数据驱动设计说明文档

潘荣涛

简要介绍数据驱动的设计思想、设计方案；介绍API的使用以及可视化调试工具的使用方法。

德赛西威

[ 数据驱动设计 3](#_Toc840)

[ 背景 3](#_Toc23639)

[ 理念 3](#_Toc31315)

[ 数据驱动在UIT框架的地位 3](#_Toc918)

[ 特性 3](#_Toc25110)

[ 架构图 4](#_Toc11248)

[ 开发指南 5](#_Toc25027)

[ 数据驱动数据配置 5](#_Toc31701)

[ 支持的数据类型 5](#_Toc23022)

[ JSON文件 5](#_Toc24037)

[ 自定义数据事件 6](#_Toc19882)

[ View（GUI）端 6](#_Toc18452)

[ 对接数据驱动层 6](#_Toc1433)

[ 定制优化view端使用的数据 6](#_Toc26865)

[ Controller（APP端） 6](#_Toc17570)

[ 设置数据 7](#_Toc21522)

[ 事件注册和处理 7](#_Toc25381)

[ Log 8](#_Toc20214)

[ 可视化调试工具Simulator 8](#_Toc3290)

[ 启动simulator 8](#_Toc10711)

[ 配置数据事件源 9](#_Toc32463)

[ 本地调试 10](#_Toc11340)

[ 数据/事件树 10](#_Toc19991)

[ 模拟面板 11](#_Toc7075)

[ Log面板 18](#_Toc17703)

[ 远程调试（PC端控制车机端） 18](#_Toc20824)

[ 车机配置网络 18](#_Toc5369)

[ 注意事项 20](#_Toc18980)

# 数据驱动设计

## 背景

在未来趋势中，数据驱动的模式会变成主流，且这种模式会体现出它诸多的优势。这些优势将体现在往后更复杂的系统中，例如：跨图层、跨GUI、跨屏、甚至跨机的交互方式。

## 理念

设想一个场景，我们需要在系统中使用linux+android，使用多个屏幕，并采用KANZI + Qt的双UI方案交互。

这里我们需要要解决的问题很多，包括：跨系统交互，跨GUI交互，跨屏交互等，如果使用传统的app耦合UI层的做法，事情将变得非常复杂，且这种跨多维度的交互实现也难以实施。

数据驱动的模型，将重新确立数据的主导位置，这种理念的核心是“内容决定表现”，且“内容不干预表现”。具体来说，数据驱动的方式，将把数据层，表示层更加清晰地区分开来，数据决定了表现层的大致形式，但却不干预它一定做成什么样：我的蓝牙开关状态变化了，UI可以表现为“文本提示”，也可以表现为“图标变化”。

当数据的主导确立，那么数据（和事件）将作为表现层和功能层交互的标准入口：可以说，通过数据驱动的方式，把UI表现层与底层功能层的交互简化到极致。

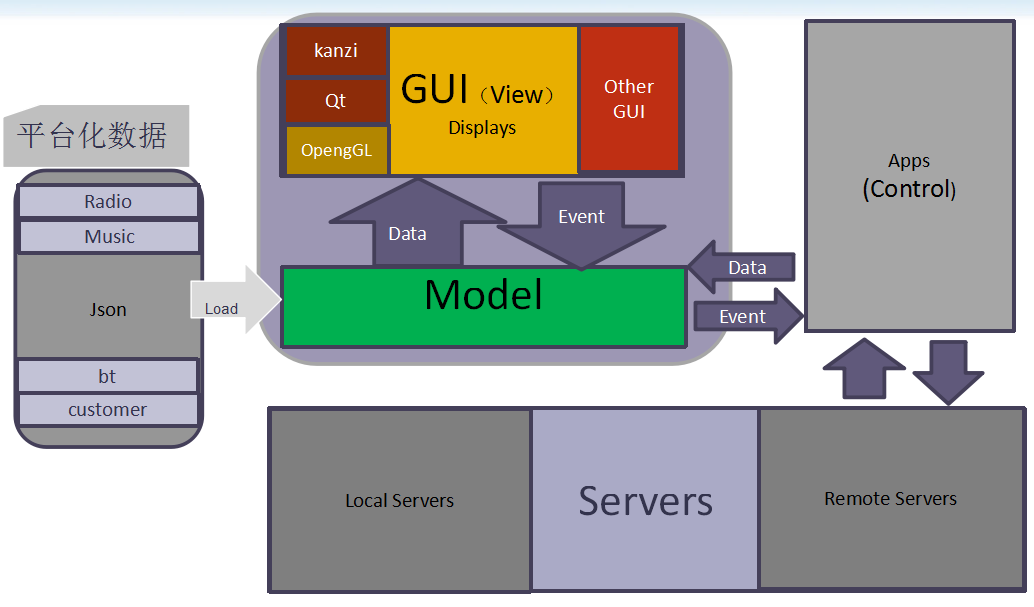
## 数据驱动在UIT框架的地位

数据驱动在UIT框架中处于非常核心的位置，因为所有与UIT交互的接口都由数据驱动层定义，这是UIT框架最核心的理念。

## 特性

1. 数据驱动服务作为独立的进程，负责加载数据和事件
2. 统一、规范、隔离GUI与APP的交互
3. 支持多种GUI接入：KANZI GUI，Qt GUI等
4. 支持多个app（控制端）接入进行数据控制、事件逻辑处理
5. 数据事件可定制，满足项目需求
6. 支持多屏交互，中控屏、仪表屏、空调屏、HUD等
7. 支持多系统交互：windows，linux，ios等
8. 支持可视化工具模拟数据事件调试

## 架构图



从图中可以了解到，Model作为与GUI和app交互的标准层，简单来说，它标准化了这两者交互的方式，即数据的更新以及事件的发送与接收。当model在系统中启动后，会加载定义好数据和事件的json文件。之后则等待GUI或者app的连接。

App的职责是控制数据的变化以及监听到事件后的逻辑处理，而GUI层则是负责具体数据绑定的刷新动作和用户交互的事件发送：当app收到底层功能服务发送的一个状态变化，此时设置一个数据给model端，model会通知给关心这个数据的显示进程，显示进程则刷新与此数据绑定的ui 元素。反过来，如果用户在ui上有动作交互，GUI层就把此事件发给model端，model再转发给关心此事件的app。

至此，一个完整的闭环交互就完成了。

# 开发指南

## 数据驱动数据配置

我们知道，数据驱动服务启动后，就会去加载etc/data-event目录中的json文件，这些json文件装着所有的数据和事件。

### 支持的数据类型

在数据驱动的框架中，除了json本身支持的基础类型外，框架定义了自己的类型，总的来说，支持了bool、int、float、string、Enum、Image、List类型。

### JSON文件

1. 【EnumDefinition.json】：定义枚举类型的地方。这些枚举定义会用在数据定义中，或者事件的参数中，代表这个数据或者参数是个枚举类型。

"GearEnum" : ["Unknown", "P", "N", "R", "D1", "D2", "D3", "D4", "D5", "D6"] 表示该枚举定义有10个枚举值。

一般而言，枚举定义都保留一个”Unknonw”的值，用作初始状态或未知状态。

1. ItemTemplate.json：列表项模板定义的地方，比如Radio.StationItemTemplate代表一个电台数据模板。一般而言，数据模板用在列表类型的数据中。
2. Data.json：数据定义。

布尔类型：Bt.Music.IsRootDir : false 初始值为false

整数类型：Bt.Music.CurrentMediaInfo.PlayingIndex : 0 初始值为0

浮点类型：Setting.SystemInfo.MemoryUsed : 0.3 初始值为0.3

字符串类型：Bt.Call.ActivedCallingInfo.FullName : “” 初始值为空字符串

枚举类型：BodyInfo.Gear: ["BodyInfo.GearEnum", 0] 枚举类型为"BodyInfo.GearEnum"，初始值为0（Unknown）

图片类型：Ipod.CurrentMediaInfo.Picture : ["Image"] 初始值为空图片

列表类型：Ipod.MediaList : ["Ipod.MediaItemTemplate"]

1. Event.json：事件定义

Navi.FullScreenEvent : {"FullScreen" : ["Global.MarkEnum", 0]},

事件名称为：Navi.FullScreenEvent

参数列表：”FullScreen” 枚举开关。

事件意义：切换Navi的全屏状态到0

### 自定义数据事件

在EnumDefinition.json中，Custom下添加新枚举类型即可

在ItemTemplate.json中，Custom下添加新列表项模板即可

在Data.json中，Custom下添加新数据即可

在Event.json中，Custom下添加新事件即可

## View（GUI）端

### 对接数据驱动层

如果你有一个GUI方案，需要对接入UIT的数据驱动层，那么你要做的，仅仅是使用相关的API，来对接即可。UIT数据驱动层会有一系列接口和回调反馈给View端，使用这些接口和回调处理GUI内部的状态，比如构建自己的数据结构和事件结构、更新数据到元素等。

关键的接口为connect、postEvent、已经数据更新通知事件onXXXChangedEvent。

如何使用这些接口，请此参照头文件datafw/View.h和example/test-view工程。

### 定制优化view端使用的数据

由于图层管理的缘故，一个系统中往往不止一个显示进程。又因每个显示进程的复杂度不一样，导致每个显示进程使用到的数据都有所不同。假如每个显示进程对每个数据都关注，显然是会浪费性能的（数据驱动层会给每个关心数据的显示进程通知数据变化）。

基于上诉原因，我们应该对不同的显示进程配置不同的数据。例如，StatusBar显示进程用到的数据不足20个，那么StausBar显示进程需要把使用到的数据和事件挑选出来，而不是加载所有数据驱动层提供的数据。这样做，就能达到提高性能的目的。因为，如果某个显示进程不关心此数据，则不需要通知它做相应的更新。

简单来说，显示进程用到的数据一定是数据驱动层提供的数据的子集。

## Controller（APP端）

详细可参照头文件uit/datafw/Controller.h和实例工程example/test-controller.h

### 设置数据

包含头文件#include <uit/datafw/Controller.h>即可使用API，下列是对不同数据类型进行设置的示例：

Controller::setInteger("Radio.CurrentMediaInfo.PlayingIndex", 1);  //设置整数

Controller::setReal("Radio.CurrentMediaInfo.Frequency", 1);     //设置浮点数

Controller::setString("Radio.CurrentMediaInfo.StationName", "desay"); //设置字符串

Controller::setEnum("BodyInfo.Gear", 3);           //设置枚举整型值

Controller::setEnum("BodyInfo.Gear", "N"); //设置枚举字符串值

Controller::setImage("Ipod.CurrentMediaInfo.Picture", data, len);//设置图片

Controller::resizeList("Keyboard.CandidateList", 10);   //列表重置大小为10

Controller::insertListItems("Keyboard.CandidateList", 2, 3); //列表从第二项开始插入3项

Controller::appendListItems("Keyboard.CandidateList", 6);    //列表在尾部追加6项

Controller::removeListItems("Keyboard.CandidateList", 0, 5); //从第0项开始移除5项

Controller::updateViewport("Keyboard.CandidateList", 3); //更新视口为3的第一可见项

Controller::checkListItem("Keyboard.CandidateList", 6);     //选中标记唯一项6

Controller::focusListItem("Keyboard.CandidateList", 6);     //聚焦列表第6项

Controller::vkey(VkeyId::vkeyLeft, VKeyState::vkeyStatePress);  //压下左按键

//尽可能一次更新多项数据，可以减少网络通讯开销

void updateList()

{

std::*set*<unsigned int> indexs = { 1, 2, 3, 5, 9};

   Poco::JSON::Array::Ptr items = new Poco::JSON::Array();

   for (int i = 0; i != indexs.*size*(); ++i)

   {

   Poco::JSON::Object::Ptr item = new Poco::JSON::Object();

   item->*set*("StationName", "xxxx");

     item->*set*("Frequency", 3.2f);

     item->*set*("BandState", 2);

     items->add(item);

   }

   Controller::updateListItems("Radio.StationList", indexs, items);

}

### 事件注册和处理

Controller::registerEvent("Radio.StationListScrollEvent"); //注册事件

Controller::userEvent() += std::bind(&ControllerApp::onUserEvent, this, std::placeholders::\_1); //添加处理函数

void ControllerApp::onUserEvent(const Controller::UserArgs & args)

{

printf("onUserEvent:\n%s:%s\n", args.eventName.data(), Poco::Dynamic::Var(args.args).toString().data());

}

## Log

数据驱动中的log以几种形式给出，分别是info、warn、error。

Info：普通的信息打印

Warn：警告级别信息，一般是一些无实际影响但不合理的使用方式，比如GUI发送了一个事件，但没有一个app关心此事件。

Error：错误级别的信息，一般来说是与预期结果不符的结论。比如用app尝试更新一个不存在的数据。此类型的log一定要留意，对排除问题是否有帮助。

(2482|0xA0C65000) [data-fw|controller] INFO: controller connect success, cost [106] ms.

(2386|0xB53521E0) [data-fw|model] WARN: no controller register event [Setting.ChangeBrightnessEvent] when view try to post it.

(2447|0x902771E0) [data-fw|model] ERROR: [VideoData.CurrentMediaInfo.Seeking] is not a valid data when call [updateReal]

(2388|0xB3E31000) [data-fw|view] INFO: view connect success, cost [74] ms

在实际调试中，请密切关注log情况，以了解使用上是否有问题。

# 可视化调试工具Simulator

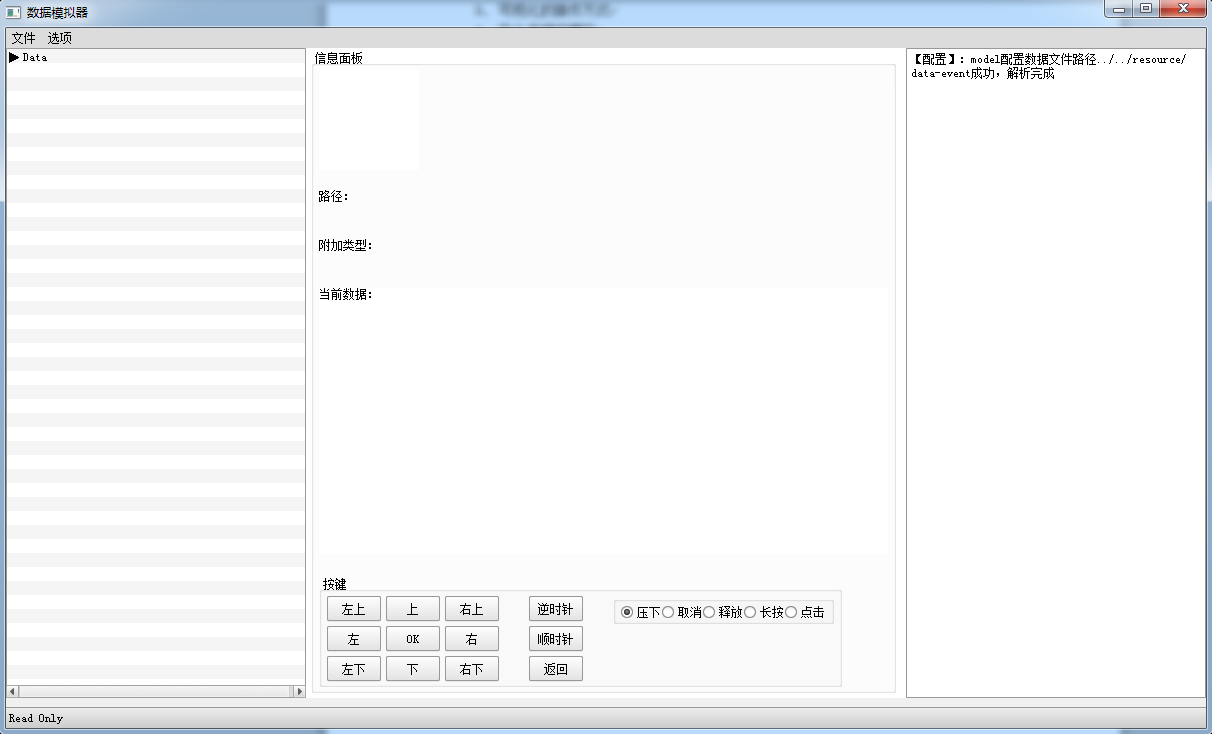
制作工具，主要为了解决在PC环境下依赖代码级调试的问题，将工具可视化，可大大提高操作性和使用便利性，基本上，可不需要写一句代码即可进行数据的模拟操作以配合GUI完成调试工作。工具具备以下特性：

1. Win32环境下运行
2. 可模拟已加载的所有数据和事件
3. 可视化的操作方式
4. 覆盖数据驱动中的所有API调用模拟
5. 可对接包括KANZI或者Qt在内的任何GUI程序，只要该GUI完成与UIT数据驱动层的对接即可
6. 无需写代码

工具名称为Simulator，基于QML开发，路径位于在bin/debug/simulator

## 启动simulator

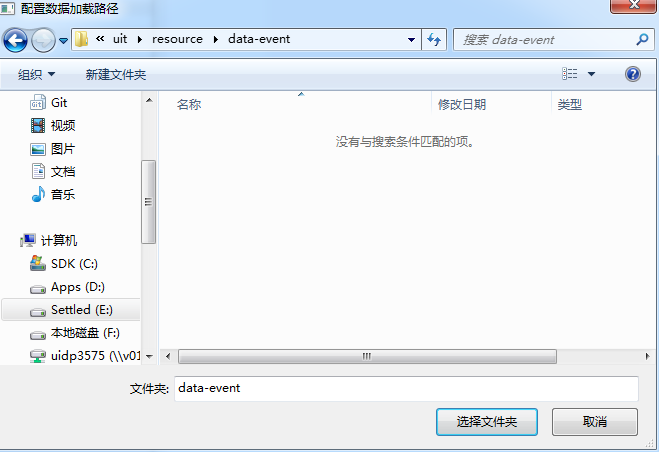
在控制台中执行环境变量[etc/uit.env.bat]后，键入[Simulator.exe]即可启动，启动后界面大致如下：



## 配置数据事件源

工具启动后，会默认加载etc/data-event下的json。如果需要修改数据源目录，使用菜单栏中的配置即可

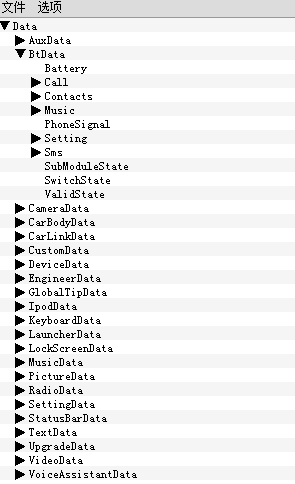




## 本地调试

### 数据/事件树

左边栏展示的是所有加载的数据/事件，意味着所有可模拟的数据都在此树中。节点的操作可以使用鼠标点击，也可以使用键盘的“上/下左/右”来进行操作。



### 模拟面板

中间是当前数据的信息面板，当当前节点变化时，面板会展示该数据的路径、类型、值以及模拟驱动操作等信息。每种类型展示的信息并不相同。

#### 数据模拟

##### 布尔类型



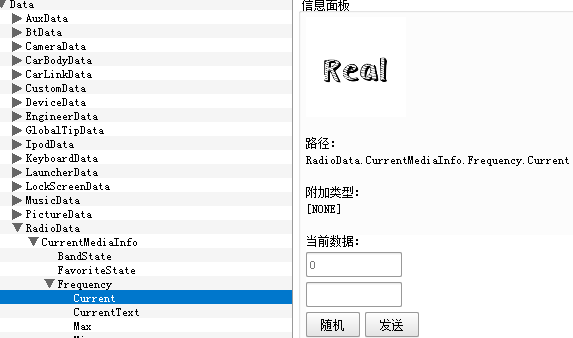
点击“当前数据”下的checkbox即可模拟bool数据的true/false值。

##### 整型



随机按钮可以快速输入一个随机整数，点击发送即可完成模拟

##### 浮点型



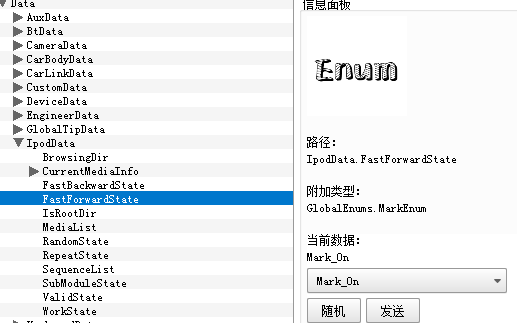
随机按钮可以快速输入一个随机浮点数，点击发送即可完成模拟

##### 字符串型



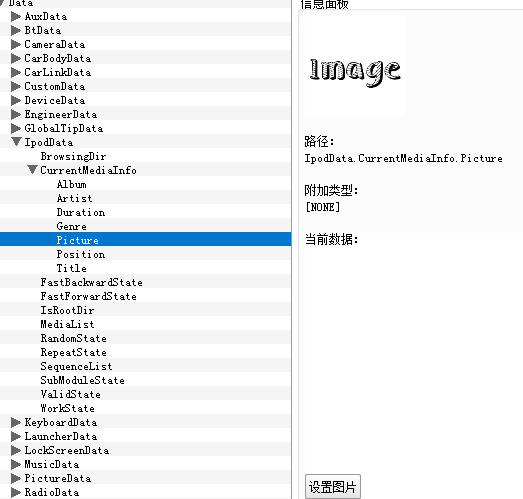
随机按钮可以快速输入一个随机字符串，点击发送即可完成模拟。

##### 枚举型



枚举类型仅可以选择该枚举定义下的枚举值来作为模拟的值，同样的，随机按钮会产生一个随机枚举值，点击发送可以完成模拟。

##### 图片



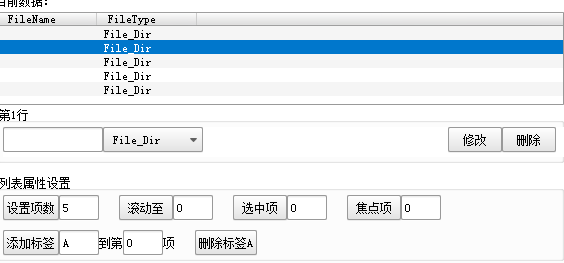
点击设置图片从系统中加载一张图片即可完成模拟，需要注意的是，图片有大小限制，解析后内存限制为5mb

##### 列表



列表操作有较多的模拟动作，包括设置项数、设置第一可见行、设置唯一选中项、设置焦点项、添加/删除标签、修改项数据等。

修改行数据操作面板会在选中某项后才显示出来，如果列表是空的，则会隐藏。



##### 按键模拟



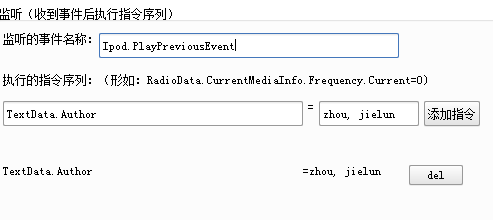
提供按键的多个状态的模拟操作。

#### 事件模拟



调整好事件参数后即可发送该事件，凡是注册了该事件的controller端都会接收得到。

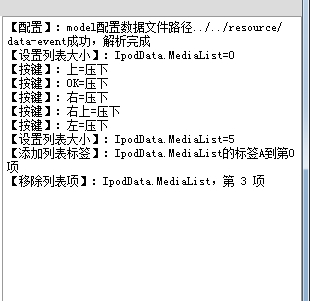
另外，事件处有个监听行为控制。即当收到某个事件时可以发送系列指令，这些指令就是操作数据的指令。



上面表示：当收到UI发送过来的，ipod上一曲消息时，把TextData.Author=”zhou, jielun”发送给UI，这在模拟一些简单的逻辑时会非常方便。

### Log面板

Log面板展示重要的信息通知，包括模拟的操作信息等。



## 远程调试（PC端控制车机端）

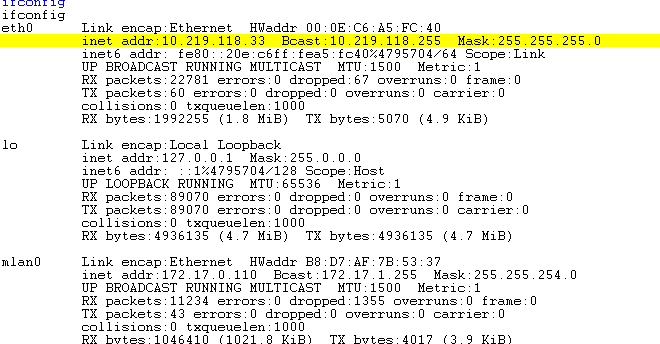
Data-similator工具支持远程的调试方式。可以通过在PC上的操作来驱动在车机上的模拟。简单来说，只要是data-similator工具支持的模拟操作，都可以驱动到车机上去。只是，需要一定的配置支持。

### 车机配置网络

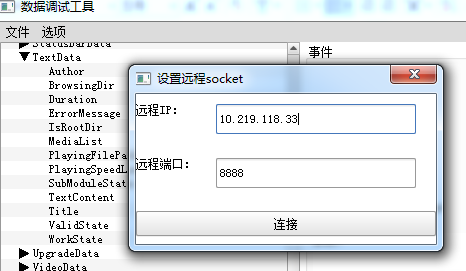
1. 必须把车机网络配置为与PC端一个网段（局域网内）。由于目前车机无法使用wifi的方式来确保与PC同一网段，则需要以网线的方式来连接，建议车机使用usb转网口的方式来连接到公司网络。



1. 连接好网先后，开机启动，执行命令udhcpc来自动获取ip地址：一般而言，分配的ip网段为10.219.118.xx。



1. 启动data-similator后选择选项->设置，设置ip为车机的ip，并且点击连接按钮即可完成连接。



1. 完成以上步骤后就可以像本地操作一样进行数据和事件的模拟了。

### 注意事项

进行远程调试时，必须先保证车机端的model先启动，后启动data-similator工具才可以正常进行连接和调试。