Location API 文档

Location

Location API 这个库提供定位，地标管理，地图，导航等功能。本文翻译 Location Overview。

命名空间

QtMobility API 放在 QtMobility 的命名空间. 这是为以后Mobility APIs 集成到 Qt做准备. 参考Quickstart guide中的例子体会这个命名空间如何影响使用QtMobility的开发development.

地理定位

地理数据包括地球表面精确的坐标——经纬度，以及以下一些数据：

- 获取位置的时间
- 设备的速度
- 海拔高度
- 与正北方的方位角，单位degree

这些数据可以通过一系列方法获得。最有名的就是GPS（Global Positioning System），这个系统通过接收卫星信号计算接收者的准确位置和时间。另一个流行的方法是CELL-ID技术是目前最简单的定位技术，它的原理是通过获取目标手机所在的蜂窝小区ID来确定其所在的位置，提供给定位用户。 这两种定位方法以及其他定位方法都可以在Location API中使用， Location API 需要的数据源的格式要求必须提供 经纬度坐标和日期/时间 上面的其他数据可以选择性提供。

创建位置数据源时继承QGeoPositionInfoSource， 通过QGeoPositionInfoSource::positionUpdated()放送信号signal，信号包含QGeoPositionInfo对象。调用startUpdates() or requestUpdate() 来触发位置数据的传递。

一些平台上可能有默认的位置数据源，调用QGeoPositionInfoSource::createDefaultSource() 创建一个默认位置数据源的实例，没有的话返回0。

类QGeoAreaMonitor 提供一个客户端程序，当接收设备进出一个特定区域的时候，得到通知。如果平台提过内置的区域监视功能，则QGeoAreaMonitor::createDefaultMonitor()返回一个默认区域显示器的实例。

类QGeoSatelliteInfoSource可以获得卫星信息。调用QGeoSatelliteInfoSource::createDefaultSource() 构造一个平台默认的卫星数据源的实例。客户端可以继承QGeoSatelliteInfoSource类来提供卫星数据源。

从数据源获取位置数据

连接connect数据源的 positionUpdated() 信号signal，调用startUpdates() 或requestUpdate()开始获得数据更新。

下面的例子显示了用QGeoPositionInfoSource::createDefaultSource()构造一个默认数据源，并从中获得位置数据。

```
class MyClass : public QObject
{
    Q_OBJECT
public:
    MyClass(QObject *parent = 0)
        : QObject(parent)
    {
        QGeoPositionInfoSource *source = QGeoPositionInfoSource::createDefaultSource(this);
        if (source) {
            connect(source, SIGNAL(positionUpdated(QGeoPositionInfo)),
                    this, SLOT(positionUpdated(QGeoPositionInfo)));
            source->setPreferredPositioningMethods(QGeoPositionInfoSource::NonSatellitePositioningMethods); // Make network positio
            source->startUpdates();
        }
    }

private slots:
    void positionUpdated(const QGeoPositionInfo &info)
    {
        qDebug() << "Position updated:" << info;
    }
};
```

设置数据源

QGeoPositionInfoSource::setUpdateInterval() 可以设定位置更新的频率。 例如调用 setUpdateInterval(30000)则程序每隔30秒刷新一次位置。如果没有设置更新的时间间隔，则setUpdateInterval() 默认设置0，数据源使用默认时间间隔，或应该提供其他内在的逻辑来决定何时更新。

QGeoPositionInfoSource::setPreferredPositioningMethods()使得程序能设定为使用某种定位方法。例如，如果程序想只使用卫星定位，这种定位在户外很准确，但是很费电，那么程序可以传入参数QGeoPositionInfoSource::SatellitePositioningMethods。这个方法应只在特殊用途的程序中使用。 大多数情况，不应改变默认的定位方法，因为一个数据源，内部实现可能使用了几种定位方法，这对程序很有用。

(C) Copyright Nokia 2013. All rights reserved.

读取NMEA数据

NMEA 是一种常见的导航数据文本协议。QNmeaPositionInfoSource使程序能实时或模拟的读取,传递NMEA数据。实时模式例如收取GPS数据流，模拟模式例如读取NMEA log文件。在模拟模式数据源依据NMEA语句的时间戳来发送更新信号。

例: 自定义位置数据源

通常QGeoPositionInfoSource::createDefaultSource()提供的默认位置数据源，或QNmeaPositionInfoSource 类提供的数据源总够为大部分应用提供位置数据。但是有时开发者希望使用自定义的位置数据源。

例子examples/logfilepositionsource中的LogFilePositionSource 类演示了如何继承QGeoPositionInfoSource 来自定义位置数据源。

这个例子从文本log.txt中读取位置数据。文本中的数据格式很简单，每行有日期/时间，经度，维度，空格隔开。日期/时间是 ISO 8601的格式，经纬度单位degree，十进制数字。例如

```
    2009-08-24T22:25:01 -27.576082 153.092415
    2009-08-24T22:25:02 -27.576223 153.092530
    2009-08-24T22:25:03 -27.576364 153.092648
```

类LogFilePositionSource 通过 positionUpdated()信号读取和发布数据。类定义如下

```
class LogFilePositionSource : public QGeoPositionInfoSource
{
    Q_OBJECT
public:
    LogFilePositionSource(QObject *parent = 0);

    QGeoPositionInfo lastKnownPosition(bool fromSatellitePositioningMethodsOnly = false) const;

    PositioningMethods supportedPositioningMethods() const;
    int minimumUpdateInterval() const;

public slots:
    virtual void startUpdates();
    virtual void stopUpdates();

    virtual void requestUpdate(int timeout = 5000);

private slots:
    void readNextPosition();

private:
    QFile *logFile;
    QTimer *timer;
    QGeoPositionInfo lastPosition;
};
```

继承的时主要重写了以下几个方法:

```
  * startUpdates(): 由客户端程序调用, 开始位置更新
  * stopUpdates():  由客户端程序调用, 结束位置更新
  * requestUpdate(): 由客户端程序调用, 请求单次更新, 有超时设置。
```

当一个位置更新有效时，这个子类发送positionUpdated()信号。

下面是关键方法的实现：

```
LogFilePositionSource::LogFilePositionSource(QObject *parent)
    : QGeoPositionInfoSource(parent),
      logFile(new QFile(this)),
      timer(new QTimer(this))
{
    connect(timer, SIGNAL(timeout()), this, SLOT(readNextPosition()));

    logFile->setFileName(QCoreApplication::applicationDirPath()
            + QDir::separator() + "simplelog.txt");
    if (!logFile->open(QIODevice::ReadOnly))
        qWarning() << "Error: cannot open source file" << logFile->fileName();
}

void LogFilePositionSource::startUpdates()
{
    int interval = updateInterval();
    if (interval < minimumUpdateInterval())
        interval = minimumUpdateInterval();

    timer->start(interval);
}

void LogFilePositionSource::stopUpdates()
{
    timer->stop();
}

void LogFilePositionSource::requestUpdate(int /*timeout*/)
{
    // For simplicity, ignore timeout - assume that if data is not available
    // now, no data will be added to the file later
    if (logFile->canReadLine())
        readNextPosition();
    else
        emit updateTimeout();
}

void LogFilePositionSource::readNextPosition()
{
    QByteArray line = logFile->readLine().trimmed();
    if (!line.isEmpty()) {
        QList<QByteArray> data = line.split(' ');
```


(C) Copyright Nokia 2013. All rights reserved.

```
            double latitude;
            double longitude;
            bool hasLatitude = false;
            bool hasLongitude = false;
            QDateTime timestamp = QDateTime::fromString(QString(data.value(0)), Qt::ISODate);
            latitude = data.value(1).toDouble(&hasLatitude);
            longitude = data.value(2).toDouble(&hasLongitude);

            if (hasLatitude && hasLongitude && timestamp.isValid()) {
                QGeoCoordinate coordinate(latitude, longitude);
                QGeoPositionInfo info(coordinate, timestamp);
                if (info.isValid()) {
                    lastPosition = info;
                    emit positionUpdated(info);
                }
            }
        }
    }
```

这个例子包含一个客户端程序的类 ClientApplication ， 这个类从LogFilePositionSource 请求位置更新。运行例子看看ClientApplication是否收到了数据。 运行例子前，你需要 make ， make install。

Location 例子

Flickr Demo
Flickr Demo 获取当前位置，然后从Flickr服务器下载这个地方图片的缩略图。

Weather Info Demo
Weather Info demo 显示本地天气信息。

Light Maps Demo
Light Maps demo 显示当前位置的街道地图。

地标

Location API 地标组件(Landmark API)负责从数据源中创建，恢复，更新，删除地标。

地标是指地理位置上的标志物，或感兴趣的位置点，表示成QLandmark 对象。相关的地标可以分组成不同的类别，如餐厅,旅馆，一个地标可以同时属于几个类别。

地标数据集用 QLandmarkManager来表示。QLandmarkManager 用于保存，获取，移除地标和地标类。我们可以定义不同的过滤条件排序方法来搜索地标，并规定结果的顺序。 QLandmarkManager 提供导入导出地标的方法，也提供当地标数据集改变时的通知服务。值得一提的是， Landmarks API提供同步和异步的机制来实现这些操作。同步操作由QLandmarkManager自己提供，异步操作由一系列请求类提供。强烈建议使用这些请求类，通常地标数据库很大，或者需要通过网络访问，这时使用异步机制可以防止程序无响应。

地标操作示例

关于Landmark的详细示例介绍见以下页面

- Creating/Saving categories and landmarks
- categories and landmarks
- Deleting categories and landmarks
- Importing landmarks
- Exporting landmarks

管理器和插件
地标管理器的名字与 实现管理器功能的plugin/backend对应，管理器名实际是域名字符串，如 "com.nokia.qt.landmarks.engines.symbian"。通常开发者不用处理管理器名，QLandmarkManager构造时不带参数，则初始时设定默认的管理器 。

要实现自己的插件，你需要继承QLandmarkManagerEngine ，实现虚函数，然后在创建一个QLandmarkMangerEngineFactory 的子类，这里实例化你的engine的实现。

下表显示各个平台的默认管理器

Manager name	Platform
com.nokia.qt.landmarks.engines.symbian	S60 3.1,3.2,5.0, Symbian^3
com.nokia.qt.landmarks.engines.sqlite	Maemo5, WindowsXp/Vista,Linux

导入,导出地标
Landmarks API 支持地标的导入导出。特定管理器支持的文件格式不同。下表列出各个平台默认管理器可能使用的格式. 你也可以通过QLandmarkManager::supportedFormats()命令来查询支持的文件格式.

Platform	Import formats	Export formats
S60 3.1,3.2	Lmx	Lmx
S60 5.0	Gpx, Lmx	Lmx
Symbian ^3	Gpx, Lmx, Kml, Kmz	Lmx
Maemo 5, windows, linux	Lmx, Gpx(version 1.1 only)	Lmx, Gpx(vesion 1.1 only)

导入操作时，输入的地标都会创建一个新地标来储存。而输入的类别会尽量映射到已有的类别上，当相同类别 找不到时才创建新类别储存。

导入导出操作中可以使用QLandmarkManager::TransferOption来控制导入的类别数据。（gpx格式不支持类别，传送选项被忽略）。通常默认选项是QLandmarkManager::IncludeCategoryData，这时任意类别被包含。使用QLandmarkManager::ExcludeCategoryData，则操作中忽略所有类别值。相反，在导入操作中，使用 QLandmarkManager::AttachSingleCategory 将把所有输入的地标存储在设定的类别中（地标中原设的类别被忽略）。

导出操作默认导出管理器中所有地标。如果想导出一部分，你可以通过提供一个地标id列表来实现。注意：gpx 格式的地标，不能导出经纬度值为NaN的地标。这时如果导出全部地标，则带NaN值的地标被略过，如果导出一部分地标，这时操作会失败。

地标Class

(C) Copyright Nokia 2013. All rights reserved.

主要的地标Class

QLandmark	地标，地图上感兴趣的点
QLandmarkCategory	标识一组相似类型的地标，如餐馆，旅馆，学校等类别
QLandmarkCategoryId	类别的唯一标识
QLandmarkId	地标的唯一标识
QLandmarkManager	存储检索地标的接口

地标选择Class

通过过滤器和排序类来选择地标。过滤器类设定匹配的条件，排序类设定显示的顺序。按距离排序使用QLandmarkProximity，在交集过滤器中使用 QLandmarkProximity可以使结果按距离升序排列（前提是只使用了默认排序方法，否则结果按设定的方法排）。过滤器可以在QLandmarkManager （同步检索）和request Class（异步检索）中都可以使用。

过滤器

QLandmarkAttributeFilter	过滤各种地标属性
QLandmarkBoxFilter	搜索指定范围中的地标
QLandmarkCategoryFilter	搜索指定类别的地标
QLandmarkFilter	过滤器基类，也是默认过滤器，用于检索所有地标
QLandmarkIdFilter	基于地标id的搜索
QLandmarkIntersectionFilter	组合过滤器，取搜索结果的交集
QLandmarkNameFilter	基于地标名的搜索
QLandmarkProximityFilter	在一个坐标为圆心一定半径的圆中搜索，结果按距离排序
QLandmarkUnionFilter	组合过滤器，去搜索结果的并集

排序

QLandmarkNameSort	按地标名排序
QLandmarkSortOrder	排序类的基类，也是默认排序类，输出结果不排序

异步请求

QLandmarkAbstractRequest	所有异步请求需继承的接口
QLandmarkCategoryFetchByIdRequest	使客户端从地标管理器异步地获取一列类别，按类别ID
QLandmarkCategoryFetchRequest	使客户端从地标管理器异步地获取一列类别
QLandmarkCategoryIdFetchRequest	使客户端从地标管理器异步地获取一列类别的ID
QLandmarkCategoryRemoveRequest	使客户端从地标管理器异步地删除一列类别
QLandmarkCategorySaveRequest	使客户端从地标管理器异步地保存一个特定类别
QLandmarkExportRequest	使客户端从地标管理器异步地导出一个地标集合
QLandmarkFetchByIdRequest	使客户端从地标管理器异步地获取一列地标，按地标id
QLandmarkFetchRequest	使客户端从地标管理器异步地获取一列地标
QLandmarkIdFetchRequest	使客户端从地标管理器异步地获取一列地标的id
QLandmarkImportRequest	使客户端从地标管理器异步地导入一个地标集合
QLandmarkRemoveRequest	使客户端从地标管理器异步地删除一个特定地标
QLandmarkSaveRequest	使客户端从地标管理器异步地保存一个特定地标

实现后端

一个管理器后端可通过继承QLandmarkManagerEngine来实现，并提供一个QLandmarkManagerEngineFactory 在需要的时候来实例化这个后端。

QLandmarkManagerEngine	地标管理器后端的抽象接口
QLandmarkManagerEngineFactory	实现QLandmarkManagerEngine功能的插件的抽象接口

地标例子

Landmark Browser

Landmark Browser 地标浏览器，演示了地标相关的类的使用。

地图和导航

地图和导航程序接口是插件的形式。因为很多地图，地理信息编码，路线信息的提供者不保证彼此的数据能共同使用。于是针对不同的服务提供者Qt Mobility使用不同的插件。通过QGeoServiceProvider来访问这些插件。Qt Mobility包含一个针对Nokia服务的插件。详见Nokia插件一章。

```
QGeoMappingManager *mappingManager = 0;
QGeoRoutingManager *routingManager = 0;
QGeoSearchManager *searchManager = 0;

QGeoServiceProvider serviceProvider("plugin name");

if (serviceProvider.error() == QGeoServiceProvider::NoError) {
     mappingManager = serviceProvider.mappingManager();
     routingManager = serviceProvider.routingManager();
     searchManager = serviceProvider.searchManager();
 }
```

公共类

QGeoBoundingArea	定义一个地理区域
QGeoBoundingBox	定义一个矩形地理区域
QGeoBoundingCircle	定义一个圆形地理区域
QGeoServiceProvider	提供对某个服务提供者的一系列地理信息的访问

地图

类QGraphicsGeoMap是显示地图和与地图交互的主要的类。这个类设计为使用 Graphics View 框架，因此也是QGraphicsWidget的子类。

类QGeoMappingManager 提供QGraphicsGeoMap需要的绝大多数方法。QGeoMappingManager 的细节只对插件的实现者来说是重要的，普通用户不需要在QGraphicsGeoMap 的构造函数以外使用QGeoMappingManager 。

```
      QGraphicsGeoMap *map = new QGraphicsGeoMap(mappingManager);
```

QGeoMapOverlay	用来在地图上叠加图示
QGeoMappingManager	对显示地图和与地图交互提供支持
QGraphicsGeoMap	用来显示地图并管理用户与地图的交互

地图对象

类QGeoMapObject和他的子类提供在地图上画图的功能，依坐标和距离。QGeoMapObject 对象可以被分成层级的组织在一起，这样可以方便的创建和管理一组对象。

QGeoMapCircleObject	QGeoMapObject， 画以某个坐标为圆心一定距离为半径的圆
QGeoMapGroupObject	QGeoMapObject，管理一组其他QGeoMapObject
QGeoMapObject	用来 在QGraphicsGeoMap实例中显示的图像元件，依赖坐标和距离
QGeoMapPixmapObject	QGeoMapObject， 用来在地图上画一幅pixmap

(C) Copyright Nokia 2013. All rights reserved.

QGeoMapPolygonObject	QGeoMapObject，用来在地图上画一个多边形
QGeoMapPolylineObject	QGeoMapObject ，用来在地图上画一条线段
QGeoMapRectangleObject	QGeoMapObject ，用来在地图上画一个矩形区域
QGeoMapRouteObject	QGeoMapObject，用来在地图桑画一条路线
QGeoMapTextObject	QGeoMapObject，用来在地图上显示文字

路线

类QGeoRoutingManager处理路线信息请求。

请求由QGeoRouteRequest实例创建，传递给QGeoRoutingManager::calculateRoute()，请求完成后返回的结果包含在QGeoRouteReply实例中。

类QGeoRoute用来描述路线结果。每条路线由若干QGeoRouteSegment实例组成，路线的分割点出现在用户指定的路径点或交通工具换乘点。

每个QGeoRouteSegment 包含一个QGeoNavigationInstruction 实例用来描述这段路线的说明以指导用户。说明中有QGeoRouteSegment线段端点的地址，以及到下一段QGeoRouteSegment的途径的文字说明。

QGeoManeuver	表示两段QGeoRouteSegments 相交点的相关信息
QGeoRoute	表示两点间的路线
QGeoRouteReply	管理QGeoRoutingManager实例发起的一个操作
QGeoRouteRequest	表示路线请求的参数和限制
QGeoRouteSegment	表示路线中的一段
QGeoRoutingManager	用来支持地理路线操作

地理信息编码和地名搜索

类QGeoSearchManager 处理地理信息编码，解释地理信息编码，支持对地名的文本搜索。

这里的文本搜索会对类似地名的文字进行地理编码，同时用地标数据库中提供的服务进行搜索。进一步可以在数据来源添加QLandmarkManager 实例，这样用户可以在线搜索地标数据库。

QGeoSearchManager	支持地理信息相关的搜索
QGeoSearchReply	管理由QGeoSearchManage实例发起的操作

Nokia 插件

Qt Mobility 自带一个地图导航编程接口的Nokia插件，用来访问Ovi提供的地理信息服务。这些服务由服务条款限制和保护，见plugins/geoservices/nokia/OVI_SERVICES_TERMS_AND_CONDITIONS.txt。

使用插件关键字"nokia"加载Ovi服务插件。

服务条款仅限制对Ovi地图服务的使用。并不限制其他可能包含在Qt Mobility 包中的其他地图及导航API的插件。

实现地图插件

插件需继承QGeoServiceProviderFactory类以及 插件想要提供的功能相应的ManagerEngine类。

继承QGeoServiceProviderFactory 只需交代一个名字和版本，通过重写 QGeoServiceProviderFactory::providerName() 和 QGeoServiceProviderFactory::providerVersion()来做。另外根据需要重写 QGeoServiceProviderFactory::createSearchManagerEngine(), QGeoServiceProviderFactory::createMappingManagerEngine() and QGeoServiceProviderFactory::createRoutingManagerEngine()。

QGeoMapData	作为 QGraphicsGeoMap和QGeoMappingManager之间的桥
QGeoMapObjectInfo	用来定义一部分QGeoMapObject 的基类，通常这些QGeoMapObject 跟QGeoMapData 的子类有关，QGeoMapObjectInfo的信息来协助构造。
QGeoMappingManagerEngine	对于想实现显示地图和地图交互的插件，提供与QGeoServiceProvider 交互的接口和适用的方法
QGeoRoutingManagerEngine	对于想提供地理路线信息访问服务的插件，提供与QGeoServiceProvider 交互的接口和适用的方法
QGeoSearchManagerEngine	对于想提供基于地理信息搜索的插件，提供与QGeoServiceProvider 交互的接口和适用的方法
QGeoServiceProviderFactory	这是地理信息相关的服务的插件接口的工厂类

碎片纹理(Tile-based)地图适用的类

大多数现有的碎片纹理地图API(tile based)都非常类似，所以Qt Mobility提供了一系列类使得构造碎片纹理地图API的插件更容易。

在使用墨卡托地图投影或其他地图拼图方案时，主要需要继承QGeoTiledMappingManagerEngine同时实现QGeoTiledMappingManagerEngine::getTileImage().

QGeoTiledMapData	继承自 QGeoMapData ，使得 当与 基于地图图像拼接的地图服务工作起来 更容易
QGeoTiledMapReply	用来管理QGeoTiledManagerEngine发起的获取地图图块的操作
QGeoTiledMapRequest	表示一个来自拼图地图的服务的请求
QGeoTiledMappingManagerEngine	提供服务，使得为拼图类地图服务写Qt地图和导航API插件更容易

地图导航例子

Map View

- Map Viewer 介绍如何使用QGraphicsGeoMap显示地图

Geo Service Demo

- Geo Service Demo 显示系统中可用的插件，介绍如何使用路线搜索

QML 支持

QML对Location API支持的细节 参看Location QML Plugin的文档。

注意：Qt Mobility 1.1.0a版本对QML下地图导航API的支持不完全，会在接下来的版本的优化改进，同时注意在Symbian（Qt Mobility 1.1.0），QML在Landmarks API会有异常表现，这是因为bug QTMOBILITY-611，在更新LandmarkAbstractModel元素时会有问题。

(C) Copyright Nokia 2013. All rights reserved.

地图QML例子

Landmark Map
- Landmark Map

Declarative Location Flickr
- Declarative Location Flickr

Declarative Map Viewer
- Declarative Map Viewer

相关链接

- Qt Mobility 开发
- Qt 开发
- Qt Quick (Chinese)

[[Link title]]

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.