LUCI 这个在百度上搜索除了一篇我的百度文库 luci 的介绍文章之外，前三页都是些不知所云的名词（足见百度在专业领域的搜索之烂），我却在大学毕业的大半年的大部分时间里与它纠结，由于开始的发懵到后来逐渐感觉到这家伙还很好玩的，现在就把我对 luci 的浅显认识介绍给大家。

官网：http://luci.subsignal.org/

有关 luci 的各个方面，你几乎都可以从这里获得，当然，只是浅显的获得，luci 的文档写的还算比较全，但是写的稍显简略，开始看的时候会有一点不知所措。

UCI 熟悉 openwrt 的人都会有所了解，就是 Unified Configuration Interface 的简称，而 luci 这个 openwrt 上的默认 web 系统，是一个独立的由严谨的德国人开发的 web 框架，是 Lua Configuration Interface 的简称，如果在您的应用里，luci 是对 openwrt 的服务，我们就有必要做一下 uci 的简介，我这里就不说了，见链接：

http://www.google.com.hk/url?sa=t&source=web&cd=5&ved=0CEMQFjAE&url=http%3A%2F%2Fnbd.name%2Fopenwrt-fosdem-09.pdf&ei=h52iTcXvOcrMcJ-xxOwD&usg=AFQjCNGFhumCIgS5tK_mDJ2dDFU4qsskfQ

有的时候，我们开发的 luci 是在自己的 Linux PC 上开发，在普通的 linux 上，一般是没有 uci 命令的，为了开发方便，可以手动编译一下，方法见链接：

https://forum.openwrt.org/viewtopic.php?id=15243

OK ,之前罗里罗嗦的说了很多，现在就进入正题，进入正题的前提是你已经 make install 正确的安装了 lua ，luci，以及编译好链接了相关的 so（如果你需要，比如 uci.so nixio.so），以及 make install 正确 web server,（我用的 web server 是 thttpd，也编译过 mongoose，lighttpd，在这三个之中，lighttpd 是功能最完善的，mongoose 是最小巧的）。

进入正题：

一：luci 的启动

在 web server 中的 cgi-bin 目录下，运行 luci 文件（权限一般是 755），luci 的代码如下：

```
1#!/usr/bin/lua          --cgi 的执行命令的路径
2require"luci.cacheloader"       --导入 cacheloader 包
3require"luci.sgi.cgi"           --导入 sgi.cgi 包
4 luci.dispatcher.indexcache = "/tmp/luci-indexcache"     --cache 缓存路径地址
5luci.sgi.cgi.run()                                                    --执行 run 方法，此方法位于*/luci/sgi/cgi.lua 中
```

run 方法的主要任务就是在安全的环境中打开开始页面（登录页面），在 run 中，最主要的功能还是在 dispatch.lua 中完成。

运行 luci 之后，就会出现登录界面：

```
-bash-4.0# pwd
/var/www/cgi-bin
-bash-4.0# ./luci

Status: 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Expires: 0
```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html class=" ext-strict"><head>

/*some html code*/

</html>

如果你成功的运行了 luci　就说明你的 luci 框架成功的跑了起来。

二：LUCI 的 MVC

1：用户管理：

在 luci 的官方网站说明了 luci 是一个 MVC 架构的框架，这个 MVC 做的可扩展性很好，可以完全的统一的写自己的 html 网页，而且他对 shell 的支持相当的到位，（因为 luci 是 lua 写的，lua 是 C 的儿子嘛，　　与 shell 是兄弟）。在登录界面用户名的选择很重要，luci 是一个单用户框架，公用的模块放置在 */luci/controller/下面，各个用户的模块放置在 */luci/controller/下面对应的文件夹里面，比如　　admin 登录，最终的页面只显示 /luci/controller/admin 下面的菜单。这样既有效的管理了不同管理员的权限。

2：controller 文件夹下的 lua 文件说明：（以 mini 用户为例）

在 mini 目录下面，会有一个 index.lua 文件，简略的代码如下：

```
module("luci.controller.mini.index", package.seeall)

function index()
    luci.i18n.loadc("admin-core")
    local i18n = luci.i18n.translate

    local root = node()
    if not root.lock then
        root.target = alias("mini")
        root.index = true
    end

    entry({"about"}, template("about")).i18n = "admin-core"

    local page   = entry({"mini"}, alias("mini", "index"), i18n("essentials", "Essentials"), 10)
    page.i18n    = "admin-core"
    page.sysauth = "root"
    page.sysauth_authenticator = "htmlauth"
    page.index = true

    entry({"mini", "index"}, alias("mini", "index", "index"), i18n("overview"), 10).index = true
    entry({"mini", "index", "index"}, form("mini/index"), i18n("general"), 1).ignoreindex = true
    entry({"mini", "index", "luci"}, cbi("mini/luci", {autoapply=true}), i18n("settings"), 10)
    entry({"mini", "index", "logout"}, call("action_logout"), i18n("logout"))
end

function action_logout()
    luci.http.header("Set-Cookie", "sysauth=; path=/")
    luci.http.redirect(luci.dispatcher.build_url())
end
```

这个文件定义了 node，最外面的节点，最上层菜单的显示等等。在其他的 lua 文件里，定义了其他菜单的显示和 html 以及业务处理路径。每个文件对应一个菜单相。

例如 system.lua 文件

```
module("luci.controller.mini.system", package.seeall)

function index()
    luci.i18n.loadc("admin-core")
    local i18n = luci.i18n.translate

    entry({"mini", "system"}, alias("mini", "system", "index"), i18n("system"), 40).index = true
    entry({"mini", "system", "index"}, cbi("mini/system", {autoapply=true}), i18n("general"), 1)
    entry({"mini", "system", "passwd"}, form("mini/passwd"), i18n("a_s_changepw"), 10)
    entry({"mini", "system", "backup"}, call("action_backup"), i18n("a_s_backup"), 80)
    entry({"mini", "system", "upgrade"}, call("action_upgrade"), i18n("a_s_flash"), 90)
    entry({"mini", "system", "reboot"}, call("action_reboot"), i18n("reboot"), 100)
end
```

mudel 是对应文件的， function index 定义了菜单，比如这一句 entry({"mini", "system", "reboot"}, call("action_reboot"), i18n("reboot"), 100)

第 1 项为菜单入口：

{"mini", "system", "reboot"}，mini 是最上层的菜单，即为用户项，system 为一个具体的菜单，reboot 为这个菜单的子菜单，如果 reboot 还需要加上子菜单的话，可以这样写：

entry({"mini", "system", "reboot"，"chreboot"}, call("action_chreboot"), i18n("chreboot"), 1),这样就会在 reboot 上产生一个新的子菜单，以此类推，可以产生 N 层菜单。

第二项为菜单对应的页面，可以是 lua 的源代码文件，也可以是 html 页面。

alias cgi form call 等定义了此菜单相应的处理方式，form 和 cgi 对应到 model/cbi 相应的目录下面，那里面是对应的定制好的 html 和 lua 业务处理。

alias 是等同于别的链接，call 调用了相应的 action_function。还有一种调用，是 template，是直接链接到 view 相应目录下面的 htm 页面。（说明：luci 框架下面的 htm 都是可以嵌入 lua 语句的，做业务处理，相当于 jsp 页面的内部的 Java 语句）。

问价查找对应简介：

entry({"mini", "system", "reboot"}, call("action_reboot"), i18n("reboot"), 100) ：对应此文件的 action_reboot function

entry({"mini", "system", "index"}, cbi("mini/system", {autoapply=true}), i18n("general"), 1)：对应*/model/cbi/mini/system.lua {autoapply=true} 这个失传的参数。

。。。。。

第三项为 i18n 显示，比如 entry({"mini", "system", "reboot"}, call("action_reboot"), i18n("reboot"), 100)，菜单的名字为 admin-core 文件内的对应显示。此处也可以这样写，i18n("reboot","重启")，即直接做了国际化。菜单上显示的就是“重启”。

第四项为现实的顺序，这个数字越小，显示越靠前，靠上。

3：model 业务处理和页面生成简介

我认为 model 的业务处理和 html 生成，是 luci 框架的精华，但是想用好它，最终扩展定义自己的页面也是最难的，但是一旦定义好了，后面的工作就会轻松高效简介统一，不失为一种好的解决方案。但是它又有缺点，就是写页面虽然统一，但是不够灵活。

下面以 SimpleForm 为例，讲解一下。

具体文件 */luci/model/cbi/passwd.lua

f = SimpleForm("password", translate("a_s_changepw"), translate("a_s_changepw1")) -- 调用 SimpleForm 页面 当然还是 I18N 从中捣乱，看上去没那么直观，不理他

pw1=f:field(Value,"pw1",translate("password")) -- SimpleForm 里面加一个 field 至于 SimpleForm 和 fiemd 是什么，一会去看 SimpleForm 页面去

pw1.rmempty=false -- 把 SimpleForm 的 rmempty 为不显示 后面就不做注释了 应该看得懂

了

```
pw2 = f:field(Value, "pw2", translate("confirmation"))
pw2.rmempty = false
function pw2.validate(self, value, section)
        return pw1:formvalue(section) == value and value
end
function f.handle(self, state, data)
        if state == FORM_VALID then      --这个就是业务处理了   你懂得   呵呵
                local stat = luci.sys.user.setpasswd("admin", data.pw1) == 0    -- root --> admin
                if stat then
                        f.message = translate("a_s_changepw_changed")
                else
                        f.errmessage = translate("unknownerror")
                end

                data.pw1 = nil
                data.pw2 = nil
        end
        return true
end
return f
```

说明:(simpleForm 位于 view/cbi   下面，可以研究一下，各个元素是如何定义的)

现在在给一个小例子：

以.*/luci/model /cbi/admin_system/version_manage.lua 为例，介绍一下 luci 中 web 页面 lua 代码

```
  6 local h = loadfile("/usr/local/luci/help.lua")
  7 if h then
  8         h()
  9 end
10 local help_txt = help_info and    help_info.version
```

加载帮助帮助文件 help.lua,关于 loadfile()的用法可以查看 lua 的手册(我还没完全弄明白，先用了)

help_txt 是一个全局变量

```
12 appadmin_path = "/usr/local/appadmin/bin/"
```

定义一个全局变量，其实跟功能跟宏一样，定义 appadmin 的绝对路径

```
14 versionlist = {}
15
16 function getline (s)
.....
32 end
33
34 function get_versionlist()
```

.....
68 end
69
70 versionlist = get_versionlist()

定义一个全局变量和两个函数，并初始化此变量

接下来就是和最终展现的 Web 页面直接相关的代码了，大部分都是对 luci 封装好的一些 html 控件（代码）的使用和扩展。luci 封装好的 html 控件

类可以在以下文件查看：./host/usr/lib/lua/luci/cbi.lua
71 m = SimpleForm("version", translate("版本管理"))
72 m.submit = false
73 m.reset = false
74 m.help = help_txt and true or false
75 m.helptxt = help_txt or ""
使用了一个 SimpleForm 的控件，SimpleForm 实际上对应一个 html 表单，是整个页面最大的"容器"，本页面内的绝大部分控件都处于 SimpleForm 内

，是它的子控件 。我知道的可以做>页面最大"容器"的控件还有一个 Map,但它需要./host/etc/config/目录下的一个配置文件，我没有使用。
submit reset 是 luci 默认就封装好的一个属性，分别控制 html 表单的"提交""重置"按钮;help helptxt 是我扩充的表单属性，分别控制 web 页面的

"帮助"功能和帮助内容。关于控件属
性的意义、实现和扩充可以按以下步骤进行：
在文件./host/usr/lib/lua/luci/cbi.lua 中查找控件名 SimpleForm, 然后可以找到以下行 664 self.template = "cbi/simpleform"这

表明 SimpleForm 的 html 模版文件为./host/usr/lib/lua/luci/view/cbi /simpleform.htm，通过研究 simpleform.htm 文件内容可以知道各属性的

功能以及模版中使用 lua 代码的方法，然后可以按类似的方法添加自定义的
属性。
77 s = m:section(Table, versionlist)
新建了一个 section,section 内定义了一个表格类，versionlist 是与其相关的变量（lua 的所有变量都可归类于 table 类型)
与 Table 关联的 table 变量应该是这种结构的：

```
t = {
    row1 = {column1 = "xxx", column2 = "xxx", .... },
    row2 = {column1 = "xxx", column2 = "xxx", .... },
    row3 = {column1 = "xxx", column2 = "xxx", .... },
    row4 = {column1 = "xxx", column2 = "xxx", .... },
}
```

然后定义 Table 的列控件

```
79 enable = s:option(DummyValue, "_enabled", translate("软件状态"))
83 appid    = s:option(DummyValue, "_appid", translate("软件版本"))
84 appname = s:option(DummyValue, "_appname", translate("软件名称"))
```

DummyValue 是只读的文本框，只输出不输入。Value 是单行文本框，可输出也可输入。Flag 是一个 checkbox,值为"1"时被选中，为"0"时未选中。

ListValue 是列表框...具体的用法可以看 ./host/usr/lib/lua/luci /model/cbi/ 下的文件（find ./host/usr/lib/lua/luci/model/cbi/ -name "*.lua" |xargs grep

"ListValue")

对于 table 内普通的字符串类的值，只需要把列控件的 id（括号内第二个值，如"_appid"）定义为 table 内对应的变量名（比如 column1 ）

对于非变通字符串类的值，或者为字符串但需要进行一定的处理然后再显示的值，可以按以下方法显示：定义该控件的 cfgvalue 函数属性

```
127         newinfo = up_s:option(TextValue, "_newifo", translate("新版本信息"))
128         newinfo.readonly = true
129         newinfo.rows = 11
130         newinfo.cfgvalue = function(self, section)
131                 local t = string.gsub(info, "Archive:[^\n]*", "")
132                 return t
133         end
```

定义 cfgvalue 后，luci 的处理函数会调用此函数为此控件赋值，（传入的 section 参数值为 row1/row2/row3 等，当处理到 row 几时值就为 row 几)

对于 DummyValue 等只输出不输入的类，还有一种赋值方法： 控件实例名（如 enable).value = xxx

对于有输入的控件 Value 等， .value 方法赋值在处理输入里会有一些问题，有什么问题以及如何解决可以做实验试试 ,也许是我使用方法不对造成的

对有输入控件的处理有两种方法：

1 定义控件的.write 属性

这种方法对处理比较独立的输入（与其它控件输入关系不大）比较适用

```
88 up_s = m:section(SimpleSection)
89 up_version = up_s:option(Button, "_up_version", translate("上传新版本"))
90 up_version.onlybutton = true
91 up_version.align = "right"
92 up_version.inputstyle = "save"
93 up_version.write = function(self, section)
94                 luci.http.redirect(luci.dispatcher.build_url("admin", "system", "version_manage", "upload"))
95 end
```

ps:只有当 Value 的 rmempty == false 时，Value 输入为空也会触发 write 函数, 需要对 rmemtpy 显示赋值为 false (xx.rmempty = false)

4：view 下面的 html 简介

这个是最好理解的　例：passwd.htm

```
<%+header%>
<h2><a id="content" name="content"><%:system%></a></h2>
<h3><%:reboot%></h3>
<p><%:a_s_reboot1%></p>
<%-
local c = require("luci.model.uci").cursor():changes()
if c and next(c) then
-%>
        <p class="warning"><%:a_s_reboot_u%></p>
<%-
end
if not reboot then
-%>
<p><a href="<%=controller%>/admin/system/reboot?reboot=1"><%:a_s_reboot_do%></a></p>
<%- else -%>
<p><%:a_s_reboot_running%></p>
<script type="text/javascript">setTimeout("location='<%=controller%>/admin'", 60000)</script>
<%- end -%>
<%+footer%>
```

<%+header%> <%+footer%>　加载公用的头部和尾部

<% lua code%>

<%:i18n%>

<%lua code%>

<%=lua 变量%>