目录

[第1章 CrossApp简介及开发环境搭建 2](#_Toc413689277)

[1.1 CrossApp简介 2](#_Toc413689278)

[1.1.1 CrossApp功能与特色 2](#_Toc413689279)

[1.1.2 CrossApp的优势 3](#_Toc413689280)

[1.1.3 官网介绍及技术支持 3](#_Toc413689281)

[1.2 CrossApp开发环境搭建 4](#_Toc413689282)

[1.2.1 Windows开发环境搭建 4](#_Toc413689283)

[1.2.2 Mac OS X开发环境搭建 4](#_Toc413689284)

[第2章 CrossApp基础概念 4](#_Toc413689285)

[2.1 架构和目录结构 4](#_Toc413689286)

[2.1.1 架构 4](#_Toc413689287)

[2.1.2 目录结构 4](#_Toc413689288)

[2.2 核心类 5](#_Toc413689289)

[2.3 内存管理 5](#_Toc413689290)

[2.4 坐标系 6](#_Toc413689291)

[2.5 适配方案 6](#_Toc413689292)

[第3章 CrossApp核心控件 6](#_Toc413689293)

[第4章 CrossApp数据解释与存储 6](#_Toc413689294)

[第5章 CrossApp设备功能调用 6](#_Toc413689295)

[第6章 CrossApp进阶 6](#_Toc413689296)

[第7章 CrossApp网络通信 6](#_Toc413689297)

[第8章 折800—实战讲解 6](#_Toc413689298)

[第9章 动漫之家—实战讲解 6](#_Toc413689299)

# CrossApp简介及开发环境搭建

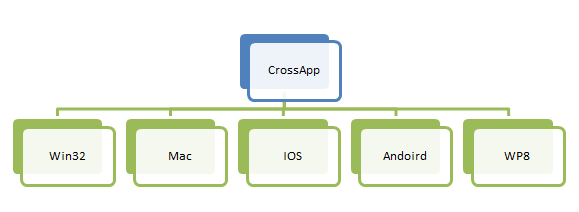
## CrossApp简介

 CrossApp是一款完全开源、免费、跨平台的移动应用开发引擎，基于最宽松的MIT开源协议，开发者根据自身情况使用CrossApp开发任何商业项目。CrossApp以C++作为开发语言，图形渲染基于OpenGL ES 2.0，采用MVC框架模式。使用CrossApp开发的应用程序支持导出到各大主流移动平台，真正实现"一次编码，多处运行"的跨平台开发技术。

CrossApp主要由9秒社团自研，官方制定了CrossApp的基本架构，确立了CrossApp的基本雏形，而其后的版本也将由来自各方的开发精英自愿组成的9秒社团常务贡献委员会共同进行更新和维护。

### CrossApp功能与特色

**跨平台**



CrossApp暂时主要支持目前最主流的移动平台IOS和Android，后续更新版本将完善更多平台的支持。

**整合第三方库**

CrossApp整合一些第三方库，例如常见的数据解析库jsoncpp、tinyxml、http等。

**基于OpenGL ES 2.0**

CrossApp的图形渲染使用的是Open GL ES 2.0，渲染效率高，可以使移动设备的GPU发挥到最佳效果。

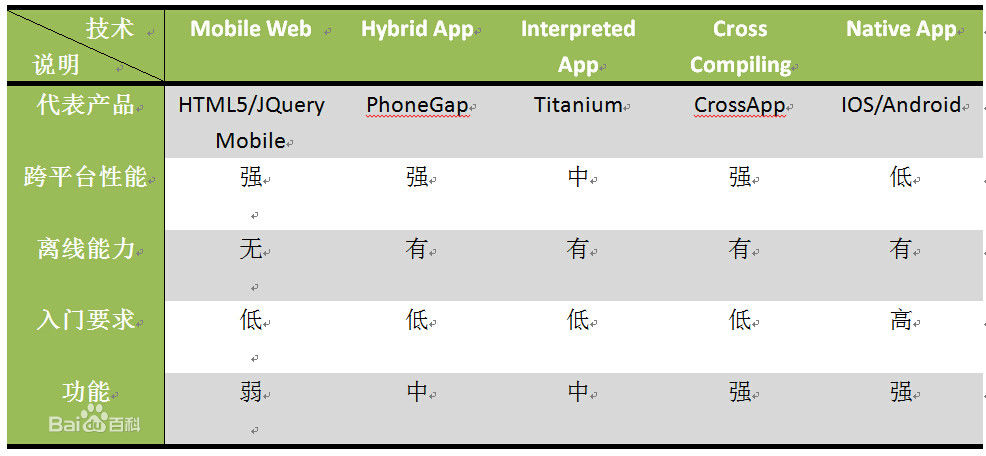
**丰富的UI控件**

CrossApp的设计宗旨在于为移动应用开发者提供快速、高效的开发解决方案。在此基础上，CrossApp封装了大量的UI控件，各类控件的功能十分丰富，开发者可以直接使用这些控件进行应用的开发。这些控件基本满足应用开发需求中的大部分功能，也免去了自己封装UI控件花费的大量时间，进一步提升开发的效率、节省开发的成本。同时CrossApp也会整合部分由第三方开发者提供的优秀控件资源，更大的丰富CrossApp的UI控件。

**CrossApp耗电优化**

由于之前CrossApp抽取了cocos2d-x的渲染驱动模式，程序生命周期中，在不断重绘，渲染驱动模式的缺点就是CPU占用高因此耗电。显而易见，这种不断的重绘方式对于游戏可能影响不大，但对于APP来说太浪费了。因此，我们针对应用程序的特性，将渲染机制改为事件驱动模式。这种模式的渲染要有外界触发才会重绘，在没有外界触发的时候画面静止，渲染停止，以达到节能的效果。

### CrossApp的优势



**通过上表分析我们可以得出：**

**1.Mobile Web**

以HTML5和JQuery为代表的web开发技术是以纯浏览器为基础的，所以没有离线能力可言，更无法充分发挥各平台的功能和特性。

**2.Hybrid App**

Hybrid App即混合模式应用程序，是指介于web和native直接的app，具备一定Native原生App优秀的用户体验和Web App跨平台的优势。但由于Hybrid仍旧以前端技术为基础，无法对内存和系统资源进行有效的管理。

**3.Interpreted App**

以前端技术为基础，同样不能有效对内存和系统资源进行管理。

**4.Native App**

原生开发以iOS和Android为代表，虽然原生开发能够实现最佳的用户体验和高优化，但开发的成本较高。而且因为原生开发几乎不支持跨平台特性，相对于跨平台技术来说，带来的实际效益更低。

综上所述，还是Native App和Cross compiling的综合效率更高，开发者需要根据实际情况，考虑各方面因素来选择合适的开发解决方案。

### 官网介绍及技术支持

## CrossApp开发环境搭建

### Windows开发环境搭建

### Mac OS X开发环境搭建

# CrossApp基础概念

## 架构和目录结构

### 架构

### 目录结构

从github下载CrossApp资源包后，或解压稳定版ZIP压缩包后，就会看到如下的目录结构：

|  |  |
| --- | --- |
| **目 录** | **说 明** |
| **CrossApp** | 此目录为引擎的源码目录，其主要的两个目录  **CrossApp.h:**  所有引擎功能需要包含的头文件，对使用CrossApp开发应用的开发者，屏蔽了底层的系统差异，在需要跨平台的情况下，这个目录下的头文件会包含CrossApp\platform目录下对应的头文件。  **CrossApp\platform:**  用一堆#if宏判断当前操作系统，不同系统include相应系统下的头文件。CrossApp基于OPENGL ES（OPENGL的子集，专用于移动设备），所以底层的图像和动画绘制已经是跨平台了，所以cocos2d中真正需要跨平台的不多（platform下的头文件并不多），一些像消息循环响应机制等不同平台有不同方式的才需要跨平台 |
| **CocosDenshion** | 音频库 |
| **extensions** | json、http、sqlite3库等第三方库 |
| **licenses** | 本引擎依赖了很多的开源工程，他们的所有许可声明都在这里了。 |
| **scripting** | 支持的脚本 |
| **template** | 包含在不同IDE和平台下来创建CrossApp工程的模板 |
| **projects** | 工程目录(创建工程后才出现此目录，默认无) |
| **samples** | 示例Demo |
| **tools** | 各种脚本处理文件、包含工程创建等 |

## 核心类

CrossApp是基于MVC模式的，MVC指的是Model(模型)、View(视图)、Controller(控制)三者各司其职，共同构建一个完整的应用程序。CrossApp的核心类也是基于这样的设计思想来设计实现的。我们通过本节学习CAVIew、CAViewController、CAWindow等能更好的了解CrossApp的MVC结构的原理与实现，能帮助我们更深刻的理解引擎的使用方法。

### CAView

CAView视图类是CrossApp引擎最基本的类，负责将各式各样的界面呈现出来，我们在App中所能看到的一切界面其实是一个或多个CAView组合构建的。CAView负责在屏幕上定义矩形区域，在展示界面及响应用户界面交互方面发挥关键作用每个CView对象要负责渲染视图区域中的内容，并响应该区域内发生的操作事件，视图是应用程序用户交互的重要机制。

除了显示内容和处理事件之外，CAView还可以管理一个或多个子视图。我们可以在一个CAView上添加多个CAView或CAView的派生类，添加CAView的做为父节点，被添加的CAView被称为子节点。父节点负责管理其子节点，并根据需要调整他们的位置和尺寸，以及响应他们没有处理的事件。

我们看一下CAView常用的一些public函数:

|  |  |
| --- | --- |
| 函数 | 说明 |
| **static** **CAView\*** create(**void**) |  |
| **static** **CAView\*** createWithFrame(**const CCRect&** rect) |  |
| **static CAView\*** createWithFrame(**const CCRect&** rect, **const CAColor4B&** color4B) |  |
| Static CAView\* createWithCenter(const CCRect& rect) |  |
| Static CAView\* createWithCenter(const CCRect& rect,const CAColor4B& color4B) |  |
| Static CAView\* createWithColor(const CAColor4B& color4B) |  |
| Virtual bool init() |  |
| Virtual bool initWithFrame(const CCRect& rect) |  |
| virtual bool initWithFrame(const CCRect& rect, const CAColor4B& color4B) |  |
| virtual bool initWithCenter(const CCRect& rect) |  |
| virtual bool initWithCenter(const CCRect& rect, const CAColor4B& color4B) |  |
| virtual bool initWithColor(const CAColor4B& color4B) |  |
| Const char\* description(void) |  |
| Virtual void setZOrder(int zOrder) |  |
| Virtual int getZOrder() |  |
| Virtual void setVertexZ(float vertexZ) |  |
| Virtual float getVertexZ() |  |
| virtual void setScaleX(float fScaleX) |  |
| virtual float getScaleX() |  |
| virtual void setScaleY(float fScaleY) |  |
| virtual float getScaleY() |  |
| virtual void setScale(float scale) |  |
| virtual float getScale(); |  |
| virtual void setScale(float fScaleX,float fScaleY) |  |
| virtual void setSkewX(float fSkewX) |  |
| virtual float getSkewX() |  |
| virtual void setSkewY(float fSkewY) |  |
| virtual float getSkewY() |  |
| void setAnchorPoint(const CCPoint& anchorPoint) |  |
| const CCPoint& getAnchorPoint() |  |
| void setAnchorPointInPoints(const CCPoint& anchorPointInPoints) |  |
| const CCPoint& getAnchorPointInPoints() |  |
| virtual void setFrame(const CCRect& rect) |  |
| virtual const CCRect& getFrame() const |  |
| virtual void setFrameOrigin(const CCPoint& point) |  |
| virtual const CCPoint& getFrameOrigin() |  |
| virtual void setBounds(const CCRect& rect) |  |
| virtual CCRect getBounds() const |  |
| virtual void setCenter(CCRect rect); |  |
| virtual CCRect getCenter() |  |
| virtual void setCenterOrigin(const CCPoint& point) |  |
| virtual CCPoint getCenterOrigin() |  |
| virtual void setVisible(bool visible) |  |
| virtual bool isVisible() |  |
| virtual void setRotation(float fRotation) |  |
| virtual float getRotation() |  |
| virtual void setRotationX(float fRotaionX) |  |
| virtual float getRotationX(); |  |
| virtual void setRotationY(float fRotationY) |  |
| virtual float getRotationY() |  |
| virtual void addSubview(CAView \* child) |  |
| virtual void insertSubview(CAView\* subview, int z) |  |
| virtual CAView \* getSubviewByTag(int tag) |  |
| virtual CAView \* getSubviewByTextTag(const std::string& textTag) |  |
| virtual const CAVector<CAView\*>& getSubviews() |  |
| virtual unsigned int getSubviewsCount(void) const |  |
| virtual void setSuperview(CAView\* superview) |  |
| virtual CAView\* getSuperview() |  |
| virtual void removeFromSuperview() |  |
| virtual void removeSubview(CAView\* subview) |  |
| virtual void removeSubviewByTag(int tag) |  |
| virtual void removeSubviewByTextTag(const std::string& textTag) |  |
| virtual void removeAllSubviews() |  |
| virtual void reorderSubview(CAView \* child, int zOrder) |  |
| virtual void sortAllSubviews() |  |
| virtual bool isRunning() |  |
| virtual void draw(void) |  |
| virtual CAView\* copy() |  |

除了上述函数，CAView还提供坐标系转换，关于坐标系的问题，我将在后面的章节详细讲解。

### CAViewController

CAViewController作为CAView的管理，其最基本的功能就是控制视图的切换。视图控制器在MVC设计模式中扮演控制层(Controller)的角色，CAViewController的作用就是管理与之关联的CAView，同时与其他CAViewController互相通信和协调。

我们来看看CAViewController的常用函数

**Public**:

|  |  |
| --- | --- |
| 函数 | 说明 |
| bool isViewRunning() |  |
| void setNavigationBarItem (CANavigationBarItem\*item) |  |
| void setTabBarItem(CATabBarItem\* item) |  |
| void presentModalViewController(CAViewController\* controller, bool animated) |  |
| void dismissModalViewController(bool animated) |  |
| virtual bool isKeypadEnabled() |  |
| virtual void setKeypadEnabled(bool value) |  |
| virtual CAView\* getView() |  |

**Protected:**

|  |  |
| --- | --- |
| 函数 | 说明 |
| virtual void viewDidLoad() |  |
| virtual void viewDidUnload() |  |
| virtual void reshapeViewRectDidFinish() |  |

### CAWindow

CAWindow是所有CAView的载体，并且负责分发触摸消息，协同CAViewController完成对应程序的管理。应用程序通常只有一个CAWindow，及时存在多个CAWindow也只有一个CAWindow能接受屏幕事件。应用程序启动时创建一CAWindow，并将多个CAView添加到CAWindow并显示出来，之后我们就很少再次的引用它。CAWindow是屏幕上显示的CAView的根节点。

## 内存管理

### CAObject

CAObject是CrossApp引擎的根类，主要是进行内存的管理，定义部分回调函数指针。引擎绝大部分类派生自CAObject，其共同遵从同一套内存管理方式。

CAObject主要的函数有:

|  |  |
| --- | --- |
| **函数** | **说明** |
| **void** retain(void) | 对象计数器+1 |
| **void** release(void) | 对象计数器-1 |
| **void** autorelease(void) | 设置内存自动管理 |
| **unsigned int** retainCount(void) | 返回当前对象计数器的值 |
| **bool** isSingleReference(void) | 实例化对象是否只有一个使用者 |
| **bool** isEqual(**const CAObject** \*pObject) | 与指定object实例对象是否相同 |

## 坐标系

## 适配方案

# CrossApp核心控件

# CrossApp数据存储与解析

在开发app时，我们需要对数据持久保持和解析，CrossApp也对常见的数据类型解析提供了支持如json、xml、sqlite3等。

## CAUserDefault简单存储

在CrossApp中提供了自带的存储类：CAUserDefault,适合存储数据量比较小，结构比较简单的数据。如果你需要存储大量的复杂数据，建议使用SQlite比较合适。

**先介绍一下CAUserDefault的API:**

|  |  |
| --- | --- |
| **方法名** | **说明** |
| **void** setBoolForKey(**const char\*** pkey,**bool** valuer) | 根据pkey存储一个bool类型 |
| **void** setIntegerForKey(**const char\*** pkey,**int** valuer) | 根据pkey存储一个int类型 |
| **void** setFloatForKey(**const char\*** pkey,**float** valuer) | 根据pkey存储一个float类型 |
| **void** setDoubleForKey(**const char\*** pkey,**double** valuer) | 根据pkey存储一个double类型 |
| **void** setStringForKey(**const char\*** pkey, **const std::string &** value) | 根据pkey存储一个string类型 |
| **bool** getBoolForKey(**const char\*** pKey) | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回false |
| **bool** getBoolForKey(**const char\*** pKey,**bool** defaultValue) | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回defaultValue |
| **int** getIntegerForKey**(const char\*** pKey**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回0 |
| **int** getIntegerForKey**(const char\*** pKey**, int** defaultValue**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回defaultValue |
| **float** getFloatForKey**(const char\*** pKey**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回0.0f |
| **float** getFloatForKey**(const char\*** pKey**,float** defaultValue**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回defaultValue |
| **double** getDoubleForKey**(const char\*** pKey**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回0.0 |
| **double** getDoubleForKey**(const cha \*** pKey**,double** defaultValue**)** | 根据pkey读取相应的值，如果没有在UserDefault.xml找出则返回defaultValue |
| **void** flush() | 存储到UserDefault.xml，不写的话不会存入 |
| **const string&** getXMLFilePath() | 获得UserDefault.xml的存储路径 |

**存储代码如下:**

//存储名字

CAUserDefault::sharedUserDefault()->setStringForKey("name", "zero");

//存储年龄

CAUserDefault::sharedUserDefault()->setIntegerForKey("age", 25);

//存储身高

CAUserDefault::sharedUserDefault()->setDoubleForKey("height",1.75);

//存储体重

CAUserDefault::sharedUserDefault()->setFloatForKey("wight", 75.0f);

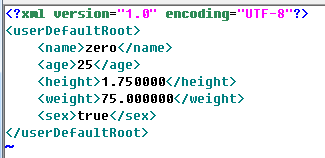
//存储性别 true为男 fasle 女

CAUserDefault::sharedUserDefault()->setBoolForKey("sex", true);

//这里一定要提交写入哦，否则不会记录到xml中，下次启动游戏你就获取不到value了

CAUserDefault::sharedUserDefault()->flush();

**我们将看到UserDefault.xml文件如下:**



**读取示例代码:**

//读取name

std::string name = CAUserDefault::sharedUserDefault()->getStringForKey("name");

//读取Email,如果没有Email则返回"default"

std::string email = CAUserDefault::sharedUserDefault()->getStringForKey("Email","default");

//读取age

int age = CAUserDefault::sharedUserDefault()->getIntegerForKey("age");

//读取id,如果没有id则返回10000

int id = CAUserDefault::sharedUserDefault()->getIntegerForKey("id", 10000);

//读取身高体重，如果没有找到则返回0

float height =CAUserDefault::sharedUserDefault()->getFloatForKey("height");

double weight =CAUserDefault::sharedUserDefault()->getDoubleForKey("weight");

//读取sex，如果没有找到sex则返回true

bool sex = CAUserDefault::sharedUserDefault()->getBoolForKey("false", true);

## SQLite的使用

在CrossApp中，简单数据存储，可以使用CAUserDefault。那么如何存储大量，不规则的数据？我们可以使用SQLite数据库存储数据。SQLite是使用非常广泛的嵌入式数据库，它有小巧 、高效、跨平台、开源免费和易操作的特点。

SQLite数据库是使用C语言来编写的,在CrossApp中使用也是非常容易的。

CrossApp已经添加了SQlite的，在CrossApp\extensions\sqlite3目录，我直接使用就可以了。

**引入头文件**

#include "CrossAppExt.h"

**创建数据库**

//数据库指针

sqlite3 \*pdb=NULL;

//保存数据库的路径

std::string path= FileUtils::getInstance()->getWritablePath()+"save.db";

std::string sql;

int result;

//打开一个数据，如果该数据库不存在，则创建一个新的数据库文件

result=sqlite3\_open(path.c\_str(),&pdb);

if(result!=SQLITE\_OK)

{

CCLog("open database failed, number%d",result);

}

**SQL语句**

//创建数据库表的sql语句

sql="create table student(ID integer primary key autoincrement,name text,sex text)";

**创建Talbe**

//创建表格

result=sqlite3\_exec(pdb,sql.c\_str(),NULL,NULL,NULL);

if(result!=SQLITE\_OK)

CCLog("create table failed");

**插入**

//向表内插入3条数据

sql="insert into student  values(1,'student1','male')";

result=sqlite3\_exec(pdb,sql.c\_str(),NULL,NULL,NULL);

if(result!=SQLITE\_OK)

    CCLog("insert data failed!");

sql="insert into student  values(2,'student2','female')";

result=sqlite3\_exec(pdb,sql.c\_str(),NULL,NULL,NULL);

if(result!=SQLITE\_OK)

    CCLog("insert data failed!");

sql="insert into student  values(3,'student3','male')";

result=sqlite3\_exec(pdb,sql.c\_str(),NULL,NULL,NULL);

if(result!=SQLITE\_OK)

    CCLog("insert data failed!");

**查询**

//查询结果

char \*\*re;

//行、列

int r,c;

//查询数据

sqlite3\_get\_table(pdb,"select \* from student",&re,&r,&c,NULL);

CCLog("row is %d,column is %d",r,c);

 //将查询出的数据通过CCLog输出

for(int i=1;i<=r;i++)

{

    for(int j=0;j<c;j++)

    {

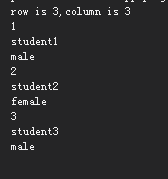
        CCLog("%s",re[i\*c+j]);

    }

}

sqlite3\_free\_table(re);

打印结果:



**删除**

sql="delete from student where ID=1";

//删除id=1的学生的信息

result=sqlite3\_exec(pdb,sql.c\_str(), NULL,NULL,NULL);

if(result!=SQLITE\_OK)

    CCLog("delete data failed!");

注意

使用sqlite一定要注意的内存管理问题，那就是打开数据库之后，数据操作完成之后，一定要关闭数据库，否侧会造成内存泄漏。

sqlite3\_close(pdb);

**SQlite保存路径**

Android

/data/data/com.youCompany.Helloworld/files/save.db

IOS

位于程序沙盒的文档目录下

../Documents/save.db

## Json解析

CrossApp使用lib\_json来解析json文件。lib\_json已经加入了libExtensions下，我们在CrossApp中使用非常的便捷。

首先引入头文件

#include "CrossAppExt.h"

using namespace CSJson;

我们先熟悉一下几个类名和函数

/\*

Value:写过脚本、弱语言的童鞋应该很清楚var，其他Value 和var一个道理，都是可以表示很多数据类型的数据类型，。

这话可能比较绕，简单说就是Value你可以理解可以是int 也可以是string 也可以是其他数据类型。

当然定义 Value value,只是个定义，还没有决定其数据类型，如果你Value value =10;那么value 就是个整型

在用于JSON时，我们常表示为一个map，其中包括 key-value，键值对

其中Value 中包括一些将其转为基础数据类型的6个方法，如下：

value.asCString();

value.asString();

value.asBool();

value.asDouble();

value.asInt();

value.asUInt();

\*/

Value map;

/\*

FastWriter:起作用是将Value数据编码成JSON格式的数据

常用函数：write(<#const Json::Value &root#>)

\*/

Json::FastWriter write;

/\*

Value:作用与FastWriter相反,是将JSON格式的数据解析成一个Value

常用函数： reader.parse(<#std::istream &is#>, <#Json::Value &root#>)

\*/

Json::Reader reader;

通过上面的注释，我们可以很清楚的了解函数的意义。下面我们开始生成Json数据和解析Json数据。

### Json数据生成

//先定义数据

Value map;

map["name"] = "9miao";

map["password"] = "123456";

map["Email"] = "9miao@longtugame.com";

map["PHONE"] = 10086;

//编码成json数据

FastWriter write;

string jsonData = write.write(map);

//打印结果

CCLog("jsonData:%s", jsonData.c\_str());

上面的打印结果：

jsonData:{"Email":"9miao@longtugame.com","PHONE":10086,"name":"9miao","password":"123456"}

### Json数据解析

有时候我们需要解析Resources目录下的json文件，首先我们就需要将json文件拷贝到Resources目录下。我们将下面的Json格式文件拷贝到Resources目录下命名为info.json.

{

"info":

[

{"name":"aaa","num":"0001"},

{"name":"bbb","num":"0002"},

{"name":"ccc","num":"0003"},

{"name":"ddd","num":"0004"},

{"name":"eee","num":"0005"},

{"name":"fff","num":"0006"},

{"name":"ggg","num":"0007"},

{"name":"hhh","num":"0008"},

{"name":"iii","num":"0009"},

{"name":"jjj","num":"0010"},

{"name":"kkk","num":"0011"},

{"name":"lll","num":"0012"},

{"name":"mmm","num":"0013"},

{"name":"nnn","num":"0014"},

{"name":"ooo","num":"0015"},

{"name":"ppp","num":"0016"}

],

"gender": "male",

"occupation": "coder"

}

我们在程序中添加以下代码对其解析：

Reader reader;

//定义Valuer

Value value;

//json文件路径

string jsonFile = CCFileUtils::sharedFileUtils()->fullPathForFilename("info.json");

//将文件生成CCString对象

CCString\* jsonData = CCString::createWithContentsOfFile(jsonFile.c\_str());

//将数据解析到value中

if (reader.parse(jsonData->getCString(),value))

{

int length = value["info"].size();

//循环解析子节点

for (size\_t index = 0; index < length; index++)

{

std::string name = value["info"][index]["name"].asString();

std::string num = value["info"][index]["num"].asString();

CCLog("name:%s", name.c\_str());

CCLog("num:%s", num.c\_str());

}

//获取方式一

Value valueGender;

valueGender = value.get("gender", valueGender);

std::string gender = valueGender.asCString();

//获取方式二

std::string occupation = value["occupation"].asCString();

CCLog("gender:%s", gender.c\_str());

CCLog("occupation:%s", occupation.c\_str());

}

## XML解析

CrossApp已经加入了tinyxml2库用于xml解析。#include "CrossApp.h"时候已经包含tinyxml2.h无须再引入头文件，这样我们就能在开发时方便的生成和解析xml文件。

### xml文档生成

**命名空间**

using namespace tinyxml2;

**生成xml代码**

//获得xml的保存路径

std::string filePath = CCFileUtils::sharedFileUtils()->getWritablePath() + "test.xml";

//在生成一个XMLDocument对象

tinyxml2::XMLDocument \*pDoc = new tinyxml2::XMLDocument();

//xml 声明（参数可选）

XMLDeclaration \*pDel = pDoc->NewDeclaration("xml version=\"1.0\" encoding=\"UTF-8\"");

//将pDel添加到XMLDocument对象中

pDoc->LinkEndChild(pDel);

//添加plist节点

XMLElement \*plistElement = pDoc->NewElement("plist");

//设置l版本

plistElement->SetAttribute("version", "1.0");

//将pDec添加到XMLDocument对象中

pDoc->LinkEndChild(plistElement);

//添加一行注释

XMLComment \*commentElement = pDoc->NewComment("this is xml comment");

//将注释添加到XMLDocument对象中

plistElement->LinkEndChild(commentElement);

//添加dic节点

XMLElement \*dicElement = pDoc->NewElement("dic");

plistElement->LinkEndChild(dicElement);

//添加key节点

XMLElement \*keyElement = pDoc->NewElement("key");

keyElement->LinkEndChild(pDoc->NewText("Text"));

dicElement->LinkEndChild(keyElement);

XMLElement \*arrayElement = pDoc->NewElement("array");

dicElement->LinkEndChild(arrayElement);

for (int i = 0; i<3; i++) {

XMLElement \*elm = pDoc->NewElement("name");

elm->LinkEndChild(pDoc->NewText("9miao"));

arrayElement->LinkEndChild(elm);

}

pDoc->SaveFile(filePath.c\_str());

CCLog("path:%s", filePath.c\_str());

pDoc->Print();

delete pDoc;

**生成XML如下：**

<?xml version="1.0" encoding="UTF-8"?>

<plist version="1.0">

<!--this is xml comment-->

<dic>

    <key>Text</key>

    <array>

        <name>9miao </name>

        <name>9miao </name>

        <name>9miao </name>

    </array>

</dic>

</plist>

### 解析XML

下面我们就来解析一下上面生成的XML文档

解析代码：

1. //解析xml的路径
2. std::string filePath = CCFileUtils::sharedFileUtils()->getWritablePath() + "test.xml";
3. //生成一个XMLDocument对象
4. tinyxml2::XMLDocument \*pDoc = new tinyxml2::XMLDocument();
5. //将xml文件读取到XMLDocument对象中
6. XMLError errorId = pDoc->LoadFile(filePath.c\_str());
7. if (errorId != 0) {
8. //xml格式错误
9. return;
10. }
11. XMLElement \*rootEle = pDoc->RootElement();
12. //获取第一个节点属性
13. const XMLAttribute \*attribute = rootEle->FirstAttribute();
14. //打印节点属性名和值
15. CCLog("attribute\_name = %s,attribute\_value = %s", attribute->Name(), attribute->Value());
16. XMLElement \*dicEle = rootEle->FirstChildElement("dic");
17. XMLElement \*keyEle = dicEle->FirstChildElement("key");
18. if (keyEle) {
19. CCLog("keyEle Text= %s", keyEle->GetText());
20. }
21. XMLElement \*arrayEle = keyEle->NextSiblingElement();
22. XMLElement \*childEle = arrayEle->FirstChildElement();
23. while (childEle) {
24. CCLog("childEle Text= %s", childEle->GetText());
25. childEle = childEle->NextSiblingElement();
26. }
27. delete pDoc;

打印结果：

attribute\_name = version,attribute\_value = 1.0

keyEle Text= Text

childEle Text= 9miao

childEle Text= 9miao

childEle Text= 9miao

**注意：**

tinyxml在android上解析assert文件夹下会有问题：解决方式如下：

1. std::string filePath = CCFileUtils::sharedFileUtils()->fullPathFromRelativeFile("test.xml");

# CrossApp设备功能调用

# CrossApp进阶

# CrossApp网络通信

## Http

CrossApp为我们封装了http的网络框架，其文件在CrossApp引擎包的\extensions\network文件下的 HttpClient、HttpRequest 、HttpResponse。

进行一次http交互，需要涉及的有三个类：

**HttpRequest**用来描述一个请求。

**HttpResponse**用来描述对应请求的响应。

**HttpClient**是一个单例模式的类，它的职责就是负责将收到的HttpRequest对象push到发送队列中，并发送一个信号量驱动工作线程工作，工作线程再将收到的数据封装成一个HttpResponse对象push接收队列，并启用调度来派送数据。

引用头文件：

#include "CrossAppExt.h"

**请求**

1. //创建request对象,这里new出来的对象不能使用autorelease()，原因后述
2. CCHttpRequest\* request = **new** CCHttpRequest();
3. //设置URL
4. request->setUrl("www.9miao.com");
5. //设置请求类型kHttpGet、kHttpPost、KHttpPostFile、kHttpPut、kHttpDelete、kHttpUnkown
6. request->setRequestType(CCHttpRequest::kHttpGet);
7. //这是回调对象和回调函数
8. request->setResponseCallback(**this**, httpresponse\_selector(FirstViewController::requestresult));
9. //设置用户标识，可以通过response获取
10. request->setTag("get");
11. //使用CCHttpClient共享实例来发送request
12. CCHttpClient::getInstance()->send(request);
13. //调用release()
14. request->release();

**接收**

1. void FirstViewController::requestresult(CCHttpClient\* clinet, CCHttpResponse\* response)
2. {
3. if (!response->isSucceed())
4. {
5. return;
6. }
7. //获取返回代码，比如200、404等
8. int statusCode = response->getResponseCode();
9. if (!strcmp("get", response->getHttpRequest()->getTag()) && (statusCode == 200))
10. {
11. std::string responseRes = "";
12. std::vector<char> \*buffer = response->getResponseData();
13. for (unsigned int i = 0; i < buffer->size(); i++)
14. {
15. responseRes += (\*buffer)[i];
16. }
17. //查找字符“官方特约”
18. string::size\_type idx = responseRes.find(UTF8("官方特约"));
19. if (idx == -1)
20. {
21. //为找到字符
22. CCLog(" Not Found");
23. return;
24. }
25. string temp = responseRes.substr(idx, 30);
26. string num = temp.substr(temp.find(UTF8("：")) + 3, temp.find("<") - temp.find(UTF8("：")) - 3);
27. CCLog("temp:%s",temp.c\_str());
28. }
29. else
30. {
31. //打印返回代码
32. CCLog("statusCode:%d", statusCode);
33. }
34. }

## Socket

# 折800—实战讲解

# 动漫之家—实战讲解