在 Java 程序中使用定位接口

定位引擎对外提供的接口可以使用 TCP 1200 端口访问。
为了方便用户二次开发，我们使用 Java 编写了一个接口客户端(RTLEAPI)，用户可以在自己的 Java 程序中调用 RTLEAPI 提供的功能。

在你的 Java 工程中应该引用库 **JLRTLEAPI.jar**，这个 jar 文件就是接口客户端。

1. 接口的启动和停止

在使用接口之前，要先启动它。使用
`RTLEAPI.start();`
启动接口。

调用 start()方法，如果不加参数，API 连接的引擎的 IP 地址在 `RtleApiConfig.xml` 中指定。`RtleApiConfig.xml` 文件与 `JLRTLEAPI.jar` 文件在同一目录。

也可以使用 `RTLEAPI.start(String RTLEIPAddress)`来连接指定地址的引擎。例如 `RTLEAPI.start("192.168.0.1")`

在程序结束时，使用
`RTLEAPI.stop();`
停止接口。

这两个函数 start() 和 stop() 是 RTLEAPI 的静态函数。
start() 会创建一个线程，并创建一个到定位引擎的连接。连接成功后，接口程序会自动在后台与定位引擎同步数据，确保定位引擎中的数据与接口侧的数据一致。

stop() 会停止接口线程。

如果在 web 应用中使用定位接口，应该使用单例模式来使用它。RTLEAPI.start() 应该只被调用一次。
为了方便 web 应用开发，我们提供了一个 RTLEAPIServletContextListener 类，把它配置在 web 应用的 web.xml 中。例如：

```
...
  <listener>
    <listener-
class>com.jinglinsoft.rtle.api.RTLEAPIServletContextListener</listener-class>
  </listener>
...
```

这样，在 web 容器如 tomcat 中，会在启动 web 应用时自动调用 RTLEAPI.start()。
关于这一点，可以参考我们提供的管理平台的 web.xml 的配置。

2. 接口提供的类和对象

接口提供了区域、基站、时钟源、标签相关的类和对象。
应用程序可以通过接口来操纵这些对象。可以对这些对象进行创建、删除、读、写等操作。

我们建议应用程序不要创建或修改这些对象，而是使用我们提供的管理平台来创建和修改。因为定位系统的软件还在发展中，也许将来会有更新，应用程序创建这些对象，也许在将来软件版本升级后会出现数据一致性上的问题。

应用程序应该尽量把定位系统当作一个单向的系统，仅只从定位系统读取各类数据，不要修改定位系统中的各类数据。

2.1. Areas

Areas 类是所有区域的列表。可以使用 Areas 来获取系统中所有的区域。例如：

```
System.out.println("以下是区域列表");
System.out.println("-----------------------------------------------");
for (Area area : Areas.getAreaList().values()) {
    System.out.println(String.format("areaId=%s areaName=%s", area.getId(), area.getName()));
}
System.out.println("");
```

Areas 类还提供一些与区域相关的方法。

`String getAreaName(String areaId)`	根据区域 Id 获取区域名称
`Area get(String areaId)`	根据区域 Id 获取区域对象
`String getNewAreaId()`	在创建一个新区域前，使用这个 getNewAreaId()获取一个空的区域 Id
`add(Area area)`	添加一个区域到区域列表中

2.2. Area

Area 类定义了区域。
可以使用 Areas.getAreaList() 得到系统中已有区域列表。
区域是定位系统中的一个逻辑概念，并不真正对应具体的硬件设备，所以系统是无法自动创建的。如果需要创建一个新的区域，用户应该使用管理平台中的区域管理功能创建新的区域。

(不建议)应用程序也可以自己创建新的区域，如果新建区域，可以使用 new Area() 创建，新

区域的 Id 可以使用 Areas.getNewAreaId() 取得，应用程序有责任保证新建的区域的 Id 不要与已有区域的 Id 重复。通过 Areas.getNewAreaId() 取得的新区域 Id，系统可以确保这个新的 Id 不会与系统中已有的区域 Id 重复。
区域有各种属性，具体可以参见对应的 Javadoc 文档。

2.3. Anchors

Anchors 类是所有基站的列表。
可以通过 Anchors. getAnchorList() 取得所有基站的列表，这是一个 HashTable，遍历它就可以取得所有的 Anchor 对象。

2.4. Anchor

Anchor 类定义了基站，一个 Anchor 对象就是一个基站。
因为基站是一种硬件设备，当基站连接到定位引擎后，定位引擎会自动创建对应的 Anchor 对象，所以应用程序不需要创建新的基站。
一个 Anchor 可以使用 Anchors.remove()删除，但是因为基站是硬件设备，如果它与引擎是连接着的，一个 Anchor 对象被删除之后，定位引擎会为对应的硬件再次创建一个 Anchor。

2.5. ClockSources

ClockSources 类是所有时钟源的列表。
可以通过 ClockSoures.getClockSourceList() 取得所有的时钟源列表，这是一个 HashTable，遍历它就可以取得所有的 ClockSource 对象。

2.6. ClockSource

ClockSourece 类定义了时钟源。一个 ClockSource 对象就是一个时钟源。
因为时钟源是一种硬件设备，当时钟源连接到定位引擎后，定位引擎会自动创建对应的 ClockSource 对象，所以应用程序不需要创建新的时钟源。
一个 ClockSource 可以使用 ClockSources.remove()删除，但是因为时钟源是硬件设备，如果它与引擎是连接着的，一个 ClockSource 对象被删除之后，定位引擎会为对应的硬件再次创建一个 ClockSource。

2.7. Tags

Tags 类是所有标签的列表。
可以通过 Tags.getTagList() 取得所有的标签列表，这是一个 HashTable，遍历它就可以取得所有的 Tag 对象。

2.8. Tag

Tag 类定义了标签。一个 Tag 对象就是一个标签。
因为标签是一种硬件设备，当基站收到标签发出的定位数据包，并转发给定位引擎后，定位引擎会自动创建对应的 Tag 对象，所以应用程序不需要创建新的标签。
一个 Tag 可以使用 Tags.remove()删除，但是因为标签是硬件设备，如果引擎接收到该标签的新的消息后，定位引擎会为对应的标签再次创建一个 Tag。

3. 侦听定位事件

RTLEAPI 使用 Java 的观察者来实现定位相关事件。

在程序中定义一个观察者类，实现 Observer 接口，大致如下所示：

```
class EventWatcher implements Observer {
    @Override
    public void update(Observable o, Object arg) {
        if(arg instanceof RTLSEvent_TagMessage){
        }
        else if(arg instanceof RTLSEvent_TagLocated){
        }
        else {
        }
    }
}
```

创建这个观察者类的对象，并调用 RTLEAPI 提供的方法，把这个对象加入到观察者列表中，如下：

```
EventWatcher ew = new EventWatcher();
RTLEAPI.getRTLSEventMonitor().addObserver(ew); // 添加观察者
```

当定位系统发生事件时，会调用 ew.update() 方法。所以，程序要在 update() 方法中处理

接收到的事件。

update() 方法的第二个参数 Object arg 是我们要关心的。根据这个 arg 的类型判断事件类型。例如：

```
if(arg instanceof RTLSEvent_TagMessage){
    RTLSEvent_TagMessage event = (RTLSEvent_TagMessage)arg;
    System.out.print("Event RTLSEvent_TagMessage: ");
    System.out.println(String.format("\ttagId=%s seq=%d Battery=%.2fV",
                    event.tagMessage.getTagId(),
event.tagMessage.getSeq64(), event.tagMessage.getBatteryVoltage()));
}
```

这段代码判断收到的事件是不是标签消息(TagMessage)，如果是，则打印标签消息中的标签 Id、定位包序号、标签电池电压等。

3.1. RTLSEvent_TagMessage

在 update()中收到的参数如果是 RTLSEvent_TagMessage 类的对象，表示定位引擎收到了基站转发的标签定位数据包。
通常，每个标签都会不断的发出无线定位数据包，每一个定位数据包都有一个序号，这个序号每次增加 1。在标签附近的基站会接收到标签发出的定位数据包，基站把收到的定位数据包转发给定位引擎。对于某一个具体的标签发出的某一个序号的定位数据包，引擎会收到不同基站发来的同标签同序号数据包。例如：某个标签 tagId=0008DEFFFE00016D，发出一个数据包，序号 seq=1227。在这个标签附近有 6 个基站，这 6 个基站都会收到这个数据包。那么，定位引擎会收到 6 个数据包，分别来自不同的基站。

RTLSEvent_TagMessage 只在引擎收到第一个基站发来的数据包时发生，后续的其他基站发来相同标签相同序号的数据包时，不会再发生这个事件，也即是说对于同一标签同一序号的定位数据包，这个事件只发生一次。当标签发送下一个序号的定位数据包，引擎又会再次发生一次这个事件。

RTLSEvent_TagMessage 有两个成员：

```
    public TagMessage tagMessage;
    public double blinkInterval;
```

其中 tagMessage 是具体的消息内容，blinkInterval 是消息发送间隔，单位是毫秒。
blinkInterval 是计算出来的，定位引擎根据收到的定位数据包的情况计算这个标签每隔多长时间发一次定位数据包。

tagMessage 是 TagMessage 类的实例。有一些方法可以取到这个定位数据包的属性。

- `getTagId()` 取标签 Id
- `getSeq64()` 取定位包序号
- `getSwitchStatus()` 取标签上的按钮状态
- `getPowerVoltage()` 取电源电压

- `getBatteryVoltage()` 取电池电压
- `getLighteness()` 取环境光亮度(*工牌标签没有安装环境光检测硬件,此方法无效*)

标签上还集成有一个加速度传感器芯片，但是因为功耗方面的原因，目前的固件没有启用这个芯片。所以与加速度传感器相关的属性也是无效的。

3.2. RTLSEvent_TagLocated

如果 update()的参数是 RTLSEvent_TagLocated 类的对象，说明定位引擎完成了一次成功的定位。
标签不断的发出无线定位数据包，只有当引擎收到了足够解方程的数据包之后，才能解出方式。并且，可能会因为干扰之类的原因，导致计算出的坐标有比较大的偏差。定位引擎只有在计算出标签的坐标，并且判断这个坐标是正确的，才会输出。这时，就会发生 RTLSEvent_TagLocated 事件。

RTLSEvent_TagLocated 类只有一个成员：

```
public Tag tag;
```

应用程序可以访问 tag 对象标签的各种属性。当然最重要的是坐标和区域 Id。