QFont Class

QFont 类指定用于绘制文本的字体的查询。

Header:	**#include < QFont>**
CMake:	find_package(Qt6 REQUIRED COMPONENTS Gui) target_link_libraries(mytarget PRIVATE Qt6::Gui)
qmake:	QT += gui

- 所有成员的列表，包括继承的成员
- 已弃用的成员
- QFont 是绘画类、隐式共享类和富文本处理 API的一部分。

注意：该类中的所有函数都是reentrant。

公共类型

enum	**Capitalization { MixedCase, AllUppercase, AllLowercase, SmallCaps, Capitalize }**
enum	**HintingPreference** { PreferDefaultHinting, PreferNoHinting, PreferVerticalHinting, PreferFullHinting }
enum	**SpacingType** { PercentageSpacing, AbsoluteSpacing }
enum	**Stretch** { AnyStretch, UltraCondensed, ExtraCondensed, Condensed, SemiCondensed, …, UltraExpanded }
enum	**Style** { StyleNormal, StyleItalic, StyleOblique }
enum	**StyleHint** { AnyStyle, SansSerif, Helvetica, Serif, Times, …, System }
enum	**StyleStrategy** { PreferDefault, PreferBitmap, PreferDevice, PreferOutline, ForceOutline, …, PreferQuality }
enum	**Weight** { Thin, ExtraLight, Light, Normal, Medium, …, Black }

公共函数

	QFont()
	QFont(const QString &*family*, int *pointSize* = -1, int *weight* = -1, bool *italic* = false)
	QFont(const QStringList &*families*, int *pointSize* = -1, int *weight* = -1, bool *italic* = false)
	QFont(const QFont &*font*, const QPaintDevice **pd*)

	QFont()
	QFont(const QFont &*font*)
	~QFont()
bool	**bold**() const
QFont::Capitalization	**capitalization**() const
QString	**defaultFamily**() const
bool	**exactMatch**() const
QStringList	**families**() const
QString	**family**() const
bool	**fixedPitch**() const
bool	**fromString**(const QString &*descrip*)
QFont::HintingPreference	**hintingPreference**() const
bool	**isCopyOf**(const QFont &*f*) const
bool	**italic**() const
bool	**kerning**() const
QString	**key**() const
qreal	**letterSpacing**() const
QFont::SpacingType	**letterSpacingType**() const
bool	**overline**() const
int	**pixelSize**() const
int	**pointSize**() const
qreal	**pointSizeF**() const
QFont	**resolve**(const QFont &*other*) const
void	**setBold**(bool *enable*)
void	**setCapitalization**(QFont::Capitalization *caps*)
void	**setFamilies**(const QStringList &*families*)
void	**setFamily**(const QString &*family*)
void	**setFixedPitch**(bool *enable*)
void	**setHintingPreference**(QFont::HintingPreference *hintingPreference*)
void	**setItalic**(bool *enable*)

	QFont()
void	**setKerning**(bool *enable*)
void	**setLetterSpacing**(QFont::SpacingType *type*, qreal *spacing*)
void	**setOverline**(bool *enable*)
void	**setPixelSize**(int *pixelSize*)
void	**setPointSize**(int *pointSize*)
void	**setPointSizeF**(qreal *pointSize*)
void	**setStretch**(int *factor*)
void	**setStrikeOut**(bool *enable*)
void	**setStyle**(QFont::Style *style*)
void	**setStyleHint**(QFont::StyleHint *hint*, QFont::StyleStrategy *strategy* = PreferDefault)
void	**setStyleName**(const QString &*styleName*)
void	**setStyleStrategy**(QFont::StyleStrategy *s*)
void	**setUnderline**(bool *enable*)
void	**setWeight**(QFont::Weight *weight*)
void	**setWordSpacing**(qreal *spacing*)
int	**stretch**() const
bool	**strikeOut**() const
QFont::Style	**style**() const
QFont::StyleHint	**styleHint**() const
QString	**styleName**() const
QFont::StyleStrategy	**styleStrategy**() const
void	**swap**(QFont &*other*)
QString	**toString**() const
bool	**underline**() const
QFont::Weight	**weight**() const
qreal	**wordSpacing**() const
QVariant	**operator QVariant**() const
bool	**operator!=**(const QFont &*f*) const
bool	**operator<**(const QFont &*f*) const

	QFont()
QFont &	**operator=**(const QFont &*font*)
QFont &	**operator=**(QFont &&*other*)
bool	**operator==**(const QFont &*f*) const

静态公共成员

void	**insertSubstitution(const QString &*familyName*, const QString &*substituteName*)**
void	**insertSubstitutions**(const QString &*familyName*, const QStringList &*substituteNames*)
void	**removeSubstitutions**(const QString &*familyName*)
QString	**substitute**(const QString &*familyName*)
QStringList	**substitutes**(const QString &*familyName*)
QStringList	**substitutions**()

相关非成员

size_t	**qHash(const QFont &*font*, size_t *seed* = 0)**
QDataStream &	**operator<<**(QDataStream &*s*, const QFont &*font*)
QDataStream &	**operator>>**(QDataStream &*s*, QFont &*font*)

详细说明

QFont 可以被视为对系统上一种或多种字体的查询。

当您创建 QFont 对象时，您可以指定希望字体具有的各种属性。Qt 将使用具有指定属性的字体，或者如果不存在匹配的字体，Qt 将使用最接近的匹配的已安装字体。实际使用的字体的属性可以从QFontInfo目的。如果窗口系统提供完全匹配exactMatch() 返回`true`. 使用QFontMetricsF获取测量值，例如使用 QFontMetrics::width() 的字符串的像素长度。

未特别设置的属性不会影响字体选择算法，优先使用默认值。

要加载特定的物理字体（通常由单个文件表示），请使用QRawFont反而。

请注意，一个QGuiApplication在使用 QFont 之前实例必须存在。您可以使用以下命令设置应用程序的默认字体QGuiApplication::setFont()。

如果所选字体不包含需要显示的所有字符，QFont 将尝试查找最接近的等效字体中的字符。当一个QPainter从字体中绘制字符，QFont 将报告它是否具有该字符；如果没有，QPainter将绘制一个未填充的正方形。

像这样创建 QFont：

```
QFont serifFont("Times", 10, QFont::Bold);
QFont sansFont("Helvetica [Cronyx]", 12);
```

构造函数中设置的属性也可以稍后设置，例如setFamily(),setPointSize(),setPointSizeF(),setWeight（）和setItalic()。其余属性必须在构建后设置，例如setBold(),setUnderline(),setOverline(),setStrikeOut（）和setFixedPitch()。QFontInfo应在设置字体属性*后*创建对象。AQFontInfo即使您更改字体的属性，对象也不会更改。相应的“get”函数，例如family(),pointSize() 等，返回设置的值，即使使用的值可能不同。实际值可从QFontInfo目的。

如果请求的字体系列不可用，您可以影响font matching algorithm通过选择一个特定的QFont::StyleHint和QFont::StyleStrategy和setStyleHint()。默认系列（对应于当前样式提示）由以下命令返回defaultFamily()。

您可以使用以下方式提供字体系列名称的替换insertSubstitution（）和insertSubstitutions()。可以使用以下命令删除替代品removeSubstitutions()。使用substitute() 检索家庭的第一个替代者，或者如果没有替代者则检索姓氏本身。使用substitutes() 检索家庭替代者列表（可能为空）。替换字体后，必须通过销毁并重新创建所有 QFont 对象来触发字体更新。

每个 QFont 都有一个key() 例如，您可以将其用作缓存或字典中的键。如果您想存储用户的字体首选项，您可以使用QSettings，写入字体信息toString() 并用以下命令读回fromString()。运算符<<() 和运算符>>() 函数也可用，但它们作用于数据流。

可以将屏幕上显示的字符高度设置为指定的像素数setPixelSize(); 然而使用setPointSize() 具有类似的效果并提供设备独立性。

加载字体可能会很昂贵，尤其是在 X11 上。QFont 包含广泛的优化，可以快速复制 QFont 对象，并缓存其所依赖的慢速窗口系统函数的结果。

字体匹配算法的工作原理如下：

1. 指定的字体系列（由setFamilies()) 被搜索。
2. 如果没有，则选择支持书写系统的替换字体。字体匹配算法将尝试为 QFont 中设置的所有属性找到最佳匹配。完成此操作的方式因平台而异。
3. 如果系统上不存在可以支持该文本的字体，则将在其位置显示特殊的“缺失字符”框。

注意：如果所选字体虽然总体上支持书写系统，但缺少一个或多个特定字符的字形，则 Qt 将尝试为该或这些特定字符查找后备字体。可以使用禁用此功能QFont::NoFontMerging风格策略。

在 Windows 中，对“Courier”字体的请求会自动更改为“Courier New”，这是 Courier 的改进版本，可以平滑缩放。可以通过设置来选择较旧的“Courier”位图字体PreferBitmap风格策略（参见setStyleStrategy()）。

找到字体后，其余属性将按优先级顺序进行匹配：

1. fixedPitch()
2. pointSize（）（见下文）
3. weight()
4. style()

如果您有一种在系列上匹配的字体，即使其他属性都不匹配，也会优先选择该字体，而不是在系列上不匹配但在其他属性上匹配的字体。这是因为字体系列是主要搜索标准。

如果点大小在请求点大小的 20% 以内，则点大小被定义为匹配。当多种字体匹配并且仅通过磅值来区分时，将选择与所请求的磅值最接近的字体。

用于绘制文本的实际系列、字体大小、粗细和其他字体属性将取决于窗口系统下所选系列的可用内容。AQFontInfo对象可用于确定用于绘制文本的实际值。

例子：

```
QFont f("Helvetica");
```

如果您同时拥有 Adobe 和 Cronyx Helvetica，您可能会选择其中一个。

```
QFont f("Helvetica [Cronyx]");
```

您可以在姓氏中指定您想要的铸造厂。上例中的字体 f 将设置为“Helvetica [Cronyx]”。

要确定窗口系统中实际使用的字体的属性，请使用QFontInfo对象，例如

```
QFontInfo info(f1);
QString family = info.family();
```

要找出字体规格，请使用QFontMetrics对象，例如

```
QFontMetrics fm(f1);
int textWidthInPixels = fm.horizontalAdvance("How many pixels wide is this text?");
int textHeightInPixels = fm.height();
```

有关字体的更多一般信息，请参阅comp.fonts FAQ。有关编码的信息可以从UTR17页。

可以参阅QFontMetrics,QFontInfo,QFontDatabase， 和Character Map Example。

成员类型文档

enum QFont::Capitalization

此字体适用的文本的渲染选项。

持续的	价值	描述
QFont::MixedCase	0	这是正常的文本呈现选项，不应用大小写更改。
QFont::AllUppercase	1	这会更改以全部大写类型呈现的文本。
QFont::AllLowercase	2	这会更改以所有小写类型呈现的文本。
QFont::SmallCaps	3	这会更改以小型大写字母类型呈现的文本。
QFont::Capitalize	4	这会将要呈现的文本更改为每个单词的第一个字符为大写字符。

enum QFont::HintingPreference

该枚举描述了可应用于字形的不同级别的提示，以提高显示器上的易读性，而这可能是由像素密度保证的。

持续的	价值	描述
`QFont::PreferDefaultHinting`	`0`	使用目标平台的默认提示级别。
`QFont::PreferNoHinting`	`1`	如果可能，渲染文本时不要暗示字形的轮廓。文本布局将在印刷上准确且可扩展，使用与打印时使用的相同的度量。
`QFont::PreferVerticalHinting`	`2`	如果可能，渲染没有水平提示的文本，但将字形与垂直方向的像素网格对齐。在密度太低而无法准确呈现字形的显示器上，文本将显得更清晰。但由于字形的水平度量没有暗示，文本的布局将可扩展到更高密度的设备（例如打印机），而不会影响换行等细节。
`QFont::PreferFullHinting`	`3`	如果可能，在水平和垂直方向上渲染带有提示的文本。文本将被更改以优化目标设备上的易读性，但由于度量将取决于文本的目标大小，因此字形、换行符和其他印刷细节的位置将不会缩放，这意味着文本布局可能看起来像在不同像素密度的设备上有所不同。

请注意，此枚举仅描述一个首选项，因为并非所有 Qt 支持的平台都支持全部提示级别。下表详细说明了给定提示首选项对选定的一组目标平台的影响。

	首选默认提示	不喜欢任何暗示	首选垂直提示	更喜欢完整提示
Qt 中启用 Windows 和 DirectWrite	全面暗示	垂直提示	垂直提示	全面暗示
自由类型	操作系统设置	没有暗示	垂直提示（浅色）	全面暗示
macOS 上的可可	没有暗示	没有暗示	没有暗示	没有暗示

注意：请注意，可以通过 DirectWrite 字体引擎更改 Windows 上的提示首选项。安装平台更新后的 Windows Vista 和 Windows 7 上可以使用此扩展。为了使用此扩展，请使用 -directwrite 配置 Qt。然后，目标应用程序将取决于目标系统上 DirectWrite 的可用性。

enum QFont::SpacingType

持续的	价值	描述
`QFont::PercentageSpacing`	`0`	值为 100 将保持间距不变；值为 200 会将字符后的间距扩大字符本身的宽度。
`QFont::AbsoluteSpacing`	`1`	正值会增加相应像素的字母间距；负值会减小间距。

enum QFont::Stretch

遵循 CSS 命名约定的预定义拉伸值。值越高，文本拉伸得越多。

持续的	价值	描述
`QFont::AnyStretch`	`0`	0 接受使用其他匹配的任何拉伸QFont属性（Qt 5.8 中添加）
`QFont::UltraCondensed`	`50`	50
`QFont::ExtraCondensed`	`62`	62
`QFont::Condensed`	`75`	75
`QFont::SemiCondensed`	`87`	87
`QFont::Unstretched`	`100`	100
`QFont::SemiExpanded`	`112`	112
`QFont::Expanded`	`125`	125
`QFont::ExtraExpanded`	`150`	150
`QFont::UltraExpanded`	`200`	200

可以参阅setStretch （） 和stretch()。

enum QFont::Style

该枚举描述了用于显示文本的不同样式的字形。

持续的	价值	描述
`QFont::StyleNormal`	`0`	无样式文本中使用的正常字形。
`QFont::StyleItalic`	`1`	专为表示斜体文本而设计的斜体字形。
`QFont::StyleOblique`	`2`	具有斜体外观的字形通常基于无样式字形，但未针对表示斜体文本的目的进行微调。

可以参阅Weight。

enum QFont::StyleHint

样式提示由font matching如果所选字体系列不可用，则使用算法查找适当的默认系列。

持续的	价值	描述
QFont::AnyStyle	5	让字体匹配算法来选择家族。这是默认设置。
QFont::SansSerif	Helvetica	字体匹配器更喜欢无衬线字体。
QFont::Helvetica	0	是 的同义词SansSerif。
QFont::Serif	Times	字体匹配器更喜欢衬线字体。
QFont::Times	1	是 的同义词Serif。
QFont::TypeWriter	Courier	字体匹配器更喜欢固定间距字体。
QFont::Courier	2	的同义词TypeWriter。
QFont::OldEnglish	3	字体匹配器更喜欢装饰字体。
QFont::Decorative	OldEnglish	是 的同义词OldEnglish。
QFont::Monospace	7	字体匹配器更喜欢映射到 CSS 通用字体系列“monospace”的字体。
QFont::Fantasy	8	字体匹配器更喜欢映射到 CSS 通用字体系列“fantasy”的字体。
QFont::Cursive	6	字体匹配器更喜欢映射到 CSS 通用字体系列“草书”的字体。
QFont::System	4	字体匹配器更喜欢系统字体。

enum QFont::StyleStrategy

风格策略告诉我们font matching算法应该使用什么类型的字体来找到合适的默认系列。

可以采用以下策略：

持续的	价值	描述
QFont::PreferDefault	0x0001	默认样式策略。它不喜欢任何类型的字体。
QFont::PreferBitmap	0x0002	更喜欢位图字体（而不是轮廓字体）。
QFont::PreferDevice	0x0004	更喜欢设备字体。
QFont::PreferOutline	0x0008	更喜欢轮廓字体（而不是位图字体）。
QFont::ForceOutline	0x0010	强制使用轮廓字体。
QFont::NoAntialias	0x0100	不要对字体进行抗锯齿处理。
QFont::NoSubpixelAntialias	0x0800	如果可能的话，避免对字体进行子像素抗锯齿。

持续的	价值	描述
QFont::PreferAntialias	0x0080	如果可能的话，抗锯齿。
QFont::NoFontMerging	0x8000	如果为特定书写系统选择的字体不包含请求绘制的字符，则 Qt 会自动选择包含该字符的外观相似的字体。NoFontMerging 标志禁用此功能。请注意，当所选字体不支持文本的书写系统时，启用此标志不会阻止 Qt 自动选择合适的字体。
QFont::PreferNoShaping	0x1000	有时，字体会对一组字符应用复杂的规则，以便正确显示它们。在某些书写系统中，例如婆罗米文字，这是为了使文本清晰易读所必需的，但在例如拉丁文字中，这仅仅是一个装饰特征。PreferNoShaping 标志将在不需要时禁用所有此类功能，这将在大多数情况下提高性能（自 Qt 5.10 起）。

其中任何一个都可以与以下标志之一进行“或”运算：

持续的	价值	描述
QFont::PreferMatch	0x0020	更喜欢完全匹配。字体匹配器将尝试使用已指定的确切字体大小。
QFont::PreferQuality	0x0040	更喜欢最优质的字体。字体匹配器将使用字体支持的最接近的标准点大小。

enum QFont::Weight

Qt 使用与 OpenType 兼容的从 1 到 1000 的权重范围。权重为 1 会很薄，而 1000 会很黑。

该枚举包含预定义的字体粗细：

枚举值	值	描述
QFont::Thin	100	100
QFont::ExtraLight	200	200
QFont::Light	300	300
QFont::Normal	400	400
QFont::Medium	500	500
QFont::DemiBold	600	600
QFont::Bold	700	700
QFont::ExtraBold	800	800
QFont::Black	900	900

成员函数文档

QFont::QFont()

构造一个使用应用程序默认字体的字体对象。

可以参阅QGuiApplication::setFont（） 和QGuiApplication::font()。

QFont::QFont(const QString &family, int pointSize = -1, int weight = -1, bool italic = false)

使用指定的构造一个字体对象*family*,*pointSize*,*weight*和*italic*设置。

如果*pointSize*为零或负数时，字体的磅值将设置为与系统相关的默认值。一般来说，这是12点。

这*family*名称还可以可选地包括铸造厂名称，例如“Helvetica [Cronyx]”。如果*family*可以从多个铸造厂获得，并且未指定铸造厂，则选择任意铸造厂。如果家庭不可用，将使用以下方式设置家庭font matching算法。

这将以逗号分隔家庭字符串并调用setFamilies() 与结果列表。要保留名称中使用逗号的字体，请使用带有QStringList。

可以参阅 Weight,setFamily(),setPointSize(),setWeight(),setItalic(),setStyleHint(),setFamilies （） ， 和QGuiApplication::font()。

[explicit]QFont::QFont(const QStringList &families, int pointSize = -1, int weight = -1, bool italic = false)

使用指定的构造一个字体对象*families*,*pointSize*,*weight*和*italic*设置。

如果*pointSize*为零或负数时，字体的磅值将设置为与系统相关的默认值。一般来说，这是12点。

每个姓氏条目*families*还可以可选地包括铸造厂名称，例如“Helvetica [Cronyx]”。如果该系列可从多个代工厂获得，并且未指定代工厂，则选择任意一家代工厂。如果家庭不可用，将使用以下方式设置家庭font matching算法。

可以参阅Weight,setPointSize(),setWeight(),setItalic(),setStyleHint(),setFamilies（） ， 和QGuiApplication::font()。

*QFont::QFont(const QFont &font, const QPaintDevice *pd)*

构造一个字体*font*用于油漆设备*pd*。

QFont::QFont(const QFont &font)

构造一个字体，它是*font*。

QFont::~QFont()

销毁字体对象并释放所有分配的资源。

bool QFont::bold() const

返回`true`如果weight() 的值大于QFont::Medium; 否则返回`false`.

可以参阅weight(),setBold（），和QFontInfo::bold()。

QFont::Capitalization QFont::capitalization() const

返回字体当前的大写类型。

可以参阅setCapitalization()。

QString QFont::defaultFamily() const

返回与当前样式提示对应的系列名称。

可以参阅StyleHint,styleHint（），和setStyleHint()。

bool QFont::exactMatch() const

返回`true`是否有与该字体设置完全匹配的窗口系统字体可用。

可以参阅QFontInfo。

QStringList QFont::families() const

返回请求的字体系列名称，即最后设置的名称setFamilies() 调用或通过构造函数。否则它返回一个空列表。

可以参阅setFamily(),setFamilies(),family(),substitutes（），和substitute()。

QString QFont::family() const

返回请求的字体系列名称。这将始终与第一个条目相同families（） 称呼。

可以参阅setFamily(),substitutes(),substitute(),setFamilies（），和families()。

bool QFont::fixedPitch() const

返回`true`是否已设置固定音调；否则返回`false`.

可以参阅setFixedPitch（） 和QFontInfo::fixedPitch()。

bool QFont::fromString(const QString &descrip)

设置此字体以匹配描述*descrip*。描述是以逗号分隔的字体属性列表，由以下命令返回toString()。

可以参阅toString()。

QFont::HintingPreference QFont::hintingPreference() const

返回使用此字体呈现的字形的当前首选提示级别。

可以参阅setHintingPreference()。

[static]void QFont::insertSubstitution(const QString &familyName, const QString &substituteName)

刀片*substituteName*进入家庭替代表*familyName*。

替换字体后，通过销毁并重新创建所有字体来触发字体的更新QFont对象。

可以参阅insertSubstitutions(),removeSubstitutions(),substitutions(),substitute（），和substitutes()。

[static]void QFont::insertSubstitutions(const QString &familyName, const QStringList &substituteNames)

插入家庭列表*substituteNames*进入替换列表*familyName*。

替换字体后，通过销毁并重新创建所有字体来触发字体的更新QFont对象。

可以参阅insertSubstitution(),removeSubstitutions(),substitutions（），和substitute()。

bool QFont::isCopyOf(const QFont &f) const

`true`如果此字体和则返回/是彼此的副本，即其中一个是作为另一个的副本创建的，并且此后都没有被修改过。这比平等严格得多。

可以参阅operator=（） 和operator==()。

bool QFont::italic() const

返回`true`如果style() 的字体不是QFont::StyleNormal

可以参阅setItalic （） 和style()。

bool QFont::kerning() const

返回`true`使用此字体绘制文本时是否应使用字距调整。

可以参阅setKerning()。

QString QFont::key() const

返回字体的键，即字体的文本表示。它通常用作字体缓存或字典的键。

可以参阅QMap。

qreal QFont::letterSpacing() const

返回字体的字母间距。

可以参阅setLetterSpacing(),letterSpacingType （） ， 和setWordSpacing()。

QFont::SpacingType QFont::letterSpacingType() const

返回用于字母间距的间距类型。

可以参阅letterSpacing(),setLetterSpacing （） ， 和setWordSpacing()。

bool QFont::overline() const

`true`如果已设置上划线则返回；否则返回`false`.

可以参阅setOverline()。

int QFont::pixelSize() const

如果设置了字体，则返回字体的像素大小setPixelSize()。如果大小设置为 -1，则返回 -1setPointSize（） 或者setPointSizeF()。

可以参阅setPixelSize(),pointSize(),QFontInfo::pointSize（），和QFontInfo::pixelSize()。

int QFont::pointSize() const

返回字体的磅值。如果字体大小以像素为单位指定，则返回 -1。

可以参阅setPointSize（） 和pointSizeF()。

qreal QFont::pointSizeF() const

返回字体的磅值。如果字体大小以像素为单位指定，则返回 -1。

可以参阅pointSize(),setPointSizeF(),pixelSize(),QFontInfo::pointSize（），和QFontInfo::pixelSize()。

[static]void QFont::removeSubstitutions(const QString &familyName)

删除所有替换*familyName*。

可以参阅insertSubstitutions(),insertSubstitution(),substitutions（），和substitute()。

QFont QFont::resolve(const QFont &other) const

返回一个新的QFont其属性复制自*other*之前尚未对此字体进行设置。

void QFont::setBold(bool enable)

如果*enable*为 true 将字体粗细设置为QFont::Bold; 否则将权重设置为QFont::Normal。

用于更精细的粗体控制使用setWeight()。

注：如果styleName() 设置后，该值可能会被忽略，或者如果平台支持，则将字体人为加粗。

可以参阅bold（） 和setWeight()。

void QFont::setCapitalization(QFont::Capitalization caps)

将此字体中文本的大小写设置为*caps*。

字体的大写使文本以选定的大写模式显示。

可以参阅capitalization()。

void QFont::setFamilies(const QStringList &families)

设置字体的家族名称列表。名称不区分大小写，并且可能包含铸造厂名称。第一个家庭在*families*将被设置为字体的主要系列。

每个姓氏条目*families*还可以可选地包括铸造厂名称，例如“Helvetica [Cronyx]”。如果该系列可从多个代工厂获得，并且未指定代工厂，则选择任意一家代工厂。如果家庭不可用，将使用以下方式设置家庭font matching算法。

可以参阅family(),families(),setFamily(),setStyleHint（），和QFontInfo。

void QFont::setFamily(const QString &family)

设置字体的家族名称。该名称不区分大小写，并且可能包含铸造厂名称。

这*family*名称还可以可选地包括铸造厂名称，例如“Helvetica [Cronyx]”。如果*family*可以从多个铸造厂获得，并且未指定铸造厂，则选择任意铸造厂。如果家庭不可用，将使用以下方式设置家庭font matching算法。

可以参阅family(),setStyleHint(),setFamilies(),families（），和QFontInfo。

void QFont::setFixedPitch(bool enable)

如果*enable*为 true，设置固定音高；否则设置固定螺距。

可以参阅fixedPitch（） 和QFontInfo。

void QFont::setHintingPreference(QFont::HintingPreference hintingPreference)

将字形提示级别的首选项设置为*hintingPreference*。这是对底层字体渲染系统使用一定级别的提示的提示，并且跨平台具有不同的支持。请参阅文档中的表格QFont::HintingPreference更多细节。

默认提示首选项是QFont::PreferDefaultHinting。

可以参阅hintingPreference()。

void QFont::setItalic(bool enable)

设置style() 的字体QFont::StyleItalic如果*enable*是真的; 否则样式设置为QFont::StyleNormal。

注：如果styleName() 设置后，该值可能会被忽略，或者如果平台支持，则字体可能会倾斜呈现，而不是选择设计的斜体字体变体。

可以参阅italic（） 和QFontInfo。

void QFont::setKerning(bool enable)

如果满足以下条件，则启用此字体的字偶距调整：*enable*是真的; 否则禁用它。默认情况下，启用字距调整。

启用字距调整后，字形度量不再相加，即使对于拉丁文本也是如此。换句话说，宽度（'a'）+宽度（'b'）等于宽度（“ab”）的假设不一定成立。

可以参阅kerning（） 和QFontMetrics。

void QFont::setLetterSpacing(QFont::SpacingType type, qreal spacing)

将字体的字母间距设置为*spacing*以及间距类型*type*。

字母间距更改字体中各个字母之间的默认间距。字母之间的间距可以按字符宽度的百分比或以像素为单位变小或变大，具体取决于所选的间距类型。

可以参阅letterSpacing(),letterSpacingType（），和setWordSpacing()。

void QFont::setOverline(bool enable)

如果*enable*为 true，则设置上划线；否则关闭上划线。

可以参阅overline（） 和QFontInfo。

void QFont::setPixelSize(int pixelSize)

将字体大小设置为*pixelSize*像素，最大尺寸为无符号 16 位整数。

使用此函数使字体设备相关。使用setPointSize（） 或者setPointSizeF() 以与设备无关的方式设置字体大小。

可以参阅pixelSize()。

void QFont::setPointSize(int pointSize)

将点大小设置为*pointSize*。点大小必须大于零。

可以参阅pointSize（） 和setPointSizeF()。

void QFont::setPointSizeF(qreal pointSize)

将点大小设置为*pointSize*。点大小必须大于零。可能无法在所有平台上达到所要求的精度。

可以参阅pointSizeF(),setPointSize（） ， 和setPixelSize()。

void QFont::setStretch(int factor)

设置字体的拉伸系数。

拉伸因子与字体的压缩或扩展版本相匹配，或者应用拉伸变换来更改字体中所有字符的宽度*factor*百分。例如，设置*factor*到 150 会导致字体中的所有字符加宽 1.5 倍（即 150%）。最小拉伸因子为 1，最大拉伸因子为 4000。默认拉伸因子为`AnyStretch`，它将接受任何拉伸因子并且不对字体应用任何变换。

拉伸因子仅适用于轮廓字体。位图字体的拉伸因子被忽略。

注意：当将字体与本机非默认拉伸因子匹配时，请求拉伸 100 会将其拉伸回中等宽度字体。

可以参阅stretch（） 和QFont::Stretch。

void QFont::setStrikeOut(bool enable)

如果*enable*为真，则设置三振；否则将取消三振。

可以参阅strikeOut（） 和QFontInfo。

void QFont::setStyle(QFont::Style style)

将字体样式设置为*style*。

可以参阅style(),italic（），和QFontInfo。

void QFont::setStyleHint(QFont::StyleHint hint, QFont::StyleStrategy strategy = PreferDefault)

将样式提示和策略设置为*hint*和*strategy*，分别。

如果没有明确设置，样式提示将默认为`AnyStyle`，样式策略将默认为`PreferDefault`。

Qt 不支持 X11 上的样式提示，因为窗口系统不提供此信息。

可以参阅StyleHint,styleHint(),StyleStrategy,styleStrategy（），和QFontInfo。

void QFont::setStyleName(const QString &styleName)

将字体的样式名称设置为*styleName*。设置后，其他样式属性如style（）和weight() 将在字体匹配时被忽略，但如果平台的字体引擎支持的话，之后可能会模拟它们。

由于人工模拟的样式质量较低，且缺乏完整的跨平台支持，因此不建议将样式名称匹配与样式属性匹配结合使用

可以参阅styleName()。

void QFont::setStyleStrategy(QFont::StyleStrategy s)

将字体的样式策略设置为*s*。

可以参阅styleStrategy（）和QFont::StyleStrategy。

void QFont::setUnderline(bool enable)

如果*enable*为 true，则设置下划线；否则关闭下划线。

可以参阅underline（）和QFontInfo。

void QFont::setWeight(QFont::Weight weight)

将字体粗细设置为*weight*，使用由下式定义的尺度QFont::Weight枚举。

注：如果styleName() 被设置后，该值在字体选择时可以被忽略。

可以参阅weight（） 和QFontInfo。

void QFont::setWordSpacing(qreal spacing)

将字体的字间距设置为*spacing*。

单词间距更改各个单词之间的默认间距。正值将字间距增加相应数量的像素，而负值则相应地减小字间距。

单词间距不适用于书写系统，在书写系统中，单个单词不被空格分隔。

可以参阅wordSpacing（） 和setLetterSpacing()。

int QFont::stretch() const

返回字体的拉伸因子。

可以参阅setStretch()。

bool QFont::strikeOut() const

`true`如果已设置删除线则返回；否则返回`false`.

可以参阅setStrikeOut()。

QFont::Style QFont::style() const

返回字体的样式。

可以参阅setStyle()。

QFont::StyleHint QFont::styleHint() const

返回StyleHint。

样式提示会影响font matching algorithm。看QFont::StyleHint查看可用提示的列表。

可以参阅setStyleHint(),QFont::StyleStrategy， 和QFontInfo::styleHint()。

QString QFont::styleName() const

返回请求的字体样式名称。这可用于匹配具有不规则样式的字体（无法在其他样式属性中标准化）。

可以参阅setStyleName(),setFamily（），和setStyle()。

QFont::StyleStrategy QFont::styleStrategy() const

返回StyleStrategy。

风格策略影响font matching算法。看QFont::StyleStrategy查看可用策略的列表。

可以参阅setStyleStrategy(),setStyleHint（），和QFont::StyleHint。

[static]QString QFont::substitute(const QString &familyName)

返回在任何时候使用的第一个姓氏*familyName*已指定。查找不区分大小写。

如果没有替代品*familyName*,*familyName*被返回。

要获取替换列表，请使用substitutes()。

可以参阅setFamily(),insertSubstitutions(),insertSubstitution（），和removeSubstitutions()。

[static]QStringList QFont::substitutes(const QString &familyName)

返回要在任何时候使用的姓氏列表*familyName*已指定。查找不区分大小写。

如果没有替代品*familyName*，返回一个空列表。

可以参阅substitute(),insertSubstitutions(),insertSubstitution（），和removeSubstitutions()。

[static]QStringList QFont::substitutions()

返回替换姓氏的排序列表。

可以参阅insertSubstitution(),removeSubstitutions（），和substitute()。

void QFont::swap(QFont &other)

将此字体实例交换为*other*。这个功能非常快并且永远不会失败。

QString QFont::toString() const

返回字体的描述。描述是以逗号分隔的属性列表，非常适合在QSettings，并由以下部分组成：

- 字体系列
- 点大小
- 像素大小
- 风格提示
- 字体粗细
- 字体样式
- 强调
- 三振出局
- 固定节距
- 始终为*0*
- 大写
- 字母间距
- 字间距
- 拉紧
- 风格策略
- 字体样式（无时省略）

可以参阅fromString()。

bool QFont::underline() const

`true`如果下划线已设置则返回；否则返回`false`.

可以参阅setUnderline()。

QFont::Weight QFont::weight() const

返回字体的粗细，使用与QFont::Weight枚举。

可以参阅setWeight(),Weight， 和QFontInfo。

qreal QFont::wordSpacing() const

返回字体的字间距。

可以参阅setWordSpacing（） 和setLetterSpacing()。

QVariant QFont::operator QVariant() const

返回字体为QVariant

bool QFont::operator!=(const QFont &f) const

`true`如果此字体与以下字体不同，则返回*f*; 否则返回`false`.

如果两个 QFont 的字体属性不同，则认为它们不同。

可以参阅operator==()。

bool QFont::operator<(const QFont &f) const

提供此字体与字体的任意比较*f*。所保证的是，`false`如果两种字体相等，则运算符返回；如果字体不相等，则 (f1 < f2) == !(f2 < f1) 。

此功能在某些情况下很有用，例如如果您想使用QFont对象作为 a 中的键QMap。

可以参阅operator==(),operator!=（） ， 和isCopyOf()。

QFont &QFont::operator=(const QFont &font)

分配*font*到此字体并返回对其的引用。

QFont &QFont::operator=(QFont &&other)

移动分配*other*对此QFont实例。

bool QFont::operator==(const QFont &f) const

`true`如果此字体等于则返回*f*; 否则返回 false。

如果两个 QFont 的字体属性相等，则它们被视为相等。

可以参阅operator!=（） 和isCopyOf()。

相关非会员

size_t qHash(const QFont &font, size_t seed = 0)

返回哈希值*font*。如果指定的话，*seed*用于初始化哈希。

QDataStream &operator<<(QDataStream &s, const QFont &font)

写入字体*font*到数据流*s*。（toString() 写入文本流。）

可以参阅Format of the QDataStream operators。

QDataStream &operator>>(QDataStream &s, QFont &font)

读取字体*font*从数据流中*s*。（fromString() 从文本流中读取。）

可以参阅Format of the QDataStream operators。