# 前言

先说一下为什么写这个教程。由于新项目打算用creator开发，服务器用的是c++，底层通信用的是tcp+protobuf，所以就要用到creator的定制引擎。由于之前没用过creator，刚开始做项目就要用定制引擎，对js也不太了解，所以这个过程中经历了非常多的坑。写这个教程就是为了给那些和我有同样经历的同学，帮你们少走一些弯路，同时提供一个还算凑合的解决方案，一个完整的demo，包含定制引擎相关操作，js与c++互调，数据如何往来，以及protobuf相关操作，bytes和repeat的案例。但不包含tcp通信相关代码，demo中只是在模拟这个操作，这个涉及到加解密以及和心跳包等，本教程不提供。

再说一下demo的适用场景。不适用于web和微信小程序，仅适用于安卓ios和windows。如果你开发的应用只需要发布到手机平台，那刚好适用。

好，来开始吧。

本人的开发环境操作系统:windows。mac下同样适用。

# 准备工作

1.creator1.8.2。

2.vs2015，xcode。

3.eclipse or android studio以及对应的sdk ndk。

4.python 2.7.14版本，32位。不要用2.7.3等低版本的python，也不要用64位python。这是第一个坑，如果低版本python，定制c++引擎的时候，执行python download-deps.py会出错。原因是低版本的openssl库版本过低，py下载远程zip包时会报错。如果用64位也会有的地方执行不成功。

下载地址：https://www.python.org/downloads/release/python-2714/

5.git。下载地址 https://git-scm.com/

6.node.js。下载地址：https://nodejs.org/en/

# 一、定制引擎

官网定制引擎流程：

http://docs.cocos.com/creator/manual/zh/advanced-topics/engine-customization.html



第一步，把github服务器上的定制引擎代码clone到本地。

首先在本地创建一个定制引擎的根目录。我的在F:\svn\client。然后命令行进入这个目录。

使用git命令把需求的分支clone本地仓库。

创建js引擎仓库

git clone -b v1.8-release https://github.com/cocos-creator/engine.git

创建c++引擎仓库

git clone -b v1.8-release https://github.com/cocos-creator/cocos2d-x-lite.git

我用的是1.8.2的creator，所以clone的是1.8-release分支。

等待clone完成。中间还出错了两次。从github上下载代码速度很慢，这里有个方法可以提高速度。https://blog.csdn.net/u013360850/article/details/77145661

然后命令行进入js引擎的根目录。

执行官网教程给出的命令。

**安装编译依赖**

# 安装 gulp 构建工具

npm install -g gulp

# 在命令行中进入引擎路径

npm install

### 进行修改然后编译

接下来您可以定制引擎修改了，修改之后请在命令行中执行：

gulp build

来编译将引擎源码编译到 bin 目录下。

然后进入c++引擎根目录。

# 安装编译依赖

npm install

# 下载依赖包，需要提前配置好 python

python download-deps.py

# 同步子 repo，需要提前配置好 git

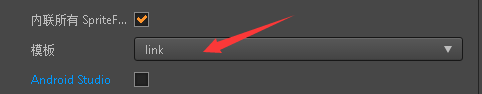
git submodule update --init

当时到第二个命令的时候一直执行失败，最后是python版本的问题，弄了半天才搞好。

如果你按照我之前说的，是通过git clone命令下载的代码，那第三个命令是可以正确执行的。如果不是的话，请重新从github上clone吧。

以上完成后，才算是把完整的c++引擎下载的本地。

由于我的项目构建发布用的是link模式，所以不执行 gulp gen-libs命令。



依次执行以下命令生成模拟器。每次修改c++代码后都要c++引擎目录执行一下命令。

# 通过 cocos console 生成模拟器

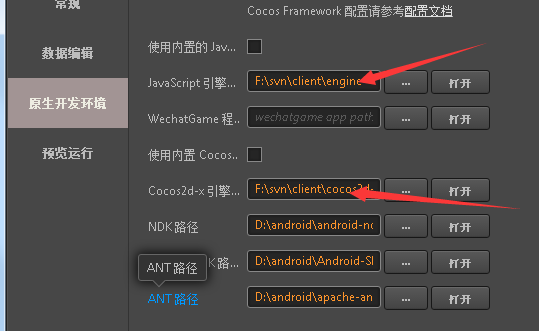
gulp sign-simulator

gulp gen-simulator

gulp update-simulator-config

gulp sign-simulator 是 1.7.0 中的新增命令，只有 Mac 需要运行。

creator的偏好设置中，修改原生开发环境配置。



箭头指示的位置修改为从github上clone的项目路径。

到目前为止，定制引擎配置完成。新建creator项目点击预览正常运行。

# 二、JSB绑定

官网jsb绑定教程：

http://docs.cocos.com/creator/manual/zh/advanced-topics/jsb/JSB2.0-learning.html

使用的是JSB2.0的自动绑定。

# 绑定流程：

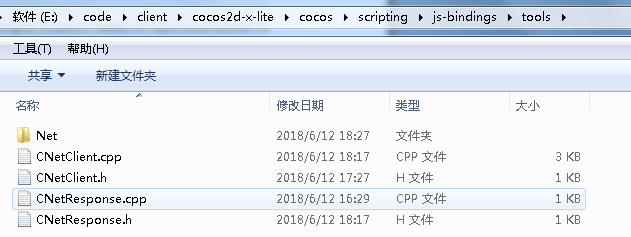
## 首先写好要绑定的c++类。

项目的github：

demo中写了两个需要绑定的类。CNetClient和CNetResponse。

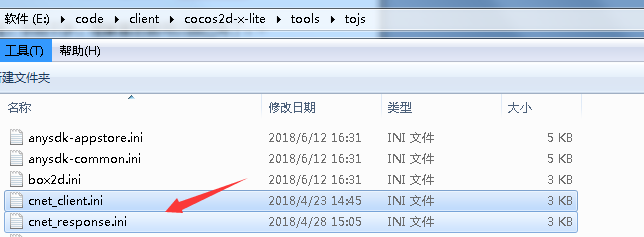
我选择把绑定的文件和cocos引擎默认需要绑定的类文件放在了一起。

在cocos2d-x-lite\cocos\scripting\js-bindings目录下新建了一个tools文件夹，把编写到c++文件放到了这里。

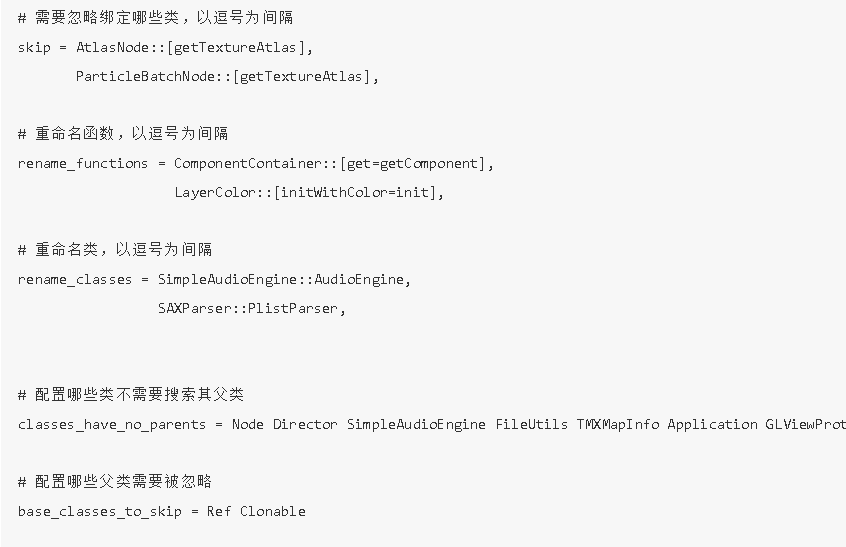


## 然后编写自动绑定的脚本。

绑定脚本在cocos2d-x-lite\tools\tojs目录下。



绑定脚本的字段意义参考官网教程。字段不多，理解意思就可以自己写了。



在genbindings.py文件中添加新写的脚本。



这些完成之后就可以通过执行命令自动生成绑定了。

进入到genbindings.py所在的目录cocos2d-x-lite\tools\tojs。

执行 python genbindings.py。

这个时候可能会报错，因为python没有安装PyYAML和Cheetah。下载并安装。（安装python的时候要安装在c根目录下）

PyYAML官网:：http://www.pyyaml.org/wiki/PyYAML

Cheetah官网：http://cheetahtemplate.org/

安装之后再执行命令。如果类和脚本没有问题会提示

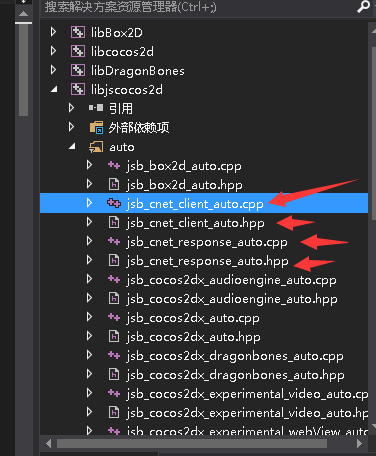


## 添加生成和jsb文件c++项目中。

进入cocos2d-x-lite\tools\simulator\frameworks\runtime-src目录下，这个是creator模拟器的工程项目。安卓，ios，win都在这里。

打开proj.win32下的simulator.sln。

在libjscocos2d项目下的auto目录添加我们自动绑定成功的c++文件。如下图:



然后打开simulator项目。在jsb\_module\_register.cpp里添加注册类到js环境。

首先添加头文件引用

#include "cocos/scripting/js-bindings/auto/jsb\_cnet\_client\_auto.hpp"

#include "cocos/scripting/js-bindings/auto/jsb\_cnet\_response\_auto.hpp"



然后在jsb\_register\_all\_modules函数下添加注册。

se->addRegisterCallback(register\_all\_cnet\_client);

se->addRegisterCallback(register\_all\_cnet\_response);



在安卓和ios项目下，同样添加c++文件的引用。篇幅有限，不在赘述。

刚刚我们只是把模拟器的项目添加了注册js的代码。新建项目的时候，新项目的c++工程下需要再修改一下jsb\_module\_register.cpp。

## 更新模拟器

每次修改c++代码都要调用执行下面命令重新生成模拟器。

# 通过 cocos console 生成模拟器

gulp sign-simulator

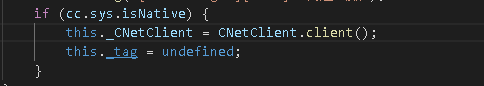
gulp gen-simulator

gulp update-simulator-config

进入cocos2dx-lite目录执行以上命令。注意执行的时候关闭vs2015和creator，否则可能会出错。建议执行命令之前先运行模拟器项目编译一下，编译完成在执行，这样可以忽略掉一些低级错误。

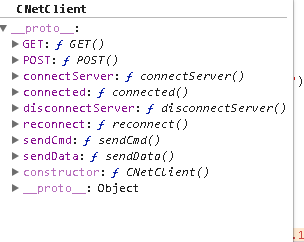
自此，定制引擎部分就完成了，我们可以在js代码中直接使用CNetClient和CNetResponse类。

使用的时候请加上if(cc.sys.isNative)。否则可能会编译不过。



我们还可以在谷歌浏览器输入以下网址调试我们的代码。

chrome-devtools://devtools/bundled/inspector.html?v8only=true&ws=127.0.0.1:5086/00010002-0003-4004-8005-000600070008



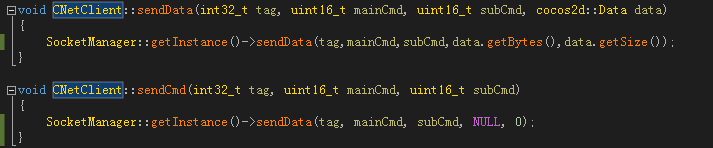
# 二、js和c++传递数据

为了简化流程，我在项目中只用c++提供一些基础的类供js调用，c++不主动调用js，需要调用的地方通过一些回调函数来传递。js可以通过调用c++的接口来传递一个函数指针。c++保存这个函数指针，在需要的时候调用它。（尽量在主ui线程中去调用）。

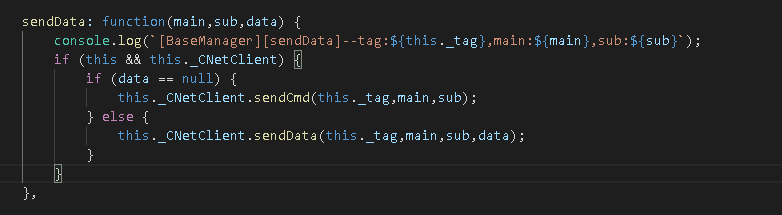
由于项目使用的是tcp。传递数据肯定会涉及到二进制流，我们可以用cocos2d::Data来传递数据，jsb自动绑定是可以直接绑定Data的。在js中对应的数据结构是Uint8Array，发送的时候传Uint8Array即可。

## 传递Data

c++接口：

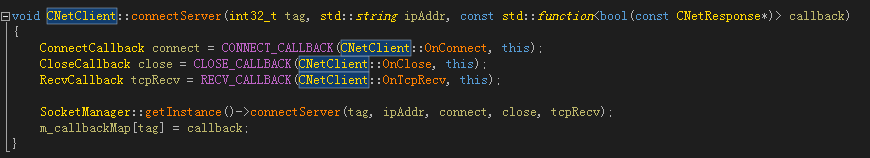


js调用：

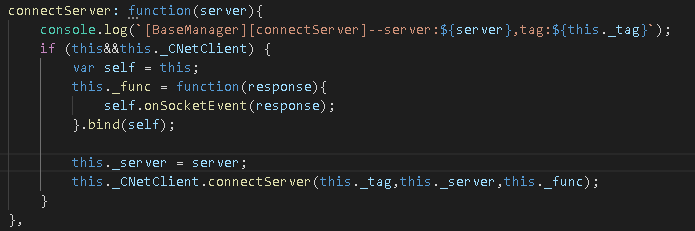


## 传递函数指针

c++接口

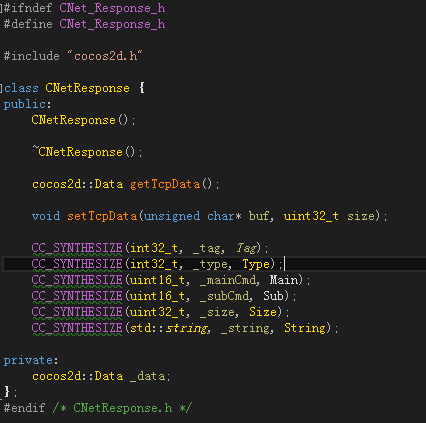


js调用设置回调函数：

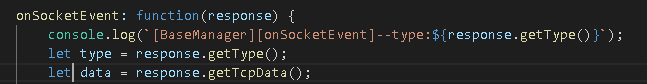


为了方便c++回调js方便，封装了一个CNetResponse类用来传递数据。

类的内容很简单，只是一些set，get函数。



c++去填充这个response，js去读其中的数据。



具体内容参考c++类以及js相关调用的实现。

js有number和string，string对应了std：：string。

还可以参考creator官网给出的一些方案来传递更多的数据类型。

http://docs.cocos.com/creator/manual/zh/advanced-topics/jsb/JSB2.0-learning.html



# 三，protobuf。

项目中的引用的protobuf最开始是使用奎神的pbkiller。pbkiller是基于protobufjs5.x的。写的过程中发现低版本protobufjs传递uint8array太麻烦了。然后果断放弃了pbkiller，使用了最新的protobufjs6.8.6.

protobufjs的github：https://github.com/dcodeIO/protobuf.js

可以通过npm install -g protobufjs命令去获取。

参考：http://forum.cocos.com/t/cocoscreator-protobuf/61045。不再赘述。