NPL 语言教程

文件编号：	版本：V1.0	日期：
编制：陈章统 李文龙 万昌龙 何睿智 彭照志 张一	审核：陈细杰	批准：

IdreamTech

深圳市爱致科技有限公司

修订履历

修改时间	修改概要	修改后版本	修改人	批准人
2017-05-27	初稿	V1.0		

IdreamTech

目录

1 NPL 介绍 6

2 快速入门 8

2.1 安装 NPL Runtime 8

2.2 Helloworld 程序 10

2.3 开发环境搭建 10

2.4 问题反馈 12

3 基本概念 12

3.1 Boostrapper 12

3.2 NPL 命令 13

3.3 NPL Load 15

3.4 Activation File 21

3.5 多线程 30

3.6 Concurrency Model 32

3.7 Common Library 37

3.8 NPL Packages 38

4 C/C++ NPL RUNTIME API 44

4.1 Core API 44

4.2 Attribute System 44

4.3 AssetManifest 45

4.4 PKG file loading 46

4.5 ParaObject 47

IdreamTech

4.6 ParaUIObject 48

5 SYSTEM LIBRARIES 48

5.1 Timer 48

5.2 Serialization 49

5.3 Encoding 50

5.4 Log 53

5.5 Http request 53

5.6 FilesAndIO 57

6 UI 69

6.1 用 2D API 绘制 69

6.2 使用 MCML 73

7 3D PROGRAMMING 80

7.1 3D Scene 80

7.2 地形 83

7.3 摄像机 84

7.4 3D 文件格式 86

7.4.1 通过 NPL 脚本读取 ParaX 文件内容 86

7.4.2 BMax 文件格式 88

7.4.3 ParX 文件格式 90

7.4.4 ParaX 文件 LOD 91

7.4.5 3D 世界文件格式 92

IdreamTech

7.4.6 全局地形格式 Global Terrain Format 93

7.4.7 方块世界文件格式 94

7.5 Mini Scenegraph 94

7.6 方块引擎 95

7.7 像素拾取 96

8 WEB 编程 97

8.1 Networking 97

8.2 WebServer 99

8.2.1 NPL Server Page 110

8.2.2 NPL Admin Site 115

8.2.3 Using TableDatabase 117

8.2.4 MySQL & PostgreSQL 123

8.2.5 HTTPS SSL 服务器 125

8.2.6 NPL 网络套接字服务端 129

IdreamTech

1 NPL 介绍

NPL 的全称是 Neural Parallel Language，是一种新的游戏引擎脚本语言。NPL 源代码完全用 C/C++编写，然后将 API 暴露给 Lua， 所以 NPL 的应用程序一般用 Lua 语言编写。 NPL 的设计以人类神经网络结构为参考，在 NPL 中，每个文件相当于一个神经元，文件之间的信息交互相当于神经网络中的神经元之间的信息传递。NPL 有以下几个特点：

1. NPL 是一种通用开源的语言，其语法与 Lua 100%兼容；
2. NPL 包含一个内置的 3D 引擎，叫作 ParaEngine，提供基本的编写 3D/2D/Web 应用程序的功能；
3. NPL 提供丰富的 C/C++ API 和丰富的 NPL 脚本库；
4. NPL 是一个单独的编程语言解决方案，主要针对下列技术：先进的交互式 GUI、复杂的 opengl/DirectX 3D 图像技术、可扩展的网站服务器，快速交互的数据库技术及分布式软件框架。它是跨平台的、高性能的、可扩展的且可调试的；
5. NPL 是一个最初被设计像大脑一样运行的语言系统。节点和连接无处不在，为了用户更加便捷的开发复杂的网络应用，线程和底层网络逻辑在 NPL 中被隐藏起来了；
6. NPL 可以将抢占式用户模式和非抢占式用户模式混合在一起。这在同一个动态的、弱类型的语言中提供了像 Erlang 一样的并行模式以及像 Java/C++一样的速度；
7. NPL 还支持 C++, C#, Mono .Net.未来还支持 JavaScript。

相比较与以下的编程语言，NPL 有以下的优缺点：

1. C/C++:

   优点：NPL 是一种易于扩展的脚本语言，动态编译并且容易使用。

   缺点：实际上，C++是唯一的跨平台语言， 但是 NPL 是在 C++的基础上编写的。

2. Mono C#, Java:

   优点：在 Windows 平台上部署相对简单，开发 C++插件简单，调用 C++函数相对更加快速。NPL 是弱类型语言，动态编译的。

   缺点：Java 更加适合于服务器和任务繁重的客户端开发，有丰富的库。

IdreamTech

3. Node.js, Electron:

   优点：在 Windows 平台上部署相对简单，开发 C++插件简单。

   缺点：Node.js, Electron 更加适合于服务器和任务繁重的客户端开发。HTML5 和 JavaScript 更加受欢迎。

4. Python, Ruby, ActionScript 和其他类似的:

   优点：性能更好，客户端开发更加更加简便，作为一种脚本语言，语法相对更加宽松。

   缺点：Python 等更加受欢迎且模块化。

5. Lua:

   优点：NPL 是完全兼容 lua，并且是完全独立的一种语言，而不像 Lua 只能作为嵌入式语言，有独立通用的运行环境和底层 API 以及丰富的库用以支持客户端和服务端的复杂开发。

   缺点：真正的动态编译语言，在 C/C++基础上有良好的扩展性，对于 C/C++开发者来说，语法非常清晰。

接下来介绍与 NPL 相关的源代码，NPL 的初学者经常会接触以下三个源代码文件：NPL Runtime, Main package, Paracraft. NPL Runtime 是 NPL 运行环境的实现代码，用 C++语言编写，这个文件可分为三层来理解：底层是 NPL 线程的实现，对 lua, C#, C/C++语言的编译执行；中间层是 NPL 线程，文件的调度；顶层是 NPL 的 API，涵盖网络连接，资源管理，2D 图形，3D 引擎等 。 Main package 是用 NPL API 编写的高级函数库， 这样用户可以更加方便的编写 NPL 程序， 当然用户也可以用 NPL API 开发自己的库。Paracraft 是用 NPL 语言编写的一个 3D 应用，同时用到了 NPL API 和 Main package，可作为 NPL 的开发环境来使用。三者的关系图可以用图 1.1 来描述：

IdreamTech

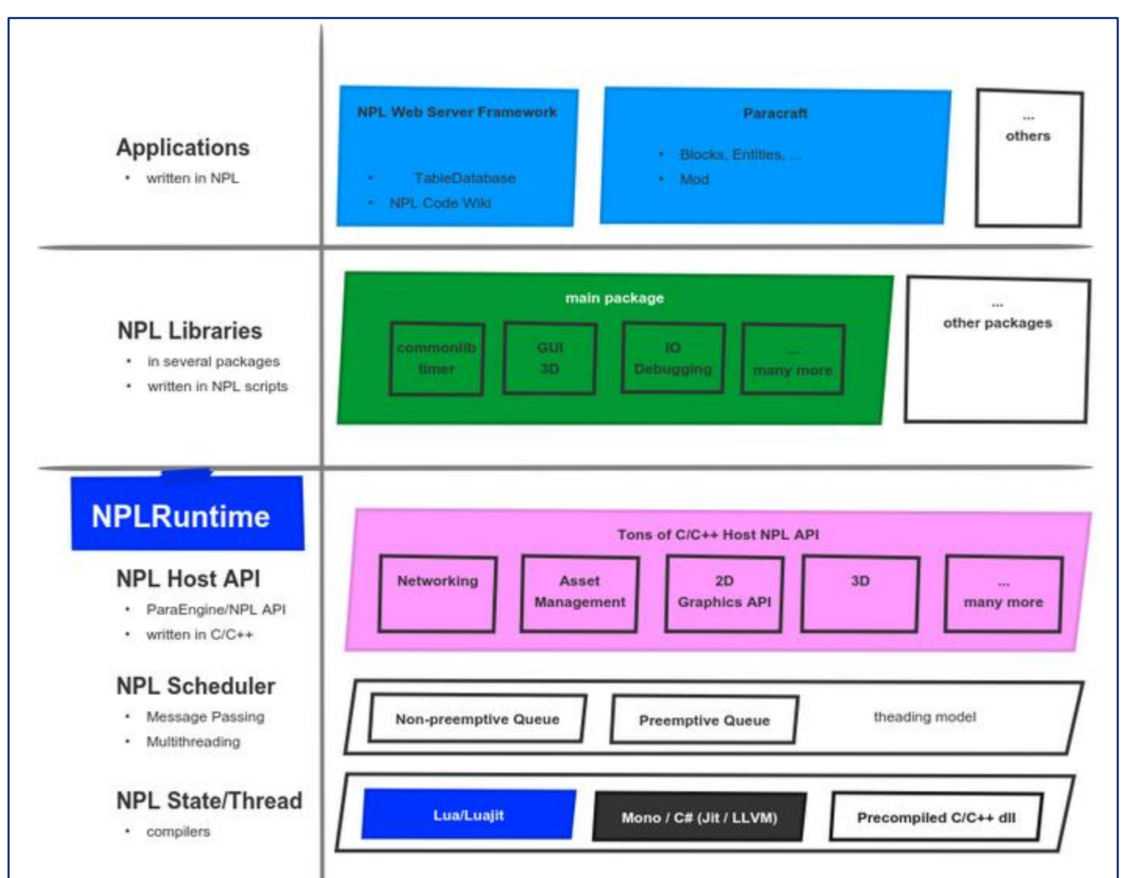


图 1.1

2 快速入门

2.1 安装 NPL Runtime

Windows

（1） 用 installer 安装

在 windows 上安装 NPLRuntime 非常简单，目前发布了两种 NPL runtime 的 win32 可运行版本：

一个是 32bits/64bits 的开发版本（推荐用于单机版的应用或者模块开发），下载网址为：https://github.com/LiXizhi/NPLRuntime/releases；

另一个是发布在 ParacraftSDK 中的官方稳定版本（推荐用于 paracraft 模块开发），下面的步骤会指导你安装：

下载 ParacraftSDK，下载网址为：https://github.com/LiXizhi/NPLRuntime/wiki/Install

Guide；

运行 redist\paracraft.exe 安装 Paracraft（同时也会安装 NPLRuntime 和 main pkg）；

运行 NPLRuntime/install.bat 会配置 NPLRuntime（即从./redist 复制 NPLRuntime 相关文件到./NPLRuntime/win/bin， 把上述 bin 目录添加进环境变量）。 后期更新只需定期运行上述两个文件。

（2） 用源文件安装， 步骤如下：

1） 下载并且编译 boost，版本号为 1.55.0 和 1.60.0 的 boost 是经过测试可用的，最新版本应该也是可用的， 如果你下载的 BOOST 文件放在 D:\boost 添加环境变量，可在命令行运行下列命令：

```
bootstrap

b2 --build-type=complete
```

上面的命令是编译 boost 的所有部分，速度较慢，用户可以只编译用到的部分，编译速度也相对更快：

```
b2 runtime-link=static --with-thread --with-date_time --with-filesystem --with-system --with-chrono --with-signals --with-serialization --with-iostreams --with-regex stage
```

编译过后，增加一个环境变量，名称是 BOOST_ROOT， 值是 D:\boost（或者你的 boost 目录），这样的话编译工具就能知道 boost 的路径；

2）下载 NPLRuntime 源代码，下载网址是：https://github.com/LiXizhi/NPLRuntime；

3）下载 cmake，编译 NPLRuntime 的客户端或者服务器端版本，它们共享一个源代码文件，但是 cmakelist.txt 不同；客户端版本包含所有功能， 服务器版本多了图像 API。

对于服务器版本：在 cmake 工具中，在 NPLRuntime/bin/win32 目录下（任何目录都行，但是不能是 CMakeList.txt 所在的目录）运行 NPLRuntime/NPLRuntime/CMakeList.txt。

对于客户端版本：在 cmake 工具中，在 NPLRuntime/bin/Client 目录下（任何目录都行，

但是不能是 CMakeList.txt 所在的目录）运行 NPLRuntime/Client/CMakeList.txt。在这之前还需要安装最新版本的 DirectX9 SDK。

如果上述步骤都成功了，那么就会生成一个 visual studio 解决方案，可以用 visual studio 打开并且编译运行。最终的可运行文件在./ParaWorld/bin/目录下。

Linux

在 linux 环境下，非常推荐用 NPLRuntime 源代码和编译工具编译 NPL runtime。步骤如下：

a. 下载 NPLRuntime 源代码；
b. 安装第三方 NPL 开发库和最新的编译工具， 一般情况下，你需要 gcc 4.9 或者更高版本,cmake, bzip2, boost, libcurl. 当然你需要编译相应的 NPL 插件，也可以选择性的安装 mysql, mono, postgresql.
c. 在根目录下运行 ./build_linux.sh，可能会需要较长的时间。需要注意的是你至少需要 2GB 的内存来编译。
d. 上述步骤都完成了的话，最终的执行文件会在./ParaWorld/bin64/目录下； 并且在 /usr/local/bin/npl 目录下会生成一个象征链接，这样你可以在任何地方运行 NPL 脚本。

2.2 Helloworld 程序

把目录_YourParaCraftSDKPath_/NPLRuntime/win/bin 添加进环境变量; 在上述目录下编写 Hello.lua， 在命令行运行,在 log 中会看到输出 hello world 的输出。Hello.lua 文件的内容如下：

```
Print(“hello world!”)
```

2.3 开发环境搭建

开发 NPL 项目有三种开发环境: visual studio, NPL code wiki, lua studio

推荐使用 lua studio。

(1) visual studio

安装 visual studio community edition。

在 vs 中， 安装 NPL_LuaLanguageService 和 NPL_LuaDebuggerPackage。在 vs 中，选择 menu::Tools::Extensions， 然后在线搜索上述两个插件进行安装。

安装好 vs 和插件之后，可以开发 npl 项目了。在 vs 中创建空 C++或者 C#项目，在项目中创建编写 lua 文件，并且对项目的相关属性进行如下配置：

a. 在调试栏中把命令属性设置为_ParaCraftSDK 目录_\redist\paraengineclient.exe；
b. 把命令参数属性设置为 booststrapper="你的程序入口文件" dev="你的项目工作目录"；
c. 工作目录设置为你的项目工作目录，和 dev 属性一样。

点击运行按钮或者 F5 运行， 在日志文件中看输出结果。

(2) NPL code wiki

这是一个可以运行 NPL 脚本的网站，你可以用本地 NPL Runtime 通过任何一个浏览器访问。NPL code wiki 可以帮助用户学习和联系 NPL 编程，它提供了通过 HTTP 协议检查和调试 NPL 代码的方法。

访问 NPL code wiki 可以运行

npl _ParaCraftSDK_/redist/script/apps/WebServer/WebServer.lua

进入， 或者安装 paracraft, 创建一个空世界，按 F11 进入.

（3） lua studio

因为 NPL 项目完全是使用 lua 语言开发的，所以可以借助一个编写 lua 脚本的 IDE lua studio 中开发 NPL 项目，配置步骤如下：

a. 用 cmake 编译 NPLRuntime 客户端；
b. 在 vs 中编译项目， 在该目录下：_your NPLRuntime-master_\Client\lib\Debug 找到 ParaEngineClient_d.pdb；放到该目录下： _your ParaCraftSDK_-master\redist;
c. 打开 lua studio，创建项目，编写 npl 程序，对项目的属性作相关设置，可参考 vs 中对项目的属性设置；
d. 运行,设置断点调试。

IdreamTech

2.4 问题反馈

如果用户在进行 NPL 开发的时候遇到任何问题可以在 NPL 的 github 网站上咨询，网址是：

https://github.com/LiXizhi/NPLRuntime/issues

3 基本概念

3.1 Boostrapper

Bootstrapping:

Bootstrapping 是指定加载第一个脚本当开启 NPL Runtime 时，它也会每 0.5 秒启动一次。通过命令行有很多种方法做到 bootstrapping。

推荐的做法是明确的指定参数 `bootstrapper=filename`。例如：

```
npl bootstrapper=”script/gameinterface.lua”
```

如果没有指定文件名，`script/gameinterface.lua` 将被使用。更简单方式是用第一个参数来指定文件名，例如：

```
npl helloworld.lua
```

内部的启动命令如下：

1. 当 NPL Runtime 启动时，它首先读取所有命令行参数如：名称，数值，然后将它们存入内部的数据结构方便之后使用。
2. NPL 识别多个预编译的命令名称，其中就有 bootstrapper。它具体指定了主循环文件。
3. NPL 自身会初始化，这个过程花费几秒钟的时间，包括读取配置、初始化图像，绑定脚本 API 等。
4. 当所有的初始化完成后，多数的 NPL 的核心 API 对于脚本接口是可以被获取的并且 NPL 加载主循环脚本是被 bootstraper 具体指定的而且每 0.5 秒调用一次它的活跃的函数。
5. 之后发生的任何操作都取决于写主循环脚本的程序。

IdreamTech

另外，对于脚本文件来说，XML 文件也可以用来坐 bootstrapper。XML 文件内容样式如下：

```
<?xml version="1.0" ?>
<MainGameLoop>(gl)script/apps/Taurus/main_loop.lua</MainGameLoop>
```

我们可以用如下命令运行：

```
npl script/apps/Taurus/bootstrapper.xml
```

理解文件路径：

当用 NPL 编程时，代码会根据文件名参考其他脚本或者有用的文件。对一个文件的搜索顺序如下：

1. 检查文件是否存在当前开发的目录地址中（在释放时间，这个和工作目录一致）。
2. 检查文件是否存在于搜索路径（默认当前的工作目录）。
3. 检查文件在加载队列中是否存在于任何已加载的档案文件（主要为 XXX.PKG 或世界/插件的 ZIP 文件）
4. 如果文件为 NPL 脚本文件，在之前的地方或者队列中的`./bin/`*`filename`*`.o` 中寻找预编译的二进制文件。
5. 检查硬盘文件是否存在于所有 npl_package 的搜索路径中。
6. 如果以上所有情况都没有找到，我们将报告文件没有找到。注意，一些 API 需要你使用特定的搜索命令，在一些地方暂时不可用，或者包含确切的目录像或者例如当前可写的目录。

3.2 NPL 命令

本地参数：

以下命令行是内置且可执行于 NPL 的：

- bootstrapper="myapp/main.lua"：设置 bootstrapper 文件
- [-d|D] [filename]: -d 使它运行在后台方(守护程序模式), -D 时强制在前方运行
- dev="C:/MyDev/"：设置 dev 工作目录，如果空或者为 "/"，则是当前的工作目录
- servermode="true"：不提供 3D 且强制运行为服务端模式
- logfile="log2016.5.20.txt"：改变默认得 log.txt 位置

IdreamTech

- single="true"：强制仅仅一个可执行的实例可以在操作系统上运行
- loadpackage="folder1,folder1,...":逗号分隔开了在 bootstrapping 前要加载得软件包列表，例如：loadpackage="npl_packages/paracraft/" npl_packages/main 常常默认加载

NPL 系统模型参数：

以下的参数是被 NPL 系统模型处理的：

- debug="main" :在 NPL 主线程中启动 IPC 调试，没有对性能不利
- resolution="1020 680"：如果是客户端应用，强制现在窗口大小

Paracraft 模型参数：

以下参数是被 Paracraft Package 处理的：

- world="worlds/DesignHouse/myworld" ：开始时加载一个已给得世界
- mc="true"：强制 paracraft (如果在 MagicHaqi 目录下运行)
- httpdebug="true"：在 http://127.0.0.1:8099 启动 npl http 调试器控制台

从 NPL 脚本中获取命令行参数：

用以下方式可以获取已给的命令行参数：

```
-- for example, if you started npl process with `npl param1="abc"`, param1 will be filled with "abc"
local param1 = ParaEngine.GetAppCommandLineByParam("param1", "default_value");
echo(ParaEngine.GetAppCommandLineByParam("bootstrapper", ""))
```

用以下命令可以获取全部命令行字符串：

```
local cmdline = ParaEngine.GetAppCommandLine();
echo(cmdline)
```

工作目录：

工作目录是读/写文件默认的根目录。默认情况下，NPLRuntime 不更改可执行文件的工作目录，但是有一种情况除外。

如果可执行文件的文件夹或其付上层文件夹包含 ParaEngine.sig 文件（文件内容不重要），那么 NPLRuntime 将会重新设置工作目录到包含 ParaEngine.sig 文件的文件夹中。

一些 NPL 应用，例如 Paracraft 的 redist 文件包含 ParaEngine.sig 文件，这意味着你

可以从任何文件夹开启可执行的 paracraft，但是工作目录经常重新设置到 redist 文件夹。这种行为影响到最先的 IO 运行，包括默认的日志位置。

3.3 NPL Load

NPL 加载文件：

在 NPL 代码中，你可以使用例如 NPL.load(“script/ide/commonlib.lua”)的命令去加载一个文件。为了可以运行，所有的 NPL 代码需要被编译。NPL.load 会自动帮你做如下的操作：

- 在 NPL zip 压缩文件夹或根据定义在 npl_packages 中的 NPL 搜索路径寻找正确的文件并且如果存在的话，自动使用预编译的二进制版本（*.o）
    - 支持相关路径，但是只推荐在私有文件上使用，如：NPL.load(“./A.lua”)或 NPL.load(“../../B.lua”)
    - 当使用相关路径时，文件的扩展将被忽略而*.npl 和*.lua 被搜索，例如：NPL.load(“./A”)或 NPL.load(“../../B”)
- 自动解决递归问题（例如文件 A 加载文件 B，同时文件 B 加载文件 A）
- 编译脚本代码如果还没有编译的话
    - 如果文件扩展是*.npl，NPL meta 编译器将会被激活。
- 代码第一次被编译时，代码块也在保护模式下利用 pcall 被执行。在多数情况下，这意味着添加新代码到全局或输出表中，因此在之后可以使用它。
- 它也可以被用来加载 C++/Mono C#/NPL 文件夹。
- 如果存在的话，返回输出文件模型或 nil（空）。如果模型找不到则返回 false。

代码注入模型：

因为在 NPL/Lua 中，表和函数是第一个类对象，我们在应用的生命周期中使用一个灵活的代码注入模型去管理所有动态的加载代码。

为了注入新的代码，我们使用像 commonlib.gettable,commonlib.inherit 之类的方法，详细请看面向对象的介绍。开发者需要确保使用注入的代码有特别的名字（例如 CompanyName.AppName.ModuleName）。换句话说就是不要污染了全局的表。

要注入新代码，用户可以调用 NPL.load 和 commonlib.gettable 在文件范围内去创建一个局部存根变量。注意，由于是 NPL.load 命令，存根可能在确切代码注入前创建。

例如，像下面一样，你在 script/MyApp/Myclass.lua 中有一个类文件：

```lua
local MyClass = commonlib.inherit(nil,
commonlib.gettable("MyApp.MyClass"));

MyClass.default_param = 1;

-- this is the constructor function.
function MyClass:ctor()
   self.map = {};
end

function MyClass:init(param1)
   self.map.param1 = param1;
   return self;
end

function MyClass:Clone()
   return MyClass:new():init(self:GetParam());
end

function MyClass:GetParam()
   return self.map.param1;
end
```

为了使用以上的类，我们可以使用：

```lua
NPL.load("(gl)script/MyApp/MyClass.lua")
local MyClass = commonlib.gettable("MyApp.MyClass");

local UserClass = commonlib.inherit(nil,
commonlib.gettable("MyApp.UserClass"));

function UserClass:ctor()
   self.map = MyClass:new();  -- use another class
end
```

基于文件的模型：

定义一个基于文件的模型有三种方法。假设你有一个叫做 A.npl 的文件，你想从中输出一些对象。

一种方法是再文件加载时调用 NPL.export。

```
-- this is A.npl file
local A = {};
NPL.export(A);

function A.print(s)
echo(s);
end
```

以下是一种更好的方法，使用了周期性的依赖关系。推荐使用下面这种方法。

```
-- this is A.npl file
local A = NPL.export();
function A.print(s)
    echo(s);
end
```

另一种方法是简单的返回和最近文件的在最后一行代码有关的对象。这也是一种兼容 lua 的方式。当有周期性依赖关系时，这种方法不适用。

```
-- this is A.npl file
local A = {};
return A;
```

最后，有一种先进的手动的方法来简单的添加到_file_mod_表中。

```
-- this is A.npl file
local A = {};
_file_mod_[NPL.filename():gsub("([^/]*$)", "").."A.npl"] = A;
```

正如你看到的，基于文件的模型简单的自动的储存从模型全部文件名到隐藏在全局表中的输出对象的映射。如果文件名或文件地址改变，映射键值也改变。这就是为什么你可以加载一个模型的多个版本并且让用户在已给的源文件中选择使用哪一个。

周期性模型参考：

当你写依赖于每个文件的文件模型时，存在周期性依赖。一个应对方法是更改默认输出对象，它往往是一个空表，而不是创建一个本地表。例如，我们有两个文件 A 和 B 相互引用：

```
-- A.lua
local B = NPL.load("./B.lua")
local A = NPL.export();
function A.print(s)
B.print(s)
end
```

```lua
-- B.lua
local A = NPL.load("./A.lua")
local B = NPL.export();
function B.print(s)
echo(s..type(A));
end
```

（NPL.export()是 NPL 中文件模型系统的核心，它会返回默认的空表来表示当前文件。这和 commonlib.gettable()解决周期性依赖使用的是一样的方法。）

面向对象类的继承模型：

下面的例子介绍了两个表继承及周期性依赖的例子：

```lua
-- base_class.lua
local derived_class = NPL.load("./derived_class.lua")
local base_class = commonlib.inherit(nil, NPL.export());
function base_class:ctor()
   derived_class:cyclic_dependency_test();
end

-- derived_class.lua
local base_class = NPL.load("./base_class.lua")
local B = commonlib.inherit(base_class, NPL.export());
function B:ctor()
end
function B:cyclic_dependency_test()
end
```

使用模型名称加载基于文件的模型：

通过调用 NPL.load(modname)可以导入一个基于文件的模型。NPL.load 是一个通用函数，它不仅可以加载标准源文件还可以加载基于文件的模型并且返回输出的模型对象。（一个模型名（modname）与文件名是不一样的，一个字符串可以当作一个模型名当且仅当它不包含文件扩展或以“./”及”../”开头，如 NPL.load(“sample_mod”)）

NPL.load(modename)将会自动寻找模型的源文件通过按顺序尝试接下来的地址直到找到一个。在 NPL.load 使用模型名称比使用清楚的文件名效率低，因为它在每次调用时都需要搜索，所以请只在文件加载时使用它，例如在文件的开头。

- 如果代码从 *npl_packages/[parentModDir]/*调用 NPL.load
  - npl_packages/[parentModDir]/npl_mod/sample_mod/sample_mod.npl
  - npl_packages/[parentModDir]/npl_packages/sample_mod/npl_mod/sample_mod/sample_mod.npl

- npl_mod/sample_mod/sample_mod.npl
- npl_packages/sample_mod/npl_mod/sample_mod/sample_mod.npl

第二个例子：NPL.load(“sample_mod.xxx.yyy”)对于*.npl 和*.lua 文件将会测试下面的文件地址：

- 如果调用的代码是来自 `npl_packages/[parentModDir]/`
  - `npl_packages/[parentModDir]/npl_mod/sample_mod/xxx/yyy.npl`
  - `npl_package/[parentModDir]/npl_packages/sample_mod/npl_mod/xxx/yyy.npl`
- npl_mod/sample_mod/xxx/yyy.npl
- npl_packages/sample_mod/npl_mod/sample_mod/xxx/yyy.npl

正如你所见的，npl_packages/和 npl_mod 是两个特殊的文件夹，它们是用来搜索什么时间英国加载基础模型的。在搜索全局的模型前，在 npl_packages/xxx/folder 中的代码经常使用本地 npl_mod 和本地 npl_packages，这允许同一个 npl_mod 的多个版本在不同的 npl_package 中同时存在。这取决于开发者如何决定去封装和发布他们的应用或模型。

加载文件夹：

使用以/.结尾的文件名调用 NPL.load 也是可以的。默认加载文件夹只是在很多地址中寻找到那个文件夹并将它加入全局搜索路径。然而，当文件夹包含一个叫做 package.npl 的文件时有例外。

例如，假设 npl_mod/sample_mod/package.npl 是像下面代码一样：

```
-- example of NPL.load folder with package folder configuration file
{
	-- do not add search path when loading the containing folder via NPL.load. default to true.
	searchpath = false,
	-- bootstrapper = "",
	-- main script
	main = "fileA.lua",
}
```

同时 npl_mod/sample_mod/fileA.lua 是这样的代码：

```
local fileA = NPL.export();
function fileA:print()
  echo("fileA")
```

```
end
```

那么我们可以用以下方式加载文件夹：

```
local fileA = NPL.load("npl_mod/sample_mod/");
echo(fileA:print())
```

文件夹不会被添加到搜索路径因为 searchpath=false、并且从 fileA 中会返回输出对象。

加载文件夹是一个 NPL package 的功能。

什么时候使用 require:

require 是加载基于文件的模型的 Lua 方法。NPL 与这种函数挂钩，经常在下达命令前使用相同的输入调用 NPL.load。更特殊的是，NPL 往 lua 的 package.loader 中注入一个自定义加载器。所以调用 require 是几乎与调用 NPL.load 是相同的。到目前为止，多种代码版本无法同时存在。

不要使用 require:

因为 NPL.load 现在与原始的 lua 的需求函数挂钩，实际中这一部分是不确定的，但是我们依然不推荐在你自己的代码中使用 require，除非你在使用第三方 lua 库。理由很简单，原始的 require 在 NPL.load 中是反向兼容的，但是原始的 require 不支持 NPL.load 提供的特点。在你自己的代码中使用 require 会使其他 lua 开发者迷惑。因此 NPL 开发者写的代码通常会使用 NPL.load.

只有在加载 lua 中 C 语言插件时才会使用 require，不要将它是用在你自己的 NPL 脚本中。require 在团体中速率低。它原始的实现时相同的，但是在平面方式使用了相似的全局隐藏表，这是个你从不知道的逻辑。而且它至今都不支持周期性依赖。

此外，一些应用需要创建沙盒环境来隔离代码的执行（代码注入到特定的全局表中），使用 require 的话无法控制代码注入。

不同的 NPL 应用有自己的沙盒环境，其中有它们自己专用的全局表，例如在 NPL 网络服务器 APP 的所有*.page 文件中每一个 URL 请求使用一个分离的全局表。

最后，相比于 NPL，require 运行慢且使用不同的搜索路径。同时 NPL.load 快且在 zip 文件和在 npl_packages 定义的搜索路径中都接受预编译，并且在所有平台可以保持一致。

IdreamTech

3.4 Activation File

NPL 在使用 NPL.activate 函数时可以和 NPL 脚本远程对话。这是一个强大的函数且在 C++和单插件中都可以使用。

基础语法：

像在神经网络中一样，所有交流都是异步且单向的，没有回调。尽管函数返回一个整数值。

```
NPL.activate(url, {any_pure_data_table_here, })
```

- @参数 url：一种 NPL 文件名实例的一个全局的独一无二的名字。一个 NPL 文件名的字符串形式如下：[(sRuntimeStateName|gl)][sNID:]sRelativePath[]下面的列表是所有的有效的文件名的结合：
  - user001@paraengine.com:script/hello.lua --这是一个在默认游戏线程 user001 中的文件
  - (world1)server001@paraengine.com:script/hello.lua -- 这 是 server001 中在线程 world1 中的一个文件
  - (worker1)script/hello.lua --这是在 worker1 线程中的一个本地文件
  - (gl)script/hello.lua --这是在当前 runtime state 的线程中的一个本地文件
  - script/hello.lua --这是当前线程中的一个文件。对于一个单线程应用来说这往往够用了。
  - (worker1)NPLRouter.dll --激活一个 C++文件。注意，在 Windows 中，这是寻找 NPLRonter.dll.在 Linux 中这是寻找./libNPLRouter.so
  - (worker1)DBServer.dll/DBServer.DBServer.cs --对于 C#文件，类必须在 CS 文件中定义有一个静态的激活函数
- @参数 msg：这是一个纯数据表块，将被传递至目标文件。

Neuron File

只有与激活函数关联的文件可以被激活。但在 NPL/C++/C#插件中是不同的：

- 在 NPL 中，内置的 NPL.this 函数可以被更灵活的使用。

```
local function activate()
```

```
  -- input is passed in a global "msg" variable
  echo(msg);
end
NPL.this(activate);
```

- msg 被传入所有文件都可见的全局的 msg 变量，并且 msg 变量将会持续存在知道线程接收到下一个激活消息
- 在 NPL 的 C++插件中，你需要定义一个 C 函数。请在 ParacraftSDK 的样例文件中查看详细
- 在 NPL 的单一 C#插件中，用静态的 activate 函数简单地定义一个类。

输入消息，msg.nid 和 msg.tid

在上述的代码中，msg 包含从发送方接受的消息加上发送者的 source id。对于未经身份验证的发送方，source id 会存在 msg.tid 中，msg.tid 是一个自动生成的数字字符串如“~1”。接收方可以一直使用这个暂时的 id：msg.tid 返回消息，例如：

```
local function activate()
  -- input is passed in a global "msg" variable
  NPL.activate(format("%s:some_reply_file.lua", msg.tid or msg.nid),
{"replied"});
end
NPL.this(activate);
```

接受方也可以通过调用 NPL.accept(msg.tid, nid_name)来重命名这个暂时的 msg.id，所以如果下次接收方从同一个发送方获得了一个消息（例如相同的 TCP 连接），msg.nid 会包含最后分配的名称同时 msg.tid 不再存在。我们通常使用 NPL.accept 去区分经过身份验证和未经身份验证的发送方，并且尽早的通过调用 NPL.reject(msg.tid)来拒绝未经身份验证的消息来节省 CPU 的循环。例如：

```
local function activate()
  -- input is passed in a global "msg" variable
  if(msg.tid) then
    -- unauthenticated? reject as early as possible or accept it.
    if(msg.password=="123") then
      NPL.accept(msg.tid, msg.username or "default_user");
    else
      NPL.reject(msg.tid);
    end
  elseif(msg.nid) then
```

IdreamTech

```
      -- only respond to authenticated messages.
      NPL.activate(format("%s:some_reply_file.lua", msg.nid), {"replied"});
   end
end
NPL.this(activate);
```

注意，msg.tid 或 msg.nid 经常连接到一个单独的低层的 TCP 连接，因此它们的名字被分享到所有的进程中的神经元文件。例如：你在一个神经元文件中接收了，其他所有神经元文件接收到的形式都为 msg.nid。

Neuron File 可见性:

因为安全性的原因，所有神经元文件可以在同一进程中被其他文件激活。这包括在同一进程的其他线程中的脚本。

为了将脚本暴露给远程电脑，有两件事需要做：

- 第一个是通过监听一个 IP 地址和端口来开启 NPL 服务端，NPL 对所有的对话都使用 TCP 协议
- 第二个是告诉 NPL runtime 给出的文件是公共神经元文件

例如：

```
    NPL.StartNetServer("0.0.0.0", 8080);
    NPL.AddPublicFile(filename, id);
```

其中“0.0.0.0”代表所有的 IP 地址，也可以使用“127.0.0.1”，“localhost”或其他任何 IP 地址；“8080”是端口数字。选择你想用的就可以。

NPL.AddPublicFile 第二个参数是整数，它是代表长文件名保存带宽而被传输的。所以如果你添加多个公共文件的话它必须是独一无二的。

注意，文件名必须和工作目录相对应，例如：NPL.AddPublicFile(“script/test/test.lua”, 1)。绝对路径在当时是不支持的。

激活远程文件:

当一个 NPL runtime 服务端暴露一个公共文件，其他客户端的 NPL runtime 可以使用 NPL.activate 函数来激活它。注意，一个 NPL runtime 既可以是服务端也可以是客户端。发起连接的一般称它为客户端。为了激活服务端，单纯的客户端也叫做 NPL.StartNetServer。但是它可以具体指明 port=”0”来表明它不会监听进来的连接。

IdreamTech

然而，在客户端，我们需要使用 NPL.AddNPLRuntime 去分配一个本地名称到远程的服务端，所以我们可以在之后所有的 NPL.activate 调用中通过名称查到这个服务端。

```
NPL.AddNPLRuntimeAddress({host = "127.0.0.1", port = "8099", nid =
"server1"})
```

我们通常在初始化的时候进行一次这种操作。在那之后我们可以在远程服务端上激活公共文件，如：

```
NPL.activate("server1:helloworld.lua", {})
```

注意，nid 具体指定的名字是随机的并且仅仅用在客户端电脑去查找另一个电脑。换句话说，不同客户端可以用不同的名字命名同一台远程电脑。

信息传递的保障：

请注意，一个电脑第一次激活远程文件时，一个 TCP 连接会自动建立，但是第一个消息未被发出。这是因为 NPL.activate()时异步的，它必须在建立连接前返回一个值。通常当消息通过一个已存在的路径发送时返回 0，并且要是第一条消息被发送到了远程系统的话返回非 0

如果没有已经建立的路径（例如没有 TCP 连接），NPL 将会立马尝试建立连接。然而，需要注意的是，消息返回非 0 是没有被传递的，尽管 NPL 立刻在之后成功的建立了一个路径到远程系统。因此，重新再激活直到 NPL.activate 返回 0 是程序员的工作.这保证了一个返回值为 0 的消息至少再在 NPL runtime 的角度发送出去了。

同样的机制可以被用来恢复断开的连接。

要写一个可容忍错误的消息传递代码，请考虑以下方法：

- 当一个 NPL runtime 进程开启，使用 NPL.activate ping 远程的进程直到返回 0 来建立 TCP 连接。这保证了跨这两个系统的所有之后的 NPL.activate 可以被投递。
- 发现断开的连接：
  - 方法一：使用一个计时器去 ping 或监听断开连接的系统事件并重新连接以防连接丢失。但是，单独的消息在期间可能会丢失。
  - 方法二：使用一个封装函数去调用 NPL.activate，函数会检查返回值。如果

IdreamTech

它非 0，无论重新连接超时还是将消息放到一个等待的队列以防连接被立即覆盖和重新发送队列消息。

当 NPL.activate 返回非 0 值时，我们将它交给程序员去解决所有的情况，因为不同的事物逻辑可能会使用不同的方法。

服务端/客户端 app 例子：

要运行示例，调用下列命令：

```
npl "script/test/network/SimpleClientServer.lua" server="true"
npl "script/test/network/SimpleClientServer.lua" client="true"
```

这个示例的源代码同时也在 ParaCraftSDK/examples 文件夹。

文件名：script/test/network/SimpleClientServer.lua

```
--[[
Author: Li,Xizhi
Date: 2009-6-29
Desc: start one server, and at least one client.
-----------------------------------------------
npl "script/test/network/SimpleClientServer.lua" server="true"
npl "script/test/network/SimpleClientServer.lua" client="true"
-----------------------------------------------
]]
NPL.load("(gl)script/ide/commonlib.lua"); -- many sub dependency included

local nServerThreadCount = 2;
local initialized;
local isServerInstance =
ParaEngine.GetAppCommandLineByParam("server","false") == "true";

-- expose these files. client/server usually share the same public files
local function AddPublicFiles()
    NPL.AddPublicFile("script/test/network/SimpleClientServer.lua", 1);
end

-- NPL simple server
local function InitServer()
    AddPublicFiles();

    NPL.StartNetServer("127.0.0.1", "60001");

        for i=1, nServerThreadCount do
                local rts_name = "worker"..i;
```

```
                local worker = NPL.CreateRuntimeState(rts_name, 0);
                worker:Start();
        end

    LOG.std(nil, "info", "Server", "server is started with %d threads",
nServerThreadCount);
end

-- NPL simple client
local function InitClient()
    AddPublicFiles();

    -- since this is a pure client, no need to listen to any port.
        NPL.StartNetServer("0", "0");

        -- add the server address
        NPL.AddNPLRuntimeAddress({host="127.0.0.1", port="60001",
nid="simpleserver"})

    LOG.std(nil, "info", "Client", "started");

        -- activate a remote neuron file on each thread on the server
        for i=1, nServerThreadCount do
                local rts_name = "worker"..i;

        while( NPL.activate(string.format("(%s)simpleserver:script/test/net
work/SimpleClientServer.lua", rts_name),
            {TestCase = "TP", data="from client"}) ~=0 ) do
            -- if can not send message, try again.
            echo("failed to send message");
            ParaEngine.Sleep(1);
       end
        end
end

local function activate()
    if(not initialized) then
       initialized = true;
       if(isServerInstance) then
          InitServer();
       else
          InitClient();
       end
        elseif(msg and msg.TestCase) then
```

```
        LOG.std(nil, "info", "test", "%s got a message", isServerInstance and "server" or "client");
                echo(msg);
        end
end
NPL.this(activate);
```

上面的服务端是多线程的。

开始时，NPL.activate 调用一个新的远程服务端（这个服务端还未建立 TCP 连接），消息被丢弃同时返回一个非 0 的值。NPLExtension.lua 包含大量的帮助函数去帮助你发送一个保障消息，例如：NPL.activate_with_timeout。你需要 include commonlib 来使用它。

信任的连接和 NID：

接收者的激活函数可以分配任何名称或 nid 来接入连接的 NPL runtime。

HelloWorld 示例：

现在这里有一个更复杂的 helloworld。它通过把一个激活函数与其关联将一个普通的 helloworld.lua 转变为一个神经元文件。文件接下来就可以被任何 NPL 线程或远程电脑通过它们的 NPL 地址（url）调用了。

```
local function activate()
   if(msg) then
      print(msg.data or "");
   end
   NPL.activate("(gl)helloworld.lua", {data="hello world!"})
end
NPL.this(activate);
```

简单的网络服务端示例：

NPL 使用了一个兼容 HTTP 的协议，所以它可以使用同一个 NPL 服务端去处理标准 HTTP 请求。当 NPL runtime 接收到一个 HTTP 请求消息，它将会通过 id -10 把消息发送到一个公共的可见文件。所以我们可以仅仅使用几行代码来创建一个简单的 HTTP 网站服务端，例如：

文件名：main.lua

```
NPL.load("(gl)script/ide/commonlib.lua");

local function StartWebServer()
        local host = "127.0.0.1";
```

TumTech

```
	local port = "8099";
	-- tell NPL runtime to route all HTTP message to the public neuron
file `http_server.lua`
	NPL.AddPublicFile("source/SimpleWebServer/http_server.lua", -10);
	NPL.StartNetServer(host, port);
	LOG.std(nil, "system", "WebServer", "NPL Server started on
ip:port %s %s", host, port);
end
StartWebServer();

local function activate()
end
NPL.this(activate);
```

文件名：http_server.lua

```
NPL.load("(gl)script/ide/Json.lua");
NPL.load("(gl)script/ide/LuaXML.lua");

local tostring = tostring;
local type = type;

local npl_http = commonlib.gettable("MyCompany.Samples.npl_http");

-- whether to dump all incoming stream;
npl_http.dump_stream = false;

-- keep statistics
local stats = {
	request_received = 0,
}

local default_msg = "HTTP/1.1 200 OK\r\nContent-Length: 31\r\nContent-Type:
text/html\r\n\r\n<html><body>hello</body></html>";

local status_strings = {
	ok ="HTTP/1.1 200 OK\r\n",
	created ="HTTP/1.1 201 Created\r\n",
	accepted ="HTTP/1.1 202 Accepted\r\n",
	no_content = "HTTP/1.1 204 No Content\r\n",
	multiple_choices = "HTTP/1.1 300 Multiple Choices\r\n",
	moved_permanently = "HTTP/1.1 301 Moved Permanently\r\n",
	moved_temporarily = "HTTP/1.1 302 Moved Temporarily\r\n",
	not_modified = "HTTP/1.1 304 Not Modified\r\n",
	bad_request = "HTTP/1.1 400 Bad Request\r\n",
	unauthorized = "HTTP/1.1 401 Unauthorized\r\n",
```

```
	forbidden = "HTTP/1.1 403 Forbidden\r\n",
	not_found = "HTTP/1.1 404 Not Found\r\n",
	internal_server_error = "HTTP/1.1 500 Internal Server Error\r\n",
	not_implemented = "HTTP/1.1 501 Not Implemented\r\n",
	bad_gateway = "HTTP/1.1 502 Bad Gateway\r\n",
	service_unavailable = "HTTP/1.1 503 Service Unavailable\r\n",
};
npl_http.status_strings = status_strings;

-- make an HTML response
-- @param return_code: nil if default to "ok"(200)
function npl_http.make_html_response(nid, html, return_code, headers)
	if(type(html) == "table") then
		html = commonlib.Lua2XmlString(html);
	end
	npl_http.make_response(nid, html, return_code, headers);
end

-- make a json response
-- @param return_code: nil if default to "ok"(200)
function npl_http.make_json_response(nid, json, return_code, headers)
	if(type(html) == "table") then
		json = commonlib.Json.Encode(json)
	end
	npl_http.make_response(nid, json, return_code, headers);
end

-- make a string response
-- @param return_code: nil if default to "ok"(200)
-- @param body: must be string
-- @return true if send.
function npl_http.make_response(nid, body, return_code, headers)
	if(type(body) == "string" and nid) then
		local out = {};
		out[#out+1] = status_strings[return_code or "ok"] or 
return_code["not_found"];
		if(body~="") then
			out[#out+1] = format("Content-Length: %d\r\n", 
#body);
		end
		if(headers) then
			local name, value;
			for name, value in pairs(headers) do
				if(name ~= "Content-Length") then
```

```
                                        out[#out+1] =
format("%s: %s\r\n", name, value);
                                end
                        end
                end
                out[#out+1] = "\r\n";
                out[#out+1] = body;

                -- if file name is "http",  the message body is raw http
stream
                return NPL.activate(format("%s:http", nid),
table.concat(out));
        end
end


local function activate()
        stats.request_received = stats.request_received + 1;
        local msg=msg;
        local nid = msg.tid or msg.nid;
        if(npl_http.dump_stream) then
                log("HTTP:"); echo(msg);
        end
        npl_http.make_response(nid, format("<html><body>hello world.
req: %d. input is %s</body></html>", stats.request_received,
commonlib.serialize_compact(msg)));
end
NPL.this(activate)
```

3.5 多线程

在 NPL 中，每个线程都是直接被创建且直接被命名的。NPL 线程在一个 NPL 进程的生命周期中是持续的且是重复使用的。

在 NPL 中，多线程是与网络对话一样被处理的。换句话说，激活的脚本在其他本地线程中是与运行在其他电脑上的调用脚本一样的，都是虚拟的。对于本地线程和远程电脑你都可以简单的利用 NPL.activate 与远程脚本文件对话。唯一的区别是目标脚本的 url 不同。

例如，要在本地线程 A 中激活一个脚本，我们可以使用：

```
        NPL.activate("(A)helloworld.lua", {});
```

要激活一个远程电脑 B 的线程 C 中的脚本，我们可以使用：

```
        NPL.activate("(C)B:helloworld.lua", {});
```

更多详细的内容可以参考 1.3.4 Activation File.

创建 NPL 工作线程

在可以被运行在线程中的脚本处理的消息到达前，NPL 工作线程必须被创建。

`NPL.active("(A)helloworld.lua", {});`不会自动创建一个线程 A。你需要使用下面的代码在线程 A 中创建 NPL runtime：

```
NPL.CreateRuntimeState("A", 0):Start();
```

在这之后，被发送到`(A)helloworld.lua` 的消息将被在一个真实的叫做 A 的线程中被处理。

Project 例子：

我们现在仅仅使用单个文件`script/test/TestMultithread.lua`来创建一个多线程应用。应用会在 5 个线程中同时打印 helloworld。

```
NPL.load("(gl)script/ide/commonlib.lua");

local function Start()
   for i=1, 5 do
      local thead_name = "T"..i;
     NPL.CreateRuntimeState(thead_name, 0):Start();
     NPL.activate(format("(%s)script/test/TestMultithread.lua", 
thead_name), {
            text = "hello world",
            sleep_time = math.random()*5,
      });
   end
end

local isStarted;
local function activate()
   if(msg and msg.text) then
      -- sleep random seconds to simulate heavy task
      ParaEngine.Sleep(msg.sleep_time);
      LOG.std(nil, "info", "MultiThread", "%s from thread %s", 
msg.text, __rts__:GetName());
   elseif(not isStarted) then
      -- initialize on first call
      isStarted = true;
      Start();
   end
```

Tech

```
    end

    NPL.this(activate);
```

运行上述文件，我们需要使用命令：

```
    NPL.activate("(gl)script/test/TestMultithread.lua");
```

输出则会是：

```
    2016-03-16 6:22:00 PM|T1|info|MultiThread|hello world from thread
T1
    2016-03-16 6:22:01 PM|T3|info|MultiThread|hello world from thread
T3
    2016-03-16 6:22:03 PM|T2|info|MultiThread|hello world from thread
T2
    2016-03-16 6:22:03 PM|T5|info|MultiThread|hello world from thread
T5
    2016-03-16 6:22:04 PM|T4|info|MultiThread|hello world from thread
T4
```

进阶的例子

下面的 NPL 模型利用了多种本地线程：

- `script/ide/System/Database/TableDatabase.lua`：这是一个通过 IO 线程按线路发送所有请求并分散加载一系列工作线程的数据库服务端。
- `script/apps/DBServer/DBServer.lua`：这是一个

3.6 Concurrency Model

NPL 的并发模型和设计原则是一个深入的话题。它经常被拿来和其它类似的用流行的语言写的模型进行比较，如 erlang，GO，Scala（java）等。

什么是并行

所有电脑语言的图画是按顺序被处理的。并行是一种关于编写代码使其并行运行的语言特性。实现这种方式传统的方法是通过线程和锁，但却在编写时很麻烦。角色模型提供了一种不同的方法实现并行，它是在 1973 年被 Carl Hewitt 提出的，可以避免线程和锁带来的问题。

有很多种语言可以实现并行模型，区别在于在不同用例中的性能以及写并行模型时程序员的所想的图画。

Idreamle

NPL 使用混合的方法是你可以在单个线程中或通过多个线程运行成百上千的任务。更重要的是，你不需要写任何的代码来创建一个虚拟进程或写易出错的消息循环。总之，NPL 运行十分迅速，可宽展并且使程序员脑海中有一个与神经元类似图画。

NPL 中的并行激活

优先的文件激活 vs 非优先的文件激活：

NPL 默认的激活函数是非优先的，程序员需要在合理的时间片段完成执行。默认的非优先模型提供给程序员关于代码如何执行的全部控制并且不同的神经元文件可以在同一个线程中使用全局表来简单的分享数据。

另一方面，NPL 也允许你做优先的激活，在激活时 NPL runtime 将会记下虚拟机命令的数量直到达到一个用户定义的值（例如 1000），然后自动停止激活函数。函数将会在下一个时间片段自动继续运行。NPL 时间片段默认值是 16ms（即 60FPS）。

为了使激活函数变为优先的，只需简单的传递一个二级参数`{PreemptiveCount, MsgQueueSize, [filename|name], clear}`到`NPL.this`如：

- PreemptiveCount：这是停止前 VM 命令的数量。如果为 nil 或者 0，它是非优先的。请注意实时编译的代码不是按照默认值计数的。
- MsgQueueSize：这是这个文件的最大的消息队列的大小，如果没有具体说明，他和 NPL 线程消息队列是一样大的。
- filename|name：虚拟文件名，如果没有具体说明，目前正在加载的文件将会被使用。
- clear：清除所有被文件使用的内存，包括消息队列。正常情况下时不需要清除的。

一个没有消息的神经元文件使用少于 100bytes 的内存（多数取决于文件名的长度）

```
-- this is a demo of how to use preemptive activate function.
NPL.this(function()
 local msg = msg; -- important to keep a copy on stack since we go preemptive.
 local i=0;
 while(true) do
          i = i + 1;
          echo(tostring(msg.name)..i);
```

IdreamTech

```
		if(i==400) then
			error("test runtime error");
		end
	end
end, {PreemptiveCount = 100, MsgQueueSize=10, 
filename="yourfilename.lua"});
```

你可以使用以下命令进行测试：

```
NPL.activate("yourfilename.lua", {name="hi"});
```

优先的激活文件的一些事实：

- 它允许你在当前同一个线程中运行成百上千的任务。每一个运行的任务有它自己的栈并且内存花销大约 450bytes。一个没有待处理消息的神经元文件使用少于100bytes 的内存（多数情况取决于文件名的长度）。对当前任务仅有的现在是你的系统内存。
- 对 VM 指令进行计数会对程序运行速率的性能有微小的损耗。
- 有优先激活函数情况下，程序员在线程中进行改动来分享数据时要格外注意，因为在任何指令下函数都有可能停止。最好的方法是绝不为了分享数据而作任何改变，但可以使用消息来改变数据。
- C/C++的 API 调用是算作一条指令的，所以如果你调用 ParaEngine.Sleep(10),它会在那个线程中阻塞当前所有的任务 10 秒钟。
- 在激活函数中异步调用的代码（如计时器，远程 API 调用）是**非**优先的。因为回调的内容是从当前激活函数的环境下激活的。

测试用例及例子：

更多的测试用例在 script/ide/System/test/test_concurrency.lua 中。

```
local test_concurrency = 
commonlib.gettable("System.Core.Test.test_concurrency");

function test_concurrency:testRuntimeError()
	NPL.this(function()
		local msg = msg; -- important to keep a copy on stack 
since we go preemptive.
		local i=0;
		while(true) do
			i = i + 1;
```

IdreamTech

```
                    echo(tostring(msg.name)..i);
                    if(i==40) then
                            error("test runtime error");
                    end
            end
    end, {PreemptiveCount = 100, MsgQueueSize=10,
filename="tests/testRuntimeError"});
    NPL.activate("tests/testRuntimeError", {name="1"});
    NPL.activate("tests/testRuntimeError", {name="1000"});
end

function test_concurrency:testLongTask()
    NPL.this(function()
            local msg = msg; -- important to keep a copy on stack
since we go preemptive.
            local i=0;
            while(true) do
                    i = i + 1;
                    echo(i);
            end
    end, {PreemptiveCount = 100, MsgQueueSize=10,
filename="tests/testLongTask"});
    NPL.activate("tests/testLongTask", {name="1"});
end

function test_concurrency:testMessageQueueFull()
    NPL.this(function()
            local msg = msg; -- important to keep a copy on stack
since we go preemptive.
            local i=0;
            for i=1, 1000 do
                    i = i + 1;
            end
            echo({"finished", msg});
    end, {PreemptiveCount = 100, MsgQueueSize=3,
filename="tests/testMessageQueueFull"});
    for i=1, 10 do
            NPL.activate("tests/testMessageQueueFull", {index=i});
    end
    -- result: only the first three calls will finish.
end

function test_concurrency:testMemorySize()
    __rts__:SetMsgQueueSize(100000);
```

```
    for i=1, 10000 do
            NPL.this(function()
                    local msg = msg; -- important to keep a copy on
stack since we go preemptive.
                    for i=1, math.random(1,1000) do
                            msg.i = i;
                    end
                    echo(msg);
            end, {PreemptiveCount = 10, MsgQueueSize=1000,
filename="tests/testMemorySize"..i});
            NPL.activate("tests/testMemorySize"..i, {index=i});
    end
    -- TODO: use a timer to check when it will finish.
end

function test_concurrency:testThroughput()
    __rts__:SetMsgQueueSize(100000);
    for i=1, 10000 do
            NPL.this(function()
                    local msg = msg;
                    while(true) do
                            echo(msg)
                    end
            end, {PreemptiveCount = 10, MsgQueueSize=3,
filename="tests/testThroughput"..i});
            NPL.activate("tests/testThroughput"..i, {index=i});
    end
end
```

NPL 消息调度：

每个 NPL runtime 可以拥有一个或多个 NPL state/线程（如实际系统的线程）。每个 NPL 状态对于其他 NPL 线程或远程进程的输入和输出有一个单独的消息队列。程序员可以设置这个队列的最大值，所以当它满了的时候，消息会被自动的丢弃。

在每一个时间片段中，NPL state 会按优先顺序处理所有在消息队列中的消息。

- 如果一个消息属于一个非优先文件，它会立即调用它的激活函数。
- 如果一个消息属于一个优先文件，它会删除来自队列的消息并且将其插入目标文件的消息队列，目标文件的消息队列的大小是不同的。

在另一个本地线程计时器中，所有的激活的优先文件（包含带处理的/处理一半的消息）

将会按优先的方式被处理/恢复。例如：我们尽可能会对于每个激活函数和中止的 VM（virtual machine）指令进行计数。

注意：所有的 TCP/IP 网络连接都是被一个全局的网络 IO 线程管理的。通过这种方法，你可以不使用很多系统的内存就有很多活跃的 TCP 连接。一个全局的 NPL 时间调度器可以自动的调度输入的网络消息到目标 NPL 状态/线程的消息队列中。通常情况下，NPL 线程的数量和电脑上 CPU 的核的数量相近。

优先的编程注意事项：

因为优先的代码和非优先的代码可以在同一个 NPL state（线程）共存。程序员需要确保优先代码不会改变全局的数据。

- 注意：如果附带 NPL 调试器的话，所有的优先的代码都会停止使得可以调试代码。
- 同时，请注意实时编译的代码不默认调用 hook。实时编译代码和编辑 src/Makefile: `XCFLAGS= -DLUAJIT_ENABLE_CHECKHOOK` 都是不支持的。这就会造成速率的损失，尽管 hook 没有被设定。我们的建议是调用 dummy（）或其他 NPL API 函数来计数。

并行激活的优先级：

PreemptiveCount 是在一个激活函数停止前其中被处理的所有指令（或一定字节的代码）的数量。在 NPL 中，PreemptiveCount 可以被每个激活函数详细说明。因此，你可以更好的控制在一个已给的时间片段内每个激活函数可以运行多少计算。

对于优先激活文件，NPL 调度会保证每个函数在每个时间片段的 PreemptiveCount*total_active_process_count 命令之后精确的运行 PreemptiveCount 命令。

所以在我们的用户模型 preemptive 调度中绝对没有死锁。唯一会导致应用出问题的因素是将内存用完了。然而，每个活跃的（运行的）函数仅仅根据使用情况占用 400-2000bytes，并且 1 百万并行运行的任务仅仅占用 1GB 内存。如果平均仅有一半的任务处于运行状态，你只会使用 500MB 多一点的内存。

3.7 Common Library

IdreamTech

NPL 常用的库含有大量的、开源的库包含：IO，网络，2d/3d 图像，网站服务端，数据结构以及其他的软件框架。

库的下载：

NPL 常用库可以按 pkg 或者 zip 或者清楚的源代码的形式被下载。当你配置你的应用时，你可以选择使用我们已制作好的 zip 文件或者你自己在一个 zip 包内配置你需要使用的文件。如果你在 Windows 平台通过 ParaCraftSDK 下载 NPL runtime，那么 NPL 常用库已经被下载到 `ParaCraftSDKGit/NPLRuntime/win/packages/main.pkg`

如果你在 Linux 上下载 NPL runtime 的源代码，NPL 常用库时没有预装的。你需要拷贝或者连接 script 文件夹到你的项目工作目录。

主要的库：

对于最小的服务端应用开发，使用：

```
NPL.load("(gl)script/ide/commonlib.lua");
```

或

```
NPL.load("(gl)script/ide/System/System.lua");
```

对于成熟的重量级 2d/3d 应用开发，使用：

```
NPL.load("(gl)script/ide/IDE.lua");
```

或

```
NPL.load("(gl)script/kids/ParaWorldCore.lua");
```

文档：

NPL 库是写在纯 NPL 脚本中的，它们通常没有外部依赖，但只是在 NPL runtime 提供的低级的 NPL API。这些 API 依次跨平台地在 C/C++中实现。

低级 API 的文档在：https://codedocs.xyz/LiXizhi/NPLRuntime/modules.html

然而，阅读低级的 API 不是很有用。你只需要快速浏览它们然后专注于源代码中的 NPL 库。

3.8 NPL Packages

NPL Packages

NPL package 是 `npl_packages/[package_name]/`下一个特殊的文件夹。其中的文件

常常像与工作目录联系一样被整理。所以这个文件夹可以被直接用作搜索路径，或者压缩其到*.zip|pkg 中被用作档案文件，或者同时被所有的文件模块名称加载。

NPL package 提供以下的目标：

- 它提供一种方法去释放被别人使用的多种软件模型
- 它可以被第三方插件用作额外的搜索路径，在下面会谈到
- 它提供一种方法使得同一种软件模型的不同版本可以共存，方法是将它们放在一个单独应用的不同 npl package 中
- 它提供了一种在运行时间可以下载第三方插件的方法
- 它提供了一种在模型间分享大型资源文件（如 texture， 3d 模型， 音频）的方法，因为所有的 npl_package 共享同一个工作目录

作为搜索路径的 package

在 npl_packages/[package_name]中的文件通常连接到根工作目录。

因此，要使用别人模型的开发者可以简单的通过使用以下代码将 npl_packages/[package_name]添加到全局搜索路径：

```
NPL.load("npl_packages/some_test_module/")
```

这种方法的文件名要以/结尾。这种方法是在 package 文件夹中寻找大量的可能的地址，然后将它们添加到全局的搜索路径。如果相关的路径（如../../test/）已经被使用，那么就将它添加到 dev 或者工作目录而不是添加到包含的脚本文件中。默认情况下，当 NPL 开启时，它就会尝试去加载官方 npl_packages/main/的 package。所以你不需要为了使用丰富的 NPL 库而调用 NPL.load("npl_packages/main/")

其他的方法是通过命令行加载下列的命令：

```
npl loadpackage="npl_packages/paracraft/" dev="/"
```

NPL 如何定位 packages

NPL 按照下列顺序定位给出相关路径的 package 文件夹：

- 如果指定了的话在 dev 文件夹中搜索
- 在当前工作目录搜索
- 在当前可执行目录中搜索

IdreamTech

- 在可执行目录的五个上级目录中递归的搜索
- 在 `foldername.zip|pkg` 中搜索

例如：

- 你没有使用 dev 文件夹
- 你当前的工作目录是`/home/myapp/`
- 你的可执行目录是`/opt/NPLRuntime/redist/bin64/`

然后 `NPL.load("npl_packages/main/")`将会搜索以下目录直到找到一个并将其添加到搜索路径。

- `/home/myapp/npl_packages/main/`
- `/opt/NPLRuntime/redist/bin64/npl_packages/main/`
- `/opt/NPLRuntime/redist/npl_packages/main/`
- `/opt/NPLRuntime/npl_packages/main/`
- `/opt/npl_packages/main/`
- `/npl_packages/main/`
- `npl_packages/main.pkg or npl_packages/main.zip`
    - zip 文件需要包含 `package.npl` 或`[anyfoldername]/package.npl`，文件包含 `package.npl` 被用作 zip 文件的根目录
    - 如果 `package.npl` 包含 `searchpath = false` 或者什么都不包含，zip 归档文件中的文件被认为是与根相关的
    - 如果 package.npl 包含 `searchpath = false`，zip 归档文件中的文件被认为是与加载文件夹相关的。例如：`npl_package/main/`

注意： "npl_packages/"文件夹的前缀不是强制的，它可以是任何的文件夹名，并且上述的地址也被搜索过。

文件系统的分层

一个 package 搜索路径可以被看成一个分层的文件系统。所有的 NPL 代码的引用文件

都与根目录路径相连。然而根目录包含多层挂载点（搜索路径），它会使用同样的根文件路径从顶层一直到底层去寻找文件。

在根目录外加载文件

正如我们在文件系统分层中看到的一样，所有 NPL 的脚本都被定义地与虚拟根目录相关。根目录的顶层是应用的启动/工作目录。如果要在工作目录外强制地加载这个目录，可以使用绝对的目录路径。

- 在 Windows 系统下，你需要像 `NPL.load("C:\\temp/my_mod/abc.npl")`一样的命令，请注意，Windows 系统下在：后要使用\因为：是预留给 nid 的。其他情况下，/和\是相同的并且推荐使用/
- 在 Linux 系统下你可以使用类似于 `NPL.load("/opt/my_mod/abc.npl")`的命令

在加载文件时不推荐使用绝对文件路径，因为它损坏了 dev 文件夹和 package 的搜索路径。使用它唯一的情况可能是要加载一些测试文件或暂时的插件代码时。

在根目录外加载文件夹

在使用 NPL.load()加载文件夹时可以使用绝对文件路径进行关联。如果使用了相关路径（例如`../../test/`），它会经常关联到 dev 或工作目录而不是包含脚本文件。

```
NPL.load("../../anotherApp/test/");
```

使用相关路径可以使得代码跨平台。但在 Windows 下也可以使用绝对路径如 `NPL.load("C:\\temp/anotherApp/test/")`或在 Linux 下使用 `NPL.load("/opt/test/")`

两种方式都不推荐在根目录外加载文件夹。我们推荐使用不带前缀`../`的标准 package 搜索路径

在加载一个 Package 后会发生什么

简短来说，什么都不发生，因为加载一个 package 只是将它的文件夹添加到全局搜索路径中。你依然需要在 package 文件夹中用 `NPL.load` 加载一些模型文件。例如，假设你已经在'/home/myapp/npl_packages/test/'中成功加载了 `NPL.load("npl_packages/test/")` 而且在 `/home/myapp/npl_packages/test/script/any_module_folder/test.lua` 中有模型文件。

接下来你可以用相关文件路径加载它。NPL.load("script/any_module_folder/test.lua")

然而，如果在你的工作目录中有一个文件，如/home/myapp/script/any_module_folder/test.lua，那这个文件将会被加载，而不是全局搜索路径中的文件。

使用 package.npl 加载文件夹

默认情况下加载文件夹只是会在大量的地址中寻找那个文件夹并将其添加到全局搜索路径。然而，有一个情况除外，那就是当一个文件夹包含一个叫做 package.npl 的文件。

例如，我们假设，npl_mod/sample_mod/package.np 是像下面的代码一样：

```
-- example of NPL.load folder with package folder configuration file
{
     -- do not add search path when loading the containing folder via NPL.load. default to true.
     searchpath = false,
     -- bootstrapper = "",
     -- main script
     main = "fileA.lua",
}
```

并且 npl_mod/sample_mod/fileA.lua 的代码类似于：

```
local fileA = NPL.export();
function fileA:print()
   echo("fileA")
end
```

然后我们可以利用下面的代码来加载文件夹：

```
local fileA = NPL.load("npl_mod/sample_mod/");
echo(fileA:print())
```

文件夹不会被添加到搜索路径因为 searchpath = fals 并且它会从 fileA 返回输出对象。

Packages.npl 配置

Package.npl 文件支持以下参数，所有参数都是可选择的：

- searchpath: boolean：在通过默认值为 true 的 NPL.load 加载包含的文件夹时是否添加搜索路径
- bootstrapper: string：加载时使用了 bootstrapper 文件。多数情况下，只有应

用文件包含它

- main: string: 文件夹被加载后需要加载的主脚本文件。这是 searchpath 为 false 时的相关路径，也是 searchpath 为 true 或 nil 时的绝对路径

文件搜索顺序

在当前工作目录的文件常常在解析到全局搜索路径前先搜索（包含压缩的归档文件）。除此之外，最后添加的搜索路径实际上是最先被搜索的。然而有一个例外，如果 NPL 在命令行随着 dev 目录运行 `dev="dev_folder"`，那么在 dev_folder 中的 packages 会在压缩的归档文件前加载。这使得我们可以在开发时使用最近的资源。

例如：

```
NPL.load("npl_packages/A/")
NPL.load("npl_packages/B/")
NPL.load("test.lua");
```

test.lua 最初是在当前工作目录和任何加载的归档文件（zip，pkg）中被搜索的。如果不存在，它会先在 npl_package/B/搜索然后在 npl_packages/A/中搜索

用法

对于 package 的开发者：

NPL packages 多数被开发人员用在开发时

例如：如果你想让其他开发人员使用你的代码，你可以将你的工作目录上传至 git。然后其他开发者可以将你的项目 clone 到他们的 `npl_packages/[your module name]`。Package 中可以不止包含源代码，也可以包含二进制的资源文件，如图片，声音，3D 模型等。

对于其他开发者：

其他开发人员在发行软件前将所有 npl_packages 合并到工作目录

例如：解析依赖性是开发者的工作，以防同一文件的多个版本存在于使用的 npl_package 中。在发行时，推荐不要重新分配 npl_poackage 文件夹，但是可以复制/合并它们的内容到工作目录，预编译所有源代码及 package 代码和/或一个或多个归档文件

在哪里寻找 NPL Packages

Idreamtech

NPLPackage 下每个工作目录都是一个被团体管理的有效的 npl_package

如何下载一个 Package

简单的在你的开发工作目录下创建一个文件夹，创建一个名称为 `npl_packages` 的替代文件夹。然后在那里运行 git clone 例如：

```
cd npl_packages
git clone https://github.com/NPLPackages/main.git
```

如何发布

不推荐在./npl_package 文件夹中更改或添加文件。推荐在你的项目开发目录中创建一个类似的目录结构如果你想添加或更改 package 的源代码。如果你想要发布到任何的 npl packages， 请在 github 上 fork 它并在 GitHub 上发送 pull 请求到它的作者。

例如，如果你想要更改或添加一个类似于`./npl_packages/main/.../ABC.lua` 的文件而不是在 npl package 文件夹中更改它，你可以简单的在根开发目录中用相同的目录结构，如`./.../ABC.lua`，来新建一个文件。在 runtime 中，你的文件版本将会被加载为 npl package 文件夹的那个版本。

当你的代码成熟后，你可以考虑在另一个地方 fork 已给的 npl_package，然后合并你的已经改变的文件并给作者发送一个 pull 请求。如果作者快速应答，他或她可能会接受你的改变然后你接下来可以在除去你原始项目中以改变的文件。

4 C/C++ NPL Runtime API

4.1 Core API

实现在 C++ 中的 API 可见：https://codedocs.xyz/LiXizhi/NPLRuntime/modules.html，使用方法可见：https://github.com/NPLPackages/main

4.2 Attribute System

几乎所有 C++API 对象如 ParaUIObject，ParaObject，甚至一些全局表（global table）如 ParaEngine，通过 ParaAttributeObject 暴露数据接口。

IdreamTech

属性系统允许我们非常方便地在 C++核心对象中通过 NPL 脚本 get 或 set 数据，如下所示：

```
local attr = ParaEngine.GetAttributeObject();
local value = attr:GetField("IgnoreWindowSizeChange", false);
attr:SetField("IgnoreWindowSizeChange", not value);
```

4.3 AssetManifest

一个图形应用通常依赖大量资源（纹理，模型等等）来加工和渲染。它通常不可能在客户端机器上安装的时候就部署所有资源，一般是在第一次使用的时候从服务器上下载。

NPL/ParaEngine 内置了资源清单系统（asset manifest system）来支持异步资源加载。基本上，它解决了两个问题：

1. 使用一个纯文本文件在服务器后台工作线程中查找和下载资源。
2. 在 ParaEngine 中的大多数 UI 和 3d 对象提供了一个同步的接口来设置资源，就好像他们早就存在了。事实上，当这些资源在本地可用时，它们将自动解析资源(以及它们的依赖项)。

资源清单管理

当一个应用启动后，NPLRuntime 将读取根目录下所有 Assets_manifest*.txt 文件，每个文件有以下内容：

```
format is [relative path],md5,fileSize
```

如果名字以.z 结尾，则它是一个压缩文件。未压缩时候大小可以是 4MB，md5 是文件代码的校验码（checksum）。fileSize 是压缩后文件大小。

```
        audio/music.mp3.z,3799134715,22032
        model/building/tree.dds.z,2957514200,949
        model/building/tree.x.z,2551621901,816
        games/tutorial.swf,1157008036,171105
```

当一个异步加载器尝试加载一个应用资源（纹理，模型等等），它将首先使用 TO-LOWER-CASED 资源路径在 AssetManifest 中搜索，例如（model/building/tree.x）。然后在"temp/cache/"目录下搜索匹配文件。

文件匹配是通过比较资源文件中的行和缓存目录中的文件名来完成的，使用它们的 md5 和 size。

IdreamTech

```
        audio/music.mp3.z,3799134715,22032 matches to file 379913471522032
```

使用示例：

```
        AssetFileEntry* pEntry = 
CAssetManifest::GetSingleton().GetFile("Texture/somefile.dds");
        if(pEntry && pEntry->DoesFileExist())
        {
                // Load from file pEntry->GetLocalFileName();
        }
```

4.4 PKG file loading

包（Package）文件仅仅是一个普通的压缩文件。我们能够在一个或多个包文件中捆绑脚本或资源，并且在运行环境中加载它们。每个被加载的包文件都是一个虚拟的只读文件系统。它在标准文件系统之上提供了一个分层的文件接口。每一个包文件就像一个搜索路径，最后打开的最先搜索。

生成包文件

有两种包文件：一种是标准的*.zip 文件，另一种叫做*.pkg 文件。我们能够使用任何外部工具来生成 zip 文件，或者使用内置的 NPL API，如下所示：

```
-- testing creating zip files
local zipname = "temp/simple.zip";
local writer = ParaIO.CreateZip(zipname,"");
--writer:ZipAdd("lua.dll", "lua.dll");
writer:ZipAdd("aaa.txt", "deletefile.list");
--writer:ZipAddFolder("temp");
-- writer:AddDirectory("worlds/", "d:/temp/*.", 4);
writer:close();
```

我们可以使用如下所示的 NPL API 创建*.pkg 文件来形成标准的 zip 文件。*.pkg 在 zip 文件上使用了一种简单的加密算法。

```
ParaAsset.GeneratePkgFile("main.zip", "main.pkg");
```

加载包文件

当应用启动后，NPL 运行环境将自动在应用启动目录加载所有 main*.pkg 和 main*.zip 文件到内存。加载的顺序是基于文件名的，因此一个在"main_patch2.pkg"中的文件将会重写在"main_patch1.pkg"中的相同文件。

请注意，加载包文件是非常快速的，它仅仅会复制整个 zip 文件到内存，个别脚本文件

或资源会在第一次使用时候解压并解析。

程序员同样能够以编程的方式使用 NPL API 加载或者卸载任何存档文件，如下所示：

```
NPL.load("pluginABC.pkg");
-- or using explicit calls
ParaAsset.OpenArchive("pluginABC.pkg", true);
```

如果另一个函数没有找到，两个函数都将会搜索 pkg 和 zip 文件扩展名。ParaAsset.OpenArchive 的第二个参数是是否使用存档文件中的相对路径。(例如在存档文件中的文件路径相对于包含的目录)

我们能够明确地指定根目录，然后在 zip 文件中的所有相对文件路径都被认为是相对于该目录。

```
local filename = "pluginABC.zip";
local zip_archive =
ParaEngine.GetAttributeObject():GetChild("AssetManager"):GetChild("CFileManager"):GetChild(filename);
zip_archive:SetField("RootDirectory", "pluginABC/");
```

我们还可以将基本目录从 zip 文件中的所有文件的相对路径中删除。

```
local filename = "pluginABC.zip";
local zip_archive =
ParaEngine.GetAttributeObject():GetChild("AssetManager"):GetChild("CFileManager"):GetChild(filename);
zip_archive:SetField("SetBaseDirectory", "pluginABC-master/Mod/pluginABC/");
```

4.5 ParaObject

ParaObject 是在 C++引擎上的 3D 对象脚本代理。在大多数情况下，它可能是一个 3d 网格，一个叫做两足动物（biped）的动画对象，一个最大模型（bmax model）或者任何自定义 3D 对象。

创建 3D 对象

使用 CreateCharacter 来创建基于 ParaX 资源文件的动画角色。

```
local player = ParaScene.CreateCharacter ("MyPlayer",
ParaAsset.LoadParaX("","character/v3/Elf/Female/ElfFemale.x"), "", true,
0.35, 0, 1.0);
player:SetPosition(ParaScene.GetPlayer():GetPosition());
ParaScene.Attach(player);
```

使用 CreateMeshPhysicsObject 来创建基于网格资源文件的静态网格对象。

```
local asset = ParaAsset.LoadStaticMesh("","model/common/editor/z.x")
local obj = ParaScene.CreateMeshPhysicsObject("blueprint_center", asset,
1,1,1, false, "1,0,0,0,1,0,0,0,1,0,0,0");
obj:SetPosition(ParaScene.GetPlayer():GetPosition());
obj:GetAttributeObject():SetField("progress",1);
ParaScene.Attach(obj);
```

获取视图参数

```
local obj = ParaScene.GetObject("MyPlayer")
local params = {};
param.rotation = obj:GetRotation({})
param.scaling = obj:GetScale();
param.facing = obj:GetFacing();
param.ViewBox = obj:GetViewBox({});
local x,y,z = obj:GetViewCenter();
```

请注意，所有资源都是异步加载的，当资源还没有加载时，对象不会渲染任何东西，上述参数中有关网格资源没有异步加载时可能不正确。

4.6 ParaUIObject

ParaUIObject 是由一类定义在 NPL Runtime 中由 C++编写的类对象的集合，它包含 container (容器)， button (按钮)，scrollbar (滚动条)， editbox (编辑框)，imeeditbox（即时编辑框）,slider（滑块）, video（视频对象）, 3dcanvas（3d 画布）, listbox（列表框）, painter（画板） and text（文本）。

当创建 UI 对象时，可以在 lua 脚本中用 ParaUI.CreateUIObject(string strType, string strObjectName, string alignment, number x, number y, number width, number height)创建与 strType 相对应的 UI 对象，strType 可以是上述类对象中任何一类，使用这样的方法，实际上是调用了 C++中的 CreateUIObject 方法，这个方法会根据我们传入的 strType 调用相应类的构造方法，从而创建相应的 UI 对象。

5 System Libraries

所有系统库（system library）都包含在 main package 中。

5.1 Timer

使用示例如下所示：

```
NPL.load("(gl)script/ide/timer.lua");

local mytimer = commonlib.Timer:new({callbackFunc = function(timer)
        commonlib.log({"ontimer", timer.id, timer.delta, timer.lastTick})
end})

-- start the timer after 0 milliseconds, and signal every 1000 millisecond
mytimer:Change(0, 1000)

-- start the timer after 1000 milliseconds, and stop it immediately.
mytimer:Change(1000, nil)

-- now kill the timer.
mytimer:Change()

-- kill all timers in the pool
commonlib.TimerManager.Clear()

-- dump timer info
commonlib.TimerManager.DumpTimerCount()

-- get the current time in millisecond. This may be faster than
ParaGlobal_timeGetTime() since it is updated only at rendering frame rate.
commonlib.TimerManager.GetCurrentTime();

-- one time timer
commonlib.TimerManager.SetTimeout(function()  end, 1000)
```

5.2 Serialization

在 API 函数层面对 table 已经集成了大量序列化函数，如下所示，只需要调用 commonlib.serialize_compact 即可将 table 序列化为字符串。，同时 NPL.LoadTableFromString 可将字符串反序列化成 table。

```
local o = {a=1, b="string"};

-- serialize to string
local str = commonlib.serialize_compact(o)
-- write string to log.txt
log(str);
-- string to NPL table again
local o = NPL.LoadTableFromString(str);
-- echo to output any object to log.txt
echo(o);
```

5.3 Encoding

UTF8 vs Default Text Encoding

编码是一个复杂的话题。在 NPL 中的经验法则是在任何可能的情况下使用 utf8 编码，例如在源代码，XML/html/page 文件，UI 文本，网络消息等等。然而，系统文件路径必须用操作系统默认的编码方式来编码，这就可能会与 utf8 不同，所以我们需要使用以下方法将系统文件路径的文本在默认编码和 utf8 编码之间转换，例如 ParaIO.open(filename) 需要 filename 用默认编码方式。

```
NPL.load("(gl)script/ide/Encoding.lua");
local Encoding = commonlib.gettable("commonlib.Encoding");
commonlib.Encoding.Utf8ToDefault(text)
commonlib.Encoding.DefaultToUtf8(text)
```

UTF8 to/from UTF16

NPL 源代码和页面文件总是被看作是 utf8 编码的文本。我们一直建议在你的应用的所有可能的场合均使用 utf8 编码。然而，如果你获得一些 ufc6 编码的字符串，例如来自 javascript 的字符串，你能够使用以下方法将之转变成 utf8。

```
NPL.load("(gl)script/ide/Encoding.lua");
local Encoding = commonlib.gettable("commonlib.Encoding");

local a = Encoding.Utf8ToUtf16("中文")

local b = Encoding.Utf16ToUtf8(a);

assert(b == "中文");
```

Json Encoding

```
NPL.load("(gl)script/ide/Json.lua");
local t = {
["name1"] = "value1",
["name2"] = {1, false, true, 23.54, "a \021 string"},
name3 = commonlib.Json.Null()
}

local json = commonlib.Json.Encode (t)
print (json)
--> {"name1":"value1","name3":null,"name2":[1,false,true,23.54,"a \u0015
string"]}
```

```
local t = commonlib.Json.Decode(json)
print(t.name2[4])
--> 23.54

-- also consider the NPL version
local out={};
if(NPL.FromJson(json, out)) then
	commonlib.echo(out)
end
```

XML Encoding

```
NPL.load("(gl)script/ide/LuaXML.lua");
function TestLuaXML:test_LuaXML_CPlusPlus()
	local input = [[first childboldsecond child]]
	local x = ParaXML.LuaXML_ParseString(input);
	assert(x[1].name == "paragraph");
end

function TestLuaXML:test_LuaXML_NPL()
	local input = [[first childboldsecond child]]
	local xmlRoot = commonlib.XML2Lua(input)
	assert(commonlib.Lua2XmlString(xmlRoot) == input);
	log(commonlib.Lua2XmlString(xmlRoot, true))
end
```

Binary Data

读取或者写入二进制数据到或从字符串，我们可以用一个特殊的被叫做<memory>的文件名来使用文件 API，如下。

- 读取二进制字符串，简单地调用 WritingString 来输入的字符串到一个储存缓冲区文件然后 seek(0)并且用你喜欢的任意方式将数据读取出来。
- 写入二进制字符串，简单地调用 WritingString, WriteBytes 或者 WritingInt，一旦完成，调用 GetText(0, -1)来获取最后输出的二进制字符串。

```
function test_MemoryFile()
	-- "<memory>" is a special name for memory file, both read/write is possible.
	local file = ParaIO.open("<memory>", "w");
	if(file:IsValid()) then
		file:WriteString("hello ");
		local nPos = file:GetFileSize();
		file:WriteString("world");
		file:WriteInt(1234);
```

```
                file:seek(nPos);
                file:WriteString("World");
                file:SetFilePointer(0, 2); -- 2 is relative to end of file
                file:WriteInt(0);
                file:WriteString("End");
                file:WriteBytes(3, {100, 0, 22});
                -- read entire binary text data back to npl string
                echo(#(file:GetText(0, -1)));
                file:close();
        end
end
```

XPath query in XML

```
NPL.load("(gl)script/ide/XPath.lua");

local xmlDocIP = ParaXML.LuaXML_ParseFile("script/apps/Poke/IP.xml");
local xpath = "/mcml:mcml/mcml:packageList/mcml:package/";
local xpath = "//mcml:IPList/mcml:IP[@text = 'Level2_2']";
local xpath = "//mcml:IPList/mcml:IP[@version = 5]";
local xpath = "//mcml:IPList/mcml:IP[@version < 6]"; -- only supported by
selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version > 4]"; -- only supported by
selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version >= 5]"; -- only supported by
selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version <= 5]"; -- only supported by
selectNodes2

local xmlDocIP =
ParaXML.LuaXML_ParseFile("character/v3/Pet/MGBB/mgbb.xml");
local xpath = "/mesh/shader/@index";
local xpath = "/mesh/boundingbox/@minx";
local xpath = "/mesh/submesh/@filename";
local xpath = "/mesh/submesh";


--
-- select nodes to an array table
--
local result = commonlib.XPath.selectNodes(xmlDocIP, xpath);
local result = XPath.selectNodes(xmlDocIP, xpath, nMaxResultCount); --
select at most nMaxResultCount result


--
-- select a single node or nil
--
```

```
local node = XPath.selectNode(xmlDocIP, xpath);

--
-- iterate on all nodes.
--
for node in commonlib.XPath.eachNode(xmlDocIP, xpath) do
        commonlib.echo(node[1]);
end
```

5.4 Log

```
-- write formatted logs to log.txt
LOG.std(nil,"info", "sub_system_name", "some formated message: %s",
"hello");
LOG.std(nil,"debug", "sub_system_name", {a=1, c="any table object"});
LOG.std(nil,"error", "sub_system_name", "error code here");
LOG.std(nil,"warn", "sub_system_name", "warning");
```

Log 重定向和配置：可以在启动时将 log.txt 重定向到不同的文件。

```
NPL.load("(gl)script/ide/log.lua");
commonlib.log("hello %s \n", "paraengine")
commonlib.log({"anything"})
local fromPos = commonlib.log.GetLogPos()
log(fromPos.." babababa...\n");
local text = commonlib.log.GetLog(fromPos, nil)
log(tostring(text).." is retrieved\n")

commonlib.applog("hello paraengine"); --> ./log.txt --> 20090711
02:59:19|0|hello paraengine|script/shell_loop.lua:23: in function
FunctionName|
commonlib.applog("hello %s", "paraengine")

commonlib.servicelog("MyService", "hello paraengine"); -
-> ./MyService_20090711.log --> 2009-07-11 10:53:27|0|hello paraengine||
commonlib.servicelog("MyService", "hello %s", "paraengine");

-- set log properties before using the log
commonlib.servicelog.GetLogger("no_append"):SetLogFile("log/no_append.log")
commonlib.servicelog.GetLogger("no_append"):SetAppendMode(false);
commonlib.servicelog.GetLogger("no_append"):SetForceFlush(true);
commonlib.servicelog("no_append", "test");

-- This will change the default logger's file position at runtime.
commonlib.servicelog.GetLogger(""):SetLogFile("log/log_2016.5.19.txt");
```

5.5 Http request

下载文件或发出标准请求：

```
System.os.GetUrl("https://github.com/LiXizhi/HourOfCode/archive/master.zip"
, echo);
```

只使用“-I”选项获取标题（headers）：

```
System.os.GetUrl("https://github.com/LiXizhi/HourOfCode/archive/master.zip"
, function(err, msg, data)  echo(msg) end, "-I");
```

用 http post 发送表单 KV 对（form KV pairs）：

```
System.os.GetUrl({url =
"http://localhost:8099/ajax/console?action=getparams", form =
{key="value",} }, function(err, msg, data)            echo(data)
        end);
```

使用 http post 发送多部分二进制表单（multi-part binary forms）：

```
System.os.GetUrl({url =
"http://localhost:8099/ajax/console?action=printrequest", form = {name =
{file="dummy.html",      data="<html><bold>bold</bold></html>",
type="text/html"}, } }, function(err, msg, data)          echo(data)
        end);
```

发送二进制数据：

```
System.os.GetUrl({url =
"http://localhost:8099/ajax/console?action=printrequest",
headers={["content-type"]="application/json"},
postfields='{"key":"value"}' }, function(err, msg, data)
        echo(data)       end);
```

为了简化 json 编码，我们可以使用以下快捷方法将表单作为 json 字符串发送：

```
System.os.GetUrl({url =
"http://localhost:8099/ajax/console?action=getparams", json = true, form =
{key="value", key2 ={subtable="subvalue"} } }, function(err, msg, data)
echo(data)  end);
```

HTTP PUT 请求：

```
System.os.GetUrl({
        method = "PUT",
        url = "http://localhost:8099/ajax/log?action=log",
        form = {filecontent = "binary string here", }
}, function(err, msg, data)              echo(data)  end);
```

HTTP DELETE 请求：

```
System.os.GetUrl({
        method = "DELETE",
        url = "http://localhost:8099/ajax/log?action=log",
        form = {filecontent = "binary string here", }
```

```
}, function(err, msg, data)                echo(data)  end);
```

在服务器端，假设是一个 NPL web 服务器，我们可以使用 request:GetBody()或者 request：getparams()来获取请求主体（request body)。前者将只会包含请求主体，或者还会包含 url 参数。

调试 HTTP 请求：

在 NPL 代码 Wiki 的控制台窗口，我们可以通过发送一个请求到 /ajax/console?action=printrequest 来测试原始 http 请求，然后检查原始请求内容的日志控制台。

```
System.os.GetUrl({url =
"http://localhost:8099/ajax/console?action=printrequest", echo);
```

本地服务器

本地服务器提供给我们一种下载文件或者使用缓存策略（cache policy）发出 url 请求的方法。本地数据库系统用来备份所有 url 请求。缓存文件可能在数据库中括者在本地文件系统中。

使用本地服务器作为一个本地数据库

我们可以使用本地服务器作为一个简单的有 cache_policy 函数的（name，value）对数据库。要查询数据库的条目调用，这里使用 web 服务商店（web service store)：

```
        NPL.load("(gl)script/ide/System/localserver/factory.lua");
        local ls = System.localserver.CreateStore(nil, 2);
        if(not ls) then
                return
        end
        cache_policy = cache_policy or
System.localserver.CachePolicy:new("access plus 1 week");

        local url =
System.localserver.UrlHelper.WS_to_REST(fakeurl_query_miniprofile,
{JID=JID}, {"JID"});
        local item = ls:GetItem(url)
        if(item and item.entry and item.payload and not
cache_policy:IsExpired(item.payload.creation_date)) then
                -- NOTE:item.payload.data is always a string, one may
deserialize from it to obtain table object.
                local profile = item.payload.data;
```

```
				if(type(callbackFunc) == "function") then
					callbackFunc(JID, profile);
				end
		else

		end
```

以下添加（更新）数据库条目调用：

```
		NPL.load("(gl)script/ide/System/localserver/factory.lua");
		local ls = System.localserver.CreateStore(nil, 2);
		if(not ls) then
				return
		end
		-- make url
		local url = 
System.localserver.UrlHelper.WS_to_REST(fakeurl_query_miniprofile, 
{JID=JID}, {"JID"});

		-- make entry
		local item = {
				entry = System.localserver.WebCacheDB.EntryInfo:new({
					url = url,
				}),
				payload = System.localserver.WebCacheDB.PayloadInfo:new({
					status_code = 
System.localserver.HttpConstants.HTTP_OK,
					data = msg.profile,
				}),
		}
		-- save to database entry
		local res = ls:PutItem(item)
		if(res) then
				log("ls put JID mini profile for "..url.."\n")
		else
				log("warning: failed saving JID profile item to local 
server.\n")
		end
```

通过 SMTP 发送电子邮件

创建一个 SMTP email 服务器不简单。然而，通过现存的 email 服务器发送电子邮件相当容易。我们能够通过 System.os.GetUrl 并选择一些合适的选项手动地实现，或者我们可以写如下方便的函数。

```
System.os.SendEmail({
```

```
        url="smtp://smtp.exmail.qq.com",
        username="lixizhi@paraengine.com", password="XXXXX",
        -- ca_info = "/path/to/certificate.pem",
        from="lixizhi@paraengine.com", to="lixizhi@yeah.net",
cc="xizhi.li@gmail.com",
        subject = "title here",
        body = "any body context here. can be very long",
}, function(err, msg) echo(msg) end);
```

5.6 FilesAndIO

查找文件：

```
NPL.load("(gl)script/ide/Files.lua");
local result = commonlib.Files.Find({}, "model/test", 0, 500,
function(item)
        local ext = commonlib.Files.GetFileExtension(item.filename);
        if(ext) then
                return (ext == "x") or (ext == "dds")
        end
end)

-- search zip files using perl regular expression. like ":^xyz\\s+.*blah$"
local result = commonlib.Files.Find({}, "model/test", 0, 500, ":.*",
"*.zip")

-- using lua file system
local lfs = commonlib.Files.GetLuaFileSystem();
echo(lfs.attributes("config/config.txt", "mode"))
```

在 Zip 文件中查找并读取文件：

```
function test_search_zipfile()
        local zipPath = "temp/test.zip";

        -- open with relative file path
        if(ParaAsset.OpenArchive(zipPath, true)) then
                local zip_archive =
ParaEngine.GetAttributeObject():GetChild("AssetManager"):GetChild("CFileMan
ager"):GetChild(zipPath);
                --      zipParentDir is usually the parent directory
"temp/" of zip file.
                local zipParentDir = zip_archive:GetField("RootDirectory",
"");

                echo(zipParentDir);

                -- search just in a given zip archive file
```

```
                local filesOut = {};
                -- ":.", any regular expression after : is supported. `.` 
match to all strings.
                commonlib.Files.Find(filesOut, "", 0, 10000, ":.", 
zipPath);

                -- print all files in zip file
                for i = 1,#filesOut do
                        local item = filesOut[i];
                        echo(item.filename .. " size: "..item.filesize);
                        if(item.filesize > 0) then
                                local file = 
ParaIO.open(zipParentDir..item.filename, "r")
                                if(file:IsValid()) then
                                        -- get binary data
                                        local binData = file:GetText(0, -
1);

                                        -- dump the first few characters 
in the file
                                        echo(binData:sub(1, 10));
                                        file:close();
                                end
                        else
                                -- this is a folder
                        end
                end
                ParaAsset.CloseArchive(zipPath);
        end
end
```

读/写二进制文件:

```
local file = ParaIO.open("temp/binaryfile.bin", "w");
if(file:IsValid()) then
        local data = "binary\0\0\0\0file";
        file:WriteString(data, #data);
        -- write 32 bits int
        file:WriteUInt(0xffffffff);
        file:WriteInt(-1);
        -- write float
        file:WriteFloat(-3.14);
        -- write double (precision is limited by lua double)
        file:WriteDouble(-3.1415926535897926);
        -- write 16bits word
        file:WriteWord(0xff00);
        -- write 16bits short integer
```

```
        file:WriteShort(-1);
        file:WriteBytes(3, {255, 0, 255});
        file:close();

        -- testing by reading file content back
        local file = ParaIO.open("temp/binaryfile.bin", "r");
        if(file:IsValid()) then
                -- test reading binary string without increasing the file
cursor
                assert(file:GetText(0, #data) == data);
                file:seekRelative(#data);
                assert(file:getpos() == #data);
                file:seek(0);
                -- test reading binary string
                assert(file:ReadString(#data) == data);
                assert(file:ReadUInt() == 0xffffffff);
                assert(file:ReadInt() == -1);
                assert(math.abs(file:ReadFloat() - (-3.14)) < 0.000001);
                assert(file:ReadDouble() == -3.1415926535897926);
                assert(file:ReadWord() == 0xff00);
                assert(file:ReadShort() == -1);
                local o = {};
                file:ReadBytes(3, o);
                assert(o[1] == 255 and o[2] == 0 and o[3] == 255);
                file:seek(0);
                assert(file:ReadString(8) == "binary\0\0");
                file:close();
        end
end
```

同时读写：

```
-- test IO write/read binary file in "rw" mode
function test_IO_BinaryFileReadWrite()

        -- "rw" mode will not destroy the content of existing file.
        local file = ParaIO.open("temp/binaryfile.bin", "rw");
        if(file:IsValid()) then
                local data = "binary\0\0\0\0file";
                file:WriteString(data, #data);
                -- write 32 bits int
                file:WriteUInt(0xffffffff);
                file:WriteInt(-1);
                -- write float
                file:WriteFloat(-3.14);
                -- write double (precision is limited by lua double)
```

```
                file:WriteDouble(-3.1415926535897926);
                -- write 16bits word
                file:WriteWord(0xff00);
                -- write 16bits short integer
                file:WriteShort(-1);
                file:WriteBytes(3, {255, 0, 255});
                file:SetEndOfFile();

                -----------------------------------------
                -- testing by reading file content back
                -----------------------------------------
                file:seek(0);
                -- test reading binary string without increasing the file
cursor
                assert(file:GetText(0, #data) == data);
                file:seekRelative(#data);
                assert(file:getpos() == #data);
                file:seek(0);
                -- test reading binary string
                assert(file:ReadString(#data) == data);
                assert(file:ReadUInt() == 0xffffffff);
                assert(file:ReadInt() == -1);
                assert(math.abs(file:ReadFloat() - (-3.14)) < 0.000001);
                assert(file:ReadDouble() == -3.1415926535897926);
                assert(file:ReadWord() == 0xff00);
                assert(file:ReadShort() == -1);
                local o = {};
                file:ReadBytes(3, o);
                assert(o[1] == 255 and o[2] == 0 and o[3] == 255);
                file:seek(0);
                assert(file:ReadString(8) == "binary\0\0");
                file:close();
        end
end
```

内存中文件和二进制缓冲区：

将二进制数据写成字符串，我们可以使用有着叫做<memory>的特殊名字的文件 API，如下所示：

```
function test_MemoryFile()
        -- "<memory>" is a special name for memory file, both read/write is
possible.
        local file = ParaIO.open("<memory>", "w");
        if(file:IsValid()) then
```

```
                file:WriteString("hello ");
                local nPos = file:GetFileSize();
                file:WriteString("world");
                file:WriteInt(1234);
                file:seek(nPos);
                file:WriteString("World");
                file:SetFilePointer(0, 2); -- 2 is relative to end of file
                file:WriteInt(0);
                file:WriteString("End");
                file:WriteBytes(3, {100, 0, 22});
                -- read entire binary text data back to npl string
                echo(#(file:GetText(0, -1)));
                file:close();
        end
end
```

读/写文件:

```
local function ParaIO_FileTest()

        -- file write
        log("testing file write...\r\n")

        local file = ParaIO.open("temp/iotest.txt", "w");
        file:WriteString("test\r\n");
        file:WriteString("test\r\n");
        file:close();

        -- file read
        log("testing file read...\r\n")

        local file = ParaIO.open("temp/iotest.txt", "r");
        log(tostring(file:readline()));
        log(tostring(file:readline()));
        log(tostring(file:readline()));
        file:close();

end

-- tested on 2007.6.7, LiXizhi
local function ParaIO_ZipFileTest()
        local writer = ParaIO.CreateZip("d:\\simple.zip","");
        writer:ZipAdd("temp/file1.ini", "d:\\file1.ini");
        writer:ZipAdd("temp/file2.ini", "d:\\file2.ini");
        writer:ZipAddFolder("temp");
        writer:AddDirectory("worlds/", "d:/temp/*.", 4);
```

```
        writer:AddDirectory("worlds/", "d:/worlds/*.*", 2);
        writer:close();
end

-- tested on 2007.6.7, LiXizhi
local function ParaIO_SearchZipContentTest()
        -- test case 1
        log("test case 1\n");
        local search_result = ParaIO.SearchFiles("","*.", "d:\\simple.zip",
0, 10000, 0);
        local nCount = search_result:GetNumOfResult();
        local i;
        for i = 0, nCount-1 do
                log(search_result:GetItem(i).."\n");
        end
        search_result:Release();
        -- test case 2
        log("test case 2\n");
        local search_result = ParaIO.SearchFiles("","*.ini",
"d:\\simple.zip", 0, 10000, 0);
        local nCount = search_result:GetNumOfResult();
        local i;
        for i = 0, nCount-1 do
                log(search_result:GetItem(i).."\n");
        end
        search_result:Release();
        -- test case 3
        log("test case 3\n");
        local search_result = ParaIO.SearchFiles("","temp/*.",
"d:\\simple.zip", 0, 10000, 0);
        local nCount = search_result:GetNumOfResult();
        local i;
        for i = 0, nCount-1 do
                log(search_result:GetItem(i).."\n");
        end
        search_result:Release();
        -- test case 4
        log("test case 4\n");
        local search_result = ParaIO.SearchFiles("temp/","*.*",
"d:\\simple.zip", 0, 10000, 0);
        local nCount = search_result:GetNumOfResult();
        local i;
        for i = 0, nCount-1 do
                log(search_result:GetItem(i).."\n");
```

```
		end
		search_result:Release();
		-- test case 5
		log("test case 5\n");
		local search_result = ParaIO.SearchFiles("","temp/*.*",
"d:\\simple.zip", 0, 10000, 0);
		local nCount = search_result:GetNumOfResult();
		local i;
		for i = 0, nCount-1 do
			log(search_result:GetItem(i).."\n");
		end
		search_result:Release();
end

local function ParaIO_SearchPathTest()
		ParaIO.CreateDirectory("npl_packages/test/")
		local file = ParaIO.open("npl_packages/test/test_searchpath.lua",
"w");
		file:WriteString("echo('from test_searchpath.lua')")
		file:close();

		ParaIO.AddSearchPath("npl_packages/test");
		ParaIO.AddSearchPath("npl_packages/test/"); -- same as above, check
for duplicate

		assert(ParaIO.DoesFileExist("test_searchpath.lua"));

		ParaIO.RemoveSearchPath("npl_packages/test");

		assert(not ParaIO.DoesFileExist("test_searchpath.lua"));

		-- ParaIO.AddSearchPath("npl_packages/test/");
		-- this is another way of ParaIO.AddSearchPath, except that it will
check for folder existence.
		-- in a number of locations.
		NPL.load("npl_packages/test/");

		-- test standard open api
		local file = ParaIO.open("test_searchpath.lua", "r");
		if(file:IsValid()) then
			log(tostring(file:readline()));
		else
			log("not found\n");
		end
```

```
        file:close();

        -- test script file
        NPL.load("(gl)test_searchpath.lua");

        ParaIO.ClearAllSearchPath();

        assert(not ParaIO.DoesFileExist("test_searchpath.lua"));
end

-- TODO: test passed on 2008.4.20, LiXizhi
function ParaIO_PathReplaceable()
        ParaIO.AddPathVariable("WORLD", "worlds/MyWorld")
        if(ParaIO.AddPathVariable("userid", "temp/LIXIZHI_PARAENGINE"))
then
                local fullpath;
                commonlib.echo("test simple");
                fullpath = ParaIO.DecodePath("%WORLD%/%userid%/filename");
                commonlib.echo(fullpath);
                fullpath = ParaIO.EncodePath(fullpath)
                commonlib.echo(fullpath);

                commonlib.echo("test encoding with a specified
variables");
                fullpath = ParaIO.DecodePath("%WORLD%/%userid%/filename");
                commonlib.echo(fullpath);
                commonlib.echo(ParaIO.EncodePath(fullpath, "WORLD"));
                commonlib.echo(ParaIO.EncodePath(fullpath, "WORLD,
userid"));

                commonlib.echo("test encoding with inline path");
                fullpath = ParaIO.DecodePath("%WORLD%/%userid%_filename");
                commonlib.echo(fullpath);
                fullpath = ParaIO.EncodePath(fullpath)
                commonlib.echo(fullpath);


                commonlib.echo("test nested");
                fullpath =
ParaIO.DecodePath("%userid%/filename/%userid%/nestedtest");
                commonlib.echo(fullpath);
                fullpath = ParaIO.EncodePath(fullpath)
                commonlib.echo(fullpath);
```

```
                commonlib.echo("test remove");
                if(ParaIO.AddPathVariable("userid", nil)) then
                        fullpath = ParaIO.DecodePath("%userid%/filename");
                        commonlib.echo(fullpath);
                        fullpath = ParaIO.EncodePath(fullpath)
                        commonlib.echo(fullpath);
                end

                commonlib.echo("test full path");
                fullpath = ParaIO.DecodePath("NormalPath/filename");
                commonlib.echo(fullpath);
                fullpath = ParaIO.EncodePath(fullpath)
                commonlib.echo(fullpath);
        end
end

function ParaIO_SearchFiles_reg_expr()
        -- test case 1
        local search_result = ParaIO.SearchFiles("script/ide/",":.*",
"*.zip", 2, 10000, 0);
        local nCount = search_result:GetNumOfResult();
        local i;
        for i = 0, nCount-1 do
                log(search_result:GetItem(i).."\n");
        end
        search_result:Release();
end

function test_excel_doc_reader()
        NPL.load("(gl)script/ide/Document/ExcelDocReader.lua");
        local ExcelDocReader =
commonlib.gettable("commonlib.io.ExcelDocReader");
        local reader = ExcelDocReader:new();

        -- schema is optional, which can change the row's keyname to the
defined value.
        reader:SetSchema({
                [1] = {name="npcid", type="number"},
                [2] = {name="superclass", validate_func=function(value)
return value or "menu1"; end },
                [3] = {name="class", validate_func=function(value)  return
value or "normal"; end },
                [4] = {name="class_name", validate_func=function(value)
return value or "通用"; end },
```

```
                [5] = {name="gsid", type="number" },
                [6] = {name="exid", type="number" },
                [7] = {name="money_list", },
        })
        -- read from the second row
        if(reader:LoadFile("config/Aries/NPCShop/npcshop.xml", 2)) then
                local rows = reader:GetRows();
                echo(rows);
        end



        NPL.load("(gl)script/ide/Document/ExcelDocReader.lua");
        local ExcelDocReader =
commonlib.gettable("commonlib.io.ExcelDocReader");
        local reader = ExcelDocReader:new();

        -- schema is optional, which can change the row's keyname to the
defined value.
        local function card_and_level_func(value)
                if(value) then
                        local level, card_gsid =
value:match("^(%d+):(%d+)");
                        return {level=level, card_gsid=card_gsid};
                end
        end
        reader:SetSchema({
                {name="gsid", type="number"},
                {name="displayname"},
                {name="max_level", type="number" },
                {name="hp", type="number" },
                {name="attack", type="number" },
                {name="defense", type="number" },
                {name="powerpips_rate", type="number" },
                {name="accuracy", type="number" },
                {name="critical_attack", type="number" },
                {name="critical_block", type="number" },
                {name="card1",  validate_func= card_and_level_func},
                {name="card2",  validate_func= card_and_level_func},
                {name="card3",  validate_func= card_and_level_func},
                {name="card4",  validate_func= card_and_level_func},
                {name="card5",  validate_func= card_and_level_func},
                {name="card6",  validate_func= card_and_level_func},
                {name="card7",  validate_func= card_and_level_func},
```

```
                {name="card8",  validate_func= card_and_level_func},
        })
        -- read from the second row
        if(reader:LoadFile("config/Aries/Others/combatpet_levels.excel.teen
.xml", 2)) then
                local rows = reader:GetRows();
                log(commonlib.serialize(rows, true));
        end
end
```

删除文件：

ParaIO.DeleteFile("temp/*.*");

Zlip 和 Gzip 的压缩和解压：

```
-- compress/decompress test
function TestNPL.Compress()
        -- using gzip
        local content = "abc";
        local dataIO = {content=content, method="gzip"};
        if(NPL.Compress(dataIO)) then
                echo(dataIO);
                if(dataIO.result) then
                        dataIO.content = dataIO.result; dataIO.result =
nil;
                        if(NPL.Decompress(dataIO)) then
                                echo(dataIO);
                                assert(dataIO.result == content);
                        end
                end
        end

        -- using zlib and deflate
        local content = "abc";
        local dataIO = {content=content, method="zlib", windowBits=-15,
level=3};
        if(NPL.Compress(dataIO)) then
                echo(dataIO);
                if(dataIO.result) then
                        dataIO.content = dataIO.result; dataIO.result =
nil;
                        if(NPL.Decompress(dataIO)) then
                                echo(dataIO);
                                assert(dataIO.result == content);
                        end
```

```
                end
        end
end
```

运行 Shell 命令：

在不论同步或异步模式下，通过 windows 批处理或 linux bash shell 运行外部命令行：

```
NPL.load("(gl)script/ide/System/os/run.lua");
if(System.os.GetPlatform()=="win32") then
        -- any lines of windows batch commands
        echo(System.os("dir *.exe \n svn info"));
        -- this will popup confirmation window, so there is no way to get
its result.
        System.os.runAsAdmin('reg add "HKCR\\paracraft" /ve /d
"URL:paracraft" /f');
else
        -- any lines of linux bash shell commands
        echo(System.os.run("ls -al | grep total\ngit | grep commit"));
end
-- async run command in worker thread
for i=1, 10 do
        System.os.runAsync("echo hello", function(err, result)
echo(result)  end);
end
echo("waiting run async reply ...")
```

请注意：

- windows 和 linux 有不同的默认 shell 程序。你可能需要使用 system.getplatform()== “win32” 来将代码定向到给定的平台。
- 我们可以编写非常长的多行命令。在内部创建一个临时的 shell 脚本文件，并使用 IO 重定向来执行。
- 有时候，我们可能更倾向异步 API 来预防调用线程被卡住，例如 NPL web 服务器为处理图像而调用一些后端程序。异步 API 将消息推送到一个处理器队列，并立即返回，我们可以使用一个或多个 NPL 线程来处理任意数量的队列 shell 命令。
- 你需要拥有运行这些脚本的权限。在 windows10 或以前，我们提供 system.os.runAdmin 来自动弹出确认对话框以作为管理员运行。在 linux 下，我们什么都不能做，你必须在./temp 文件夹下指定执行许可。

读取图像文件：

```
function test_reading_image_file()
	-- reading binary image file
	-- png, jpg format are supported.
	local filename = "Texture/alphadot.png";
	local file = ParaIO.open(filename, "image");
	if(file:IsValid()) then
		local ver = file:ReadInt();
		local width = file:ReadInt();
		local height = file:ReadInt();
		-- how many bytes per pixel, usually 1, 3 or 4
		local bytesPerPixel = file:ReadInt();
		echo({ver, width=width, height = height, bytesPerPixel = 
bytesPerPixel})
		local pixel = {};
		for y=1, height do
			for x=1, width do
				pixel = file:ReadBytes(bytesPerPixel, 
pixel);
				echo({x, y, rgb=pixel})
			end
		end
		file:close();
	end
end
```

6 UI

6.1 用 2D API 绘制

绘制 2D 图形对象存在两种低级方式：

- 一个是通过创建 ParaUIObject 控件（例如按钮，容器，文本框）。这些控件都在 C++中管理，在 NPL 脚本中创建。
- 另一个是创建一个专属的（只用于某一特定用途）用于绘制的 ParaUIObject（原文为 The other is creating owner draw ParaUIObject），然后所有绘制均通过绘画 API（Painting API）在 NPL 脚本中进行。例如绘制矩形，线条，文字等等。

此外，在 NPL 中还有许多高级的图形库，因此我们能够更轻松地创建 2D 用户界面。

- IDE 控件是经过 NPL 包装的低级 C++API。它提供了比未经过处理的

ParaUIObjects 更通用的控件。

- 旧的实现方式是基于 ParaUIObject 控件。
- 新的实现方式是基于绘画 API，在 System.Window.UIElement 命名空间中。

- MCML/NPL 是一个用来创建用户界面的想 HTML/JS 一样的标记语言。
  - 旧的实现方式是基于 ParaUIObject 控件。
  - 新的实现方式是基于绘画 API，在 System.Window.mcml 命名空间中。

以下是关于使用 C++端的低级绘画 API 创建 GUI 元素实例：

按钮（button）

创建一个拥有纹理背景和文本信息的按钮。根元素（root element）是一个容器。请在工作目录下存放一个 png 纹理 Texture/alphadot.png。

```
NPL.load("(gl)script/ide/System/System.lua");

local clickCount = 1;
local function CreateUI()
   local _this = ParaUI.CreateUIObject("button", "MyBtn", "_lt", 10, 10,
64, 22);
   _this.text= "text";
   _this.background = "Texture/alphadot.png";
   _this:SetScript("onclick", function(obj)
      obj.text = "clicked "..clickCount;
      clickCount = clickCount + 1;
   end)
   _this:AttachToRoot();
end
CreateUI();
```

容器（container）

创建一个容器内有一个子对象按钮。

```
local _parent, _this;
        _this = ParaUI.CreateUIObject("container", "MyContainer", "_lt",
10, 110, 200, 64);
        _this:AttachToRoot();
        _this.background = "Texture/alphadot.png";
        _parent = _this;

        -- create a child
        _this = ParaUI.CreateUIObject("button", "b", "_rt", -10-32, 10, 32,
```

```
22);
        _this.text= "X"
        _this.background = "";
        _parent:AddChild(_this);
```

系统字体（System Fonts）

在 ParaEngine/NPL 中，我们支持从*.ttf 文件载入字体。我们可以在 fonts/*.ttf 中安装自定义字体，然后可以通过文件名使用它们，例如 Verdana;bold。请注意，在 ParaEngine 中所有控件默认使用“System”字体。“System”字体对应着当前计算机默认使用的字体。然而，我们能够将之改变成安装在 fonts/*.ttf 中的自定义字体。假如你有一个字体文件在 fonts/ParaEngineThaiFont.ttf，你能通过以下方式改变系统字体。

```
Config.AppendTextValue("GUI_font_mapping","System");
Config.AppendTextValue("GUI_font_mapping","ParaEngineThaiFont");
```

它是一对函数调用，第一个是名字，第二个是值。我们需要记得在 script/config.lua 中这么做，它会在任何 UI 控件创建之前被加载。

注意，对于每种有着不同大小（size），粗细（weight），和名字（name）的字体，我们将会创建一个不同的内部临时图像文件用来渲染，所以我们建议在你的应用中只使用少数几种字体。

GUI 图片

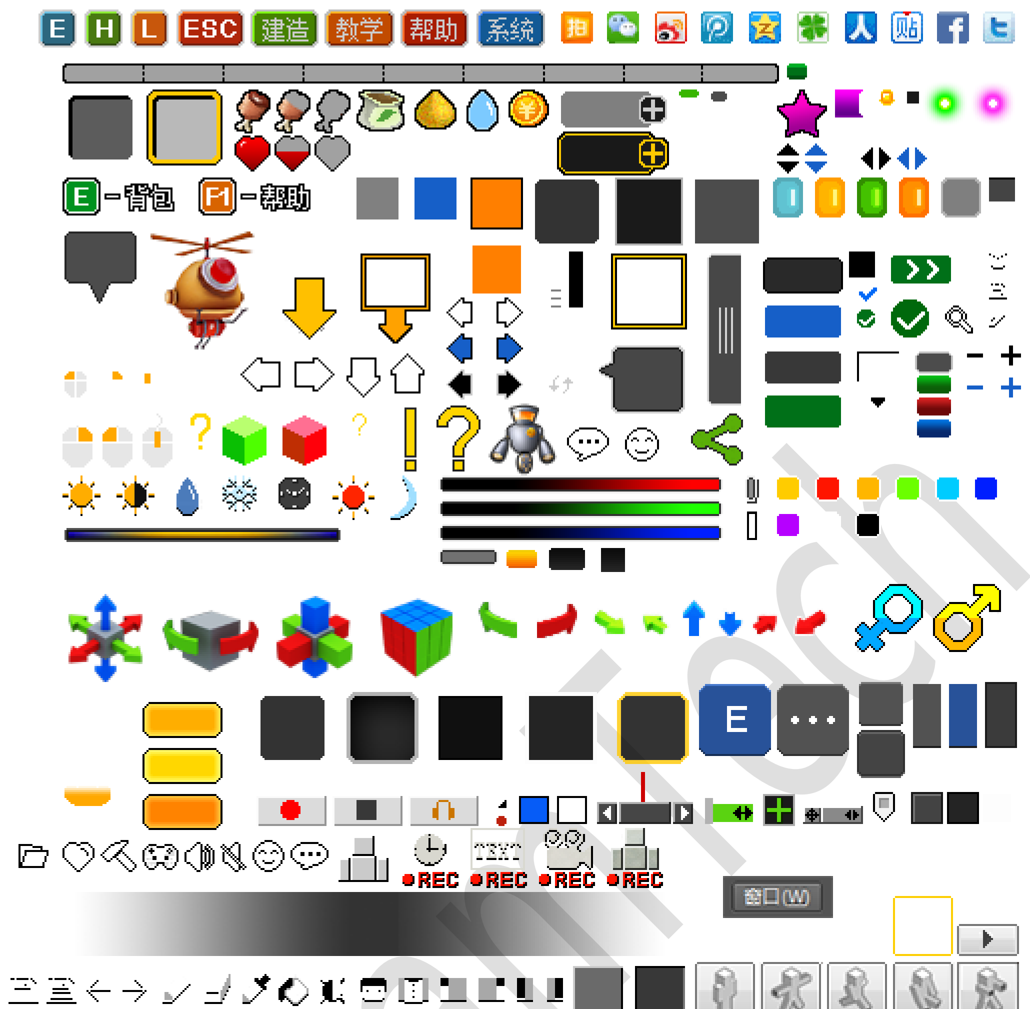
ParaUIObject.background 属性支持一个叫做图片裁剪的功能。他能够使用一张图片的特定区域。将图片分成 9 个区域，形成 9 张子图片，然后自动拉伸中间的 5 张图片到目标大小（角上 4 张图片不会被拉伸）。这在创建不同大小的好看的按钮时非常有用。如下图所示。

IdreamTech

操作语法：Filename.png;left top width height:px py pr pb

- 例如：obj.background="someimage_32bits.png;0 0 64 21: 10 10 10 10"
- ;left top width height 是可选的，它指定了一个子区域。如果没有指定表明选择整个图片。
- :px py pr pb 意思是和左，上，右，下的距离。
- _32bits.png 意味着我们应该使用 32bits 颜色，否则我们将对颜色进行压缩来节省视频内存（vedio memory）。
- 图片大小必须为 2 的幂（order of 2），像 2,4,8,16,64,128,256 等等。因为 directX/openGL 仅仅支持 2 的幂大小的图片。如果你的图片不是 2 的幂大小，它们将会被拉伸而且渲染完成后会模糊不清。

下图是一个用于实际 3D 应用（Paracraft）UI 图片的实例，在实际操作中，往往会将各种 UI 图片集合到一张图片上，在使用时调用相关方法，选取图片中的特定区域。



6.2 使用 MCML

MCML（Micro Cosmos Markup Language）是从 2008 年来写在 NPL 中的一种元语言（meta language），从那时开始，它就成了 NPL 中编写 2D 用户界面的标准。MCML 的灵感来自于早期版本的 ASP.net，但是它是用本地架构来实现本地用户界面。

使用示例

以下是在一个 .html 文件中的 mcml 页面示例，例如 source/HelloWorldMCML/mcml_window.html。使用 html 扩展名仅仅是为了让我们能够在其他 html 代码编辑器中预览这个界面。

```
<pe:mcml>
<script refresh="false" type="text/npl" src="mcml_window.lua"><![CDATA[
    function OnClickOK()
        local content = Page:GetValue("content") or "";
        _guihelper.MessageBox("你输入的是:"..content);
```

ldreamtech

```
    end
]]></script>
<div style="color:#33ff33;background-color:#808080">
    <div style="margin:10px;">
        Hello World from HTML page!
    </div>
    <div style="margin:10px;">
        <input type="text" name="content" style="width:200px;height:25px;"
/>
        <input type="button" name="ok" value="确定" onclick="OnClickOK"/>
    </div>
</div>
</pe:mcml>
```

想要渲染该界面，我们可以用如下窗口启动它。

```
NPL.load("(gl)script/ide/System/Windows/Window.lua");
        local Window = commonlib.gettable("System.Windows.Window")
        local window = Window:new();
        window:Show({
                url="source/HelloWorldMCML/mcml_window.html",
                alignment="_lt", left = 300, top = 100, width = 300,
height = 400,
        });
```

MCML V1 vs V2

MCML 存在两种实现方式：v1 和 v2。上述例子是用 v2 实现的，也是首选的实现方式。然而，v1 是更加成熟和稳定的实现方式，v2 是全新的并且现在没有 v1 那么多的内置控件。我们现在仍在完善 v2，希望它能够和 v1 100%兼容，和 v1 一样强大。

MCML 标签

以下是可以用来创建交互内容（interactive content）的标签。此外我们也可以用 UI 控件创建自定义标签。所有 mcml 内置的标签都在 pe: xml 命名空间，pe 表示 ParaEngine：pe:div、pe:button、pe:container、pe:custom、pe:editbox、pe:identicon、pe:if、pe:input、pe:repeat、pe:script、pe:span、pe:text

标准的 html 标签对应关系如下：

div 标签对应 pe:div 类；plain text 对应 pe:text；input type=button 对应 pe:button 等等。

示例

以下是一个可以在 paracraft 包中浏览到的例子，其中大部分是用 mcml v1 编写的，然而，对 mcml v2 来说，语法是一样的。

```
<pe:mcml>
<script type="text/npl" refresh="false">
<![CDATA[
globalVar = "test global var";
function OnButtonClick()
    _guihelper.MessageBox("refresh page now?", function()
        Page:SetValue("myBtn", "refreshed");
        Page:Refresh(0);
    end);
end

repeat_data = { {a=1}, {a=2} };
function GetDS()
    return repeat_data;
end
]]>
</script>
    <div align="right" style="background-
color:#ff0000;margin:10px;padding:10px;min-width:400px;min-height:150px">
        <div align="right"
style="padding:5px;background:url(Texture/Aries/Common/ThemeKid/btn_thick_h
l_32bits.png:7 7 7 7)">
            <div style="float:left;background-
color:#000000;color:#ffd800;font-size:20px;">

                hello world 中文<font color="#ff0000">fontColor</font>

            </div>
            <div style="float:left;background-color:#0000ff;margin-
left:10px;width:20px;height:20px;">
            </div>
        </div>
        <div valign="bottom">
            <div style="float:left;background-
color:#00ff00;width:60px;height:20px;">
            </div>
            <div style="float:left;background-color:#0000ff;margin-
left:10px;width:20px;height:20px;">
            </div>
        </div>
    </div>
    <div>
```

```
        hello world 中文<font color="#ff0000">fontColor</font>

        <span color="#ff0000">fontColor</span>
    </div>
    <span color="#ff0000">
        <%=Eval('globalVar')%>
        <%=document.write('hello world')%>
    </span>
    <pe:container style="padding:5px;background-color:#ff0000">
        button:<input type="button" name="myBtn" value="RefreshPage"
onclick="OnButtonClick" style=""/>
        editbox:<input type="text" name="myEditbox" value="" style="margin-
left:2px;width:64px;height:25px"/><br/>
        pe_if: <pe:if condition='<%=globalVar=="test global
var"%>'>true</pe:if>
    </pe:container>
    <pe:repeat value="item in repeat_data" style="float:left">
        <div style="float:left;"><%=item.a%></div>
    </pe:repeat>
    <pe:repeat value="item in GetDS()" style="float:left">
        <div style="float:left;"><%=item.a%></div>
    </pe:repeat>
    <pe:repeat DataSource='<%=GetDS()%>' style="float:left">
        <div style="float:left;"><%=a%></div>
    </pe:repeat>
</pe:mcml>
```

NPL 代码支持和嵌入

在 MCML 页面中，存在三种方式嵌入 NPL 代码。

- 使用<script src="helloworld.lua">，这将会根据当前页面文件加载 npl 文件。
- 在页面文件任何地方将 NPL 代码嵌入<script>中。
- 在 xml 属性中嵌入代码，就像 attrname='<%= %>'。请注意这处决于标签实现是否支持它。对于每一个属性，你能够将代码和文本混合起来，所有在 MCML 中的代码都是在运行时处理而不是预处理。

示例如下：

```
<script type="text/npl" src="somefile.lua" refresh="false">
<![CDATA[
globalVar = "test global var";
function OnButtonClick()
    _guihelper.MessageBox("refresh page now?", function()
```

```
        Page:SetValue("myBtn", "refreshed");
        Page:Refresh(0);
    end);
end
repeat_data = { {a=1}, {a=2} };
function GetDS()
    return repeat_data;
end
]]>
</script>
<div style='<%=format("width:%dpx", 100) %>'></div>
```

页面变量作用域

定义在行内脚本（inline script）中的全局变量或函数作用域只在定义其的页面内，在页面外不可见。所以如果你想要暴露某些值到外部环境或者简化你的代码，可以考虑在文件后台使用分离的代码（consider using a separate code behind file）。

有一个叫做 Page 的变量，我们能够使用它访问页面的沙盒环境。

页面刷新

对于一个动态页面，在 NPL 脚本中改变一些变量并且刷新页面来展现对应的变化是很常见的。<script refresh='false'>意思是这个部分的代码在页面刷新的时候不会重新解析。同时还需要注意全局变量或函数在一些脚本区域或多个页面刷新之间是共享的。

当窗口重新构建时，在行内脚本块中的有更改代码，通常会被重新解析。然而，如果你在模型后使用代码（use code behind model），npl 文件被缓存，你需要重启应用来产生影响。最好先在行内脚本中编写，当代码稳定再将它们移动到页面后的代码中。这样将使你在频繁的页面刷新或重建的过程中页面加载更快。

编写动态页面

MCML 有它自己的方式来编写动态页面。简略地说，mcml 使用频繁地使用全页面刷新来构建应答 UI 布局。请注意每个 mcml 页面刷新不会重构 DOM（Document Object Model）树实体。它仅仅重新使用旧的 DOM 树然后每次重新执行嵌入的代码。如果代码被 refresh="false"标记，它将会使用上一次计算的值。

通常 MCML v1 使用单一的传递（single）来渲染整个 web 页面。MCML v2 使用两

IdreamTech

个传递。

在 mcml 中，不建议用我们自己的代码去改变 DOM。我们建议你数据绑定你的代码（databind your code）并且使用条件组件（conditional component）来创建不同的代码路径，例如<pe:if>，当一切都设置好后，你用 Page:refresh()来刷新整个页面。这有一点像 AngularJS，但是我们在实时执行的行内代码中这么做，mcml 使用单一传递来渲染 mcml 标签并且执行相应代码将 DOM 转换成真实的 NPL 控件。

另一个事情是，在 mcml 中，我们需要手动地调用 Page:refresh(delayTimeSeconds)来刷新整个页面，例如当用户点击一个按钮来改变布局。请注意，改变 DOM 不会自动刷新页面，我们仍然需要调用刷新方法。

最后，在渲染之前，mcml 页面不会被预处理成一个第二 DOM 树（second DOM tree）。在 mcml 中的每个标签都是一个真实的 mcml 控件或页面元素（PageElement：https://github.com/NPLPackages/main/blob/master/script/ide/System/Windows/mcml/PageElement.lua），DOM 树通过页面刷新持续存在。这样使整个页面刷新 Page:refresh()比其他技术快得多。请注意，mcml v2 在页面刷新上甚至比 mcml v1 更快，这是因为在 mcml v2 中，NPL 控件都是纯脚本对象，而在 v1 中他们都是 ParaEngine 中的 ParaUIObject。前者在创建或销毁甚至在处决于每个页面元素实现的页面刷新中更加快。

MCML 控件

MCML 是意在替代整个应用中的冗长的控制而创造的，不是一个无状态和孤立的 web 页面。每个 mcml 页面是应用的一个不可分割的部分，他们运行在相同的线程中，并且能够利用线程中的一切资源。他们通常和 3D 场景和其他 UI 控件直接互动。可以用 Page:FindControl(name)获取下层控件（underlying control）。

MCML Page:Refresh(DelayTime)

Page:Refresh(DelayTime)在 DelayTime 秒后刷新整个页面。在延迟时间周期内，如果另一个用更长的延迟时间来调用这个函数，实际的页面刷新激活将被进一步延迟。当 MCML 页面包含例如 pe:name 等异步内容时，这个函数通常用一个设计模式来使用。无论一个异步标签在什么时候创建，它将首先检查用于显示的数据在那个时刻是否可用，如果可用，它

将之作为静态内容展示，如果不可用则会用回调函数来检索数据。在回调函数中，它将在延迟时间后调用与页面相关的刷新函数。因此，当上一个延迟时间达到后，页面将被重建，届时动态的内容将被访问。

- @param DelayTime:如果为 nil，它将默认为 self.DefaultRefreshDelayTime（通常是 1.5 秒）。
- tip:如果将之设为一个负数，这将会导致页面立即刷新。

页面样式和主题

MCML 在 style 和 class 参数中对行内（inline）css 支持有限制。MCML v1 和 v2 支持风格标签，我们可以以标准的 NPL 表的形式定义行内或基于文件的 css。语法不是 text/css 而是 text/mcss。它只是一个带有键名（key name）和值对（value pair）的表对象（table object）。键名可以是标签类名或者类名，不支持包含 tag，id，class 的复合名过滤。

以下是一个用不同类名定义样式的例子，所有的 mcml 标签能够有 0 或更多的类名。

```
<pe:mcml>
<style type="text/mcss" src="script/ide/System/test/test_file_style.mcss">
{
    color_red = { color = "#ff0000" },
    color_blue = { color = "#0000ff", ["margin-left"] = 10 },
    bold = { ["font-weight"]="bold"    }
}
</style>
<span class="color_blue bold">css class</span>
</pe:mcml>
```

在应用程序的整个生命周期内加载和缓存外部 css 文件。它的内容也是 NPL 表，例如 script/ide/System/test/test_file_style.mcss:

```
-- Testing mcml css: script/ide/System/test/test_file_style.mcss
{
["default"] = { color="#ffffff" },
["mobile_button"] = {
        background = "Texture/Aries/Creator/Mobile/blocks_UI_32bits.png;1 1 34 34:12 12 12 12",
},
color_red = { color = "#ff0000" },
color_blue = { color = "#0000ff", ["margin-left"] = 10, ["font-size"] = 16 },
```

```
bold = { ["font-weight"] = "bold"    },
["pe:button"] = {padding = 10, color="#00ff00"},
}
```

设置重点（Key Focus）

对于 mcml v1，我们能够通过以下代码设置重点，他仅适用于文本框。

```
local ctl = Page:FindControl(name);
if(ctl) then
   ctl:Focus();
end
```

对于输入文本框，同样能够使用：

```
<input type="text" autofocus="true"  ... />
```

但是在一个窗口中仅仅能够存在一个那样的控件。如果存在多个，最后一个将会获得关注。例如：

```
<pe:mcml>
    <div>
        Text: <input type="text" name="myAutoFocusEditBox" />
    </div>
    ... many other div here...
    <!-- This should be the last in the page -->
    <script type="text/npl" refresh="true">
        local ctl = Page("#myAutoFocusEditBox"):GetControl()
        ctl:Focus();
        ctl:SetCaretPosition(-1);
    </script>
</pe:mcml>
```

7 3D Programming

NPL/ParaEngine 提供了丰富的图形 API 和库去创造精致而复杂的，可交互的 3D 应用。ParaEngine 是个基于 NPLRuntime 的，功能全面的游戏引擎。它是用 C/C++编写的，但是他的所有 API 都可以用 NPL 脚本来使用。

7.1 3D Scene

在 3D 场景中，可以通过脚本 API 来创建 3D 对象。例如网格，物理网格，天空盒，摄像机，人物对象，光，粒子，容器，覆盖层，高度图地形，方块，动画，着色器，3D 资源等等。

要在 3D 场景中生成一个对象并且能对它操作，需要如下几个步骤：

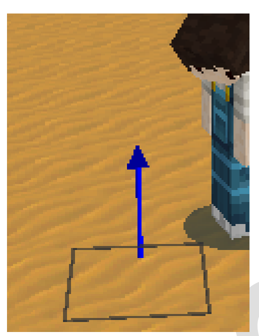
首先是资源加载，用了 ParaAsset 里面的 Load*（...）函数。

```
//加载资源，文件必须是.x 或者.fbx 文件
local asset = ParaAsset.LoadStaticMesh("","model/common/editor/z.x")
```

然后调用 ParaScene 的 Create*（...）方法，可以创建创建物体，并且返回该物体的实例，用这个实例可以更改这个对象的属性，调用相关方法来操作这个物体。

```
//生成对象
local obj = ParaScene.CreateMeshPhysicsObject("blueprint_center", asset,
1,1,1, false, "1,0,0,0,1,0,0,0,1,0,0,0");
//设置对象属性，
obj:SetPosition(ParaScene.GetPlayer():GetPosition());
obj:SetField("progress",1);
obj:GetEffectParamBlock():SetBoolean("ztest", false);
ParaScene.Attach(obj);
```

这段代码生成的对象：



生成人物对象和上面生成方式类似：

```
local player = ParaScene.CreateCharacter ("MyPlayer1",
ParaAsset.LoadParaX("","character/v3/Elf/Female/ElfFemale.x"), "", true,
0.35, 0, 1.0);
local x,y,z = ParaScene.GetPlayer():GetPosition()
player:SetPosition(x+1,y,z);
player:SetField("RenderOrder", 100)
player:GetEffectParamBlock():SetBoolean("ztest", false);
ParaScene.Attach(player);
```

人物对象可以播放动画：

三角形限制：

我们可以设置 3D 场景中每帧动画角色的最大总三角形数目，例如：

```
ParaScene.GetAttributeObject():SetField("MaxCharTriangles", 150000);
```

渲染管线：

当前所有的固定的放方法，着色器和延迟渲染分享了同一个预定好的渲染管线。渲染的顺序受 RenderImportance, RenderOrder,和 Shader files 性质的影响。一般的，这是一个在大多数情况下适用的固定通用目的渲染管线。

渲染管线的源代码在 `SceneObject.cpp` 的 `AdvanceScene()`方法中。对于每一个活动的视口，他会遍历场景中所有的物体，并且对每个可见物体调用 IPrepareRender()方法，将这些物体放入渲染队列中。

NPL 中有下列几种渲染管道：

- PIPELINE_3D_SCENE,渲染众多 3D 物体
- PIPELINE_UI，通过遍历 GUIRoot 渲染所有可见 GUI 对象
- PIPELINE_POST_UI_3D_SCENE ，只有 MiniSceneGraph 中 GetRenderPipelineOrder() == PIPELINE_POST_UI_3D_SCENE 的物体才会在这个渲染管道。
- PIPELINE_COLOR_PICKING

渲染定制：

RenderImportance 只影响在给定渲染队列中的排序，

RenderOrder 会影响这个物体呗放入那个渲染队列，也会影响他在这个队列中的排序。

当 RenderOrder 大于 100 的时候，会被放入 post render queue

7.2 地形

地形的相关操作被封装在 ParaTerrain 的命名域中，这里集合了所有对地形的相关操作方法。

首先是地形的创建，这个方法放在 ParaScene 命名域中，

```
static void CreateGlobalTerrain(float fRadius, int nDepth, const char* 
sHeightmapfile, float fTerrainSize, float fElevscale, int bSwapvertical, 
const char*  sMainTextureFile, const char* sCommonTextureFile, int 
nMaxBlockSize, float fDetailThreshold);
```

其中 fRadius 是地形的大小，nDepth 是四叉树结构的深度，sHeightmapfile 是高度图文件的路径，fTerrainSize 是在游戏中实际地形的大小，sMainTextureFile 纹理图文件的路径，nMaxBlockSize，LOD 中最大的 Block 数量，fDetailThreshold,LOD 中的细节阈值

例子：

```
ParaScene.CreateGlobalTerrain(2048, 7, "timg.png", 5.0, 15.0, 1, 
"testu.jpg", "testu.jpg", 64, 10.0)
1
```

获取地形内的相关信息，返回相关信息：

```
float GetElevation(float x, float y);
```

//获取 x,y 坐标的高度值/海拔值

```
ParaAssetObject GetTexture(float x, float y, int nIndex);
```

//通过索引获取纹理，

```
bool ReplaceTexture(float x, float y, int nIndex, ParaAssetObject& TextureAsset);
```

//更换纹理。更换对应索引的纹理。

```
void SaveTerrain(bool bHeightMap, bool bTextures);
```

//保存地形到硬盘，bHeightMap 为真则保存高度图，bTextures 为真则保存纹理图

Block 地形的 API：

//这个方法启动 block 地形，参数是摄像机的坐标

```
void EnterBlockWorld(float x,float y,float z);
```

//关闭 block 地形

```
void LeaveBlockWorld();
```

//rayX,rayY,rayZ 是起始位置

//dirX,dirY,dirZ,是方向

//当 length>fMaxDistance 表示没有碰撞发生

//函数返回选中的物体

```
object  Pick(float  rayX,float  rayY,float  rayZ,float  dirX,float  dirY,float dirZ,float fMaxDistance,const object& result,uint32_t filter=0xffffffff);
```

7.3 摄像机

ParaEngine 有给通用摄像机对象提供的内置 C++支持，叫做 AutoCamera。AutoCamera 是你在 3D 世界中默认使用的摄像机。

它能以各种方式配置，为你处理大多数的键盘/鼠标输入，同时控制它所聚焦的两足动物对象。

我们可以用以下的方式来获取或设置摄像机的属性。NPL Code Wiki 中的 Object Browser 可以查看它所有的属性。Camera 是 Scene object 的子类。

```
local attr = ParaCamera.GetAttributeObject()
```

用你的代码控制所有东西

大多数情况下，我们可以将摄像机聚焦到场景中一个两足动物玩家对象上，以此来直接控制玩家的移动。然而，写一个好用而流畅的玩家摄像机控制器是很难的，至少需要 3000 行代码，如果你确实想要自己控制所有东西，那就创造一个不可见的虚拟对象，然后把它设为摄像机聚焦的对象。然后把虚拟对象的 IsControlledExternally 属性设置为 true。

```
your_dummy_obj:SetField("IsControlledExternally", true)
```

至此之后，这个 AutoCamera 再也不会尝试去控制聚焦的虚拟对象了。完全由你来控制玩家对象，比如用 WASD 或方向键控制移动，用鼠标旋转朝向等。

代码示例

以下代码来自 ParaEngineExtension.lua

```
-- set the look at position of the camera. It uses an invisible avatar as the camera look at position.
-- after calling this function, please call ParaCamera.SetEyePos(facing,
```

IdreamTech

```
height, angle) to change the camera eye position.
function ParaCamera.SetLookAtPos(x, y, z)
	local player = ParaCamera.GetDummyObject();
	player:SetPosition(x, y - 0.35, z);
	player:ToCharacter():SetFocus();
end

function ParaCamera.GetLookAtPos()
	return unpack(ParaCamera.GetAttributeObject():GetField("Lookat
position", {0,0,0}));
end
-- it returns polar coordinate system.
-- @return camobjDist, LifeupAngle, CameraRotY
function ParaCamera.GetEyePos()
	local att = ParaCamera.GetAttributeObject();
	return att:GetField("CameraObjectDistance", 0),
att:GetField("CameraLiftupAngle", 0), att:GetField("CameraRotY", 0);
end

-- create/get the dummy camera object for the camera look position.
function ParaCamera.GetDummyObject()
	local player = ParaScene.GetObject("invisible camera");
	if(player:IsValid() == false) then
		player = ParaScene.CreateCharacter ("invisible camera",
"", "", true, 0, 0, 0);
		--player:GetAttributeObject():SetField("SentientField",
0);--senses nobody
		player:GetAttributeObject():SetField("SentientField",
65535);--senses everybody
		--player:SetAlwaysSentient(true);--senses everybody
		player:SetDensity(0); -- make it flow in the air
		player:SetPhysicsHeight(0);
		player:SetPhysicsRadius(0);
		player:SetField("SkipRender", true);
		player:SetField("SkipPicking", true);
		ParaScene.Attach(player);
		player:SetPosition(0, 0, 0);
	end
	return player;
end

-- set the camera eye position by camera object distance, life up angle and
rotation around the y axis. One must call ParaCamera.SetLookAtPos() before
calling this function.
```

```
-- e.g.ParaCamera.SetEyePos(5, 1.3, 0.4);
function ParaCamera.SetEyePos(camobjDist, LifeupAngle, CameraRotY)
	local att = ParaCamera.GetAttributeObject();
	att:SetField("CameraObjectDistance", camobjDist);
	att:SetField("CameraLiftupAngle", LifeupAngle);
	att:SetField("CameraRotY", CameraRotY);
```

7.4 3D 文件格式

ParaEngine 支持以下几种自定义或标准的文件格式：

- ParaX 格式(*.x)：这是用于预定义动画了的人物，粒子等对象的内置格式。有需要的情况可以下载安装针对于 3dsmax9(32 位/64 位)的 ParaEngine 输出插件。然而并不支持 3dsmax 的最新版本。相关使用时请考虑 fbx 格式。
- FBX 格式(*.fbx)：FBX 是 AutoDesk 公司的 MAYA/3dsmax 的通用文件格式，它也是工业上得到支持最多的文件格式之一。目前 ParaEngine 只支持 2013 版本及以下的 FBX 文件，2014，2015 版还在测试中。
- BMAX (*.bmax):这个是内置的方块 max 文件格式，被 Paracraft 用来存储静态和动画方块。
- STL format (*.stl)：用于 3D 打印的文件格式

7.4.1 通过 NPL 脚本读取 ParaX 文件内容

ParaXModelAttr 可以被用来获取关于 ParaX 模型的资源文件，用来导出纹理，顶点，矩阵，骨骼等。它有两种对应的 API，一种是 cdata API，另一种是脚本 API。

- cdata API 的方法名格式是 xxxCData：例如 GetObjectNumCData,它能参考内部 C++ParaXModel 来使用相同的数据而不产生任何的内存占用。因此提供了一种在脚本环境下不需要任何内存占用就能读写大量 C++数据的方法。而且由于 luajit ffi，参考 cdata 部分的效率很接近原生 C++的效率。
- 脚本 API 方法名是 xxx：例如 ParaXModelAttr:GetVertices() 会复制 cdata 到标准的 lua 表并缓存结果，但这个过程会比较慢并且需求大量的内存占用。

以下是几种文件加载的代码示例：

IdreamTech

■ Table 1 从 cdata 加载到 NPL 的 table 对象

```
NPL.load("(gl)script/ide/System/Scene/Assets/ParaXModelAttr.lua");
local ParaXModelAttr =
commonlib.gettable("System.Scene.Assets.ParaXModelAttr");
local attr = ParaXModelAttr:new():initFromPlayer(ParaScene.GetPlayer())
echo(attr:GetObjectNum());
echo(attr:GetRenderPasses());
echo(attr:GetGeosets());
for i=1, attr:GetObjectNum().nTextures do
	echo({texture = attr:GetTextureName(i-1)});
end
for i=1, attr:GetObjectNum().nBones do
	local bone = attr:GetBone(i-1);
	echo({bone_name = bone:GetField("name", ""), PivotPoint =
bone:GetField("PivotPoint", {}), ParentIndex = bone:GetField("ParentIndex",
-1) });
end
echo(attr:GetVertices());
echo(attr:GetIndices());
```

■ Table 2 从给定的*.x 文件或*.fbx 文件加载数据

```
NPL.load("(gl)script/ide/System/Scene/Assets/ParaXModelAttr.lua");
local ParaXModelAttr =
commonlib.gettable("System.Scene.Assets.ParaXModelAttr");
local attr =
ParaXModelAttr:new():initFromAssetFile("character/bmax/test_multianim.fbx",
function(attr)
	attr:DrawStaticAsText()
end)
```

■ Table 3 不在脚本环境下占用内存分配而直接使用 cdata

```
NPL.load("(gl)script/ide/System/Scene/Assets/ParaXModelAttr.lua");
local ParaXModelAttr =
commonlib.gettable("System.Scene.Assets.ParaXModelAttr");
local attr =
ParaXModelAttr:new():initFromAssetFile("character/bmax/test_multianim.fbx",
function(attr)
	local vertices = attr:GetVerticesCData();
	local nCount = attr:GetObjectNumCData().nVertices;
	for i=0, nCount-1 do
		echo({vertices[i].pos.x, vertices[i].pos.y,
vertices[i].pos.z})
	end
end)
```

7.4.2 BMax 文件格式

- BMax 即 Block Max,是一种仅用于 Paracraft 来保存和交换方块数据的文件格式. 在 Paracraft 里，世界是由方块组成的，每一个方块都包含以下四种数据：
- **x,y,z** 方块位置坐标
- **block_id** 方块的种类 Id,在 block type。Xml 里可以查看完整的方块种类 ID 表
- **block_data** 方块的数据（32 位），通常用来指示方块种类或基本信息 (orientation)
- **custom_data** 可以是任何保存着附加的方块数据 NPL table 对象。大多数静态的方块都不包含 custom_data。然而，像电影方块，命令方块都会在这个地方 (custom_data)里保存动画数据或命令。

使用者可以在 Paracraft 里选择几个方块然后把他们保存为 bmax 文件来进行对照, 下面是 bmax 文件的示例(包括一个按钮，2 个电影方块，一个中继器和一条线)：

```
<pe:blocktemplate>
        <pe:blocks>{
{-2,0,-3,105,5,},

{-2,0,-2,228,[6]={{name="cmd","/t 6
/end",},{{"{timeseries={lookat_z={times={0,5348,},data={20001.56125,20004.6
5625,},ranges={{1,2,},},type=\"Linear\",name=\"lookat_z\",},eye_liftup={tim
es={0,5348,},data={0.28568,0.26068,},ranges={{1,2,},},type=\"Linear\",name=
\"eye_liftup\",},lookat_x={times={0,5348,},data={20000.51959,20000.58203,},
ranges={{1,2,},},type=\"Linear\",name=\"lookat_x\",},eye_rot_y={times={0,53
48,},data={-3.11606,-
3.11668,},ranges={{1,2,},},type=\"LinearAngle\",name=\"eye_rot_y\",},is_fps
={times={0,5348,},data={0,0,},ranges={{1,2,},},type=\"Discrete\",name=\"is_
fps\",},lookat_y={times={0,5348,},data={-127.08333,-
127.08333,},ranges={{1,2,},},type=\"Linear\",name=\"lookat_y\",},eye_dist={
times={0,5348,},data={8,8,},ranges={{1,2,},},type=\"Linear\",name=\"eye_dis
t\",},has_collision={times={0,5348,},data={1,1,},ranges={{1,2,},},type=\"Di
screte\",name=\"has_collision\",},eye_roll={times={0,5348,},data={0,0,},ran
ges={{1,2,},},type=\"LinearAngle\",name=\"eye_roll\",},},}",name="slot",att
r={count=1,id=10061,},},{"{timeseries={time={times={},data={},ranges={},typ
```

```
e=\"Linear\",name=\"time\",},music={times={},data={},ranges={},type=\"Discrete\",name=\"music\",},tip={times={},data={},ranges={},type=\"Discrete\",name=\"tip\",},movieblock={times={},data={},ranges={},type=\"Discrete\",name=\"movieblock\",},cmd={times={},data={},ranges={},type=\"Discrete\",name=\"cmd\",},blocks={times={},data={},ranges={},type=\"Discrete\",name=\"blocks\",},text={times={},data={},ranges={},type=\"Discrete\",name=\"text\",},},}",name="slot",attr={count=1,id=10063,},},name="inventory",{"{timeseries={blockinhand={times={},data={},ranges={},type=\"Discrete\",name=\"blockinhand\",},x={times={0,},data={20000.51959,},ranges={{1,1,},},type=\"Linear\",name=\"x\",},pitch={times={},data={},ranges={},type=\"LinearAngle\",name=\"pitch\",},y={times={0,},data={-
127.08333,},ranges={{1,1,},},type=\"Linear\",name=\"y\",},parent={times={},data={},ranges={},type=\"LinearTable\",name=\"parent\",},roll={times={},data={},ranges={},type=\"LinearAngle\",name=\"roll\",},block={times={},data={},ranges={},type=\"Discrete\",name=\"block\",},scaling={times={},data={},ranges={},type=\"Linear\",name=\"scaling\",},gravity={times={},data={},ranges={},type=\"Discrete\",name=\"gravity\",},HeadUpdownAngle={times={},data={},ranges={},type=\"Linear\",name=\"HeadUpdownAngle\",},anim={times={},data={},ranges={},type=\"Discrete\",name=\"anim\",},bones={R_Forearm_rot={times={0,34,1836,},data={{0.00024,0.00175,-0.01261,0.99992,},{0.00024,0.00175,-
0.01261,0.99992,},{0.00022,-
0.05946,0.42959,0.90107,},},ranges={{1,3,},},type=\"Discrete\",name=\"R_Forearm_rot\",},R_UpperArm_rot={times={0,34,1836,},data={{-0.01933,-
0.00286,0.03036,0.99935,},{-0.01802,-0.03834,0.32796,0.94375,},{-0.0137,-
0.01077,0.14324,0.98954,},},ranges={{1,3,},},type=\"Discrete\",name=\"R_UpperArm_rot\",},isContainer=true,},speedscale={times={},data={},ranges={},type=\"Discrete\",name=\"speedscale\",},assetfile={times={0,},data={\"actor\",},ranges={{1,1,},},type=\"Discrete\",name=\"assetfile\",},skin={times={},data={},ranges={},type=\"Discrete\",name=\"skin\",},z={times={0,},data={20001.56125,},ranges={{1,1,},},type=\"Linear\",name=\"z\",},facing={times={},data={},ranges={},type=\"LinearAngle\",name=\"facing\",},HeadTurningAngle={times={},data={},ranges={},type=\"Linear\",name=\"HeadTurningAngle\",},name={times={0,},data={\"actor3\",},ranges={{1,1,},},type=\"Discrete\",name=\"name\",},opacity={times={},data={},ranges={},type=\"Linear\",name=\"opacity\",},},tooltip=\"actor3\",}",name="slot",attr={count=1,id=10062,},},},name="entity",attr={bz=19201,bx=19200,class="EntityMovieClip",item_id=228,by=5,},},}
,

{-2,0,-1,197,3,},
{-2,0,0,189,},
{-2,0,1,228,[6]={{name="cmd","/t 30
/end",},{name="inventory",{name="slot",attr={count=1,id=10061,},},},name="entity",attr={bz=19204,bx=19200,class="EntityMovieClip",item_id=228,by=5,},}
,},}
```

```
        </pe:blocks>
</pe:blocktemplate>
```

7.4.3 ParX 文件格式

ParaX 是 NPL Runtime 所特有的用于带动画(animated)3D 人物资源的二进制文件格式。ParaX 就像 FBX 文件的简化版本(FBX 就像 autodesk 3dsmax/maya 版的 bmax 文件)。NPL Runtime 也支持读取 FBX 文件并在 C++环境下作为 ParaXModel 对象加载。FBX 的具体使用请见：

https://github.com/LiXizhi/ParaCraftSDK/wiki/FBXModel

- C++里的 BMAX 到 ParaX 转换（缺少对电影方块里的动画数据的支持）：
  https://github.com/LiXizhi/NPLRuntime/tree/master/Client/trunk/ParaEngineClient/BMaxModel
- ParaX 模型:
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/ParaXModel/XFileCharModelParser.h
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/3dengine/ParaXSerializer.h
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/3dengine/ParaXSerializer.h

BMAX 电影方块到 ParaX 文件格式的转换细节

- 把 BMAX 文件中的所有电影方块都定位，忽视其他的所有方块。电影方块在 3D 空间中需要处于相同的序列(order)
- 第一个电影方块的动画 ID 总是设置为 0 (空闲状态/常态 动画)
- 第二个电影方块的动画 ID 总设置为(走动动画)
- 第三个及以后的电影方块的动画 ID 可以被他们自己的电影方块指令设定
- 第一个电影方块的 BMAX 模型是用来定义骨骼和网格(顶点等)

https://github.com/LiXizhi/NPLRuntime/tree/master/Client/trunk/ParaEngineClient/BMaxModel 这个网址里的代码说明了如何把方块信息转换成骨骼，顶点和子网格。其他所有电影方块里的 BMAX 模型都是被忽略的

- 电影方块里的动画数据是用来生成 ParaXModel 中对应于每个动画 ID 的骨骼动画的

7.4.4 ParaX 文件 LOD

在 ParaEngine 中，LOD 的确立主要依赖于两个部分：

1. ParaMeshXMLFile：这个类的作用是在加载 ParaX mesh 或静态网格 Static mesh 文件时起参考作用，包含了 LOD 文件相关的结构和属性定义(LOD 信息，shader 信息)
2. ContentLoader：这个类的核心作用是被用于解析文件并向上层提供数据，为 MeshEntity 和 ParaXEntity 所引用，但是在这个过程中涵盖了 LOD 的创建，分层等功能。

此外，LOD 的创建有时需要依赖 XML 文件，对于 XML 文件的作用，NPLProject.com 给出如下解释：

XML 文件（LOD 网格 XML 文件）是用来把各个.x 模型文件聚合分组。这样当摄像机离观察物体的距离在一个特殊范围内的时候，我们就可以使用定义在 XML 文件里的一个对应的.x 文件。也就是说，LOD 网格 XML 文件就是一种涵括多个.x 文件集合的元文件。当物体在一个特定的半径距离下时它会指定对应情况下该使用哪个文件。

为了生成网格 LOD 文件，要经过以下步骤：

- 把所有的 LOD 网格的模型文件，纹理都放在同一目录下，每个文件命名为 objectName_LOD10.x, objectName_LOD20.x. The number in the trailing

"_LOD[number]" 末尾的数字的意思是表示在哪种距离情况（单位米）下这种文件应该被使用

- 在同一个目录下创建一个叫做 objectName.xml 的 XML 文件

例：假设在目录下有 chat.dds, char_LOD5.x, char_LOD10.x, char_LOD30.x. XML 文件就应该命名为 char.xml，然后 LOD 的设置就可以从文件名检索到。

NOTE:所有的 LOD 对于所有动画都必须保证有同样的动画长度。因为游戏引擎会对它所有的 LOD 都使用 LOD0 的动画帧数。

The format of xml file

```
<?xml version="1.0" encoding="utf-8"?>
<mesh version="1" type="0">
        <boundingbox minx="-0.336675" miny="0.000000" minz="-0.366781"
maxx="0.312267" maxy="1.247474" maxz="0.335574"/>
        <submesh loddist="5" filename="ElfFemale_LOD5.x"/>
        <submesh loddist="15" filename="ElfFemale_LOD15.x"/>
        <submesh loddist="30" filename="ElfFemale_LOD30.x"/>
</mesh>
```

默认情况下，第一个 LOD 包含所有的三角面，第二个 LOD 的三角面不会超过 2000 个，第三个 LOD 三角面不会超过 500 个。对于三角面少于 500 个的网格，不需要任何 LOD。2000 个三角面的网格，2 个 LOD 就够了。三角面多于 4000 的网格，就需要 3 个 LOD 了。但实际上也可以完全由开发者来决定一个模型到底需要多少 LOD。

7.4.5 3D 世界文件格式

ParaEngine/NPL 能自动保存或加载 3D 场景对象到硬盘文件

要想创建一个空世界，可以使用如下代码：

```
local worldpath = "temp/clientworld";
```

```
ParaIO.DeleteFile(worldpath.."/");

ParaIO.CreateDirectory(worldpath.."/");

ParaWorld.NewEmptyWorld(worldpath, 533.3333, 64);
```

3D 世界如同平铺的瓦片。每一片瓦片(tile)通常都是 512*512 米，但这里默认是 533.3333。

在执行上述代码后，你会看到如下文件：

temp/clientworld/worldconfig.txt 3d 世界的进入文件，它的内容如下:

```
-- Auto generated by ParaEngine
type = lattice
TileSize = 533.333313
(0,0) = temp/clientworld/flat.txt
(0,1) = temp/clientworld/flat.txt
...
```

它包含了从瓦片 tile(x,y)到瓦片配置文件的映射。

temp/clientworld/flat.txt 单个地形瓦片 tile 的配置文件，内容如下:

```
-- auto gen by ParaEngine
Heightmapfile = temp/clientworld/flat.raw
MainTextureFile = terrain/data/MainTexture.dds
CommonTextureFile = terrain/data/CommonTexture.dds
Size = 533.333313
ElevScale = 1.0
Swapvertical = 1
HighResRadius = 30
DetailThreshold = 50.000000
MaxBlockSize = 64
DetailTextureMatrixSize = 64
NumOfDetailTextures = 0
```

7.4.6 全局地形格式 Global Terrain Format

Global Terrain Format 是基于地形引擎 terrain engine 的 LOD 三角瓦片 tile。合法的 tile 尺寸被定义在 3D 世界配置文件中，例如前面说的 533.3333.然后，它的实际尺寸总是 512*512 并保存为***.raw** 文件。

7.4.7 方块世界文件格式

ParaEnging 有一个内置的方块引擎来渲染 3D 方块世界。方块世界也是瓦片化的(tiled)，格式如同全局地形格式 Global Terrain Format，合法的 tile 尺寸被定义在 3D 世界配置文件中，例如 533.3333，然而它的实际尺寸总是 512*512，保存为二进制的方块区域 Block region 文件到*.raw 中。但方块世界的方块区域 Block region 文件*.raw 和 Global Terrain 的*.raw 文件是不同，后者只是单纯的高度示图(height maps)。

当前的 ParaEngine 中，每个方块都包括 **x,y,z, block_id, block_data[, custom_data],** 但 **custom_data** 不被方块区域 Block region*.raw 文件支持的。方块区域 Block region 文件是被 increase-by-integer 算法编码压缩的

细节代码在 BlockRegion.cpp 中可见。

7.5 Mini Scenegraph

Mini scene graph 是被用来在场景中展示 可交互的 3D helper 对象，并且一般情况下只希望在可视的 mini graph node 节点上添加少量的对象，例如一些 旋转提示标志，物体网格上的依附点或者一些其他的 helper 3D handles。这些对象都不是永久存在的，也就是说，mini scene graph 常用来显示暂时存在的 3D 对象

Mini scene graph 不会对它存储的对象进行空间划分，而是使用父子类继承关系来联系各个对象成分。大多数的场景对象都被 mine scene graph 依附，开发者可以展示/隐藏，添加/删除这些 mini scene graph，如同对 2D 对象操作一样。Mini scene graph 里的内容会在主场景渲染完之后再进行渲染工作。

几个 mini scene graph 可被用到的情况：

- 简单的雨雪效果：在当前摄像机的位置每隔几秒生成新的粒子系统，并删除之前的粒子系统。这个可借助脚本借口或游戏引擎本身来实现。
- 在网格对象上显示最外层边界点/顶峰点或依附点。例如当一个游戏人物靠近一个网格

IdreamTech

的边界点/依附点时，就会展示一个 3D 图标来提示可以在此产生互动。

- 显示一些特殊的 3D 光标或指针。例如鼠标点击，移动或拖拉 3D 对象时，相关目标的响应状态。

7.6 方块引擎

方块引擎管理着世界中所有的方块，最主要的类是 BlockWorld，是单例模式。

BlockWorld

一个 BlockWorld 对象代表一个方块世界，它最多有 32768 x 32768 x 256 个方块，其中 256 是高度。BlockWorld 会动态异步地即时加载 BlockRegion（方块区域）

获取 BlockWorld 对象：

```
local world = ParaBlockWorld.GetWorld(nil)
```

获取方块世界属性对象：

```
local attr = ParaBlockWorld.GetBlockAttributeObject(world)
```

可以用 attr:SetField()方法来设置方块世界的属性。

将方块 A（1,1,1）的方块模板复制给方块 B（2,2,2）：

```
local id = ParaBlockWorld.GetBlockId(world, 1, 1, 1)
ParaBlockWorld.SetBlockId(world, 2, 2, 2, id)
```

BlockRegion

它管理着 512 x 512 x 256 个方块，保存在一个单一文件中。

加载及卸载方块（x, y, z）所在区域(其中 x, y, z 是方块的世界坐标)：

```
local world = ParaBlockWorld.GetWorld(nil)
ParaBlockWorld.LoadRegion(world, x, y, z)
ParaBlockWorld.UnloadRegion(world, x, y, z)
```

BlockChunk

它保存 16 x 16 x 16 大小的区域的方块模型和光线信息。每个 Chunk 会被转换并添加到 BlockRenderTask 的一个队列中，用于排序和渲染。

BlockLightGrid

IdreamTech

它在独立线程计算太阳和方块的光线，并把结果保存在 BlockChunk 中，用于渲染。

BlockModel

它通常是立方体型的 3D 模型，但它不是直接用来渲染的 3D 对象。它通常被 BlockTemplate 用来提供渲染和物理信息。

7.7 像素拾取

BufferPicking 类允许程序员将 3D 对象渲染到一个 2D 缓存（纹理）中，然后从缓存中读取像素值。这项技术常常用在 3D 场景中的像素精度鼠标拾取。

有两种预先定义好的缓存：

- "backbuffer"：这是屏幕上显示画面的缓存
- "overlay"：这是在 3D 场景之上的对象的一个附加渲染通道

用户同时也可以创造自定义的拾取缓存

从"backbuffer"拾取

如果只需要从屏幕显示的画面拾取像素颜色，可以利用下面的代码：

```
NPL.load("(gl)script/ide/System/Scene/BufferPicking.lua");
local BufferPicking = commonlib.gettable("System.Scene.BufferPicking");
local result = BufferPicking:Pick(nil, nil, 2, 2);
echo(result);
echo({System.Core.Color.DWORD_TO_RGBA(result[1] or 0)});
```

从"overlay"对象拾取

如果要从 overlay 对象拾取，需要用到 OverlayPicking.lua，OverlayPicking 是 BufferPicking 的子类，所以使用方法与上面的 BufferPicking 类似。ParaCraft 中所有的操纵器都是用 Overlay 对象渲染的，可以通过 OverlayPicking 拾取。

从自定义缓存拾取

程序员可以创造自定义缓存来渲染任何 3D 对象。

```
NPL.load("(gl)script/ide/System/Scene/BufferPicking.lua");
local MyPickBuffer =
commonlib.inherit(commonlib.gettable("System.Scene.BufferPicking"),
commonlib.gettable("Tests.MyPickBuffer"));
MyPickBuffer:Property("Name", "MyCustomBuffer");
-- only called when Pick function is called
```

```
function MyPickBuffer:paintEvent(painter)
        self:SetColorAndName(painter, "#ff0000");
        painter:PushMatrix();
        local obj = ParaScene.GetPlayer();
        local x, y, z = obj:GetPosition();
        local vOrigin = self:GetRenderOrigin();
        x, y, z = x - vOrigin[1], y - vOrigin[2], z - vOrigin[3];
        painter:TranslateMatrix(x,y,z);
        painter:DrawSceneObject(obj, 0)
        painter:PopMatrix();
end
MyPickBuffer:InitSingleton();

-- for debugging purposes, we will show the picking buffer into the gui.
MyPickBuffer:DebugShow("_lt", 10, 10, 128, 128)

-- test picking here
commonlib.Timer:new({callbackFunc = function(timer)
        -- always redraw (force paintEvent to be invoked)
        MyPickBuffer:SetDirty(true);
        -- pick at the current mouse position
        echo(MyPickBuffer:Pick(nil, nil, 2, 2));
end}):Change(0, 1000)
```

注意上面 paintEvent 中的这两行代码：

```
local vOrigin = self:GetRenderOrigin();
x, y, z = x - vOrigin[1], y - vOrigin[2], z - vOrigin[3];
```

世界坐标有双重精度，并且通常离摄像机原点很远。渲染 3D 对象的时候，对象的世界坐标必须减去这个向量，才能让对象在进行坐标转换之前，能更靠近摄像机。

8 Web 编程

在 NPL 中你不需要创建任何的套接字或为了和远程的电脑通信而写消息循环。所有的通信都很有效的被 NPL 管理。用户只需要一个函数来建立连接、认证并于远程电脑通信。

8.1 Networking

在 NPL 中，我们不需要创建任何套接字（socket）或者写任何消息循环来和远程计算机通讯。所有的通讯都被 NPL 非常高效地管理起来。用户仅仅只需要单个函数来建立链接，认证还有和远程计算机通讯。

初始化网络接口

在默认情况下，NPL 启动时不会监听任何端口，除非你调用一些库，例如 WebServer。在 NPL 中开启网络，我们需要调用：

```
-- a server that listen on 8080 for all IP addresses
NPL.StartNetServer("0.0.0.0", 8080);
```

客户端同样需要调用 NPL.StartNetServer。

```
-- a client that does not listen on any port. Just start the network
interface.
NPL.StartNetServer("127.0.0.1", "0");
```

服务器参数

以下是 WebServer 的配置文件。你可以根据这个配置文件对 NPL 中配置的内容有一个大致的了解。

```
    <!--HTTP server related-->
    <table name='NPLRuntime'>
      <!--whether to use compression for incoming connections. This must be
true in order for CompressionLevel and CompressionThreshold to take effect-
->
      <bool name='compress_incoming'>true</bool>
      <!---1, 0-9: Set the zlib compression level to use in case
compresssion is enabled.
      Compression level is an integer in the range of -1 to 9.
                     Lower compression levels result in faster execution, but
less compression. Higher levels result in greater compression,
                     but slower execution. The zlib constant -1, provides a
good compromise between compression and speed and is equivalent to level
6.-->
      <number name='CompressionLevel'>-1</number>
      <!--when the NPL message size is bigger than this number of bytes, we
will use m_nCompressionLevel for compression.
                     For message smaller than the threshold, we will not
compress even m_nCompressionLevel is not 0.-->
      <number name='CompressionThreshold'>204800</number>
      <!--if plain text http content is requested, we will compress it with
gzip when its size is over this number of bytes.-->
      <number name='HTTPCompressionThreshold'>12000</number>
      <!--the default npl queue size for each npl thread. defaults to 500.
may set to something like 5000 for busy servers-->
      <number name='npl_queue_size'>20000</number>
      <!--whether socket's SO_KEEPALIVE is enabled.-->
```

```
        <bool name='TCPKeepAlive'>true</bool>
        <!--enable application level keep alive. we will use a global idle timer to detect if a connection has been inactive for IdleTimeoutPeriod-->
        <bool name='KeepAlive'>false</bool>
        <!--Enable idle timeout. This is the application level timeout setting.-->
        <bool name='IdleTimeout'>false</bool>
        <!--how many milliseconds of inactivity to assume this connection should be timed out. if 0 it is never timed out.-->
        <number name='IdleTimeoutPeriod'>1200000</number>
        <!--queue size of pending socket acceptor-->
        <number name='MaxPendingConnections'>1000</number>
    </table>
```

请注意，IdleTimeout 在客户端和服务器端都应该设置，因为超时（timeout）在两端都可能发生。以编程的方式设置参数，见以下示例：

```
        local att = NPL.GetAttributeObject();
        att:SetField("TCPKeepAlive", true);
        att:SetField("KeepAlive", false);
        att:SetField("IdleTimeout", false);
        att:SetField("IdleTimeoutPeriod", 1200000);
        NPL.SetUseCompression(true, true);
        att:SetField("CompressionLevel", -1);
        att:SetField("CompressionThreshold", 1024*16);
        -- npl message queue size is set to really large
        __rts__:SetMsgQueueSize(5000);
```

TCP 保持活跃（TCP Keep Alive）

如果你想要在就算超过 20 秒没有消息发送的情况下，仍然维持已认证的链接，你可能想到生成一些定期的 ping 消息。因为不这样，不管是服务器端还是客户端肯呢个认为 TCP 链接已经失效，然后你将丢失在服务器端的已认证会话（session）。

尽管 NPL 允许你在全局范围使用 KeepAlive 属性来配置服务器端的应用级别的 ping 消息，但是不建议在服务器端启用它，或者将它的值设置为一个非常大的值（例如几分钟）。建议在客户端对所有远程进程定期地初始化 ping 消息。

对于一个强客户端应用（robust client application），客户端同样也需要处理链接丢失和恢复或者在应用级别上重新认证。

8.2 WebServer

NPL Web Server

NPL Runtime 对于编写 Web server 应用提供一个类似于 PHP 的内置的自包含的框架，而且提供类似于 Node.js 的异步 API。

为什么使用 NPL 编写 Web 服务器

- **将异步代码转变为同步代码**

  许多在服务端的任务有异步 API 如数据库运行，远程程序调用，背景任务，计时器等。异步 API 从 IO 范围或持续运行程序释放运行的线程。使用异步 API（像 NodeJS 中的一样）使得 web 服务器运行更快，但是编写起来比较复杂，因为存在很多函数调用会破坏别的序列及拒绝编写的程序的代码。

  在 NPL 页面文件中，任何的异步 API 可以通过 `resume/yield` 被用作同步 API，于是建立一个网站页面就像用 PHP 编写一个文件。在内部，它依然是异步 API 并且每秒可以处理很多请求尽管这需要花很长的时间。下面是一个例子：

```
<?
local value = "Not Set"
-- any async function with callback
local mytimer = commonlib.Timer:new({callbackFunc =
function(timer)
   value = "Set";
   resume();
end})
-- start the timer after 1000 milliseconds
mytimer:Change(1000, nil)

-- async wait when job is done
yield();
assert(value == "Set");
```

  请注意，resume/yield 不是包含锁和信号的多线程程序。它是一个特殊的内部协程。代码是完全无锁的且线程是不等待而且处理其他 URL 请求的。

- **自包含客户/服务端解决方案**

  NPL Web Server 运行迅速且可以在所有平台（与外部甚至数据库都没有关联的平台）都可以轻易地配置的。你可以拥有一个十分复杂的 3D 客户端应用，它既可以是客户端也可以是服务端，如 Paracraft

我们相信，每个软件，不论是运行在中心服务端或者端点如家庭或移动设备，都需要可以通过 TCP 连接和/或 HTTP（web server）提供服务。在我们很多应用中，一个单独的软件既作为服务端有作为客户端。由于那些 web 框架的多线程（多进程）性质，很少一部分已存在的编程语言可以提供一种简易的方法或架构，去分享客户端和服务端代码或一个无锁模型中的变量。此外，一个服务端应用常常很难去配置，这使得它很难在家用电脑或移动设备上下载。

NPL 是一种解决这些问题并提供丰富的客户端 API 以及 可以在为你的客户端和服务端代码分享全部 runtime 环境时处理与专业 web 服务器一样多的请求的 web 服务端框架语言。

NPL Web Server 的编程模型

NPL Runtime 为了编写 web 服务端应用提供了一种与 PHP 类似的内置的框架

NPL Web Server 推荐像 Nodejs 一样的异步编程模型，但是又不丢掉像 PHP 一样简洁的同步混合代码的方法。

下面是一个 helloworld.page 服务页面的例子：

	query database and wait for database result	MVC Render
duration	95%	5%

一个页面的处理通常包含两个阶段：

- 一个是从数据库引擎获取数据，这花费超过总时间的 95%
- 另一个是页面渲染，处理器受限的且只花费全部请求时间的 5%

当使用 NPL yield 方法时，它允许同时处理其他网络请求在 90%的时间间隔，同时等待数据库作用在同一个系统级的线程。下面代码可以看出将异步代码和基于模板的页面渲染代码混合在一起有多么容易。它允许我们在单个线程中同时提供每秒 5000 条请求，即使每个请求从数据库取回需要 30ms。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html>
<head>
  <title>NPL server page test. </title>
</head>
<body>
<p>
    <?npl
            echo "Hi, I'm a NPL script!";
    ?>
</p>

<?
-- connect to TableDatabase (a NoSQL db engine written in NPL)
db = TableDatabase:new():connect("database/npl/", function() end);

-- insert 5 records to database asynchronously.
local finishedCount = 0;
for i=1, 5 do
    db.TestUser:insertOne({name=("user"..i), password="1"},
function(err, data)
            finishedCount = finishedCount + 1;
            if(finishedCount == 5) then
                    resume();
            end
    end);
end
yield(); -- async wait when job is done

-- fetch all users from database asynchronously.
db.TestUser:find({}, function(err, users)  resume(err, users); end);
err, users = yield(); -- async wait when job is done
?>

<?npl for i, user in ipairs(users) do ?>
    i = <?=i?>, name=<? echo(user.name) ?> <br/>
<?npl end ?>

<p>
    1.  <?npl echo 'if you want to serve NPL code in XHTML or XML
documents, use these tags'; ?>
</p>
<p>
    2.  <? echo 'this code is within short tags'; ?>
```

```
    Code within these tags <?= 'some text' ?> is a shortcut for this
code <? echo 'some text' ?>
</p>
<p>
    3.  <% echo 'You may optionally use ASP-style tags'; %>
</p>

<% nplinfo(); %>

<% print("<p>filename: %s, dirname: %s</p>", __FILE__,
dirname(__FILE__)); %>

<%
some_global_var = "this is global variable";

-- include file in same directory
include("test_include.page");
-- include file relative to web root directory. however this will not
print, because we use include_once, and the file is already included
before.
include_once("/test_include.page");
-- include file using disk file path
local result = include(dirname(__FILE__).."test_include.page");

-- we can also get the result from included file
echo(result);
%>

</body>
</html>

<%
--assert(false, "uncomment to test runtime error");

function test_exit()
    exit();
end
test_exit();
assert(false, "can not reach here");
%>
```

在上述代码中引用的 test_include.page 内容是：

```
<?npl
```

```
echo("<p>this is from included file: "..(some_global_var or
"").."</p>");

-- can also has return value.
return "<p>from test_include.page</p>";
```

当在一个 web 浏览器例如 `http://localhost:8099/helloworld` 打开时，上述服务页面的输出：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>NPL server page test. </title>
</head>
<body>
<p>
     Hi, I'm a NPL script!</p>

     i = 1, name=user1 <br/>
     i = 2, name=user2 <br/>
     i = 3, name=user3 <br/>
     i = 4, name=user4 <br/>
     i = 5, name=user5 <br/>
<p>
     1.  if you want to serve NPL code in XHTML or XML documents, use
these tags</p>
<p>
     2.  this code is within short tags    Code within these tags some
text is a shortcut for this code some text</p>
<p>
     3.  You may optionally use ASP-style tags</p>

<p>NPL web server v1.0</p><p>site url:
http://localhost:8099/</p><p>your ip: 127.0.0.1</p><p>{
<br/>["Host"]="localhost:8099",
<br/>["rcode"]=0,
<br/>["Connection"]="keep-alive",
<br/>["Accept"]="text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,*/*;q=0.8",
<br/>["Accept-Encoding"]="gzip, deflate, sdch",
<br/>["method"]="GET",
<br/>["body"]="",
<br/>["tid"]="~1",
<br/>["url"]="/helloworld.page",
```

```
<br/>["User-Agent"]="Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116
Safari/537.36",
<br/>["Upgrade-Insecure-Requests"]="1",
<br/>["Accept-Language"]="en-US,en;q=0.8",
<br/>}
<br/></p><p>filename: script/apps/WebServer/test/helloworld.page,
dirname: script/apps/WebServer/test/</p><p>this is from included file:
this is global variable</p><p>this is from included file: this is
global variable</p><p>from test_include.page</p></body>
</html>
```

Web Server 源代码

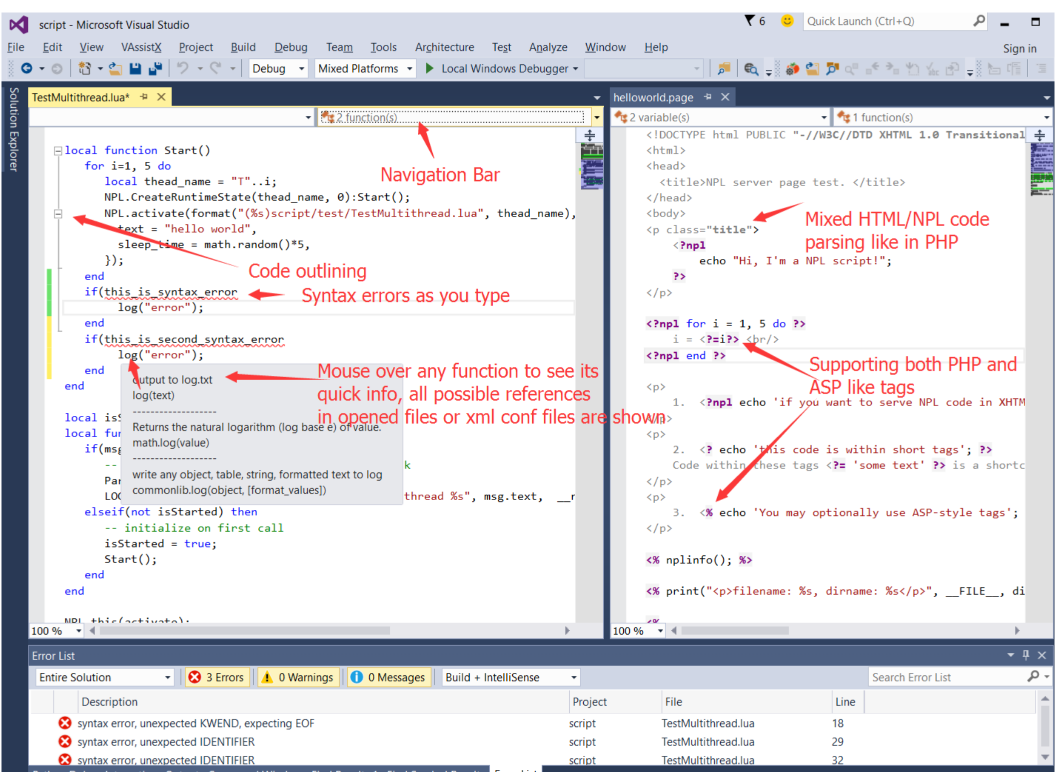
- script/apps/WebServer：用 NPL 实现的一个 web server（类似于 Apache）
- script/apps/WebServer/Webserver.lua：整个文件
- script/apps/WebServer/admin：用 NPL 编写的一个像 php 网站的框架

介绍

NPL web server 是使用 NPL 自己的消息系统实现的，并且是仅用 NPL 脚本编写的，没有任何外部支持。它允许你使用 NPL Runtime 在后端创建一个网站然后在客户端使用 Javascript/HTML5

在 Visual Studio 中的 NPL 语言服务插件提供了一个良好的编程环境：

IdreamTech



网站示例

- WebServerExample: 使用 NPL wiki 代码编写的一个简单的网站
- Wikicraft: 完整的网站示例
- Script/apps/WebServer/admin 是一个类似于著名的 WordPress.org 的基础网站框架。它推荐你拷贝目录和文件至你自己的网站根目录然后在那里进行工作
- Script/apps/WebServer/test 是一个测试站点，你可以在那里测试代码

程序化的开启一个 web server

从一个 web 根目录开启一个 server：

```
NPL.load("(gl)script/apps/WebServer/WebServer.lua");
WebServer:Start("script/apps/WebServer/test", "0.0.0.0", 8099);
```

在你自己的网页浏览器打开 `http://localhost:8099/helloworld.lua` 来测试

在 web 根目录有一个 `webserver.config.xml` 文件。如果没有找到配置文件，`default.webserver.config.xml` 文件被使用，它会将所有除了扩展的请求指向 `index.page` 和 *.lua,*.page 以及所有根目录中的静态文件，同时它会在 http://127.0.0.1:8099 host 所有的 NPL wiki 代码。

从命令行启动一个 wwb server

你可以使用内置文件去 bootstrap 一个 web server，运行下列程序：

```
npl bootstrapper="script/apps/WebServer/WebServer.lua"  port="8099"
```

- config： 可以被删掉，它在当前目录中是默认在 config/WebServer.config.xml 的

Web Server 配置文件

更多消息，参见 script/apps/WebServer/test/webserver.config.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- web server configuration file: this node can be child node, thus embedded in shared xml -->
<WebServer>
  <!--which HTTP ip and port this server listens to. -->
  <servers>
    <!-- @param host, port: which ip port to listen to. if * it means all. -->
    <server host="*" port="8099" host_state_name="">
      <defaultHost rules_id="simple_rule"></defaultHost>
      <virtualhosts>
        <!-- force "http://127.0.0.1/" to match to iternal npl_code_wiki site for debugging  -->
        <host name="127.0.0.1:8099" rules_id="npl_code_wiki" allow='{"127.0.0.1"}'></host>
      </virtualhosts>
    </server>
  </servers>
  <!--rules used when starting a web server. Multiple rules with different id can be defined. -->
  <rules id="simple_rule">
    <!--URI remapping example-->
    <rule match='{"^[^%.]+$", "robots.txt"}' with="WebServer.redirecthandler" params='{"/index.page"}'></rule>
    <!--npl script example-->
    <!--<rule match="%.lua$" with="WebServer.makeGenericHandler" params='{docroot="script/apps/WebServer/test", params={}, extra_vars=nil}'></rule>-->
    <rule match='{"%.lua$", "%.npl$"}' with="WebServer.npl_script_handler" params='%CD%'></rule>
    <!--npl server page example-->
    <rule match="%.page$" with="WebServer.npl_page_handler" params='%CD%'></rule>
    <!--filehandler example, base dir is where the root file directory is. %CD% means current file's directory-->
```

```
    <rule match="." with="WebServer.filehandler" params='{baseDir = 
"%CD%"}'></rule>
  </rules>
</WebServer>
```

每个 server 可以拥有一个默认主机以及多个虚拟主机。每个主机必须与一个 rule_id 相关联。rule_id 是在 rule 节点被查询的，它包含多个 rule 指定请求的 url 如何映射到它们的处理器。npl_code_wiki 是一个内部 rule_id，他常常被主机 `http://127.0.0.1:8099/`仅用来调试效果。如果你想要改变你的端口号，你需要因此更新配置文件否则 npl_code_wiki 将不可获得。

每一个 rule 包含三个属性：match，with 和 params

- “match”是一个规则的字符串表达式或是一个像上述示例代码中显示的注册表文件的数组表
- “with”是当 url 连接“match”时被调用的一个处理函数。更准确的说，它是处理函数的创建者，它返回实际的处理函数（请求，响应）的结尾。当 rule 在开启被编译时，这些创造函数在生成确切的处理函数时被调用
- “params”是一个选项符串或者传到处理器创造函数去生成实际的处理函数的表

rule 按顺序的被应用当它在配置文件中出现时。于是它们常常按重定向器，动态脚本和文件处理器的顺序被定义。

处理器的创作者

一些常见的处理器在 `common_handlers.lua` 中被实现。下面列举了一些最重要的处理器：

- **WebServer.redirecthandler**

  这是 url 重定向处理器。它创建了一个代替“match”和“params”的处理器

  ```
  <rule match="^[^%./]*/$" with="WebServer.redirecthandler" 
  params='{"readme.txt"}'></rule>
  ```

  上述代码将 `http://localhost:8080/` 重定向到了 `http://localhost:8080/readme.txt`

- **WebServer.filehandler**

  它产生了一个服务文件在目录中指定”params”或”params.baseDir”的处理

器

```
<rule match="." with="WebServer.filehandler" params='{baseDir =
"script/apps/WebServer"}'></rule>
 alternatively:
 <rule match="." with="WebServer.filehandler"
params='script/apps/WebServer'></rule>
```

上述代码将会从 `http://localhost:8080/readme.txt` 映射到磁盘文件 `script/apps/WebServer/readme.txt`

- **WebServer.npl_script_handler**

  当前情况它是和 WebServer.makeGenericHandler 一样的。将来他将会支持异步运行在其他线程的远程处理器。所有它建议使用者使用这个函数而不是 WebServer.makeGenericHandler 使用 npl 文件产生动态响应的服务请求。这对 XML 或 json 形式的类似于 REST 的 web 服务十分有用。

```
<rule match="%.lua$" with="WebServer.npl_script_handler"
params='script/apps/WebServer/test'></rule>
 alternatively:
 <rule match="%.lua$" with="WebServer.npl_script_handler"
params='{docroot="script/apps/WebServer/test"}'></rule>
```

上述代码会将 `http://localhost:8080/helloworld.lua` 映射到脚本文件 `script/apps/WebServer/test/helloworld.lua`

NPL Web Server 的缓存

1. 我们使用文件监视 API 来监视所有 web 根目录中的文件改变
2. 对于动态页面文件：如果改变的话就重新编译动态页面。所有编译过的*.page 代码一般都在缓存中
3. 对于静态文件：对直接的 HTTP 响应，所有文件都缓存在原始的或 gzip 形式。所有的静态文件都被使用一个在分离的为了避免在主线程中 IO 的影响的 NPL 线程来服务

应用代码的缓存

对于本地线程的缓存，有一个叫做 mem_cache 的帮助类

```
NPL.load("(gl)script/apps/WebServer/mem_cache.lua");
local mem_cache = commonlib.gettable("WebServer.mem_cache");
local obj_cache = mem_cache:GetInstance();
```

```
obj_cache:add("name", "value")
obj_cache:replace("name", "value1")
assert(obj_cache:get("name") == "value1");
assert(obj_cache:get("name", "group1") == nil);
obj_cache:add("name", "value", "group1")
assert(obj_cache:get("name", "group1") == "value");
```

不同电脑中的缓存

mem_cache 可以从远程电脑被配置到读/写数据中。

8.2.1 NPL Server Page

NPL 服务页面是一个混合了 HTML 和 NPL 的文件，通常有.page 的扩展

它是如何工作的

在 runtime 中，服务页面是被预处理到纯 NPL 脚本中然后进行处理。例如：

```
<html><body>
<?npl  for i=1,5 do ?>
   <p>hello</p>
<?npl  end ?>
</body></html>
```

上述的服务页面将会被预处理到下面的 NPL 页面脚本中，然后缓存之后的请求。

```
echo ("<html><body>");
for i=1,5 do
 echo("<p>hello</p>")
end
echo ("</body></html>");
```

沙盒环境

当一个 HTTP 请求来到且重定向到 NPL 页面处理器，一个特殊的沙盒环境表就被创建了，所有的与请求相关的页面脚本都在这个新创建的环境中被处理。所以你可以安全的创建全局变量并且对于每个页面请求，你可以期望它们是未初始化的。

然而，沙盒环境也有读/写进入到全局的每个 NPL Runtime 线程环境中，在那里 NPL 的类被加载。

NPL.load vs include

在一个页面文件中，可以调用 NPL.load 来加载一个已给的 NPL 类，例如 mysql.lua；

或者使用页面命令 `include` 来加载另一个页面文件到沙盒环境。不同之处在于被 NPL.load 加载的类将每次只被一个线程加载。而 `include` 将会被每个 HTTP 请求加载，再被工作线程处理。另外，NPL web Server 将会对所有页面文件监视文件的改动并且预编译它们当它被开发者更改时。对于含有 NPL.load 的文件，你需要重启你的 server 或使用特殊的代码去重新加载它。

将异步代码和暂停/继续结合

一个页面的处理通常包含两个阶段:

- 一个是从数据库引擎获取数据，这花费超过总时间的 95%
- 另一个是页面渲染，处理器受限的且只花费全部请求时间的 5%

	query database and wait for database result	MVC Render
duration	95%	5%

当使用 NPL yield 方法时，它允许同时处理其他网络请求在 90%的时间间隔，同时等待数据库作用在同一个系统级的线程。下面代码可以看出将异步代码和基于模板的页面渲染代码混合在一起有多么容易。它允许我们在单个线程中同时提供每秒 5000 条请求，即使每个请求从数据库取回需要 30ms。

下面是摘自我们 `helloworld.page` 的例子:

```
<?
-- connect to TableDatabase (a NoSQL db engine written in NPL)
db = TableDatabase:new():connect("database/npl/", function() end);

-- insert 5 records to database asynchronously.
local finishedCount = 0;
for i=1, 5 do
    db.TestUser:insertOne({name=("user"..i), password="1"},
function(err, data)
            finishedCount = finishedCount + 1;
            if(finishedCount == 5) then
                    resume();
            end
    end);
```

```
end
yield(); -- async wait when job is done

-- fetch all users from database asynchronously.
db.TestUser:find({}, function(err, users)  resume(err, users); end);
err, users = yield(true); -- async wait when job is done
?>

<?npl for i, user in ipairs(users) do ?>
    i = <?=i?>, name=<? echo(user.name) ?> <br/>
<?npl end ?>
```

代码解释：当第一个 yield()被调用时，页面渲染的处理被停止。它将根据前一个异步任务的结果恢复处理。在我们的示例中，当所有的五个使用者茶如道数据库中，我们将会调用 resume()，它会立马从最后一次暂停代码的位置继续处理页面。

请注意，我们建议你传递一个 boolean 的 `err` 作为第一个参数到 `resume`，因为我们所有的异步 API 都遵循一个规则。同时也请注意，我们传递 true 到第二个 `yield` 函数，它会告诉页面渲染器输出一个错误并且立马停止处理。如果你想自己处理错误，请什么都不要传递到 yield，如：`err, users = yield()`

页面命令

下面的命令只可以在一个 NPL 页面文件中被调用。它们是长命令的快捷方式：

```
Following objects and functions can be used inside page script:
    request:   current request object: headers and cookies
    response:   current response object: send headers or set cookies, etc.
    echo(text):   output html
    __FILE__: current filename
    __LINE__: line number
    page: the current page (parser) object
    _GLOBAL: the _G itself

following are exposed via meta class:
    include(filename, bReload):  inplace include another script
    include_once(filename):  include only once, mostly for defining functions
    gettable(tabNames): similar to commonlib.gettable() but in page scope.
    createtable(tabNames, init_params): similar to commonlib.createtable() but in page scope.
```

```
    inherit(baseClass, new_class): -- same as commonlib.inherit()
    print(...):  output html with formated string.
    nplinfo():   output npl information.
    exit(text), die():   end the request
    dirname(__FILE__):   get directory name
    site_config(): get the web site configuration table
    site_url(path, scheme):
    addheader(name, value):
    file_exists(filename):
    log(obj)
    sanitize(text)  escape xml '<' '>'
    json_encode(value, bUseEmptyArray)   to json string
    json_decode(str)  decode from json string
    xml_encode(value)    to xml string
    include_pagecode(code, filename):  inplace include page code.
    get_file_text(filename)
    util.GetUrl(url, function(err, msg, data) end):
    util.parse_str(query_string):
    err, msg = yield(bExitOnError)  pause execution until resume() is
called.
    resume(err, msg)  in async callback, call this function to resume
execution from l
```

在 script/apps/WebServer/npl_page_env.lua 中查看具体的文件

请求/响应对象

- 请求对象：当前请求对象：header 和 cookie
- 响应对象：当前响应对象：发送 header 或设置 cookie 等

内存缓存以及全局对象

NPL Web Server 有一个内置简便的本地内存缓存功能。你可以使用它去存储在主线程上被所有请求分享的对象。将来它或许被配置去运行在像 `memcached` 一样的分布式服务端上。

例如：

```
NPL.load("(gl)script/apps/WebServer/mem_cache.lua");
local mem_cache = commonlib.gettable("WebServer.mem_cache");
local obj_cache = mem_cache:GetInstance();
obj_cache:add("name", "value")
obj_cache:replace("name", "value1")
assert(obj_cache:get("name") == "value1");
assert(obj_cache:get("name", "group1") == nil);
```

```
obj_cache:add("name", "value", "group1")
assert(obj_cache:get("name", "group1") == "value");
```

另外，你也可以创建被所有 NPL 线程中请求分享的全局对象，通过使用 commonlib.gettable()方法。被 commonlib.gettable()创建的表对象与页面文件中的 gettable()不一样。之后我们将在本地页面范围创建表，它只在一个已给的 http 请求的生命周期存在。

文件上传

NPL web server 支持 multipart/form-data，通过它你可以上传二进制文件到服务端。推荐使用一个分部是服务端去上传文件，因为它是 IO 的范畴求当文件很大时会消耗带宽。

这里是一个客户端的代码去上传像 enctype="multipart/form-data"一样的代码

```
<form name="uploader" enctype="multipart/form-data" class="form-horizontal" method="post" action="/ajax/fileuploader?action=upload">
  <input name="fileToUpload" id="fileToUpload" type="file" class="form-control"/>
  <input name="submit" type="submit" class="btn btn-primary" value="Upload"/>
</form>
```

这里有一个 NPL 服务页面代码的示例，二进制文件的内容在 request:getparams()["fileToUpload"].contents 中。最大的允许上传文件的大小可以被 NPL runtime 属性配置。默认值是 100MB。

```
local fileToUpload = request:getparams()["fileToUpload"]
if(fileToUpload and request:getparams()["submit"] and fileToUpload.name and fileToUpload.contents) then
	local target_dir = "temp/uploads/" .. ParaGlobal.GetDateFormat("yyyyMMdd") .. "/";
	local target_file = target_dir .. fileToUpload.name;
	local fileType = fileToUpload.name:match("%.(%w+)$"); -- file extension

	-- check if file already exists
	if(file_exists(target_file)) then
		response:send({err = "Sorry, file already exists."});
		return
	end
	-- check file size
```

```
    if (fileToUpload.size and fileToUpload.size> 5000000) then
            response:send({err = "Sorry, your file is too large."});
            return
    end

    -- Allow certain file formats
    if(false and fileType ~= "jpg" and fileType ~= "png" and
fileType~="txt" ) then
            response:send({err = "Sorry, only JPG, PNG & TXT files are
allowed."});
            return
    end

    -- if everything is ok, try to save file to target directory
    ParaIO.CreateDirectory(target_dir);
    local file =
ParaIO.open(commonlib.Encoding.Utf8ToDefault(target_file), "w");
    if(file:IsValid()) then
            file:write(fileToUpload.contents, #fileToUpload.contents);
            file:close();
            response:send({contents = target_file, name =
fileToUpload.name, size = fileToUpload.size, ["content-type"] =
fileToUpload["content-type"], });
            return;
    else
            response:send({err = "can not create file on disk. file
name invalid or disk is full."});
    end
end
response:send({err = "unknown err"});
```

服务端重定向

```
<?npl
response:set_header("Location", "http://www.paracraft.cn/")
response:status(301):send_headers();
```

8.2.2 NPL Admin Site

NPL 管理端框架

`WebServer/admin` 是一个开源的机遇 NPL 的网站框架。它来自于 main package 并且包含所有的 NPLCodeWiki 源文件。它被当做你自己网络应用的一个演示程序和调试器被

处理。他也默认和你的网站 localhost:8099 同时运行在 127.0.0.1:8099/上。

如何运行 NPL 管理站点

```
npl script/apps/WebServer/WebServer.lua
```

或者执行更多的选项如

```
npl bootstrapper="script/apps/WebServer/WebServer.lua"  port="8099"
```

一旦开始运行，你就可以从 http://localhost:8099 访问管理站点.

介绍

根据著名的博客 WordPress.org 已经将它实现了。默认的主题模板是基于"sensitive"的，这是一个博客的自由的主题。"sensitive"主题与媒体没有关联并且可以根据屏幕宽度调整显示与布局，使得其适合在手机登设备上显示。

如何使用

1. 通过引用使用（推荐）

   基本来说，如果你想使用 admin 框架而不复制文件，你需要简单的添加下述代码到你的 rule 文件

```
<!--wp framework related js, css, files-->
    <rule match="^/?wp%-" with="WebServer.filehandler"
params='{baseDir = "script/apps/WebServer/admin/"}'></rule>
```

   在你的页面文件 index.page 中，你可以包含 wp 框架的主文件，例如：

```
<?npl

-- we will not load complete framework, but only ajax and helper
functions
WP_USE_MINI_LOADER = true;
include_once("script/apps/WebServer/admin/wp-main.page");

-- call your router, such as
include_once("./myproj/routes.page");
```

2. 通过复制使用

- 将这个文件夹里的所有文件拷贝到你自己的根网站
- 改变站点描述和菜单的 wp-content/database/*.xml
- 将你自己的网站页面添加到 wp-content/pages/，它可以被 url 通过文件名进入

- 入股你想要更多的定制外观，更改 wp-content/themes/sensitive 或创建你自己的主题文件夹。记得在 `wp-content/database/table_sitemeta.xml` 设置你的主题，它包含所有你站点的选项

3. 添加新的特性

   将你自己 `script/apps/WebServer/admin/`中的文件放入你的 dev 目录或工作目录，你的文件就会在 `npl_package/main` 中的文件前先被加载

架构

```
架构是基于 Wordpress.org (4.0.1)的. 尽管所有内容都是使用 NPL 编写的，它依然包含所有的函数，过滤器以及和博客相同的文件名。框架的源代码在`wp-includes/`
```

代码位置：* framework 加载器在 wp-settings.page；* site options 在 `wp-content/database/table_sitemeta.xml`，例如主题，默认菜单等；* menus 在 `wp-content/database/table_nav_menu.xml`

Ajax 框架

任何以’ajax/xxx’开头的 url 使用’wp-admin/admin-ajax.page’。它自动加载 page xxx 并且激活’do_action(‘wp_ajax_xxx’)’。如果请求 url 以’ajax/xxx?action=yyy’开头，那么 page xxx 将会被加载，并且’do_action(‘wp_ajax_xxx’)’将会被激活。一个处理 ajax 请求的页面需要调用’add_action(‘wp_ajax_xxx’, fuction_name)’来为 ajax 活动注册一个处理器。详情参见’wp-content/pages/aboutus.page’

8.2.3 Using TableDatabase

Table Database

Table database 是在 NPL 中实现的并且它是默认且被推荐的数据库引擎。它可以在没有任何外部支持或配置的情况下被使用。

Table Database 的介绍

它是一个无模式的，无服务器的，非关系型的数据库，它可以在 NPL 多线程中处理大

数据，含有绝对直观的类表型 API，可以自动索引并进行非常迅速的搜索，以及在一个类似于人脑的系统中保存数据和算法

性能

下述内容是在 Intel-i7-3GHZ CPU 上测试的

1. **运行保护代码**

下面是单个线程超过 100000 多操作的平均值

- Random non-index insert：43478 inserts/s
  - 在一个单线程中异步 API 被默认配置的 1 百万个记录测试
- Round trip latency call:
  - Blocking API: 20000 query/s
  - Non-blocking API: 11ms 或 85 query/s（因为 NPL 时间片段）
  - 例如 Round strip 意思是在前一个返回后开启下一个操作。这是延迟测试
- 使用 auto-index 的 Random select: 18761 query/s
  - 和上述例子一样但是有 findOne 操作
- Randomly mixingCRUD 操作： 965-7518 query/s
  - 和上述例子一样，但是在同一个 auto-indexed 表中随机调用 Create/Read/Update/Delete(CRUD)
  - 可以减慢混合读/写的速度当数据库变大的时候。例如，你可以在仅仅 10000 记录中获取 18000 CRUD/s 速度

2. **运行入侵模式**

你也可以使用内存日志文件或者忽略操作系统磁盘以 30%-100%的吞吐率将反馈写到远处的数据库。这个功能默认是关闭的。

代码示例：

```
NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
local TableDatabase =
commonlib.gettable("System.Database.TableDatabase");
```

IdreamTech

```
-- this will start both db client and db server if not.
local db = TableDatabase:new():connect("temp/mydatabase/",
function() end);

-- Note: `db.User` will automatically create the `User` collection
table if not.
-- clear all data
db.User:makeEmpty({}, function(err, count) echo("deleted"..(count
or 0)) end);
-- insert 1
db.User:insertOne(nil, {name="1", email="1@1",}, function(err,
data) assert(data.email=="1@1") end)
-- insert 1 with duplicate name
db.User:insertOne(nil, {name="1", email="1@1.dup",}, function(err,
data) assert(data.email=="1@1.dup") end)

-- find or findOne will automatically create index on `name` and
`email` field.
-- indices are NOT forced to be unique. The caller needs to ensure
this see `insertOne` below.
db.User:find({name="1",}, function(err, rows) assert(#rows==2);
end);
db.User:find({name="1", email="1@1"}, function(err, rows)
assert(rows[1].email=="1@1"); end);
-- find with compound index of name and email
db.User:find({ ["+name+email"] = {"1", "1@1"} }, function(err,
rows) assert(#rows==1); end);

-- force insert
db.User:insertOne(nil, {name="LXZ", password="123"}, function(err,
data)  assert(data.password=="123")       end)
-- this is an update or insert command, if the query has result, it
will actually update first matching row rather than inserting one.
-- this is usually a good way to force uniqueness on key or
compound keys,
db.User:insertOne({name="LXZ"}, {name="LXZ", password="1",
email="lixizhi@yeah.net"}, function(err, data)
assert(data.password=="1") end)

-- insert another one
db.User:insertOne({name="LXZ2"}, {name="LXZ2", password="123",
email="lixizhi@yeah.net"}, function(err, data)
assert(data.password=="123") end)
-- update one
```

```
db.User:updateOne({name="LXZ2",}, {name="LXZ2", password="2",
email="lixizhi@yeah.net"}, function(err, data)
assert(data.password=="2") end)
-- remove and update fields
db.User:updateOne({name="LXZ2",}, {_unset = {"password"},
updated="with unset"}, function(err, data)
assert(data.password==nil and data.updated=="with unset") end)
-- replace the entire document
db.User:replaceOne({name="LXZ2",}, {name="LXZ2",
email="lixizhi@yeah.net"}, function(err, data)
assert(data.updated==nil) end)
-- force flush to disk, otherwise the db IO thread will do this at
fixed interval
db.User:flush({}, function(err, bFlushed) assert(bFlushed==true)
end);
-- select one, this will automatically create `name` index
db.User:findOne({name="LXZ"}, function(err, user)
assert(user.password=="1");       end)
-- array field such as {"password", "1"} are additional checks, but
does not use index.
db.User:findOne({name="LXZ", {"password", "1"}, {"email",
"lixizhi@yeah.net"}}, function(err, user)
assert(user.password=="1");       end)
-- search on non-unqiue-indexed rows, this will create index
`email` (not-unique index)
db.User:find({email="lixizhi@yeah.net"}, function(err, rows)
assert(#rows==2); end);
-- search and filter result with password=="1"
db.User:find({name="LXZ", email="lixizhi@yeah.net", {"password",
"1"}, }, function(err, rows) assert(#rows==1 and
rows[1].password=="1"); end);
-- find all rows with custom timeout 1 second
db.User:find({}, function(err, rows) assert(#rows==4); end, 1000);
-- remove item
db.User:deleteOne({name="LXZ2"}, function(err, count)
assert(count==1);end);
-- wait flush may take up to 3 seconds
db.User:waitflush({}, function(err, bFlushed)
assert(bFlushed==true) end);
-- set cache to 2000KB
db.User:exec({CacheSize=-2000}, function(err, data) end);
-- run select command from Collection
db.User:exec("Select * from Collection", function(err, rows)
assert(#rows==3) end);
```

```
-- remove index fields
db.User:removeIndex({"email", "name"}, function(err, bSucceed)
assert(bSucceed == true) end)
-- full table scan without using index by query with array items.
db.User:find({ {"name", "LXZ"}, {"password", "1"} }, function(err,
rows) assert(#rows==1 and rows[1].name=="LXZ"); end);
-- find with left subset of previously created compound key
"+name+email"
db.User:find({ ["+name"] = {"1", limit=2} }, function(err, rows)
assert(#rows==2); end);
-- return at most 1 row whose id is greater than -1
db.User:find({ _id = { gt = -1, limit = 1, skip == 1} },
function(err, rows) assert(#rows==1); echo("all tests succeed!")
end);
```

为什么是一个新的数据库系统

当前 SQL/NoSQL 的实现不能同时满足以下需求：

- 使数据和计算保持相近，类似我们的大脑
- 不使用架构储存任意的代码
- 基于使用请求的自动索引
- 提供 blocking 和 non-blockingAPI
- 为了最大化本地性能不使用网络连接的多线程架构
- 能够本地储存大量的 GB 级数据
- NPL 表的本地文件储存形式
- 超简便客户端 API，例如操纵标准 NPL/lua 表
- 用 NPL runtime 容易建立及部署
- 没有服务端配置，在第一次使用时调用客户端 API 将会自动的开启服务端

关于事务处理

每个写入/插入操作默认一个写命令（虚拟事务处理）。我们将会周期性的（默认是 50ms）将全部队列里的命令传到磁盘。在这期间的每件事情不是成功就是失败。如果你担心数据损失，你可以手动的唤醒 flush 命令，然而这样做会很大程度的影响性能。请注意 flush 命令将会在数据库系统中影响整体的吞吐量。通常来说，每秒你只可以获得大约 20-100 flush（实际的处理）。除去每个命令的强制事务处理，你可以轻易地获得一个 6000 吞吐量的写

命令每秒（可以快 100 倍）。

下述的解决方案提供给单个命令了 ACID。在处理完一组很重要的命令且你想要确保这些命令确实像一个事务处理一样成功，客户端可以处理一个 waitflush 命令去检查之前的命令是否成功了。注意，waitflush 命令需要 50ms 或 Store.AutoFlushInterval 来返回。你可以使所有的调用都是异步的，于是 99.99%的情况下用户可以迅速获得一个反馈，但是用户在 50ms 后依然有很小的概率收到失败的消息。在客户端，你也需要避免用户解决下一个事物在 50ms 内。多数情况下，用户不那么快的点击鼠标，所以这个隐藏的逻辑基本不会被发现。

上述方法的局限性在于你仅可以知道一组命令是否被成功的传到了磁盘，但你不能回退你的改变，而且如果是多个命令的话你也不能得知哪些命令失败了或哪些成功了。

软件架构的选择

像我们的大脑，我们推荐每个电脑管理它们自己的数据块。随着现代的电脑拥有多个核，一个单独的(虚拟)机器本地管理 100-500GB 数据或每秒处理 1000 个请求变得有可能了。使用本地数据库引擎是这种情况下对性能和部署来说最好的选择。

为了增加更多的数据和请求，我们用机械分隔数据并使用更高级的编程逻辑去交流。这样，我们用自己的代码控制所有逻辑，而不是使用通用的方法像 MangoDB（NoSQL）或 SQLServer 等。

实现细节

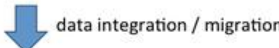
- 每个 data table（收集的数据）被存在一个单独的 sqlite 数据库文件中
- 每个数据库文件包含一个从 object-id 到 object-table（文件）的索引的映射
- 每个数据库文件包含一个架构表描述用到的额外的索引值
- 每个数据库文件对于每个用户索引值都包含一个 object-id 值
- 所有数据库 IO 操作都在一个专门的单个线程中处理

上述所有普通的思想可以在商业数据库引擎，如 CortexDB，中找到：

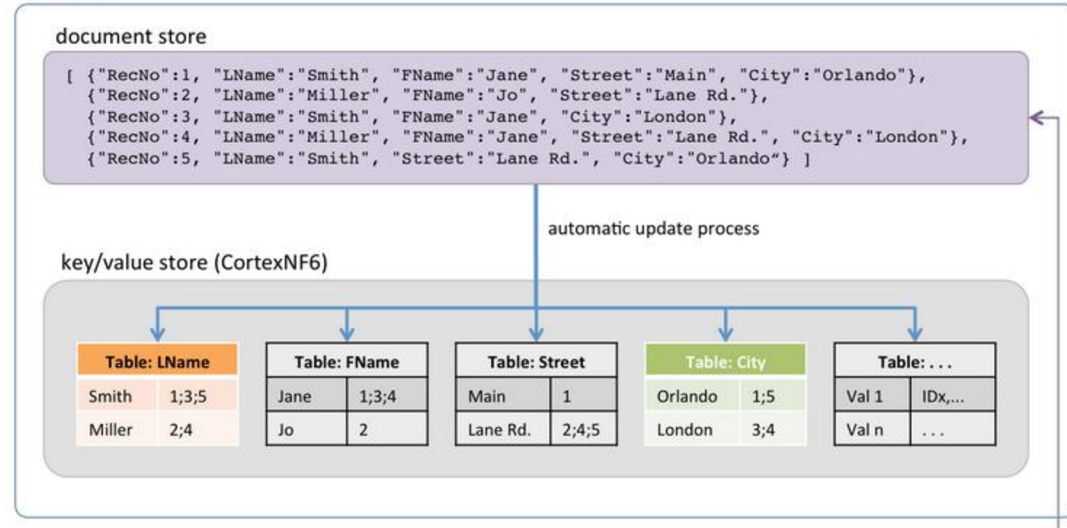
IdreamTech

Relational model / csv files

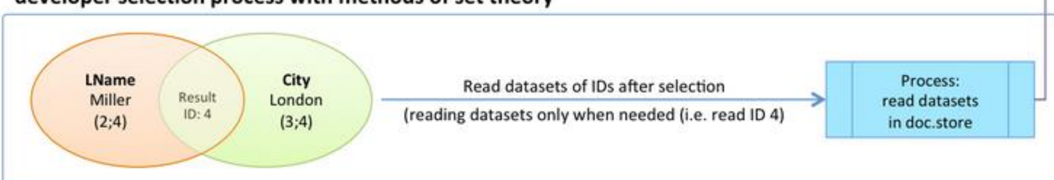
RecNo (ID)	LName	Fname	Street	City
1	Smith	Jane	Main	Orlando
2	Miller	Jo	Lane Rd.	
3	Smith	Jane		London
4	Miller	Jane	Lane Rd.	London
5	Smith		Lane Rd.	Orlando



CortexDB



developer selection process with methods of set theory



User Gide

Please see details on

https://github.com/LiXizhi/NPLRuntime/wiki/UsingTableDatabase。

8.2.4 MySQL & PostgreSQL

下载指南

- **在 Linux 上下载**

  在从源代码建立 NPLRuntime 之前，确保你已经下载了 mysql-client 和 libmysqlclient-dev

```
apt-get install mysql-client
apt-get install libmysqlclient-dev
```

如果你没有做这些，建立的程序将跳过 mysql 支持。

在 Linux 平台下，mysql 客户端在你从源代码创建 NPLRuntime 时被默认下载了（你已经下载了 mysql-client 和 libmysqlclient-dev 的情况下）。你需要在 NPL Runtime 目录下找到 `luasql.so`。请从你的程序开始的地方复制 `luasql.so` 到你的工作目录，这样 NPL 会自动寻找这个插件。

- **在 Windows 上下载**

  在 Windows 平台下，你必须手动创建并下载 mysql 客户端连接插件并从你的程序开始的地方复制 luasql.dll 到工作目录。https://github.com/LiXizhi/luasql

使用示例

当下载好之后，你可以像下面一样使用：

```
NPL.load("(gl)script/ide/mysql/mysql.lua");
local MySql = commonlib.gettable("System.Database.MySql");
local mysql_db = MySql:new():init(db_user, db_password, db_name,
db_host, db_port);
```

更好的编写代码的方式

推荐编写一个封装文件，如 `my_db.page`，它暴露一个包含函数的，类似 my_db 的全局对象到你实际的数据库中。

一个管理网站的框架示例如下：

- `script/apps/WebServer/admin/wp-includes/wp-db.page`

它会管理来自全局配置文件的连接字符串（密码，ip，池等）。在所有其他的服务页面你可以简单的包含这些文件并用下面的方法调用里面的函数：

```
local query = string.format("select * from wp_users where %s = '%s'
",db_field,value);
local user = wpdb:get_row(query);
```

使用 PostgreSQL

PostgreSQL 默认包含在 NPLRuntime 的 luasql 插件里如果在你的电脑中可以找到它们。下载方法和下载 MySQL 方法一样。

8.2.5 HTTPS SSL 服务器

用 SSL(https)部署 NPL Web Server

这里告诉你如何使用 SSL 作为反向代理服务器为 NPL Web Server 配置 Nginx。

为什么 NPL 本身不支持 SSL?

NPL 协议是基于 TCP 的，它可以在同一端口运行 HTTP 协议和 NPL 的 TCP 协议。如果 TCP 连接是用 SSL 编码的，它会损坏 NPL 内部的 TCP 协议。这也是为什么 NPL 不支持本地 SSL。然而 NPL 服务器可以通过 https 取回数据，所以它可以很好的调用 NPL server 中剩下的 SSL 保护的 API。

介绍

默认情况下，NPL 使用自己内置的 web server，它监听 8099 端口。如果你要运行一个私有的 NPL 实例的话这是很方便的，或者如果你只是想快速实现一些内容且不关心安全性。一旦你有实际的产品数据进入到你的主机，使用一个更安全的 web server 代理服务器如 Nginx 是一个更好的方案。

必备事物

这部分将详细介绍如何使用 SSL 作为反向代理服务器为 NPL Web Server 配置 Nginx。这个教程采取了一些类似于 Linux 的命令，一个工作的 NPL Runtime 的下载以及一个 Ubuntu 14.04 的下载。

你可以根据教程随后下载 NPL runtime 如果你还没有下载的话。

- **第一步——配置 Nginx**

  最近几年 Nginx 作为 web server 因为其速度和灵活性被广大用户喜爱，所以这是我们需要使用的 web server。

  下载 Nginx

  更新你的 package 列表并下载 Nginx：

```
sudo apt-get update
sudo apt-get install nginx
```

ldreamTech

获取认证

接下来你需要购买或创建一个 SSl 认证。这些命令是为了自签名的认证的，但是你需要获得官方授权的证明如果你想避免浏览器警告的话。

转移到正确的目录并创建一个证书：

```
cd /etc/nginx
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/nginx/cert.key -out /etc/nginx/cert.crt
```

你将被提示去输入一些关于证书的消息。你可以填写任何你想填写的东西，只需注意这些信息将会在证书特性中会显示。我们将数字设为 2048 bit 因为这是被 CA 签名的最小需要的大小。如果你想得到证书签名，你需要创建一个 CSR。

编辑配置

接下来你需要编辑默认的 Nginx 配置文件。

```
sudo nano /etc/nginx/sites-enabled/default
```

这就是最后的配置文件的样子。片段被分解并且解释如下。你可以更新或替代已存在的配置文件，尽管你可能想要先快速拷贝。

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}

server {

    listen 443;
    server_name npl.domain.com;

    ssl_certificate           /etc/nginx/cert.crt;
    ssl_certificate_key       /etc/nginx/cert.key;

    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers
HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log            /var/log/nginx/npl.access.log;
```

```
    location / {

      proxy_set_header        Host $host;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For
$proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;

      # Fix the “It appears that your reverse proxy set up is
broken" error.
      proxy_pass          http://localhost:8099;
      proxy_read_timeout  90;

      proxy_redirect      http://localhost:8099
https://npl.domain.com;
    }
  }
```

在我们的配置中，cert.crt 和 cert.key 设置反映我们创建 SSL 证书的位置。你将需要更新服务器名称并且用你自己的域名将线路 proxyredirect。并且 Nginx 有一些其他神奇的地方，它告诉请求需要被 Nginx 读取并且在响应端重写来保证反转代理服务器是工作的。

第一部分是让 Nginx 服务器监听在 80 端口（默认 HTTP）的所有请求并且将它们重定向到 HTTPS。

```
..
server {
   listen 80;
   return 301 https://$host$request_uri;
}
...
```

接下来设置 SSL。这是一个很好的默认设施但是可以被扩展。

```
...
  listen 443;
  server_name npl.domain.com;

  ssl_certificate           /etc/nginx/cert.crt;
  ssl_certificate_key       /etc/nginx/cert.key;

  ssl on;
```

```
  ssl_session_cache  builtin:1000  shared:SSL:10m;
  ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers
HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
  ssl_prefer_server_ciphers on;
  ...
```

最后的部分是代理发生的地方。它基本上获取所有的进入的请求并将他们代理到在本地网络接口监听 8099 端口的 NPL 实例。这是一个略微不同的情况，但是这部分教程有一些关于 Nginx 代理设置的好的信息。

```
...
location / {

    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header        X-Forwarded-Proto $scheme;

    # Fix the “It appears that your reverse proxy set up is broken"
error.
    proxy_pass          http://localhost:8099;
    proxy_read_timeout  90;

    proxy_redirect      http://localhost:8099 https://npl.domain.com;
}
```

另外，如果你没有一个域名来解析你的 NPL 服务器，那么上述的代理重定向的状态将不会正常工作除非作出调整。

- **第二步——配置 NPL Runtime**

在进行下面操作前，我们假设你已经下载好了 NPL。

对于要和 Nginx 一起运行的 NPL，我们需要更新 NPL 配置使其只监听 localhost 接口而不是所有（0.0.0.0），来保证消息传输可以适当的被处理。这一步很重要因为如果 NPL 依然监听所有接口的话，它将会依然可以从初始的端口（8099）被接入。

例如，应该这样开启 NPL web server：

```
pkill -9 npl
npl -d root="WebServerExample/" ip="127.0.0.1" port="8099"
bootstrapper="script/apps/WebServer/WebServer.lua"
```

注意，ip=“127.0.0.1”设置急需要被添加也需要被更改。

然后重启 NPL 及 Nginx：

```
sudo service nginx restart
```

现在你可以访问你的 HTTP 或 HTTPS 域，NPL 网站将会被安全的服务。如果你使用自签名的正式你将看到一个证书警告。

结论

剩下要做的事情就是确保每个操作都正确的完成。像上面提到的一样，你现在应该可以打开你的心得配置的 URL-npl.domain.com-不论用 HTTP 或 HTTPS。你应该重定向安全站点并查看一些站点信息，包括你最新更新的 SSL 设置。向前面所述，如果你没有通过 DNS 使用主机名，你的重定向将不会正常运行。在那种情况下你需要更改 Nginx 配置文件中 proxy_pass 部分。

8.2.6 NPL 网络套接字服务端

NPL 网络套接字服务器是一个 TCP 应用，它监听所有遵循 rfc6455 协议的服务器端口，并且它是用 NPL 和网络浏览器/NPL 服务器同时通过 json 传递信息实现的

编写一个 NPL 服务端页面

- 注册一个 NPL 文件来接收消息

```lua
-- 20 is short id for server address
 NPL.AddPublicFile("Mod/NplWebSocketSample/main.lua", 20);
```

- 接受消息

```lua
--Mod/NplWebSocketSample/main.lua
local function activate()
   if(msg and msg.nid)then
       local user_id = msg.nid; -- a user id
       local json = msg[1];
       local out={};
       if(NPL.FromJson(json, out)) then
           -- received msg from client
           local received_msg = out[1];
       end
   end
end
```

```
NPL.this(activate)
```

- 发送消息

```
local json_from = commonlib.Json.Encode(msg); -- the type of msg is
string/number/table
-- nid is a user id
NPL.activate(string.format("%s:websocket",nid),json_from);
```

- 使用 http 握手

```
--nplsocketsample.page
NPL.load("(gl)script/ide/System/Encoding/sha1.lua");
local Encoding = commonlib.gettable("System.Encoding");
if(is_ajax()) then
 add_action('wp_ajax_handshake', function()

        local user_id = request:get("user_id");
        if(not user_id or user_id == "")then
            user_id = "leio"; -- a test id
        end
        local key = request:header("Sec-WebSocket-Key");

        if(key and key ~= "")then
            response.headers = {};
            key = key .. "258EAFA5-E914-47DA-95CA-C5AB0DC85B11";
            key = Encoding.sha1(key,"base64");
            response:add_header("status", "101 Switching
Protocols");
            response:set_header("Connection", "Upgrade");
            response:set_header("Upgrade", "websocket");
            response:set_header("Sec-WebSocket-Accept", key);
            response:send_headers();

            -- a temp tcp connection id in npl server
            local tid = request.nid;
            -- quick authentication and establish connection
            NPL.accept(tid, user_id);
            -- Changing protocol type to 1,so we can use websocket
protocol. default 0 is npl protocol.
            NPL.SetProtocol(user_id,1);
            return;
        end
    end)
    return
end
```

- 改变传输协议

```
--change user's protocol to websocket
NPL.SetProtocol(user_id,1);
--change user's protocol to npl
NPL.SetProtocol(user_id,0);
```

编写一个客户端请求

- 创建一个网络套接字的实例

```
//@param nplsocketsample:it is a name of npl web page
//@param action:a name for ajax callback
//@user_id:a unique id for identify whose connection can be
accpeted or refused
 var ws = new
WebSocket("ws://localhost:8099/ajax/nplsocketsample?action=handshak
e&user_id=" + user_id);
```

- 客户端消息的格式

Key	Desc
s_id	server address
msg	client information

- 发送客户端消息

```
var server_address_id = 20;
var msg = "Hello World!";
var data = {
    s_id: server_address_id,
    msg: [msg]
}
var s = JSON.stringify(data);
ws.send(s);
```