请注意: 可能会因为软件版本更新导致本手册与软件有不致之处，请及参阅更新版本的手册

RTLE3 使用手册

V 1.03

贵阳联创智能网络科技有限公司

2021 年 3 月 22 日

目录

1. 概述

RTLE 是联创精确定位系统的定位引擎，是定位系统的核心，其主要的任务是接收基站发来的标签定位数据包，计算标签的坐标。
本使用手册仅适用于 RTLE 版本 3.x，本手册对老版本的软件不适用。

RTLE 的主要功能：

- 侦听 TCP 1200 端口，接受基站和时钟源的 TCP 链接，接收基站和时间源的各类消息
- 侦听 TCP 1201 端口，接受 TCP 接口程序的连接(例如 JLAPI.jar)，交互消息为二进制格式
- 侦听 TCP 1202 端口，接受 Restful 风格的 API 连接，以及 Websocket 连接
- 侦听 TCP 1203 端口，接受 TCP 接口程序的连接，交互消息为 json 格式文本

- 计算标签的坐标
- 处理区域切换
- 检测警戒区，触发警戒区相关事件

请注意：RTLE 3.x 版本不支持 ActiveMQ。对于之前使用 ActiveMQ 的老客户，请使用 TCP 文本接口或 Restful 接口与 RTLE 对接。这两种接口方式也使用 json 文本作为交互数据。我们不再支持 activeMQ 的原因是 activeMQ 的 C++依赖库太多，导致程序很臃肿。

1.1. 系统的目录结构

RTLE 可以放在硬盘上的任意目录。与 RTLE.exe 同级的目录下，有以下 3 个目录：

- config – 保存配置数据
- files – 保存用户文件
- logs – 保存日志

1.1.1. config

config 目录下总会有一个文件 RTLEConfig.db，这个是 sqlite3 格式的数据库文件。这个文件保存了 RTLE 有关的配置信息。这个文件不能修改，即使使用 sqlite3 数据库管理软件对其进行修改，也有可能因其中的数据不合法，导致定位引擎运行出错。
如果 RTLE 运行时，没有找到 RTLEConfig.db，它会创建一个新的 RTLEConfig.db，一些基本

参数按缺省的设置。

可能还会有一个文件 config.ini，这个文件是附加的配置文件。可以手工修改。
这个文件负责配置一些不适合记录在 RTLEConfig.db 中配置参数。
如果 RTLE 启动时没有找到 config.ini，它不会自动创建 config.ini。

1.1.2. files

files 目录下是用户文件。
从本质上来说，RTLE 只是一个定位系统，计算标签坐标。
一般来说，应用程序拿到坐标数据后，总要以某种方式显示出来，最直观的方式是以地图的方式显示。为了显示地图，需要为客户端提供平面图之类的底图。
为此，RTLE 提供了简单的文件管理功能，以减少客户端的工作量。

1.1.3. logs

logs 目录记录系统日志，日志以天为单位生成单独的文件。

1.2. 程序的运行方式

在 Windows 下，RTLE 有两种运行方式：作为 Windows 服务运行；作为命令行程序运行。

1.2.1. Windows 服务

如果作为 Windows 服务运行，当计算机启动后，用户不需要登录到桌面。RTLE 会作为 Windows 服务在后面运行。推荐生产环境使用这种方式运行。

把 RTLE 安装为 Windows 服务，使用以下命令：

```
RTLE /install
```

把 RTLE 从 Windows 服务器卸载，使用以下命令：

```
RTLE /uninstall
```

1.2.2. 命令行程序

如果作为命令行程序运行，用户需要登录到 Windows 桌面，手工运行 RTLE；或者把 RTLE 配置到启动项中，登录桌面后立即自动运行。这种方式不建议在生产环境中使用，只适合临时测试时使用。

1.3. 命令行

通常，RTLE 在运行的时候不需要使用命令行参数。在大多数时候，需要的参数都事先保存在配置文件中了，RTLE 直接从配置文件中加载参数。

为了方便使用，RTLE 也允许部分参数的以命令行参数的方式进行配置。命令行参数仅在以命令行程序运行时才有效。

请注意：RTLE 作为服务运行时，不支持命令行参数。

1.4. 可能存在的端口冲突

通常，RTLE 需要侦听 TCP 1200、1201、1202、1203 端口，以及 UDP 1200 端口。
如果其他程序占用了这些端口，在启动 RTLE 时，RTLE 绑定端口失败，会导致 RTLE 无法启动。
提示：在 RTLE 的开发过程中我们发现，微信的 Windows 桌面版、千牛工作台等程序，有时会占用这几个端口。建议把 RTLE 标签为 Windows 服务运行，以保证 RTLE 在这些程序之前启动。

如果启动 RTLE 时报错，提示绑定端口失败，可以在 Windows 命令窗口下，使用 `netstat` 命令检查占用端口的程序。

例如

```
C:\Users\Zhang>netstat -a -n -o | grep 1202
File STDIN:
  TCP    127.0.0.1:1202         127.0.0.1:1203         ESTABLISHED     6952
  TCP    127.0.0.1:1203         127.0.0.1:1202         ESTABLISHED     6952
```

上面的提示表示 ID 为 6952 的进程占用了 1202/1203 端口。打开 Windows 任务管理器，可以查看这个进程的可执行文件名，或者直接结束这个进程。

1.5. 应用程序接口

重要提醒：所有的对外接口中，RTLE 都使用 UTF-8 表示汉字。应用程序如果使用 GBK 编码与 RTLE 交互，将会出现乱码。

RTLE 的向应用程序提供的数据有两类：静态的配置信息 和 动态的事件消息。
静态的配置信息主要是指系统内部各类配置信息，例如有区域信息、基站信息、标签信息等。这些信息大多是由用户配置的，如区域的定义等；部分是系统自己生成的，如基站的 IP 地址、MAC 地址等等。
动态的事件消息主要是指系统发生的各类事件，例如：定位成功事件、电子围栏进出事件等等。

RTLE 提供了以下多种方式的接口：TCP 二进制消息接口、RESTful 风格 http 接口+websocket、TCP 文本消息接口、TCP 自定义二进制消息接口、串行文本消息接口。

目前静态配置信息只以下面两种方式提供访问：TCP 二进制接口、RESTful 风格 http 接口。其他方式的接口目前只提供单向的输出功能，不支持输入，所以支持动态消息的发送。

1.5.1. TCP 二进制消息接口

RTLE 二进制消息接口侦听 TCP 1201 端口，通过这个端口与客户端进行二进制数据包的交互。
我们最初设计这个接口，是为了适应与多种语言的对接。随着系统的发展，接口变得越来越复杂，二进制接口的数据包类型有数百种之多，无论是使用哪种语言来实现客户端，都是一个艰难的任务。

我们最初实现了这个二进制接口的 Java 的客户端，打包成一个 jar 文件，并使用 Java 来开发 RTLE 的管理控制台，以方便对 RTLE 进行管理和配置。

1.5.2. RESTful 风格接口

因为二进制接口的日益复杂，让客户的开发人员直接使用会很麻烦。所以，我们新提供了两种新的接口：RESTful 风格的 http 接口+Websocket，和 TCP 文件消息接口。这两种接口都使用 json 文本格式来进行数据交互。
json 几乎是自解释的数据格式，并且格式灵活，有足够的表达能力和扩充性。Json 唯一的缺点是因为期使用文本格式，所以数据量比较大，效率较低。但是在小型系统中，这算不上什么大问题。

RESTful 风格 http 接口其实就是作为一个 http 服务器提供给客户端访问，供客户端访问 RTLE

的内部数据。但是 http 协议是被动式的(请求-响应式)。如果需要主动推送消息，有两种实现方式：SSE(Server Side Event)和 websocket。其实 SSE 就是客户端与服务器维持一个长连接，服务器不断发送新的数据给客户端。相比之下，websocket 提供的消息传递机制更灵活一些，它可以双向传送数据。虽然我们目前只提供单向的消息传送，由服务器发送消息给应用程序，但是将来可以扩展。
RTLE 的 http 服务侦听 1202 端口。

1.5.3. TCP 文本消息接口

TCP 文本消息接口侦听 TCP 1203 端口，它与 websocket 接口很相似。都是使用 json 格式的文本消息进行数据传送。目前 TCP 文本消息接口也只提供单向的消息传送，由服务器发送消息给应用程序。

1.5.4. TCP 自定义二进制消息接口

有时客户的系统会在一些特殊的限制，例如有些工控系统没有 double 类型，或者有些客户希望得到较短的消息，以节省网络带宽加快数据传输速度等等。为此，我们提供了自定义二进制消息接口。
缺省情况下，TCP 自定义二进制消息接口侦听 TCP 1204 端口，它与系统提供的标准的 TCP 二进制消息接口很相似，不同之处在于消息的格式是由客户指定的。

1.5.5. 串行文本消息接口

串行文件消息接口使用 RS232 串口进行数据输出。串行文本消息接口与 TCP 文本消息接口很相似，都是使用 json 格式的文本消息进行数据传送。目前串行文本消息接口只提供单向的消息传送，由服务器通过 RS232 串口发送消息给应用程序。

2. 配置文件 config.ini 格式

在 RTLE.exe 同级目录下有一个子目录 config，这个 config 目录中有一个文件叫 config.ini，。既然 RTLE 的配置参数已经保存在数据库文件 RTLEConfig.db 中了，为什么还要另外搞一个文件 config.ini 呢?
RTLEConfig.db 是 sqlite3 格式的数据库文件，修改这个文件需要专门的 sqlite3 编辑软件。并且，RTLEConfig.db 中的很多配置参数有固定的格式，如果用户直接手工修改，不小心可能会弄乱格式。所以，**我们不建议用户手工修改 RTLEConfig.db 文件**。
RTLE 的一些参数，有时需要手工修改，以提供更灵活的配置，config.ini 这个文件就是提供给用户手工修改的配置文件。

Config.ini 中的配置参数的格式：
每行一个参数
格式为 <参数名称>=<参数值>

如果参数名称无效，对系统没有影响。如果在文件中需要使用注释，通常在行首加#号，表明这行是注释。其实，因为不会有参数名称以#开始，所以所有的注释本质上都是无效的参数或参数值，它不会对系统产生影响。

如果 RTLE 启动时，config.ini 不存在，RTLE 会对相关的参数使用缺省值。如果 config.ini 存在，RTLE 会加载 config.ini 中的配置作为相关参数的值。

2.1. API 用户名和密码

API 用户名和密码在 config.ini 中定义或修改。API 用户名和密码的缺省值都是“`rtleapi`”。
API 用户名的参数名为 api_username，API 密码的参数名为 api_password。
例如：

```
api_username=test_username
api_password=test_pass
```

以上两行修改 API 用户名为 `test_username`，修改 API 密码为 `test_pass`。

2.2. 日志配置

当 RTLE 作为控制台程序运行时，会有一个控制台窗口。控制台窗口中会显示 RTLE 运行过程中产生的日志。当 RTLE 作为服务运行时，就没有控制台窗口了。
无论如何，RTLE 运行过程中产生的日志，都会保存到日志文件中。RTLE 的日志文件保存在 RTLE.exe 所在目录下的子目录 logs 中，以自然日为单位，每天一个文件。

RTLE 的日志有 6 个级别(off 为关闭，不算日志)：

trace	跟踪日志
debug	调试日志
info	信息日志
warn	警告日志
error	错误日志
critical	致使错误日志
off	关闭日志

其中 trace 最详细, critical 最简略。日志级别从上到下之间是包含关系，例如 trace 为最详细的日志，它除了显示 trace 类型的日志之外，还显示其他所有类型的日志。
系统缺省为 warn 级别(输出 warn/error/eritical 这三种类型的日志)。

日志级别的参数名称为 log_level。例如：

log_level=trace
配置日志级别为 trace，输出所有的日志。

又如：
log_level=warn
配置日志级别为 warn，输出警告、错误和致命错误三种类型的日志，而跟踪、调试、信息等三种类型的日志则不输出。

3. Java 接口

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。

本质上讲，Java 接口是一个中转接口，或者说是翻译的接口。
RTLE 侦听 TCP 1201 端口，通过这个端口与客户端进行二进制数据包的交互。Java 接口作为客户端，连接到 RTLE 的 TCP 1201 端口，与 RTLE 进行交互。应用程序只需要访问 Java 接口中提供的类、对象即可。

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。

3.1. 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。

3.2. 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。

3.2.1. 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

3.2.2. 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。

3.2.3. 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。

3.2.4. 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。

3.2.5. 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。

3.2.6. 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。

3.2.7. 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。

3.2.8. 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。

3.3. 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。

3.3.1. 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。

3.3.2. 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。

4. Restful 接口

RTLE 支持 Restful 风格的接口。
在进行应用程序与 RTLE 间的接口开发时，建议使用 **`curl`** 进行测试。以下的示例都是使用 curl 为例。

通常，Restful 风格的接口，服务器端是一个标准的 web 服务器，应用程序作为客户端，向

服务器端请求数据或提交数据。交换的数据以 json 文本构成。

当应用程序与服务器端交互时，会把资源作为 URL 的一部分，把方法作为操作。

例如：
`http://localhost:1202/anchors` 这个 URL 代表本地服务器的端口 1202 的一个资源，/anchors 表示我们需要的资源是 anchors。如果向服务器使用 GET 方法请求这个 URL，表示请求基站列表，。
`curl http://localhost:1202/anchors` ← 这个命令将返回全部基站的信息
`curl http://localhost:1202/anchors/0008DEFFFE000537` ← 这个命令将返回 ID 为 `0008DEFFFE000537` 的基站的信息

请注意，与服务器交互的数据中如果有中文，使用 Unicode 表示，而不是 GBK。
如果在 Windows 的命令窗口下使用 curl 进行测试，请确认窗口使用的代码是 65001(UTF-8)，而不是 936(ANSI/OEM – 简体中文 GBK)。可以使用命令 `chcp 65001` 切换代码页为 UTF-8。

```
curl -H "Content-Type: application/json" -X POST -d "{\"user_id\":
\"123\", \"coin\":100, \"success\":1, \"msg\":\"OK!\" }"
```

4.1. 接口规范

4.1.1. 动词

GET（SELECT）：从服务器取出资源（一项或多项）。
POST（CREATE）：在服务器新建一个资源。
PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源或部分需要修改的属性）。
PATCH（UPDATE）：与 PUT 方法作用完全相同
DELETE（DELETE）：从服务器删除资源。

4.1.2. 返回码

200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。

202 Accepted - [*]：表示一个请求已经进入后台排队（异步任务）

204 NO CONTENT - [DELETE]：用户删除数据成功。

400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

401 Unauthorized - [*]：表示用户没有权限（令牌、用户名、密码错误）。

403 Forbidden - [*] 表示用户得到授权（与 401 错误相对），但是访问是被禁止的。

404 NOT FOUND - [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。

406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求 JSON 格式，但是只有 XML 格式）。

410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。

500 INTERNAL SERVER ERROR - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

4.1.3. 返回结果

GET /collection：返回资源对象的列表（数组）
GET /collection/resource：返回单个资源对象
POST /collection：返回新生成的资源对象
PUT /collection/resource：返回完整的资源对象
PATCH /collection/resource：返回完整的资源对象
DELETE /collection/resource：返回一个空文档

4.2. 基站列表 /anchors

应用程序向服务器请求基站列表，使用 `/anchors` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/anchors` 即可取得基站列表。

可以在路径后加 /<anchor id> 来获取指定基站的数据。例如 `http://localhost:1202/anchors/0008DEFFFE000626` 返回 anchorId 为 `0008DEFFFE000626` 的基站的数据。

例如，获取系统中全部基站的信息：

```
C:\Users\Zhang>curl http://localhost:1202/anchors
```

例如，获取 AnchorId 为 **0008DEFFFE000626** 的基站的信息：

```
C:\Users\Zhang>curl http://localhost:1202/anchors/0008DEFFFE00040B
{
  "enable": true,
  "serialNo": "20191222193357",
  "id": "0008DEFFFE00040B",
  "name": "2222",
  "x": 1.0,
  "y": 1.0,
  "z": 2.6,
  "comments": "",
  "mac": "0008DE00040B",
  "mainAnchorId": "0000000000000000",
  "uwbModuleNum": 1,
  "ip": "0.0.0.0",
  "hwModel": 5,
  "hwVersion": "3.1",
  "fwVersion": "3.31",
  "panId": "0xC5D5",
  "rtleAutoDiscover": true,
  "rtleIP": "192.168.99.2",
  "rtlePort": 1200,
  "autoCloseConnection": true,
  "closeConnectionTimeout": 60,
  "sendAliveInterval": 2,
  "receiveClockMessageFromSamePanIdOnly": true,
  "receiveRangeMessageFromSamePanIdOnly": true,
  "csFilter": 1,
  "csIgnoreRatio": 20,
  "receiveSpecifyClockSourceOnly": true,
  "specifyClockSources": [
    "0008DEFFFE00040C"
  ],
  "clockSyncInterval": 200,
  "ethAutoGetIP": true,
  "ethStaticIP": "192.168.99.10",
  "ethSubnet": "255.255.255.0",
  "ethGateway": "192.168.99.1",
  "reportCSSyncPacket": true,
  "useWatchdog": true,
  "reportRangeMessageWithLocalTime": true,
  "uwbChannel": 2,
  "uwbDataRate": "BR_6M8",
  "uwbPRF": "PRF_64M",
  "uwbTxPreambLength": "PLEN_1024",
  "uwbRxPAC": 3,
  "uwbTxCode": 9,
  "uwbRxCode": 9,
  "uwbNSSFD": false,
  "uwbSfdTO": 4161,
  "uwbPhrMode": "PHRMODE_EXT",
  "uwbSmartPowerEn": true
}
```

基站属性表

属性名称	类型	JSON 类型	是否只读	说明	例子
enable	boolean	boolean		是否允许基站。如果值为 true 表示基站正常工作，如果为 false 表示禁用基站。	"enable": true
serialNo	string	string	只读	序列号。唯一不重复。	"serialNo": "20191222193357"

id	EUI64	string	只读	基站标识。唯一不重复。	"id": "0008DEFFFE00040B"
name	string	string		基站名称。可修改，系统不关心，主要用于人工识别。	"name": "东北角基站"
x	double	number		基站 X 坐标，单位：米	"x": 1.0
y	double	number		基站 Y 坐标，单位：米	"y": 2.0
z	double	number		基站 Z 坐标，单位：米	"z": 2.6
comments	string	string		备注。系统不关心，供人工查看。	"comments": "备注说明"
mac	EUI48	string	只读	基站的 MAC 地址	"mac": "0008DE00040B"
mainAnchorId	EUI64	string		主基站 ID。如果这个属性值为"0000000000000000"，表示这是一个主基站。	"mainAnchorId": "0000000000000000"
uwbModuleNum	integer	number	只读	基站内部 UWB 模块数量	"uwbModuleNum": 1
ip	string	string	只读	基站的 IP 地址	"ip": "192.168.9.168"
hwModel	integer	number	只读	基站的型号	"hwModel": 5
hwVersion	string	String	只读	基站硬件版本号	"hwVersion": "3.1"
fwVersion	string	string	只读	基站固件版本号	"fwVersion": "3.31"
panId	uint16	string		PANID	"panId": "0xC5D5"
rtleAutoDiscover	boolean	Boolean		是否自动发现定位引擎	"rtleAutoDiscover": true
rtleIP	string	string		定位引擎的 IP 地址	"rtleIP":"192.168.99.2"
rtlePort	integer	number		定位引擎的端口号	"rtlePort":1200
autoCloseConnection	boolean	boolean		长时间未收到数据是否自动断开网络连接	"autoCloseConnection":true
closeConnectionTimeout	integer	number		自动断开网络连接的超时时间，单位：秒	"closeConnectionTimeout":60
sendAliveInterval	integer	number		发送心跳包间隔，单位：秒	"sendAliveInterval":2
receiveClockMessageFromSamePanIdOnly	boolean	boolean		是否仅接收来自相同 PANID 的时钟同步消息	"receiveClockMessageFromSamePanIdOnly":true
receiveRangeMessageFromSamePanIdOnly	boolean	boolean		是否仅接收来自相同 PANID 的测距消息	"receiveRangeMessageFromSamePanIdOnly":true
csFilter	integer	number		是否使用滤波器。0表示不使用，1 表示使用	"csFilter":1
csIgnoreRatio	integer	number		时钟同步数据忽略率，单位：%。当时钟偏差超过 n%时认时钟数据不正确，忽略之	"csIgnoreRatio":20
receiveSpecifyClockSourceOnly	boolean	boolean		只接收指定的时钟源的数据	"receiveSpecifyClockSourceOnly":true

specifyClockSources	array	array		指定的时钟源的 ID 的数组	"specifyClockSources":["0008DEFFFE00040C"]
clockSyncInterval	integer	number		作为时钟源时，时钟同步数据包发送间隔。单位：毫秒	"clockSyncInterval":200
ethAutoGetIP	boolean	boolean		以太网是否自动获取 IP 地址	"ethAutoGetIP":true
ethStaticIP	string	string		以太网指定的静态 IP 地址	"ethStaticIP":"192.168.99.10"
ethSubnet	string	string		以太网指定的子网掩码	"ethSubnet":"255.255.255.0"
ethGateway	string	string		以太网指定的网关	"ethGateway":"192.168.99.1"
reportCSSyncPacket	boolean	boolean		向定位引擎报告收到的时钟同步数据包	"reportCSSyncPacket":true
useWatchdog	boolean	boolean		是否使用看门狗。如果使用，当看门狗检测到基站的各个子程序长时间没有反应时会自动重启基站	"useWatchdog":true
reportRangeMessageWithLocalTime	boolean	boolean		是否使用本地时间报告测距消息。用于 0 维定位(标签存在性检测)	"reportRangeMessageWithLocalTime":true
uwbChannel	integer	number		UWB 频道，有效值：1、2、3、4、5、7	"uwbChannel":2
uwbDataRate	enum	string		UWB 数据通讯速率：BR_6M8 、 BR_850K 、 BR_110K ， 分别对应 6.8Mbps、850Kbps、110Kbps	"uwbDataRate":"BR_6M8"
uwbPRF	enum	string		UWB 脉冲重复频率，有效值：PRF_16M、PRF_64M	"uwbPRF":"PRF_64M"
uwbTxPreambLength	enum	string		UWB 发送前导码长度，有效值：PLEN_64、PLEN_128、PLEN_256 、 PLEN_512 、 PLEN_1024 、 PLEN_1536 、 PLEN_2048、PLEN_4096。发送前导码越长，接收端越容易识别，但会发送端会花费更多的时间。	"uwbTxPreambLength":"PLEN_1024"
uwbRxPAC	integer	number		UWB 接收前导码采集块大小，有效值：PAC8、PAC16、PAC32、PAC64。 建议值：PAC_8 对应前导码长度 128 及以下、PAC_16 对应前导码长度 256、PAC_32 对应前导码长度 512 、PAC_64 对应前导码长度 1024 及以上。	"uwbRxPAC":"PAC64"
uwbTxCode	integer	number		UWB 发送使用的前导码	"uwbTxCode":9
uwbRxCode	integer	number		UWB 接收使用的前导码	"uwbRxCode":9
uwbNSSFD	boolean	boolean		是否使用非标签 SFD	"uwbNSSFD":false

uwbSfdTO	integer	number		SFD 超时值	"uwbSfdTO":4161
uwbPhrMode	enum	string		PHY 头模式，有效值：PHRMODE_STD 、PHRMODE_EXT。通常使用 PHRMODE_EXT	"uwbPhrMode":"PHRMODE_EXT"
uwbSmartPowerEn	boolean	boolean		是否使用智能功率控制	"uwbSmartPowerEn":true

4.3. 时钟源列表 /clockSources

应用程序向服务器请求时钟源列表，使用 `/clockSources` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/clockSources` 即可取得全部时钟源列表。

可以在路径后加 /<clockSourceId> 来获取指定时钟源的数据。例如 `http://localhost:1202/clockSources/ 0008DEFFFE0001B5` 返回 clockSourceId 为 `0008DEFFFE0001B5` 的时钟源的数据。

例如，获取系统中全部时钟源的信息：

```
C:\Users\Zhang>curl http://localhost:1202/clockSources
[{
  "serialNo": "20190823154033",
  "id": "0008DEFFFE0001A0",
  "name": "LG-时钟",
  "x": -2.086792428538827,
  "y": 2.3886095620538774,
  "z": 2.9,
  "comments": "",
  "mac": "0008DE0001A0",
  "uwbModuleNum": 1
},
{
  "serialNo": "20190925111748",
  "id": "0008DEFFFE0001B5",
  "name": "py5-时钟",
  "x": -14.075708566309434,
  "y": 6.726446578962981,
  "z": 2.6,
  "comments": "",
  "mac": "0008DE0001B5",
  "uwbModuleNum": 1
},
{
  "serialNo": "20200224161430",
  "id": "0008DEFFFE0004E2",
  "name": "演示基站 4 副基站",
  "x": 1.0,
  "y": 1.0,
  "z": 2.6,
  "comments": "",
  "mac": "0008DE0004E2",
  "uwbModuleNum": 1
}]
```

例如，获取 ClockSourceId 为 **0008DEFFFE0001B5** 的时钟源的信息：

```
C:\Users\Zhang>curl http://localhost:1202/clockSources/0008DEFFFE0001B5
{
  "serialNo": "20190925111748",
  "id": "0008DEFFFE0001B5",
  "name": "py5-时钟",
  "x": -14.075708566309434,
  "y": 6.726446578962981,
  "z": 2.6,
  "comments": "",
  "mac": "0008DE0001B5",
  "uwbModuleNum": 1
}
```

4.4. 区域列表 /areas

应用程序向服务器请求区域列表，使用 **/areas** 作为路径。例如在本地服务器上访问 **curl http://localhost:1202/areas** 即可取得区域列表。

可以在路径后加 /<areald> 来获取指定区域的数据。例如 **http://localhost:1202/areas/0008D10000000001** 返回 areald 为 **0008D10000000001** 的区域的数据。

例如，获取系统中全部区域的信息：

```
C:\Users\Zhang>curl http://localhost:1202/areas
[{
  "enable": true,
  "id": "0008D10000000000",
  "name": "我的测试区域",
  "clockSourceId": "0008DEFFFE0001B5",
  "comments": "测试",
  "defaultZ": 1.11,
  "polygon": {
    "vertexes": [
      {
        "x": -18.21506664819029,
        "y": 1.533093552888531
      },
      {
        "x": -18.36836395992635,
        "y": 9.869286498912452
      },
      {
        "x": -10.511265308567463,
        "y": 9.65848628570967
      },
      {
        "x": -10.453781111124517,
        "y": 1.4181115414742282
      }
    ]
  },
  "discardPointsOfOutOfArea": true,
  "locationTrigger": 1,
  "specifyAnchorsId": [
    "0008DEFFFE0000CD",
    "0008DEFFFE0001B0",
    "0008DEFFFE00005C"
  ],
```

```
  "useRTLEFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 10,
  "kalmanFilterLevel": 10,
  "datumMarks": []
},
{
  "enable": true,
  "id": "0008D10000000001",
  "name": "走廊",
  "clockSourceId": "0008DEFFFE0001A0",
  "comments": "",
  "defaultZ": 1.0,
  "polygon": {
    "vertexes": [
      {
        "x": -10.228068170844884,
        "y": 1.7546577888798163
      },
      {
        "x": -10.244749902461688,
        "y": 3.256122470274138
      },
      {
        "x": 5.5873600812598205,
        "y": 3.239439530248615
      },
      {
        "x": 5.637409564838767,
        "y": 1.7379748456904032
      }
    ]
  },
  "discardPointsOfOutOfArea": true,
  "locationTrigger": 1,
  "specifyAnchorsId": [
    "0008DEFFFE0004E2",
    "0008DEFFFE00019F"
  ],
  "useRTLEFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 100,
  "datumMarks": []
}]
```


例如，获取 Areald 为 **0008D10000000001** 的区域的信息：

```
C:\Users\Zhang>curl http://localhost:1202/areas/0008D10000000001
```


```
{
  "enable": true,
  "id": "0008D10000000001",
  "name": "走廊",
  "clockSourceId": "0008DEFFFE0001A0",
  "comments": "",
  "defaultZ": 1.0,
  "polygon": {
    "vertexes": [
      {
        "x": -10.228068170844884,
        "y": 1.7546577888798163
      },
      {
        "x": -10.244749902461688,
        "y": 3.256122470274138
      },
      {
        "x": 5.5873600812598205,
        "y": 3.239439530248615
      },
      {
```

```
        "x": 5.637409564838767,
        "y": 1.7379748456904032
      }
    ]
  },
  "discardPointsOfOutOfArea": true,
  "locationTrigger": 1,
  "specifyAnchorsId": [
    "0008DEFFFE0004E2",
    "0008DEFFFE00019F"
  ],
  "useRTLEFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 100,
  "datumMarks": []
}
```


4.5. 标签列表 /tags

应用程序向服务器请求标签列表，使用 `/tags` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/tags` 即可取得标签列表。

可以在路径后加 /<tagId> 来获取指定标签的数据。例如 `http://localhost:1202/tags/ 0008DEFFFE000590` 返回 tagId 为 `0008DEFFFE000590` 的标签的数据。

例如，获取系统中全部标签的信息：

```
C:\Users\Zhang>curl http://localhost:1202/tags
```


```
[{
  "enable": true,
  "id": "0008DEFFFE000590",
  "tagTypeId": "0008D20000000000",
  "name": "tag-0008DEFFFE000590",
  "comments": "tag-0008DEFFFE000590 added @ 2020-09-23 14:59:13",
  "icon": "",
  "useTagTypeFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 50,
  "useTagTypeTagOnlineThreshold": true,
  "tagOnlineThreshold": 5000,
  "useTagTypeVoteTrigger": true,
  "voteTriggerOnSeqDifferent": true,
  "voteTriggerOnNewSeqCalculated": true,
  "voteTriggerOnCalculatedDelayTime": 50,
  "outPutRangeMessage": false,
  "paramList": {}
}]
```


例如，获取 tagId 为 `0008DEFFFE000590` 的标签的信息：

```
C:\Users\Zhang>curl http://localhost:1202/tags/0008DEFFFE000590
```


```
{
  "enable": true,
  "id": "0008DEFFFE000590",
  "tagTypeId": "0008D20000000000",
  "name": "tag-0008DEFFFE000590",
  "comments": "tag-0008DEFFFE000590 added @ 2020-09-23 14:59:13",
```

```
  "icon": "",
  "useTagTypeFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 50,
  "useTagTypeTagOnlineThreshold": true,
  "tagOnlineThreshold": 5000,
  "useTagTypeVoteTrigger": true,
  "voteTriggerOnSeqDifferent": true,
  "voteTriggerOnNewSeqCalculated": true,
  "voteTriggerOnCalculatedDelayTime": 50,
  "outPutRangeMessage": false,
  "paramList": {}
}
```

4.6. 标签类型列表 /tagTypes

应用程序向服务器请求标签类型列表，使用 `/tagTypes` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/tagTypes` 即可取得标签类型列表。

可以在路径后加 /<tagTypeId> 来获取指定标签类型的数据。例如 `http://localhost:1202/tagTypes/ 0008D20000000000` 返回 tagTypeId 为 `0008D20000000000` 的标签类型的数据。

例如，获取系统中全部标签类型的信息：

```
C:\Users\Zhang>curl http://localhost:1202/tagTypes
[{
  "id": "0008D20000000000",
  "name": "基本标签",
  "comments": "缺省的标签类型，由 RTLE 自动创建",
  "defaultIcon": "defaultIcon.png",
  "useRTLEFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 50,
  "useRTLETagOnlineThreshold": true,
  "tagOnlineThreshold": 5000,
  "useRTLEVoteTrigger": true,
  "voteTriggerOnSeqDifferent": true,
  "voteTriggerOnNewSeqCalculated": true,
  "voteTriggerOnCalculatedDelayTime": 50
}]
```

例如，获取 tagTypeId 为 `0008D20000000000` 的标签类型的信息：

```
C:\Users\Zhang>curl http://localhost:1202/tagTypes/0008D20000000000
{
  "id": "0008D20000000000",
  "name": "基本标签",
  "comments": "缺省的标签类型，由 RTLE 自动创建",
  "defaultIcon": "defaultIcon.png",
  "useRTLEFilter": true,
  "useAverageFilter": true,
  "useKalmanFilter": true,
  "averageFilterSampleTimeLength": 1000,
  "kalmanFilterLevel": 50,
  "useRTLETagOnlineThreshold": true,
  "tagOnlineThreshold": 5000,
  "useRTLEVoteTrigger": true,
```

```
  "voteTriggerOnSeqDifferent": true,
  "voteTriggerOnNewSeqCalculated": true,
  "voteTriggerOnCalculatedDelayTime": 50
}
```


4.7. 底图列表 /basemaps

应用程序向服务器请求底图列表，使用 `/basemaps` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/basemaps` 即可取得底图列表。

可以在路径后加 /<basemapId> 来获取指定底图的数据。例如 `http://localhost:1202/basemaps/0008DEFFFE000590` 返回 basemapId 为 `0008DEFFFE000590` 的底图的数据。

例如，获取系统中全部底图的信息：


```
C:\Users\Zhang>curl http://localhost:1202/basemaps
[{
  "id": "0008D30000000006",
  "name": "测试底图 1",
  "comments": "测试底图 1",
  "basemapType": 0,
  "basemapBitmapFilename": "office2019101401.png",
  "centerX": 106.6280289,
  "centerY": 26.6220112,
  "opacity": 0.0,
  "rotate": 270.0,
  "scaleX": 0.0074,
  "scaleY": 0.0073
}]
```


例如，获取 basemapId 为 `0008D30000000006` 的底图的信息：


```
C:\Users\Zhang>curl http://localhost:1202/basemaps/0008D30000000006
{
  "id": "0008D30000000006",
  "name": "测试底图 1",
  "comments": "测试底图 1",
  "basemapType": 0,
  "basemapBitmapFilename": "office2019101401.png",
  "centerX": 106.6280289,
  "centerY": 26.6220112,
  "opacity": 0.0,
  "rotate": 270.0,
  "scaleX": 0.0074,
  "scaleY": 0.0073
}
```


4.8. 楼层列表 /floors

应用程序向服务器请求楼层列表，使用 `/floors` 作为路径。例如在本地服务器上访问 `curl http://localhost:1202/floors` 即可取得楼层列表。

可以在路径后加 /<floorId> 来获取指定楼层的数据。例如 `http://localhost:1202/`

basemaps/0008D40000000008 返回 floorId 为 0008D40000000008 的楼层的数据。

例如，获取系统中全部楼层的信息：

```
C:\Users\Zhang>curl http://localhost:1202/floors
[{
  "id": "0008D40000000000",
  "name": "地面",
  "comments": "默认楼层",
  "ground": true,
  "basemapIds": [],
  "areaIds": []
},
{
  "id": "0008D40000000008",
  "name": "测试楼层一",
  "comments": "测试简写",
  "ground": true,
  "basemapIds": [
    "0008D30000000006"
  ],
  "areaIds": [
    "0008D10000000000",
    "0008D10000000001"
  ]
}]
```

例如，获取 floorId 为 0008D40000000008 的楼层的信息：

```
C:\Users\Zhang>curl http://localhost:1202/floors/0008D40000000008
{
  "id": "0008D40000000008",
  "name": "测试楼层一",
  "comments": "测试简写",
  "ground": true,
  "basemapIds": [
    "0008D30000000006"
  ],
  "areaIds": [
    "0008D10000000000",
    "0008D10000000001"
  ]
}
```

4.9. 警戒区(电子围栏)列表 /guardAreas

应用程序向服务器请求警戒区列表，使用 /guardAreas 作为路径。例如在本地服务器上访问 curl http://localhost:1202/guardAreas 即可取得警戒区列表。

可以在路径后加 /<guardAreaId> 来获取指定警戒区的数据。例如 http://localhost:1202/ guardAreas/0008D50000000008 返回 guardAreaId 为 0008D50000000008 的警戒区的数据。

例如，获取系统中全部警戒区的信息：

```
C:\Users\Zhang>curl http://localhost:1202/guardAreas
[{
  "enable": true,
  "id": "0008D50000000000",
  "name": "电子围栏 2",
```

```
  "comments": "电子围栏测试",
  "floorId": "0008D40000000008",
  "triggerMode": 1,
  "detectQueueTimeLength": 1000,
  "inActionMinPointsNum": 3,
  "outActionMinPointsNum": 3,
  "polygon": {
    "vertexes": [
      {
        "x": -18.310880848774264,
        "y": 4.40764346174316
      },
      {
        "x": -18.23422440775059,
        "y": 5.65328155303846
      },
      {
        "x": -16.432840154476157,
        "y": 5.672445213961443
      },
      {
        "x": -16.394514559320022,
        "y": 4.33098880539567
      }
    ]
  }
},
{
  "enable": true,
  "id": "0008D50000000008",
  "name": "测试警戒区二",
  "comments": "测试警戒区一",
  "floorId": "0008D40000000008",
  "triggerMode": 1,
  "detectQueueTimeLength": 1000,
  "inActionMinPointsNum": 3,
  "outActionMinPointsNum": 3,
  "polygon": {
    "vertexes": [
      {
        "x": -12.592389400400243,
        "y": 8.414852416043109
      },
      {
        "x": -13.550571294200445,
        "y": 9.353871617763614
      },
      {
        "x": -10.71435012411528,
        "y": 9.373035273940761
      },
      {
        "x": -10.758980134593735,
        "y": 8.846406396790318
      },
      {
        "x": -10.810169366966853,
        "y": 8.242379494629335
      }
    ]
  }
}]
```


例如，获取 guardAreald 为 **0008D50000000008** 的警戒区的信息：

```
C:\Users\Zhang>curl http://localhost:1202/guardAreas/0008D50000000008
```


```
{
  "enable": true,
  "id": "0008D50000000008",
  "name": "测试警戒区二",
  "comments": "测试警戒区一",
  "floorId": "0008D40000000008",
  "triggerMode": 1,
```

```
  "detectQueueTimeLength": 1000,
  "inActionMinPointsNum": 3,
  "outActionMinPointsNum": 3,
  "polygon": {
    "vertexes": [
      {
        "x": -12.592389400400243,
        "y": 8.414852416043109
      },
      {
        "x": -13.550571294200445,
        "y": 9.353871617763614
      },
      {
        "x": -10.71435012411528,
        "y": 9.373035273940761
      },
      {
        "x": -10.758980134593735,
        "y": 8.846406396790318
      },
      {
        "x": -10.810169366966853,
        "y": 8.242379494629335
      }
    ]
  }
}
```


4.10. Websocket 接口

RTLE 使用 Websocket 接口作为事件通讯用途。
当 RTLE 系统发生一些事件时，会通过 websocket 把该事件通知到已经建立 websocket 连接的全部客户端。

Websocket 连接的路径是 `/ws`，协议是 `ws`。例如，连接本地服务器上的 RTLE 上的连接 URL 是：`ws://localhost:1202/ws`

当 websocket 连接建立后，RTLE 会向客户端不断发送 json 文本消息。

为了让客户端容易区分消息，我们使用回车消息间分隔符，每条消息的内容都是在一行内；每条消息都是一次性发送，并且每次只发送一条消息，使每条消息间有一个时间上的间隔。

每条消息都会有一个字符串类型的字段 `messageType`，表示消息的类型。

4.10.1. 定位成功消息 TagLocated

RTLE 对标签定位成功，会发出一条定位成功消息。

4.10.2. 标签测距消息 TagRangeMessage

标签只要有电，就会持续发出 UWB 定位数据包，每一个 UWB 定位数据包都会有一个序列号(32 位无符号整数)，这个序列号是递增的。如果标签复位，这个序列号会重新从 0 开始递增。这个序列号我们称为 seq32。
标签发出的 UWB 定位数据包会被标签附近的基站收到。
本消息就是基站收到标签发出的 UWB 定位消息后，转发给 RTLE 的，消息中会有一个 anchorId 字段表示是哪个基站收到的，还会有一个 clockSourceId 字段表示该基站与哪个时钟源同步。

请注意：本消息准确的称呼应该叫“标签定位消息”。但是，从语义上看，“标签定位消息”容易与“标签定位成功消息”混淆。为了能够明显的区分两种类型的消息，我们称之为“标签测距消息”。本系统使用 TDOA，计算标签坐标使用的是时间差，按电波在空气中的传送速度转换为距离后，或称距离差；我们不使用标签到基站之间的距离，不需要测距。

对于某个特定的标签发出的某个特定的序列号的 UWB 定位数据包，可能会被多个基站收到，这些基站又可能会与多个时钟源同步时钟，所以 RTLE 会从基站那里收到很条 tagId 和 seq32 都相同的测距消息。
有志于研究 TDOA 计算标签坐标的开发人员，可以开发自己的算法，使用本消息计算标签的坐标。

4.10.3. 标签消息 TagMessage

本消息与“标签测距消息”相似，可以理解为是去重之后的测距消息。
它是 tagId 和 seq32 相同的很多消息中的第一条。

5. TCP 文本消息接口

RTLE 在 TCP 端口 1203 提供了 json 格式的文本消息接口。
当应用程序连接到 RTLE 的 TCP 端口 1203 后，RTLE 会不断发送 json 格式的文本消息给应用程序。
这些消息与 Websocket 消息相似。

例如，在 Windows 命令行窗口下使用 `telnet localhost 1203` 命令，可以连接到本地服务器的 tcp 1203 端口，窗口中就会不断显示 telnet 收到的 json 格式的文本消息。

消息的格式和类型，请参考 Websocket 接口部分的介绍。

6. TCP 自定义二进制消息接口

缺省情况下，TCP 自定义二进制消息接口侦听 TCP 1204 端口，它与系统提供的标准的 TCP 二进制消息接口很相似，不同之处在于消息的格式是由客户指定的。

6.1. 配置参数

TCP 自定义二进制消息接口相关的参数在 config.ini 中配置。以下是相关的配置项目的说明：

6.1.1. api_custom_enable

`api_custom_enable` 项目定义是否允许使用自定义消息接口，缺省是不允许。如果要使用自定义消息接口，该项目配置为 `true`；如果配置为 `false` 表示不允许。
注意：如果不使用自定义消息接口，请配置该项目为 false。因为处理数据需要消耗系统资源，配置为 false 禁止该接口，可以节约一些系统资源。

缺省配置：

```
api_custom_enable=false
```

6.1.2. api_custom_port

`api_custom_port` 项目配置自定义接口使用的 TCP 端口。系统缺省使用 1024 端口，如果不是很有必要，建议不要修改。

缺省配置：

```
api_custom_port=1204
```

6.1.3. api_custom_client_ip_limited

`api_custom_client_ip_limited` 项目配置限制使用自定义接口的客户端 IP。为了保证数据安全，只允许指定的 IP 地址连接到服务器。如果配置为空，则不限制连接的 IP。

缺省配置:

```
api_custom_client_ip_limited=127.0.0.1
```

6.1.4. api_custom_out_messages

`api_custom_out_messages` 项目配置自定义接口输出的消息类型。
目前可以输出的消息类型有以下 3 种:
`tag_located` 标签定位成功消息
`tag_range` 标签发送的原始消息(目前暂不支持)
`clock_sync` 时钟同步消息(目前暂不支持)

缺省配置:

```
api_custom_out_messages=tag_located
```

6.1.5. api_custom_message_format_tag_located

`api_custom_message_format_tag_located` 项目定义标签定位成功消息的格式。
消息由多个字段组成，用户可以使用常数(整数)或系统预定义的变量作为字段。**字段之间使用逗号“,”作为分隔符**。

常数字段

常数使用方括号“[”、“]”括起来，每一个常数的大小是一个字节，能表达的值是十进制的 0~255，或者是十六进制的 0x00~0xFF。如果需要表达更大的整数，可以使用多个常数字段组合。常数字段中使用的整数，可以使用十进制数字，或者八进制或十六进制。与 C 语言对常数的表示方式类似，数字前导 0 表示是八进制，前导 0x 表示十六进制，第一个数字非 0，表示是十进制。常数字段通常用于定义数据包头，用于识别数据；或者用于定义数据包的长度；或者用于定义数据包的类型等。请注意：常数字段只支持正整数，不支持负数，也不支持浮点数。

变量字段

系统预定义了一些变量，可以在格式定义中直接使用:

校验和: `u8_crc` 是 1 字节无符号整数，表示从该字节之后开始的数据包的 CRC8 的值

标签标识: 系统中标签标识 tagId 长度是 64 位二进制,即 8 个字节,用 `u64_tag_id` 表示。

为适应不同类型的应用系统,我们提供了截断只保留低位的标签短标识，`u16_tag_id` 表示低 2 字节 tagId, `u32_tag_id` 表示低 4 字节的 tagId

区域标识: 系统中区域标识 areaId 长度是 64 位二进制,即 8 个字节,用 `u64_area_id` 表

示。为适应不同类型的应用系统,我们提供了截断只保留低位的区域短标识，`u16_area_id` 表示后 2 字节 areald，`u32_area_id` 表示后 4 字节的 areald

定位消息序号：系统中定位消息序号是 4 字节无符号整数，用 `u32_seq` 表示。也可以用 `u16_seq` 表示低 2 字节，`u8_seq` 表示低 1 字节

x 坐标：系统中 x 坐标使用 64 位双精度浮点类型表示。为方便应用程序侧处理,我们提供以下变量表示：`i16cm_x` 代表 2 字节有符号整数单位为厘米的 X 坐标值(表达范围为-655.36 米~+655.35 米)，`i32cm_x` 代表 4 字节有符号整数单位为厘米的 X 坐标值
yz 坐标的表示与 x 坐标类似，分别使用 `i16cm_y`，`i32cm_y` 和 `i16cm_z`，`i32cm_z` 来表示

按钮状态：1 字节，`u8_switch`，按钮状态的第 7 位(最高位)表示警报状态，第 0 位(最低位)表示报警按钮是否被按下

电池电压：1 字节，`u8_battery_voltage`，单位为 0.1 伏，即该值除以 10 之后得到伏为单位的电压值，例如 39 表示 3.9V

充电电压：1 字节，`u8_power_voltage`，单位为 0.1 伏，即该值除以 10 之后得到伏为单位的电压值，例如 39 表示 3.9V

如果要输出多种类型的消息,需要一个字节来区别消息类型,对此,系统不做规定,由客户自行定义一个常数作为消息类型。

以下是一个例子：

```
api_custom_message_format_tag_located=[0x55],[0xAA],[20],u8_crc,[0x01],u16_tag_id,u16_seq,u16_area_id,i16cm_x,i16cm_y,i16cm_z,u8_switch,u8_battery_voltage,u8_power_voltage
```

首先定义了两个常数 0x55 和 0xAA 作为数据包头，第 3 个节是常数数据包长度 20，第 4 字节是数据包的 CRC8 的值(CRC 的计算从本字节后一字节开始到数据包结束)，第 5 字节是常数数据包类型 0x01，第 6、7 字节是标签标识的后 2 字节，第 8、9 字节是定位消息序号号的低 2 字节，第 10、11 字节是区域标识的后 2 字节，第 12、13 字节是单位为厘米的 X 坐标值，第 14、15 字节是单位为厘米的 Y 坐标值，第 16、17 字节是单位为厘米的 Z 坐标值，第 18 字节是按钮状态，第 19 字节是电池电压，第 20 字节是电源电压(充电电压)。

6.2. 数的大小端

RTLE 输出的所有的数都使用 Little-Endian 模式，即低位在前，高位在后。例如 2 字节长度的整数，前面字节表示低 8 位，后面字节表示高 8 位。

需要特别说明的是 EUI64 标识，使用 64 位二进制(8 字节)表示，其变化一般是从后到前，例如顺序为两个实体分配标识的时候，通常是最后一个字节增加 1，如果需要进位，则前一字节增加 1，并且把后一字节置 0，我们认为 EUI64 标识是大端在前。所以，对 EUI64 标识，总是使用 Big-Endian 模式。

如果对整数进行截取低位，是抛弃后面的高字节；如果对 EUI64 标识截取低位，是抛弃前面的高字节。

附录一 UWB 配置参数

定位系统中有三种类型的设备(标签、基站、时钟源)涉及到 UWB 的配置。
总体来说，一个系统中全部有 UWB 设备都需要使用相同的配置才能互通。

频道选择

IEEE 802.15.4 标准中为 UWB PHY 定义了 16 个频道，我们支持其中的 1、2、3、4、5、7 共 6 个频道。

这 6 个频道中，1、2、3、5 这 4 个频道占用宽带都是 500MHz，4、7 这两个频道占用带宽是 1GHz。2 频道和 4 频道的中心频率相同，5 频道和 7 频道的中心频率相同。

在大多数情况下，我们使用 2 频道。如果在 2 频道有干扰，或者需要部署第 2 个或更多的互不相干的定位系统，可以选择 1 频道或 3 频道。

通讯速率选择

系统中的 UWB 设备间的通讯速率有 3 种：110 kbps，850 kbps 和 6.8 Mbps。
通讯速率对系统的影响非常大。较低的速率支持较远通讯距离，较高的速率下通讯距离变近；较低的速率下发送相同大小的数据包需要较多的时间，较高的速率下发送相同大小的数据需要较少的时间。
如果选择较低的通讯速率，例如 110kbps，定位区域的覆盖范围会比较大，但是发送定位数据包需要的时间会比较长，会导致标签的耗电增加，定位区域内能支持的标签数量较少。
如果选择较高的通讯速率，例如 6.8Mbps，定位区域的覆盖范围会比较小，但是发送定位数据包需要的时间会比较短，标签的耗电会比较小，定位区域内能支持的标签数量较多。

通常，我们建议仅在室内部署系统时，因为每个定位区域都比较小，可以使用 6.8Mbps 的通讯速率，这样可以减少标签的电量消耗，让标签有更长的待机时间，同时也可以支持更多的标签。
如果定位系统有室外部分时，可能单个定位区域需要有较大的覆盖，可以使用 850Kbps 的通讯速率。在 850Kbps 速率下，单个定位区域的覆盖范围可以达到 100 米*100 米，在覆盖范围和标签的电力消耗方面可以达到平衡。
如果对单个定位区域的覆盖要求很大，可以使用 110Kbps 的通讯速率。但是这时标签的待

机时间会大大降低。

其它 UWB 参数

PRF(脉冲重复频率)，系统支持 16M 和 64M 两种 PRF。16 和 64 是“标称的”，因为实际频率与所使用的 499.2 MHz 基本时间单位有关，并且在帧的前导码部分和有效负载部分之间略有不同。 **较高的 PRF 可以在第一路径时间戳上提供更高的精度，并可能会稍微改善工作范围，但这是以增加功耗为代价的**。