现错误或警告
1	未使用过
2	发现了错误;该文件未被处理
3	警告无错误

注释：

- PDF 文件可能存在 qpdf 检测不到的问题。
- 使用 --warning-exit-0 选项时，即使有警告，退出状态仍会被使用。
- QPDF 不会以状态 1 退出，因为如果 shell 无法调用该命令，则会使用该退出代码。
- 如果同时发现错误和警告，则使用出口状态 2。
- --is-encrypted 和--需要-password 选项使用不同的退出码。详情请查看他们的帮助。

6.2.1 相关选项

--警告-出口-0

如果只有警告且没有错误，则以出口代码 0 退出，而不是 3。与 --no-warn 结合时，qpdf 会完全忽略警告。

6.3 壳牌完成

QPDF 为 ZSH 和 BSH 提供自己的补全支持。你可以用评估 “\$(qpdf --completion-bash)” 启用 bash 补全，用 eval “\$(qpdf --completion-zsh)” 实现 zsh 补全。如果 qpdf 不在你的路径中，你应该在上述调用中使用绝对路径到 qpdf。如果你用相对路径调用它，它会警告你，而且如果你在不同目录里，补全功能就无法生效。QPDF 会使用 ARGV[0] 来确定其可执行文件的位置。这在某些情况下可能会产生不想要的结果，尤其是当你试图使用完备处理直接从源树运行的 qpdf 副本，或者用包装脚本调用时。你可以在 QPDF\_EXECUTABLE 环境变量中指定你想用来完成的 qpdf 的完整路径。



### 6.3.1 相关选项

——完成敲击

输出一个可以评估的完成命令，从而从 bash 中启用壳体补全。

——完备 -zsh

输出一个可以评估的完成命令，从而从 zsh 中启用 shell 补全。

## 6.4 帮助/信息

帮助选项提供了关于 qpdf 本身的一些信息。帮助选项只能作为第一个也是唯一的命令行参数。

### 6.4.1 相关选项

——求助[=--选项|主题]

显示命令行调用帮助。使用 `--help=--option` 来帮助特定选项，使用 `--help=topic` 来帮助某个帮助主题，同时还提供可用的帮助主题列表。

——版本

显示 qpdf 的版本。显示的版本号是编译到 qpdf 库中的版本号。如果你没有看到预期的版本号，说明你可能安装了多个版本的 qpdf，库路径设置不正确。

——版权

显示版权和许可信息。

——展示加密货币

在单独的一行中展示可用的加密提供商列表。默认服务商总是排在最前面。有关加密提供商的更多信息，请参见“加密提供商”。

——job-json-help

描述 `--job-json-file` 所用的 QPDFJob JSON 输入格式。有关 QPDFJob 的更多信息，请参见 QPDFJob：基于岗位的界面。

——索普弗利

如果已编译 zopfli 支持，请注明是否激活，然后正常退出。否则，表示未编译，并以错误代码退出。如果 zopfli 被编译进去，通过设置 `QPDF_ZOPFLI` 环境变量来激活它。参见 Zopfli 压缩算法。

## 6.5 通用期权

本节描述了控制 QPDF 行为的通用选项。它们不一定与正在执行的具体作相关，无论是否生成输出文件，都可能使用。

### 6.5.1 相关选项

——password=password

指定访问加密、受密码保护文件的密码。要从文件或标准输入读取密码，可以使用 `--password-file`。

在 8.4.0 之前，对于包含字符不属于 7 位 US-ASCII 的密码，qpdf 将提供正确编码的加密和解密密质的责任留给了用户。从 qpdf 8.4.0 开始，qpdf 在大多数情况下会自动完成此作。如需深入讨论，请参见 Unicode 密码。之前

该手册的版本描述了使用 `iconv` 命令的变通方法。从 qpdf 8.4.0 开始, 此类变通方法不再被要求或推荐。不过, 为了向后兼容, qpdf 会尝试检测这些变通方法, 并在大多数情况下做正确的作。

#### `--password-file=filename`

读取指定文件的第一行, 并用作访问加密文件的密码。文件名可能是 `-` 用来读取标准输入的密码, 但如果这样做密码会被回声且没有提示, 所以请谨慎使用。请注意, 密码中前置和后置空格并未被移除。

#### `--冗长`

增加产出的冗长度。这包括创建文件、图像优化以及其他几项作的信息。在某些情况下, 当使用检查选项 (见 PDF 检查) 时, 还会显示额外信息。

#### `--进步`

在写入输出文件时表示进度。进度指示直到写入开始才开始, 因此如果在写入过程开始前应用了复杂的变换, 进度指示器可能会有延迟。

#### `--不警告`

禁止写警告给 `stderr`。如果检测到警告并被抑制, qpdf 仍会以出口代码 3 退出。要完全忽略警告, 还要指定 `--warning-exit-0`。使用时请谨慎, 因为 qpdf 并不总能成功从导致警告发出的情况中恢复。

#### `--确定性 ID`

使用确定性值生成安全、随机的文档 ID。这防止在 ID 生成时使用时间戳和输出文件名信息。相反, 在一定的运行时间成本下, ID 字段被生成为包含输出 PDF 文件重要部分的摘要。这意味着每个 qpdf 作每次运行时都应生成相同的 ID, 这在缓存结果或生成某些测试数据时非常有用。使用该标志与创建加密文件不兼容。这个选项对测试很有用。另见 `--static-id`。

虽然 qpdf 在相同的输出 PDF 下会产生相同的确定性 ID, 但不能保证不同版本的 qpdf 会针对相同的输入和选项生成完全相同的 PDF 输出。虽然会注意避免对 qpdf 的 PDF 生成进行无谓更改, 但新版本的 qpdf 可能会包含一些更改或漏洞修复, 导致生成略有不同的 PDF 代码。这些变化会在发布说明中说明。

#### `--允许弱加密货币`

使用 40 位密钥或 128 位密钥 (无 AES) 的加密 PDF 文件使用不安全的 RC4 加密算法。从 11.0 版本开始, qpdf 的默认行为是拒绝使用 RC4 加密写入文件。使用此选项允许创建此类文件。在 10.4 至 10.6 版本中, 尝试创建弱加密文件时, 没有该标志时会被警告而非错误。更多细节请参见弱密码学。

在保留加密参数或从其他文件复制加密参数时, 不进行弱加密检查。其原理在《弱密码学》中有详细讨论。

#### `--保持文件打开=[y|n]`

该选项控制 qpdf 在合并时是否保持单个文件开启。默认情况下, qpdf 在合并时会保持文件开放, 除非指定超过 200 个文件, 此时文件按需打开, 完成后关闭。反复开闭文件可能会在某些文件系统, 尤其是网络文件系统中带来较大的性能损失。如果你知道你的开放文件限制足够大且存在性能问题, 或者你的开放文件限制小于 200, 你可以使用这个选项覆盖默认行为, 比如指定 `--keep-files-open=y` 强制 qpdf 保持文件开放, 或者 `--keep-files-open=n` 强制它只在需要时打开文件。另见 `--keep-files-open-threshold`。

历史说明: 在 8.1.0 版本之前, qpdf 始终保持所有文件开放, 但这意味着可合并的文件数量受限于作系统的开放文件限制。8.1.0 版本可以按原样打开文件

每次读取后都引用并关闭了它们，但这导致了显著的性能影响。8.2.0 版本优化了性能，但对本地文件系统来说，性能损失虽小但不可避免，而对于网络文件系统，性能影响可能非常高。当前的行为是在 qpdf 8.2.1 版本中引入的。

--保持文件开阈值=计数

如果指定，会覆盖作为 qpdf 决定是否保持文件打开的阈值 200 的默认值。详情请参见 --keep-files-open。

## 6.6 高级控制选项

高级控制选项以用户通常不必要的方式控制 QPDF 的行为，但这对开发者或调查特定文件问题的人可能有用。

### 6.6.1 相关选项

--密码是六进制键

通过明确指定加密密钥，覆盖通常从用户/所有者密码计算/检索 PDF 文件加密密钥的过程。当指定该选项时，--password 选项的参数被解释为十六进制编码的密钥值。这只适用于打开主输入文件时使用的密码。它不适用于由 --pages 或其他选项打开的文件，也不适用于正在写入的文件。

大多数用户永远不需要这种选项，也没有标准的查看器支持这种作模式，但它在取证或调查方面非常有用。例如，如果 PDF 文件用未知密码加密，直接使用密钥进行暴力破解攻击有时比使用密码更有效。此外，如果文件严重损坏，可能可以直接推导出加密密钥并恢复文件的部分内容。要暴露你可以正常打开的加密文件所使用的加密密钥，可以使用 --show-encryption-key 选项。

--suppress-password-recovery

通常，qpdf 会尝试自动补偿编码错误字符的密码。这个选项可以抑制这种行为。在正常情况下，没有理由使用这个选项。关于讨论，请参见 Unicode 密码。

--password-mode=mode

该选项可用于微调 qpdf 如何解释命令行传递的 Unicode（非 ASCII）密码字符串。除十六进制字节模式外，这些密钥仅适用于加密文件时提供的密码。十六进制字节模式也适用于指定用于读取文件的密码。关于支持的密码模式及使用时间，请参见 Unicode 密码。支持以下模式：

- 自动：自动判断指定密码是否为正确编码的 Unicode（UTF-8）字符串，并根据 PDF 规范的要求根据所应用的加密类型进行转码。从 Windows 8.4.0 版本开始，以及几乎所有其他现代平台，入信密码都会被正确编码为 UTF-8，所以这几乎总是你想要的。
- unicode：告诉 qpdf 输入密码是 UTF-8，覆盖自动检测判定的任何信息。该模式与自动模式的唯一区别是，如果密码非有效 UTF-8，qpdf 会失败并显示错误信息，而不是退回到字节模式并发出警告。
- 字节：将密码解释为字面字节串。对于非 Windows 平台，qpdf 8.4.0 之前的版本就是这样做的。对于 Windows 平台，无法直接在命令行中指定二进制数据字符串，但你可以使用 @filename 选项或 --password-file 来实现，此时该选项会强制 qpdf 遵守提供的字节串。请注意，这个选项可能导致你用其他读者无法使用的密码加密 PDF 文件。

- 十六进制字节：将密码解释为十六进制编码字符串。这为所有平台（包括 Windows）提供了将二进制数据作为密码传递的方式。与字节类似，该选项可能允许创建其他读取器无法打开的文件。该模式影响 qpdf 对文件解密和加密密码的解释。它使得字符串可以以系统默认编码以外的方式编码。

#### ——抑制-恢复

防止 qpdf 在读取文件对象时尝试重建文件的交叉引用表。康复是由多种情境触发的。虽然通常有效，但它使用的启发式方法并不适用于所有文件。如果给出了这个选项，qpdf 在遇到的第一个错误时就会失败。

#### --ignore-xref-streams

告诉 qpdf 忽略所有交叉引用流，退回到任何嵌入的交叉引用表或触发文档恢复。通常，qpdf 会读取 PDF 文件中存在的交叉引用流。如果指定了该选项，qpdf 将忽略混合 PDF 文件的任何交叉引用流。混合文件的目的是让一些不了解交叉引用流的观众能够获得内容。几乎从来都不应该忽视它们。只有在测试混合 PDF 文件创建时，想看看不理解对象流和交叉引用流的 PDF 消费者如何解读此类文件时，才可能需要使用此功能。

## 6.7 PDF 变换

本节讨论的选项告诉 qpdf 应用变换，改变 PDF 文件的结构而不改变其内容。例如创建线性化（网页优化）文件、添加或移除加密、为旧版查看器重构文件，以及为人工检查重写文件。另见 PDF 修改。

### 6.7.1 相关选项

#### ——线性化

创建线性化（网页优化）输出文件。线性文件的格式允许合规阅读者在 PDF 文件完全下载前就开始显示。通常，整个文件必须完整存在后才能渲染，因为重要的交叉参考信息通常出现在文件末尾。

#### --加密 [选项] --

该标志启动加密选项，用于创建加密文件。详情请参见加密部分。

#### ——解密

即使输入文件加密，也要创建一个没有加密的输出文件。该选项覆盖默认保留输入文件中加密的行为。此功能不用于绕过版权限制或制作方对文件施加的其他限制。另见 --copy-encryption 和 --remove-restrictions。

#### ——移除限制

移除与数字签名 PDF 文件相关的安全限制。它可以与 --decrypt 结合，允许自由编辑之前签署/加密的文件。该选项会使所有数字签名失效并禁用，但保留其视觉外观。

#### --copy-encryption=file

将所有加密参数，包括用户密码、所有者密码及所有安全限制，从指定文件复制，而非保留输入文件中的加密细节。即使只知道其中一个用户密码或所有者密码，这个方法也能正常工作。如果加密文件需要密码，请使用 --encryption-file-password 选项来设置。注意，从文件复制加密参数也会从文件复制 /ID 的前半部分，因为这是加密参数的一部分。如果你需要解密文件以手动修改，或者在 qpdf 之外更改文件，这个选项会很有用

恢复文件的原始加密，无需手动指定所有单独设置。另见 `---decrypt`。  
该作故意不对弱密码算法进行检查。有关其原理，请参见“弱密码学”一节。

`--encryption-file-password=password`

如果指定为 `--copy-encryption` 的文件需要密码，则使用该选项提供密码。此选项是必要的，因为 `--password` 选项适用于输入文件，而非加密复制文件。

`---QDF`

创建一个适合在文本编辑器中查看和编辑的 PDF 文件。这是编辑 PDF 代码，而不是页面内容。要编辑 QDF 文件，你的文本编辑器必须保留二进制数据。在 QDF 文件中，所有可解压缩的流都是未压缩的，内容流也会被规范化，除此之外还有其他变化。配套工具 `fix-qdf` 可用于修复手工编辑的 QDF 文件。QDF 是 qpdf 工具的专属功能。更多信息请参见 QDF 模式。注意 `--linearize` 会禁用 QDF 模式。

QDF 模式完全支持对象流，但有时如果禁用对象流，定位特定对象会更容易。当试图通过检查现有文件来理解某个 PDF 结构时，将 `--qdf` 与 `--object-streams=disable` 结合起来会很有用。

该标志会改变其他选项的一些默认值：流数据为未压缩，内容流被规范化，加密被移除。这些默认仍然可以通过指定 `--qdf` 来覆盖。此外，在 QDF 模式长度作为间接对象存储，对象的格式化效率较低但更易读，文档中穿插注释，方便用户查找内容，同时使 `fix-qdf` 能够正常工作。编辑 QDF 文件时，无需维护对象格式。

在规范化内容时，如果 qpdf 遇到词汇错误，会打印警告，提示内容可能已损坏。如果你想创建没有内容规范化的 QDF 文件，请指定 `--qdf --normalize-content=n`。你也可以用 `--stream-data=uncom` 创建一个包含未压缩流的非 QDF 文件。无论哪种方式，都会解压所有流，但不会尝试规范化内容。请注意，如果你使用内容规范化或 QDF 模式手动检查文件，无需担心这些。

另见 `--no-original-object-ids`。

`--无原始对象标识`

抑制在 QDF 文件中包含原始对象 ID 注释。这在生成 QDF 文件进行测试时非常有用，尤其是在比较两个 PDF 文件是否内容相同时。原始对象 ID 注释默认存在，因为它方便追踪对象到原始文件。

`--压缩流=[y|n]`

默认情况下，或者 `--compress-streams=y` 时，qpdf 会使用 flate 压缩算法（zip 和 gzip 使用），除非这些流以其他方式被压缩。该分析是在 qpdf 尝试解压流之后进行的，因此与 `--decode` 级别密切相关。为了抑制这种行为并保持流不压缩，可以使用 `--压缩-streams=n`。在 QDF 模式下（参见 QDF 模式和 `--qdf`），默认保持流不压缩。

`--解码级=参数`

控制 qpdf 尝试解码哪些流。默认是通用的。

参数的值如下：

- 无：请勿尝试解码任何流。这是默认的 `--json-output`。

- 广义：通过支持的广义过滤器过滤的译码流：/LZWDecode、/FlateDecode、/ASCII85Decode 和 /ASCIHexDecode。我们将通用过滤器定义为用于通用压缩或编码的过滤器，而非专门为图像数据设计的过滤器。这是默认设置，除非给出了 --json-output。
- 专用：除了通用外，还支持非有损专用滤波器的译码流;目前这只是 /RunLengthDecode
- 所有：除了通用和专用外，还支持有损滤波器的解码流;目前仅为 /DCTDecode (JPEG)

qpdf 不支持几个过滤器。无论选择哪种方式，这些都保持不动。未来的 qpdf 版本可能会支持额外的过滤器。由于默认值是泛化的，qpdf 的默认行为是解压任何使用 qpdf 理解的非有损过滤器编码的流。如果 --compress-streams=y 也在生效，这是默认设置（参见 --compress-streams），总体效果是 qpdf 会用 flate 压缩的通用过滤器重新压缩流，有效消除基于 LZW 和 ASCII 的过滤器。这通常是理想的行为，但可以用 --decode-level=ne 来禁用。注意，当指定 --json-output 时，默认为 --decode-level=none，但在这种情况下也可以被覆盖。作为一个特殊情况，已经用 /FlateDecode 压缩的流不会被解压和重新压缩。

你可以用 ---recompress-flate 来改变这种行为。另见优化文件大小。

--stream-data=参数

控制流数据的转换。该选项早于 --compress-streams 和 --decode-level 选项。这些选项可以用来实现同样的效果，但控制力更大。参数的值可以是以下之一：

- 压缩：在可能的情况下重新压缩流数据（默认）；等价于 --compress-streams=y --decode-level=广义。除非也指定了 --recompress-flate，否则不会重新压缩已压缩过的流 /FlateDecode。
- 保留：保持所有流数据原样；等价于 --compress-streams=n --decode-level=none
- 解压：尽可能使用广义过滤器压缩流数据；等价于 --compress-streams=n --decode-level=广义

---重新压缩-脱离

PDF 默认使用的通用压缩方案是 flate (/FlateDecode)，与 zip 和 gzip 相同。通常 qpdf 会不动这些。该选项会告诉 qpdf 对用 flate 压缩的流进行解压和重新压缩。这与压缩水平结合使用时非常有用。使用这个选项可能会让 qpdf 在写输出文件时变慢很多。另见优化文件大小。

--compress-level=水平

在编写用 /FlateDecode 压缩的新流时，使用指定的压缩级别。电平的值应为 1 到 9 的数字，直接传递给 zlib，zlib 实现了放气压缩。数字越小，压缩越少，速度越快；数字越大，压缩越多，速度越慢。注意，qpdf 默认不会对带 Flate 压缩的流进行解压和重新压缩。要让这个选项适用于已经压缩好的流，你还应该指定 --recompress-flate。如果你的目标是缩小 PDF 文件大小，你也应该使用 --object-streams=generate。如果你省略了这个选项，qpdf 会遵循压缩库的默认行为。另见优化文件大小。

--jpeg-quality=水平

在用 --optimize-images 重写图像时，将新图像的质量设置为 0（最低）到 100（最高）。高质量会产生更大的图像，低质量则会得到更小的图像。一定要检查你的输出，看看质量是否合格。这个选项只有在与 --optimize-images 结合时才有效。这个选项还会导致已经用 JPEG 压缩过的文件

未压缩和再压缩，可能会带来额外的图像质量损失。另见优化文件大小。

--normalize-content=[y|n]

启用或禁用 PDF 内容流中换行的规范化为 UNIX 风格换行，这对于跨多个平台以程序员友好的文本编辑方式查看文件非常有用。内容规范化默认关闭，但通过 --qdf（参见 QDF 模式）会自动启用。不建议在生产中使用此选项。如果 qpdf 在规范化内容时遇到词汇错误，会打印警告，提示内容可能受损。

--object-streams=mode

控制对象流的处理。模式的值可以是以下之一：

表 2：对象流模式

保留	保留原始对象流，如果有（默认设置）禁用无对象流的创建输出文件，生成创建对象流，并尽可能压缩对象
禁用	禁用对象流，所有对象都写成普通的未压缩对象
生成	生成对象流，所有对象都写成普通的未压缩对象

对象流是包含其他对象的 PDF 流。将对象放入对象流中可以压缩 PDF 对象本身，从而产生更小的 PDF 文件。将此选项与 --压缩-级别和-recompress-flate 结合，通常可以生成较小的 PDF 文件。

对象流，也称为压缩对象，于 2003 年前后在 PDF 规范 1.5 版本中引入。一些古老的 PDF 查看器可能不支持带有对象流的文件。QPDF 可用于将带有对象流的文件转换为无对象流的文件，反之亦然。

在保留模式下，对象与包含它们的流之间的关系会从原始文件中保留。如果文件没有对象流，qpdf 就不会添加任何对象流。在禁用模式下，所有对象都写成普通的未压缩对象。最终的文件结构上应能被旧版 PDF 阅读器读取，尽管文件中仍有可能包含一些旧版阅读器无法支持的内容。在生成模式下，qpdf 会创建自己的对象流。这通常会令 PDF 文件更紧凑。在此模式下，qpdf 还会确保头部的 PDF 版本号至少是 1.5。

--保留-无引用

告诉 qpdf 保留写文件时未被引用的对象。通常情况下，任何未在从拖尾词典遍历文档时被引用的对象都会被丢弃。禁用此默认行为可能有助于处理损坏文件或检查已知未被引用对象的文件。

对于线性化文件，该标志被忽略，导致新文件中的对象按对象 ID 从原始文件排序。这并不意味着目标编号会相同，因为 qpdf 可能以不同方式创建流长度，无论是直接还是间接，且原始文件的编号可能存在空白。

另见 --preserve-unreferenced-resources，它做的事情完全不同。

--remove-unreferenced-resources=parameter 参数：  
auto（默认），是或不是。

从 qpdf 8.1 开始，在分割页面时，qpdf 能够尝试移除页面未使用的图片和字体，即使它们在页面资源字典中被引用。当共享资源被使用时，这种行为可以大幅减少拆分页的文件大小，但分析速度非常缓慢。在 8.1 至 9.1.1 版本中，qpdf 默认进行了这项分析。从 qpdf 10.0.0 开始，如果使用 auto，qpdf 会对文件进行快速分析，以判断文件页面上是否有未引用对象，这种模式在资源词典跨多个页面共享时很常见，平时很少发生。如果发现了这种模式，就会尝试移除未被引用的资源。通常这意味着只有当文件会变小时，才会看到分拆速度变慢。你可以通过指定“否”来完全抑制未引用资源的移除，或者通过指定“是”强制 qpdf 执行完整算法。

除非你不在意文件大小，而是非常在意运行时，使用这个选项的理由很少，尤其是现在自动模式已经被支持。使用这个方法的一个原因是，如果你怀疑 qpdf 在移除它本不该移除的资源。如果你遇到这种情况，请在 <https://github.com/qpdf/qpdf/issues/> 报告为 bug。

#### --保留-未引用-资源

这是 `--remove-unreferenced-resources=no` 的同义词。参见 `--remove-unreferenced-resources`。

另见 `---preservation-unreferenced`，它做了完全不同的事情。为了减少混淆，你应该用 `--remove-unreferenced-resources=no` 代替。

#### --换行前于 endstream

告诉 qpdf 在任何流内容后插入一个换行，但不计入长度，即使流的最后一个字符是换行。在某些情况下，这可能导致两个换行。这是 PDF/A 的要求。虽然 qpdf 并不专门知道如何生成符合 PDF/A 的 PDF，但至少这能防止它对已经符合标准的文件进行删除。

#### ——内容融合

当页面内容被拆分到多个流时，这个选项会导致 qpdf 将它们合并为一个流。这种功能在日常使用中从来不是必需的，但在某些情况下处理某些文件时会有所帮助。例如，可以与 QDF 模式或内容规范化结合使用，使页面内容更易于一次性查看。PDF 写作者通常会创建多个内容流，原因多种多样，比如更方便修改页面内容，以及将非常大的内容流拆分，以便 PDF 查看器能节省内存。

#### --外部化-内联-图像

将内联图片转换为普通图片。默认情况下，选中该选项时，数据至少为 1024 字节的图像会被转换。使用 `--ii-min-字节` 来更改大小阈值。当选择 `--optimize-images` 时，除非同时指定 `--keep-inline-images`，否则该选项会隐式选择。

#### --ii-最小字节=字节大小

避免将大小低于规定最小尺寸的内联图片转换为普通图片。默认是 1,024 字节。用 0 表示无最低限度。

#### --最小版本=版本

强制输出文件的 PDF 版本至少是版本。换句话说，如果输入文件的版本低于指定版本，就会使用指定的版本。如果输入文件版本更高，则使用输入文件的原始版本。很少需要使用此选项，因为 qpdf 在添加需要新 PDF 阅读器的功能时会自动扩展版本。

版本号可以用 `major.minor[.extension-level]` 的形式表示。如果。扩展级别，版本在扩展层级被解释为 `major.minor`。例如，版本 1.7.8 代表了扩展级别 8 的版本 1.7。请注意，命令行进行的语法检查非常简洁。QPDF 不检查指定版本是否真的需要。

#### --force-version=version

该选项强制 PDF 版本为指定的精确版本，即使文件中可能包含该版本不支持的内容。版本号的解释方式与 `--min-version` 相同，以便设置扩展级别。在某些情况下，强制输出文件的 PDF 版本低于输入文件，会导致 qpdf 禁用文档的某些功能。具体来说，如果版本小于 1.7 且扩展等级为 8，则禁用 256 位密钥（但已废弃且不支持的“R5”格式允许扩展等级 3 到 7），如果版本低于 1.6，则禁用 AES 加密；如果小于 1.5，禁用明文元数据和对象流；低于 1.4，禁用 128 位加密密钥，如果低于 1.3，所有加密功能都将被禁用。即使采取了这些预防措施，qpdf 也无法取消新版 PDF 中可能新增的图像压缩方案、透明度组或其他功能。



一般来说，除了像使用对象流或 AES 加密这样的重大结构性问题外，PDF 查看器应该忽略它们不支持的功能。这意味着强制更改版本可能会让你打开使用较旧版本的 PDF 文件，但请注意原始文档的一些功能可能会丢失。

## 6.8 页数范围

多个 qpdf 命令行选项使用页面范围。本节描述页面范围的语法。

- 普通数字表示从 1 开始编号的页面，因此 1 代表第一页。
- 前面有 r 的数字从末尾开始，比如 r1 是最后一页，r2 是倒数第二页，依此类推。
- 字母 z 代表最后一页，与 r1 相同。
- 页码可以按逗号分隔的任意顺序出现。
- 两个用划号分隔的页码代表从第一页到第二页的包容范围。如果第一个数字高于第二个数字，则表示页面的范围反向。
- 数字或用破折号分隔的数字范围可以加上 x（摘自 qpdf 11.7.1）。这意味着要排除该范围内那些之前没有以 x 开头的页面。
- 该区间可以附加：odd 或：even，以仅从所得区间中选择奇数或偶数位置的页面。在这里，奇偶指的是最终区间的位置，而不是原始页码是奇数还是偶数。

表 3：示例页面范围

1, 6, 4	第 1、6、4 页，按顺序
3-7	第 3 至 7 页，按递增顺序排列
7-3	第 7、6、5、4、3 页，依次阅读
1-2, 按顺序排列所有页面	
Z-1 所有页面按倒序排列	1、3、5-9、15-12 页 第 1、3、5、6、7、8、9、15、14、13 和 12 页 按顺序为 R3-R1
文档的最后三页 R1-R3 文档的最后三页按倒序排列：偶数页从 20 到 20	5、7、9、12 页 第 5、7、8、9 页，以及 12 页
5,7,9,12 页：奇数页 5、8 和 12 页，这些是原始 5、7、8、9、12	5、7、9、12 中奇数位置的页面：偶数页 7 和 9，是原始 5、7、8、9、12 的偶数页，第 1 至 10 页（第 3 页和第 4 页除外，1， 2，以及 5 到 10）
4-10, x7-9, 12-8, xr5	
在 15 页的文件中，依次是 4、5、6、10、12、10、9 和 8。也就是第 4 到第 10 页，除了第 7 到第 9 页，接着是第 12 到第 8 页，除了第 11 页（倒数第五页）	

## 6.9 PDF 修改

修改选项对 PDF 的某些部分进行系统性修改，导致 PDF 的渲染方式与原始内容不同。另见 PDF 变换。

## 6.9.1 相关期权

--页面 [--文件=] 文件 [选项] [...] --

该标志启动页面选择选项，用于从一个或多个输入文件中选择页面，执行拆分、合并和整理文件等作。

有关页面选择的详细信息，请参见页面选择。

另见 --split-pages、--collate、页区。

--file=file

指定当前页面作的文件。该选项与 --pages、--覆盖层和 --底层一起使用，并出现在选项和终止--. 之间

详情请参见页面选择。

--范围=数值范围

用 --pages 指定当前页面作的页面范围。如果省略，所有页面均被选中。该选项用于 --页面，并出现在 --pages 和 --. 之间

详情请参见页面选择。

--对照 [=n[, m, ...]]

该选项使 qpdf 整合而非串接带有--pages 的页面。用数值参数，将 n 个为一组。默认是 1。用逗号分隔的数值参数，从第一个文件取 n，从第二个文件取 m，依此类推。

详情请参见页面选择。

--分页 [=n]

将每组 n 页写入一个独立的输出文件。如果未指定 n，则创建单页。输出文件名的生成方式如下：

- 如果字符串 %d 出现在输出文件名中，则会被从 1 开始的零填充页码范围替代。
- 否则，如果输出文件名以.pdf 结尾（大小写不区分），则在文件扩展名前插入一个零填充页区间，前置破折号。
- 否则，文件名会加上一个零填充页区间，前置破折号。

文件名中的所有页码都添加了零填充，使所有数字长度相同，这导致输出文件名按数字顺序按词法排序。页面范围在单页组中为单一数字，否则为两个数字被破折号分隔。

这里有一些例子。在这些例子中，infile.pdf 有 12 页。

- qpdf --分页 infile.pdf %d-out: 输出文件为 01 输出至 12 输出，无扩展名。
- QPDF --分割页=2 infile.pdf outfile.pdf: 输出文件通过 outfile-11-12.pdf outfile-01-02.pdf
- qpdf --split-pages infile.pdf something.else 会生成文件 something.else-01 通过 something.else-12。扩展 .else 在数字位置上没有特别处理。

请注意，原始 PDF 文件的大纲、线程及其他文档级功能不会被保存。对于每一页输出，该选项创建一个空白的 PDF，并从输出中复制一页到该 PDF 中。如果你需要文档层级的数据，你需要为每页运行一次带有 ---pages 选项的 qpdf。如果不需要文档层面的数据，使用分页会快得多。未来的 qpdf 版本可能会支持保留部分文档级信息。

--叠加文件 [选项] --

在输出上叠加另一个 PDF 文件的页面。  
详情请参见叠加层和底层。

--底层文件 [选项] --

在输出上加另一个 PDF 文件的底层页面。  
详情请参见叠加层和底层。

--平转

对于每一页使用页面字典中的/Rotate 键旋转的页面，移除/Rotate 键，并通过修改页面内容实现相同的旋转语义。这个选项对于准备那些无法正确处理旋转页面的 PDF 应用程序文件非常有用。除非你在绕过某个特定问题，否则通常没有理由使用这个选项。

--flatten-annotations=参数

该选项将注释压缩到页面内容中，并对表单字段进行特殊处理。通常，注释会单独呈现，并且放在页面顶部。将注释合并到页面内容中实际上冻结了注释的位置，使它们在多次页面变换后看起来就一样。支持这个选项的库功能是为了方便那些想创建 n-up 页面布局和其他类似功能、但这些内容不适合注释的程序。参数值可以是以下任一：

表 4：注释参数平坦化

所有标注均包含未标记为隐形或隐藏打印的注释，仅包含应在页面打印时出现的注释，省略不应出现在屏幕上的注释。
--

在 PDF 文件中，交互式表单字段有一个值，并且独立地有一组称为外观的指令，用于渲染填充的字段。如果一个程序填写表单，程序不知道如何更新外观，这些表单可能会与字段的值不一致。如果 qpdf 检测到这种情况，其默认行为是不将这些注释平整，因为这样做会导致表单字段的值丢失。这样你就有机会用懂得如何生成表象的程序回去保存表单。QPDF 本身可以生成外观，但有一些限制。详情请参见 --generate-appearances 选项。  
在一些损坏的文件中，带有交互式表单字段，文档 AcroForm 结构中的表单字段可能与页面上相应的小部件注释不同步。在这种情况下，一些观众在平整后可能会以略微偏移的方式显示某些场值两次。在这种情况下，在扁平化后，使用 --remove-acroform 选项删除文档目录中的 AcroForm 条目可能会有帮助。

--旋转=[+|-]角度[:p 年龄范围]

将指定范围的页面按指定角度旋转，角度必须是 90 度的整数。  
角度的数值可以是 0、90、180 或 270。

关于页区段语法的描述，请参见页区段。如果省略了页数范围，旋转会应用到所有页面。  
如果+在角度前加，角度加起来，因此角度+90 表示顺时针旋转 90 度。如果前加了 -，则减去角度，因此 -90 是一个逆时针 90 度旋转，与 +270 完全相同。  
如果前置 + 或 - 均未加，则旋转角度被精确设定。你几乎总是需要+或-，因为如果不查看 PDF 代码，无法判断看起来被旋转的页面是“自然”旋转的，还是通过指定旋转来确定的。例如，如果页面显示的是横向模式图像，且该图像旋转了 90 度，使图像顶部位于页面右边缘，则无法通过

目视检查图像的字面顶部是否是页面顶部，还是图像的字面顶部是右边，且页面已经在 PDF 中旋转。指定旋转角为-90，无论哪种情况，图像都会显示为直立。绝对旋转角度的使用应保留给你对 PDF 文件结构有具体了解的情况。

示例：

- QPDF in.pdf out.pdf --旋转=+90: 2,4,6 --旋转=+180: 7-8: 将第 2、4、6 页旋转 90 从原始旋转方向顺时针方向
- QPDF in.pdf out.pdf --rotate=+180: 将所有页面旋转 180 度
- QPDF in.pdf out.pdf --rotate=0: 强制每页以自然方向显示，这将消除之前在页面元数据中施加的任何旋转效果。

参见 ---flatten-rotation。

#### --生成外观

如果文件包含交互式表单字段，并显示外观与表单值过时，该标志将重新生成外观，但有一定限制。注意通常没有必要这样做，但在使用 --flatten-annotations 选项之前，有时需要这样做。以下是限制性的总结。

- 单选按钮和复选框的外观使用 PDF 文件中的预设值。QPDF 只是确保根据字段的值显示正确的外观。对于正确创建表单的 PDF 文件来说，这没问题。有些 PDF 编写者会在字段更改时保留“外观”，这可能导致某些控件外观不一致。
- 对于文本字段和列表框，任何不属于 US-ASCII 或检测到“Windows ANSI”或“Mac Roman”编码的字符，都会被替换为性格。QPDF 对字体和编码了解不足，无法正确表示超出该范围的字符。
- 对于默认外观流要求字体自动大小的可变文本字段，则使用固定字体大小而非计算字体大小。
- 夸达被忽视了。夸德用于指定字段内容应与字段对齐的左、中还是右。
- 富文本、多行及其他更复杂的格式指令则被忽略。
- 不支持多选字段或签名字段。

qpdf 生成的外观应足以满足由 ASCII 字符组成的简单表单，且原始文件遵循 PDF 规范并提供了文本字段外观的模板信息。如果 qpdf 对你的表格处理不够好，可以用外部应用程序保存填写好的表格，再用 qpdf 处理。大多数支持填写表单的 PDF 查看器会生成外观流。有些甚至会为填入原始字体字符范围外字符的表单，通过嵌入额外字体来实现。

#### --优化图像

该标志使 qpdf 会使用 DCT 压缩重新压缩所有未使用 DCT (JPEG) 压缩的图像，只要能减少图像数据的字节大小且图像尺寸不低于最小指定尺寸。与 --verbose 结合使用时，提供了有用的信息。另见 --oi-min-width、--oi-min-height 和 --oi-min-area 选项。默认情况下，内联图像会被转换为普通图像并进行优化。使用 --keep-inline-images 来防止内联图片被包含。另见优化文件大小。

#### --oi-min-width=宽度

避免优化宽度低于规定长度的图像。如果省略，默认为 128 像素。用 0 表示无最低限度。

--oi-min-height=高度

避免优化高度低于指定长度的图像。如果省略，默认为 128 像素。用 0 表示无最低限度。

--oi-min-area=像素面积

避免优化像素数量（宽度×高度）低于规定数量的图像。如果省略，默认为 16,384 像素。用 0 表示无最低限度。

--保持内嵌-图像

防止内联图像被包含在由 --optimize-images 进行的图像优化中。

--移除-Acroform

将交互式表单字典排除在输出文件中。该选项仅会将交互式表单词典从文档目录中移除。它不会移除表单字段词典或相关的控件注释。

在一些带有交互式表单字段的损坏文件中，文档 AcroForm 结构中的表单字段可能与页面上相应的小部件注释不同步。在这种情况下，不同的查看器可能显示不同的字段值，在使用 --flatten-annotations 选项平整注释后，有些查看器可能会以轻微偏移的方式重复显示某些字段值。在这种情况下，使用该选项删除文档目录中的 AcroForm 条目可能会有所帮助。用户使用此选项后应仔细检查输出文件。

--删除信息

通过在文档拖尾中的 /Info 字典中省略除 /ModDate 外的所有条目，从而从输出文件中排除文件信息（修改日期除外）。另见 --remove-metadata, --remove-structure。

--删除元数据

通过在文档目录中省略 /Metadata 字典，将元数据排除在输出文件中。另见 --remove-info, --remove-structure。

--移除页面标签

通过在文档目录中省略 /PageLabels 字典，从输出文件中排除页面标签（明确的页码）。另见 --set-page-labels。

--移除结构

通过在文档目录中省略 /StructTreeRoot 和 /MarkInfo 字典，将结构树排除在输出文件中。另见 --remove-info, --remove-metadata。

--套页-标签-标签-规格 --

为整个文件设置页面标签（明确的页码）。这会在文档目录中生成一个 /PageLabels 字典。PDF 文件的页面可以通过页面标签明确编号。PDF 文件中的页面标签有可选类型（阿拉伯数字、大小写字母、大小写罗马数字）、可选前缀和可选起始值，默认为 1。qpdf 页面标签规范的形式为

首页：[类型][开始[/前缀]]

其中

- 首页表示一个连续的页码，采用与页区间相同的格式（参见页区间）：数字，前加 r 表示从末尾计数，z 表示最后一页
- 类型可以是以下一种
  - D: 阿拉伯数字（数字）
  - A: 大写字母字符
  - a: 小写字母字符
  - R: 大写罗马数字

- r: 小写罗马数字
- 省略: 页码不显示, 但如果指定前缀仍会显示
- 起始必须是数字  $\geq 1$
- 前缀可以是任意字符串, 并且会在每个页面标签前加上

第一个标签规范的首页值必须为 1, 表示文档的第一页。如果指定了多页标签规范, 必须按递增顺序给出。给定的页面标签规范会使页面按照该方案编号, 从首页开始, 一直到下一个标签规范或文档结尾。如果你想省略从某页开始的编号, 可以用第一页作为规格。

以下是一些页面标记方案的示例。对于这些例子, 假设文档长达 50 页。

- 第一阶段 5: D
  - 前四页编号为 a 至 d, 剩余页码为 1 至 46。
- 1: r 5: D 12: 14: D/10 r 5: D//A- z:// “结尾注”:
  - 前四页编号为 i 到 iv
  - 第 5 页编号为 1, 页码依序编号至第 11 页, 第 11 页编号为 7
  - 第 12 页和第 13 页将不贴标签
  - 第 14 页编号为 10。页面将依序编号至第 45 页, 第 41 页编号
  - 从第 46 页 (倒数第五页) 开始, 直到第 49 页, 页面将标注为 A-1 至 A-4
  - 第 50 页 (最后一页) 将标注为尾注。

页面标签格式范围的限制如 PDF 规范 ISO 第 12.4.2 节所规定 32000.

另见 --remove-page-labels。

## 6.10 加密

本节描述创建加密文件的选项。关于其他与加密相关的选项, 请参见 ---decrypt 和 --copy-encryption。有关 PDF 加密内部工作的更深入技术讨论, 请参见 PDF 加密。

要创建加密文件, 请使用以下

```
--加密 \ [--用户-密码=用户-密码] \ [--owner-
password=owner-password] \ --bits=密钥长度
[options] --
```

或者

```
--加密用户-密码所有者-密码密钥长度 [选项] --
```

第一种形式，带有密码和位长标志，在 qpdf 11.7.0 中引入。只有 --bits 选项是强制的。这个表格允许你使用任何文本作为密码。如果指定密码，必须在 --bits 选项之前提供。

第二种表格自始至终就存在 qpdf 格式，并将持续支持。用户密码和所有者密码中，其中一个或两个都可以是空字符串。

密钥长度参数必须是 40、128 或 256。用户和/或所有者密码可以省略。省略任一密码可以使 PDF 文件无需密码即可打开。为用户和所有者密码指定相同的值，以及指定空的所有者密码都被认为是不安全的。

加密选项会被——自动终止。

40 位加密不安全，128 位加密不安全，没有 AES 的 128 位加密也是如此。除非有特定原因使用不安全的格式，比如测试或兼容非常老旧的浏览器，否则建议使用 256 位加密。你必须使用 --allow-weak-crypto 标志来创建使用不安全密码算法的加密文件。---allow-weak-crypto 标志出现在 --encrypt ... -- 之外（在 --encrypt 之前或之后）。

如果密钥长度为 256，则最小 PDF 版本为 1.7，扩展为 8 级，且基于 AES 的加密格式即为 PDF 2.0 规范中描述的格式。使用 128 位加密时，PDF 版本至少必须是 1.4，如果使用 AES 则为 1.6。使用 40 位加密时，PDF 版本至少需要 1.3。

当使用 256 位加密时，拥有者密码为空的 PDF 文件安全性较低。要创建此类文件，必须指定 --allow-insecure 选项。可选选项因键长度而异。并非所有读者都尊重所有限制。每个权限选项的默认都是完全允许的。这些限制可能由任何特定读者执行，也可能不会。QPDF 允许非常细致的限制设置。有些读者可能认不出你指定的选项组合。如果你指定了某些限制组合，但遇到的读卡器似乎没有像你预期那样遵守，那很可能不是 qpdf 的 bug。QPDF 本身不遵守文件上已施加的加密限制。这样做毫无意义，因为 qpdf 可以用来完全移除文件中的加密。

以下是加密选项的总结。每个选项的帮助部分都有详细信息。

表 5：仅 40 位加密的选项

--annotate=[y n]	限制评论、填写表单和签名
--extract=[y n]	限制文本/图形提取
--modify=[y n]	限制文档修改
--print=[y n]	限制印刷

表 6：128 位或 256 位加密的选项

--accessibility=[y n]	限制无障碍（通常忽略）
--annotate=[y n]	限制注释/填写表单字段
--assembly=[y n]	限制文档组装
--extract=[y n]	限制文本/图形提取
--form=[y n]	限制填写表单字段
--modify=other=[y n]	限制其他
修改 --modify=modify-opt	控制 按级别修改访问
--print=print-opt	控制 打印访问
--cleartext-metadata	防止元数据加密

表 7：仅 128 位加密的选项

--use-aes=[y n]	表示是否使用 AES 加密
——原力 V4	强制使用 V=4 加密处理程序

表 8: 仅 256 位加密的选项

<code>--force-R5</code>	强制使用已废弃的 R=5 加密算法
<code>--允许-不安全</code>	允许用户密码但拥有者密码为空

表 9: 打印-OPT 的数值

无人禁止打印	
低分辨率仅允许低分辨率打印	
满载	允许全打印

表 10: 修改-opt 的值

不允许任何修改	
Assembly 仅允许文档组装	
表单组装权限加上填写表单字段和签名	
注释表单权限，以及注释和修改表单	
全部	允许全面修改文档

6.10.1 相关选项

`--用户密码=用户密码`

设置加密文件的用户密码。符合标准的读卡器会对使用用户密码打开的文件施加安全限制。

`--owner-password=owner-password`（所有者-密码）

设置加密文件的拥有者密码。符合标准的读卡器允许在使用拥有者密码打开文件时更改或覆盖安全限制。

`--bits={48|128|256}`

为加密文件设置密钥长度。除非你有充分理由创建加密较弱的文件，否则你应该始终使用 `--bits=256`。

`--accessibility=[y|n]`

启用或禁用文本提取功能，以便对视障用户无障碍。默认做法是完全宽容。当 AES 与 128 位加密或 256 位加密一起使用时，qpdf 库会忽略该字段。除非你明确是为了创建测试文件，否则绝不应该禁用无障碍功能。PDF 规范规定，符合规范的读者应忽略此权限，始终允许无障碍访问。

40 位加密时不支持此选项。

`--annotate=[y|n]`

启用或禁用修改注释，包括评论和填写表单字段。默认做法是完全宽容。对于 128 位和 256 位加密，这也支持编辑、创建和删除表单字段，除非也指定了 `--modify-other=n` 或 `--modify=none`。

`--assemble=[y|n]`

启用/禁用文档组装（页面的旋转和重新排序）。默认做法是完全宽容。

40 位加密时不支持此选项。

`--extract=[y|n]`

启用或禁用文本/图形提取功能，目的除了无障碍。默认做法是完全宽容。



--形式=[y|n]

启用/禁用即使禁止修改注释，是否允许填写表单字段。默认做法是完全宽容。

40 位加密时不支持此选项。

--modify-other=[y|n]

启用/禁用那些未被 --assemble、---annotate 或 --form 控制的修改。--modify-other=n 被其他 --modify 选项（除 --modify=all 外）都隐含。默认做法是完全宽容。

40 位加密时不支持此选项。

--modify=modify-opt

对于 40 位文件，modify-opt 只能是 y 或 n，并控制文档修改的所有方面。默认做法是完全宽容。  
对于 128 位和 256 位加密，修改-OPT 值允许启用和禁用限制级别，类似于某些 PDF 创建工具的做法：

表 11: 修改-选择 128 位和 256 位加密

不允许修改，组装允许文档组装，仅表单组装权限，表单设置权限，填写表单字段、签名、注释表单权限，以及注释和修改表单，所有表单都允许完全修改文档（默认）

修改选项对应于更细致的选项，具体如下：

表 12: 映射修改-选择至其他选项

无 --modify-other=n --annotate=n --form=n --assemble=n 组装 --modify-other=n --annotate=n --form=n form --modify-other=n --annotate=n 注释 --modify-other=n 注释 --modify-other=n
所有其他修改选项（默认）

你可以将此选项与上述选项结合使用。如果有，后期选项会覆盖之前的选项。

--print=print-opt

控制允许的印刷类型。默认做法是完全宽容。对于 40 位加密，print-opt 只能是 y 或 n，并启用或禁用所有打印。对于 128 位和 256 位加密，print-opt 可能具有以下数值：

表 13: 印刷-OPT 值

无：禁止打印低分辨率，允许低分辨率打印，仅允许全打印（默认）

--明文元数据

如果指定了，文档中的任何元数据流即使文档其他部分都加密，也将保持未加密状态。这也强制 PDF 版本至少是 1.5。

40 位加密时不支持此选项。

`--use-aes=[y|n]`

启用/禁用更安全的 AES 加密, 采用 128 位加密。指定 `--use-aes=y` 会强制 PDF 版本至少为 1.6。该选项仅支持 128 位加密。默认是 `n`, 出于兼容性考虑。改用 256 位加密。

`--allow-insecure`

允许创建带有 256 位密钥的 PDF 文件, 其中用户密码非空, 所有者密码为空。以这种方式创建的文件不安全, 因为它们可以在没有密码的情况下打开, 且不会执行限制。用户通常不会想创建这样的文件。如果你用 qpdf 故意创建奇怪的文件进行测试 (qpdf 的有效用途!), 这个选项允许你创建这些不安全的文件。该选项仅支持 256 位加密。

关于这个问题的更技术性讨论, 请参见用户和所有者密码。

`---v4`

使用此选项时, 文档加密词典中的 `V` 和 `R` 参数必须设置为 4。因为 qpdf 会自动在需要时这样做, 所以根本没有理由再用这个选项。它主要用于测试 qpdf 本身。这个选项还强制 PDF 版本至少是 1.5。

`---r5`

使用一种未公开、未支持且已弃用的加密算法, 该算法仅存在于 Acrobat IX 版本中。该选项除兼容性测试外不应使用。如有指定, qpdf 在扩展层 3 将最低版本设置为 1.7。

## 6.11 页数选择

QPDF 允许你使用 `--pages` 选项, 通过从一个或多个输入文件中选择页面来拆分和合并 PDF 文件。

```
QPDF primary-input.pdf \ --pages \
\
--file=input.pdf \ [--range=
页数范围] \ [--password=密码] \
[...] \ -- output.pdf
```

或者

```
QPDF primary-input.pdf \ --pages \ input.pdf [--
password=password] [page-range] \ [...] -- output.pdf
```

注释:

- 第一个形式, 包含 `--file` 和 `--range`, 在 qpdf 11.9.0 中引入。在此形式中, `--range` 和 `--password` 选项适用于最近指定的 `--file` 选项。
- 密码选项仅适用于受密码保护的文档。如果你多次指定同一个文件, 只需在第一次输入密码。
- 页数范围可省略。如果省略, 所有页面均包含在内。
- 文档层级信息, 如大纲、标签等, 取自主输入文件 (如上述示例中为 `in.pdf`), 并保存在 `out.pdf` 中。你可以用 `--empty` 代替输入文件, 从空文件开始, 然后从所有文件中平均复制页面。

- 你可以使用 `o` 作为主输入文件的简写，如果不是空的话。

关于指定页数范围的帮助，请参见页码范围。

使用 `--collate=n` 使页面以 `n` 页为一组（默认 1），而不是将输入串接。

使用 `--collate=i, j, k, ...` 先取 `i`，再取 `j`，再取 `k`，再取 `i`，依此类推。

注意，`--collate` 出现在 `--pages` 之外.....（在 `--pages` 之前或之后）。每份文件依次提取页面。当文档页数不足时，会被跳过。请参见下方示例。

### 6.11.1 示例

- 先从 `in.pdf` 开始，附加 `a.pdf` 的所有页面和 `b.pdf` 的偶数页，然后写输出到 `out.pdf`。`in.pdf` 的文档级信息被保留。注意 `o` 的使用。指的是 `in.pdf`。

```
QPDF in.pdf --o a.pdf b.pdf 1-z: 偶数 -- out.pdf
```

- 取 `a.pdf` 的所有页，倒过来 `b.pdf` 页，`c.pdf` 的第 3 页和第 6 页，然后写成 `out.pdf`。文档级元数据从所有输入文件中被丢弃。密码 `x` 用于打开 `b.pdf`。

```
QPDF --empty --pages --file=a.pdf \ --file=b.pdf --  
-password=x --range=z-1 \ --file=c.pdf --  
range=3,6 -- out.pdf
```

- 通过扫描正面为 `odd.pdf`，将背面扫描为 `even.pdf` 来扫描双面打印文件。将结果汇总成 `all.pdf`。这占用了 `odd.pdf` 的第一页、`even.pdf` 的第一页、`odd.pdf` 的第二页、`even.pdf` 的第二页，依此类推。

```
qpdf --整理 odd.pdf --o even.pdf -- all.pdf OR qpdf --collate --empty --pages  
odd.pdf even.pdf -- all.pdf OR qpdf --collate --empty --pages --file=odd.pdf --  
file=even.pdf -- all.pdf
```

- 在整理时，可以指定任意数量的文件和页区间。如果某个文件页数较少，当该文件的页面全部包含后，该文件会被跳过。例如，如果你跑

```
qpdf --collate --empty --pages a.pdf 1-5 b.pdf 6-4 c.pdf r1 -- out.pdf
```

你会按以下顺序获得以下页面：

- a.pdf 第 1 页
- b.pdf 第 6 页
- c.pdf 最后一页
- a.pdf 第 2 页
- b.pdf 第 5 页
- a.pdf 第 3 页
- b.pdf 第 4 页
- a.pdf 第 4 页
- a.pdf 第 5 页

- 你可以指定一个数值参数来 `--collate`。使用 `--collate=n` 时，像往常一样从每个文件中拉取 `n` 页的组，页面数尽时停止。例如，如果你跑

```
QPDF --collate=2 --empty --pages a.pdf 1-5 b.pdf 6-4 c.pdf R1 -- out.pdf
```

你会按以下顺序获得以下页面：

- a.pdf 第 1 页
  - a.pdf 第 2 页
  - b.pdf 第 6 页
  - b.pdf 第 5 页
  - c.pdf 最后一页
  - a.pdf 第 3 页
  - a.pdf 第 4 页
  - b.pdf 第 4 页
  - a.pdf 第 5 页
- 你可以指定多个数值参数来进行 `--collate`。使用 `--collate=i, j, k` 时，先从第一个文件拉取 `i` 页，再从第二个 `j` 个，再从第三个 `k` 个，重复这个过程。参数数必须等于组数。例如，如果你跑

```
qpdf --collate=2,1,3 --empty --pages a.pdf 1-5 b.pdf 6-4 c.pdf r1-r4 -- out.pdf
```

你会按以下顺序获得以下页面：

- a.pdf 第 1 页和第 2 页
  - b.pdf 第 6 页
  - c.pdf 最后三页，顺序相反
  - a.pdf 第 3 页和第 4 页
  - b.pdf 第 5 页
  - 倒数第四页 c.pdf
  - a.pdf 第 5 页
  - b.pdf 第 4 页
- 从 file1.pdf 取第 1 到第 5 页，从 file2.pdf 倒过来取第 11 到 15 页，从文档级元数据取 file2.pdf。

```
QPDF file2.pdf -第 1-5 页 file1.pdf。15-11 -- outfile.pdf
```

- 这里有一个更牵强的例子。如果你想用密码传递的加密文件 `encrypted.pdf` 的第一页，在输出文件中重复两次，且两个副本没有共享数据，并且想丢弃文档级元数据但保持加密，你可以运行

```
qpdf --空 --copy-encryption=encrypted.pdf \
--encryption-file-password=pass \ --pages
encrypted.pdf --password=pass 1 \
./encrypted.pdf --password=pass 1 -- \
outfile.pdf
```

注意我们三次都必须指定密码，因为给出密码 `---encryption-file-password` 不计入页面选择，而在 qpdf 上，密码是加密的。PDF 和 `./encrypted.pdf` 是两个独立的文件。（这是刻意为之。关于讨论，请参见限制。）这些都是大多数用户希望永远不用遇到的角落情况。

6.11.2 局限性

除了页面标签（页码）外，qpdf 尚未完全支持与页面相关的文档级数据处理。某些文档层面的功能，如表单字段、大纲（书签）和文章标签等，会从主输入文件中完整复制。从 qpdf 8.3 版本开始，除非指定 `-remove-page-labels`，否则所有文件的页面标签都保留。

预计未来版本的 qpdf 将拥有更完整且可配置的文档级元数据行为。与此同时，拆分和合并的语义在不同特征中有所不同。例如，文档的大纲（书签）指向实际的页面对象，所以如果你选择某些页面而不选其他页面，指向输出文件中页面的书签会有效，而剩余的书签则无法使用。如果你不想保留主文件的元数据，可以用 `--empty` 作为主输入文件。

访问标有“页面”的 qpdf 期刊，或查看 qpdf 源代码发行版中的 TODO 文件，了解一些想法。

在 qpdf 8.4 版本之前，无法直接从同一文件中指定同一页面多次，因此需要通过多种方式指定同一文件来变通。8.4 版本取消了这一限制，但当同一页面被复制多次时，所有数据会在页面间共享。有时这样做没问题，但有时可能无法正常工作，尤其是当有表单字段或你打算对某一页面进行其他修改时。未来的 qpdf 版本应能更全面地解决这个问题。你可以通过用两种不同的方式指定同一个文件来绕过这个问题。例如 `qpdf in.pdf --pages . 1 ./in.pdf 1`——完毕。PDF 会创建一个包含输入第一页两份副本的文件，且这两个副本不会共享任何共同对象。这包括字体、图片以及页面引用的其他内容。

6.12 叠加与底层

你可以用 qpdf 将其他文件的页面叠加或底层叠加到 qpdf 生成的输出上。指定覆盖层或底层如下：

```
{--叠加|--底层} [--file=]file [options] --
```

叠加和底层选项处理较晚，因此可以与其他选项如合并合并，并应用于最终输出。`--覆盖层`和`---底层`选项的工作原理相同，只是底层页面会在应用页面下方绘制，可能会被原始页面遮挡，而叠加文件则绘制在应用页面上，可能会遮挡页面。qpdf 11.9.0 新增了使用 `--file` 选项指定文件的功能。你可以把叠加和底层结合起来。从 qpdf 11.9.0 开始，你可以多次指定这些选项。最后一页会是一个叠加层，按出现顺序排列底层，然后是原始页面，最后是按出现顺序排列叠加层。

叠加层和底层的默认行为是，页面会依次从叠加层/底层文件中提取，并应用到输出中的相应页面，直到没有更多的输出页为止。如果覆盖层或底层文件页数用尽，剩余输出页将保持原样。这种行为可以通过选项进行修改，选项设置在 `--覆盖层`或`--底层`标记与`--选项`之间。支持以下选项：

`--to=`页区间

指定一个页面范围（参见页面范围），指示输出中哪些页面应应用覆盖层/底层。如果未指定，所有页面都会应用叠加/底层。

--from=[页数范围]

指定一个页面范围，指示覆盖层/底层文件中哪些页面用于叠加或底层。如果未特别说明，所有页面将被使用。“from”页面被使用直到用尽，之后使用带有 ---重复的页面。如果你使用 --repeat 选项，可以使用 --from= 来提供一组空的“from”页面。

--重复=页幅范围

指定一个可选的页面范围，指示覆盖层/底层文件中哪些页面在“发件人”页用完后会重复。如果你想重复从输出的第一页开始的一系列页面，可以显式使用 --from=。

### 6.12.1 示例

- 将文件 o.pdf 的前三页叠加到输出的前三页上，然后将 o.pdf 页的第 4 页叠加到输出的第 4 页和第 5 页上。保持剩余输出页不动。

```
QPDF in.pdf --覆盖层 o.pdf --to=1-5 --from=1-3 --repeat=4 -- out.pdf
```

- 在所有奇数输出页上 footer.pdf 第 1 页底层，所有偶数输出页 footer.pdf 底层为第 2 页。

```
QPDF in.pdf --底层 footer.pdf --from= --重复=1,2 -- out.pdf
```

- 合并两个文件，然后将 watermark.pdf 的单一页面叠加在结果上。

```
QPDF --空页 --pages a.pdf b.pdf -- \
--覆盖 watermark.pdf --from= --重复=1 -- out.pdf
```

## 6.13 嵌入文件/附件

可以列出、添加或删除嵌入文件（也称为附件），并从其他文件复制附件。另见 --list-attachments 和 --show-attachment。

### 6.13.1 相关选项

--添加附件文件 [选项] --

该标志开始添加附件选项，用于向文件添加附件。

--add-attachment 标志及其选项可以重复使用以添加多个附件。详情请参见添加附件选项。

--复制-附件-从文件 [选项] --

该标志启动复制附件选项，用于从其他文件复制附件。

--copy-attachments-from 标志及其选项可以重复使用，以复制多个文件中的附件。  
详情请参见复制附件选项。

--移除附件=键

移除指定的附件。这不仅将附件从嵌入文件表中移除，还清除了文件规范，确保附件实际上不存在于输出文件中。这意味着任何潜在的附件内部链接都会被切断。用 --verbose 查看移除状态。使用 --list-attachments 来查找附件密钥。此选项可重复以移除多个附件。

6.13.2 PDF 日期格式

当需要日期时，日期应符合 PDF 日期格式规范，即 D: yyyymmddhhmmsz，其中 z 要么是大写 Z 表示 UTC，要么是时区偏移，形式为-hh' mm' 或+hh' mm'。负时区偏移表示时间早于 UTC。正偏移表示晚了多久。例如，美国东部标准时间（America/New\_York）为-5' 00'，印度标准时间（亚洲/加尔各答）为+05' 30'。

表 14：PDF 日期示例

D: 20210207161528-05'00'	2021 年 2 月 7 日下午 4: 15: 28
D: 20210207211528Z	2021 年 2 月 7 日 21: 15: 28 UTC

6.13.3 添加附件的选项

这些选项在 --add-attachment 和 --. 之间有效

--key=key

在嵌入文件表中指定用于附件的密钥。它默认使用附加文件名的最后一个元素。例如，如果你说 --add-attachment /home/user/image.png，默认键就是 image.png。

--filename=name

指定附件所需的文件名。这通常是显示给用户的名称，也是大多数图形化 PDF 浏览器保存文件时使用的名称。它默认使用附加文件名的最后一个元素。例如，如果你说 --add-attachment /home/user/image.png，默认键就是 image.png。

--create-date=日期

请以 PDF 格式说明附件的创建日期;默认为当前时间。有关日期格式的信息，请参见 PDF 日期格式。

--moddate=日期

请以 PDF 格式指定附件的修改日期;默认为当前时间。有关日期格式的信息，请参见 PDF 日期格式。

--mimetype=类型/子类型

指定附件的哑剧类型，比如文本/纯文、应用程序/PDF、图片/png 等。qpdf 库并不会自动确定哑剧类型。在类 UNIX 环境中，文件命令通常可以提供这些信息。在 MacOS 中，你可以使用文件 -l 文件名。在 Linux 中，文件名为-i。

实现说明：mime 类型出现在 PDF 文件中的/Subtype 字段，但该字段实际上包含了 mime 类型的完整类型和子类型。这是因为 /Type 在 PDF 中已经有别的含义了。

--description= “text”

为附件提供描述性文本，部分 PDF 查看器显示。

——替换

说明任何带有相同钥匙的现有附件应被新的附件替换。否则，如果已有带有该密钥的附件，qpdf 会出错。

6.13.4 附件复制选项

本节中的选项在 --copy-attachments-from 和 --. 之间有效

--前缀=前缀

只有当复制附件的附件键与文件中已有附件冲突时才需要。在这种情况下，指定的前缀会在每个键前加上。这只影响嵌入文件表中的键，而不影响文件名。PDF 规范并不排除多个附件使用相同文件名。

6.14 PDF 检查

这些选项提供了检查 PDF 文件的工具。当指定本节中的任何选项时，不能给出输出文件。

6.14.1 相关选项

——被加密了

静默退出，使用显示文件加密状态的代码：

表 15：已加密的出口代码

0	该文件是加密的
1	未使用过
2	该文件未加密

即使你不知道密码，这个选项也可以用于密码保护的文件。

这个选项对 shell 脚本很有用。如果有其他选项，则忽略。这个选项和 --requires-password 是互斥的。对于未加密文件，这个选项和“要求密码”退出时状态为 2。

--需要密码

静默退出，并输入代码显示文件密码状态：

表 16：--需要-password 的出口代码

0	必须使用除提供的密码外的密码
1	未使用过
2	该文件未加密
3	文件已加密，且提供了正确的密码（如有）

使用--password 选项来指定测试密码。

选择退出状态 0 表示需要密码，是为了启用类似的代码

```
如果 QPDF --要求密码 file.pdf;则
# 提示密码
fi
```

如果密码中包含了 --password，则使用该密码打开文件，就像正常调用 qpdf 一样。这意味着使用带有 --password 的选项可以用来检查密码的正确性。在这种情况下，退出状态为 3 意味着文件能使用提供的密码。这个选项与 --is-encrypted 互斥。对于未加密文件，这个选项和 --is-encrypted 都会以状态 2 退出。



## ——检查

检查文件结构以及流数据的加密、线性化和编码，并将文件信息写入标准输出。退出状态为 0 表示 PDF 文件的语法正确性。注意 `--check` 在所有文件都有效时不会写入标准错误，所以如果你用它批量程序验证文件，运行时不把输出重定向到 `/dev/null`，只需检查 0 退出码是安全的。

对于 `--check` 报告无错误的文件，可能在流数据内容上仍有错误，或者包含不符合 PDF 规范的结构，但应在语法上有效。如果 `--check` 报告任何错误，qpdf 将以状态 2 退出。有一些可恢复的条件——检查检测到。这些错误作为警告而非错误发布。如果 qpdf 没有发现错误但发现了警告，则会以状态 3 退出。当 `--check` 与其他选项结合时，检查总是在处理其他选项之前完成。对于错误文件，`--check` 会使 qpdf 尝试恢复，之后其他选项实际上会对恢复的文件进行作。以这种方式与其他选项结合使用，有助于手动恢复严重损坏的文件。

另见退出状态。

## ——显示加密

该选项显示文档加密参数。如果拥有者密码已提供，且文件采用了允许用户密码恢复的旧加密格式，文件还会显示文档的用户密码。（有关该功能的技术讨论，请参见 PDF 加密。）`--show-encryption` 的输出包含在 `--check` 的输出中。

## ——显示加密密钥

当显示加密信息时，如给出 `--check` 或 `--show-encryption` 时，将计算或检索的加密密钥显示为十六进制字符串。该值通常对用户无用，但如果指定了 `--password-is-hex-key`，它可以用作 `--password` 的参数。注意，当 PDF 文件被加密时，密码和其他元数据仅用于计算加密密钥，而加密密钥实际上是用于加密的。这使得检索该密钥成为可能。有关技术讨论，请参见 PDF 加密。

## ——校验线性化

检查文件是否线性化，如果是，线性化提示表是否正确。QPDF 并不检查线性化的所有方面。带有线性化错误但其他方面正确的线性化 PDF 文件，几乎总能被 PDF 查看器读取。因此，PDF 线性化中的“错误”被 qpdf 视为警告。

## ——显示线性化

检查并显示线性化提示表中的所有数据。

## ——显示-xref

以人文易读的形式展示交叉引用表或流的内容。交叉引用数据给出了常规对象的偏移量、对象流 ID 和基于 0 的对象流索引，用于压缩对象。这对于带有交叉引用流的文件尤其有用，这些文件以二进制格式存储。如果文件无效且进行了交叉引用表重建，该选项会显示重建后的表中的信息。

## --show-object={trailer|obj[, gen]}

展示给定对象的内容。这对于检测处于对象流中的对象（也称为“压缩对象”）尤其有用。

## ——原始流-数据

当与 `--show-object` 一起使用时，如果对象是流，则将原始（压缩）二进制数据写入标准输出，而非对象内容。避免与其他检测方法结合使用，以避免流数据与其他输出混合。另见 `--filtered-stream-data`。

### --过滤流-数据

当与 `--show-object` 一起使用时, 如果对象是流, 则将过滤后的 (未压缩、可能是二进制) 流数据写入标准输出, 而不是对象的内容。如果流被 qpdf 不支持的过滤, 就会报错。该选项表现得就像指定了 `--decode-level=all` (参见 `--decode-level`), 因此它会解压使用支持有损压缩方案的图像。避免与其他检测方法结合使用, 以避免流数据与其他输出混合。

该选项可与 `--normalize-content` 结合使用。如果你这样做, qpdf 会尝试运行内容归一化, 即使流不是内容流, 这很可能导致无法使用的结果。

另见 `--raw-stream-data`。

### --show-npages

将输入文件中的页数单独打印成一行。由于页面数单独显示在一行中, 如果你需要知道文件中的页数, 这个选项在脚本编写时非常有用。

### ---展览页

显示每个页面、字典对象以及与该页面关联的每个内容流的对象和生成编号。拥有这些信息使检查特定页面中的物品更加方便。另见 `---with-images`。

### ---带图片

当与 `--show-pages` 一起使用时, 还显示了每页图像对象的对象和生成编号。

### --列表附件

显示每个嵌入文件的密钥和流编号。使用 `--verbose`, 还会显示更多信息, 包括首选文件名、描述、日期等。密钥通常 (但不总是) 等于文件名, 且是其他选项所必需的。另见嵌入文件/附件。请注意, 这个选项会显示 PDF 时间戳语法中的日期。当附件信息包含在 json 输出中的“附件”键 (参见 `--json`) 时, 日期会以 ISO-8601 格式显示 (仅在该对象内)。

### --show-attachment=key

将指定附件内容写入标准输出的二进制数据。密钥应与 `--list-attachments` 显示的某个密钥匹配。如果该选项多次出现, 则只显示最后一个附件。另见嵌入文件/附件。

## 6.15 JSON 选项

可以查看 PDF 文件的 JSON 格式信息。有关 qpdf JSON 格式的详细信息, 请参见 qpdf JSON。

### 6.15.1 相关期权

#### --json[=version]

生成文件的 JSON 表示。这在 qpdf JSON 中有详细描述。版本参数可用于指定应输出的 qpdf JSON 格式的哪个版本。版本号应为数字或最新。默认是最新的。截至 qpdf 11, 最新版本为 2。如果你有读取 qpdf JSON 输出的代码, 可以通过输出中的“version”键判断你使用的 JSON 输出版本。使用 `--json-help` 选项获取 JSON 对象的描述。

从 qpdf 11 开始, 当指定该选项时, 输出文件为可选 (以保证向后兼容), 并默认为标准输出。你可以指定一个输出文件来将 JSON 写入文件, 而不是标准输出。(示例: `qpdf --json in.pdf out.json`)

只有在指定 `--json-output` 或传递到 `--json-stream-data` 的值时, 才会包含流数据。

`--json-help[=version]`

通过将与 qpdf 生成的 JSON 相同的结构写入标准输出，描述对应版本的 JSON 输出格式。在 `--json-help` 编写的输出中，每个键的值都是该键的描述。qpdf 在 JSON 表示中所保证的具体合同，在 qpdf JSON 中有更详细的解释。帮助的默认版本是版本 2，就像 `--json` 标志一样。

`--json-key=key`

这个选项是可以重复的。如果给出，只有指定的顶层密钥会包含在 JSON 输出中。否则，所有钥匙都将包含在内。如果未提供，所有键都会被包含，除非指定了 `--json-output`，在这种情况下，默认只包含“qpdf”键。如果没有给出 `--json-output`，版本键和参数键总是会出现在输出中。

`--json-object={trailer|obj[, gen]}`

这个选项是可以重复的。如果给出，只有指定的对象会在 JSON 输出的对象字典中显示。否则，所有物体都会被显示出来。有关 qpdf JSON 格式的详细信息，请参见 qpdf JSON。

`--json-stream-data={none|inline|file}`

当与 `--json` 一起使用时，该选项控制 JSON 输出中的流是省略、内联写入（base64 编码）还是写入文件。如果选择文件，该文件将是输出文件的名称，后加上 `-nnn`，其中 `nnn` 是目标编号。流数据文件前缀可以用 `--json-stream-` 前缀覆盖。默认值为 `non`，除非指定了 `--json-output`，此时默认值为内联。

`--json-stream-prefix=file-prefix`

当与 `--json-stream-data=file` 一起使用时，`--json-stream-data=file-prefix` 设置了流数据文件的前缀，覆盖默认设置，即使用输出文件名。这里给出的内容会加上 `-nnn`，生成包含 `nnn` 对象流数据的文件名称。

`--json-output[=version]`

这意味着在指定版本下使用 `--json`。该选项会更改若干默认值，所有这些值都可以通过指定指定选项来覆盖：

- `--json-stream-data` 的默认值会从无变为内联。
- `--decode`-级的默认值会从泛化变为无。
- 默认情况下，JSON 输出中只包含“qpdf”键，但你可以用 `--json-key` 添加额外的键。
- “version”和“parameters”键将被排除在 JSON 输出中。

如果你想像在 QDF 模式下（参见 QDF 模式）一样轻松查看流的内容，可以使用 `--decode-level=generalized` 和 `--json-stream-data=file`，方便地实现这一点。

`--json-input`

将输入文件视为 qpdf JSON 格式的 JSON 文件。输入文件必须完整，包含所有流数据。JSON 版本必须至少是 2。顶层键除“qpdf”外，所有顶层键均被忽略。有关 PDF 与 JSON 之间的转换信息，请参见 qpdf JSON。

`--更新-from-json=qpdf-json-file`

此选项是从指定的 qpdf JSON 文件中更新 PDF 文件。有关如何使用该选项的信息，请参见 qpdf JSON。

## 6.16 全球期权

以下选项会调整 qpdf 的整体行为。这包括修改实现限制和改变作模式。

## 6.16.1 相关选项

--全局[选项] --

开始设置全局选项和限制。

## 6.16.2 全局限制

QPDF 使用多种全局限制来保护自己免受损坏和专门构建的 PDF 文件的影响。没有这些限制，这些文件可能导致 qpdf 崩溃和/或消耗过多的处理器和内存资源。极少数合法 PDF 文件超过这些限制，但在必要时，可以通过以下选项修改或完全取消这些限制。

---无默认限制

禁用所有可选的默认限制。明确设定的限制不受影响。某些限制，尤其是那些设计用于防止堆栈溢出的限制，无法通过此选项移除，但可以进行修改。如果是这种情况，相关选项的条目中会有说明。

### 解析器限制

--解析器最大嵌套=n

在解析对象时设置最大嵌套级别。最大嵌套级别不会被无默认限制禁用。默认是 499。

--解析器最大错误=n

设置解析间接对象时允许的最大错误数。值为 0 表示不施加最大值。默认是 15。

--解析器最大容器大小=n

在解析时设置容器中允许的最大顶层对象数量。当 PDF 文档的 xref 表未损坏且对象本身可以解析且无错误时，该限制适用。默认上限为 4,294,967,295。

另见 --parser-max-container-size-damaged。

--解析器最大容器大小-损坏=n

在解析时设置容器中允许的最大顶层对象数量。当 PDF 文档的 xref 表损坏或对象本身损坏时，该限制适用。这个限制在解析 xref 流时同样适用。

默认上限是 5,000。另见 --parser-max-container-size。

### 流与滤波器极限

--max-stream-filters=n

过滤流时设置允许的最大过滤器数量。过滤流过滤器过多通常表明文件损坏或结构特殊。如果某条溪流的极大值超过，该溪流将被视为不可过滤。默认限制是 25。

## 6.17 测试或调试选项

以下选项在编写包含 qpdf 创建文件的自动化测试代码或测试 qpdf 本身时非常有用。在对 qpdf 进行更改时，会注意避免无谓更改 PDF 文件的输出。这是为了方便在测试套件中直接对比 qpdf 创建的文件。然而，没有保证 PDF 输出不会发生变化，比如 qpdf 生成的某些部分被修复或功能增强。

### 6.17.1 幂性

关于字节逐字节内容的幂等性注意：一般情况下不期望 qpdf 是幂等的。换句话说，当 qpdf 运行在自身输出上时，并不期望它会生成逐字节相同的输出，尽管它会生成语义相同的文件。这背后有多种原因，包括文档 ID 生成（其中包含随机元素），以及流长度编码与字典键排序的相互作用。

通过使用 --确定性 id（或仅测试时的 --static-id）选项搭配 qpdf，并运行三次，从而对其输出处理，从而获得幂等等现象。例如，在以下命令序列中：

```
QPDF any-file.pdf 1.pdf QPDF --确定性 ID
1.pdf 2.pdf QPDF --确定性 ID 2.pdf 3.pdf
```

文件 2.pdf 和 3.pdf 应字节逐字节相同。qpdf 测试套件依赖于这种行为。另见 --static-aes-iv，该方法也应仅用于测试。

### 6.17.2 相关期权

--静态 ID（静态 ID）

文档 ID 使用固定值（拖尾片中的/ID）。这只供测试使用。千万别用它来做生产文件。如果你每次都想为某个文件获得相同的 ID，且没有生成加密文件，可以考虑使用--deterministic-id 选项。

——静电艾斯四

AES-CBC 使用静态初始化矢量。这仅用于测试，以便输出文件能够被复现。千万别用它来做生产文件。这个选项尤其不安全，因为它显著削弱了加密。结合 --static-id 并使用幂零度中描述的三步过程，可以用 PDF 文件创建逐字节幂等元输出，这些文件采用 256 位加密，以辅助创建可重复的测试套件。

--linearize-pass1=file

将第一遍线性化写入命名文件。最终生成的文件不是有效的 PDF 文件。该选项仅用于调试 QPDFWriter 的线性化代码。当 qpdf 对文件进行线性化时，它会分两次写入文件，第一遍用于计算提示表和线性化词典所需的大小和偏移量。通常，第一次通路会被弃用。该选项允许捕获，允许在将第 1 遍计算的值插入文件 2 前检查文件。

--test-json-schema

qpdf 测试套件用此工具检查 qpdf --json 输出与 qpdf --json-help 输出之间的一致性。该选项会导致生成的 JSON 在内存中多出现一份副本，因此不适合处理大型文件。这也是为什么默认情况下它不是开启的。

--报告-内存-使用

qpdf 的性能测试套件利用该数据报告支持环境中最大内存使用量。

## 6.18 Unicode 密码

在库 API 层面，所有执行加密和解密的方法都将密码解释为字节串。呼叫者有责任确保他们的编码得到适当。从 qpdf 8.4.0 版本开始，qpdf 将尝试让你在通过命令行界面与 qpdf 交互时更方便。PDF 规范要求用于加密 40 位或 128 位加密文件的密码必须采用 PDF 文档编码。这种编码

是一种单字节编码, 支持 ISO-Latin-1 及其他一些常用字符。它与 Windows ANSI 有很大重叠, 但并不完全相同。通常没有办法在命令行中提供 PDF 文档编码字符串。因此, 8.4.0 之前的 qpdf 版本常常会生成无法用其他软件打开的 PDF 文件, 当获得非 ASCII 字符的密码以加密 40 位或 128 位加密文件时。从 qpdf 8.4.0 开始, qpdf 识别参数编码并根据需要转码。本节其余部分详细介绍了 qpdf 的具体行为方式。大多数用户不需要知道这些信息, 但如果你一直在绕过 qpdf 的旧行为, 或者用 qpdf 生成加密文件来测试其他 PDF 软件, 可能会很有用。

关于 Windows 的一个说明: qpdf 构建时, 它会尝试确定在 Windows 上使用 wmain 而不是 main 需要做什么。wmain 函数是一个替代入口点, 所有参数以 UTF-16 编码字符串的形式接收。当 qpdf 以这种方式启动时, 它会将所有字符串转换为 UTF-8 编码, 然后调用常规主编码。这意味着, 就 qpdf 而言, 它的命令行参数采用 UTF-8 编码, 就像在任何现代 Linux 或 UNIX 环境中一样。

如果文件采用 40 位或 128 位加密, 且提供的密码不是有效的 UTF-8 字符串, qpdf 会退回到将密码解释为字节串的行为。如果你有旧脚本通过将 iconv 输出传递到 qpdf 来加密文件, 那就不需要再这么做了, 但如果需要, qpdf 应该还是能用。唯一的例外是极不可能出现的密码, 虽然用单字节编码, 但密码是有效的 UTF-8。这样的密码会包含偶数字符串, 字符在带重音的字母和符号之间交替出现。如果你极不可能故意使用这些密码, 而 qpdf 将其解读为 UTF-8 阻挠, 你可以使用 `--password-mode=bytes` 来抑制 qpdf 的自动行为。

如本章前述所述, 可以使用 `--password-mode` 选项来更改 qpdf 对所提供密码的解释。使用这个选项的理由非常少。其中一种是前一段描述的不太可能的情况, 即提供的密码恰好是有效的 UTF-8, 但实际上不应该是 UTF-8。你最好的办法是直接提供一个有效的 UTF-8 字符串密码, 但你也可以使用 `--password-mode=bytes`。另一个使用 `--password-mode=bytes` 的原因是故意生成用未正确编码密码加密的 PDF 文件。qpdf 测试套件通过生成无效文件, 以测试其密码恢复能力。如果你试图为类似目的故意创建错误文件, bytes 密码模式可以让你做到这一点。

当 qpdf 尝试解密包含非 ASCII 字符的密码时, 它会通过将密码解释为几种不同的编码系统, 然后将其转码为所需格式, 生成一系列备用密码。这有助于弥补错误编码系统中提供的密码, 比如之前用过的 iconv 变通方法时可能出现的情况。它还通过反向作生成密码: 将密码编码错误地从正确转换。这将使 qpdf 能够使用被加密文件的软件错误编码的密码解密文件, 包括未正确编码密码的旧版 qpdf。这两种恢复方法的结合, 应该能让 qpdf 透明地打开大多数加密文件, 密码输入正确但编码系统错误。这种行为没有什么真正的缺点, 但如果你不想让 QPDF 这样做, 可以使用 `--suppress-password-recovery` 选项。这样做的一个原因是确保你知道文件被加密的确切密码。

通过这些变化, qpdf 现在在大多数情况下都能生成符合规定的密码。当然也有一些例外。特别是, PDF 规范要求合规编写者规范 Unicode 密码, 并对带有双向文本的密码进行某些转换。实现此功能需要使用真正的 Unicode 库, 比如 ICU。如果使用 qpdf 的客户端应用需要这样做, qpdf 库会接受生成的密码, 但 qpdf 本身不会执行这些转换。未来版本的 qpdf 可能会对此进行处理。

QPDFWriter 方法允许对输出文件进行加密, 接受密码作为字节串。

请注意, `--password-is-hex-key` 选项与此无关。该标志完全绕过了从密码到加密密钥的正常过程, 允许直接指定原始加密密钥。这种行为对取证目的或暴力破解未知密码文件非常有用, 与文档的实际密码无关。

## 6.19 优化文件大小

虽然 qpdf 的主要功能不是优化 PDF 文件大小，但你可以做一些事情来让文件变小。请注意，qpdf 不会重新采样图像或进行修改内容的优化，除非可能使用 DCT（JPEG）压缩对图像进行重新压缩。

以下选项将为您提供 qpdf 能生成的最小文件：

- --压缩-streams=y：确保流被压缩（参见--compress-streams）
- --解码级=推广：应用任何非专用滤波器（参见 --解码级）
- --recompress-flate：解压并重新压缩已经用 zlib（flate）压缩的流
- --压缩水平=9：使用最高压缩水平（参见 Zopfli 压缩算法和——压缩级别）
- --优化图像：如果用 JPEG 替换非 JPEG 图像，可以减少它们的尺寸。并非所有类型的图片都支持，但 qpdf 只会保留支持并缩小尺寸的图片。图像不会被重新采样，但请记住 JPEG 是有损的，因此图像可能会有伪影。这些通常对普通观察者来说是察觉不到的。
- --jpeg-quality=n：设置 --optimize-images 在编写 JPEG 文件时所使用的 JPEG 质量。对于更小、质量较低的图像，n 的数字较小。大多数 JPEG 库的默认设置是 75。数字越少，图像质量越低，但又更小。qpdf 12.1 中新增了 --jpeg-quality 选项。这只适用于与 ---optimize-images 结合使用。
- --object-streams=生成：生成对象流，这意味着 PDF 文件中更多的结构内容会被压缩（参见--object-streams）

## 6.20 Zopfli 压缩算法

如果 qpdf 是支持 zopfli 的（参见支持 zopfli 构建），你可以让 qpdf 用 Zopfli Compression 算法代替 zlib。在此模式下，qpdf 速度较慢，但输出的压缩量略小。（根据他们的文档，zopfli 比 zlib 慢大约 100 倍，输出比其他库的最佳压缩效率高约 5%。）为了让这个功能有用，你应该运行带有重新压缩流选项的 qpdf。请参见“优化文件大小”一节。要使用 zopfli 外，除了支持 zopfli 外，你还必须将 QPDF\_ZOPFLI 环境变量设置为非禁用值。注意，在使用 zopfli 时，压缩级别没有影响，因为 zopfli 总是优先优化大小。

以下是支持的 QPDF\_ZOPFLI 数值：

表 17: QPDF\_ZOPFLI 数值

禁用或未设置，即使有 Zopfli 也不要使用	
如果有，静音使用 Zopfli;否则，默然回归 ZLIB	
如果有强制使用 ZOPFLI 强制执行;如果没有其他值，则以错误失败	
如果有，使用 ZOPFLI;否则发出警告并撤回 ZLIB	

注意，警告和错误行为由 QPDFJob 管理，并影响 qpdf 可执行文件。对于直接使用 qpdf 库的代码，行为是 zopfli 启用时设置了除禁用外的任何值，但会默然退回到 zlib。如果你希望你的应用程序在 zopfli 上表现得和 qpdf 执行文件一样，可以调用 PL\_Flate::zopfli\_check\_env ()。请参阅 qpdf/PL\_Flate.hh 的文档。





## QDF 模式

在 QDF 模式下，qpdf 生成我们称之为 QDF 形式的 PDF 文件。QDF 格式的 PDF 文件，有时称为 QDF 文件，是一种完全有效的 PDF 文件，其第三行为 %QDF-1.0（在 pdf 头和二进制字符之后），并且具有某些其他特征。QDF 格式的目的是使得在普通文本编辑器中编辑 PDF 文件成为可能，但对其有一定限制。这对于尝试不同的 PDF 结构或对 PDF 文件进行一次性编辑非常有用（当然还有其他原因导致这方法不一定总是有效）。注意，QDF 模式不支持线性化文件。如果你启用线性化，QDF 模式会自动被禁用。

在文本编辑器中编辑 PDF 文件通常非常困难，原因有两个：PDF 文件中大多数有意义的数据都被压缩了，而且 PDF 文件充满了偏移量和长度信息，这使得添加或删除数据变得困难。QDF 文件的组织方式是，如果编辑内容在某些约束范围内，随 qpdf 一起发布的 fix-qdf 程序能够将编辑后的文件恢复到正确状态。

```
fix-qdf [文件名 [outfilename]]
```

在没有参数的情况下，fix-qdf 从标准输入读取可能被编辑的 QDF 文件，并将修复后的文件写入标准输出。你也可以将输入和输出文件指定为命令行参数。对于一个参数，该参数被取为输入文件。有两个参数时，第一个参数是输入文件，第二个参数是输出文件。

关于在编辑器中处理 PDF 文件的另一种方法，请参见 qpdf JSON。使用 qpdf JSON 格式可以让你语义编辑 PDF 文件，而无需担心 PDF 语法。然而，QDF 文件实际上是有效的 PDF 文件，因此如果用 PDF 阅读器预览，反馈周期可能会更快。另外，因为 QDF 文件是有效的 PDF，你可以尝试 PDF 文件的所有方面，包括语法。

以下属性是 QDF 文件的特征：

- 所有对象在 PDF 文件中以数字顺序出现，包括对象出现在对象流中。
- 对象以易读格式印刷，所有行尾都规范化为 UNIX 行尾。
- 除非特别覆盖，否则流看起来是未压缩的（当 qpdf 支持过滤器并采用非损压缩方案压缩时），大多数内容流是归一化的（行尾被转换为 UNIX 风格的换行）。
- 所有流的长度都表示为间接对象，流长度对象总是流之后的下一个对象。如果流数据没有以换行结尾，会插入一个额外的换行，并在流后出现特殊注释，表示已完成换行。
- 如果 PDF 文件包含对象流，如果对象流 n 包含 k 个对象，则这些对象编号为 n+1 到 n+k，且每个对象的对象编号/偏移对显示在单独的行中。此外，对象流中的每个对象前面都会有一个注释，标明其对象编号和索引。这使得在对象流中查找对象变得非常容易。
- 所有对象的起始、流标记、终流标记和 endobj 标记单独出现在行中。每个 endobj 标记后面都有一行空行。

- 如果存在交叉引用流，则为未过滤。
- 页面词典和页面内容流带有特别注释，便于查找。
- 每个对象前都有注释，标注原始文件中对应对象的对象编号。

编辑 QDF 文件时，只要保持上述约束，任何编辑都可以进行。这意味着你可以自由编辑页面内容，不用担心弄坏 QDF 文件。也可以添加新对象，只要这些对象是在文件中最后一个对象之后添加的，或者后续对象被重新编号。如果 QDF 文件里有对象流，你总可以在 xref 流之前添加新对象，然后更改 xref 流的编号，因为一般没有什么文件会用数字来引用它。

通常在不扰乱对象编号的情况下从 QDF 文件中移除对象并不现实，但如果你移除了对该对象的所有引用而不移除该对象本身（通过移除所有指向该对象的间接对象），这样该对象将不会被引用。然后你可以在文件上运行 qpdf（运行 fix-qdf 后），qpdf 会省略现在被孤儿化的对象。

当 fix-qdf 运行时，它会对文件进行重新计算，并重新计算文件的以下部分：

- 所有对象流字典的 /N、/W 和 /First 键
- 表示对象编号和对象流中对象偏移量的数字对
- 所有流路长度
- 交叉引用表或交叉引用流
- 在 startxref 令牌之后，偏移量指向交叉引用表或交叉引用流

## 使用 QPDF 库

### 8.1 使用 C++ 中的 qpdf

qpdf 包的源代码树有一个示例目录，包含一些示例程序。libqpdf/QPDFJob.cc 源文件也是一个有用的示例，因为它几乎涵盖了 qpdf 库的全部公共界面。关于该库本身最好的文档来源是阅读 include/qpdf/QPDF.hh、include/qpdf/QPDFWriter.hh 和 include/qpdf/QPDFObjectHandle.hh 中的评论。

所有头文件都安装在 include/qpdf 目录中。建议你使用 `#include < qpdf/QPDF`。

HH>与其把 include/QPDF 加到 include 路径里，

QPDF 安装了一个名为 libqpdf 的 pkg-config 配置，以及名为 qpdf 的 cmake 配置。libqpdf 目标导出到 qpdf:: 命名空间中。以下是单文件可执行文件的 CMakeLists.txt 文件示例，链接于 qpdf:

```
cmake_minimum_required (版本 3.16) 项目 (某些应用语言 CXX)
find_package (qpdf) add_executable (某些应用 some-
application.cc) target_link_libraries (某些应用 qpdf: :
libqpdf)
```

qpdf 库在多线程程序中是安全的，但不能同时在多个线程中使用单个 qpdf 对象实例（包括 QPDF、QPDFObjectHandle 或 QPDFWriter）。多个线程可以同时处理这些及所有其他 qpdf 对象的不同实例。

### 8.2 使用其他语言的 qpdf

qpdf 库采用 C++ 实现，这使得直接在其他语言中使用变得困难。有几件事可以帮助你。

#### “C”

qpdf 库包含一个 “C” 语言接口，提供整体功能的子集。头文件 qpdf/qpdf-c.h 包含了关于其用途的信息。只要你用 C++ 链接器，你就可以用 qpdf 链接 C 程序并使用 C API。对于能够直接从共享库加载方法的语言，C API 也很有用。有人报告通过直接调用 DLL 中的函数，成功使用其他语言的 C API。

#### 蟒蛇

一个名为 pikepdf 的 Python 模块为 qpdf 库提供了一套干净且功能强大的 Python 绑定。使用 pikepdf，你可以以自然的方式处理 PDF 文件，并将 qpdf 的功能与 Python 丰富标准库和可用模块提供的其他功能结合起来。

#### 其他语言

从版本 11.0.0 开始，qpdf 命令行工具可以生成明确的 JSON 表示

也可以使用该 JSON 表示创建或更新 PDF 文件。qpdf 版本从 8.3.0 到 10.6.3 的 JSON 输出格式更为有限。qpdf JSON 格式使得从命令行或任何能处理 JSON 数据的语言中，检查和修改 PDF 文件的结构到对象层面成为可能。详情请参见 qpdf JSON。

#### 包装

qpdf 维基包含了围绕 qpdf 的封装器列表。这些设备的功能性和完整性各不相同。如果你知道（或已经写过）想要包含的包装，<https://github> 开启一期。并请求将其加入列表中。

## 8.3 关于 Unicode 文件名的说明

当字符串以 `char*` 或 `std::string` 形式传递到 qpdf 库例程时，除非另有说明，否则它们被视为字节数组。当需要 Unicode 时，qpdf 需要 UTF-8，除非在头文件中的注释中另有说明。在现代 UNIX/Linux 环境中，这通常是正确的做法。在 Windows 中，情况会更复杂一些。从 qpdf 8.4.0 开始，包含 Unicode 字符的密码处理得到更好的处理，从 qpdf 8.4.1 开始，库尝试正确处理文件名中的 Unicode 字符。特别是在 Windows 中，如果 QPDF 或 QPDFWriter 文件中使用 UTF-8 编码字符串，内部会转换为 `wchar_t*`，并使用支持 Unicode 的 Windows API。因此，只要文件名为 UTF-8 编码并传递到 qpdf 库 API，qpdf 通常能在名称中非 ASCII 字符的文件正常运行，但仍存在一些粗糙之处，比如错误消息或 CLI 输出消息中文件名的编码。对于 Windows 中 Unicode 文件名持续存在的问题，欢迎补丁或 bug 报告。

## 弱密码学

如需帮助解决 qpdf 11.0 及更新版本编译器错误，请参见 qpdf 11.0 中的 API 破坏性变更。

自 2006 年以来，PDF 规范提供了在不使用弱加密的情况下创建加密 PDF 文件的方法，尽管许多 PDF 阅读器和写入者花了几年时间才跟上进度。仍然需要支持弱加密算法来读取使用弱加密算法创建的加密 PDF 文件，包括所有在现代格式引入或广泛支持之前创建的 PDF 文件。

从 10.4 版本开始，qpdf 开始采取措施减少用户误创建加密不安全 PDF 文件的可能性，但将继续允许在明确确认的情况下无限期创建此类文件。

qpdf 11 对弱密码使用的限制更加严格。

### 9.1 弱密码算法的定义

我们将弱加密算法分为两类：弱加密和弱哈希。加密是将数据编码成需要某种密钥来解码数据。哈希是指从数据中创建一个短值，使得找到两份具有相同哈希值的文档（已知存在哈希碰撞）极不可能，且极难有意创建具有特定哈希值的文档或两份具有相同哈希值的文档。

当我们说加密算法弱时，要么意味着发现了数学缺陷，使其本质上不安全，要么意味着算法足够简单，现代计算机技术使得使用“暴力破解”成为可能。例如，当 40 位密钥最初被引入时，尝试所有可能的密钥并不现实，但如今这已经成为可能。

当我们说哈希算法弱时，意思是，无论是由于数学缺陷还是复杂度不足，在计算上可以有意构造哈希碰撞。

虽然应始终避免弱加密，但在某些情况下，在安全性不为因素时，使用弱哈希算法是安全的。例如，弱哈希算法不应作为测试文件是否被篡改的唯一机制。换句话说，你不能用弱哈希作为数字签名。不过，只要允许哈希碰撞，使用弱哈希作为排序或索引文档的方式并无妨碍。弱哈希作为校验和也很常见，这通常用于检查文件在传输或存储过程中未受损，但为了真正的完整性，强哈希会更好。

注意，qpdf 必须始终保留对弱加密算法的支持，因为这是读取使用该算法的旧 PDF 文件所必需的。此外，qpdf 始终保留使用弱密码算法创建文件的能力，因为作为开发工具，qpdf 明确支持创建旧版或已弃用的 PDF 文件类型，这些文件有时是测试或处理旧版软件所必需的。即使其他密码库停止支持 RC4 或 MD5，qpdf 也可以随时回退到其内部实现这些算法，因此这些算法不会从 qpdf 中消失。

## 9.2 弱加密在 qpdf 中的应用

当 PDF 文件使用 40 位加密或 128 位加密（不含 AES）时，则使用弱 RC4 算法。你可以通过始终使用 256 位加密来避免在 qpdf 中使用弱加密。除非你是想创建那些需要用 2010 年前的 PDF 阅读器打开的文件（那时大多数阅读器已经支持更强的加密算法），或者专门为测试或类似目的创建不安全的文件，否则没有理由使用除 256 位加密以外的任何方式。

默认情况下，qpdf 拒绝写入使用弱加密的文件。你可以通过指定 `--allow-weak-crypto` 选项来明确允许。

在 qpdf 11 中，所有可能导致文件被弱加密编写的库方法都被弃用，启用弱加密的方法要么被明确命名以表明这一点，要么接受必要的参数以支持不安全行为。

有一个例外：当加密参数从输入文件或其他文件复制到输出文件时，没有禁止甚至警告使用不安全的加密。原因在于许多 qpdf 作只是保留了现有的加密内容，而要求确认以维护不安全的加密，会导致在对已加密的文件进行非加密作时，qpdf 会崩溃。在这种情况下失败或触发警告，很可能会让人们盲目使用“允许弱加密”选项，这比直接放任这些文件、让明显、有意识地选择弱加密更易被发现更糟糕。你可能会问，为什么这既适用于复制加密，也适用于默认保持行为的加密？答案是——复制加密是输入未加密文件，这使得工作流程成为可能，比如从文件开始，解密以防万一，执行一系列作，然后如果有加密，再重新应用原始加密。此外，可能有一个用于加密的模板，可以应用于各种输出文件，如果每个输出文件都被警告会很烦人。

## 9.3 弱哈希在 qpdf 中的应用

PDF 规范在多个地方使用了弱 MD5 哈希算法。虽然 MD5 被用于加密算法，但在使用 256 位加密时，破解 MD5 不足以破解加密文件，因此使用 256 位加密足以避免在安全敏感性事件中使用 MD5。

MD5 的使用方式包括以下非安全敏感性：

- 文档 ID 的生成。文档 ID 是文档加密的输入参数，但本身并不被认为是安全的。它们本应是唯一的，但在非加密的 PDF 文件中并不防篡改，必须容忍哈希碰撞。

PDF 规范建议但不要求在生成文档 ID 时使用 MD5。通常文档 ID 生成还会随机生成。有一个针对 qpdf 的特性，可以生成确定性 ID（参见 `--deterministic-id`），该功能同样使用 MD5。虽然确实可以更改确定性 ID 算法以不使用 MD5，但这样做会破坏之前所有确定性 ID（使该功能在许多情况下变得无用），而且几乎没有益处，因为即使是安全生成的文档 ID 本身也不是安全敏感值。

- 嵌入文件流中的校验和——PDF 规范规定了使用 MD5。

因此，无法完全避免 MD5 与 qpdf 的使用，但只要你使用 256 位加密，它就不会以安全敏感的方式使用。

## 9.4 qpdf 11.0 中的 API 破坏性变更

在 qpdf 11 中，移除了若干弃用的函数和方法。这些方法提供的 API 不完整。qpdf 8.4.0 中添加了替代选项。被移除的功能包括

- C API: qpdf\_set\_r3\_encryption\_parameters、qpdf\_set\_r4\_encryption\_parameters、qpdf\_set\_r5\_encryption\_parameters、qpdf\_set\_r6\_encryption\_parameters
- QPDFWriter: 这些方法的超载版本, 参数更少: setR3EncryptionParameters, setR4EncryptionParameters、setR5EncryptionParameters 和 setR6EncryptionParameters。

此外, 剩余的函数/方法名称被更改, 以表明它们不安全, 并迫使开发者做出决策。如果你想继续使用不安全的密码算法并创建不安全的文件, 你可以修改代码, 只需在函数末尾添加 `_insecure` 或 `Insecure` 即可。(注意部分 C 函数中 `2` 的消失。)更好的是, 你应该迁移代码, 使用 QPDFWriter.hh 中文档中描述的更安全的加密。使用 R6 方法 (或其对应的 C 函数) 创建 256 位加密的文件。

表 1: 重命名函数

旧名	新名称
qpdf_set_r2_encryption_parameters	qpdf_set_r2_encryption_parameters_insecure
qpdf_set_r3_encryption_parameters2	qpdf_set_r3_encryption_parameters_insecure
qpdf_set_r4_encryption_parameters2	qpdf_set_r2_encryption_parameters_insecure
QPDF 写作:: setR2 加密参数 QPDF 写作:: setR2 加密 不安全 QPDF 写作:: setR3 加密参数 QPDF 写作:: setR3 加密参数不安全 QPDF 写作:: setR4 加密参数 QPDF 写作:: setR4EncryptionParametersInsecure	





## QPDF JSON

### 10.1 概述

从 qpdf 11.0.0 版本开始，qpdf 库和命令程序可以生成 PDF 文件的 JSON 表示。qpdf 11 版本引入了 JSON 格式 2。在 qpdf 11 之前，8.3.0 版本及以后，JSON 表示更为有限，只能通过命令行访问。关于具体变化，请参见“从 JSON v1 到 v2 的更改”。本章其余部分将文档说明 QPDF JSON 版本 2。

请注意：本章讨论的是 qpdf JSON 格式，表示 PDF 文件的内容。这与 QPDFJob JSON 格式不同，后者提供更高级的接口，类似于命令行工具与 qpdf 的交互。有关相关信息，请参见 QPDFJob：基于岗位的界面。

qpdf JSON 格式是针对 qpdf 的专用格式。--json 命令行标志会生成 PDF 文件中的对象的 JSON 表示，以及文件中其他信息的 JSON 格式摘要。此功能内置于 QPDFJob 中，可以通过 qpdf 命令行工具或 QPDFJob C 或 C++ API 访问。

从 qpdf JSON 版本 2 开始，qpdf 11.0.0 版本的 JSON 输出包含了 PDF 对象和头部的明确且完整的表示。不含 JSON 格式的其他信息摘要也可通过 QPDF::writeJSON 方法获取。

默认情况数据会从 JSON 数据中省略，但可以通过指定--json-stream-data 选项来包含流数据。包含流数据后，生成的 JSON 文件完全代表 PDF 文件。你可以把这看作是用 JSON 作为表示 PDF 文件的替代语法。利用 qpdf JSON，可以将 PDF 文件转换为 JSON，在底层作对象的结构或内容，并将结果转换回 PDF 文件。此功能可以通过命令行中的 --json-input 和 --update-from-json 标志访问，或通过 API 中的 QPDF::createFromJSON 和 QPDF::updateFromJSON 方法访问。--json-output 标志会修改一些默认值，使最终的 JSON 尽可能接近原始输入，并准备好转换回 PDF。

qpdf JSON 数据包含未引用的对象。未来版本的 qpdf 可能会对此进行解决。目前，这意味着包含了某些在 JSON 表示中无用的对象。这包括线性化和加密词典、线性化提示流、对象流，以及与尾部词典相关的交叉引用（xref）流（如适用）。为了获得最佳的 qpdf JSON 体验，你可以先通过 qpdf 运行文件，以去除加密、线性化和对象流。例如：

```
qpdf --decrypt --object-streams=disable in.pdf out.pdf qpdf --
json-output out.pdf out.json
```

### 10.2 JSON 术语

术语说明：

- 在 JavaScript 和 JSON 中，带有键和值的那个东西通常被称为对象。

- 在 PDF 中，包含键和值的东西通常被称为词典。对象是 PDF 对象，如整数、实数、布尔、空、字符串、数组、字典或流。
- 一些使用 JSON 的语言将对象称为字典、映射或哈希。
- 有时，如果它有固定键，就叫对象;如果有变量键，则叫字典。

这本手册在词典与宾语的使用上并不完全一致，因为有时某个术语在语境中更为清晰。只是要注意阅读说明书时存在的模糊性。我们经常用词典来指代 JSON 对象，因为其与 PDF 术语的一致性，尤其是在指包含信息 PDF 对象的词典时。

## 10.3 qpdf JSON 不是什么

请注意，qpdf JSON 提供了便捷的语法，用于利用 JSON 语法在低层次作 PDF 文件。JSON 语法比原生 PDF 语法更容易作，而且几乎所有常用编程语言都有很好的 JSON 库。用 JSON 处理 PDF 对象可以免除对流长度、交叉引用表以及出现在内容流之外的 Unicode 或二进制字符串的 PDF 专用表示的需求。这并不排除理解 PDF 文件语义结构的需求。使用 qpdf JSON 仍然需要熟悉 PDF 规范。

特别是，qpdf JSON 不提供以下任何功能：

- 文本提取。虽然你可以用 qpdf JSON 语法导航页面内容流和字体结构，但页面内的文本仍然使用 PDF 语法编码，且没有文本提取的辅助。
- 文本重写，文档结构。qpdf JSON 不会为 PDF 文件内容添加任何新信息或见解。如果你的 PDF 文件没有任何结构信息，qpdf JSON 无法帮你解决这些问题。

这就是我们所说的 JSON 为处理 PDF 数据提供了替代语法的含义。语义上，它与原生 PDF 相同。

## 10.4 qpdf JSON 格式

本节介绍 qpdf 如何以 JSON 格式表示 PDF 对象。它还描述了如何使用 qpdf JSON 来创建或修改 PDF 文件。

### 10.4.1 qpdf JSON Object Representation

本节描述了 qpdf JSON 版本 2 中 PDF 对象的表示方式。一个示例出现在 qpdf JSON 示例中。

PDF 对象表示在 qpdf JSON 文件的“qpdf”条目中。“qpdf”条目是一个两元素数组。第一个元素是一个字典，包含文件的头部信息，如 PDF 版本。第二个元素是一个包含 PDF 文件中所有对象的词典。我们称之为对象词典。

第一个元素包含以下密钥：

- “jsonversion”——表示用于写入的 JSON 版本的数字。这永远是 2。
- “pdfversion”——包含 PDF 版本的字符串，如 PDF 头部所示（例如“1.7”、“2.0”）
- pushedinheritedpageresources——一个布尔值，表示库是否将继承资源推送到页面层。某些库调用会导致这种情况，qpdf 需要在读取 JSON 文件时知道是否应该这样做，因为这可能导致某些对象重新编号。当未给出 --update-from-json 时，该字段会被忽略。

- `calledgetallpages`——一个布尔值，表示在写入 JSON 输出之前是否被调用了 `getAllPages`。这种方法会导致页面树修复，可能会导致某些对象重新编号（极少数页面树损坏的情况），因此 qpdf 在读取 JSON 文件时需要知道这些信息。当未给出 `--update-from-json` 时，该字段会被忽略。
- “`maxobjectid`”——表示文件中编号最高对象的对象 ID。此功能旨在帮助软件在文件中添加新对象，因为创建新对象时可以安全地从高于该数字的对象开始。注意，“`maxobjectid`”的值可能高于输入 PDF 中实际出现的最大对象，因为它考虑了原始文件中悬挂的间接对象引用。这样可以防止你无意中创建一个不存在但被引用的对象，避免产生意想不到的副作用。（PDF 规范明确允许悬挂引用，并表示应将其视为空。如果从 PDF 文件中移除对象，就会发生这种情况。）

第二个元素是对象字典。对象字典中的每个键要么是“`trailer`”，要么是形式为“`obj: O G R`”的字符串，其中 O 和 G 分别是对象和生成数，R 是字面字符串 R。这是 PDF 语法中，前置于 `obj:` 的间接宾语引用。该值代表对象本身，是一个 JSON 对象，其结构如下描述。

#### 顶层流对象

流对象以带有单一密钥“流”的 JSON 对象表示。流对象有一个称为“`dict`”的键，其值即流字典中的对象值（如下所述），省略了“`/Length`”键。其他键由 json 流数据的值（`--json-stream-data`，或类型为 `qpdf_json_stream_data_e` 的参数）确定，具体如下：

- 无：未表示流数据；没有指定其他调。
- 内联：流数据以 base64 编码字符串的形式出现，作为“`data`”键的值
- 文件：流数据被写入文件，文件路径存储在“`datafile`”键中。当调用 qpdf 时，相对路径被解释为相对于当前目录。

除“`dict`”、“`data`”和“`datafile`”外，其他键被忽略。这主要是为了未来兼容，以防新版 qpdf 包含额外信息。与原生 PDF 表示方式一样，流数据必须与流字典中指定的过滤器和解码参数保持一致。

#### 顶层非流对象

非流对象以单一键“值”表示为字典。其他密钥则被忽略以备将来兼容。该值的结构详见下文“对象值”。

注意：在使用对象流的文件中，尾部“字典”实际上是一个流，但在 JSON 表示中，“尾部”键的值总是写成字典（像其他非流对象一样带有“`value`”键）。还会有一个流对象，其键是交叉引用流的对象 ID，尽管该流通常未被引用。这使得可以假设“尾部”指向字典，而无需考虑文件是否使用对象流。这也与 C++ API 中 `QPDF::getTrailer` 的行为一致。

#### 对象值

在“价值”或“流”之内。”`dict`“，PDF 对象表示如下：

- 类型为布尔或空的对象被表示为同类型的 JSON 对象。
- 数字对象在 JSON 中以数字形式表示，不考虑精确度。在内部，qpdf 以字符串形式存储数值，因此在读写 JSON 时会保留任意精度的数值。其他 JSON 读写器可能会根据实现方式处理超出范围的数值。
- 名称对象以 JSON 字符串表示，以 / 开头，后面紧跟 PDF 名称的规范形式，所有 PDF 语法都已解析。例如，根据 PDF 规范，规范形式为 `text/plain` 的名称在 JSON 中表示为“`/text/plain`”，在 PDF 中表示为“`/text#2fplain`”。

从 qpdf 11.7.0 开始, “n: /pdf-syntax” 作为替代格式被接受。它可以用于任何名称 (例如 “n: /text#2fplain”), 但当名称包含二进制字符时则是必要的。例如, /one#a0two 必须表示为 “n: /one#a0two”, 因为单个字节 a0 在 JSON 中无效。

- 间接对象引用以 JSON 字符串表示, 看起来像 PDF 间接对象引用, 形式为 “O G R”, O 和 G 分别是对象和生成编号, R 是字面字符串 R。例如, “3 0 R” 表示对象 ID 为 3、生成为 0 的对象。
- PDF 字符串以 JSON 字符串的形式表示, 有两种方式之一:
  - “u: utf8-encoded-string”: 当 PDF 字符串可以明确表示为 Unicode 字符串且不含不可打印字符时, 使用此格式。无论输入字符串编码为 UTF-16、UTF-8 (PDF 2.0 允许的) 还是 PDF 文档编码, 均适用此法。字符串只有在能够编码且不丢失信息的情况下才会以这种方式表示。
  - “b: hex-string”: 此格式用于表示任何无法以 Unicode 字符串表示的二进制字符串值。十六进制字符串必须有偶数个字符, 字符范围从 a 到 f, A 到 F, 或 0 到 9。

QPDF 将空字符串写为 “U:”, 但 “b:” 和 “u:” 都是空字符串的有效表示。

UTF-16 替代对完全支持。编码为 “b: ...” 的二进制字符串是内部 PDF 表示。因此, 以下两者是等价的:

- “u: \ud83e\udd54” – 在 JSON 语法中将 U+1F954 作为代理对表示
  - “b: FEFFD83EDD54” – 将 U+1F954 表示为 PDF 语法中 UTF-16 字符串的字节, 首位 FEFF 表示 UTF-16
  - “b: efbfff09fa594” – 以 PDF 语法 (PDF 2.0 允许) 中, 以 UTF-8 字符串的字节表示 U+1F954, 首字母为 EF、BB、BF 序列 (即 FEFF 的 UTF-8 编码)。
  - 一个 JSON 字符串, 其内容为 u: 后面是 U+1F954 的 UTF-8 表示。这是土豆表情。很遗憾, 我无法在这本手册的 PDF 版本中渲染它。
- PDF 数组表示为如上所述的对象的 JSON 数组
  - PDF 词典被表示为 JSON 对象, 其键是名称的字符串表示, 其值是 PDF 对象的表示。

注意, JSON 输出是用 QPDF 完成的, 不是 QPDFWriter。因此, QPDFWriter 所做的任何内容都不适用。这包括对流的重新压缩、对象重新编号、移除未引用的对象、加密、解密、线性化、QDF 模式等。更深入的讨论请参见 “编写 PDF 文件”。这有几个值得注意的含义:

- 解密由 qpdf 透明处理。由于库内部也没有 qpdf API, 允许原始加密数据检索, qpdf JSON 始终包含解密数据。未来 qpdf 版本可能会允许访问原始加密字符串和流数据。
- 与 PDF 文件结构相关的对象 (而非内容) 会包含在 JSON 输出中, 尽管它们并不特别有用。在未来的 qpdf 版本中, 这个问题可能会被修复, 并且可以使用 --preserve-unreferenced 标志来获得现有的行为。目前, 为了避免这种情况, 可以将文件通过 qpdf --decrypt --object-streams=disable in.pdf out.pdf 生成一个不包含无引用或结构性对象的新 PDF 文件。
  - 线性化 PDF 文件包含一个线性化词典, 不引用其他对象, 并通过偏移量引用线性化提示流。线性化 PDF 文件中的 JSON 包含这两个对象, 尽管它们在 JSON 中无用。偏移信息不在 JSON 中表示, 所以无法从 JSON 中找到线性化提示流。如果用写入的 JSON 创建了新的 PDF, 这些对象会被读回, 但只是未被引用的对象, 在文件重写时 QPDFWriter 会忽略这些内容。

- 来自带有对象流的文件中的 JSON 将包含原始对象流，同时也将流中的所有对象作为顶层对象。
- 在带有对象流的文件中，尾部“字典”是一个流。在 qpdf JSON 文件中，“尾部”键包含一个字典，包含与流相关的所有键，流也会显示为未被引用的对象。
- 加密文件已解密，但加密字典仍显示在 JSON 输出中。

## 10.4.2 qpdf JSON 示例

下面的 JSON 展示了一个以 qpdf JSON 格式表示的简单 PDF 文件示例。

```
{
  "QPDF" : [
    {
      "jsonversion" : 2, "pdfversion" :
      "1.3", "pushedinheritedpageresources" :
      false, "calledgetallpages" : false,
      "maxobjectid" : 6 }, { "obj: 1 0 R" : {
        "value" : {
          "/页数" : "3 0 R",
          "/类型" : "/目录"
        }
      },
      "obj: 2 0 R" : {
        "value" : {
          "/作者" : "u: π 的数字", "/CreationDate" : "u: D:
          20220731155308-05' 00' ", "/Creator" : "u: 一个用 Emacs
          输入的人", "/关键词" : "u: potato, 示
          例", "/ModDate" : "u: D: 20220731155308-
          05' 00' ", "/Producer" : "u: qpdf", "/Subject" : "u:
          Example", "/Title" : "u: 土豆相关的东西" } }, "obj: 3
          0 R" : {
            "value" : {
              "/伯爵" : 1,
              "/孩子们" : [
                "4 0 R"
              ],
              "/类型" : "/页数"
            }
          },
          "obj: 4 0 R" : {
            "value" : {
              "/Contents" : "5 0 R",
              "/MediaBox" : [
                0,
```

(续见下一页)

(续自上一页)

```

    0,
    612,
    792
  ],
  “/父”: “3 0 R”,
  “/资源”: {
    “/字体”: {
      “/F1”: “6 0 R”
    }
  },
  “/类型”: “/页面”
},
“obj: 5 0 R”: {
  “流”: {
    ‘data’: ‘eJxzCuFSUNB3M1QwM1EISQ0yzY2AyEAhJAXIlgjILOksyddUCMnicg3hAgDLAQnI’,
    “DICT”: {
      “/Filter”: “/FlateDecode” } }
}, “obj: 6 0 r”: {
  “价值”: {
    “/BaseFont”: “/Helvetica”,
    “/Encoding”: “/WinAnsiEncoding”,
    “/Subtype”: “/Type1”, “/Type”:
    “/Font” } }, “trailer”: {
    “价值”: {
      “/ID”: [
        “b: 98b5a26966fba4d3a769b715b2558da6”,
        “b: 6bea23330e0b9ff0ddb47b6757fb002e” ],
      “/Info”: “2 0 R”, “/Root”: “1 0
      R”, “/Size”: 7 }
    }
  }
}

```

### 10.4.3 qpdf JSON Input

qpdf JSON 输出有两种不同用途:

- 通过使用 `--json-input` 标志或调用 `QPDF::createFromJSON` 代替 `QPDF::processFile`, 可以用 qpdf JSON 文件代替 PDF 文件作为 qpdf 的输入。
- 通过使用 `--update-from-json` 标志或在已初始化的 QPDF 对象上调用 `QPDF::updateFromJSON` 文件, 可以使用 qpdf JSON 文件对现有的 QPDF 对象进行更改。那个 QPDF 对象可能已经来了

来自任何来源，包括 PDF 文件、QPDF JSON 文件，或任何其他过程的结果，生成有效且初始化的 QPDF 对象。

以下是关于 qpdf JSON 输入的一些重要信息。

- 当 qpdf JSON 文件作为主要输入文件时，必须是完整的。这意味着
  - 必须在第一个数组元素中用 “jsonversion” 键指定 JSON 版本号
  - PDF 版本号必须在第一个数组元素中用 “pdfversion” 键指定
  - 所有流必须有流数据
  - 必须包含尾部字典，但只需 “/根” 键。
- 无论是创建还是从 JSON 文件更新，输入中的某些字段都会被忽略：
  - “maxobjectid” 被忽略，因此添加新对象时无需更新。
  - “/长度” 在所有流式词典中都被忽略。qpdf 在生成 JSON 输出时不会放进去，也不需要添加它。
  - 如果 “/Size” 出现在拖尾器词典中，则忽略，因为该词典总是由 QPDFWriter 重新计算。
  - 文件顶层、“qpdf” 内、每个 PDF 对象顶层（包含 “value” 或 “stream” 键的词典内）和直接 “stream” 中的未知键均被忽略，以保证未来兼容性。这包括由 qpdf 本身生成的其他顶层键（如 “页面”）。因此，如果要修改 JSON 文件以转换回 PDF，这些键不必与 “qpdf” 键一致。如果你想存储应用特定的元数据，可以通过添加一个以 x- 开头的键来实现。qpdf 保证不会添加任何以 x- 开头的键。注意，顶层的任何 “版本” 键都被忽略。JSON 版本来自 “qpdf” 字段第一个元素中的 “jsonversion” 键。
- 创建文件时忽略了 “calldgetallpages” 和 “pushedinheritedpageresources” 这两个值。在更新文件时，如果遗漏了这些文件，会被视为虚假。
- 当 qpdf 读取 PDF 文件时，内部对象编号始终被保留。然而，当 qpdf 使用 QPDFWriter 写入文件时，QPDFWriter 会自行编号，通常不会保留输入对象号。这意味着用于更新现有 PDF 的 qpdf JSON 文件必须具有与其修改的输入文件匹配的对象编号。实际上，这意味着你不能用一个 PDF 文件创建的 JSON 文件来修改该文件上运行的 qpdf 输出。

更具体地说，以下是有效的：

```
qpdf --json-output in.pdf pdf.json # 编辑 pdf.json qpdf
in.pdf out.pdf --update-from-json=pdf.json
```

相比之下，以下结果将产生不可预测且可能错误的结果，因为 out.pdf 的对象数与 pdf.json 和 in.pdf 不同。

```
qpdf --json-output in.pdf pdf.json # 编辑 pdf.json qpdf
in.pdf out.pdf --update-from-json=pdf.json # 再次编辑
pdf.json # 不要这样做 qpdf out.pdf out2.pdf --update-from-
json=pdf.json
```

- 当从 JSON 文件 (--update-from-json, QPDF::updateFromJSON) 更新时，现有对象会在原地更新。这有以下含义：

- 如果你更新的对象是流，可以省略“data”和“datafile”。在这种情况下，原始流数据会被保留。你必须始终提供流式词典，但可能词典是空的。注意，空流字典会清除旧字典。没有办法表明旧的流式词典应保持原样，所以如果你的意图是替换流数据并保留词典，原始词典必须出现在 JSON 文件中。
- 你可以将一个对象类型更改为另一个对象类型，包括用非流替换流或用流替换非流。如果你用流替换非流，你必须为流提供数据。
- 你不想修改的对象可以从 JSON 中省略。这也包括预告片。这意味着你可以用 --json-object 编写的 qpdf JSON 文件输出，只包含你打算修改的对象。

——你可以省略“pdfversion”键。输入的 PDF 版本将被保留。

#### 10.4.4 qpdf JSON Workflow: CLI

本节包含一些使用 qpdf JSON 的示例。

- 将 PDF 文件转换为 JSON 格式，编辑 JSON，再转换回 PDF。这是在文本编辑器中修改 PDF 文件时，使用 QDF 模式（参见 QDF 模式）的一种替代方案。每种方法都有其优缺点。

```
qpdf --json-output in.pdf pdf.json # 编辑 pdf.json
qpdf --json-input pdf.json out.pdf
```

- 只需将特定对象解压到 JSON 文件中，修改该对象的 JSON，并用修改后的对象更新原始 PDF。在这里，我们编辑的是对象 4，不管那是什么。你还得通过其他方式知道你想编辑哪个对象，比如查看其他 JSON 输出，或者用工具（可能但不一定是 qpdf）来识别对象。

```
QPDF --json-output in.pdf pdf.json --json-object=4,0 # 编辑 pdf.json
qpdf in.pdf --update-from-json=pdf.json out.pdf
```

与其像上面示例那样使用 --json-object，不如编辑 JSON 文件，删除不需要的对象。你也可以直接留在那里，不过更新过程会慢一些。

你也可以通过把新对象添加到 pdf.json 文件中。只要确保对象编号不会和现有对象冲突就行。原始输出中的“maxobjectid”字段可以帮助实现这一点。如果你添加了对象，不需要更新，因为当文件重新读取时，这个功能会被忽略。

- 将 --json-input 和 --json-output 一起使用来演示如何保持对象号。在这个例子中，a.json 和 b.json 的对象和对象编号是相同的。文件可能并不完全相同，因为字符串可能经过规范化，字段出现顺序不同等等。不过 b.json 和 c.json 很可能是相同的。

```
QPDF --json-输出 in.pdf a.json qpdf --json-input --json-output a.json b.json
qpdf --json-input --json-output b.json c.json
```

#### 10.4.5 qpdf JSON Workflow: API

所有用 qpdf CLI 完成的作，都可以用 C++ API 完成。详情请参见 QPDF.hh 中的注释，关于 writeJSON、createFromJSON 和 updateFromJSON。



## 10.5 JSON 兼容性保证

qpdf JSON 表示包括 PDF 文件中原始对象的 JSON 序列化，以及一些更易提取的计算信息。qpdf 对其 JSON 格式提供一定保证。这些保证旨在简化开发者使用 JSON 格式的体验。

### 兼容性

顶层的 JSON 对象是一个字典（“JSON”对象）。JSON 输出包含各种嵌套字典和数组。除了由文件中 PDF 对象字段填充的词典外，所有字典实例都保证拥有完全相同的键。

顶层 JSON 结构包含一个“版本”键，其值为简单整数。如果发生不兼容的更改，版本密钥的值会增加。不兼容的更改是指涉及移除密钥、改变密钥指向的数据格式，或需要对已有密钥进行不同解释的语义变化。注意，从版本 2 开始，JSON 版本也出现在“qpdf”字段的第一个元素的“jsonversion”字段中。

在特定的 qpdf JSON 版本中，未来的 qpdf 版本可以自由添加额外键，但不能移除键或更改键指向的对象类型。这意味着 qpdf JSON 的使用者应忽略他们不了解的密钥。

### 文献资料

qpdf 命令可以通过 `--json-help` 选项调用。这将输出一个 JSON 结构，其结构与 qpdf 生成的 JSON 输出相同，只是帮助输出中的每个字段都是对应字段的描述。具体保证如下：

- 帮助输出中的字典意味着实际 JSON 输出中对应的位置也是具有完全相同键的词典;也就是说，帮助中没有出现的键在真实输出中缺失，而在真实输出中也不会有不在帮助中的键。有可能某个键存在且其值显式为 `空`。作为一个特殊情况，如果字典有一个以 `<` 开头、以 `>` 结尾的单一键，则表示 JSON 输出是一个可以有任意值键的字典。这适用于字典键是对象 ID 等物品的情况。
- 帮助输出中的字符串是对实际输出对应位置的项目描述。对应的输出可以是包括空值在内的任意值。
- 帮助输出中的单元素数组表示实际输出中对应的位置要么是单个项目，要么是任意长度的数组。数组中的单个项或每个元素的格式与帮助输出数组的单个元素所隐含的相同。
- 帮助输出中的多元数组表示实际输出中对应的位置是相同长度的数组。输出数组的每个元素都具有对应帮助输出数组中对应元素所暗示的格式。

例如，帮助输出显示包含一个“pagelabels”键，其值是一个元素的数组。这个元素是一个带有“index”和“label”键的词典。除了描述这些键的含义外，这还告诉你实际的 JSON 输出会包含一个 pagelabels 数组，每个元素是一个包含索引键、标签键且没有其他键的字典。

### 直接与简洁

JSON 输出包含文件中每个对象的值，但也包含一些摘要数据。这类似于 qpdf 库界面的工作原理。摘要数据类似于辅助功能，允许你查看 PDF 文件的某些方面，而无需理解 PDF 规范的所有细节，而原始对象则允许你挖掘 PDF 中高层接口所缺乏的内容。创建带有“pages”和“qpdf”键的 JSON 文件，并利用“pages”信息来查找页面，而不是手动浏览页面树，这尤其有用。这可以安全完成，并且可以对对象字典进行更改，无需担心“页面”的更新，因为读取文件时页面会被忽略。

## 10.6 JSON: 特殊考虑

大多数情况下, 内置的 JSON 帮助会告诉你关于 JSON 格式的所有信息, 但也有一些不那么明显的需要注意的地方:

- 如果 PDF 文件的页面树存在某些类型的错误 (例如页面对象是直接的, 或多个页面共享同一对象 ID), qpdf 会自动修复页面树。如果你在没有其他键的情况下指定 “qpdf” (或者用 qpdf JSON 1 的 “objects” 或 “objectinfo”), 你会看到原始页面树, 没有任何修正。如果你指定了需要页面树遍历的键 (例如 “pages”、“outline” 或 “pagelabel”), 那么 “qpdf” (以及 “objects” 和 “objectinfo”) 会显示修复后的页面树, 使整个文件的对象引用保持一致。你可以通过查看 “qpdf” 数组第一个元素中的 “calledgetallpages” 和 “pushedinheritedpageresources” 字段来判断是否发生了这种情况。
- 虽然 qpdf 保证帮助中存在的键也会出现在输出中, 但如果文件中信息未知或缺失, 这些字段可能是空字段或空的。另外, 如果你指定了 --json-key, 那些未被列出的键将完全排除, 除了 --json-help 说总是存在的那些。
- 在少数地方, 有些键的名称包含 pageposfrom1。这些键的值为空或整数。如果是整数, 则指向文件内的页面索引, 编号从 1 开始。注意, JSON 索引是从 0 开始的, 你也会用 API 进行基于 0 的索引。然而, 基于 1 的索引在此情况下更为简单, 因为命令行中指定页面范围的语法是基于 1 的。如果你要写一个程序, 通过 JSON 查找特定页面的信息, 然后用命令行提取这些页面, 基于 1 的索引会更容易。而且, 在真实编程语言中减去 1 比在 shell 代码中加 1 方便得多。
- JSON 输出页面部分中包含的图像信息包含关键字 “可过滤”。注意, 该字段的值可能取决于你调用 qpdf 时的 --decode-level。JSON 输出包含顶层密钥 “参数”, 指示用于计算流是否可过滤的译码级别。例如, jpeg 图片默认显示为不可过滤, 但运行 qpdf --json --decode-level=all 时, 它们会显示为可过滤。
- 加密密钥的值将被填充为未加密文件。有些值会是空的, 有些则适用于未加密的文件。
- qpdf 库本身从不加载整个 PDF 到内存中。对于以 JSON 格式表示的 PDF 文件, 这一点依然成立。一般来说, qpdf 在文件完全读取后会将整个对象结构存储在内存中 (对象加载得很懒惰, 但加载后会保持在内存中), 但它一次内存中流的副本永远不会超过两个。不过, 如果你让 qpdf 写入 JSON 到内存, 它会做到, 所以如果你处理的是非常大的 PDF 文件, 要小心这一点。qpdf 库本身没有任何内容阻止处理远大于可用系统内存的 PDF 文件。qpdf 既可以读取也能写入 JSON 格式的此类文件。如果你需要在内存中使用 PDF 文件的 json 表示, 建议用 none 或 file 作为 --json-stream-data 的参数, 或者如果使用 API, 可以用 qpdf\_sj\_none 或 pdf\_sj\_file 作为 json 流数据值。如果没有, 你可以用其他方式获取流数据。

## 10.7 从 JSON v1 到 v2 的变更

qpdf 的 JSON 输出格式在第二版中进行了以下更改。

- 物体的表现方式发生了变化。详情请参见 qpdf JSON 对象表示。
  - 字符串的表示现在对所有字符串都变得无歧义。字符串 a 前缀为 u: 表示 Unicode 字符串, b: 表示字节字符串。
  - 名称以 qpdf 的规范形式显示, 而非 PDF 语法。(例如: PDF-语法名称 /text#2fplain 在 v1 中显示为 “/text#2fplain”, 但在 v2 中显示为 “/text/plain”。在 qpdf 11.7.0 中, 修复了包含二进制字符的名称, 允许使用 “n: /pdf-syntax”。

——“对象”中的顶层表示对象是一个包含“值”键或“流”键的词典，使得流与其他对象的区分成为可能。

- “objectinfo”和“objects”键已被移除，取而代之的是包含头信息并区分流与其他类型对象的“qpdf”表示。在 v1 中，无法区分“对象”中的字典流，PDF 版本也完全无法捕获。
- 在对象字典中，键现在是“obj: O G R”，O 和 G 分别是对象和生成编号。“拖车”仍然是拖车词典的关键词。在 v1 中，obj: 前缀没有出现。这一变更的理由如下：
  - 拥有唯一的前缀（obj:）使得在 JSON 文件中搜索对象定义变得更加容易
  - 键中仍包含 O G R 使从间接引用构建键更容易。
 你只需要在 obj 前面加一个。无需解析间接对象引用。
- 在“encrypt”对象中，v1 中“modifyannotations”被误拼成“moddifyannotations”。此点已更正。

### 10.7.1 qpdf JSON version 2 的动机

qpdf JSON 2 版本旨在使 PDF 文件能够使用 JSON 语法而非原生 PDF 语法来作。这使得几乎任何编程语言的 PDF 文件都能进行底层更新，甚至可以用 jq 或任何能处理 JSON 文件的编辑器，在命令行中进行更新。JSON 格式 1 存在若干限制，使得这变得不可能：

- 原始 PDF 文件中的字符串、名称和间接对象引用均被转换为 JSON 表示中的字符串。对于普通的人类观察来说，这没问题，但在一般情况下，无法区分看起来像名字或间接对象引用的字符串与真实名称或间接对象引用的区别。
- PDF 字符串在 JSON 格式中并不明确表示。qpdf JSON v1 表示字符串的方式是尝试将字符串转换为 UTF-8。这是通过假设未明确标记为 Unicode 的字符串被编码为 PDF 文档编码来实现的。问题在于 Unicode 和 PDF 文档编码之间没有完美的双向映射，所以如果某个二进制字符串包含无法双向映射的字符，就无法返回原始 PDF 字符串。即使可能，试图从二进制字符串的 JSON 表示映射回原始字符串，也需要了解 PDF 文档编码与 Unicode 之间的映射。
- 没有流数据的表示。如果你想提取流数据，可以用 --show-object，所以这对检查来说不那么重要，但它阻碍了从 JSON 回到 PDF 的转换。qpdf JSON 2 版本允许将流数据作为 base64 编码的数据内嵌包含。还有一个选项可以将所有流数据写入外部文件，这使得即使使用试图将整个 JSON 结构读取到内存的工具，也能处理非常大的 JSON 格式 PDF 文件。
- PDF 头的 PDF 版本未在 qpdf JSON v1 中表示。



## 为 QPDF 贡献

### 11.1 源代码仓库

qpdf 源代码保存在 <https://github.com/qpdf/qpdf>。

在 <https://github.com/qpdf/qpdf/issues> 创建问题（错误报告、功能请求）。如果您有一般性的问题或讨论话题，可以在 <https://github.com/qpdf/qpdf/discussions> 创建讨论。

### 11.2 代码格式化

qpdf 源代码采用 clang-format 格式，源代码树顶部有 .clang-格式文件。format-code 脚本会重新格式化仓库中的所有源代码。你的路径中必须有 clang-format，且至少是版本 20。

对于 emacs 用户，.dir-locals.el 文件配置 emacs cc 模式，缩进风格与 clang-format 类似但不完全相同。当存在差异时，clang-格式具有权威性。由于 emacs 中的语法解析器不如 clang-格式复杂，无法让 cc-mode 和 clang-格式完全匹配。

不应格式化的代码块可以用注释 `// clang-format off` 和 `// clang-format on` 包围。有时 clang 格式会尝试以不利的方式组合行数。在这种情况下，我们遵循在自己行加上//换行的惯例。

具体细节请参阅.clang-format。以下是格式规则的大致部分总结：

- 用空格，不要用制表符。
- 尽可能保持行数在 100 列以内。
- 大括号在类和函数（以及类似的顶层构造）之后独立排列，其他情况下是紧致的。
- 括号是附加在前一材料上，而不是单独的行。

README 维护文件中有一些额外的注释，但对于不对 qpdf 进行深度修改的人来说可能并不重要。

### 11.3 自动化测试

qpdf 的测试风格随着时间演变。较新的测试称为 assert ()。老测试是按标准输出打印内容，然后与参考文件进行比较。许多测试都是这些技术的混合。

qtest 的测试风格是通过应用程序测试所有内容。所以实际上，大多数测试都是“集成测试”或者“端到端测试”。

有关 qtest 的详细信息，请参阅 QTest 手册。阅读时请记住，尽管文件日期较新，但绝大多数文档来自 2007 年之前，且早于许多当今使用的测试框架和方法。

测试说明：

- 大多数情况下，代码中的某些部分通过集成测试进行测试，尽管测试套件非常全面。许多测试通过 qpdf CLI 进行。其他文件则通过 qpdf 目录中的其他文件驱动，尤其是 test\_driver.cc 和 qpdf-ctest.c。这些程序只使用公共 API。
- 在某些情况下，确实存在真正的“单元测试”，但它们是通过各种独立程序进行的，这些程序以特定方式锻炼图书馆，包括一些能够访问图书馆内部结构的程序。这些都在 libtests 目录里。

### 11.3.1 覆盖范围

你会在代码中看到调用 QTC:: TC。这是一个“手动覆盖”系统，qtest 文档中有详细描述。它的工作原理是确保在测试过程中以每种配置方式调用 QTC:: TC。简要说明：

- QTC:: TC 有两个强制选项和一个可选选项：
  - 前两个参数必须是字符串面值。这是因为 qtest 通过词汇方式查找覆盖案例。
  - 第一个参数是范围名称，通常为 qpdf。这意味着源目录中有一个 qpdf.testcov 文件。
  - 第二个论点是一个案例名称。每个案例名称都会出现在 qpdf.testcov 中，后面通常有数字，通常是 0。
  - 如果第三个参数存在，则是一个数字。qtest 确保 QTC:: TC 至少被调用一次，且第三个参数设置为 0 到 n 的每个值，n 是覆盖调用后的数字。
- QTC:: TC 只有设置某些环境变量才会做任何事。因此，QTC: TC 调用不应有副作用。（在某些语言中，编译时可能会关闭这些功能，尽管 qpdf 实际上并不这样做。）

举个例子，如果你有这样的代码：

```
QTC: : TC ( “qpdf” , “跳过 xref 前的空格的 QPDF e” ,
           skipped_space? 0 : 1 );
```

以及 qpdf.testcov 中的这行：

```
QPDF eof 跳过 xref 1 之前的空格
```

测试套件只有在该代码行至少被调用一次 skipped\_space == 0 和至少一次 skipped\_space == 1 时才会通过。

手动覆盖方法确保读者在测试中已涵盖某些条件。使用 QTC:: TC 只是整体策略的一部分。

我不要求对拉取请求进行测试，但很感激测试，而且我不会合并任何未测试的代码。通常有人会提交一个没有充分测试但贡献良好的拉取请求。在这种情况下，我通常会拿代码，通过测试添加，然后接受修改，而不是按提交合并拉取请求。

## 11.4 个人评论

QPDF 始于 2002 年作为一个工作项目。首个开源版本于 2008 年发布。虽然有少数贡献者，但绝大多数代码是由一个人多年作为副项目编写的。

我对向后兼容保持非常坚定的承诺。因此，代码中有许多方面显得有些陈旧。虽然我认为代码库质量很高，但如果我现在从零开始做，有些事情我会做得不同。有时有人会建议一些我喜欢但因兼容性问题无法接受的改动。

虽然我欢迎贡献，也渴望与贡献者合作，但我的标准很高。我只接受那些我愿意长期维护的内容，并且乐于帮助人们将作品提交到那个状态。





## 设计与图书馆注释

### 12.1 介绍

本节是在 qpdf 库实现之前编写的，随后经过修改以反映实现情况。在某些情况下，为了解释，它可能与实际实现略有不同。一如既往，源代码和测试套件都是权威的。即使存在一些错误，这份文件也应作为理解该代码工作原理的路线图。

一般来说，写作时应严格遵守规范，但在阅读时应保持宽松。这样，我们的软件产品将被最广泛的其他程序接受，我们也能接受最广泛的输入文件。该库尽可能遵循这一理念，同时也旨在为想要验证 PDF 文件的人提供严格的检查。如果你不想看到警告，并且想写一些宽容的内容，可以调用 `setSuppressWarnings (true)`。如果你想在第一个错误中失败，可以调用 `setTryRecovery (false)`。默认行为是生成可恢复问题的警告。请注意，即使恢复能够通过文件，也不一定能达到预期效果。与大多数其他 PDF 文件会发出诸如“此文件已损坏”等通用警告不同，qpdf 通常会发出对 PDF 开发者最有用的详细错误信息。这是有意为之，因为目前似乎缺少 PDF 验证工具。事实上，这也是最初创建 qpdf 的主要动机之一。不过，qpdf 并不是严格的 PDF 检查器。PDF 文件可能因多种方式不符合规范，而 qpdf 不会注意到或报告这些情况。

### 12.2 检查模式

上述方法出于务实原因，随着时间推移有所变化。许多本可以安全完成的重要维修工作，即使使用 `setAttemptRecovery (false)` 也在进行。同时，为避免产生干扰噪音，对 PDF 标准的若干轻微违规行为也会被悄悄处理。这对使用 qpdf 进行内容保留转换的用户非常有帮助，他们通常希望 qpdf 能可靠工作，并生成符合 PDF 标准的正确 PDF 输出文件，即使输入文件不够完美。

然而，qpdf 还有另一个明确的目的——提供一个用于研究和分析 PDF 文件的工具。以这种方式使用时，修复输入文件中的错误或防止生成不可用的输出文件（通常伴随不可接受的资源消耗）反而适得其反，而非实用。

为了满足使用 qpdf 作为检查和调查 PDF 文件工具的用户的需求，qpdf 12.3 版本引入了一种特殊的检查模式，该模式通过 `qpdf::global::options::inspection_mode` 功能启用。在检查模式下，只支持非常有限的基本作，并且禁用了部分自动修复功能。不支持输入文件的转换，如线性化文件、创建对象流或加密文件，文档和对象辅助工具的使用也同样支持。

检查模式用于人工检查和维修。因此，稳定性比 qpdf 的其他用途更不重要。具体检测模式的具体效果会随着版本而变化。

## 12.3 设计目标

qpdf 库支持读取和重写 PDF 文件。它旨在隐藏涉及对象位置、修改（附加）PDF 文件、对象流使用以及流过滤（包括加密）的详细信息。它并不旨在隐藏对对象层级或内容流内容的了解。换句话说，qpdf 库的用户应了解 PDF 文件的工作原理，但不必跟踪文件位置等簿记细节。

访问对象时，库用户无需在意对象是直接还是间接，因为所有访问对象的访问都是透明处理的。所有内存管理细节也由库处理。在修改对象时，可以判断对象是否间接，并在需要时复制该对象。

内存主要通过 `std::shared_ptr` 对象管理，以尽量减少显式内存处理。该库还利用一种技术，通过使用带有朋友的公有子类，仅使用私有成员，而私有成员调用包含类的私有方法，从而为其他类提供细粒度访问。参见 `QPDFObjectHandle::Factory` 作为示例。

最高级别的 QPDF 课程是 QPDF。QPDF 对象表示一个 PDF 文件。该库提供访问和变更 PDF 文件的方法。

与 PDF 对象交互的主要类是 `QPDFObjectHandle`。该类实例可以通过值传递、复制、存储在容器中等，开销非常低。`QPDFObjectHandle` 对象包含指向底层对象的内部共享指针。通过读取文件创建的 `QPDFObjectHandle` 实例总会包含回溯其创建的 QPDF 对象的引用。`QPDFObjectHandle` 可以是直接的，也可以是间接的。如果是间接的，则对象最初未解决。在这种情况下，首次尝试访问底层对象时，会通过调用被引用的 QPDF 实例来解析该对象。这使得基于直接对象和间接对象，导致某些内容对某些 PDF 文件有效而对其他文件无效的编码错误几乎不可能。在需要判断对象是否间接的情况下，这些信息可以从 `QPDFObjectHandle` 获得。也可以将直接宾语转换为间接宾语，反之亦然。

`QPDFObjectHandle` 实例可以通过 `QPDFObjectHandle` 类中的静态工厂方法直接创建和修改。每种对象都有工厂方法，还有一个方便方法 `QPDFObjectHandle::parse`，可以从对象的字符串表示创建对象。`_qpdf` 用户自定义字符串文字也可用，使得创建带有 “(pdf-syntax)” `_qpdf` 的 `QPDFObjectHandle` 实例成为可能。现有的 `QPDFObjectHandle` 实例也可以通过多种方式进行修改。详情请参见 `QPDFObjectHandle.hh` 中的评论。

QPDF 实例通过使用类的默认构造函数或使用 `QPDF::create()` 构建。如有需要，QPDF 对象可配置多种方法以改变其默认行为。然后 `QPDF::processFile` 方法会传递一个 PDF 文件的名称，从而将该文件永久关联到该 QPDF 对象。访问受密码保护的文件时，也可以设置密码。QPDF 不强制加密参数，用户和所有者密码会同等对待。任一密码都可以用于访问加密文件。QPDF 允许恢复用户密码，前提是拥有者密码。输入的 PDF 文件必须是可寻址的。`QPDFWriter` 编写的输出文件不必是可寻址的，即使创建线性化文件。在构建过程中，QPDF 验证 PDF 文件的头部，然后读取交叉引用表和尾部词典。QPDF 类只保留第一个预告片词典，但它会阅读所有预告片词典，以便检查/Prev 键。QPDF 类用户可以特别请求根对象和尾部词典。交叉参考表保持私密。随后可以通过数字或遍历对象树来请求对象。

当 PDF 文件采用交叉引用流而非交叉引用表和尾部时，请求文档尾部词典时，会返回交叉引用流中的流字典。

对于非常常见的作，比如走动页面树和返回所有页面对象的向量，有一些方便例程。详情请参见头文件 `QPDF.hh` 和 `QPDFObjectHandle.hh`。还有一些额外的辅助类，为某些文档结构提供更高级的 API 函数。这些内容在辅助职业中被讨论。

## 12.4 辅助职业

QPDF 8.1 版本引入了辅助类的概念。辅助类旨在包含更高级的 API，允许开发者以高于 QPDFObjectHandle 的抽象层次处理某些文档结构，同时忠实于 qpdf 不对开发者隐藏文档结构的理念。和 qpdf 一般一样，目标是省去处理 PDF 文件时较为繁琐的簿记部分，而不是消除开发者理解相关 PDF 结构的必要性。创建辅助类的推动力是允许在不污染主要顶层类 QPDF 和 QPDFObjectHandle 接口的情况下，在不影响 qpdf 中更高层接口的演进。

帮助类有两种：文档辅助类和对象辅助类。文档辅助工具基于 QPDF 对象的引用构建，提供处理文档层结构的方法。对象辅助器是用 QPDFObjectHandle 实例构建的，提供处理特定类型对象的方法。

文档辅助器的例子包括 QPDFPageDocumentHelper，它包含作文档页面树的方法，如枚举文档的所有页面以及添加和删除页面；以及 QPDFAcroFormDocumentHelper，包含与交互式表单相关的文档级方法，如枚举表单字段和创建表单字段与注释之间的映射。

对象辅助器的例子包括用于页面作（如页面旋转和部分内容流作）、QPDFFormFieldObjectHelper（用于交互式表单字段作）以及 QPDFAnnotationObjectHelper（用于注释处理）。

总可以从文档助手中检索底层的 QPDF 引用，也可以从对象助手中检索底层的 QPDFObjectHandle 引用。帮手设计成帮手，而不是包装。其意图是，通常可以自由混合使用辅助工具的作和使用底层对象的作。文档和对象助手不试图为所帮助的事物提供完整的接口，也不试图封装底层结构。他们只是提供一些方法来帮助处理容易出错、重复或复杂的任务。在某些情况下，辅助对象可能会缓存一些收集成本高昂的信息。在这种情况下，辅助类的实现使其自身的方法保持缓存一致性，头文件会提供使缓存失效的方法以及描述哪些作会使缓存失效。如果有疑问，你总可以丢弃一个辅助类，创建一个包含相同底层对象的新类，这样可以确保你已经丢弃了任何过时的信息。

按照惯例，文档助手称为 QPDFSomethingDocumentHelper，源自 QPDFDocumentHelper；对象助手称为 QPDFSomethingObjectHelper，源自 QPDFObjectHelper。有关具体辅助器的详细信息，请参阅它们的头文件。你可以通过查看 include/qpdf/QPDF\*DocumentHelper.hh 和 include/qpdf/QPDF\*ObjectHelper.hh 来找到它们。

为避免产生循环依赖，辅助类遵循以下一般指导原则：

- 核心类接口不了解辅助类。例如，QPDF 或 QPDFObjectHandle 的任何方法都不会在接口中包含辅助类。
- 对象助手的界面通常不会在接口中使用文档助手。这是因为文档助手拥有返回对象助手的方法更为有用。大多数 PDF 文件中的作是从文档层面开始，然后从那里到对象层面，而不是反过来。有时将对象级结构映射回文档级结构是有用的。如果需要这样做，通常会由文档辅助类中的某个方法提供。
- 大多数情况下，对象辅助器并不知道其他物体辅助器。然而，在某些情况下，一种对象类型可能是另一种对象的容器，在这种情况下，外在对象知道内对象是合理的。例如，QPDFPageObjectHelper 中有些方法知道 QPDFAnnotationObjectHelper，因为页面词典中包含了对注释的引用。
- 任何辅助工具或核心库类都可以在其实现中使用辅助工具。

在 qpdf 8.1 版本之前，更高级的接口作为“便利功能”被添加到 QPDF 或

QPDF 标题。为了兼容性，旧的方便页面作函数仍会保留在这些类中，尽管辅助类中提供了替代方案。未来，将通过辅助类提供新的高级接口。

## 12.5 实现说明

本节包含关于 qpdf 内部实现的一些说明，特别是它在首次处理文件时所做的事情。本节对其实际作用略为简化，但可以作为理解实现者的起点。本节没有你需要知道的内容才能使用 qpdf 库。

在 PDF 文件中，对象可以是直接的或间接的。直接对象是指其表示直接出现在 PDF 语法中的对象。间接对象是通过 ID 对对象的引用。qpdf 库使用 QPDFObjectHandle 类型来保留对象，并在大多数情况下抽象对象是直接还是间接。

在内部，QPDFObjectHandle 保留了指向底层对象值的共享指针。当直接对象由客户端代码程序生成（而非从文件读取）时，持有该对象的 QPDFObjectHandle 不会与 QPDF 对象关联。当创建间接对象引用时，它起始处于未解析状态，必须关联到 QPDF 对象，QPDF 对象被视为其所有者。要访问对象的实际值，必须先解析该对象。当该对象以任何方式被访问时，这会自动发生。

为了解析一个对象，qpdf 会检查其对象缓存。如果缓存中找不到，它会尝试从与 QPDF 对象关联的输入源读取该对象。如果找不到，则返回一个空对象。空对象是一种对象类型，就像布尔值、字符串、数字等一样。它不是空指针。PDF 规范规定，间接引用不存在的对象应视为空。最终生成的对象，无论是空对象还是实际读取的对象，都会存储在缓存中。如果该对象后来被替换或交换，底层对象保持不变，但其值会被替换。这样，如果你有一个 QPDFObjectHandle 指向一个间接对象，并且通过调用 QPDF::replaceObject 或 QPDF::swapObjects 替换了该数字的对象，你的 QPDFObjectHandle 就会反映该对象的新值。这与如果你替换文件中对象定义时，PDF 对象会发生的情况是一致的。

当从输入源读取对象时，如果请求的对象位于对象流中，首先会读取该对象流本身到内存中。然后分词器根据存储在流中的偏移信息从内存流中读取对象。这些单个对象被缓存，之后存放对象流内容的临时缓冲区被丢弃。通过这种方式，当第一次请求对象流中的对象时，流中的所有对象都会被缓存。

以下示例应说明 qpdf 如何处理一个简单文件。

- 客户端构建 QPDF pdf，并调用 pdf.processFile (“a.pdf”) ;。
- QPDF 类检查 a.pdf 的开头是否有 PDF 头部。然后读取文件末尾提到的交叉引用表，确保在最后一个%%EOF 之前查找。进入 trailer 关键字后，它调用了解析器。
- 解析器识别<<，因此改变状态并开始累积词典的键和值。
- 在词典创建模式下，解析器会不断累积对象，直到遇到>>。每个被读取的对象都会被推送到一个堆栈中。如果读取 R，则检查堆栈上的最后两个对象。如果它们是整数，则从栈中弹出，并利用其值从 QPDF 类获得间接对象句柄。QPDF 类会查阅其缓存，必要时插入一个新的未解决对象，返回指向缓存条目的对象句柄，然后将其推送到栈中。当最终读取>>时，堆栈会被转换为一个 QPDF\_Dictionary（API 无法直接访问），并将其放入 QPDFObjectHandle 中并返回。
- 生成的词典被保存为尾部词典。
- 搜索 /Prev 键。如果存在，QPDF 会尝试到该点并重复，但新的拖尾词典不会被保存。如果没有 /Prev，则初始解析过程完成。

- 如果存在加密词典，文档的加密参数会被初始化。
- 客户端通过从尾部字典获取 /Root 键的值来请求根对象并返回。它是一个未解析的间接 QPDFObjectHandle。
- 客户端请求根 QPDFObjectHandle 的 /Pages 键。QPDFObjectHandle 发现这是一个未解析的间接对象，因此请求 QPDF 解析它。QPDF 检查交叉参考表，获取偏移量，并读取该偏移量上的物体。对象缓存条目未解析的值被实际值取代，这使得之前指向该对象的未解析 QPDFObjectHandle 对象现在共享了实际对象的副本。通过任何此类 QPDFObjectHandle 进行的修改都会反映在所有版本中。随着客户端继续请求对象，每个新请求对象都会遵循相同的过程。

## 12.6 qpdf 物体内部结构

QPDFObjectHandle 的内部结构以及 qpdf 如何存储对象，在 qpdf 11 和 12 中进行了重大重写。以下是一些额外的细节。

### 12.6.1 对象内部结构

QPDF 对象有一个对象缓存，包含从文件中读取或作为间接对象添加的每个对象的共享指针。可以通过 QPDFObjectHandle 方法对这些对象中的任意一个进行更改。任何此类更改对所有指向同一对象的 QPDFObjectHandle 实例均可见。当 QPDF 对象由 QPDFWriter 编写或序列化为 JSON 时，任何更改都会被反映出来。

### 12.6.2 qpdf 11 及更新版本中的对象

#### qpdf 11

QPDF 中的对象缓存包含一个指向 QPDFObject 的共享指针。任何从该对象间接引用解析的 QPDFObjectHandle 都包含该共享指针的副本。每个 QPDFObject 对象都包含一个指向类型为 QPDFValue 的对象的共享指针。QPDFValue 类型是一个抽象基类。每种基本对象类型（数组、字典、空、布尔、字符串、数字等）都有实现，还有一些特殊对象类型，如未初始化、未解析、保留和销毁。当对象首次创建时，其底层的 QPDFValue 类型未解析。当对象首次被访问时，缓存中的 QPDFObject 会被从文件读取时的 QPDFValue 替换为该对象。由于所有引用 QPDFObjectHandle 对象以及拥有 QPDF 对象的 QPDF 对象都共享 QPDFObject 对象，这确保了未来对该对象的任何更改，包括通过调用 QPDF::replaceObject 或 QPDF::swapObject 替换该对象，都会反映在所有引用该对象的 QPDFObjectHandle 对象上。源自 PDF 输入源的 QPDFValue 会维护一个指向读取它的 QPDF 对象（其所有者）的指针。当该 QPDF 对象被销毁时，它会通过清除所有可访问对象的所有者来断开连接。对于间接对象（对象缓存中的所有对象），它还会用 Destroyd 类型的对象替换该对象的值。这意味着，如果仍然存在任何引用 QPDFObjectHandle 对象，请求它们拥有的 QPDF 会返回一个空指针，而不是指向无效或指向其他东西的 QPDF 对象的指针，任何试图访问与已销毁 QPDF 对象关联的间接对象都会抛出异常。该作还会破坏任何循环引用（这些引用常见且在某些情况下是 PDF 规范要求的），从而防止在 QPDF 对象被销毁时发生内存泄漏。

#### qpdf 12

在 qpdf 12 中，每个 QPDFObject 中包含的共享指向 QPDFValue 的指针被 std::variant 取代。基础类 QPDFValue 被合并进 QPDFObject，其子类成为独立类。

### 12.6.3 qpdf 11 之前的对象

在 qpdf 11 之前, QPDFValue 和 QPDFObject 类的功能都包含在一个 QPDF 类中, 该类既是 QPDF 的缓存条目, 也是所有不同 PDF 对象类型的抽象基类。行为几乎相同, 但存在一些问题:

- 虽然所有引用 QPDFObjectHandle 对象通过变异对 QPDFObjectHandle 进行变更都能看到, 但用 QPDF::replaceObject 或 QPDF::swapObject 替换对象时, QPDF 无法通知指向旧 QPDFObject 的 QPDFObjectHandle 对象。为了绕过这个问题, 每次尝试访问 QPDFObjectHandle 指向的底层对象时, 都必须询问拥有该对象是否发生变化, 如果发生了, QPDF 必须更换其内部的 QPDFObject 指针。这会增加每次间接对象访问的开销, 即使没有更改对象。
- 当 QPDF 对象被销毁时, 任何引用该对象的 QPDFObjectHandle 对象都会保留一个潜在无效的指针作为拥有的 QPDF。实际上, 这通常不是问题, 因为通常人们不需要维护被销毁的 QPDF 对象的 QPDFObjectHandle 副本, 但在可能的情况下, 其他软件需要自行做账务, 以确保对象的所有者仍然有效。

这些问题通过将 QPDFObject 拆分为 QPDFObject 和 QPDFValue 来解决。

## 12.7 选角政策

本节描述了选角政策及其后续 qpdf 的实施过程。这对 qpdf 的终端用户来说无关紧要, 对使用 qpdf 的代码编写者来说也无关紧要, 但对于将 qpdf 移植到新平台或对代码进行修改的人来说, 可能会引起兴趣。

qpdf 中的 C++ 代码不含旧式铸造, 除非不可避免 (例如旧式铸造出现在第三方头文件提供的宏中)。当需要 cast 时, 按偏好顺序处理, 通过重写代码以避免 cast、调用 const\_cast、调用 static\_cast、调用 reinterpret\_cast 或上述组合来处理。作为最后手段, 编译器专用 #pragma 可以用来抑制我们不愿修复的警告。例如, 可能包括在 C 和 C++ 代码共享的代码中, 抑制关于使用旧式铸造的警告。

QIntC 命名空间由 include/qpdf/QIntC.hh 提供, 实现了整数类型转换的安全函数。这些函数会进行范围检查, 并在从一种整数类型转换到另一种整数类型导致信息丢失时抛出 std::range\_error, 这是 std::runtime\_error 的子类。由于交互接口中使用的不兼容整数类型, 我们不得不在不同整数类型之间切换。有些问题是不可避免的, 比如在大小和偏移量之间切换, 另一些则是因为代码过于根深蒂固, 无法修复而不破坏源代码兼容性, 给用户带来麻烦。qpdf 编译时带有额外警告, 用于检测潜在数据丢失的转换, 所有此类情况应通过 QIntC 的函数或 static\_cast 来解决。

当仅仅是因为在不兼容的接口之间交换数据而切换类型时, 可以使用 QIntC。这就是常见的情况。然而, 在某些情况下, 我们明确打算使用完全相同的比特模式但类型不同。这在切换签名和无符号字符时最为常见。很多 qpdf 代码内部使用无符号字符, 但 std::string 和 char 是有符号的。使用 QIntC::to\_char 来从无符号字符转换为带符号字符是错误的, 因为负字符值和对应的大于 127 的无符号字符值表示的是同一件事。还有一些情况下, 我们在处理位域时会用 static\_cast 表示, 而不是表示一个数值, 而是用一堆比特组合成某种整数类型。还要注意, size\_t 和 long 通常在 32 位和 64 位环境中存在差异, 因此有时在某一平台上不需要显式投射以避免警告, 但在另一个平台上可能需要。即使底层大小相同, 类型不同时也应始终使用 QIntC 转换。QPDF 的自动构建支持 32 位和 64 位平台, 测试套件非常全面, 因此很难在构建或测试中犯任何潜在错误而不被发现。

## 12.8 加密

qpdf 对加密支持透明。打开 PDF 文件时，如果存在加密字典，QPDF 对象会使用提供的密码（如有）处理该字典。主解密密钥被计算并缓存。此后不再访问加密词典。当从文件中读取对象时，所包含对象的对象 ID 和生成过程总是已知。利用这些信息以及存储的加密密钥，所有流和字符串对象都能透明地解密。原始加密对象绝不会存储在内存中。这样，库里就没有人会知道或在意它是否在读取加密文件。

还提供了一个接口，用于写入给定加密密钥的加密流和字符串。QPDFWriter 在重写加密文件时会使用这个方法。

在复制加密文件时，除非另有指示，qpdf 会保留原始文件中正在进行的加密。QPDF 可以通过用户密码或所有者密码实现这一点。不同密码在能力上没有区别。当使用 40 位或 128 位加密密钥时，用户密码可以用拥有者密码恢复。拥有 256 个密钥时，用户和所有者密码分别用于加密实际加密密钥，因此无论哪种密钥都可以使用，但所有者密码已无法用来恢复用户密码。

从 4.0.0 版本开始，qpdf 可以读取未加密但包含加密附件的文件，但无法写入此类文件。QPDF 还要求密码必须指定才能打开文件，而不仅仅是提取附件，因为一旦文件打开，所有解密都是透明处理的。在保持加密的情况下复制文件时，qpdf 会对文件中的所有内容施加加密，而不仅仅是附件。解密文件时，qpdf 会解密附件。一般来说，复制带有多种加密格式的 PDF 文件时，qpdf 会选择最新的格式。唯一的例外是，如果原始文件中明文元数据保持为明文，则会被保留为明文。

有些人对加密 PDF 文件的一个困惑是，加密和密码保护不同。密码保护的文件总是加密的，但也可以创建没有密码的加密文件。内部文件使用空字符串作为密码，大多数读取器先尝试空字符串以验证是否有效，只有在空字符串无效时才提示密码。通常这类文件的用户密码为空，拥有者密码为空。这样，如果普通读者在未指定密码的情况下打开文件，就可以执行加密词典中规定的限制。大多数用户甚至不会意识到这样的文件是加密的。由于 qpdf 总是忽略限制（除了报告限制），qpdf 不在乎你用哪个密码。QPDF 允许你创建带有非空用户密码和空所有者密码的 PDF 文件。有些阅读器在打开这些文件时会要求输入密码，而有些则会在没有密码的情况下打开文件，且不会强制执行限制。用户密码非空且拥有者密码空，其实不太合理，因为那样用用户密码打开文件会比完全不提供密码更受限制。QPDF 还允许你创建带有与用户和所有者密码相同密码的 PDF 文件。有些读取器永远不会允许无限制访问这些文件，因为如果密码能作为用户密码使用，他们从未尝试过使用拥有者密码。尽管如此，qpdf 的一个强大优势是它允许你精细指定加密文件的创建方式，即使结果对部分读者无用。其中一个用例是测试 PDF 阅读器，确保它能处理输入文件的奇怪配置。如果你尝试创建不安全的加密文件，qpdf 会警告你，并要求你明确说明你打算创建不安全的文件。所以虽然 qpdf 可能会生成不安全的文件，但它不会让你误做。

## 12.9 随机数生成

QPDF 生成随机数以支持加密数据的生成。从 qpdf 10.0.0 开始，qpdf 使用加密提供商作为随机数的来源。旧版本使用操作系统提供的安全随机数源，或者如果构建时允许，可以使用来自 stdlib 的不安全随机数。从 5.1.0 版本开始，你可以在构建时禁用作系统提供的安全随机数。如果你想避免依赖 Microsoft 的密码学 API，这在 Windows 上尤其有用。你也可以自己提供随机数据提供商。关于如何作的详细信息，请参阅源发行版中的顶层 README.md 文件以及 QUtil.hh 中的注释。

## 12.10 新增和删除页面

虽然 qpdf 的 API 支持添加和修改对象已有一段时间，但 3.0 版本引入了专门的添加和删除页面的方法。这些主要是方便例程，处理两个棘手问题：将可继承资源从 /Pages 树推送到单个页面，以及作 /Pages 树本身。详情请参见 QPDF.hh 中的 addPage 及相关方法。

## 12.11 预留对象编号

qpdf 3.0 版本引入了保留对象的概念。这些通常不需要用于普通作，但在某些情况下，你可能想向 QPDF 对象添加一系列相互引用的间接对象。这会带来问题，因为你无法确定一个新的间接对象的 ID，除非你用 QPDF::makeIndirectObject 将其添加到 QPDF 对象中。在 3.0 版本之前，唯一能向 QPDF 对象添加两个相互指涉对象的方法是先添加新对象，然后添加后再让它们相互引用。现在可以使用 QPDFObjectHandle::newReserved 创建保留对象。这是一个间接对象，即使查询其类型，也保持“未解析”状态。所以现在，如果你想创建一组相互指涉的对象，你可以为每个对象创建预留，并用这些预留来构建引用。完成后，你可以调用 QPDF::replaceReserved 来用真实的对象替换保留的对象。大多数应用程序永远不需要此功能，但 QPDF 内部在从其他 PDF 文件复制对象时会使用它，详见《从其他 PDF 文件复制对象》一文。关于如何使用保留对象的示例，可以在 qpdf 的来源中搜索 test\_driver.cc 中的 newReserved。

## 12.12 从其他 PDF 文件复制对象

qpdf 3.0 版本引入了将对象从不同的 QPDF 对象复制到另一个 QPDF 对象的功能，我们称之为外部对象。这允许任意合并 PDF 文件。qpdf 命令行工具对基本页面选择（包括从其他文件合并页面）提供了有限的支持，但库的 API 允许实现任意复杂的合并作。复制外部对象的主要方法是 QPDF::copyForeignObject。它从另一个 QPDF 中获取一个间接对象，递归地复制到该对象中，同时保留所有对象结构，包括循环引用。这意味着你可以用 QPDF::makeIndirectObject 添加你从零创建的直接对象到 QPDF 对象，也可以从 QPDF::copyForeignObject 添加一个间接对象。QPDF::makeIndirectObject 不会自动检测并复制外来对象，这其实是一个明确的设计决策。复制一个外来物体似乎是一个足够重要的行为，应该明确进行。

另一种复制外部对象的方式是通过调用 QPDF::addPage 将页面从一个 QPDF 传递到另一个 QPDF。与 QPDF::makeIndirectObject 不同，该方法会自动区分当前文件中的间接对象、外部对象和直接对象。

当你将对象从一个 QPDF 复制到另一个时，原始文件的输入源必须保持有效，直到你完成目标对象的处理。这是因为输入源仍用于从复制对象中检索任何引用的流数据。如有需要，有方法强制复制数据。详情请参见 include/qpdf/QPDF.hh 中 copyForeignObject 声明附近的评论。

## 12.13 编写 PDF 文件

qpdf 库支持通过 QPDFWriter 类将 QPDF 对象的文件写入 PDF 文件。QPDFWriter 类有两种写入模式：一种用于非线性文件，另一种用于线性化文件。关于线性化的实现描述，请参见线性化。本节介绍了我们如何编写非线性化文件，包括创建 QDF 文件（参见 QDF 模式）。

这个大纲是在实施之前写的，虽然不完全准确，但它展现了写作的本质。查看 QPDFWriter 中的代码以获取详细信息。



- 初始化状态：
  - 下一个对象编号 = 1
  - 对象队列 = 空
  - 重编号表：旧对象 ID/生成到新 ID/0=空
  - xref 表：新 ID ->偏移量=空
- 从文件创建一个 QPDF 对象。
- 为新 PDF 文件写入头部。
- 请求预告片词典。
- 对于每个间接对象的值，通过返回并递增该数字的作，获取下一个对象编号。将对象映射到重编号表中的新数字。把对象推到队列。
- 虽然队列中还有更多对象：
  - 弹尾巴。
  - 在重新编号表中查找物体的新编号 n。
  - 将当前偏移存储到 xref 表中。
  - 写作：samp: '{n}' 0 obj.
  - 如果对象是空的，无论是直接还是间接，都写出空，从而消除不可解析的间接对象引用。
  - 如果对象是流流，将流内容写入，并根据需要通过任何过滤程序传输到内存缓冲区。利用该缓冲区来确定流的长度。
  - 如果对象不是流、数组或词典，则写出其内容。
  - 如果对象是数组或词典（包括流），遍历其元素（对数组）或值（对词典），处理递归词典和数组，寻找间接对象。当发现间接宾语时，如果无法解析，则忽略。（写作时处理此情况。）否则，去重新编号表里查一下。如果找不到，获取下一个可用的对象号，将被引用的对象分配到重新编号表中的对象，然后将被引用的对象推入队列。作为特殊情况，在编写流字典时，可以根据需要替换长度、滤波器和解码参数。

写出字典或数组，将所有不可解析的间接对象引用替换为空（pdf 规范中说引用不存在对象是合法的，且解析为空），而所有可解析的引用则引用到重新编号的对象。

——如果对象是流，写流\n、流内容（来自内存缓冲区）和\nendstream\n。

——完成后，写下 endobj。

完成队列后，所有引用对象都会被写出，所有已删除或未被引用的对象都会被跳过。新的交叉引用表将包含每个新对象号的偏移量，从 1 到写入的对象数。这可以用来写出新的 xref 表。最后，我们可以编写带有适当计算 /ID（参见规范 8.3，文件标识符）、交叉引用表偏移量和 %%EOF 的尾部词典。

## 12.14 过滤流

流的支持通过为该库设计的管道接口实现。

读取流时，创建一系列管道对象。Pipeline 抽象基要求实现 `write ()` 和 `finish ()`，并提供了 `getNext ()` 的实现。每个流水线对象在接收数据后，会执行它将要做的事情，然后将数据（可能经过修改）写入其后继对象。或者，管道也可以是终端管道，将输出存储到文件或内存缓冲区，忽略后继程序。更多详情请查看 `Pipeline.hh`。

QPDF 可以读取原始或过滤流。当读取过滤流时，QPDF 类会为每个相应的过滤对象创建一个管道对象，并将其串联起来。最后一个过滤器应写入所需的输出类型。QPDF 类具有一个接口，可以将原始或过滤后的流内容写入给定的管道。

## 12.15 对象访问器方法

有关如何访问 `QPDFObjectHandle` 实例的一般信息，请参见 `QPDFObjectHandle.hh` 中的评论。搜索“Accessor methods”。本节对该行为及其背后的理由进行了更深入的讨论。

为什么类型错误会被当成警告？当类型检查在早期引入 qpdf 时，人们预期类型错误只会因程序员错误而发生。然而，实际上，PDF 文件格式错误会发生，因为代码中的假设，包括 qpdf 库内的代码和库用户编写的代码。最常见的情况是通过串联调用 `getKey ()` 来访问字典深层的密钥。在许多情况下，qpdf 能够从这些情况下恢复，但旧行为往往导致崩溃而非优雅恢复。因此，错误被改为警告。

既然用户通常无能为力，为什么还要警告类型错误？类型警告在开发过程中极为宝贵。由于编译时无法发现字典键名中的错别字或 PDF 文件结构的逻辑错误，类型警告的存在可以节省大量开发者时间。它们也被证明有助于揭示 qpdf 本身可能未被发现的问题。

有没有一个类型安全的 `QPDFObjectHandle`？在 qpdf 11 发布时，正积极致力于创建一个更符合类型安全、更贴近当前 C++ 标准库的 PDF 对象工作方式。希望这项工作能使现代语言如 Rust 中写 qpdf 绑定变得更容易。如果实现这一点，通常是通过提供 `QPDFObjectHandle` 的替代方案，提供通往底层对象的独立路径。具体细节仍在敲定中。从根本上说，PDF 对象并不是强类型。它们类似于 JSON 对象或动态语言中的对象，比如 Python：有些事情只能对特定类型的对象做，但你可以用另一种类型的对象替换一个对象。因此，运行时总会有一些检查。

为什么 C 和 C++ API 在类型异常的行为上会有所不同？在 C 语言中，除了 `setjmp` 和 `longjmp` 这样的方法，没有办法抛出并捕捉异常，而这种方法无法跨语言障碍移植。由于 C API 经常被用来替代其他语言，保持尽可能简单非常重要。从 qpdf 10.5 开始，曾经用 C API 导致代码崩溃的异常将默认写入 `stderr`，并且可以注册错误处理程序。错误处理程序完全可以以某种方式模拟异常处理，比如使用 `setjmp` 和 `longjmp`，或者设置某个变量，在库调用后可以检查。事后看来，如果 C API 对象处理方法像其他方法一样返回错误码，并在传递指针中设置返回值，可能会更好，但这会使实现和库的使用变得复杂，因为这种情况实际上非常罕见且大多可以避免。

我怎样才能完全避免出现类型警告？对于每个返回请求类型值并对错误类型对象发出警告的 `getSomethingValue` 访问器，还有一个 `getValueAsSomething` 方法（自 qpdf 10.6 起）对于错误类型的对象返回 `false`，否则返回 `true` 并初始化引用。这些方法从不生成类型警告，并且提供了在调用访问器方法前显式检查对象类型之外的替代方案。

## 12.16 智能指针

本节描述了 qpdf 10.6.0 和 11.0.0 中对智能指针使用的变更。

在 qpdf 11.0.0 中, `PointerHolder` 被 qpdf 公共 API 中的 `std::shared_ptr` 取代。提供了一个向后兼容的 `PointerHolder` 类, 使得大多数代码可以保持不变。`PointerHolder` 可能最终会被完全从 qpdf 中移除, 但这还要等一段时间, 以便需要支持多个版本的 qpdf 用户更容易使用。

在 10.6.0 版本中, `PointerHolder` 进行了一些增强以简化过渡。这些中间步骤仅适用于 10.6.0 至 10.6.3 版本, 但仍能帮助代码的渐进修改。

`POINTERHOLDER_TRANSITION` 预处理器符号在 qpdf 10.6.0 中引入, 帮助用户从 `PointerHolder` 过渡到标准 `::shared_ptr`。如果你不定义这一点, `PointerHolder` 将被完全排除在 API 之外 (从 qpdf 12 开始)。下面解释了该符号的不同可能值及其含义。

从 qpdf 11.0.0 开始, 包括定义了符号 `POINTERHOLDER_IS_SHARED_POINTER`。这可以与条件编译结合使用, 使得支持不同版本的 qpdf 成为可能。

本节其余部分将提供详细信息。

### 12.16.1 过渡性增强至指针持有者

在 qpdf 10.6.0 中, `PointerHolder` 做了一些调整, 以便更容易准备过渡到标准版 `::shared_ptr`。这些改进也让你更容易逐步升级代码。`PointerHolder` 进行了以下修改, 使其行为更接近标准化程序 `::shared_ptr`:

- `get()` 作为 `getPointer()` 的替代选项被添加
- `use_count()` 是作为 `getRefCount()` 的替代方案添加的
- 一个新的全局辅助函数 `make_pointer_holder` 表现类似于 `std::make_shared`, 所以你可以用 `make_pointer_holder(args...)` 创建一个以新的 `T(args...)` 作为指针的 `PointerHolder`。
- 一个新的全局辅助函数 `make_array_pointer_holder` 取一个大小并创建一个指向数组的 `PointerHolder`。它是新加入的 `QUtil::make_shared_array` 方法的对应, 后者对 `std::shared_ptr` 实现同样的处理。

`PointerHolder` 有一个长期存在的漏洞: `const PointerHolder` 只能在其 `getPointer` 方法中提供 `T const*`。这是错误的, 也不是标准库 C++ 智能指针或普通指针的行为方式。正确的语义是, `Const PointerHolder` 在创建后不会接受新的指针 (`PointerHolder` 一直以这种方式正确表现), 但仍允许你修改被指向的物品。如果你不想变异它指向的东西, 可以用 `PointerHolder`。新的 `get()` 方法表现正常。因此它与 `getPointer()` 不完全相同, 但它的行为与 `get()` 在 `std::shared_ptr` 下表现相同。这对任何正确编写的代码都没有影响。

### 12.16.2 `PointerHolder` 和 `std` 的区别: `::shared_ptr`

以下是从 `PointerHolder` 迁移到标准版时需要考虑的事项清单: `::shared_ptr`。列表结束后, 我们将讨论如何使用 `POINTERHOLDER_TRANSITION` 预处理器符号或其他 C++ 编码技术来对每个问题进行处理。

- `PointerHolder` 有一个隐式构造函数, 取一个 `T*`, 这意味着你可以直接将 `T*` 分配给一个 `PointerHolder`, 或者把 `T*` 传递给期望 `PointerHolder` 作为参数的函数。`STD::shared_ptr` 没有这种行为, 不过你仍然可以将 `nullptr` 分配给 `std::shared_ptr`, 并将 `nullptr` 与 `std::` 进行比较 `shared_ptr`。以下是一些你可能需要修改代码的示例:

旧代码:

```
指针持有者 x_p; X* x = 新
X (); x_p = x;
```

新代码:

```
自动 x_p = 标准: : make_shared (); X* x
= x_p.get (); 或者, 安全性较低但更接
近: STD: : shared_ptr x_p; X* x = 新 X
(); x_p = 标准: : shared_ptr (x);
```

旧代码:

```
指针持有者 base_p; Derived* 派生 = new
Derived (); base_p = 导出;
```

新代码:

```
STD: : shared_ptr base_p; Derived* 派生 = new
Derived (); base_p = std: : shared_ptr (派生);
```

- PointerHolder 必须 getPointer () 才能获取底层指针。它还拥有很少使用的 getRefCount () 方法来获取引用计数。STD:: shared\_ptr 有 get () 和 use\_count ()。  
在 qpdf 10.6 中, PointerHolder 还有 get () 和 use\_count ()。

### 12.16.3 解决差异

如果你还没准备好采取行动, 可以在包含任何 qpdf 头文件前 #define POINTERHOLDER\_TRANSITION 0, 或者在构建中添加该符号的定义。这将提供向下兼容的 PointerHolder API, 且不会有任何弃用警告。这应该是临时措施, 因为 PointerHolder 未来可能会消失。如果你需要支持新版和旧版的 qpdf, 还有其他选项, 下面会详细说明。

注意, 即使是 0, 你也应该重建并测试你的代码。如果你有 PointerHolder 容器, 编译器可能会出错, 但大多数代码应该能不做任何修改即可编译。在 qpdf 的 API 中, PointerHolder 的容器没有被使用。

你可以做两件重要的事情来减少从 PointerHolder 切换到 PointerHolder 的影响

STD:: shared\_ptr:

- 在处理与 qpdf API 交换的 PointerHolder 变量时, 尽量使用 auto 和 decltype。
- 使用 POINTERHOLDER\_TRANSITION 预处理器符号来识别并解决上述差异。

要使用 POINTERHOLDER\_TRANSITION, 你需要在包含任何 qpdf 头文件之前先 #define 它, 或者在构建中指定其值。下表描述了 POINTERHOLDER\_TRANSITION 的数值。这些信息也在 include/qpdf/PointerHolder.hh 中总结, 所以你无需查阅这本手册就能随时掌握。

表 1: POINTERHOLDER\_TRANSITION 数值

价值含义	
a-	和 4: PointerHolder 没有定义。
的——	
罚款	
0	提供向后兼容的 PointerHolder 并抑制所有弃用警告;支持所有之前的 QPDF 版本
1	明确 PointerHolder (T*) 构造函数;生成代码支持所有之前的 qpdf 版本
2	弃用 getPointer () 和 getRefCount () ;需要 QPDF 10.6.0 或更高版本。
3	弃用所有 PointerHolder;需要 QPDF 11.0.0 或更高版本
4	禁用 qpdf/PointerHolder.hh 的所有功能, 使 #include 除定义 POINTERHOLDER_IS_SHARED_POINTER 外没有其他影响;需要 QPDF 11.0.0 及更高版本。

基于上述内容, 以下是准备代码的流程。这就是 qpdf 代码本身所采用的程序。

你可以在不中断 qpdf 版本支持的情况下完成以下步骤:

- 查找代码中所有指向者的位置。看看这些选项是否可以直接被 std:: shared\_ptr 或 std:: unique\_ptr 替换。如果你在采用 C++11 之前一直在使用 qpdf, 并且把 PointerHolder 作为通用智能指针使用, 你可能会遇到可以用这种方式替换的案例。

例如:

- 简单 PointerHolder 构造可以被等效的 std:: shared\_ptr 构造替代, 或者如果构造函数是公的, 则用 std:: make\_shared (args. ...). 如果你创建的智能指针从不被复制, 可能可以用 std: : unique\_ptr。

- 数组分配需要重写。

为数组分配一个指针持有者如下:

```
指针持有者 p (true, 新 X[n]);
```

要将 std:: shared\_ptr 分配到数组:

```
auto p = std: : shared_ptr (new X[n], std: :d efault_delete ()) ;
如果你不介意用 QUtil, 可以参考 QUtil: : make_shared_array (n) 。// 如果你用的是
c++20, 可以用 std: : make_shared (n) // 来获得 std: : shared_ptr代替 std: :
shared_ptr。
```

要将 std:: unique\_ptr 分配给数组:

```
auto p = std: : make_unique (n) ;
或者, 如果 X 有私有构造子:
auto p = std: : unique_ptr (new X[n]) ;
```

- 如果 PointerHolder 不能被标准库智能指针替代, 因为它用旧的 qpdf API 调用, 也许可以用 auto 或 decltype 声明, 这样在用更新的 qpdf API 变更构建时, 代码只需重新编译即可。
- #define POINTERHOLDER\_TRANSITION 1 用于对所有隐含构造从普通 T\* 构建的 PointerHolder 进行弃用警告。找到后, 明确构建指点持有者。

——旧代码:

```
指针持有者 x = 新 X ( ) ;
```

– 新代码:

```
auto x = 指针持有者 (new X ( ... ) ) ; // qpdf 所有版本
或者, 如果 X ( ... ) 是公共的:
自动 x = make_pointer_holder ( ... ) ; // 仅 10.6 及以上
```

其他例子也在上文中出现。

如果你需要支持 10.6 以上的旧版 qpdf, 这就是你能做到的极限, 无需条件编译。

从 qpdf 11.0.0 开始, 包括定义了符号 `POINTERHOLDER_IS_SHARED_POINTER`。如果你想支持旧版 qpdf, 同时切换到不使用向后兼容的 `PointerHolder`, 可以通过测试 `POINTERHOLDER_IS_SHARED_POINTER` 预处理器符号来区分旧代码和新代码, 如下

```
#include < qpdf/PointerHolder.hh> #ifdef
POINTERHOLDER_IS_SHARED_POINTER
STD: : shared_ptr X;
#else
指针持有者 x;
#endif // POINTERHOLDER_IS_SHARED_POINTER
x = decltype (x) (new X ( ) )
```

或

```
#include < qpdf/PointerHolder.hh> #ifdef
POINTERHOLDER_IS_SHARED_POINTER
自动 x_p = 标准: : make_shared ( ) ; X* x = x_p.get
( ) ; #else 自动 x_p = 指针持有者 (新 X ( ) ) ; X* x
= 1 x_p.getPointer ( ) ; #endif //
POINTERHOLDER_IS_SHARED_POINTER x_p->doSomething
( ) ; x->doSomethingElse ( ) ;
```

如果你不需要支持较旧的 qpdf 版本, 可以继续这些步骤, 而不用预处理器符号保护更改。以下是剩余的变更。

- `#define POINTERHOLDER_TRANSITION 2` 以实现 `getPointer ( )` 和 `getRefCount ( )` 的弃用
- 将 `getPointer ( )` 替换为 `get ( )`, `getRefCount ( )` 替换为 `use_count ( )`。这些方法在 10.6.0 之前并不存在。

当你的代码能够干净地编译成 `POINTERHOLDER_TRANSITION=2` 时, 你就已经很有可能彻底淘汰 `PointerHolder`。目前代码无法支持 10.6.0 之前的任何 qpdf 版本。

为了支持 qpdf 11.0.0 及更新版本并从代码中移除 `PointerHolder`, 请继续以下步骤:

- 除了字面语句外, 所有 `PointerHolder` 的出现都替换为 `std: : shared_ptr` `#include <qpdf/PointerHolder.hh>`
- 将所有 `make_pointer_holder` 的出现替换为 `STD: : make_shared`
- 将所有 `make_array_pointer_holder` 的出现替换为 `QUtil: : make_shared_array`。如果你还没写, 你需要补充。

- 确保你在时都包含在内。
- 如果你使用了任何数组 PointerHolder 对象，请按上述方式替换它们。你可以让编译器帮你找到这些。
- #define POINTERHOLDER\_TRANSITION 3 以实现所有 PointerHolder 构建的弃用。
- 组装和测试。修复所有剩余的问题。
- 如果不支持旧版 qpdf，请删除所有对的引用。否则，你仍然需要包含它，但可以 #define POINTERHOLDER\_TRANSITION 4 个，以防止 PointerHolder 被定义。POINTERHOLDER\_IS\_SHARED\_POINTER 符号依然会被定义。

#### 12.16.4 历史背景

自诞生以来，qpdf 库就使用了自己的智能指针类 PointerHolder。PointerHolder 类最初创建于标准:: shared\_ptr 之前，qpdf 本身直到 2019 年底发布的 9.1.0 版本才开始要求 C++11 编译器。在当前的 C++ 版本中，qpdf 不再需要拥有自己的智能指针类。





## QPDFJOB：基于岗位的界面

qpdf 命令行执行文件的所有功能都可以通过 C++ 库中的 QPDFJob 类实现。有几种方式可以访问此功能：

- 命令行选项
  - 运行 qpdf 命令行
    - 使用 C++ API 中的 QPDFJob: : initializeFromArgv
    - 使用 C API 并 qpdfjob\_run\_from\_argv qpdfjob-c.h.如果你是从 Windows 风格的主系统调用，并且有一个 argv 数组 wchar\_t，你可以用 qpdfjob\_run\_from\_wide\_argv。
- 作业 JSON 文件格式
  - 使用 CLI 中带有 --job-json-file 参数的
    - 使用 C++ API 中的 QPDFJob: : initializeFromJson
    - 使用 C API 并 qpdfjob\_run\_from\_json qpdfjob-c.h
  - 注意：这与 --json 无关，但可以与之结合。关于 qpdf JSON（与 QPDFJob JSON），参见 qpdf JSON。
- The QPDFJob C++ API

如果你能理解如何使用 qpdf CLI，你也能理解 QPDFJob 类和 JSON 文件。QPDF 保证上述所有方法同步。具体流程如下：

表 1: QPDFJob 接口

CLI	JSON	C++
--some-option "someOption" : " " config () ->someOption () --some-option=value "someOption" : "value" config () ->someOption ( "value" )		
立场论元	"otherOption" : "value" config () ->otherOption ( "value" )	

在 JSON 文件中，JSON 结构是一个对象（字典），其键是转换为 camelCase 的命令行标志。位置参数有对应的键，你可以通过运行带有 --job-json-help 标志的 qpdf 来找到。例如，输入和输出文件由 CLI 上的位置参数命名。在 JSON 中，它们出现在 “inputFile” 和 “outputFile” 键中。以下是等价的：

CLI:

```
QPDF infile.pdf outfile.pdf \ --页。other.pdf --
password=x 1-5 -- \ --加密用户所有者 256 --print=low
-- \ --object-streams=generate
```

作业 JSON:

```
{
  "inputFile": "infile.pdf",
  "outputFile": "outfile.pdf",
  "侍从": [
    {
      "文件": "。"
    },
    {
      "file":
        "other.pdf",
      "password": "x",
      "范围": "1-5"
    }
  ],
  "encrypt": {
    "userPassword": "user",
    "ownerPassword": "owner",
    "256bit": {
      "print": "低"
    }
  },
  "objectStreams": "生成"
}
```

C++代码:

```
#include < qpdf/QPDFJob.hh>
#include #include

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

    试试看
    {
        QPDF Job;
        j.config ()
            ->inputFile ( "infile.pdf" ) ->outputFile
              ( "outfile.pdf" ) ->pages () ->pageSpec
              ( ".", "1-z" ) ->pageSpec
              ( "other.pdf", "1-5", "x" ) -
            >endPages () ->encrypt (256, "user",
                                   "owner" ) ->print ( "low" ) -
            >endEncrypt ()

        ->objectStreams ( "生成" ) -
        >checkConfiguration () ;j.run () ;} catch
        (QPDFUsage& e) {

            std::cerr << "配置错误: " << e.what () << std::endl;
            返回 2;
        }
    }
}
```

(续见下一页)

(续自上一页)

```

    } catch (std::exception&
e) {
    std::cerr << “其他错误: ” << e.what() << std::endl;
    返回 2;
    }
    返回 0;
}

```

请注意上面提到的 QPDFUsage 例外。每当配置出现错误时，都会抛出该程序。这些信息完全对应 qpdf CLI 发出的使用信息，用于遗漏输出文件、多次指定 -页或其他无效选项组合。QPDFUsage 由 argv 和 JSON 接口以及原生的 QPDFJob 接口抛出。

也可以从 CLI 中混合使用命令行选项和 JSON。例如，你可以创建一个名为 my-options.json 的文件，包含以下内容：

```

{
  “加密” : {
    “userPassword” : “”,
    “ownerPassword” : “owner”,
    “256bit” : { },
    “objectStreams” : “generate”
  }
}

```

并与其他选项结合使用，创建 256 位加密（但无限制）的对象流文件，同时在命令行中指定其他参数，如

```
qpdf infile.pdf outfile.pdf --job-json-file=my-options.json
```

另见源代码发行版中的 examples/qpdf-job.cc 以及 QPDFJob.hh 中的评论。

## 13.1 QPDF 乔布设计

本节介绍了 QPDFJob 背后的一些设计理念和历史。

QPDFJob 的文档分为三个部分：

- README-maintainer 中的“如何添加命令行参数”提供了一个快速提醒，说明如何添加命令行参数。
- 源文件 generate\_auto\_job 详细说明了 QPDFJob 和 generate\_auto\_job 如何协同工作。
- 手册的这一章节还有其他细节。

在 qpdf 10.6.0 版本之前，qpdf CLI 执行文件内置了许多功能，这些功能本身无法从库中调用。这带来了许多问题：

- qpdf.cc 中的一些逻辑相当复杂，比如图像优化、生成 JSON 输出以及许多页面作。虽然这些都可以用 C++ API 编写，但会有很多重复的代码。
- 随着 QPDF 支持更广泛的文档层面选项，页面分割和合并会随着时间变得更复杂。如果能把这些信息暴露给库用户，而不是全部内置在 CLI 里，那就太好了。

- 其他语言的用户如果只是想做一个 CLI 能做的事情的接口，却没有很好的方式，比如直接给库调用一组命令行选项或一个等效的 JSON 对象，这些对象可以作为字符串传递。
- qpdf 的 CLI 本身代码接近 8,000 行。它需要重构、清理和拆分。
- 通过命令行暴露新功能需要对大量代码进行大量小修改，而且很容易忘记。添加代码生成器虽然在某些方面复杂，但大大降低了扩展 qpdf 时的错误概率。

以下是关于 QPDFJob 及其各种界面设计决策的一些说明。

- 裸命令行选项（无参数的标志）映射到不接受任何选项的配置函数和要求为空字符串的 JSON 键。其理由是，之后我们可以将这些裸选项改为带有可选参数的选项，而不会破坏 CLI 或 JSON 的向后兼容性。取可选参数的选项会产生两个配置函数：一个没有参数，另一个带有 `char const*` 参数。这意味着在之前空缺的选项中添加可选参数也不会破坏二进制兼容性。
- 在 `job.yml` 中添加新参数会自动触发几乎所有需要实现的内容。这样，一旦你拿到代码编译和链接，就知道没有遗漏任何内容。有两个棘手的情况：
  - 如果参数处理程序需要执行特殊作，比如调用嵌套配置方法或选择选项表，你必须手动实现。这在 `generate_auto_job` 中有讨论。
  - 当你添加一个带有可选参数或选择的选项时，上述两个处理程序都会声明，但只有引用参数的那个。你必须记住实现那个不接受参数的，否则别人尝试调用时会收到链接错误。假设带有可选参数的事物一开始就是裸的，所以无参数版本已经存在。
- 如果你需要添加一个需要独立选项表的新选项，你就需要做一些额外工作，比如添加新的嵌套配置类，在 `QPDFJob_argv.cc` 中添加 `ArgParser` 的配置成员变量，在 `QPDFJob_json.cc` 中添加 `Handler`，并确保手动实现的 `handler` 彼此一致。最好为所有达到该选项的各种方式添加明确的测试用例。

## 线性化

本章介绍了 QPDF 和 QPDFWriter 如何实现线性化 PDF 的创建和处理。

### 14.1 线性化的基本策略

为了避免 qpdf 库自己验证线性化文件的复杂问题，我们采用了一种特殊的线性化文件检查模式，可以通过 `qpdf --check-线性化`（或 `qpdf --check`）调用。该模式读取线性化参数字典和提示流，并验证对象排序、参数和提示流内容的正确性。验证代码首先在外部工具（Acrobat 和 `pdlin`）创建的线性化文件中进行测试，随后用于验证由 QPDFWriter 自身创建的文件。

### 14.2 准备线性化

在从任何其他 PDF 文件创建线性化 PDF 文件之前，必须修改 PDF 文件，使所有页面属性向下传递到页面层级（且不继承 /Pages 树中的父属性）。我们还需要知道哪些对象指向哪些其他对象，涉及页面边界和其他一些情况。我们将准备 PDF 文件的这一部分称为优化，详见优化部分。注意，在此语境中，优化一词是 qpdf 术语，线性化是 PDF 规范中的术语。不要被许多应用将线性化称为优化或网页优化所混淆。

在从优化的 PDF 文件创建线性化 PDF 文件时，实际上只需要解决几个问题：

- 提示表的创建
- 正确放置物体
- 填充偏移量和字节大小

### 14.3 优化

为了执行线性化和文件拆分等各种作，需要知道哪些对象被哪些页面引用、页面缩略图以及根和尾部字典键。还必须确保所有页面级属性直接出现在页面层面，而不是从页面树中的父属性继承而来。

我们称执行这些约束的过程为优化。如上所述，注意一些应用将线性化称为优化。虽然这一优化最初是出于创建线性文件的需求，但我们分别使用了这两个术语。

PDF 文件优化在 `QPDF_optimization.cc` 源文件中实现。该文件注释丰富，是优化过程的主要参考。

优化完成后，QPDF 中的私有成员变量 `obj_user_to_objects` 和 `object_to_obj_users` 已被填充。任何在

object\_to\_obj\_users 张桌子是共用的。任何在 object\_to\_obj\_users 表中恰好有一个值的对象都是私有的。要找到页面、尾部或根字典键中的所有私有对象，只需对该页面或键的每个元素在 obj\_user\_to\_objects 表中做出此判定。请注意，页面和缩略图的用户类型不同，因此上述页面测试不会包含页面缩略图字典引用的对象，其他都不包括。

## 14.4 编写线性文件

我们将创建仅包含主提示流的文件。我们绝不会写溢出提示流。（截至 PDF 1.4 版本，Acrobat 也没有，而且它们从来都不是必需的。）提示流包含指向如果提示流不存在时它们所在位置的对象的偏移信息。这意味着我们必须计算所有物体的位置，才能生成并编写提示表。这意味着我们必须分两次生成文件。为了保证可靠性，QPDFWriter 在线性化模式下会调用完全相同的代码两次，将文件写入管道。

在第一遍中，目标流水线是一个计数流水线，连接到丢弃流水线。计数流水线只是将数据传递到链中的下一个流水线，但可以返回在中间点通过的字节数。丢弃流水线是一种终端流水线，直接丢弃数据。提示流不被写入，带有足够填充的虚拟值存储在第一个交叉引用表、线性化参数字典和第一个尾部词典的/Prev 键中。所有偏移量、长度、对象重新编号信息以及我们第二次处理所需的其他信息都会被存储。

第一次处理结束时，这些信息会传递给 QPDF 类，QPDF 类在内存缓冲区中构建一个压缩的提示流并返回。QPDFWriter 利用这些信息将完整的提示流对象写入内存缓冲区。此时，提示流的长度已知。

在第二遍中，流水线链的末端是一个普通文件，而不是丢弃流水线，我们已经有了所有偏移量和长度的已知值，而第一遍没有这些。我们必须根据提示流开始后出现的偏移量，按已知的提示流长度进行调整。任何长度可变的都被填充，填充码包围了在两次处理中不同的写入代码。这确保了表示方式的改变不会导致第一遍收集的偏移在第二遍时变得不正确。

利用这种策略，我们可以将线性化文件写入不可寻址的输出流，只需一次传递到磁盘或输出的任何地方。

## 14.5 线性化数据的计算

一旦文件优化完成，我们就能知道哪些对象访问哪些其他对象。然后我们可以处理这些表格，决定每个对象包含在哪个部分（如 PDF 规范中的“线性化 PDF 文档结构”中描述的那样）。这告诉我们对象书写的确切顺序。QPDFWriter 类请求这些信息，并按正确顺序排队对象。它还会开启一个检查，如果遇到尚未排队的对象，会触发异常。（这只有在用于计算线性化数据的遍历代码存在错误时才会发生。）

## 14.6 线性化的已知问题

这种线性化代码存在一些已知问题。这些问题似乎不会影响线性化文件的行为，这些文件仍能按预期工作：网页浏览器可以在文件完全下载前就开始显示它们。事实上，似乎其他创建线性文件的程序也存在许多类似的问题。这些条目涉及 PDF 规范线性化附录中使用的术语。

- 线程词典的信息键出现在第 4 部分，与线程的其他部分一起，而不是第 9 部分。第 9 部分中的对象在功能上并未被归类。
- 我们并不是在内容流中计算共享对象位置的分子，也没有在内容流中交错它们。

- 我们只生成页面偏移表、共享对象和大纲提示表。添加一些额外的表格相对容易。我们收集了制作缩略提示表所需的大部分信息。代码中有相关注释。

## 14.7 调试说明

qpdf --show-linearization 命令可以显示线性化提示流的完整内容。要查看原始数据，你可以使用 qpdf --show-object=n --filtered-stream-data 提取线性化提示表的过滤内容。然后，要将数据转换为比特流（因为线性化表是不考虑字节边界的比特流），你可以通过以下 perl 代码将所得数据传输：

```
使用字节；
binmode STDIN;
undef $/;
我的$a = ;我的@ch = split (//, $a); map {
printf ( "%08b", ord ($_) ) } @ch; print
"\n" ;
```





## 对象流和交叉引用流

本章提供了关于 qpdf 中对象流和交叉引用流支持的实现信息。

### 15.1 对象流

对象流可以包含任何普通对象，但以下除外：

- 流对象
- 生成为>0 的对象
- 加密词典
- 包含另一流/Length 的对象

此外，Adobe 阅读器（至少从版本 8.0.0 起）似乎无法处理如果文件加密，文档目录出现在对象流中，尽管规范并未明确禁止此作。

线性化文件还有额外的限制。详情请参见线性文件的启示。

PDF 规范将对象流中的对象称为“压缩对象”，无论对象流是否被压缩。

对象流中每个对象的生成数必须为零。可以删除并替换对象流中的一个对象，并用普通对象替换。

对象流词典具有以下键：

- /N：对象数量
- /First：第一个对象的字节偏移量
- /Extends：间接指称其延伸的流

流集合由 /Extend 组成。它们必须形成一个有向无环图。这些可以用于语义信息，对 PDF 文档的语法结构没有意义。虽然 qpdf 保存了流集合，但它从不生成这些数据，也不会以任何方式利用这些信息。

规范建议限制对象流中的对象数量，以提高读译的效率。Acrobat 6 对于线性化文件，每个对象流的对象不超过 100 个对象，对于非线性化文件，每个流最多使用 200 个对象。QPDFWriter 在对象流生成模式下，从不在一个对象流中放置超过 100 个对象。

对象流内容由 N 对整数组成，每个整数对分别是对象编号和相对于流中第一个对象的字节偏移量，随后是连接的对象本身。

## 15.2 交叉引用流

对于非混合文件，startxref 之后的值是 xref 流的字节偏移量，而不是字 xref。

对于混合文件（包含 xref 表和交叉引用流的文件），xref 表的尾部字典包含键/XRefStm，其值为补充 xref 表的交叉引用流的字节偏移量。符合 PDF 1.5 标准的应用程序应先读取 xref 表。然后它应该用 xref 流中定义的任意对象替换它已经看到的任何对象。然后它应该跟随原始 xref 表的拖尾函数中的任何 /Prev 指针。规范中没有明确说明，如果 xref 流中的 /Prev 指针被 xref 表引用，应该做些什么。QPDF 类忽略它，这可能是合理的，因为如果出现这种情况，任何合理的 PDF 文件，之前的 xref 表很可能有对应的/XRefStm 指针。例如，如果附加了一个混合文件，附加的部分将拥有自己的 xref 表和 /XRefStm。附加的 xref 表会指向之前的 xref 表，而之前的 xref 表则指向 /XRefStm，这意味着新的 /XRefStm 不必指向它。

由于 xref 流必须非常早地读取，因此它们可能不加密，且可能不包含读取所需的间接对象，这些包括：

- /类型：值 /XRef
- /Size：值 n+1：其中 n 是最高的对象数（与拖尾词典中的 /Size 相同）
- /索引（可选）：值 [: samp: '{n count}' ...]用于确定该流中存储哪些对象的信息。默认值为 [0 /Size]。
- /前：value offset：前一个 xref 流的字节偏移量（与拖尾词典中的 /Prev 相同）
- /W [...]: xref 表中每个字段的大小

xref 流中的其他字段，如果需要可以是间接的，是 xref 表尾部字典中字段的合并。

### 15.2.1 交叉引用流数据

流数据为二进制，按大端序编码。条目被串接，每个条目长度等于上述 /W 条目总和。每个条目由一个或多个字段组成，第一个字段是字段的类型。每个字段的字节数由上述 /W 给出。0 在 /W 中表示该栏被省略，且为默认值。字段类型的默认值是 1。其他默认值都是 0。

PDF 1.5 有三种字段类型：

- 0：自由对象。格式：0 obj，下一代，与传统交叉引用表中的自由表相同
- 1：规则非压缩物体。格式：1 次偏移生成
- 2：对象流中的对象。格式：2 个对象流编号索引，即包含该对象的对象流数量以及该对象流中的索引。

如果没有删除对象，表格中的第一个条目通常会 0 0 0 0，而不是 0 0 ffff。

## 15.3 线性文件的影响

对于线性化文件，线性化词典、文档目录和页面对象可能不包含在对象流中。存储在对象流中的对象在主页和首页交叉引用部分中，对象编号范围最大。

使用交叉引用流代替常规的 xref 表是可以的。有特别的考虑因素。

提示数据指的是对象流本身，而不是流中的对象。还应对对象流进行共享对象引用。在任何提示表中都没有关于压缩对象（对象流内对象）对象编号的参考。

在给对象编号时，线性化文件的第一半和第二半部分内所有共享对象必须在该半部分所有正常未压缩对象之后连续编号。

## 15.4 实现说明

写入对象流有三种模式：禁用、保留和生成。在禁用模式下，我们不生成任何对象流，同时生成一个 xref 表，而不是 xref 流。这可以用来生成可通过旧阅读器查看的 PDF 文件。在保留模式下，我们写入对象流，使得写入的对象流包含与原始文件相同的对象和 /Extends 关系。如果文件没有对象流，这就等同于禁用。在生成中，我们自己创建对象流，将允许进入对象流的对象分组为不超过 100 个对象的集合。我们还确保生成模式下 PDF 版本至少为 1.5，但在其他模式下保留版本头。默认是保留。

我们不支持创建混合文件。写入文件时，即使是保留模式，也会丢失所有 xref 表并合并所有附加的部分。



## PDF 加密

本章以一般性的方式讨论 PDF 加密，并从 QPDF 中了解其工作原理。本章并非旨在取代 PDF 规范。详情请参阅规范。

### 16.1 PDF 加密概念

#### 加密

加密是用加密文本（也称为密文）替代明文。如果已知加密密钥，可以从密文中检索明文。

PDF 文件由对象结构组成。PDF 对象可以有多种类型，包括（其中包括）数字、布尔值、名称、数组、字典、字符串和流。在 PDF 文件中，只有字符串和流被加密。

#### 安全处理员

自 PDF 诞生以来，文件加密方式经历了多次修改。加密由安全处理者负责。标准的安全处理程序基于密码。这是 qpdf 唯一实现的安全处理程序，所有材料都聚焦于标准安全处理程序。有各种标志用标准安全处理程序控制加密的具体细节。以下将讨论这些内容。

#### 加密密钥

这指的是加密和解密算法实际使用的密钥。它与密码是不同的。主加密密钥随机生成并加密存储在 PDF 文件中。用于保护 PDF 文件的密码（如果有的话）用于保护加密密钥。这种设计使得可以使用不同的密码（例如用户密码和所有者密码）来获取加密密钥，甚至在不更改加密密钥的情况下更改文件密码。QPDF 在使用--显示-加密密钥选项时可以暴露加密密钥，且在使用-password-is-hex-key 选项时，可以接受十六进制编码的加密密钥代替密码。

#### 密码保护

密码保护与加密是不同的。这一点常被误解。PDF 文件可以加密而不使用密码保护。PDF 加密的初衷是设置两个密码：用户密码和所有者密码。任一密码都可以用来获取加密密钥。符合标准的读卡器应当遵守安全限制，如果使用用户密码打开文件，则不遵守。QPDF 不区分打开文件时使用的密码。通过区分用户密码和所有者密码，使得创建无密码保护的加密文件变得普遍。这是通过使用空字符串作为用户密码，使用某个秘密字符串作为拥有者密码来实现的。当用户打开 PDF 文件时，空字符串被用来检索加密密钥，使文件可用，但符合规范的读取器会限制用户的某些作。

这一切意味着什么？这里有几点需要注意。

- 由于用户密码和所有者密码都用于恢复单一加密密钥，根本无法阻止应用程序无视文件的安全限制。任何能读取加密文件的软件都拥有加密密钥。因此，对 PDF 文件施加的限制仅由软件强制执行。任何开源 PDF 阅读器都可以被轻易修改，以忽略文件的安全限制。PDF 规范对此有明确说明。这意味着对非密码保护文件的 PDF 限制只会限制那些不知道如何绕过这些限制的用户。
- 如果文件有密码保护，你必须至少知道用户或所有者中的一个密码才能获取加密密钥。然而，在 40 位加密的情况下，实际的加密密钥只有 5 字节，且很容易被暴力破解。因此，无论密码多强，采用 40 位加密的文件都不安全。在 128 位加密中，默认的安全处理程序使用 RC4 加密，但该加密也被认为是不安全的。因此，使用标准安全处理程序（截至 2022 年本章最后一次评测）安全加密 PDF 文件的唯一方法是使用 AES 加密。这是唯一支持 256 位加密的算法，并且也可以选择与 128 位加密一起使用。然而，AES 没有理由使用 128 位加密。如果你打算用 AES，就用 256 位加密吧。带有强密码的 256 位 AES 加密 PDF 文件的安全性相当于使用通用加密工具如 gpg 或 openssl 对 PDF 文件进行加密，但使用 PDF 加密的优势在于无需除普通 PDF 查看器外的软件。

16.2 PDF 加密详情

本节介绍了关于 PDF 加密的一些细节。它没有描述所有细节。关于这些，请阅读 PDF 规范。不过，这里提供的细节应该能大大帮助普通用户/开发者理解加密 PDF 文件的运作。

这里有更多需要理解的概念。

算法参数 V 和 R

使用标准安全处理程序控制加密细节的两个参数：V 和 R。

V 是编码，指定用于加密文件、处理密钥等的算法。它可以具有以下任一的值：

表 1：加密算法：V

V 含义	
1	最初的算法是用 40 位密钥加密文件。
2	这是原始算法的扩展，允许更长的密钥。在 PDF 1.4 中引入。
3	一种未公开的算法，允许文件加密密钥长度从 40 位到 128 位不等。在 PDF 1.4 中引入。qpdf 被认为能够读取 V = 3 的文件，但不写入此类文件。
4	算法的扩展，允许通过额外规则参数化处理字符串和流。在 PDF 1.5 中引入。
5	一种允许为字符串、流以及嵌入文件指定独立安全处理程序的算法，并支持 256 位密钥。在 PDF 1.7 扩展第 3 级引入，后来在第 8 级扩展中扩展。这就是 PDF 2.0 规范 ISO-32000 中的加密系统。

R 是规范标准处理器版本的代码。它与 V 的值紧密耦合。R 可以具有以下任一值：

表 2: R 与 V 的关系

R	预期 V
2	V 必须是 1
3	V 必须是 2 或 3
4	V 必须是 4
5	V 必须是 5;该扩展从未被完全规范，且在某些 Acrobat 版本中存在过一段时间。QPDF 能够读取和写入该格式，但不应用于测试与该格式兼容性的其他用途。
6	V 必须是 5。这是唯一一个在 PDF 2.0 规范 ISO-32000 中未被弃用的值。

加密词典

加密的 PDF 文件有一个加密字典。有几个领域，但以下是我们目的中重要的几个：

- 如上所述的 V 和 R。
- O、U、OE、UE：用于从用户和所有者密码恢复加密密钥的算法所使用的数值。这些定义和使用方式取决于 R 的值。
- P：描述存在哪些限制的位域。这部分内容将在 PDF 安全资料中详细讨论限制

加密算法

PDF 文件可以用过时且不安全的 RC4 算法或更安全的 AES 算法进行加密。另请参见“弱密码学”一节讨论。40 位加密始终使用 RC4。128 位可以使用 RC4（出于兼容性的默认选项），或者从 PDF 1.6 开始使用 AES。256 位加密始终使用 AES。

16.3 PDF 安全限制

PDF 的安全限制由一个位字段描述，其值存储在加密词典中的 P 字段中。算法利用 P 的值来恢复密码后加密密钥，这使得 P 的值具有防篡改性。

P 是一个 32 位整数，被视为带符号的二补数。任何位位置的 1 表示许可已授予。PDF 规范将位编号为 1（最低有效位）到 32（最高有效位），而不是更常见的 0 到 31。为符合规范，本节剩余部分采用以 1 为基础的编号。

仅使用第 3、4、5、6、9、10、11 和 12 位。其他所有位都设置为 1。由于第 32 位始终设为 1，P 的值总是为负数。（qpdf 代表那些将 P 视为无符号的有错误写入者，识别一个正数。此类文件曾在野外被发现。）

以下是钻头位置的含义。所有未列出的位必须为 1，唯独第 1 和第 2 位必须为 0。然而，表中其他位的值会被忽略，因此错误的值在大多数情况下可能不会破坏任何东西。值为 1 表示许可已获批准。

表 3：P 位值

Bit	含义
3	对于 R = 2 次印刷;对于 R ≥ 3，低分辨率打印
4	修改文档，除非由第 6、9 和 11 位控制
5	提取文本和图形，用于非视障用户的无障碍用途
6	添加或修改注释，填写交互式表单字段;如果第 4 位也被设置，则创建或修改交互式表单字段
9	对于 R ≥ 3，即使第 6 位是清空的，也要填写交互式表单字段
10	未被使用;曾授权提取无障碍材料，但规范现禁止限制无障碍，符合规范的读者应视此位为固定，无论其值如何
11	对于 R ≥ 3，组建文档，包括插入、旋转或删除页面，或创建文档大纲或缩略图
12	对于 R ≥ 3，允许以全分辨率打印

16.4 qpdf 如何处理安全限制

本节详细描述了 qpdf 库在不同安全选项设置下对 P 的作用。

- 开始时，除第 1 和第 2 位外的所有位都已设置，这两位已被清除
- 清晰的部分，详见下表：

表 4：命令行参数和 P 位值

R	论点	比特已清除
R = 2	--print=n	3
R = 2	--modify=n	4
R = 2	--extract=n	5
R = 2	--注释=n	6
R = 3	--可达性=N 10	
R ≥ 4	--可及性=n 被忽略	
R ≥ 3	--extract=n	5
R ≥ 3	--打印=无	3, 12
R ≥ 3	--print=低	12
R ≥ 3	--modify=non	4, 6, 9, 11
R ≥ 3	--修改=assembly 4, 6, 9	
R ≥ 3	--modify=形式	4, 6
R ≥ 3	--modify=注释 4	
R ≥ 3	--assembly=n	11
R ≥ 3	--注释=n	6
R ≥ 3	--形式=n	9
R ≥ 3	--modify-other=n 4	

在 CLI 和库层面，qpdf 的选项比大多数工具（包括 Adobe Acrobat）更能细致地清除权限位。因此，PDF 查看器可能会根据提交给 qpdf 的选项产生意想不到的反应。如果你观察到这种情况，很可能不是因为 qpdf 的某个漏洞。



## 16.5 用户与所有者密码

当你用 qpdf 显示加密参数，打开带有所有者密码的文件时，有时 qpdf 会显示用户密码，有时不会。原因如下。

对于  $V < 5$ ，用户密码实际上存储在 PDF 文件中，该文件用源自所有者密码的密钥加密，主加密密钥则用源自用户密码的密钥加密。当你打开 PDF 文件时，阅读器首先尝试将给定密码视为用户密码，并用它来恢复加密密钥。如果这样有效，你就有限制了（假设读者选择执行）。如果无效，读卡器会将密码视为所有者密码，用它恢复用户密码，然后用用户密码获取加密密钥。这就是为什么用  $V < 5$  创建一个拥有相同用户密码和所有者密码的文件，会导致一些阅读器永远不允许你以所有者身份打开。当文件创建时给出空的所有者密码，用户密码会同时作为用户密码和所有者密码使用。通常当读卡器遇到  $V < 5$  的文件时，首先会尝试将空字符串视为用户密码。如果这样可以，文件是加密的，但没有密码保护。如果不行，就会提示密码提示。

对于  $V \geq 5$ ，主加密密钥分别使用用户密码和所有者密码进行加密。没有办法从所有者密码中恢复用户密码。限制是否施加取决于使用的密码。在这种情况下，如果有提供的密码，会同时尝试作为用户密码和所有者密码，使用有效的密码。通常密码会先用所有者的密码来尝试。（这正是 PDF 规范中所说的。）因此，指定用户密码而保持所有者密码空白，会导致文件以所有者身份打开但没有密码，实际上使安全限制失效。这就是为什么 qpdf 要求你通过 `-allow-insecure` 才能在使用 256 位加密时创建一个拥有者密码为空的文件。



## 发行说明

这是一份经过精心整理的用户和开发者面向的变更列表。在版本 12 之前，文件 ChangeLog 包含更多细节。从版本 12 开始，请参阅版本控制历史以获取更多细节。

12.3.2: 2026 年 1 月 24 日

- 漏洞修复
  - 修复 12.3.0 引入的错误。如果同一文件多次指定 `--password` 会触发使用错误。在使用 QPDFJob 界面时，在 `--pages` 选项中多次指定密码很常见。

12.3.1: 2026 年 1 月 19 日

- 漏洞修复
  - 修复在调用被销毁的 QPDFJob 对象副本时，`QPDFJob::run` 和 `QPDFJob::createQPDF` 的失败。这会影影响使用 `pikepdf` 的作业界面。

12.3.0: 2026 年 1 月 10 日

- 发行变更
  - 从 12.3.0 版本开始，我们使用联名签名而非 GPG 来签署发布。请参阅顶层 README.md 获取说明。我们将继续在 12.x 系列中使用 GPG。从 `qpdf` 13 版本开始，只使用联名签名。
- 构建变更
  - 现在构建或测试 `qpdf` 需要 C++20 编译器。所有公共头文件仍兼容 C++17。
  - `ApplImage` 和 Linux 独立二进制发行版现已基于 Ubuntu 22.04 构建，以支持 C++20，这意味着它们无法在某些较旧的 Linux 发行版上运行。如果您需要支持较旧的发行版，请使用 12.2.0 或更低版本。
- 漏洞修复
  - 始终在 `QPDFFormFieldObjectHelper::getTopLevelField` 中设置 `is_different` 标志。在——仅当该场地不是顶级赛场时，旗帜才会被设置为真，其他状态保持不变。
  - 解析 `qpdf` JSON 输入文件允许空名称对象。这些是 PDF 规范允许的，但之前被拒绝了。
  - `QPDFAcroFormDocumentHelper::fixCopiedAnnotations` 现在正确更新任何注释的 `/P` 条目，指向拥有页面。
- 图书馆增强

- 添加 QPDFNameTreeObjectHelper 和 QPDFNumberTreeObjectHelper 构造器重载，允许传递函数以验证树中的值。
- 新增 QPDFNameTreeObjectHelper 和 QPDFNumberTreeObjectHelper 验证方法，用于验证并可选修复名称/数字树。
- 为所有 DocumentHelper 类添加新的获取和验证方法。get 方法检索共享的 DocumentHelper，避免了反复验证底层文档结构和/或构建内部缓存的开销。如果直接修改底层文档结构（不使用 DocumentHelpers），验证方法会重新验证结构并重新同步内部缓存。
  - 新增缓冲区方法：移动、视图、数据、大小和空。新方法将缓冲区呈现为字符（而非无符号字符）容器，便于将内容高效迁移到 `std::string` 中。
- 在 `qpdf::global` 命名空间中添加各种新功能，用于访问和设置/修改全局设置及限制。详情请参见全局选项和头文件 `qpdf/global.hh`。
- 添加新的 C-API 函数 `qpdf_global_get_uint32` 和 `qpdf_global_set_uint32` 以访问和设置/修改各种全局设置和限制。

#### • 构建修复

- 尝试检测任何 C++17 变更被偷偷带入任何公共头部，并检查所有私有头部是否独立编译。

#### • CLI 增强功能

- 禁止与不兼容的选项 `---deterministic-id` 一起使用，不兼容的加密或复制加密。
- 选项 `---check` 现在包含了对 AcroForm、Dests、Outline 和 PageLabels 结构的额外基本检查。
- 添加新选项 `--global`，用于设置或修改各种全局选项和限制。详情请参见“全球选项”。
- 修正补全脚本和处理，以避免在完成过程中参数泄漏到环境中，并正确处理 `zsh` 用户的 `bashcompinit`。
- 新增 `--remove-acroform` 选项，以排除 AcroForm 词典的输出 PDF。当损坏的 PDF 文件使用 `--flatten-annotations` 选项时，这个选项特别有用。

#### • 其他增强

- 新增检测模式，以协助检查和手动修复损坏的 PDF 文件。在此模式下，如果 PDF 文件损坏且无法修复，部分异常会被警告替换，并抑制部分自动修复。仅支持非常有限的作范围。检查模式通过新函数 `qpdf::global::options::inspection_mode` 选择。更多细节请参见检查模式。

- QPDFWriter 在写入空流时将不再添加过滤器。
- 当恢复带有损坏 xref 表的文件时，增加了更多合理性检查。

#### • 其他变化

- QPDF 代码库的大部分部分经历了重大内部重构。
- 默认情况下，过滤条件超过 25 个的流现在被视为不可过滤。大量过滤器通常出现在损坏或专门构建的文件中，可能导致资源过度使用

和/或堆栈溢出。如有需要，可以通过新的 `--max-stream-filters` CLI 选项或新的 `qpdf::global::max_stream_filters` 函数更改限制。

- 在使用 GnuTLS 加密提供者的 FIPS 环境中运行时，调用 GnuTLS 现在使用“LAX”模式，因为解密现有文件需要使用弱算法，且 PDF 标准规定该用途与加密无关。用户需确保在必要时符合 FIPS 标准。
- 现在在带有无效尾部的文件上调用 `QPDF::getRoot` 时，会跳出 `damaged_pdf` 错误，提示为“无法找到 /Root 字典”，而非内部错误。
- 无效的根对象/类型条目现在被无条件修复。以前只有在使用 `--check` 选项时才会修复。
- 将 `--compress-streams` 设置为 `n` 或 `QPDFWriter::setCompressStreams(false)` 不再自动导致输出文件解密。如果这是预期的行为，则设置 `--decrypt`。
- 流过滤也进行了一些重构。这些过滤器针对未注册用户提供的流过滤的常见情况进行了优化，方法是调用 `QPDF::registerStreamFilter`。如果您提供自己的流过滤，请开工单。

• 以下内容被认为未被使用，已被弃用。如果你依赖他们，请开工单。

- `QPDF::compute_encryption_key`
- 所有 `QPDF::EncryptionData` 方法。这些方法不会导出到共享库中，只能在静态链接程序中使用。

## 12.2.0: 2025 年 5 月 4 日

• 即将到来的 C++ 版本变更

- 预计这将是最后一个支持 C++17 的 qpdf 小型版本。我们将在 12.3.0 版本中切换到 C++20。

• 漏洞修复

- 在 `QPDF::getAllPages` 中检测共享的 /Kids 数组，以避免（特别构建的）损坏输入文件中的栈溢出。
- 修复 `QPDFFormFieldObjectHelper` 中一些（特别构建的）损坏输入文件的严重性能问题。
- 添加缺失的 `QPDFFormFieldObjectHelper::isChecked` 实现。
- 修复 `QPDFNameTreeObjectHelper` / `QPDFNumberTreeObjectHelper` 中的 bug。在某些条件下——插入树时，/Range 条目被写入树根节点，这是不允许的。其中一个可能的后果是，有些读者无法识别嵌入或附加文件。
- 在 `QPDFFormFieldObjectHelper::getChoices` 中，如果 /Opt 条目是导出值和显示字符串的对，而非代表两个值的单一字符串，则返回显示字符串。此前如果条目不是单一字符串，则不会返回任何值。

• 构建修复

- 提升 Windows 本地开发体验。使用 MinGW 时，Perl 不再被要求构建。qpdf 应该在 Windows 上“开箱即用”构建，使用像 JetBrains CLion 这样的集成环境（该集成环境捆绑 mingw），或者使用 Visual Studio 安装。运行测试仍需使用 Perl 和类似 POSIX 的环境，如 msys2。

<sup>1</sup> 未进入检查模式

- 修复 Android 构建问题。
- 修正 12.1.0 引入的--jpeg-quality 特性引入的 jpeg 库错误使用。这导致部分平台的构建失败。
- 其他增强
  - 在恢复带有损坏 xref 表的文件时，增加了更多合理性检查，以避免长时间运行和大量内存使用。具有非常大数组或词典（超过 5000 个元素）且重复页面的对象被忽略，因为它们几乎肯定无效。

#### 12.1.0: 2025 年 4 月 6 日

- 漏洞修复
  - 在 QPDF::isLinearized 中，如果文件中的第一个对象不是线性化参数词典，或者其/L 条目不是整数对象，则返回 false。此前，如果第一个词典对象不是线性化参数词典，该方法会返回 false。
  - 固定包含未用空白分隔对象的对象流的解析。2020 年前版本的 PDF 规范错误地说明了对象之间需要留白。QPDF 在解析对象流时依赖于这一点。
  - 修复两个报告错误对象 ID 的对象流错误/警告消息。
  - 在 qpdf 作业 JSON 中接受旋转数组，因为它是一个可重复的选项。
  - 在阅读带有明文元数据的加密 PDF 时，只期望顶层的 /Metadata 是明文。编写带有明文元数据的加密 PDF，只需保持顶层未加密。qpdf 一直错误地将所有 /Metadata 流作为带有明文元数据的特殊处理。
- 图书馆增强
  - 添加函数 PL\_DCT::make\_compress\_config，返回 PL\_DCT::CompressConfig 唯一指向从 std::function 的 CompressConfig 指针，提供更现代的配置选项。
- CLI 增强功能
  - 新增 --remove-structure 选项，将文档结构树排除在输出 PDF 中。
  - 新的 --jpeg-quality 选项，用于设置 --optimize-images 的 jpeg 质量。
- 其他增强
  - 对于带有损坏 xref 表的文件恢复方式进行了进一步增强。
- 构建变更
  - .idea/cmake.xml 文件已被删除。我们不再以 CLion 专用配置随某些 CMake 配置文件出厂，而是包含一个 CMakePresets.json。有关于在 README-maintainer.md 中使用它的信息。对大多数用户来说，正常运行 cmake 是可以的。欢迎提出建议。官方版本在最初引入时都没有使用 cmake 预设。
- 其他变化
  - QPDF::optimize 方法被认为未被使用，已被弃用。如果你依赖它，请开一个工单。
  - 对象流的解析，包括错误/警告消息的创建和对象描述，已重构，同时在运行时间和内存使用方面有所改进。
  - QPDFWriter 经过了一些重构，包括对象流的写入方式，并提升了性能。

#### 12.0.0: 2025 年 3 月 9 日

## • API 的破坏性变更

– 如果包含了 qpdf/QPDFObject.hh 的头文件，现在会生成错误。这样做是为了防止包含该内容的代码因为系统某处安装了旧版本而意外工作。

不要包含那个头部，而是加上，并在代码中用 `::ot_` 替换 `QPDFObject::ot_`。

– 已废弃的 `QPDFObjectHandle::replaceOrRemoveKey` 方法已被移除，因为它与 `QPDFObjectHandle::replaceKey` 相同。

– 已废弃的 `JSON::checkDictionaryKeySeen` 函数已被移除。如果 `JSON::parse` 遇到重复键，最后一个值会被静默接受，而不是抛出运行时错误。这与 JSON 规范一致。

– 已废弃的 `QPDFObjectHandle::getJSON` 无版本超载功能已被移除。

– 已废弃的缓冲区复制构造器和赋值作符已被移除。缓冲区复制成本高昂，因为它们总是涉及复制缓冲区内容。使用 `buffer2 =`

`buffer1.copy ()` ;或 `Buffer buffer2{buffer1.copy ()}`;明确表示复制是有意为之。

– `QIntC.hh` 包含函数名中的拼写错误 `substract`，现已修正为 `subtract`。

– 受保护的 `QPDFObjectHelper::oh` 数据成员已被新的访问器方法 `QPDFObjectHelper::oh ()` 取代。

– 除抽象类和以下列出的例外外，不支持 qpdf 类的子类化。这些职业从未设计为基础职业，最终将在 13 版中确定。如果您有延长这些课程的使用场景，请开启工单。

例外情况：

\* `QPDFDocumentHelper`

\* `QPDFObjectHelper`

– 不支持将上抛至 `QPDFObjectHelper` 和 `QPDFDocumentHelper`。他们的毁灭者将在 13 版中被保护。

– 不支持捕捉因使用未初始化 `QPDFObjectHandle` 而抛出的逻辑错误。在版本 13 中，未初始化的对象句柄将被视为不可变的共享空对象。

使用它们不会再抛出逻辑错误，但可能在适当情况下生成类型警告或异常。

## • 以下内容被认为未被使用，已被弃用。如果你依赖他们，请开工单。

– 所有 `QPDFTokenizer` 推送模式方法。

– `QPDFObjectHandle::parse overload` 取 `QPDFTokenizer` 参数。

## • CLI 的故障变更

– 为支持未来引入子命令，不再支持使用无扩展名且路径元素为首参数的文件名，未来结果可能会发生变化。例如，qpdf 检查 out.pdf 目前将文件检查复制到 out.pdf 但

未来可能会检查 out.pdf。请使用 `qpdf ./check out.pdf` 或 `qpdf --请 check out.pdf`。

## • 漏洞修复

– 在对象流中，忽略偏移量无效的对象。报告 ID 或偏移量不佳的对象。

## • 图书馆增强

- QPDFObjectHandle 支持移动构造/分配。这个变化对大多数开发者来说是看不见的，但如果你依赖于 QPDFObjectHandle 底层对象引用数量的特定行为，可能会导致代码崩溃。你得专门写代码来实现这一点，所以如果你不确定，也不用担心。
- 大多数 QPDFObjectHandle 访问器方法现在已具备 const 资格。
- QPDFObjectHandle 及所有对象辅助类现在都可以显式转换为 QPDFObjGen，因此在需要 QPDFObjGen 时可以作为参数传递。冗余的超载方法已被移除。
- 所有对象辅助类现在都可显式转换为 QPDFObjectHandle。
- 构建变更
  - 如果 POINTERHOLDER\_TRANSITION 未定义，则自动定义为 4，这完全移除了指点持有者。它不再出现在任何 qpdf 头中。  
这意味着未完成 PointerHolder 转换的代码会报错，除非它定义了 POINTERHOLDER\_TRANSITION，任何使用 PointerHolder 的文件都必须明确包含它，而不是依赖其他头部来实现。
- 其他变动
  - 对象的内部实现经过了大量重构，使用 std::variant 去除了一层间接处理。这为每个对象保存了一个共享指针，同时在运行时间和内存占用方面都有所提升。新增了一个类 BaseHandle 作为 QPDFObjectHandle 和 QPDFObjectHelper 的通用基类，以提供适用于所有类对象句柄类（如可转换为 QPDFObjGen 的作符）的通用功能。BaseHandle 是一个实现细节，库用户无法直接使用。
  - qpdf 内部对数组和词典的迭代方式进行了重大重构。
  - 用于检查对象大小以确认不同版本之间二进制兼容性的内部机制已发生变化。因此，CHECK\_SIZES 维护者专用构建选项已被移除。

#### 11.10.1: 2025 年 2 月 15 日

- 构建修复
  - 修正 zopfli 检测错误。
  - 在运行测试套件时，将 cygwin perl 识别为 Windows。

#### 11.10.0: 2025 年 2 月 8 日

- 漏洞修复
  - 检测并打破轮廓（书签）树中的循环。
  - 正确处理以词典形式提供带有“/D”条目的大纲（书签）项。
  - 加载对象流时，忽略未包含在 xref 表中的对象。PDF 规范要求任何不在 xref 表中的对象都被视为空对象。
  - 以 JSON 写实数时，确保它们不以尾数点结尾。后尾带“.”的数字是有效的 PDF 格式，但在 JSON 格式中不适用。
- 当调用 QPDF::getObject、getObjectByObjGen 或 getObjectByID 时，QPDFObjGen XREF 中不存在该函数，对象表返回的是直接的空。此前，这些方法在对象表中插入对空对象的间接引用，可能隐藏具有相同对象 ID 的有效对象。



- 修正混合参考文件中某些已删除对象的处理。此前，如果删除的对象出现在/XRefStm 条目指定的交叉引用流中，qpdf 会错误加载该对象。
- 将流解码级别默认为通用。此前解码级别错误地默认为无，影响了 --decode-level CLI 选项和 QPDFWriter:: setDecodeLevel 方法。
- 拒绝带参数的 CLI 标志。此前该参数被简单忽略（例如，--encrypt=n 被视为 --encrypt）。

- CLI 增强功能

- fix-qpdf 命令现在允许将输出文件指定为可选的第二个参数。  
这对于无法将二进制文件写入标准输出的环境非常有用（如 PowerShell 5）。
- 新增了 ---remove-metadata 和 --remove-info 选项，用于排除文档元数据和信息。

- 图书馆增强

- QPDF 现在可以支持 ZOPFLI 构建。详情请参见 Zopfli 压缩算法。
- 添加 QPDFObjectHandle 算子布尔。如果对象句柄被初始化，则该作符返回为真，并且是 isInitialized 方法的替代。更多详情请参见 qpdf 维基。
- 新的 C API 函数，qpdf\_oh\_free\_buffer 释放 malloc 分配的缓冲区。

- 其他增强

- 在加载 PDF 文件时，xref 表的处理过程进行了部分重构，包括重建损坏文件的 xref 表。作为这些内容的一部分，还增加了额外的验证。因此，一些损坏的文件在加载时会出现错误，而不是在后续处理或写入时出现。损坏文件的修复得到了改进。
- 作为 PDF 文件加载时额外验证的一部分，非字典对象现在会自动从页面树中移除。
- 处理腐败过滤流的方式发生了变化。如果压缩流无法成功解压，qpdf 现在会写入原始（编码）流，即使已设置译码级别的广义或专用流。尝试解码损坏流的结果通常无法使用，且可能非常庞大。

### 11.9.1: 2024 年 6 月 7 日

- 漏洞修复

- 重做一段线性化，以避免在非常复杂文件上出现栈溢出

- 建筑改进

- 在 Windows 上添加 CLion 构建配置，用于使用静态库，使用 Visual C++。该配置可直接与 CLion、Visual C++ 及外部库二进制分发版配合使用，无需额外外部工具。
- 调整使用 std:: string\_view 以应对 C++ 标准即将发生的变化。

### 11.9.0: 2024 年 2 月 24 日

- CLI 增强功能

- 添加新的命令行参数 --file 和 --range 可用于 --页面，替代位置参数。允许 --file 也在 --overlay 和 --底层中使用 --file。  
这些新选项可以自由地与立场论点混合。

——允许——叠加和——底层叠加重复。它们可以在命令行中多次出现，并且会按出现顺序叠加。在 QPDFJob JSON（参见 QPDFJob：基于作业的接口）中，覆盖和底层键可能包含数组。为了兼容性，它们也可能包含一个词典。

- 图书馆增强

- 将 `file ()`、`range ()` 和 `password ()` 添加到 `QPDFJob::PagesConfig`，作为 `pageSpec` 的替代方案。
- 添加 `QPDFObjectHandle::writeJSON` 以直接将对象的 JSON 表示写入管道。这比调用 `QPDFObjectHandle::getJSON` 快得多。

- 其他增强

- JSON 解析器的可靠性在用户眼前有所改进。JSON 解析器已被加入 OSS-Fuzz 的模糊测试中。

### 11.8.0: 2024 年 1 月 8 日

- 漏洞修复：

- 在扁平注释时，保留那些本质上没有外观信息的超链接和其他注释。

- CLI 增强功能

- 在数值区间语法中引入 `x`，以允许排除页面范围内的页面。详情请参见页面范围。
- 支持逗号分隔的数值，使用 `--collate` 以从不同组中选择不同数量的页面。

- 添加 `--set-page-labels` 选项，以完全覆盖输出中的页面标签。

- 图书馆增强

- 添加 API 以支持 `--set-page-labels`：

- \* `QPDFJob::Config::setPageLabels`
- \* `pdf_page_label_e` 枚举类型
- \* `QPDFPageLabelDocumentHelper::pageLabelDict`

- 改进文件恢复逻辑，更好地处理交叉引用流的文件。这应该能让 qpdf 恢复一些之前会报告“找不到拖车词典”的文件。

### 11.7.0: 2023 年 12 月 24 日

- 漏洞修复：

- 在 `--compress-streams=n` 时，qpdf 仍然压缩交叉引用流、线性化提示流和对象流。这个问题已经修复了。
- 修复 qpdf JSON：语法“`n: /pdf-syntax`”现已被接受为表示名称的替代方式。它可以用于任何名称（例如“`n: /text#2fplain`”），但当名称包含二进制字符时则是必要的。例如，`/one#a0two` 必须表示为“`n: /one#a0two`”，因为单个字节 `a0` 在 JSON 中无效。
- qpdf JSON 将科学记谱法中出现的浮点数转换为不动点表示法，因为 PDF 不支持科学记谱法。
- 设置复选框值时，允许除 `/Off` 以外的任何值表示已勾选。这是规范允许的。此前，除 `/Yes` 或 `/Off` 外，其他值都被拒绝。

- CLI 增强功能：

- 在加密 PDF 文件时，允许 `--encrypt --user-password=user-password --owner-password=owner-password --bits={40,128,256}` 的语法。

这是语法的替代方案——`--encrypt user-password owner-password {40,128, 256}`，该语法将继续被支持。新语法在 shell 补全中表现更好，并允许创建以 `-` 开头的密码。

- `--remove-restrictions` 标志现在也会禁用文件中的数字签名。

- 构建强化：

- 当 qpdf 与另一种 zlib 实现连接时，qpdf 测试套件现已通过。qpdf 测试套件中没有任何对特定 zlib 输出的依赖。请参阅 README-maintainer.md 的 ZLIB 兼容性部分，了解如何维护该功能。

- 官方 Windows 安装程序现在提供在安装 qpdf 时修改 PATH 的选项。

- 软件包增强：

- 现在文档会自动生成一个 UNIX man 页面。它包含与 qpdf 相同的文本 `--help=all`。

- 图书馆改进：

- 向 C API 添加 `qpdf_c_wrap` 和 `qpdf_c_get_qpdf` 的 C++ 函数，使自定义 C++ 代码更容易与 C API 互作。参见示例/扩展 c-api。

- 向缓冲区添加方法，以便更轻松高效地处理 `std::string`。

- 添加 `QPDFAcroFormDocumentHelper::disableDigitalSignatures`，禁用所有数字签名字段，保留其视觉表示。

#### 11.6.4: 2023 年 12 月 10 日

- 漏洞修复：

- 运行 `cmake --install --component dev` 时，安装之前从开发组件中省略的 `cmake` 文件

- 修复 Linux 二进制构建，使其使用较旧的库，以便在 AWS Lambda 及其他较旧的执行环境中继续工作。

#### 11.6.3: 2023 年 10 月 15 日

- 漏洞修复：

- 修复一个 bug：qpdf 可能会丢弃二进制字符串中的字符，如果该字符前面有少于三位数字的八进制脱出字符串。该漏洞是在 11.0.0 版本中引入的。这个漏洞不会适用于默认设置的内容流。

- 线性化规范禁止需要偏移量超过 4 GB 的线性文件。qpdf 里有个 bug，当偏移量必须超过 2 GB 时它就无法工作。此点已更正。

#### 11.6.2: 2023 年 10 月 7 日

- 漏洞修复：

- 修复一个非常古老的错误，可能导致 qpdf 在某些流解码错误时调用内部结束函数两次。对于某些错误的输入文件，这可能导致 qpdf 调用 `gnutls` 或 `openssl 1`，导致它们崩溃。

- 开发变更：

——控制一些 JetBrains CLion 的 .idea 文件。我们将在未来版本中不断迭代，使 CLion 中的 qpdf 作更便捷。

11.6.1: 2023 年 9 月 5 日

- 漏洞修复:

- 修复 11.6.0 中引入的逻辑错误，修复 copyForeignObject 的修复。这个漏洞可能导致某些页面无法被复制。

11.6.0: 2023 年 9 月 3 日

- 漏洞修复:

- 修正 ASCII85 解码器中的角格。

- 在 --页面被使用且警告出现在主文件以外的地方时，正确报告警告。
  - 改进 --bash-completion 和 --zsh-completion，以更好地支持带有空格的路径。
  - 将随机数设备的检测从编译时移至运行时，以改善交叉编译。
  - 修复尝试用 copyForeignObject 复制 /Pages 对象的错误（该工具明确不允许这样做）。

11.5.0: 2023 年 7 月 9 日

- 漏洞修复

- 重复复制同一页面时，确保注释是复制的，而不是多页共享。

- 构建变更

- 新增未来构建选项。这个选项允许你测试代码与拟议的 qpdf API 变更。详情请参见构建选项。打包者：不要在启用 FUTURE 选项的情况下打包 qpdf，因为开启该选项时没有 API 或 ABI 兼容性保证。

- 图书馆增强

- 添加新方法：Buffer::copy and deprecate 缓冲区复制构造器和赋值运算符。  
缓冲副本成本高昂，应该明确进行。

- 其他变更

- 源代码格式化为 100 列，原为 80 列。进行了许多外观上的改动以及 clang-tidy 建议的改动。霍尔格先生完成了所有艰难的工作。

11.4.0: 2023 年 5 月 21 日

- CLI 增强功能

- --optimize-images 选项现在在 XObject 格式中优化图像。

- 图书馆增强

- 允许 QPDFJob 的工作流程分为读取阶段和写入阶段，以便调用者在 QPDF 对象写入前作。这增加了方法 QPDFJob::createQPDF

- 以及 QPDFJob::writeQPDF 及其对应的 C API 函数 qpdfjob\_create\_qpdf 和 qpdfjob\_write\_qpdf。

- 添加 QPDF::newReserved 作为 QPDFObjectHandle::newReserved 的更好替代方案。

- 如果你向数组添加未初始化的 QPDFObjectHandle，qpdf 会抛出一个 logic\_error。这一直都是无效的，但以前这种行为通常不会被发现。

- 漏洞修复

- 当注释只有一个出现时，忽略其出现状态。这防止了 qpdf 的注释扁平逻辑在注释状态设置错误时丢弃了注释的外观，正如某些 PDF 文件中所见。

### 11.3.0: 2023 年 2 月 25 日

- CLI 增强功能

- 新选项 `--remove-restrictions` 移除数字签名文件的安全限制。
  - 改进叠加/底层，使得带有不平衡图形状态运算符（q/Q）的页面内容不会影响后续页面的显示方式。这会改变所有叠加/底层作的输出。

- 库增强

- 新方法 `QPDF::removeSecurityRestrictions` 移除数字签名文件的安全限制。

- 漏洞修复

- 线性化警告现在像普通警告一样，包含文件名，并以 `--no-warn` 选项抑制。

- 性能提升

- 包含更多代码整理和 M. Holger 的性能改进。

### 11.2.0: 2022 年 11 月 20 日

- 构建变更

- 现在需要一个 C++-17 编译器。

- 库增强

- 将流创建函数移到 QPDF 对象中，移到它们应归属的位置。QPDFObjectHandle 里的这些文件没有被弃用，会保留。
  - 在 `QPDFTokenizer::Token` 中添加一些方便方法，用于测试令牌类型。这是 qpdf 词汇层的一部分，大多数开发者不会需要。

- 漏洞修复

- 修正 mingw 构建中缺少符号的问题。
  - 修复 OpenSSL 加密提供商的重大性能漏洞。这个 bug 导致 OpenSSL 3 时加密文件速度会降慢 6 倍到 12 倍。这包括随 qpdf 版本发布的默认 Windows 版本。
  - 修复涉及附加文件重复使用之前作为交叉引用流的对象编号的晦涩错误。

### 11.1.1: 2022 年 10 月 1 日

- 漏洞修复

- 为初始字符恰好与 Unicode 标记重合的字符串，通过字符编码解决边缘情况。
  - 修复 `ApplImage` 在被调用为嵌入可执行文件名称时丢弃第一个命令行参数的问题。另外，`fix-qdf` 出于未知原因使用了错误的运行路径，并且会使用系统上安装的 qpdf 库。

- 测试改进

- 在自动化测试中, 练习字符默认无符号的情况。
- 在 CI 中添加 ApplImage 专属测试, 以确保 ApplImage 能以预期的多种方式工作。

- 其他变化

- 包含更多代码整理和 M. Holger 的性能改进。

11.1.0: 2022 年 9 月 14 日

- 构建修复

- 移除 LL\_FMT 测试, 这些测试因交叉编译而被破坏。代码现在只用 %lld。
  - 部分符号未正确导出用于 Windows DLL 构建。
  - 强制项目专属的头文件排在构建中所有其他文件之前, 这样之前的 qpdf 安装不会破坏构建 qpdf 源代码。

- 11.0.0 发布说明中省略了包装说明:

- 在 GitHub 上, 发布标签现在由 release-qpdf-X.Y.Z 改为 vX.Y.Z, 以更符合当前做法。

11.0.0: 2022 年 9 月 10 日

- 用 STD 替换 PointerHolder: : shared\_ptr

- qpdf 专用的 PointerHolder 智能指针实现现已通过 qpdf API 完全被 std::shared\_ptr 取代。有关此变更及全面迁移计划, 请参阅 Smart Pointers。请注意, 游戏默认提供了向下兼容的 PointerHolder 类, 并且是启用的。会发出警告, 但可以通过按照手册中列出的迁移步骤关闭。

- qpdf JSON 版本 2

- QPDF 的 JSON 输出模式现为版本 2。这修复了版本 1 的几个缺陷。版本 2 的 JSON 输出明确且完整, 支持 JSON 与 PDF 之间的双向转换。提供命令行选项和库 API, 用于从 PDF 创建 JSON、从 JSON 创建 PDF, 并在对象层面从 JSON 更新现有 PDF。

- 新的命令行参数: --json-output, --json-input, --update-from-json
  - 新的 C++ API 调用: QPDF::writeJSON, QPDF::createFromJSON, QPDF::updateFromJSON
  - 新的 C API 调用: qpdf\_create\_from\_json\_file、qpdf\_create\_from\_json\_data、qpdf\_update\_from\_json\_file、qpdf\_update\_from\_json\_data 和 qpdf\_write\_json。
  - 完整文档可在 qpdf JSON 中找到。从版本 1 到版本 2 的完整变更列表可在“从 JSON v1 到 v2 的变更”中找到。

- 构建被 cmake 替换

- 旧的基于自动对话的构建已被 CMake 取代。需要使用 CMake 3.16 或更新版本。  
详情请阅读《构建与安装 qpdf》, 如果你为发行版打包 qpdf, 请阅读《Notes for Packagers》。
  - 大多数情况下, 除了熟悉如何用 cmake 构建外, 你需要知道的转换构建内容可以在《从自动配置转换到 cmake》中描述。以下是  
需要注意的一些行为变化:
    - \* 示例源默认安装在文档目录中。

\* 启用图像比较和大文件测试的配置选项已被环境变量取代。旧选项是在后台设置环境变量的。以前，要跳过图像测试，必须设置 `QPDF_SKIP_TEST_COMPARE_IMAGES=1`，这是默认的。

现在这些设置默认关闭，你必须设置 `QPDF_TEST_COMPARE_IMAGES=1` 才能启用。

\* 在默认配置中，只有在明确请求或没有其他选项时才会选择本地加密提供商。详见构建期加密货币选择（Build-time Crypto Select）一节。

\* 如果找到外部库目录，Windows 外部库默认会被检测到。

zlib、libjpeg 和 openssl 的静态库如 README-windows.md 所述提供了。

它们只兼容非调试版本。

\* 新增了一个名为 `pkg-tests` 的新目录，包含可用于烟雾测试已安装 qpdf 包的短 shell 脚本。这些工具被 Debian autopkgtest 框架使用，但其他框架也可以使用。详情请参见 `pkg-test/README.md`。

#### • 性能改进

– 新增了许多性能提升。在开发者性能基准测试中，已观察到约 20% 的提升。大部分工作，包括对 qpdf 词汇层和解析层的重大优化，均由 M. Holger 完成。

#### • CLI：重大变更

– `--display-encryption` 标志现在即使未提供正确密码也能提供加密信息。如果你指望它在这种情况下不起作用，可以参考 `--requires-password` 作为可靠的测试。

– 当指定 `--json` 时，默认的 json 输出版本已从 1 改为最新，即 2。

—— 允许弱加密标志现在在显式创建带有弱加密算法的文件时是强制的。关于讨论，请参见“弱密码学”。

#### • API：破坏性变更

– 在 qpdf 12 中弃用 `QPDFObject.hh`。包含 `qpdf/QPDFObject.hh` 的唯一用例是获取 `QPDFObject::object_type_e`。自 10.5.0 版本起，这成为 `qpdf/Constants.h` 定义的 `qpdf_object_type_e` 别名。要修复你的代码，替换任何包含的

`qpdf/QPDFObject.hh` 替换为 `qpdf/Constants.h`，并将所有 `QPDFObject::ot_` 替换为 `::ot_`。如果你需要代码向后兼容 10.5.0 之前的 qpdf 版本，可以检查预处理器符号 `QPDF_MAJOR_VERSION` 是否被定义，并且 `>= 11`。作为权宜之计，你可以 `#define QPDF_OBJECT_NOWARN` 抑制警告。

– `Pipeline::write` 现在将无符号 `char const*` 取代 `unsigned char*`。来电者不会需要修改任何东西，但你不再需要把可写的指针传递到管道。如果你已经实现了自己的流水线类，你需要更新它们。

– 移除已废弃的 `QPDFAcroFormDocumentHelper::copyFieldsFromForeignPage`。这种方法从未奏效，只在 qpdf 10.2.x 版本中出现过一些功能。

– 移除不接受 `QPDF&` 参数的已废弃 `QPDFNameTreeObjectHelper` 和 `QPDFNumberTreeObjectHelper` 构造器。

– `QPDFJob::doVerbose` 传递并调用的函数现在采用 `Pipeline&` 参数，而非 `std::ostream&` 参数。

– 故意破坏 API，以引起对写入不安全加密文件作的注意：

\* 从 `QPDFWriter` 中移除 pre qpdf-8.4.0 加密 API 方法及其对应的 C API 函数

\* 在某些 QPDFWriter 方法的名称中添加不安全，并在某些 C API 函数的名称中\_insecure，但不改变其行为

具体细节\*请参见 qpdf 11.0 中的 API 破坏变更，以及关于一般讨论的“弱密码学”。

– QPDFObjectHandle:: warnIfPossible 不再需要可选参数来抛出异常，如果没有描述。如果没有描述，它会写入默认的 QPDFLogger 错误流。

(QPDFLogger 在 qpdf 11 中新增——见下文。)

– QPDF 对象不再可以复制或分配到。由于图书馆代码的假设，这样做从来不安全。现在 API 阻止了它。如果遇到问题，可以使用 QPDF:: create () 创建 QPDF 共享指针（或者如果你需要与旧版 qpdf 兼容，可以用其他方式创建）。

#### • CLI 增强功能

– QPDF --列表附件 --冗长 包含一些关于附件的额外信息。

附件的额外信息也包含在带有 --json 的附件 JSON 键中。

– 对于加密文件，qpdf --json 在指定密码与用户密码不匹配且使用所有者密码恢复用户密码时揭示用户密码。当使用 256 位密钥时，用户密码无法从拥有者密码中恢复。

现在可以在将输出 PDF 写成标准输出时使用——--verbose 和--progress。在这种情况下，冗长和进度消息会被写入标准错误。

#### • 图书馆增强

– 新增了一个对象 QPDFLogger。详情见 include/qpdf/QPDFLogger.hh。

\* QPDF 和 QPDFJob 默认使用默认日志，但可以覆盖它们的日志。

setOutputStreams 方法在这两个类别中均已弃用。

\* QPDFObjectHandle 中一些曾经是例外的东西现在用默认日志器写错了。

\* 通过配置默认记录器，可以捕捉 setOutputStreams 遗漏的输出和错误。

\* C API 可在 include/qpdf/qpdflogger-c.h 中获得。

\* 参见示例示例/qpdfjob-save-attachment.cc 和 examples/qpdfjob-c-save-attachment.cc。

– 在 QPDFObjectHandle 中，新增方法 insertItemAndGetNew、appendItemAndGetNew，replaceKeyAndGetNew 返回新添加的项。新的方法包括 eraseItemAndGetOld、replaceKeyAndGetOld 和 removeKeyAndGetOld，返回刚刚移除的项目，或者在 replaceKeyAndGetOld 的情况下，如果该对象之前不存在，则返回空对象。

– QPDFObjectHandle:: isDestroyed 方法可用于检测间接对象 QPDFObjectHandle 是否属于已被销毁的 QPDF。任何尝试解解析此类 QPDFObjectHandle 都会抛出逻辑错误。

– QPDFObjectHandle:: getOwningQPDF 方法现在在拥有的 QPDF 对象被销毁时返回的是空指针，而非无效指针。拥有 QPDF 的间接对象将失效。直接对象只是失去了 QPDF 的所有权，但仍然有效。

– QPDFObjectHandle:: getQPDF 方法是 的替代方案

QPDF 标题:: getOwningQPDF。它返回的是 QPDF&，而不是 QPDF\*，并且当已知对象拥有拥有 QPDF 时可以使用。如果对象没有拥有 QPDF，它会抛出异常。只有间接对象才保证拥有拥有的 QPDF。直接宾语



如果最初是从仍然有效的 PDF 输入源读取的，可能就有直接对象，但也有可能存在没有拥有 QPDF 的直接对象。

- 添加方法 `QPDFObjectHandle::isSameObjectAs`，用于测试是否两个 `QPDFObjectHandle`。

对象指向同一底层对象，意味着一个对象的变化会反映在另一个对象上。请注意，这种方法不会比较对象的内容，因此两个结构上不同但完全相同的对象不会被视为同一个对象。

- 新工厂方法 `QPDF::create()` 返回 `std::shared_ptr`。

- 新增了管道方法以减少铸造量：

\*写：过载版本，除了取无符号的 `char const*` 外，还取 `char const*`  
查特续\*

\* `writeCstr`：写一个空终止的 C 字符串

\* `writeString`：写一个 `std::string`

\* 运算符 `<<`：用于空端 C 字符串、`std::strings` 和整数类型

- 新管道类型 `Pl_OStream` 写入 `std::ostream`。

- 新管道类型 `Pl_String` 附加到 `std::string`。

- 新的流水线类型 `Pl_Function` 可用于写入时调用任意函数。它支持 C++ 代码的 `std::function`，并且可以接受 C 风格的函数，这些函数通过返回值表示成功，并额外接收一个参数用于传递用户数据。

- `QUtil` 新增了将 PDF 时间戳和 `QPDFTime` 对象转换为 ISO-8601 时间戳的方法。

- 增强 JSON 类，更好地支持增量读写大量数据，而无需将所有数据存储在内存中。

- 向 `qpdfjob` 的 C API 添加使用 `qpdfjob_handle` 的新函数。就像常规的 `Qpdf` API 一样，你必须先调用 `qpdfjob_init`，把句柄传给函数，最后再调用 `qpdfjob_cleanup`。该界面比旧接口更具灵活性，旧接口仍然可用。

- 添加 `QPDFJob::registerProgressReporter` 并 `qpdfjob_register_progress_reporter`  
允许自定义进度报告器与 `QPDFJob` 一起使用。`QPDFJob` 对象必须配置为报告进度（通过命令行参数或其他方式）才能使用。

- 向 `QPDFObjectHandle::StreamDataProvider::provideStreamData` 添加新的超载  
采用 `QPDFObjGen const&` 代替单独的对象 ID 和生成参数。旧版本将继续被支持，不会被弃用。

- 在 `QPDFPageObjectHelper` 中，为大多数页面边界框方法添加 `copy_if_fallback` 参数，并在评论中说明 `copy_if_shared` 和 `copy_if_fallback` 的区别。

- 向 `Buffer` 类添加移动构造子。

#### • 其他变化

—— 在 GitHub 上，发布标签现在由 `release-qpdf-X.Y.Z` 改为 `vX.Y.Z`，以更符合当前做法。

- 在 JSON v1 模式下，如果指定“pages”（或任何具有修复页面树效果的键），“objects”键现在会反映已修复的页面树。要查看有未修复页面树错误的原始对象，请单独指定“object”和/或“objectinfo”。这与 JSON v2 的表现是一致的。

- 文档中新增了关于如何贡献 QPDF 的内容。参见贡献于 qpdf。
- qpdf 源代码现自动格式化为 clang-格式。详情请参见代码格式。
- 在开发过程中启用 QTC 测试覆盖，但默认从分布式 qpdf 二进制文件中编译出来。这带来了显著的性能提升，尤其是在 Windows 上。QTC::TC 仍然可以在库中使用，并且即使库内部调用它被关闭，最终用户代码仍然可以使用。内部还增加了缓存功能，以减少运行时反复读取环境变量的开销。
- 仓库顶部 performance\_check 脚本使用的测试文件现已可在 qpdf/performance-test-files 的 github 仓库中获取。除了运行时间外，内存使用情况也会在性能测试结果中被包含在内。performance\_check 工具仅在 Linux 上进行了测试。
- M. Holger 在多个拉取请求中贡献了大量代码清理和重构工作。

这包括用于检测属于被销毁 QPDF 对象的 QPDFObjectHandle 对象所需的工作。

#### 10.6.3: 2022 年 3 月 8 日

- 即将发布的变更公告：
  - QPDF 11 将使用 Cmake 构建。qpdf 11 文档将包含详细的迁移说明。
- 漏洞修复：
  - 识别 PDF 2.0 规范允许的 UTF-8 编码字符串。
  - 通过外观流生成解决表单字段中/DA 字段缺乏正确字体大小指定或指定自动大小的边缘情况。目前，qpdf 不支持自动调整尺寸。
  - 为迁移 cmake 做准备，对构建和文档进行小幅、非功能性修改，以适应更广泛的编译环境。

#### 10.6.2: 2022 年 2 月 16 日

- 漏洞修复：
  - 识别编码为 UTF-16LE 的字符串为 Unicode。PDF 规范只允许 UTF-16BE，但大多数读卡器也接受 UTF16-LE。
  - 修复命令行参数解析中的回归问题，以恢复一些人依赖的此前未被记录的行为。
  - 修复 Unicode 映射到 PDF 文档编码的又一个问题

#### 10.6.1: 2022 年 2 月 11 日

- 修复部分平台上的编译错误

#### 10.6.0: 2022 年 2 月 9 日

- 为更换指针持有者的准备工作

qpdf 的下一个主要版本将用 std::shared\_ptr 替换 PointerHolder，覆盖 qpdf 所有公共 API。目前无需采取任何行动，但如果你想提前准备，请阅读 include/qpdf/PointerHolder.hh 中的评论，并查看 Smart Pointers，了解你现在可以做些什么，如何创建能够继续与旧版 qpdf 兼容、并且在 qpdf 11 发布时更容易切换到的代码。

- 为新的 JSON 输出版本做准备

– `--json` 选项采用一个可选参数，指示 JSON 输出的版本。目前只有一个 JSON 版本（1），但计划在即将发布的版本中更新。

在 qpdf 11 发布之前，`--json` 的默认值为 1 以保证兼容性。一旦 qpdf 11 发布，默认版本将是最新版本。如果你依赖代码的具体格式 `--json`，建议先用 `--json=1` 来准备。

#### • 新的 QPDFJob API 揭示了 CLI 功能

在 qpdf 10.6 之前，qpdf CLI 执行文件实现的许多功能内置于可执行文件本身，库中无法提供。qpdf 10.6 引入了一个新对象 QPDFJob，它暴露了所有命令行功能。这包括原生的 QPDFJob API，具有流畅接口，镜像命令行语法；一个用于指定命令行调用等价物的 JSON 语法，以及通过传递空终端数组 qpdf 命令行选项来运行 qpdf “作业” 的能力。命令行参数数组和调用 QPDFJob 的 JSON 方法也暴露在 C API 中。详情请参见 QPDFJob：基于岗位界面。

#### • 其他图书馆增强

– 新的 QPDFObjectHandle，使用 C++ 用户自定义的文字语法。你可以使用

```
自动 oh = "<>"_qpdf;
```

以创建 QPDFObjectHandle。它是 QPDFObjectHandle::parse 的简写。

– 预处理器符号 QPDF\_MAJOR\_VERSION、QPDF\_MINOR\_VERSION 和 QPDF\_PATCH\_VERSION

现已可用，并可用于简化编写支持多个版本 QPDF 的代码。你不需要添加任何新的头文件来获得这些文件，这使得编写如下代码成为可能：

```
#if ! 定义 (QPDF_MAJOR_VERSION) || QPDF_MAJOR_VERSION < 11
用 QPDF 10 或更早的 API 做点什么 #else
用 QPDF 11 或更新的 API 做一些 #endif
```

由于该技术仅在 qpdf 10.6.0 版本中引入，测试未定义的 QPDF\_MAJOR\_VERSION 值等同于检测到 10.6.0 之前的版本。

符号 QPDF\_VERSION 也定义为包含与 QPDF::QPDFVersion 返回相同版本号的字符串。请注意，如果您的头文件和库不同步，QPDF\_VERSION 可能与 QPDF::QPDFVersion () 不同。

– 方法 QPDF::QPDFVersion 及其对应的 C 语言 API 调用 qpdf\_get\_qpdf\_version 现在都保证返回静态字符串的引用（或指针），因此如果你在软件中使用它们，无需复制它们。它们总是返回静态值。现在它们返回静态值是 API 合同的一部分，可以放心依赖。

– QPDFObjectHandle 的新访问器方法。除了传统的访问程序，如 getIntValue、getName 等，还有一类新访问者，其名称形式为

getValueAsX。行为上的差异如下：

- \* 较早的访问器方法（将继续支持）如果对象是预期类型，则返回其值。否则，他们会返回备用值并发出警告。
- \* 较新的访问者方法返回一个布尔值，表示该对象是否属于预期类型。如果是，则初始化对正确类型变量的引用。

在许多情况下，新接口将使代码更紧凑，且永远不生成类型警告。感谢 M. Holger 贡献这些配件。在 include/qpdf/QPDFObjectHandle.hh 中搜索 getValueAs 可获得完整列表。

这些功能也在 C API 中以 `qpdf_oh_get_value_as` 开头的函数中被暴露。

- QPDF 中的新便捷方法: `isDictionaryOfType`, `isStreamOfType` 和 `isNameAndEquals` 允许对词典进行更紧凑的查询。同时也加入了 C API: `qpdf_oh_is_dictionary_of_type` 和 `qpdf_oh_is_name_and_equals`。感谢 M. Holger 的贡献。
- QPDFObjectHandle 中的新便利方法: `getKeyIfDict` 在调用 `null` 时返回 `null`, 否则调用 `getKey`。这使得访问可选的低层词典变得更容易。它在 C API 的 API `qpdf_oh_get_key_if_dict` 中被公开。感谢 M. Holger 的贡献。
  - QUtil: `make_shared_cstr` 新增函数, `make_unique_cstr` 将 `std::string` 复制到 `std::shared_ptr` 和 `std::unique_ptr`。这些是前一一的替代方案我指的是 QUtil:: `copy_string` 函数, 它提供了其他获得 C 字符串且内存管理更安全的方法。
- 新功能 QUtil:: `file_can_be_opened` 通过尝试打开和关闭文件来测试文件是否真的可以被打开。
- 有新版本的 QUtil:: `call_main_from_wmain`, 它取一个 `const argv` 数组, 调用一个主数组, 主数组取一个 `const argv` 数组。
- QPDF:: `emptyPDF` 已暴露于 C API 中, 如 `qpdf_empty_pdf`。这使得使用 C API 从零开始创建 PDF 成为可能。
- 新的 C API 函数 `qpdf_oh_get_binary_utf8_value`  
`qpdf_oh_new_binary_unicode_string` 采用长度参数, 这使得能够处理带有嵌入 NUL 字符的 UTF-8 编码 C 字符串。感谢 M. Holger 的贡献。
- 新增了一个 `PDFVersion` 类, 用于表示 PDF 版本号, 能够比较和排序 PDF 版本。方法 QPDF: `getVersionAsPDFVersion` 和新版 QPDFWriter:: `setMinimumPDFVersion` 使用。这使得创建输出文件变得更容易, 其 PDF 版本是所有贡献该文件的输入文件中最大值。
- qpdf 库中的 JSON 对象已增强, 包含解析器和从 JSON 对象获取数值的能力。此前它是只写接口。即便如此, qpdf 的 JSON 对象并非像 `include/qpdf/JSON.hh` 中讨论的那样, 旨在成为通用的 JSON 实现。
- JSON 对象的“模式”检查功能现在允许可选键。请注意, 这种“模式”功能并不符合任何标准。它只是为了配合 QPDF 自带的 JSON 支持来帮助错误报告。

#### • 文档增强

- 命令行工具的文档已被完全重写。这包括对手册中运行 qpdf 的从上到下重写。命令行参数现在已被索引, 文档中可以显示内部链接。
- `qpdf -help` 的输出来自手册, 并划分为与手册章节相对应的帮助主题。运行 `qpdf--help` 时, 不会看到一长串文字, 而是提供基本使用信息和帮助主题列表。你可以针对任何单个主题或特定命令行选项请求帮助, 或者获取所有可用帮助文本的导出。

手册内容更丰富, 示例更多。

#### • 漏洞修复

- 部分字符未正确从 PDF 文档编码翻译为 Unicode。
- 在拆分或合并页面时, 确保所有输出文件的 PDF 版本大于或等于所有输入文件的最大版本。

## 10.5.0: 2021 年 12 月 21 日

## • 包装变更

- 预建文档不再随源发行版分发。ApplImage 和 Windows 二进制发行版仍包含嵌入文档，且 qpdf 发布网站提供单独的文档分发文件。相关文件现已可在 <https://qpdf.readthedocs> 查阅。

从 10.5 版本开始，所有主要/次要版本都使用 IO。请参阅包装文档，了解包装商应如何处理文档的详细信息。

- 文档源已从文档手册切换到用 Sphinx 处理的 reStructuredText。这会破坏之前的文档链接。主网站上设有重定向功能。文档的全面审查计划在即将发布时推出。

## • 图书馆增强

——自 qpdf 版本 8 起，在 QPDFObjectHandle 实例上使用对象访问器方法，如果对象类型不符，可能会触发警告。这些警告现在的错误代码从 qpdf\_e\_damaged\_pdf 变成了 qpdf\_e\_object。另外，QPDFObjectHandle.hh 也添加了评论，更详细地解释了这种行为。更深入的讨论请参见对象访问器方法。

- 添加 Pl\_Buffer::getMallocBuffer () 以初始化使用 malloc () 的缓冲区，以提升跨语言互作性。

## • C API 增强

——非常感谢 M. Holger 的贡献，对这些 C API 的改进产生了巨大影响。

他的多项建议、拉取请求、提问以及对文档和注释的批判性阅读，带来了 C API 的显著可用性改进。

- 对对象处理函数的错误处理进行全面重修，C API。一些以前会导致崩溃的罕见错误条件现在被捕获并报告，生成这些错误的函数会返回后备值。详情请参见 include/qpdf/qpdf-c.h 的错误处理部分评论。特别是，当调用对象访问者时，底层 C++ 代码抛出的异常会被捕获并转换为错误。可以通过调用 qpdf\_has\_error 来检查这些错误。使用 qpdf\_silence\_errors 防止错误写入 stderr。

- 在 C API 中添加 qpdf\_get\_last\_string\_length，以获得最后返回字符串的长度。这对于处理包含嵌入空字符串的字符串是必要的。

- 在 C API 中添加 qpdf\_oh\_is\_initialized 和 qpdf\_oh\_new\_uninitialized，使未初始化对象能够使用。

- 向 C API 添加 qpdf\_oh\_new\_object。这允许你克隆一个对象的手柄。

- 添加 qpdf\_get\_object\_by\_id、qpdf\_make\_indirect\_object 和 qpdf\_replace\_object，暴露对应的方法在 QPDF 和 QPDFObjectHandle 中。

- 添加多个用于页面作的功能。详情请参见 include/qpdf/qpdf-c.h 中的页面功能。

- 添加多个用于处理流的函数。详情请参见 include/qpdf/qpdf-c.h 中的流函数。

——加 qpdf\_oh\_get\_type\_code 加 qpdf\_oh\_get\_type\_name。

- 添加 qpdf\_oh\_get\_binary\_string\_value 和 qpdf\_oh\_new\_binary\_string，以便更容易处理包含嵌入空字符串的字符串。

## 10.4.0: 2021 年 11 月 16 日

## • 弱密码算法的处理

- 根据 qpdf CLI, 现在要求在使用 RC4 加密明确创建 PDF 文件时, 抑制警告。虽然 qpdf 始终保留读取和写入此类文件的能力, 但今后需要明确确认。对于 qpdf 10.4, 这一变化仅影响命令行工具。从 qpdf 11 开始, 也会有一些小幅的 API 变更, 要求在这些情况下明确确认。更多信息请参见弱密码学。

- 漏洞修复

- 在处理壳补全时, 修正在给定虚假输入时可能发生的潜在边界错误。
- 正确处理完全空白页面 (无资源词典) 的覆盖层/底层。
- 修复在使用包含表单字段的文件时, 某些条件下可能发生的崩溃。

- 图书馆增强

- 将 QPDF:: findPage 函数公开。
- 向 PL\_Flate 添加方法, 以便能够接收某些可恢复状况的警告。
- 在插入时而非写入文件时, 在库中增加额外检查, 检测是否直接插入了外部对象 (而非使用 QPDF:: copyForeignObject)。更早发现错误会更容易找到错误代码。

- CLI 增强功能

- 改进解析分析的诊断 ---pages 命令行选项

- 包装变更

- Windows 二进制发行版现已采用 OpenSSL 3.0 提供的加密技术构建。

### 10.3.2: 2021 年 5 月 8 日

- 漏洞修复

- 在生成文件并保留对象流时, 除非指定 --preservation-unreferenced, 否则未被引用的对象会被正确移除。

- 图书馆增强

- 添加已存在的页面时, 只需做一个浅层复制, 而不是抛出异常。这使得库的行为与 CLI 的行为一致。更多说明请参见变更日志。

### 10.3.1: 2021 年 3 月 11 日

- 漏洞修复

- 表单字段复制失败, 且 /DR 是文档级表单词典中的直接对象。

### 10.3.0: 2021 年 3 月 4 日

- 漏洞修复

- 在复制 10.2.0 页面时处理表单字段的代码并不完全正确, 在许多情况下无法正常工作, 例如同一页被多次复制或存在资源或字段名称冲突时。10.3.0 代码经过更彻底的测试, 涉及更复杂的案例和众多读者, 应该更接近正确。10.2.0 代码在页面拆分或将带有表单字段的页面复制到尚未有表单字段的文档时表现尚可, 但在处理现场级资源方面仍不完全正确。

- 当调用 QPDF:: replaceObject 或 QPDF:: swapObjects 时, 现有的 QPDFObjectHandle 在站姿不再指向旧物体。下次访问时, 它们会自动注意到底层对象的变化并更新自己。这解决了一个长期存在的困惑, 尽管是在一个极少使用的方法调用中。

- 修正表单字段处理代码，以便在正确位置查找默认外观、四分之一和默认资源。代码并没有在文档级交互式表单词典中寻找它应该在那里找到的内容。这需要在 QPDFFormFieldObjectHelper 中添加一些新方法。

- 图书馆增强

- 重写了处理页面作中复制注释和表单字段的代码。10.2.0 版本为公共 API 增加了一些方法，10.2.0 中也为某个方法进行了一个弃用版本。大多数 API 变更都是大多数人不会调用的方法，希望这些方法能被更高级的页面复制处理接口所取代。详情请参阅变更日志文件。
- 新增了 QPDF::numWarnings 方法，以便你判断在特定代码块内是否发生过警告。

10.2.0: 2021 年 2 月 23 日

- CLI 行为变更

- 用于合并页面的作在保护表单字段方面要好得多。特别是，--split-pages 和 ---pages 现在通过从原始文件复制相关的表单字段信息，保留了交互表单的功能。此外，如果你只从 --pages 中选择原始输入文件中的部分页面，未使用的表单字段会被移除，这会防止保留大量未使用的注释。
- 默认情况下，qpdf 不再允许创建用户密码非空且所有者密码为空的加密 PDF 文件，当使用 256 位密钥时。--allow-insecure 选项在 --encrypt 选项中指定，允许创建此类文件。在可能的情况下，CLI 行为的变更会被避免，但这里因安全相关而作出了例外。QPDF 必须始终允许为测试目的创建奇怪的文件，但不应默认允许用户无意中创建不安全的文件。

- 库行为变化

- 注：本节变更在某些情况下会导致输出差异。这些差异改变了 PDF 的语法，但不改变语义（含义）。我会尽量避免对 qpdf 输出做出无谓更改，以免这些更改影响测试。在这种情况下，这些更改显著提升了生成 PDF 的可读性，且不影响通过简单转换产生的输出。如果你对必须更新测试文件感到烦恼，请放心，这类变动一直以来都很罕见，也将继续如此。
- QPDFObjectHandle::newUnicodeString 现在使用 ASCII、PDFDocEncoding 或 UTF-16 中，任何一个足以编码字符串中所有字符的格式。这减少了可以用 ASCII 编码字符串的 UTF-16 不必要的编码。当用 QPDFFormFieldObjectHelper 设置表单字段值时，qpdf 可能会生成与之前不同的输出，但不会改变输出的含义。
- 将形式 XObject 放置的代码，同时也是将旋转、修整、从实数中修正尾随零的代码。这会导致生成的外观流和表单在覆盖/底层代码中或调用将 XObject 形式置入页面的方法时存在细微（但语义等效）差异。

- CLI 增强功能

- 新增命令行选项，用于列出、保存、添加、删除和复制文件附件。  
详情请参见嵌入文件/附件。
- 页面分割和合并作，以及 --flatten-rotation 在注释和交互式表单字段方面表现更好。在大多数情况下，交互式表单字段的功能以及正确的格式化和注释功能都通过这些作得以保留。现在还有

有些情况并不完美，比如注释的功能依赖于 QPDF 尚未理解的文档级数据，或者表单字段与注释之间的引用完整性存在问题（例如，单一表单字段对象或其关联注释在多个页面上共享，这种情况超出规范，但大多数浏览器中都能使用）。

- 选项 `--password-file=filename` 现在可以用来从文件中读取解密密码。你可以用 `-` 作为文件名，从标准输入中读取密码。这比用 `@file` 来读取文件或标准输入密码更简单、更直观。
- 在 JSON 输出中添加一些附件信息，并将附件作为额外的 JSON 键添加。这里包含的信息仅限于首选名称和内容流以及对文件规范对象的引用。这些细节足以让客户避免浏览名称树的麻烦，并提供了基本的枚举和附件提取所需的信息。通过参考文件规格对象，可以获得更详细的信息。
- 添加数值选项 `---collate`。如果给定了 `--collate=n`，则从给定文件中取 `n` 个为一组的页面。

- 现在提供 `--rotate=0` 以清除页面旋转是有效的。

#### • 图书馆增强

- 本版本包含了对 API 的多项新增内容。并非所有变动都在这里列出。请参阅源发行版中的 `ChangeLog` 文件，获取详尽列表。以下是重点内容。
- 添加 `QPDFObjectHandle::d_items()` 和 `QPDFObjectHandle::aitems()`，使 C++ 风格成为可能迭代，包括范围对迭代，基于字典和数组 `QPDFObjectHandles`。详情请参见 `include/qpdf/QPDFObjectHandle.hh` 和 `examples/pdf-name-number-tree.cc` 中的评论。
- 添加 `QPDFObjectHandle::copyStream`，用于在同一 QPDF 实例内复制流。
- 添加新的辅助类以支持文件附件，也称为嵌入文件。

新增的类包括 `QPDFEmbeddedFileDocumentHelper`、`QPDFFileSpecObjectHelper` 和 `QPDFEFStreamObjectHelper`。详情和示例 `pdf-attach-file.cc` 请参见各自的标题。

- 添加一个 `QPDFObjectHandle::parse` 版本，该版本将 QPDF 指针作为上下文，以便解析包含间接对象引用的字符串。这一点在 `examples/pdf-attach-file.cc` 中有说明。
- 重新实现 `QPDFNameTreeObjectHelper` 和 `QPDFNumberTreeObjectHelper`，使其更高效- 熟练地添加基于迭代器的 API，赋予他们修复损坏树木的能力，并创建修改树的方法。通过这一变化，qpdf 实现了强大的名称树和数字树读写实现。
- 添加新的 `QPDFObjectHandle::replaceStreamData`，这些版本可取 `std::function` 对象，适用于需要介于静态字符串和完整 `StreamDataProvider` 之间的物件。  
结合 `QUtil::file_provider` 是从文件内容创建流的非常简单方法。
- `QPDFMatrix` 类，原为私有内部类，已被添加到公共 API。详情请参见 `include/qpdf/QPDFMatrix.hh`。本课程主要用于处理变换矩阵。  
`QPDFPageObjectHelper` 中的某些方法利用这一点来提供关于变换矩阵的信息。示例请参见 `examples/pdf-overlay-page.cc`。
- `QPDFAcroFormDocumentHelper` 新增了多种方法，用于添加、删除、获取表单字段信息和枚举。
- `Add` 方法 `QPDFAcroFormDocumentHelper::transformAnnotations`，该方法对页面上的每个注释进行转换。



- 添加 QPDFPageObjectHelper::copyAnnotations, 它将注释及相关表单字段从一页复制到另一页, 可能变换矩形。

- 构建变更

- 构建 qpdf 现在需要 C++-14 编译器。暂时没有打算要求比这更新的设备。C++-14 对 C++-11 进行了适度增强, 支持范围似乎与 C++-11 相当。

- 漏洞修复

- --flatten-rotation 选项对页面上可能存在的任何注释应用变换。
- 如果 XObject 形式缺少资源字典, 则将该形式 XObject 中的任何名称都从包含的页面引用。这符合较旧的 PDF 版本。还要检测任何形式的 XObjects 是否有未解析名称, 如果有, 不要从这些名称或包含这些名称的页面中删除未引用资源。不幸的是, 这也会导致在某些情况下, 当出现与资源无关联的名称时, 比如带有标签的 PDF, 无法删除未引用的资源。这只是一个角落情况, 实际上不太可能引发重大问题, 但唯一的副作用是共享资源的移除。未来的 qpdf 版本可能会在检测涉及资源的名称方面更加复杂。

- 在外部化内联图像时, 如果字符串出现在内联图像词典中, 请正确处理。

10.1.0: 2021 年 1 月 5 日

- CLI 增强功能

- 添加 --flatten-rotation 命令行选项, 使所有使用页面词典参数旋转的页面在页面内容中被完全相同的旋转。该更改对合规的 PDF 阅读器用户无法直观看到, 但可用于解决无法正确处理页面旋转的 PDF 应用程序。

- 图书馆增强

- 支持用户提供的 (可插拔、模块化) 流过滤。现在可以从 QPDFStreamFilter 派生类并注册 QPDF, 使得包括 QPDFWriter 使用的常规库方法能解码使用库不直接支持的滤波器流。示例示例/pdf-custom-filter.cc 展示了如何使用这一功能。
- 在 QPDFPageObjectHelper 中添加方法, 以遍历页面上的 XObject 或形式 XObject, 可能递归为嵌套形式 XObject: forEachXObject、ForEachImage、forEachFormXObject。
- 如评论中所述, 增强 QPDFPageObjectHelper 中的多种方法, 使其能够处理表单 XObject 和页面。完整列表请参见 ChangeLog。

- 在 QPDFPageObjectHelper 中重命名部分函数, 同时保留旧名称以保证兼容性:

- \* getPageImages 以获取图片
- \* filterPageContents 到 filterContents
- \* pipePageContents 到 pipeContents
- \* parsePageContents to parseContents

- Add 方法 QPDFPageObjectHelper::getFormXObjects, 直接返回页面或 XObject 表单上的 XObject 格式映射

- 向 QPDFObjectHandle 添加新的辅助方法: isFormXObject, isImage

- 添加可选的 `allow_streams` 参数 `QPDFObjectHandle::makeDirect`。当 `QPDFObjectHandle::makeDirect` 就是这样调用的，它保留了对流的引用，而不是抛出异常。
- 添加 `QPDFObjectHandle::setFilterOnWrite` 方法。在流中调用此功能可以防止 `QPDFWriter` 尝试解压、重新压缩或以其他方式过滤流，即使它能做到。  
开发者可以利用它来保护那些经过优化的流，避免因其他原因受到 `QPDFWriter` 默认行为的影响。
- 为 `QPDFObjGen` 添加 `ostream` 的 `<<` 算符。这对调试非常有用。
- 添加方法 `QPDFPageObjectHelper::flattenRotation`，替换页面的 `/Rotate` 键-  
通过在内容流中旋转页面并修改页面边界框，使渲染保持一致。这可以用来绕过那些无法正常转页的 PDF 阅读器。
- C API 增强
  - 在 C API 中添加若干用于处理对象的新函数。这些是 `QPDFObjectHandle` 中许多方法的封装器。它们的加入为 C API 带来了相当多的新能力。
  - 向 C API 添加 `qpdf_register_progress_reporter`，对应 `QPDFWriter::registerProgressReporter`。
- 性能提升
  - 改进 `QPDFWriter` 准备 QPDF 对象的步骤，使写入性能提升约 8%，同时允许间接对象出现在 `/DecodeParms` 中。
  - 在提取页面时，`qpdf CLI` 只从保留的页面中移除未引用资源，从而在从大型复杂文档中提取少量页面时显著提升性能。
- 漏洞修复
  - `QPDFPageObjectHelper::externalizeInlinedImages` 未外部化来自页面上出现的 `XObject` 表单中的图片。
  - `QPDFObjectHandle::filterPageContents` 在拥有多个内容流的页面时被破坏。
  - 调整 `zsh` 补全代码，使其在路径补全方面表现更好。

#### 10.0.4: 2020 年 11 月 21 日

- 漏洞修复
  - 修正几个整数溢出。这包括通过模糊检测发现的情况，以及 `qpdf` 不对 `xref` 流中未使用的值进行范围检查的情况。

#### 10.0.3: 2020 年 10 月 31 日

- 漏洞修复
  - 关于带有间接过滤的复制流的漏洞修复错误，并引入了一个更严重的新漏洞。原始的 bug 已经正确修复，10.0.2 中引入的 bug 也已修复。

#### 10.0.2: 2020 年 10 月 27 日

- 漏洞修复
  - 在串接内容流时，如 `--coalesce-contents`，有时 `qpdf` 会合并两个词汇标记，产生无效结果。如果合并内容流中还没有换行，则会插入换行。

- 修复在复制使用流数据提供者替换的外部流时可能发生的内部错误，前提是这些流具有间接过滤器或解码参数。这是一个罕见的角落案例。
  - 确保调用方的本地设置不会影响 qpdf 库内部进行的数值转换结果。请注意，这里的问题只有在 qpdf 库被编程使用时才会出现。使用 qpdf 时，CLI 已经忽略了用户的数字转换所在地。
  - 修正若干警告未被抑制的实例——无警告和/或错误或警告写入标准输出而非标准错误。
  - 修复了在特定情况下使用 `--object-streams=generate` 可能发生的内存泄漏。
- 
- 修复 OSS-Fuzz 项目发现的各种整数溢出及类似条件。
- 增强功能
    - 新选项 `--warning-exit-0` 使得 qpdf 退出状态由 0 而非 3 变为，前提是有警告但无错误。结合 `--no-warn` 可以完全忽略警告。
    - 对 QPDF: `:p rocessMemoryFile` 进行了性能改进。
    - OpenSSL 加密提供商会生成更详细的错误信息。
  - 构建变更
    - qpdf 的 `./configure` 脚本现在支持 `--disable-rpath` 选项。一些发行版的包装标准建议使用此选项。
    - 长期以来，`printf` 格式字符串的选择已从 `IFDEFS` 转向自动配置测试。如果你用的是自己的构建系统，你需要在 `libqpdf/qpdf/qpdf-config.h` 中为 `LL_FMT` 提供一个值，通常是 `“%lld”`，或者在某些 Windows 编译器中是 `“%I64d”`。
    - 对 OpenSSL 加密提供商的构建时配置进行了多项改进。
    - qpdf 版本现已包含一个几乎独立的 Linux 二进制压缩文件。它基于较旧（但支持的）Ubuntu LTS 版本，但适用于大多数较新的 Linux 发行版。它只包含可执行文件和必要的共享库，这些在最小系统中是不存在的。它可以用于在最小环境中包含 qpdf，比如 docker 容器。该压缩文件也被认为可以作为 AWS Lambda 中的一层使用。
    - qpdf 的自动化构建已从 Azure Pipelines 迁移到 GitHub Actions。
  - Windows 专属变更
    - 随 qpdf 发布的 Windows 可执行文件默认使用 OpenSSL 加密提供者。本地加密提供者也被编译进去，运行时可通过 `QPDF_CRYPTO_PROVIDER` 环境变量选择。
    - 在原生 Windows 加密实现中，密码学提供商的获取方式得到了改进。不过，大多数情况下，OpenSSL 默认使用，掩盖了这一点。

#### 10.0.1: 2020 年 4 月 9 日

- 漏洞修复
  - 10.0.0 引入了一个漏洞，即调用无法过滤的流的 `QPDFObjectHandle::getStreamData` 时，会返回原始数据而非抛出异常。现在这个问题已经解决了。
  - 修复一个导致 qpdf 无法与某些平台某些版本的 clang 链接的 bug。
- 增强功能

- 改进 pdf-invert-images 示例，避免所有图像同时加载到内存中。

10.0.0: 2020 年 4 月 6 日

- 性能提升

- qpdf 库和可执行文件在本版本中运行速度应比过去几次版本快得多。  
已经进行了多项内部库优化，页面分割的行为也有所改进。该版本的 qpdf 应能优于任何 8.x 或 9.x 版本。

- 不兼容的 API（源代码级）变更（小幅）

- QUtil::srandom 方法被移除。除非编入了不安全的随机数，否则它什么都没用，而且这些随机数默认已经关闭很久了。如果你打电话，直接删掉电话，反正也没什么用。

- 组装/包装变更

- 添加一个 openssl 加密提供商，该提供商使用 OpenSSL，同时也与 BoringSSL 兼容。  
感谢 Dean Scarff 的贡献。如果你为发行维护 qpdf，特别注意确保你支持你想要的加密服务商。软件包维护者必须权衡允许用户在运行时选择加密提供商的优势与增加更多依赖的缺点。

- 允许 qpdf 构建在 C/C++ 库缺少 wchar\_t 类型的精简系统上。详情请见 QPDF 的 README.md 中搜索 wchar\_t。这应该非常罕见，但在某些嵌入式环境中已知是有用的。

- CLI 增强功能

- 在 JSON 输出中添加 objectinfo 键。这里将是一个放置计算出的元数据或其他关于 PDF 对象的信息的地方，这些信息在其他方面不明显，或者因其他原因看起来有用。在这个版本中，会提供每个对象的信息，比如它是否是流，如果是，以及它的长度和过滤器。没有这些，仅凭 JSON 输出无法确定一个对象是否是流。请运行 qpdf --json-help 获取详细信息。
- 新增选项 --remove-unreferenced-resources，该选项将自动、是或否作为参数。

新的自动模式是默认的，它在分割页面时会对 PDF 文件进行快速启发式分析，以判断寻找和删除未引用资源的昂贵过程是否可能带来好处。对于大多数文件来说，这种新的默认设置将显著提升页面拆分性能。

- --preserve-unreferenced-resources 现在只是 --remove-unreferenced-resources=no 的同义词。

- 如果在调用 qpdf --bash-completion 或 qpdf --zsh-completion 时设置了 QPDF\_EXECUTABLE 环境变量，其输出的补全命令将使用该变量的值来指代 qpdf，而不是 qpdf 确定其可执行路径的值。这在用脚本包裹 qpdf、处理源树中的版本、使用 ApplImage 或其他需要间接处理的情况时非常有用。

- 图书馆增强

——随机数生成现由加密提供商负责。旧的做法仍然被本地加密提供商使用。你仍然可以提供自己的随机数生成器。

- 添加一个新版本的 QPDFObjectHandle::StreamDataProvider::provideStreamData，该版本接受 suppress\_warnings 和 will\_retry 选项，并允许返回成功码。

这使得实现调用 pipeStreamData 的 StreamDataProvider 成为可能

流，并将响应反馈回呼叫者，从而使这些代理流的错误处理更为高效。

- 更新 QPDFObjectHandle::pipeStreamData，返回一个超越过滤数据是否成功写入的整体成功代码。这使得对未过滤错误的案例处理得更好。你必须明确地称呼这个。已有 API 中的方法语义与之前相同。
- QPDFPageObjectHelper::placeFormXObject 方法现在允许单独控制是否愿意缩小或展开对象以更好地适应目标矩形。之前的做法是允许缩小但不允许扩张。之前的行为仍然是默认的。
- 调用 C API 时，任何传递给布尔参数的非零值都被视为 TRUE。之前只接受值 1。这使得 C API 的行为更接近大多数 C 接口，并且已知能提升与某些动态加载 DLL 并调用其函数的 Windows 环境的兼容性。
- 仅用于复制顶级字典键或数组项，添加 QPDFObjectHandle::unsafeShallowCopy。这不安全，因为会造成一个对象中较低级别的项变化也可能在另一个对象中改变，但对于你知道只插入或替换顶层项的情况，它比 QPDFObjectHandle::shallowCopy 快得多。
- 添加 QPDFObjectHandle::filterAsContents，将流的数据过滤为内容流。

这对于解析 XObject 形式的内容非常有用，就像解析页面内容流一样。

#### • 漏洞修复

- 在分页过程中检测并删除未引用资源时，穿越到 XObjects 形式，并处理其资源字典。
- 合并或拆分页面时，错误恢复同样适用于主输入文件以外的流。

### 9.1.1: 2020 年 1 月 26 日

#### • 组装/包装变更

- fix-qdf 程序从 perl 转换为 C++。因此，qpdf 不再依赖 perl 运行时。

#### • 图书馆增强

- 新增辅助例程 QUtil::call\_main\_from\_wmain，将 wchar\_t 参数转换为 UTF-8 编码字符串。这对 qpdf 很有用，因为库方法要求文件名采用 UTF-8 编码，即使在 Windows 上也是如此
- 新增了 QUtil::read\_lines\_from\_file 方法，该方法接受 FILE\* 参数，并允许保留行尾字符。这也修复了一个 QUtil::read\_lines\_from\_file 无法正常处理 Unicode 文件名的漏洞。

#### • CLI 增强功能

- 增加了选项 --is-encrypted 和 --demand-password，用于测试文件是否加密或是否需要提供（或空）密码以外的密码。它们通过退出状态进行通信，因此对 shell 脚本非常有用。它们还能处理密码未知的加密文件。
- 为 JSON 选项添加了加密密钥。除了旧加密格式的重构用户密码外，这提供了与 ---show-encryption 相同的信息，但格式一致且可解析。详情请参见 qpdf --json-help 的输出。

#### • 漏洞修复

- 在 QDF 模式下，确保不要为一个文件写入超过一条 XRef 流，即使使用了 `--preserve-unreferenced`。fix-qdf 假设只有一个 XRef 流，并且它出现在文件的末尾。
- 在外部化内联图像时，应正确处理色彩空间引用到页面资源词典中对象的图像。

- Windows 专用修复，用于通过新密钥集获取加密上下文。

#### 9.1.0: 2019 年 11 月 17 日

##### • 构建变更

- 现在构建 qpdf 需要 C++-11 编译器。
- 一款使用 gnutls 进行加密功能的新型加密服务提供商现已可用，且可在构建时启用。有关加密提供商的更多信息，请参见“加密提供商”，以及关于构建的具体信息，请参见“构建期加密选择”。

##### • 图书馆增强

- 纳入细田正道的贡献，通过不将签名字典包含在对象流中，将 Contents 键格式化为十六进制字符串，并将 /Contents 键排除在加密和解密中，以正确处理签名字典。
- 纳入细田正道的贡献，提供新的 API 调用，用于获取输入和输出文件的文件级信息，从而支持文件层级而非对象层的某些作。新冰毒——

ods 包括 QPDF::getXRefTable ()、QPDFObjectHandle::getParsedOffset ()、QPDFWriter::getRenumberedObjGen (QPDFObjGen) 和 QPDFWriter::getWrittenXRefTable ()。

- 支持构建时和运行时可选的加密提供者。这包括新增的职业 QPDFCryptoProvider 和 QPDFCryptoImpl 以及 QPDF\_CRYPTO\_PROVIDER 环境变量的识别。加密服务提供商在《加密服务提供商》中有详细介绍。

##### • CLI 增强功能

- 新增了支持可选加密提供商的展示加密选项，详见加密提供商。
- 允许在数字范围中附加：偶数或奇数，以便从指定页面中指定偶数或奇数页。

- 修复构建我的 MSVC 时 qpdf.exe 壳万用符扩展行为 (\*和?)。

#### 9.0.2: 2019 年 10 月 12 日

##### • 修复错误

- 修正 `--replace-input` 使用的临时文件名称，使其无需路径分割，且可兼容包含路径的目录。

#### 9.0.1: 2019 年 9 月 20 日

##### • 漏洞修复/增强

- 修复大端系统和默认无符号字符编译器中的构建和测试问题。问题主要发生在构建和测试阶段。qpdf 库本身没有涉及元序性或无符号字符的实际错误。  
—— 当词典有重复的密钥时，请带警告报告。此时库的行为未变，但错误条件不再被无声忽略。

——当表单字段的显示矩形错误地用倒置坐标指定时，检测并纠正此情况。这样可以避免在文件中平整注释时，某些表单字段被翻转。

9.0.0: 2019 年 8 月 31 日

- 不兼容的 API（源代码级）变更（小幅）

- 方法 `QUtil::strcasecmp` 已更名为 `QUtil::str_compare_nocase`。这在——对于使 QPDF 能够在定义 `strcasecmp` 为宏的平台上构建，必须进行兼容的更改。

- `QPDF::copyForeignObject` 方法有一个超载版本，使用了一个未被使用的布尔参数。如果你用的是这个版本，直接省略额外的参数。

- 曾有一个版本 `QPDFTokenizer::expectInlinelImage`，不接受任何参数。该版本已被移除，因为它导致分词器返回错误的内联图片。之前新增了一个版本，能输出正确的输出。这是一个非常低层次的方法，在 qpdf 词汇引擎之外调用并不合理。有更高级的方法用于内容流的代币化。

- 将 `QPDFOutlineDocumentHelper::getTopLevelOutlines` 和 `QPDFOutlineObjectHelper::getKids` 改为返回 `std::vector`，而不是 `QPDFOutlineObjectHelper` 对象列表。

- 移除方法 `QPDFTokenizer::allowPoundAnywhereInName`。这个函数允许创建名称标记，其值在未解析时会变化，但这绝非正确的行为。

- CLI 增强功能

- `--replace-input` 选项可以代替输出文件名。这会导致 qpdf 用输出覆盖输入文件。详情请参见 `--replace-input` 的描述。

- `--recompress-flate` 指示 qpdf 重新压缩已用 `/FlateDecode` 压缩的流。对压缩级别很有用。

- `--compression-level=level` 设定了 `/FlateDecode` 压缩流所使用的 `zlib` 压缩级别。与-再压缩-平整结合效果最佳。

- 图书馆增强

- qpdf/QIntC.hh 提供的全新命名空间 `QIntC`，提供不同整数类型间安全的转换方法。这些转换方法会进行距离检查，以确保铸造过程不会丢失信息。图书馆中每次使用 `static_cast` 都会被检查是否能使用这些安全转换器。更多详情请参见选角政策。

- 方法 `QPDF::anyWarnings` 显示是否有警告，但未清除警告列表。

- 方法 `QPDF::closeInputSource` 关闭或以其他方式释放输入源。这使得输入文件可以被删除或重命名。

- `QUtil` 新增了字符串与无符号整数之间转换的新方法：`uint_to_string`、`uint_to_string_base`、`string_to_uint` 和 `string_to_ull`。

- `QPDFObjectHandle` 新增了返回整数对象值（`int`）或无符号（`uint`）的值，并带有范围检查和合理的返回值，同时新增了返回无符号值的方法。这使得编写避免意外数据丢失的代码变得更容易。

函数：`getUIntValue`，`getIntValueAsInt`，`getUIntValueAsUInt`。

- 当用 `QPDFObjectHandle::ParserCallbacks` 来解析内容流时，开发者可以覆盖 `handleObject`（`QPDFObjectHandle`）

`handleObject`（`QPDFObjectHandle`，`size_t` 偏移量，`size_t` 长度）。如果是这种方法

定义后，它将与对象及其偏移量和长度一起调用，包含在被解析的整体内容中。中间的空格和注释不包含在偏移量和长度中。此外，还可以实现新的 `contentSize (size_t)` 方法。如果存在，则在首次调用 `handleObject` 之前，会调用合并内容的总大小（字节数）。

- 新方法 `QPDF::userPasswordMatched` 和 `QPDF::ownerPasswordMatched` 已出现  
新增后，呼叫者可以判断所提供的密码是用户密码、所有者密码，还是两者兼有。这些信息也通过 `qpdf --显示加密和 qpdf--检查显示`。
- 静态方法 `PL_Flate::setCompressionLevel` 可以调用，以全局设置所有在放气模式下 `PL_Flate` 实例使用的 `zlib` 压缩级别。
- 方法 `QPDFWriter::setRecompressFlate` 可以调用，告诉 `QPDFWriter` 解压并重新压缩已用/`FlateDecode` 压缩的流。
- `qpdf` 数组的底层实现得到了增强，在处理大量空值的数组时，内存效率大幅提升。这使得 `qpdf` 在某些类型文件中能大幅减少内存需求。
- 在遍历页面树时，如果遇到无效类型的节点，类型会被固定，并发出警告。
- 新增了辅助方法 `QUtil::read_file_into_memory`。
- `QPDF::checkLinearization ()` 之前报告的所有错误条件现以警告形式呈现。
- 包含#字符且前无两个十六进制数字的名称标记（在 PDF 1.2 及以上版本中无效）由库正确处理：会生成警告，且名称标记在输出中被正确保留，即使无效。有关此次变更的更完整描述，请参见变更日志。

#### • 漏洞修复

- 少量内存问题、断言失败和未处理异常，这些问题可能发生在严重损坏的输入文件上，已被修复。这些问题大多由谷歌的 `OSS-Fuzz` 项目发现。
- 当 `qpdf --check` 或 `qpdf --check` 线性化遇到带有线性化警告但无错误的文件时，它现在正确地以退出代码 3 而非 2 退出。
- `--completion-bash` 和 `--completion-zsh` 选项在调用 `qpdf` 作为 `ApplImage` 时现在能正常工作。
- 调用尚未设置输出文件名的 `QPDFWriter` 对象的 `QPDFWriter::set*EncryptionParameters`，不再产生分段错误。
- 读取加密文件时，应更严格遵循加密密钥长度的规范。这使得 `qpdf` 在大多数情况下，当加密字典中/`Length` 密钥无效或缺失时，可以打开加密文件。

#### • 构建变更

——在支持它的平台上，`qpdf` 现在构建时设置为 `-fvisibility=hidden`。如果你用自己的构建系统构建 `qpdf`，现在就可以安全使用了。这防止了非公共 API 的方法被共享库导出，使 `qpdf` 的 ELF 共享库（用于 Linux、MacOS 及大多数 UNIX 版本）表现得更像 Windows DLL。由于 DLL 本身就有类似表现，不太可能出现意外未导出的方法。

然而，对于 ELF 共享库，某些类的类型信息必须显式导出。如果动态链接代码在捕捉异常或子类化时存在问题，这可能是原因。

如果你看到这条，请在 <https://github.com/qpdf/qpdf/issues/> 举报一个 bug。



- QPDF 现已编译为启用整数转换和符号转换警告。为了确保安全，图书馆进行了多项改造。
- QPDF 的 Make Install Target 明确指定安装文件时使用的模式，而非依赖用户的 UMASK。之前有些文件会这样，但有些文件没有。
- 如果有 pkg-config，可以用它来定位 libjpeg 和 zlib 的依赖，如果失败则回退到旧的行为。

- 其他注释

- qpdf 已被完全集成进谷歌的 OSS-Fuzz 项目。该项目通过随机变异输入练习代码，非常适合发现隐藏的安全崩溃和安全问题。OSS-fuzz 发现的几个漏洞已经在 qpdf 中修复。

#### 8.4.2: 2019 年 5 月 18 日

本版本仅有一项更改：修正用于打开文件的 Windows 代码中的缓冲区溢出。Windows 用户应该接受这次更新。没有任何代码变更会影响非 Windows 版本。

#### 8.4.1: 2019 年 4 月 27 日

- 增强功能

- 当运行 qpdf 版本时，它会检测 qpdf 的 CLI 是否使用了与库不同版本的 qpdf，这可能表明安装存在问题。
  - 新增选项 `--remove-page-labels` 会在生成输出前移除页面标签。以前如果你运行 `qpdf --empty --pages ...`，但行为在 QPDF 8.3.0 中发生了变化。这个选项让依赖旧行为的人重新获得它。
  - 新选项 `--keep-files-open-threshold=count` 可用于覆盖 qpdf 将使用的文件数量，从而触发合并文件时不保持所有文件打开的行为。如果你的系统允许同时打开少于默认的 200 个文件，这可能是必要的。

- 漏洞修复

- 在 Windows 上处理文件名中的 Unicode 字符。Windows 中对 CLI 支持 Unicode 的更改导致 Windows 的 Unicode 文件名失效。
  - 稍微收紧判断物体是否为页面的逻辑。这应当解决一些罕见文件中某些非页面对象通过 qpdf 测试时，导致页面拆分作中错误丢失的问题。
  - 恢复包含保留-split-pages 中大纲（书签）的更改。8.3.0 和 8.4.0 版本的实现方式导致某些文件分离时性能大幅下降。未来的 qpdf 版本可能会以更高性能、更准确的方式重新引入该行为。
  - 在 JSON 模式下，即使输入中不存在，也将缺失的前导 0 加到-1 到 1 之间的十进制值。JSON 规范要求前置 0。PDF 规范则没有。

#### 8.4.0: 2019 年 2 月 1 日

- 命令行增强功能

- 不兼容的 CLI 变更：qpdf 命令行工具对命令行输入的密码解释方式与之前版本不同，尤其是当密码包含非 ASCII 字符时。在某些情况下，行为与之前版本有所不同。关于当前行为的讨论，请参见

Unicode 密码。不兼容点如下：

- \* 在 Windows 上，qpdf 现在如果能找到合适的编译/链接选项，所有命令行选项都会以 Unicode 字符串的形式接收。至少在 MSVC 和 mingw 构建中，这个功能是启用的。这意味着如果在 Windows 中将非 ASCII 字符串传递给 qpdf CLI 时，qpdf 现在将正确

接收它们。过去，它们要么被编码为 Windows 代码页 1252（也称为“Windows ANSI”），要么编码为无法理解的内容。在几乎所有情况下，qpdf 现在都能正确解释 Unicode 参数，而过去几乎无法正确解释。结果是，Windows 上非 ASCII 密码分配给 qpdf CLI 的密码，现在更有可能生成可被多种读者打开的 PDF 文件。过去，通常使用非 ASCII 密码从 Windows 命令组加密的文件，大多数查看者无法读取。请注意，当前版本的 qpdf 能够解密之前用之前提供的密码创建的文件。

\* PDF 规范要求密码编码为 UTF-8 以实现 256 位加密，使用 PDF Doc 编码以实现 40 位或 128 位加密。旧版 qpdf 则由用户自行提供正确编码的密码。qpdf CLI 现在检测到带有 UTF-8 编码的密码，并自动转码为 PDF 规范要求的密码。虽然这几乎总是正确的行为，但如果原因，也可以覆盖该行为。这在《Unicode 密码》中有更深入的讨论。

- 新增选项 `--externalize-inline-images`、`--ii-min-bytes` 和 `--keep-inline-images`  
控制 QPDF 对内联图像的处理及其可能转换为普通图像。默认情况下，`--optimize-images` 现在也适用于内联图片。
- 添加选项 `--overlay` 和 `--underlay`，用于叠加或底层叠加其他文件的页面到输出页上。详情请参见叠加层和底层。
- 当打开带有密码的加密文件时，如果指定的密码无效且密码中包含任何非 ASCII 字符，qpdf 会尝试多种替代密码以弥补可能的字符编码错误。这种行为可以通过 `--suppress-password-recovery` 选项来抑制。完整讨论请参见 Unicode 密码。
- 添加 `--password-mode` 选项，以微调 qpdf 对密码参数的解释，尤其是当密码参数包含非 ASCII 字符时。更多信息请参见 Unicode 密码。
- 在 `--pages` 选项中，现在可以从同一文件复制同一页面多次，而无需使用之前指定同一文件两条不同路径的变通方法。
- 在 `--pages` 选项中，允许使用“.”作为主输入文件的快捷方式。这样，你可以用 qpdf 来做 `---pages in.pdf. 1-2—out.pdf`，不用在命令里重复 `in.pdf`。
- 在使用 128 位和 256 位加密时，新增的加密选项——`assemble`、`--annotate`、`--form` 和 `--modify-other` 允许配置更细粒度。之前，`--modify` 选项只配置某些预定义的权限组。

#### • 漏洞修复与增强

- 潜在的数据丢失漏洞：8.1.0 至 8.3.0 版本的 qpdf 存在一个漏洞，如果 PDF 文件内部结构共享这些资源列表，且输出中的部分页面未引用所有字体和图像，可能会导致页面拆分和合并丢失部分字体或图像资源。使用 `--preserve-unreferenced-resources` 选项可以绕过错误的行为。该漏洞源于代码中的一个拼写错误和测试套件的缺陷。触发错误的案件是已知的，只是处理不当。该案例现已在 qpdf 的测试套件中进行并妥善处理。
- 在优化图像时，检测并拒绝优化因位深或色彩空间限制无法转换为 JPEG 的图像。
- 线性化和页面作 API 现在能够检测并恢复页面树中存在重复页面对象的文件。
- 使用旧选项 `--stream-data=` 压缩对象流，对象流和 xref 流均未被压缩。

- 当分词器返回内联图像标记时，ID 和 EI 运算符后的分隔符不再被排除。这使得能够可靠地提取实际图像数据成为可能。

- 图书馆增强

- 添加方法 `QPDFPageObjectHelper::externalizeInlinelImages`，将内联图片转换为普通图片。
- 添加方法 `QUtil::possible_repaired_encodings()` 生成字符串列表，表示该字符串可能被编码的其他方式。这就是 qpdf CLI 在恢复错误编码的 Unicode 密码时用来生成尝试字符串的方法。
- 新增 `QPDFWriter::setR{3,4,5,6}EncryptionParameters`，允许更细致地设置权限位。详情请参见 `QPDFWriter.hh`。
- 将 UTF-8 转码器的新版本添加到 `QUtil` 中报告成功或失败的单字节编码系统，而非仅替换指定的未知字符。
- 添加方法 `QUtil::analyze_encoding()` 以判断字符串是否具有高位字符，并且看起来是 UTF-16 编码还是有效的 UTF-8 编码。
- 添加新方法 `QPDFPageObjectHelper::shallowCopyPage()` 以复制一个“浅层复制”页面的新页面。最终生成的对象是一个间接对象，可以传递给 `QPDFPageDocumentHelper::addPage()`，无论是原始 QPDF 对象还是其他对象。这就是 qpdf 命令行工具在拆分和合并作中多次从同一文件复制同一页面的方法。
- 添加方法 `QPDF::getUniqueId()`，该方法返回给定 QPDF 对象的唯一标识符。

该标识符在整个应用生命周期中都是唯一的。返回的值可以安全地用作映射键。

- 添加方法 `QPDF::setImmediateCopyFrom`。这进一步增强了 qpdf 允许被复制对象的 QPDF 对象在目标对象写入前就已超出作用域的能力。如果你在 QPDF 实例上调用此方法，从该实例复制的对象会立即被复制，而不是懒惰地复制。该选项占用更多内存，但允许源对象在目标对象写入前先出作用域。详情请参见 `QPDF.hh` 的评论。
- 添加方法 `QPDFPageObjectHelper::getAttribute`，用于从页面词典中获取属性，考虑继承情况，并可选择复制该属性，前提是你打算修改该属性。
- 修复 `QPDFPageObjectHelper::getPageImages` 长期存在的限制，使其现在能够正确报告继承资源词典中的图片，从而消除了在此情况下调用 `QPDFPageDocumentHelper::pushInheritedAttributesToPage` 的必要。
- 添加方法 `QPDFObjectHandle::getUniqueResourceName`，用于在资源词典中查找未使用的名称。

- Add 方法 `QPDFPageObjectHelper::getFormXObjectForPage` 生成表单 `XObjectForPage` 相当于一页。生成的对象可以在同一文件中使用，也可以通过 `copyForeignObject` 复制到另一个文件中。这对于实现底层、叠加、n-up、缩略图或其他需要在其他上下文中复制页面的功能非常有用。

- Add 方法 `QPDFPageObjectHelper::placeFormXObject`，用于生成内容流文本，将给定表单 `XObject` 置中并置中并嵌入指定矩形内。该方法负责计算正确的变换矩阵，并可选择性地补偿目标页面的旋转或缩放。

- `QPDFJob::run()` 返回的退出码，C API 封装器现在在 `qpdf_exit_code_e` 类型中定义为 `qpdf/Constants.h`，因此可从 C API 访问。它们之前只定义为 `qpdf/QPDFJob.hh` 中的常数。

- 建筑改进

- 添加新的配置选项 `--enable-avoid-windows-handle`，这会导致预处理器符号 `AVOID_WINDOWS_HANDLE` 被定义。定义后，qpdf 会避免引用 Windows HANDLE 类型，而某些版本的 Windows SDK 不允许引用 HANDLE。
- 对于 Windows 构建，尝试确定需要传递给编译器和链接器哪些选项，以启用 `wmain`。这导致定义了预处理器符号 `WINDOWS_WMAIN`。如果你用其他编译器自己构建，可以定义这个符号来让 `wmain` 被使用。

这是为了让 Windows qpdf 命令能够接收 Unicode 命令行选项所必需的。

### 8.3.0: 2019 年 1 月 7 日

- 命令行增强功能

- 壳体补全：你现在可以使用 `eval $(qpdf --completion-bash)` 和 `eval $(qpdf --completion-zsh)` 来实现 `bash` 和 `zsh` 的壳体补全。
- 现在，在使用 `--pages` 和 `--split-pages` 选项合并和拆分文件时，页面编号（也称为页面标签）得以保留。
- 在使用 `--分割页` 选项分割页面时，书签部分保留。具体来说，大纲词典和部分支持的元数据会被复制到拆分文件中。结果是原始文件中的所有书签都会出现，指向保存页面的书签有效，指向未保存页面的书签则无效。这是在拆分和合并作中，为书签提供适当支持的过渡步骤。
- 页面整理：添加新选项 `--整理`。当指定时，`--页` 的语义从连接变为排序。有关示例和讨论，请参见页面选择。
- 以 JSON 格式生成信息，主要用于用于 C++ 以外的语言的 qpdf。添加新的选项 `--json`、`--json-key` 和 `--json-object` 来生成 PDF 文件的 JSON 表示。运行 `qpdf --json-help` 获取 JSON 格式的描述。更多信息请参见 qpdf JSON。
- 如果 PDF 文件显示表单字段显示过时，qpdf 将使 qpdf 为表单字段生成出现。当程序不知道如何重新生成填写字段的外观时，就会发生这种情况。
- `--flatten-annotations` 标志可用于平整注释，包括形式字段。通常，注释是与页面分开绘制的。扁平化注释是指将它们的外观合并到页面内容中的过程。如果你打算用不懂注释的工具来旋转或合并页面，你可能需要这样做。使用该标志时，你也可以使用 `--generate-appearances`，因为过时表单字段的注释不会被平整化，否则会导致信息丢失。
- `--optimize-images` 标志指示 qpdf 使用所有图像进行 DCT (JPEG) 压缩，只要图像尚未被有损压缩，且重新压缩图像体积会减小。额外的选项——`--oi-min-width`、`--oi-min-height` 和 `--oi-min-area` 防止了宽度、高度或像素面积（宽度×高度）低于指定阈值的图像的重新压缩。
- `--show-object` 选项现在可以表示为 `--show-object=trailer`，以显示 trailer 字典。

- 漏洞修复与增强

- QPDF 现在能自动检测并恢复悬挂引用。如果 PDF 文件包含对不存在对象的间接引用（这是有效的），在添加新对象时，新对象可能会使用悬挂引用的对象 ID，从而使悬挂引用指向新对象。此案现已被阻止。

- 修正字段设置代码：字符串始终以 UTF-16 格式书写，复选框和单选按钮在值和外观状态同步方面得到正确处理。
- QPDF::checkLinearization () 不再导致程序在检测线性化数据问题时崩溃。它会发出正常的警告或错误。
- 通常 qpdf 将形式为 @file 的参数视为命令行选项应从文件中读取。如果文件不存在但 @file 有，qpdf 会把 @file 当作常规选项。这使得处理名称以 @ 开头的 PDF 文件更为便捷。

#### • 图书馆增强

- 取消在大多数情况下，QPDF::copyForeignObject 调用中使用的源 QPDF 对象必须保留直到目标 QPDF 写入的限制。例外情况是源流通过 QPDFObjectHandle::StreamDataProvider 获取数据。

如需更深入的讨论，请参见 QPDF.hh 中关于 copyForeignObject 的评论。

- 添加新方法 QPDFWriter::getFinalVersion ()，返回最终将写入最终文件的 PDF 版本。有关使用限制，请参见 QPDFWriter.hh 的评论。
- 为 PDF 中使用的一些字符集添加多种字符串转码方法

文件：QUtil::utf8\_to\_ascii、QUtil::utf8\_to\_win\_ansi、QUtil::utf8\_to\_mac\_roman 和 QUtil::utf8\_to\_utf16。对于仅支持有限字符集的单字节编码，这些方法用指定的替代替换替代不支持的字符。

- 为 QPDFAnnotationObjectHelper 和 QPDFFormFieldObjectHelper 添加新方法  
查询标志并解释不同字段类型。在 qpdf/Constants.h 中定义常量，以帮助解释旗帜值。
- 添加新的方法 QPDFAcroFormDocumentHelper::generateAppearancesIfNeeded 和 QPDFFormFieldObjectHelper::generateAppearance，用于生成外观流。有关限制，请参见 QPDFFormFieldObjectHelper.hh 中的讨论。
- 添加两个新的资源字典辅助函数：QPDFObjectHandle::getResourceNames () 返回所有二级键的列表，对应资源名称，QPDFObjectHandle::mergeResources () 合并两个资源词典，只要它们的键不冲突。这些方法适用于某些类型能够解析多处资源的对象，如表单字段。
- 添加方法 QPDFPageDocumentHelper::flattenAnnotations () 和 QPDFAnnotationObjectHelper::getPageContentForAppearance ()，用于处理注释扁平化的低层细节。
- 新增辅助类：QPDFOutlineDocumentHelper、QPDFOutlineObjectHelper、QPDFPageLabelDocumentHelper、QPDFNameTreeObjectHelper 和 QPDFNumberTreeObjectHelper。
- 添加方法 QPDFObjectHandle::getJSON ()，返回对象的 JSON 表示。调用 serialize () 将结果转换为字符串。
- 添加一个简单的 JSON 串行器。这不是一个完整的通用 JSON 库。它允许在某些限制下组装和序列化 JSON 结构，这些限制在头文件中描述。这就是 qpdf 新 JSON 表示中使用的串行器。
- 新增 QPDFObjectHandle::Matrix 类，并加入一些方便处理六元数值数组作为矩阵的方法。
- 添加新方法 QPDFObjectHandle::wrapInArray，如果是数组，则返回对象本身，否则返回包含该对象的数组。这是 PDF 中常见的构造。这种方法可以防止

你不用明确测试某物是单元素还是数组。

- 建筑改进

- 不再需要运行 `autogen.sh` 从崭新的收银台开始建造。自动生成的文件现在已提交，这样可以直接从仓库的干净检出到无需自动约束的平台上构建。配置脚本会检测文件是否过期，同时确定有工具可以重新生成文件。
- 拉取请求和主分支现在在 Azure Pipelines 中自动构建，开源项目免费。该构建包含 Linux、mac、Windows 32 位和 64 位，配合 mingw 和 MSVC，以及 Appliance 构建。官方 qpdf 版本现已采用 Azure 管道构建。

- 包装员笔记

- 文档中新增了一个面向包装员的注释部分。请参阅包装商说明。
- qpdf 检测自动生成的文件过时。如果你的打包系统会自动刷新 `libtool` 或 `autoconf` 文件，可能会导致这次检查失败。为了避免这个问题，可以通过 `--disable-check-autofiles` 来配置。
- 如果您希望自动启用 qpdf 补全，可以在发行版的默认位置安装补全文件。你可以在完成目录中找到可以安装的示例完成文件。

### 8.2.1: 2018 年 8 月 18 日

- 命令行增强功能

- 添加 `--keep-files-open=[yn]` 以覆盖合并时是否保持文件打开的默认决定。更多细节请参见 “`--keep-files-open`” 讨论。

### 8.2.0: 2018 年 8 月 16 日

- 命令行增强功能

- 添加无警告选项以抑制发出警告信息。如果有任何条件会导致警告，退出状态仍然是 3。

- 漏洞修复与优化

- 性能修复：优化页面合并作，避免文件合并时不必要的开闭调用。这解决了合并某些类型文件时出现的显著卡顿问题。
- 优化 TIFF 预测器的内存使用方式，大幅提升使用 Flate 压缩的高分辨率图像文件的性能和内存使用率。
- 修复错误：在某些情况下，行尾字符在字符串中未被正确处理。
- 修复错误：在非常小的文件上使用 `--progress` 可能导致无限循环。

- API 增强

- 添加新的类 `QPDFSystemError`，源自 `std::runtime_error`，现在由 `QUtil::throw_system_error` 抛出。这使得触发错误值能够被检索。
- 添加关闭文件输入源：`stayOpen` 方法，使关闭文件输入源保持在手动指示的高活动时段开启，从而减少频繁开关作的开销。

- 构建变更

- 对于 mingw 构建，将 DLL 导入库名称从 `libqpdf.a` 改为 `libqpdf`。DLL.A 更准确地反映的是，它是一个导入库，而不是静态库。这可能

为未来支持静态库铺平了道路，尽管目前 qpdf 的 Windows 构建仅构建 DLL 和可执行文件。

#### 8.1.0: 2018 年 6 月 23 日

- 可用性改进

- 在拆分文件时，qpdf 检测文档元数据声称引用的字体和图像，这些字体和图像实际上并非从输出文件中被引用。这一变化可能导致某些软件包创建的文件分割 PDF 文件体积显著缩小。在某些情况下，这也会让分页变慢。之前的 qpdf 版本会相信文档元数据，有时会包含所有其他页面的所有图片，尽管这些页面已经不存在。在极不可能需要使用旧行为，或者页面分割非常慢的情况下，可以通过指定 `--preserve-unreferenced-resources` 来启用旧行为（和速度）。
- 合并多个 PDF 文件时，qpdf 不再将所有文件保持打开。这使得能够合并可能超过作系统最大开放文件数量限制的文件数量。
- `--rotate` 选项的语法已被扩展，使页面范围变得可选。如果你指定 `--rotate=角度` 但未指定页面范围，旋转将应用到所有页面。这对于调整从多页文档倒置扫描生成的 PDF 时尤其有用。
- 合并多个文件时，`--verbose` 选项现在会打印每个文件在该文件上作时的信息。
- 当指定 `--progress` 选项时，qpdf 会打印一个持续显示的最佳猜测写入过程进展的指示器。注意，和所有进度计量表一样，这只是一个近似值。该选项的实现方式使其对使用 qpdf 库的软件非常有用;详见下文的 API 增强。

- 漏洞修复

- 正确解密使用标准安全处理程序第 3 版但使用 40 位密钥的文件（尽管第 3 版支持 128 位密钥）。
- 限制嵌套数据结构的深度，以防止某些类型格式不佳（恶意）PDF 导致崩溃。
- 在“endstream 前换行”模式下，在对象流结束处插入所需的额外换行。此前有一例被遗漏。

- API 增强

- 首批高级“辅助”接口已推出。这些工具旨在提供比直接使用 `QPDFObjectHandle` 更便捷的文档功能交互方式。有关助手的详细信息，请参见助手课程。具体的附加接口将在下文描述。
- 新增两个文档助手类：`QPDFPageDocumentHelper` 用于处理页面，`QPDFAcroFormDocumentHelper` 用于交互式表单。没有旧方法被移除，但 `QPDFPageDocumentHelper` 现在是执行页面作的首选方式，而不是直接调用 `QPDFObjectHandle` 和 QPDF 中的旧方法。头文件中的注释会引导你访问新的接口。详情请参见头文件和更新日志。
- 新增三个对象辅助类：`QPDFPageObjectHelper` 用于页面，

`QPDFFormFieldObjectHelper` 用于交互式表单字段，`QPDFAnnotationObjectHelper` 用于注释。目前这三个职业都相当稀少，但它们有一些实用且基础的功能。

- 新增了一个示例程序 `examples/pdf-set-form-values.cc`, 展示了新文档和对象辅助器的使用。
- 已添加方法 `QPDFWriter::registerProgressReporter`。这种方法允许你注册一个函数, `QPDFWriter` 调用它通过写出输出来更新你对其百分比的猜测。客户端程序可以利用这一点实现相当准确的进度表。`qpdf` 命令行工具利用这一点实现了其 `--progress` 选项。
- 新方法 `QPDFObjectHandle::newUnicodeString` 和 `QPDFObject::unparseBinary` 具有添加后, 是为了方便创建使用大端 UTF-16 显式编码的字符串。这对于创建出现在内容流之外的字符串非常有用, 比如标签、表单字段、大纲、文档元数据等。
- 新增了一个类 `QPDFObjectHandle::Rectangle`, 以便于 PDF 矩形的作, 这些矩形只是四个数值的数组。

#### 8.0.2: 2018 年 3 月 6 日

- 当在跟踪交叉引用流或表时检测到循环, 应视为损坏, 而不是默默无息地忽略之前的表。这防止了某些损坏文件中本可恢复的数据丢失。
- 正确处理没有内容的页面。

#### 8.0.1: 2018 年 3 月 4 日

- 解压/`FlateDecode` 流时请忽略数据检查错误。这与大多数其他 PDF 阅读器一致, 并允许 `qpdf` 从另一类格式错误的 PDF 文件中恢复数据。
- 在命令行指定页面范围时, 支持在页号前加 “r”, 表示应从末尾计数。例如, `r3-r1` 范围表示文档的最后三页。

#### 8.0.0: 2018 年 2 月 25 日

- 包装与发行变更
  - `qpdf` 现在除了其他分发方式外, 也以 `ApplImage` 的形式分发。`ApplImage` 可以在下载区与其他软件包一起找到。感谢 Kurt Pfeiple 和 Simon Peter 的贡献。
- 漏洞修复
  - `QPDFObjectHandle::getUTF8Val` 现在正确地将非 Unicode 字符串视为使用 PDF 文档编码。
  - 改进处理 PDF 文件中非预期类型对象的处理。在大多数情况下, `qpdf` 能够针对此类情况发出警告, 而不是有例外地失败。之前的 `qpdf` 版本有时会因错误错误而失败, 比如“尝试对错误类型的对象进行字典对象作”。

这种情况现在基本或完全应该被消除。

- `qpdf` 命令行工具的改进。这里列出的所有新选项均有更详细的文档, 详见“运行 `qpdf`”。
- 新增了 `--linearize-pass1=file` 选项用于调试 `qpdf` 的线性化代码。
- 选项 `--coalesce-contents` 可用于将页面中内容为数组流的内容流合并为单一流。
- API 增强。所有新的 API 调用都记录在各自类的头文件中。API 没有不兼容的更改。
- 向 C API 添加函数 `qpdf_check_pdf`。该函数进行基本检查, 是 `qpdf --check` 执行的子集。



——对 qpdf 词汇层进行了重大增强。完整的增强列表请参阅变更日志文件。大多数改动带来了 qpdf 处理错误文件的能力提升。程序还可以将空白、注释和内联图片作为令牌处理。

– 用于词汇层面处理 PDF 内容流的新 API。新的类 `QPDFObjectHandle::TokenFilter` 允许开发者提供令牌处理程序。令牌过滤器可以在 `QPDFObjectHandle` 中与多种不同方法一起使用，也可以用于低层接口。详情请参见 `QPDFObjectHandle.hh` 中的评论，以及新的示例 `pdf-filter-tokens.cc` 和 `examples/pdf-count-strings.cc`。

#### 7.1.1: 2018 年 2 月 4 日

- 修复错误：/ID 字段长度不超过 16 字节的文件现在可以被正确线性化
- 部分平台已修正部分编译和链接问题。

#### 7.1.0: 2018 年 1 月 14 日

- PDF 文件包含流，这些流可以通过各种压缩算法进行压缩，在某些情况下，这些算法还可以通过各种预测函数进行增强。此前仅支持巴布亚新几内亚的上升预测器。在此版本中，支持所有 PNG 预测器和 TIFF 预测器。这增加了 qpdf 能够处理的文件范围。
- QPDF 现在允许在打开加密文件时指定原始加密密钥代替密码，并可选地显示文件所使用的加密密钥。这是一种非标准作，但在某些情况下非常有用。更多详情请参见 `QPDF.hh` 中关于 `--password-is-hex-key` 的讨论或 `QPDF::setPasswordIsHexKey` 相关的评论。
- 修复错误：以小数点尾数结尾的数字现在被正确识别为数字。
- 修复错误：在某些平台（尤其是 MacOS）从源码构建 qpdf 时，可能会被系统上安装的旧版本 qpdf 混淆。此点已更正。

#### 7.0.0: 2017 年 9 月 15 日

- 包装与发行变更
    - qpdf 的主要许可证现为 Apache 许可证的 2.0 版本，而非艺术许可的 2.0 版本。你仍然可以选择将 qpdf 视为艺术许可 2.0 版本的授权。
    - qpdf 不再依赖于 PCRE（Perl 兼容正则表达式）库。qpdf 现在对 JPEG 库增加了依赖。
  - 漏洞修复
    - 本版本包含许多针对各种无限循环、内存泄漏及其他内存错误的错误修复，这些错误可能出现在专门制作或错误的 PDF 文件中。
  - 新功能
    - qpdf 现在支持用 JPEG 或 RunLength 编码的读写流。  
为控制该行为，已添加库 API 增强和命令行选项。请参见命令行选项 `--compress-streams` 和 `--decode-level` 以及方法 `QPDFWriter::setCompressStreams` 和 `QPDFWriter::setDecodeLevel`。
- QPDF 在从损坏文件中恢复方面要好得多。在大多数情况下，qpdf 会跳过无效对象，并通过不尝试过滤断裂流来保留断裂的流数据。QPDF 现在能够恢复，或者至少不会崩溃，这些文件是我这几年收到的。
- 页面旋转现在支持并可从库和命令行访问。

- QPDFWriter 支持以保持 PCLm 合规性的方式编写文件，以支持无人驾驶打印。这非常专业化，只对已经知道如何创建 PCLm 文件的应用程序有用。
- qpdf 命令行工具的改进。这里列出的所有新选项均有更详细的文档，详见“运行 qpdf”。
  - 命令行参数现在可以通过 @file 或 @- 语法从文件或标准输入中读取。  
请参见基础召唤。
  - --旋转：请求页面旋转
  - 换行前于 endstream：确保文件中每个 endstream 关键字之前都出现换行；用于防止 QPDF 在已有合规文件上违反 PDF/A 合规性。
  - --preserve-unreferenced：在输入 PDF 中保留未引用对象
  - --分割页：将输出拆分为固定页数的块
  - --verbose：打印每个创建的输出文件名称
  - --压缩流和---解码级替换--流-数据以提升的粒度  
控制流数据的压缩和解压。--stream-data 选项将继续开放。
  - 运行 qpdf 时——检查其他选项，检查总是先进行。这使得 qpdf 能够在输出其他信息之前完成完整的恢复逻辑。这在手动恢复损坏文件、查看 qpdf 重生成的交叉引用表或其他类似作时尤其有用。
  - 提前处理 ---页面，以便在页面分割/合并完成后，其他选项如 --show-pages 或 --split-pages 等作文件。
- API 变更。所有新的 API 调用都记录在各自类的头文件中。
  - QPDFObjectHandle:: rotate 页面：对页面对象应用旋转
  - QPDFWriter:: setNewlineBeforeEndstream：强制换行出现在 endstream 之前
  - QPDFWriter:: setPreserveUnreferencedObjects：保留出现在输入 PDF 中的未引用对象。默认行为是丢弃它们。
  - 新的流水线类型 PL\_RunLength 和 PL\_DCT 面向希望直接生产或使用 RunLength 或 DCT 流数据的开发者开放。examples/pdf-create.cc 示例展示了它们的使用。
  - QPDFWriter:: setCompressStreams 和 QPDFWriter:: setDecodeLevel 方法控制不同类型流压缩的处理。
  - 添加新的 C API 函数 qpdf\_set\_compress\_streams、qpdf\_set\_decode\_level、qpdf\_set\_preserve\_unreferenced\_objects 和 qpdf\_set\_newline\_before\_endstream，对应新的 QPDFWriter 方法。

6.0.0: 2015 年 11 月 10 日

- 实现 --deterministic-id 命令行选项和 QPDFWriter:: setDeterministicID，以及用于为非加密文件生成确定性 ID 的 C API 函数 qpdf\_set\_deterministic\_ID。当选择此选项时，文件的 ID 取决于输出文件的内容，而非时间戳或输出文件名等临时项。
- 让 qpdf 对 xref 表条目长度不对的文件更宽容。

5.1.3: 2015 年 5 月 24 日

- 修复错误: fix-qdf 未能正确处理包含超过 255 个对象的对象流的文件。
- 修复错误: qpdf 在尚未创建密钥的新 Windows 安装上, 未能正确初始化 Microsoft 的安全加密提供商。
- 修正谷歌安全团队的 Gynael Coldwind 和 Mateusz Jurczyk 发现的一些错误。详情请参阅更新日志。
- 正确处理那些根本没有内容的页面。在许多情况下, qpdf 处理得很好, 但有些方法盲目获取页面内容, 并考虑可能没有内容。
- 让 qpdf 更稳健, 以应对更多可能出现在无效 PDF 文件中的问题。

#### 5.1.2: 2014 年 6 月 7 日

- 修复错误: 线性化文件可能会在极不可能的情况下导致输出文件损坏。详情请参见更新日志。被击中的概率非常低, 但确实有一个人被击中了。
- 修复错误: qpdf 无法写入带有解码参数引用其他流的流的文件。
- 新示例程序: pdf-split-pages: 高效地将 PDF 文件拆分成单独页面。示例程序比用 qpdf--pages 更高效地实现了这一点。
- 打包修复: Visual C++ 二进制文件不支持 Windows XP。通过更新用于生成发布二进制文件的编译器, 这一问题得到了解决。

#### 5.1.1: 2014 年 1 月 14 日

- 性能修复: 某些类型的文件复制外来物体可能会非常慢。这很可能在分页时显现, 有时是因为多次穿越同一对象。

#### 5.1.0: 2013 年 12 月 17 日

- 新增运行时选项 (QUtil:: setRandomDataProvider) 以提供您自己的随机数据提供者。如果你想避免使用操作系统提供的安全随机数生成功能或 stdlib 安全性较低版本, 可以使用这个功能。详情请参见 include/qpdf/QUtil.hh 中的评论。
- 固定图像比较测试不生成每像素 12 位图像, 因为某些版本的 tiffcmp 在比较时存在 bug。这增加了图像比较测试所需的磁盘空间, 而图像比较测试默认是关闭的。
- 在 MacOS 最新的 clang 和 Windows 的 Visual C++ 上引入一些小的编译修复。
- 能够处理那些 xref 表头尾加空格而不是换行的坏文件。

#### 5.0.1: 2013 年 10 月 18 日

- 感谢 Florian Weimer 和红帽产品安全团队的详细审查, 本次发布包含了多项用户不可见的强化安全变更。完整列表请参见源发行版中的 ChangeLog 文件。
- 在有条件的情况下, 作系统专用的安全随机数生成用于生成初始化向量及加密或文件创建过程中使用的其他随机值。对于 Windows 版本, 这增加了对 Microsoft 密码学 API 的依赖。要禁用作系统专用密码并使用旧版本, 请将 --enable-insecure-random 选项传递到 ./configure。
- qpdf 命令行工具现在在较新加密版本中指定 -accessibility=n 时会发出警告, 表示该选项被忽略。根据规范, QPDF 一直忽略这个标志, 但之前是静默地这样做的。该警告仅由命令行工具发出, 库本身不触发。图书馆对该旗帜的处理方式未变。

## 5.0.0: 2013 年 7 月 10 日

- 修复错误：之前的 qpdf 版本在生成对象流时会在生成时 != 0 丢失对象。修复这个问题需要对公开 API 进行修改。
- 从公共 API 中移除了那些本应只被 QPDFWriter 调用、实际上无法在其他地方调用的方法。详情请参见更新日志。
- 新增了 QPDFObjGen 类，用于表示对象 ID/生成对。  
QPDFObjectHandle::getObjGen() 现在比 QPDFObjectHandle::getObjectID() 和 QPDFObjectHandle::getGeneration() 更受青睐，因为这减少了人们误写忽略生成编号的代码。更多说明请参见 QPDF.hh 和 QPDFObjectHandle.hh。
- 在 qpdf 命令中添加 --show-npages 命令行选项，以显示文件的页数。
- qpdf 命令允许在 --pages 内省略页面范围。省略时，页面范围隐含为文件中的所有页面。
- 为支持不同类型的损坏文件或读取器，进行了各种增强。详情可见 ChangeLog。

## 4.1.0: 2013 年 4 月 14 日

- 给在发行版中包含 qpdf 的人说明：libtool 生成的 .la 文件现在由 qpdf 的 Make Install Target 安装。之前，这些设备并未安装。这意味着如果你的发行版不愿意包含 .la 文件，你必须在打包过程中将它们移除。
- 重大改进：API 改进以支持内容流解析。此次增强包括以下变化：
  - QPDFObjectHandle::parseContentStream 方法解析内容流中的对象，并调用回调类中的处理程序。示例 examples/pdf-parse-content.cc 展示了该方法的使用方式。
  - QPDFObjectHandle 现在可以表示运算符和内联图像，这些对象类型只能出现在内容流中。
  - 方法 QPDFObjectHandle::getTypeCode() 返回一个枚举的类型值，代表底层对象类型。方法 QPDFObjectHandle::getTypeName() 返回一个描述 QPDFObjectHandle 对象类型名称的文本字符串。这些方法可用于更高效的解析和调试/诊断消息。
- QPDF——检查现在除了进行其他检查外，还解析所有页面的内容流。虽然仍有许多类型的错误无法检测，但内容流中的语法错误将被报告。
- 为便于支持更广泛的编译器和版本，进行了少量编译改进。
  - 警告标志已移入独立变量 autoconf.mk
  - configure flag --enable-werror 工作，适用于 Microsoft 编译器
  - 所有 MSVC CRT 安全警告均已解决。
  - C++ 代码中所有 C 风格的 cast 已被 C++ cast 取代，许多原本为抑制某些编译器更高警告级别而包含的 cast 已被删除，主要是为了清晰。整数型强制发生的地方也被仔细审视过。手册中记录了新的选角政策。这主要是那些将 qpdf 移植到新平台或编译器的用户所关注的。对于使用该库编写代码的程序员来说，它并不可见
  - 在将数字转换为字符串的代码中，一些内部限制已被移除。这对用户来说基本不可见，但确实会触发一些旧版本 mingw-w64 的 C++ 库中的一个 bug。参见

如果你觉得这会影响你，可以 README-windows.md 源发行版。与 qpdf 二进制分发的 DLL 副本不受此问题影响。

- 之前随 qpdf 附带的 RPM 规范文件已被移除。这是因为几乎所有 Linux 发行版都包含了 qpdf，因为它依赖于 CUPS 过滤器。
- 包含一些漏洞修复：
  - 被覆盖的压缩对象得到正确处理。以前，有些结构会导致 qpdf 看到某些对象的旧版本。最常见的表现形式是某些文件中已填充的表单值丢失。
  - 安装不再使用某些命令的 GNU/Linux 特定版本，因此请使用原生工具让安装在 Solaris 上运行。
- 64 位 mingw 的 Windows 二进制包不再包含 32 位 DLL。

#### 4.0.1: 2013 年 1 月 17 日

- 修复测试套件中二进制附件检测的问题，以避免某些平台上的错误测试失败。
- 在 QPDF.hh 中添加澄清注释，说明通过更新的加密格式，已无法在已知用户密码的情况下恢复用户密码。在早期的加密格式中，用户密码在文件中用所有者密码加密。在较新的加密格式中，文件使用单独的加密密钥，该密钥通过用户密码和所有者密码独立加密。

#### 4.0.0: 2012 年 12 月 31 日

- 重大改进：新增了对 Adobe Acrobat X 版本支持的新型加密方案的支持。这包括使用 127 字符密码、256 位加密密钥，以及 ISO 32000-2 (PDF 2.0 规范) 中规定的加密方案。该方案可通过命令行指定使用 256 位密钥来选择。qpdf 还支持已废弃的 Acrobat IX 加密方法。这种加密方式存在已知的安全弱点，不应在实际作中使用。不过，这类文件“存在于”荒野“中，因此对该方案的支持仍然有用。新增的方法 QPDFWriter::setR6EncryptionParameters (针对 PDF 2.0 方案) 和 QPDFWriter::setR5EncryptionParameters (针对已废弃方案) 已被添加，以支持这些新的加密方案。相应的功能也被添加到了 C API。
- PDF 版本信息中对 Adobe 扩展级别的全面支持从 PDF 1.7 版本 (对应 ISO 32000) 开始，Adobe 通过增加扩展级别而非增加版本来增加新功能。这种支持包括增加了 QPDF::getExtensionLevel 方法用于获取文档的扩展级别，增加了接受扩展级别的 QPDFWriter::setMinimumPDFVersion 和 QPDFWriter::forcePDFVersion 版本，以及用于在命令行中指定强制和最小版本的扩展语法，如--force-version 和--min-version 所述。相应的功能也被添加到了 C API。
- 一些小修复，防止 qpdf 引用文件中未被文件整体结构引用的对象。大多数文件没有此类对象，但有些文件包含未引用且有错误的对象，因此这些修复防止了 qpdf 无谓地拒绝或抱怨此类对象。
- 添加新的通用方法，用于从程序员定义的源码读写文件。方法 QPDF::processInputSource 允许程序员使用任何输入源，而 QPDFWriter::setOutputPipeline 则允许程序员通过任意流水线写入输出文件。这些方法使得能够执行各种专业作，比如访问外部存储系统、为拥有自身 I/O 系统的编程语言创建 qpdf 绑定等。
- 添加新的方法 QPDF::getEncryptionKey，用于获取文件中所用底层加密密钥。

- 本版本包含少量不兼容的 API 变更。虽然努力避免此类更改，但该版本中所有不兼容的 API 变更都是针对 API 中可能永远不会在库外使用的部分。在所有情况下，修改的方法或结构都是 QPDF 中公开的部分，以便从 QPDFWriter 调用，或者是验证代码中过度报告文件中通常不会引用的问题部分的一部分。无论如何，移除的方法都没有做过更糟糕的事情，比如错误报告文件中那些无关紧要的错误。以下 QPDF 级公共部件被不兼容地更改：

- 更新嵌套的 QPDF::EncryptionData 类，以添加新加密格式所需的字段，成员变量改为私有，以便未来的更改无需破坏向后兼容。
- 向 compute\_data\_key 增加了额外参数，QPDFWriter 用于计算用于加密特定对象的加密密钥。
- 移除了 flattenScalarReferences 方法。此方法以前在写入新 PDF 文件前使用过，但它有不良副作用，即 qpdf 读取文件中未被引用的对象。有些文件本身存在未引用且有错误的对象，因此可能导致 qpdf 拒绝几乎所有其他 PDF 阅读器都能接受的文件。事实上，qpdf 只依赖于 flattenScalarReferences 功能中的一小部分，因此只有这部分被保存，现在它直接在 QPDFWriter 中完成。
- 移除了方法的 decodeStreams。该方法被 qpdf 命令行工具的 --check 选项用于强制解码文件中的所有流，但也存在打开未被引用流的问题，因此可能报告误报。--check 选项现在会让 qpdf 按照原始文件写入新文件，因此它总是会引用和检查普通查看者会检查的文件部分。
- 移除了 trimTrailerForWrite 方法。QPDFWriter 使用该方法修改原始 QPDF 对象，删除拖尾词典中不适用于新文件的字段。这一功能虽然通常无害，但实现效果不佳，现已被 QPDFWriter 在复制拖车时过滤掉这些内容取代，而非修改原始 QPDF 对象。（注意，qpdf 从不修改原始文件本身。）

- 允许 PDF 头出现在文件前 1024 字节的任意位置。这与其他读者的做法一致。
- 把 pkg-config 文件设置成在 Requires.private 里列出 zlib 和 pcre，这样可以用 pkg-config 更好地支持静态链接。

### 3.0.2: 2012 年 9 月 6 日

- 修复错误：QPDFWriter::setOutputMemory 在不配合 QPDFWriter::setStaticID 时无法使用，几乎没用。这个问题已经修复了。
- 新的 API 调用 QPDFWriter::setExtraHeaderText 会在 PDF 文件的头部附近插入额外文本。预期用例是插入可能被下游应用消耗的注释，尽管可能存在其他用例。

### 3.0.1: 2012 年 8 月 11 日

- 3.0.0 版本增加了 pkg-config 文件，但发布说明中未提及此事。3.0.0 的发布说明也更新了，提到了这一点。
- 修复错误：如果一个对象流以标量对象结束且没有空格，qpdf 会错误报告它遇到了过早的 EOF。这个 bug 从 QPDF 版本 2.0 就存在了。

### 3.0.0: 2012 年 8 月 2 日

- 致谢：感谢托比亚斯·霍夫曼为 qpdf 3.0 版本发布所做的贡献。他负责了新页面作 API 的大部分实现和设计，并为 3.0 版本中许多改进贡献了代码和创意。

没有他的作品，这次发行肯定不会这么快，甚至可能永远不会。

- 不兼容的 API 变更：

- QPDFObjectHandle::replaceStreamData 方法，使用 StreamDataProvider 进行 pro-而流数据不再需要长度参数。该参数被移除，因为这为用户提供了简化调用代码的机会。该方法在 2.2 版本中引入。当时，长度参数是必要的，以确保每次调用流数据提供商时，调用时对特定流的长度相同。特别是线性化代码依赖于此。相反，qpdf 3.0 及以后版本会明确检查该约束。当流数据提供者首次被调用特定流时，实际长度会被保存，后续调用也需要返回相同数量的字节。这意味着调用码不再需要提前计算长度，这可以大大简化。如果你的代码因为多了一个参数而无法编译，而且你又不想对代码做其他修改，那就直接省略这个参数。

- 许多方法需要很长的时间，而其他整数类型则不同。大多数（如果不是全部）现有代码在这次更改后应该都能正常编译，因为这些参数以前一直是更小的类型。此更改是为了支持大小超过 2GB 的文件。

- 新增了对大文件的支持。测试套件验证了对大于 4GB 文件的支持，手动测试也验证了对大于 10GB 文件的支持。只要编译器和底层平台支持，32 位和 64 位平台均支持大文件。

- qpdf 命令行工具已支持页面选择（拆分和合并 PDF 文件）。参见页选。

- qpdf 命令行工具中新增了 --copy-encryption 选项，用于从其他文件复制加密参数。

- QPDF 对象新增了添加和删除页面的方法。参见添加和删除页面。

- QPDF 对象新增了从其他 PDF 文件复制对象的方法。参见复制来自其他 PDF 文件的对象

- 新增了一种方法 QPDFObjectHandle::parse，用于从字符串描述构建 QPDFObjectHandle 对象。

- QPDFWriter 新增了写入已打开的 stdio FILE\* 的方法，除了写入标准输出或命名文件外。QPDF 新增了处理已打开 stdio 文件\*的文件的方法。这使得能够从在完全读取或写入前已解除绑定的安全临时文件中读写 PDF。

- QPDF::emptyPDF 可以用来从零开始创建 PDF 文件。示例/pdf-create.cc 展示了它的使用方式。

- 现在有多种方法可以接受 std::string 参数。

- 库中新增了许多便捷方法，大多数在 QPDFObjectHandle 中。完整列表请参见 ChangeLog。

- 在支持 ELF 共享库的平台上构建时（如 Linux），默认启用符号版本。可以通过将 --disable-ld-version-script 传递到 ./configure 来禁用它们。

- 现在安装了文件 libqpdf.pc，以支持 pkg-config。

- 现在默认关闭了图片对比测试，因为不需要它们来验证正确的构建或 QPDF 移植。只有在更改 qpdf 实际生成的 PDF 输出时才需要它们。如果你对 QPDF 本身做了深度修改，应该启用它们。详情请参见 README.md。
- 大文件测试默认关闭，但可以通过 ./configure 或在测试套件运行前设置环境变量来开启。详情请参见 README.md。
- 当 qpdf 的测试套件失败时，失败默认不会再打印到终端。相反，应该在构建/qtest.log 中找到它们。对于使用自动构建器的打包器，你可以在 ./configure 中添加 --enable-show-failed-test-output 选项以恢复旧行为。

#### 2.3.1: 2011 年 12 月 28 日

- 修复因非线程安全使用 PCRE 库而导致的线程安全问题。
- 做了一些小的文档修正。
- 在测试套件中添加针对某些版本 ghostscript 中出现的漏洞的变通方法
- 修复 Visual C++ 2010 的小型构建问题。

#### 2.3.0: 2011 年 8 月 11 日

- 修复错误：当保留加密文件的明文元数据时，旧版 qpdf 会产生无有效密码的密码保护文件。这个作现在奏效了。该漏洞仅影响通过复制现有加密参数创建的文件;明确加密并指定明文元数据的做法曾经有效，现在依然有效。
- 用一个新的构造器增强 QPDFWriter，允许你延迟输出文件的指定。使用该构造函数时，你现在可以调用 QPDFWriter:: setOutputFilename 指定输出文件，或者使用 QPDFWriter:: setOutputMemory 使 QPDFWriter 将生成的 PDF 文件写入内存缓冲区。然后你可以使用 QPDFWriter:: getBuffer 来获取内存缓冲区。
- 添加新的 API 调用 QPDF:: replaceObject，用于用对象 ID 替换对象
- 添加新的 API 调用 QPDF:: swapObjects 以通过对象 ID 交换两个对象
- 添加 QPDFObjectHandle:: getDictAsMap 和 QPDFObjectHandle:: getArrayAsVector，以允许检索字典对象作为映射，将数组对象作为向量检索。
- 在 C API 中添加 qpdf\_get\_info\_key 和 qpdf\_set\_info\_key 函数，用于作文档/Info 字典中的字符串字段。
- 在 C API 中添加 qpdf\_init\_write\_memory、qpdf\_get\_buffer\_length 和 qpdf\_get\_buffer 函数，以便将 PDF 文件写入内存缓冲区而非文件。

#### 2.2.4: 2011 年 6 月 25 日

- 修复安装和编译问题;没有功能变更。

#### 2.2.3: 2011 年 4 月 30 日

- 处理一些带有错误字符的流，关键词后出现错误字符。
- 在规范内容流时，改进内嵌图像的处理。
- 增强错误恢复功能，以正确处理将对象 0 作为常规对象的文件，而规范明确禁止此类内容。

#### 2.2.2: 2010 年 10 月 4 日

- 在 C API 中添加新函数 qpdf\_read\_memory 以调用 QPDF:: processMemoryFile。这是 qpdf 2.2.1 中遗漏的。

#### 2.2.1: 2010 年 10 月 1 日



- 新增方法 QPDF:: setOutputStreams, 用其他流替换 std:: cout 和 std:: cerr, 用于生成诊断消息和错误消息。这对图形界面或其他希望捕获库生成的任何输出以以其他方式呈现给用户的应用程序非常有用。注意, qpdf 除非在 QPDF.hh 中明确说明, 否则不会写入 std:: cout (或指定的输出流), 错误流的唯一用途是用于警告。还请注意, 当调用 setSuppressWarnings (true) 时, 警告的输出会被抑制。
- 新增方法 QPDF:: processMemoryFile, 用于作加载到内存中的 PDF 文件, 而非磁盘文件。
- 给出警告, 但其他情况下忽略空的 PDF 对象, 将其视为空。空对象在 PDF 规范中不被允许, 但已知在某些实际 PDF 文件中会出现。
- 当图像滤镜缩写显示为流滤镜缩写时, 处理内联图像滤镜缩写。PDF 规范不允许以这种方式使用流过滤缩写, 但 Adobe Reader 和其他一些 PDF 阅读器接受, 因为有时它们在实际 PDF 文件中会出现错误。
- 对 PointerHolder 和 Buffer 实施其他改进以支持其他变更。

#### 2.2.0: 2010 年 8 月 14 日

- 在 QPDFObjectHandle 中添加新方法 (newStream 和 replaceStreamData 用于创建新流和替换流数据)。这使得能够执行以前无法实现的广泛作。
- 在 QPDFObjectHandle (addPageContents) 中添加新的辅助方法, 用于附加或前置新内容流到页面。这种方法使得作内容流成为可能, 而无需担心页面内容是单一流还是数组流。
- 在 QPDFObjectHandle 中添加新方法: replaceOrRemoveKey, 它用给定值替换字典键, 除非该值为空, 否则它会移除键。
- 在 QPDFObjectHandle 中添加新方法: getRawStreamData, 将原始 (未过滤) 流数据返回缓冲区。这补充了 getStreamData 方法, 后者返回过滤 (未压缩) 的流数据, 且仅在流数据可过滤时使用。
- 提供两个新示例: pdf-double-page-size 和 pdf-invert-images, 说明新添加的界面。
- 修复内存泄漏, 导致每个涉及对象引用周期中单个对象丢失几个字节。感谢建马提醒我关于泄密的注意。

#### 2.1.5: 2010 年 4 月 25 日

- 取消文件标识符字符串 16 字节的限制。这一不必要的限制阻止了 qpdf 对标识符字符串长度不完全是 16 字节的文件进行加密或解密。规范中没有施加这样的限制。

#### 2.1.4: 2010 年 4 月 18 日

- 将 2.1.2 版本的填充计算修复也应用到主交叉引用流中。
- 由于 qpdf --check 只执行有限的检查, 请澄清输出, 以明确可能仍有 qpdf 无法检查的错误。这应该会让人们对另一个 PDF 阅读器无法读取 qpdf 认为合适的文件感到不那么惊讶。

#### 2.1.3: 2010 年 3 月 27 日

- 修复一个可能在重写包含无引用对象流的 PDF 文件时导致的错误, 而这些对象又引用间接标量。
- 不要抱怨那些 (无效的) AES 流, 它们不是 16 字节的倍数。相反, 先填充内容再解密。

### 2.1.2: 2010 年 1 月 24 日

- 修复线性化文件中前半交叉参考流填充的 bug。这个漏洞可能导致线性化某些不幸文件时断言失败。

### 2.1.1: 2009 年 12 月 14 日

- 功能没有变化;在内部库头文件中插入缺失的包含以支持 GCC 4.4, 并更新测试套件以忽略损坏的 Adobe Reader 安装。

### 2.1: 2009 年 10 月 30 日

- 这是首个包含 Windows 支持的 qpdf 版本。在 Windows 上, 可以构建 DLL。此外, 还引入了部分 C 语言 API, 使得调用非 C++ 环境中的 qpdf 函数成为可能。我非常感谢 Z̃rko Gajic' (<http://zarko-gajic.iz.hr/>) 不懈地测试了该 DLL 的多个预发布版本, 并提供了许多改进界面的优秀建议。  
关于编程到 C 接口, 请参见头文件 `qpdf/qpdf-c.h` 和示例 `pdf-linearize.c`。
- Z̃rko Gajic 编写了一个 qpdf 的 Delphi 封装器, 可从 qpdf 下载端下载。Z̃rko 的 Delphi 封装采用与 qpdf 相同的授权条款发布, 并附带免责声明: “Delphi 封装单元 `qpdf.pas` 由 Z̃rko Gajic' (<http://zarko-gajic.iz.hr/> 年) 创建。使用请自行承担风险, 且用途随你所愿。不提供任何支持。提供示例代码。”
- 新增了对 AES 加密和加密过滤的支持。虽然 qpdf 目前不支持基于 PKI 加密的文件, 但通过 AES 和加密过滤器的加入, qpdf 现在能够打开大多数用新版 Acrobat 或其他 PDF 制作软件创建的加密文件。注意我没能用这种方式加密太多文件, 所以可能还有些 qpdf 无法处理的情况。如果你发现了, 请举报。
- 许多错误信息已被改进, 包含更多信息, 希望使 qpdf 成为 PDF 专家手动恢复受损 PDF 文件时更实用的工具。
- 尽量避免压缩元数据流。这与其他 PDF 制作应用一致。
- 提供新的命令行选项, 用于 AES 加密、明文元数据, 以及设置输出文件的最小和强制 PDF 版本。
- 在 QPDF 对象中添加额外的方法以查询文档权限。虽然 qpdf 不强制执行这些权限, 但它确实提供了这些权限, 以便使用 qpdf 的应用程序能够强制执行权限。
- qpdf 的 `--check` 选项已被扩展, 包含了一些额外信息。
- 不兼容的 API 变更:
  - QPDF 的异常处理机制现在使用 `STD::logic_error` 处理内部错误, 使用 `std::runtime_error` 处理运行时错误, 转而使用之前版本中已移除的 `QEXC` 类。`QEXC` 例外类早于将头文件添加到 C++ 标准库之前。qpdf 库本身抛出的大多数例外仍然是 `QPDFExc` 类型, 现在是从 `std::runtime_error` 派生的。捕获 `std::exception` 实例并通过调用 `what()` 方法显示的程序无需更改。
  - `QPDFExc` 类现在内部表示错误条件的各个字段, 并提供查询接口。字段中有一个数字错误代码, 可以帮助应用程序在 (少数) 不同错误条件下做出不同的行为。详情请参见 `QPDFExc.hh`。
- 警告可以从 qpdf 中以 `QPDFExc` 实例形式检索, 而非字符串。

– 嵌套的 QPDF::EncryptionData 类构造函数会额外引用一个参数。该类主要供 QPDFWriter 使用。终端用户应用其实没什么实用功能。它本来就不应该成为公开界面的一部分。

同样，计算内部加密字典参数的一些方法也已改变，以支持/R=4 加密。

– QPDF::getUserPassword 方法已被移除，因为它并未达到人们预期的功能。现在有两种新方法：

QPDF::getPaddedUserPassword 和

QPDF::getTrimmedUserPassword。第一种方法实现了旧 QPDF::getUserPassword 方法的做法，即返回带有 PDF 规范中可能二进制填充的密码。第二个返回的是人类可读的密码字符串。

– 曾经嵌套在 QPDFWriter 中的枚举类型已移至顶层枚举类型，并定义在 qpdf/Constants.h 文件中。这使得它们可以同时被 C 和 C++ 接口共享。

#### 2.0.6: 2009 年 5 月 3 日

- 不要尝试解压那些解码参数不存在的流。早期版本的 qpdf 会拒绝包含此类流的文件。

#### 2.0.5: 2009 年 3 月 10 日

- 改进 LZW 解码器中的错误处理，并修正前一版本中关于处理完整表时引入的小错误。LZW 解码器在本版本中得到了更严格的验证。

#### 2.0.4: 2009 年 2 月 21 日

- 为未标记“早期代码变更”标志的 LZW 流提供适当支持。特别感谢 Atom Smasher 报告了这个问题，并提供了我之前没有的压缩输入文件。

- 对文件恢复逻辑实施一些改进。

#### 2.0.3: 2009 年 2 月 15 日

- 用 GCC 4.4 干净地编译。
- 正确处理编码为 UTF-16BE 的字符串。

#### 2.0.2: 2008 年 6 月 30 日

- 更新测试套件，使其能正常运行非 bash 的/bin/sh 和 Perl 5.10。本次发布未对 qpdf 的原始代码本身做任何更改。

#### 2.0.1: 2008 年 5 月 6 日

- 功能和界面没有变化。本版本对源代码进行了修复，使 qpdf 能够正确编译并通过更广泛平台的测试套件。详情请参见源发行版中的 ChangeLog。

#### 2.0: 2008 年 4 月 29 日

- 首次公开发布。



## 致谢

qpdf 最初于 2001 年创建，并在 2001 年至 2005 年间我在 Apex CoVantage 工作期间进行了多次修改。离开 Apex 后，公司慷慨地允许我接管软件并继续作为开源项目维护，我对此非常感激。自那以后，我对它做了相当大的改进。我很幸运能为那些会做出这样决定的人工作。没有他们的支持，这项工作无法实现。

2020 年，我加入了 Advent Health Partners，该公司曾赞助过一些之前的 qpdf 工作，并且慷慨地允许我花一些“公司时间”维护 qpdf。



指数

• qpdf-options





## QPDF 命令行选项

### a

--accessibility, 38 --add-attachment, 44 --allow-insecure, 40 --allow-weak-crypto, 24 --annotate, 38

——集合, 38

### b

——比特, 38

### c

--检查, 46 --检查线性化, 47 --明文元数据, 39 --coalesce-contents, 30 --collate, 32 --completion-bash, 23 --completion-zsh, 23 --压缩流, 27 --压缩级别, 28 --copy-attachments-from, 44 --copy-encryption, 26 --copyright, 23 --创建日期, 45

### d

——解码级, 27  
——解密, 26  
--描述, 45 --确定性 ID, 24

### e

——空的, 21  
--encrypt, 26 --encryption-file-password, 27 --externalize-inline-images, 30 --extract, 38

### f

——档案, 32  
——文件名, 45

--filtered-stream-data, 47 --flatten-annotations, 33 --flatten-rotation, 33 --force-R5, 40 --force-V4, 40

——原力版, 30  
——形式, 38  
——来自, 43

### g

——产生曝光次数, 34 次——全球, 50 次

### h

——救命, 23

i--忽略 XREF-streams, 26 个--ii-min-字节, 30 个--is-encrypted, 46 个

j--job-json-file, 22 --job-json-help, 23 --jpeg-quality, 28 --json, 48 --json-help, 49 --json-input, 49 --json-key, 49 --json-object, 49 --json-output, 49 --json-stream-data, 49 --json-stream-prefix, 49

### k

--保持文件打开, 24 --保持文件-打开-阈值, 25 --保持-inline-images, 35 --key, 45

l--线性化, 26

--linearize-pass1, 51 --list-attachments, 48

## m

--max-stream-filters, 50 --mimetype, 45 --min-version, 30 --moddate, 45 --modify, 39 --modify-other, 39

## n

--换行-前-end-stream, 30 个--无默认限制, 50 个--no-original-object-ids, 27 个--no-warn, 24--normalize-content, 29

## O

--对象流, 29 --oi-min-area, 35 --oi-min-height, 34 --oi-min-width, 34 --优化图像, 34 --叠加, 33 --owner-password, 38

## p

--pages, 32 --parser-max-container-size, 50 --parser-max-container-size-damaged, 50 --parser-max-errors, 50 --parser-max-nesting, 50 --password, 23 --password-file, 24 --password-is-hex-key, 25 --password-mode, 25 --prefix, 45 --preserve-unreferenced, 29 --preserve-unreferenced-resources, 30 --print, 39 --progress, 24

## q

——QDF, 27

## r

——范围, 32——原始流数据, 47——重新压缩-平整, 28——移除-格式, 35——删除附件, 44——删除信息, 35——移除元数据, 35

--remove-page-labels, 35 --remove-restrictions, 26 --remove-structure, 35 --remove-unreferenced-resources, 29 --重复, 44 --replace, 45 --replace-input, 21 --report-memory-usage, 51 --requires-password, 46 --rotate, 33

## S

--set-page-labels, 35 --show-attachment, 48 --show-crypto, 23 --显示加密, 47 --显示加密密钥, 47--显示线性化, 47 --显示-n 页面, 48 --显示对象, 47

——展览版, 48 页

--显示-XREF, 47

——分页, 32 页

——静电-AES-IV, 51

——静电识别, 51

--流-数据, 28 --suppress-password-recovery, 25 --suppress-recovery, 26

t--test-json-schema, 51 --to, 43

## U

--underlay, 33 --update-from-json, 49 --use-aes, 39 --user-password, 38

## V

——冗长, 24

——版本, 23

## W

--警告-出口-0,22 --带图片, 48

## Z

——佐普利, 23