# eyBuild 中文手册

**版本状态**

| 文件状态：<br>［ ］草稿<br>［√］正式发布<br>［ ］正在修改 | 文件标识： | eyBuild Group |
|---|---|---|
| | 当前版本： | 0.4.0 |
| | 作　　者： | newzy |
| | 完成日期： | 2006-4-25 |

**修订日志**

| 版本 | 日期 | 变更位置 | 变更内容 | 变更者 |
|---|---|---|---|---|
| 0.1.0 | 2006-1-22 | | 新建 | newzy |
| 0.2.0 | 2006-2-6 | 全部 | 完成初版 | newzy |
| 0.2.1 | 2006-2-9 | CSP 注释符 | 修改 CSP 注释符 | newzy |
| 0.2.2 | 2006-2-13 | 头部 | CSP 的解释、联系信息等 | newzy |
| 0.3.0 | 2006-3-25<br>2006-3-28 | 全文 | 为 eybuild-0.8 更新 | newzy |
| 0.4.0 | 2006-4-23<br>2006-4-25 | 全文 | 添加 FastCGI | newzy |
| | | | | |

**更多信息：**

请访问eyBuild的网站 http://www.eybuild.com

或发送email到: eybuild@hotmail.com, newzy@eybuild.com, xxt@eybuild.com

在线支持Skype: newzyx86　MSN: eybuild@hotmail.com　QQ: 178565448

目录

# 第1章 序言

VB/JAVA/PHP 等脚本直接嵌入在 HTML 中叫 ASP/JSP/PHP，那么用 C 直接嵌入在 HTML 中叫 CSP 吗?

是的, 现在我们可以直接将 C 语句嵌入在 HTML 中并叫它 CSP。C 语言天然好的"移植性/高效性/灵活性"，一直以来都是最受程序员青睐的语言。现在用 CSP 技术我们就可以轻松地将 C 语句直接嵌入到 HTML 源文件来快速编写 CGI 程序。

## 一般工作步骤:

编辑好的 CSP 源程序，用 eyBuild 开发包提供的 CSP2BIN 工具将 CSP 源文件生成 C 程序的源文件，再链接上 eyBuild 提供的高效 CGI 运行库，就可以在各种平台生成移植性非常高的 CGI 程序。

## CSP VS cgilib:

传统的 cgilib 的直接使用标准函数 printf 等语句输出 HTML 代码。不但使得 C 程序和 HTML 程序交织的混乱不堪，还使得页面输出的流程控制变得异常复杂。现在 ASP/JPS/PHP 等几乎完全取代用 cgilib。CSP 与 cgilib 的开发模式不同，它充分吸取了 ASP/JSP/PHP 等以 HTML/XML 为模板嵌入脚本等诸多优点，并充分融合 C 语言的语言特性。使得 CSP 的开发变得快速、高效，并大大提了最终代码的可读性和维护性。CSP 及其开发环境 eyBuild 是 cgilib 的继承和发展，同时目前也是开发高效率 WEB 应用的最佳选择。

## FastCGI 组件:

众所周知，FastCGI 程序比 CGI 程序响应速度高 3 至 5 倍。eyBuild 提供的编译选项 HAVE_FASTCGI，可以使原有的 CGI 源程序在不作任何修改的情况下作为 FastCGI 程序使用，这使得编写 FastCGI 程序更容易。同时，eyBuild 提供样板程序，可以在编译 FastCGI 和 CGI 程序之间切换更容易，调试更方便。

## To ASP/JSP/PHP 的程序员:

编写 CSP 程序就跟编写 ASP/JSP/PHP 一样，可以以先编写 HTML 文件为模板， 再在其中插入 CSP 的语句。 甚至有些时候，就可以直接拿 ASP/JSP/PHP 的源文件稍加修改后作为 CSP 的源文件了，因为它们都用类似 <% 和 %> 的标签进行标记的嘛。 如果你是 ASP/JSP/PHP 的程序员，并熟悉 C 语言，半天时间你就能把 CSP 全学会。

## To 嵌入式 WEB 开发:

CSP 设计的最原始的初衷，就是要为嵌入式开发定制的一套类似 ASP/JSP/PHP 的 C 语言开发工具。 因为嵌入式设备(如路由器/交换机/VoIP 网关 PBX 等)上用的开发语言主要是 C，而传统的 CGI 库 cgilib 以及开发模式远远不能跟上现代的开发需求。

现在 CSP 的 eybuild 开发环境提供的 PC 和嵌入式设备上高效移植的开发库，让服务器

上应用和嵌入开发进行了有效统一，使得两者上的开发变得更为容易。 同时，优秀的跨平台的移植性也是 eybuild 的重要特性。

实践证明，CSP 及其开发工具 eybuild 是嵌入式设备 WEB 开发的最理想工具，它能大大节缩短发周期(一般提高 4-6 倍)，提高最终代码的可读性、可维护性(HTML 和 C 代码进行了有效的分离，所以代码维护更容易)。

## 高效的页面/图片/CSS 集成技术:

通过 eyBuild 提供的集成技术，你可以把许多 CSP/HTML 页面集成生成到一个 CGI 中(包括页面相关的图片，CSS 文件及其它静态文件)。 甚至，你可以将一个小型的网站或 WEB 应用生成到一个 CGI 文件中，这使得最终的可执行脚本文件管理变得异常简单。 这一点在嵌入式设备上特别有用，因为它们中的很多只有有限的外存储器(如 Flash ROM)和文件系统。eyBuild 为最后生成的 CGI 程序在其内部建立了虚拟目录，使得页面间的链接和引用跟一般 HTML 的编写方法一样，非常方便建立和维护。同时对服务器级应用，这也将是一个非常有利的选择。

## 可以直接调用任意 C 的函数

在 CSP 源程序中还可以非常容易地包含 C 程序的头文件，这样在 HTML 代码中你就可以像写编写 C 文件一样调用外部函数或系统函数了，跟直接编辑 C 程序几乎没有差别。

## CSP 的宏指令指示符 @

用宏指令指示符不仅可以进行包含 C 程序的头文件，还可以包含其它 CSP 文件。 这样当许多页面需要引用共通的一部分时(如页头/页脚或其它部分)，包含其它 CSP 源文件这个功能显示特别有用。

## 有效的页面输出缓冲控制

跟 ASP/JSP/PHP 一样，通过宏指令指示符还可以有效控制页面输出时的 MIME 头，页面缓冲区大小等等。这种使得页面上的流程控制变得更简单更直观。

# 第2章　什么是 eyBuild

eyBuild，是 Easy Build 的简写，它是 CSP（C Language Service Page）开发套件的总称。eyBuild 是嵌入式设备上 WEB 应用的最理想的开发工具，同是也是服务器 WEB 应用开发最有益的补充，它主要包括 CSP2BIN、WEB2BIN、DONEMAP、eyBuildLib 等开发组件。

CSP 是 C Language Service Page 的缩写，它是一种基于 C 语言的脚本开发技术。跟 ASP、JSP 类似，它们都是 CGI（Common Gateway Interface）的一种变种。我们可以直接将 C 语句插入到 HTML/XML 等模板文件中来编写 CSP 的源文件，用 CSP2BIN 将该文件转换成 C 语言的源程序。

## 2.1　CSP 与 CGI

自 CGI 产生以来，C 语言以其高效性、灵活性和通用性一直是开发交互式 WEB 应用的最有吸引力的选择。但近年来，能直接内嵌于 HTML 文档中间的各种脚本工具，以其简便性易用性使一部分用户开始放弃了直接用 C 来开发 CGI 脚本。

但还有两类用户没有放弃用 C 来开发它们的应用，一是对性能追求较高的高端开发者，二是嵌入式设备的开者。前者选择 C 语言来开发它们的 WEB 应用，是因为 C 高效性、灵活性和通用性是各种脚本工具无法取代的。后者选择 C 语言，是因嵌入式设备的特点（内存、CPU 等资源有限等，不可在设备上运行如 ASP，PHP，PERL 等的脚本的运行环境）决定的（另外，目前嵌入式设备主要以 C 语言开发为主）。

自 CGI 出现以来，就有人开发方便 C 语言开发的开发库 cgilib，从目前的情况看 cgilib 并没有得到广泛的应用；反观下各种脚本开发工具，它们则受到 WEB 开发者的直接异常青睐。对比后不难发现，基于 C 语言的 cgilib 并没有从根本上解决交互式 WEB 应用的真正需求，这是因为直接用 C 语言无法像其它脚本一样嵌入在 HTML 文档中间，使得用输出 HTML 文档时非常烦琐，源代码可读性较差，维护困难。

现在用 CSP 来开发 WEB 应用程序则与之不同，我们可以像 ASP、JSP、PHP 等工具一样直接将脚本语言（这里指 C 语言）的语句嵌入到 HTML/XML 等模板文件中。因为在源文件中用<% %>标记分隔，使得 C 程序和 HTML/XML 代码程序进行了有效的分离，同时也使得代码的可读性、可维护性大大增强，开发周期大大缩短。

如果你是个 C/C++或 JAVA 程序员，你不必重新学习任何一种新语言，即可以像高效地开发 WEB 应用，工作模式跟原来几乎一样。如果曾经用 ASP、JSP、JSP 开发过并有一定的 C 语言基础，你会发现 CSP 简单就是它们的一种变种。无需太多学习即可用 CSP 做开发了。

下面是一段简单的 CSP 代码（头文件包含不是必须的，仅用作示范），它是将一段 C 程序直接嵌套在 HTML 模板文件中：

| | | |
|---|---|---|
| 1 | <html> | |
| 2 | <body> | |
| 3 | <% | 语句起始标签 |
| 4 | @ include <stdio.h> | C 语言头文件包含 |

| | | |
|---|---|---|
| 5 | @ include <my_header.h> | C 语言头文件包含 |
| 6 | %> | 语句结束标签 |
| 7 | 输出：<% = “hello world<BR>” %> | 输出字符串 |
| 8 | 也可以这样输出： | |
| 9 | <% | |
| 10 | $print (“%s<BR>”, “*** hello world ***”); | C 语句 |
| 11 | %> | |
| 12 | </body> | |
| 13 | </html> | |

## 2.2 为什么要选择 eyBuild 开发 Web 站点

eyBuild 是基于 CSP 技术的开发套件的总称，它具有如下特点：

1. 嵌入式设备 WEB 应用开发的最佳选择

   目前嵌入式设备上的 WEB 应用最主要还是 C 语言直接开发 CGI 程序。除在序言中提到了直接用 C 开发 CGI 的诸多缺点外，直接将 HTML 代码用 C 语句输出，不能对两都进行有效的分离，将大大降低 HTML 源程序和 C 源程序的可读性、维护性；直接用 C 语言开发还必须手工对页面输出的流程进行控制，还将大大降低了开发的效率。

   eyBuild 提供的 CSP 翻译工具，将 CSP 源程序翻译成 C 语言源程序。开发都只要编辑好 HTML 页面模板，然后在适当地方嵌入 C 语句即可以有效控制页面的输出；上文提到，还可以通过指示指令对输出页面的进行控制。使得 WEB 界面的开发和执行控制动作进行了有效的分离。使得嵌入式设备的 WEB 开发变得高效、有序。

2. 服务器 WEB 应用开发工具的最有益补充

   用 eyBuild 开发的 WEB 应用程序可与任意服务器 WEB 应用进行有效的接合，因为 eyBuild 开发的 WEB 应用程序是一个独立的自治单元，它不依赖于 WEB Serever 或其它脚本程序。所以你可以在其它任意脚本程序中通过 URL 发起 eyBuild 开发程序。eyBuild 最终会将源文件生成 C 代码，它生成的应用具有极高的执行效率。这样，你就可以用 eyBuild 来开发所需要的脚本组件，以解决效率的瓶颈。

3. 通用、高效、跨平台性

   这是因为：（1）eyBuild会将CSP源文件及其它非程序文件翻译成C语言的源程序，所以CSP天然地继承了C语言的一切特性。（2）eyBuild库的实现是完全遵循CGI1.1标准，所以你可以在任意操作系统（支持：Windows、Linux、BSD Unix、Free BSD、Sun Solaris、VxWorks、WinCE、uC/OS-II）上任意遵循CGI1.1 标准的WEB服务器（支持Apache，IIS，GoAhead，httpd，mini-httpd ……）下运行最终的CGI程序。

4. 规则简单，易学易用

   简单的说 CSP 就是直接 C 语句嵌入到 HTML 页面中，用 C 语言来响应请求、控制页面的动态输出行为，并通过简单的指令来控制页面的流程。用翻译器将 CSP 源文件翻译成 C 程序的文件。eyBuild 还提供了 MAP 工具，通过它可以非常容易将 CSP 源程序和其它非程序文件（如图片、静态 HTML 文件、CSS 文件）集成。

5. 用 FastCGI 使最终程序执行更高效

   众所周知，CGI 程序效率的瓶颈在于每次发起一个新的进程处理请求，而 FastCGI

则通过后台守护进程等方式使得处理效率能提高 4~5 倍。eyBuild 提供的 HAVE_FASTCGI 组件，使 CGI 源程序在不作任何修改的情况下即可转化为 FastCGI 程序，FastCGI 程序开发更简单。

6. 模板选择有更大的灵活性

   很幸运，你几乎可以以任意文本文件（如 HTML、XML、JS、CSS、TXT）为模板，在其中插入 CSP 语句。这一点与其它解释性脚本语言大不相同，这是因为 CSP2BIN 仅从中卒取 CSP 语句解析翻译，而视其它的数据为普通文本。这一特性，给 CSP 模板的选择大大增加了灵活性。

7. 上传文件更容易

   eyBuild专为大量数据传送（如提供一般表单项同时上传文件，见“在CSP中获取输入”）提供了一个API接口。使得多个文件上传，以及上传到服务器后的文件管理更加简单。同时还为特殊应用提供了场合提供了HOOKS接口。

8. eyBuild 提供了高效的运行库

   eyBuild 为不同操作系统平台提供了高效的运行库，可以直接在 CSP 中的 C 语句直接调用也可以在独立的 C 文件中调用。eyBuild 运行库类似于 cgilib，如获取用户输入参数、进行页面输出缓冲、处理文件上传）。

9. 使得脚本程序管理更简单

   eyBuild 提供了模块化处理功能，即将许多相关的 CGI 处理页面及相关文件（如：图片、CSS、.js 文件）生成到一个 cgi 执行文件中，并为每个处理页面或文件在 cgi 执行文件内部提供虚路径的映射。这样，每个最终生成的 cgi 文件就一个具有相关页面集合的功能单。甚至将整个站点生成在一个 cgi 文件中，或者作为一个独立的模块集成到其它应用程序中。这一点在站点的发布和维护有用上非常有用，应用实例见后文。

10. 源程序安全保密

    因为最终会将所有的 CSP 源文件生成 C 文件并最终编译成在一个二进制文件，而不像其它脚本程序一样由脚本解释器解释脚本源程序执行，所以并不存在源代码安全隐患。对源代码具有很好的保密性。

# 第3章 CSP 的语法

在 CSP 源文件中界于<%和%>之间的语句（可以跨越多行）都被称为 CSP 语句，CSP 翻译工具 CSP2BIN 将对<%和%>之间 CSP 语句进行解释。在 CSP 语句的任意字符都将被视为 HTML 源程序的语句，CSP2BIN 工具一般不对其进行解析。

## 3.1 CSP 的语句

CSP 语句包括单行语句和多行语句。凡<%和%>出现在同一个自然行的 CSP 语句称为单行语句；凡<%和%>跨越两个或两个以上的自然行的 CSP 语句称为多行语句。如下所示：

单行语句输出的例子：

```
1  单行语句输出：<% = “hello world<BR>” %>再输出一次<% = “hello world<BR>” %>
```

多行语句输出的例子：

```
1  <%
2  = “这是第二行<BR>”
3  = “这是第三行<BR>”
4  %>
```

## 3.2 语句格式

CSP 语句以行为解析单位，每一行作为一个独立的语句 CSP 语句行的格式如下：

CSP 语句总是以上边界符“<%”为开始，以下边界符“%>”为结束。CSP 语句内部每一个自然行为一个 CSP 语句行。CSP 翻译器以 CSP 语句行为单位对 CSP 语句进行解析。每一个语句行由三部分组成：前缀符、语句体、分号。如下所示：

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | <% | [前缀符] | [语句体 1] | [;] | | |
| 2 | | [前缀符] | [语句体 2] | [;] | | |
| 3 | | [前缀符] | [语句体 3] | [;] | %> | |

CSP 语句行中的三部分全部省略时，称为空语句。空语句用于书写时的格式控制，翻译时翻译器将跳过空语句行。

## 3.3 CSP 的语句行宏定义前缀符

宏定义前缀符是在 CSP 源文件被翻译成 C 程序源程序时处理，它主要用于简单页面处理的工作流程，目前 eyBuild 的 CSP 语句行包括如下宏定义前缀符：

| 前缀符 | 意 义 |
|---|---|
| // | 注释，// 后的内容不作任何解释 |
| @ | 指令指示符，用于对 CSP 进行特殊控制，如包含头文件、包含其它 CSP 页面 |
| $ | 调用页面内建函数，如$echo, $print (两者功能相同) |
| = | 字符串输出，计算 = 后面的函数、变量或常量的值并作为字符串进行输出 |
| =$ | 数字输出，计算 = 后面的函数、变量或常量的值并作为十进制数字进行输出 |
| =% | 自定义格式输出, |
| % | 保留，未定义 |
| ? | 保留，未定义 |
| 备注： | 其它可打印字符视为 C 语句的一部分，不作为前缀符解释，如&, !, ~ |

**说明：**

1. 当 CSP 语句行包含前缀符时，行末的分号不是必须的，翻译器会根据需要为最终果添加必要的 C 语句结束符——分号（;）。
2. 当 CSP 语句行**没**有前缀符时，翻译器会将其作为 C 语言的语句直接输出，且**不**会自动在语句最后为其加上分号。
3. 类似于 ASP、JSP 等，CSP 也可以用等进行语句输出，等号后面可以是一个常量、变量或一个函数，但是值（如函数返回的值）必须是一个字符串；当输出数据时可使用前缀符 “=$”，相当于 printf 格式化字符中的“%d”；同时，可用“=%”输出任意指定格式的数据类型。
4. 在 CSP 语句行中插入 C 语句时，注意行首字符不要跟 CSP 的前缀符冲突（如将所有运符写在行尾等方法）。一般地，按 K&R 著的 The C Programming Language 的 C 程序风格式，一般就不会与 CSP 语法冲突。

例 1：加与不加分号：

```
1  <% @ include <my_header.h>                语句最后不必加分号
2    $print(“xx”, “yyy”);                    语句最后不必加分号
3    $echo(“xx”, “yyy”);                     . .
4    my_func (                               语句最后不能加分号
5                  parameter1,               . .
6                  parameter2,               . .
7                  parameter3                . .
8                  );                        必须手工加上分号
9  %>
```

例 2：用 = 进行输出：

```
1  <% {
2  char * pstr = “this is a test<BR>”;       定义一个 C 变量
```

| | | |
|---|---|---|
| 3 | = "Build Date:" __DATE__ __TIME__ "<BR>" | 输出常 |
| 4 | = pstr | 输出变量 |
| 5 | = cgiPrefix() | 输出函数返回的字符串 |
| 6 | =$ (123+456) | 输出数值 |
| 7 | =%x, 128 | 输出按 16 进制输出 128 |
| 8 | } %> | |

## 3.4 CSP 的注释前缀符（//）

如果 CSP 语句行以前缀符“//”为开头，那么该行将被作为注释处理，即忽略#后面的任何内容。当然，也可以直接用 C 的注释：“/* xxx */”。它们的区别是，翻译器会将 C/C++的注释语句写入目录 C 程序中去，而“//”则注释语句则不会。注释行的语句体格式如下：

| | |
|---|---|
| // | 任意内容 |

例 1：CSP 的注释和 C 的注释

| | | |
|---|---|---|
| 1 | <% | |
| 2 | // This is CSP comment | CSP 注释 |
| 3 | var++; /* this is C/C++ comment */ | C 语句 + 注释 |
| 4 | %> | |

## 3.5 CSP 的指令前缀符（@）

包含前缀符“@”的语句行，被称为指令行。指令语句可以写在 CSP 源文件的任意处，因为它总是在翻译阶段被处理。指令行的语句体格式如下：

| | | | | | |
|---|---|---|---|---|---|
| 指令 | 分隔符 | ['] | 指令的值 1 | ['] | [, {指令的值 2}] |

1. CSP 的指令定义及其意义如下：

| | |
|---|---|
| pagesize | 设置页面缓冲区的大小，默认为 4K |
| autoflush | 是否允许自动 Flush 缓冲区，默认为 TRUE |
| mimeheader | 设置页面的 MIMI 头 |
| require | 指令所在处语句处插入另外一个页面 |
| include | C 语言的文件包含语句，相当于 #include |

2. 分隔符可以为：空格、TAB 或等号，等号两边的空格或 TAB 将被忽略
3. 指令值两边的双引号是可选的，即可以输入也可以不输入，但必须成对出现。

例 1：指令分隔符

```
1  <%
2  @pagesize = 4K          用 = 作分隔
3  @pagesize   4K          用空格或 TAB 分隔
4  %>
```

例 2：指令两边的分号

```
1  <%
2  @ autoflush = “TRUE”    加上引号
3  @ autoflush = FALSE     不加引号
4  %>
```

例 3：其它例子：

```
1  <%
2  @mimeheader = “Content-Type=html/text\n”          MIME 第一句
3  @mimeheader = “Cache-Control: no-cache\n\n”       将连在上一句后面
4  @include “myheader.h”                            包含头文件
5  @include <myheader.h>                            包含头文件
6  %>
```

## 3.6 CSP 的 require 指令

require指令用于在请求处插入另外一个页面。在翻译时仅是做了一个标记，并不把请求的文件原样复制到请求处。脚本程序执行时，会根据标记将请求页面的执行结果插入到请求处。被请求页面与请求页面共用一个请求页面的缓冲区，必要时可以通过pagesize指令设置请求页面的缓冲区大小。另外，require 的最大引用深度为 6 级。

require 指令可以包含最多两个参数，参数间用逗号分隔。第一个参数用于指明被请求页面的虚路径，第二个参数用于向被请求页面传递参数。参数的类型被定义为 void *__handle，所以可以被请求页面中使用 __handle 来获取传入的参数。

需要特别说明的是，require 只能引用当前模块（即同一 CGI 脚本程序）中的已存在的页面，否则 C 连接器会报告找不到所需要的函数错误。

例 1：用 require 其它 CSP 引用页面

```
1  Hello world. This is an English Version
2  <Add any thing here>
```
文件：/demo/body_en.csp

```
1  你好，这是一个中文版
2  <添加其它内容>
```
文件：/demo/body_ch.csp

```
1  <html><body>
2  <%
3      if (!strcmp(“English”, getParameter(“version”))
4          @require “/demo/body_en.csp”;
5      else
6          @require “/demo/body_ch.csp”;
7  %>
8  </body></html>
```
文件：/demo/main.csp

获取版本类型

引用英文版的内容

引用中文版的内容

例 2：引用其它页面，带参数：

```
1  <% char * title = “Hello world.”
2  @require “/demo/header.csp”
3  @require “/demo/header.csp”, “This title”
4  @require “/demo/header.csp”, title
5  %>
```
将另一页面插入在此处

带一个字符串常参数

带一个字符串指针参数

注意：在被请求的 “/demo/header.csp” 页面中，可通过指针变量__handle 获取传入参数（__handle 被定义成 void * __handle），如：

```
1  <html>
3  <head>
2  <% if (NULL != __handle) { %>
4  <title>
5  <% =(char*)__handle %>
6  </title>
7  <% } %>
8  </head>
```
如果指定了 title 则输出

输出 title

## 3.7 CSP 的内建函数前缀符（$）

前缀符“$”用于指示内建函数。目前支持的内建函数有：echo 和 print。翻译器翻译 CSP 源文件时会将 echo 或 print 替换成 ebprintf（__ebfp, fmt, …）格式输出。目的是为了兼容书写习惯，同时简化输出语句。格式参考 C 标准函数 printf()。

```
1  <%
2  $echo “hello word<BR>”                输出常字符串
3  $echo (“%s”, “hello world<BR>”)
4  $print “hello word<BR>”               输出常字符串
5  $print (“%d <BR>”, 123)               输出数值常量
6  %>
```

## 3.8 CSP 的字符串输出前缀符（=，=$ 和 =%）

类似于 ASP、JSP 等，CSP 也可以用等号进行输出字符串常量、变量、和函数返回值。由于 C 语言是一种强类型语言，对字符串和数值类型严格区分，所以在输出十进制数值时可使用数值输出类型“=$”。当然对于复杂的数据输出我们也可以通过=%自己控制输出格式化输出，语句行的格式如下：

| 输出前缀符 | 任意 C 语句表达式 | [;] |
|---|---|---|

例 1：用字符串输出前缀符输出

```
1  <%
2  char * pstr = “Hello world.<BR>”;     定义并初始化 C 变量
3  = “Build Date:” __DATE__ __TIME__<BR>  输出常字符串
4  = pstr                                 输出变量
5  = strchr(pstr, ‘w’)                    输出函数返回值
6  = (TRUE==FALSE) ? “ERROR”: “RIGHT”     输出表达式的结果
7  = “<BR>”
8  %>
```

例 2：用数值常输出前缀符输出

```
1  <%
2  int   var=123;                         定义并初始化 C 变量
3  =$var                                  输出数值常量
```

| | | |
|---|---|---|
| 4 | =$var+128 | 输出数值表达式 |
| 5 | %> | |

例 3：格式化输出前缀符输出

| | | |
|---|---|---|
| 1 | <% | |
| 2 | int var=123; | 定义并初始化 C 变量 |
| 3 | =%d-%02x, var, var | 输出两个变量 |
| 4 | =%s=%d<br>, “var”, var | 输出两个变量 |
| 5 | %> | |

注意，格式化字符串中以逗号结束，当格式化字符串中需要包含逗号时请选择 $echo 或$print 进行格式化输出。

## 3.9 在 CSP 中输出总结

从前文已经看到，这里只是做一些补充和总结：

1. 直接“=”，“=$”，“=%”等前缀符输出
2. 用 $echo, $print 输出
3. 直接调用 eyBuildLib 中库函数 ebprintf 输出

**注意：**

无论在 CSP 源文件中还是在与基相关的 C 程序源文件中都不要直接用标准输出函数 printf()等进行输出，因为页面是不会对它们的输出进行缓冲的，当后文通过 ebHeader 等控制输出“Cookie”或其它“HTTP”控制信息时，这些信息将不能正确的输出。另外它们的输出还会打乱正常的输出次序，使输出结果中的次序错乱。

## 3.10 在 CSP 中获取输入

通常，客户端通过 URL 和表单（FORM）两种形式进行提交请求。通过 URL 时采用的是 HTTP 的 GET 方法，通过 FORM 形式提交时我们可以指定是采用 GET 方法还是 POST 方法。在采用 POST 方法时，我可以通过“ENCTYPE”在表单中指明采用的编码方式为“application/x-www-form-urlencoded”或“multipart/form-data”（前者为默认的编码方式）。我们知道，在上传文件时必须采用“multipart/form-data”编码方式。

eyBuildLib 提供了两类 API 函数，它们分别用于：

（1）从 URL 或 POST 方法 application/x-www-form-urlencoded 编码方式中获取数据（参阅 getParameter()及其相关 API）

（2）从 POST 方法下编码方式为“multipart/form-data”的提交方式取得数据（参阅 dopost()及其相关 API

下面给出向个例子以说明如何使用它们。

**例1. 从 URL 中获得请求参数**

```
1 <html><body>
2 连接到页面：
3 <a href="<% =cgiPrefix(NULL) %>/demo.csp&arg1=Hello+world">
4 demo.csp
5 </a>
6 </body></html>
```

```
1 <html><body>
2 这是在 demo.csp 页面中，参数 arg1=
3 <% =getParameter(“arg1”) %>
4 </a>
5 </body></html>
```

```
1 这是在 demo.csp 页面中，参数 arg1= Hello world
```

**说明：**

第一段程序用于显示在客户端，提交参数 arg1=Hello+world

第二段程序为服务器端程序

第三段为当点击了客户端的超级连接后在浏览器上的输出结果

其中的“cgiPrefix(NULL) ”用于设置 CGI 前缀。

**例2. 通过表单提交**

```
1 <form action="<% =cgiPrefix(NULL) %>/demo.csp" method="post">
2 请在输入: <input type="text" name="inputbox1"><br>
3 <input type="submit" name="testit" value="testit"><br>
4 </form>
```

```
1 <html><body>
2 你输入的是: <% =getParameter("inputbox1") %><BR>
3 </form>
```

**例3. 从上传文件中获取输入**

```
1 <form method=post action="<% =cgiPrefix(NULL) %>/demo/uploads.csp"
2 ENCTYPE="multipart/form-data">
```

```
选择文件 1：<input type="file" name="upload_file1" size=250><BR>
选择文件 2：<input type="file" name="upload_file2" size=250><BR>
上传到目录：<input type="text" name="destpath" value="c:/upload" size=100><BR>
<input type="submit" name="Send" value="Send">
</form>
```

文件/demo/uploads.csp:

```
<html>
<head><title>Upload Server</title></head>
<body>
<%
@ include <ebpost.h>
@ include <direct.h>
{
        POST_PARAM *      p_param = NULL;
        THIS_POST *        p_post;
        int                     i, filecount;
        char *                  destdir = NULL;

        /* create post parameters:
         *
         * extern POST_PARAM * createPostParam
         *         (
         *         int                 max_post_size,        /@ max post size @/
         *         int                 file_max_size,        /@ max size each file @/
         *         int                 max_file_number,      /@ max file may uploads @/
         *         char *              upload_tmp_dir,       /@ where to store temp @/
         *         int                 options               /@ other options @/
         *         );
         */
        if (NULL == (p_param=createPostParam(0, 0, 3, NULL, 0)))
                return ERROR;

        /* done post */
        if (NULL == (p_post=dopost(p_param, __ebfp)) )
                return ERROR;

        mkdir (destdir=getParameter("destpath"));
        filecount = getPostFileCount(p_post);
        for (i=0; i < filecount; i++)
        {
```

```
            int       ret = movePostFile(p_post, i, destdir);

%>
    <table>
        <tr><td>源地址:</td>
            <td><%$print("%s", getPostSrcFileName(p_post, i)); %></td>
        <tr><td>存储到目录: </td>
            <td><% = destdir %></td>
        <tr><td>状态:</td>
            <td><%$ print("%s", (OK==ret ? "OK": "ERROR")); %></td>
    </table>
<%
    } /* end for */
    distoryPost(p_post, TRUE);
    p_post = NULL; p_param = NULL;
} /* end */
%>
```

执行结果举例（类似于）：

```
<html>
<head><title>Upload Server</title></head>
<body>
<table>
    <tr><td>源地址:</td>
        <td>C:\cspc.c</td>
    <tr><td>存储到目录: </td>
        <td>E:/uploads</td>
    <tr><td>状态:</td>
        <td>OK</td>
</table>
<table>
    <tr><td>源地址:</td>
        <td>c:\demo.c</td>
    <tr><td>存储到目录: </td>
        <td>E:/uploads</td>
    <tr><td>状态:</td>
        <td>OK</td>
</table>
</table>
```

**说明：**

第一段程序用于显示在客户端

第二段程序为服务器端程序

第三段为当客户端提交两个文件(cspc.c，demo.c)并指定目标目录为“E:/upload”时的输出结果

## 3.11 CGI 前缀和 ROM 前缀

什么是 CGI 前缀和 ROM 前缀？

CGI 前缀和 ROM 前缀分别是指 eyBuildLib 的两个 API 函数 cgiPrefix()和 romPrefix()。它们的函数原型为：

```
char * cgiPrefix(char * cginame);
char * romPrefix(char * cginame);
```

参数cginame是指可执的CGI文件的文件名。

函数cgiPrefix()和romPrefix()返回以cginame为基础构成的前缀，若cginame为NULL，则返回用自身URL构成的前缀，如（以/cgi-bin/demo.cgi中执行结果为例）：

```
1  <%
2  = cgiPrefix(NULL);
3  = cgiPrefix(“/cgi-bin/test.cgi”);
4  = romPrefix(NULL);
5  = romPrefix(“/cgi-bin/test.cgi”);
6  %>
```

结果

```
demo.cgi?cgi=
/cgi-bin/test.cgi?cgi=
demo.cgi?file=
/cgi-bin/test.cgi?file=
```

为什么要用 CGI 前缀和 ROM 前缀，如何使用它们？

一般地我们会将一系列相关的CSP源程序文件、CSS文件、图片文件等编译后生成一个独立的CGI文件（或称其为一个模块）。这个模块具有很高的自治性。当我们需要引用其中的文件时，可以通过CGI/ROM前缀来引用指定模块的名称，通过前缀后面的页面名称指定其中的子文件。子文件引用本模块的子模块时，不需要指定模块名称。下面给几个例子予以说明：

设：x.cgi 是由 a.csp、b.csp两个CSP文件和一个图片文件 c.jpg 所生成。

y.cgi 是由 d.csp 和图片文件 e.jpg 两个文件组成。

例 1. 在 d.csp 中引用 e.jpg 和 a.csp、b.csp、和 c.jpg，d.csp 的源程序如下：

```
1  <html><body>
2  显示图片 e.jpg: <img src=”<% =romPrefix(NULL) %>/e.jpg”>
3  <br>
4  显示图片 x.cgi->c.jpg: <img src=”<% =romPrefix(“x.cgi”) %>/e.jpg”>
5  <br>
```

```
6 <a href=”<% =cgiPrefix(“x.cgi”) %>/a.csp>转到 x.cgi->a.csp</a>
7 <br>
8 <a href=”<% =cgiPrefix(“x.cgi”) %>/b.csp>转到 x.cgi->b.csp</a>
9 </body></html>
```

当该客户端通过 URL（类似于 http://xx.yy.com/cgi-bin/y.cgi?cgi=d.csp）取得 d.csp 的页面时，其结果将为：

```
1 <html><body>
2 显示图片 e.jpg: <img src=”y.cgi?file=/e.jpg”>
3 <br>
4 显示图片 x.cgi->c.jpg: <img src=” y.cgi?file=/e.jpg”>
5 <br>
6 <a href=”y.cgi?cgi=/a.csp>转到 x.cgi->a.csp</a>
7 <br>
8 <a href=”y.cgi?cgi=/b.csp>转到 x.cgi->b.csp</a>
9 </body></html>
```

**说明：**

第一段程序为工作在服务器端的 CSP 源程序

第二段是输出到客户端浏览器的结果，即当该客户端通过 URL（类似于 http://xx.yy.com/cgi-bin/y.cgi?cgi=d.csp）取得 d.csp 的页面时输出的结果。

上传给出了如何通过 CGI/ROM 前缀取得指定的文件。其中第二行是引用当前模块中的文件，第四、六、八行引用了别的 CGI 模块中的文件。

**注意：**

以 CGI 前缀为开头的文件，必须为可执行的 CSP 文件，以 ROM 前缀为开头的可以是任意文件，如图片、CSS、文本文件。

在用 CGI/ROM 前缀时不要忘记前面的等号，它表示将函数返回值输出在当前位置。

# 第4章 eyBuild 开发环境简介

## 4.1 安装 eyBuild

目前 eyBuild 还没有针对各种开发平台的集成开发环境（IDE），不过它的安装和设置也十分简单。一般只需要如下简单的几步操作。

### Windows 环境安装：

1. 将 eyBuild 开发包 eybuild-xx-yy-zz.tar.gz 解压到一个目录，如 E:\eybuild
2. 设置环境变量 EYBUILD_BASE=E:\eybuild，方法是：
   A. 从菜单“开始→设置→控制面板→系统”打开系统属性对话框
   B. 从“高级”选项卡上点击“环境变量”按钮
   C. 可根据需要在“用户变量”或“系统变量”将 EYBUILD_BASE 正确添加进去

### Linux 环境安装：

1. 将 eybuild 开发包 eybuild-xx-yy-zz.tar.gz 解压到一个目录，如$HOME/eybuild
2. 打开.bash_profile 添加如下语句，以设置环境变量：
   EYBUILD_BASE=$HOME/eybuild
   export EYBUILD_BASE
3. 退出并重新登录当前 shell

## 4.2 eyBuild 目录结构

下面是安装好了的目录结构：

```
eyBuild/
│   License.txt
│   Log.txt
│   Readme.txt
│
├──bin/
│       csp2bin
│       csp2bin.exe
│       donemap.bat
│       donemap.unix
│       web2bin
│       web2bin.exe
│       wintar.exe
│
├──doc/
```

```
│   ├──API/
│   │       dopost.html
│   │       ebfrm.html
│   │       ebio.html
│   │       ebrequest.html
│   │       ebrespond.html
│   │
│   └──manual/
│           eyBuild_manual_ch.doc
│
├──include/
│       ebdef.h
│       ebfrm.h
│       ebio.h
│       eblib.h
│       ebpost.h
│       ebrequest.h
│       ebrespond.h
│       itemlist.h
│       undef.h
│
├──fastcgi/
│   │   Readme.txt
│   │
│   ├──include/
│   │       fastcgi.h
│   │       fcgiapp.h
│   │       fcgimisc.h
│   │       fcgio.h
│   │       fcgios.h
│   │       fcgi_config.h
│   │       fcgi_config_x86.h
│   │       fcgi_stdio.h
│   │       LICENSE.TERMS
│   │
│   └──lib/
│           libfcgi.dll
│           libfcgi.lib
│           mod_fastcgi-2.4.2-AP20.dll
│
├──lib/
│   │   eybuild.lib
│   │   eybuildlib
│   │
```

```
│   └─fastcgi/
│           eybuild.lib
│           eybuildlib
│
└─project/
    ├─demo/
    │   │   cgimain.c
    │   │   cspmap.map
    │   │   rommap.map
    │   │
    │   ├─csp/
    │   │       main.csp
    │   │       test.csp
    │   │
    │   ├─cspsrc/
    │   │   │   csp_maplist.c
    │   │   │   rom_maplist.c
    │   │   │
    │   │   ├─csp_p/
    │   │   │       _demo_main_csp.c
    │   │   │       _demo_test_csp.c
    │   │   │
    │   │   └─usr/
    │   ├─img/
    │   │       face.gif
    │   │
    │   ├─unix/
    │   │       domap.unix
    │   │       Makefile
    │   │
    │   └─win32/
    │           demo.dsp
    │           demo.dsw
    │           domap.bat
    │
    ├─raw/
    │   │   cgimain.c
    │   │
    │   ├─unix/
    │   │       Makefile
    │   │
    │   └─win32/
    │           raw.dsp
    │           raw.dsw
```

```
|
└──review/
    …
```

从上图可以看出，本版 eyBuild 开发环境下包括 6 个主目录：

1. bin　　eyBuild 所有需要的工具包（包括 Linux 和 windows）
2. doc　　eyBuild 的相关的文档手册
3. include　　C 头文件所在的目录
4. fastcgi　　对FastCGI支持的相关文件，可从FastCGI 官方网站获得
5. lib　　库文件所在的目录
6. project　　示例程序所在的目录

**说明：**

1. CSP 源程序可以由 WEB 程序员先编辑好 HTML 模板，再由 CGI 程序在模板上嵌入相关的程序
2. 可以根据平台需要可自由选择 C 程序编译器和链接器，如 gcc，VC++等均可。需要说明的是，必须用与之平台对应的 eyBuildLib 库。

## 4.3 project 的目录结构

从 eyBuild 的安装目录结构可以看出,, project 主要包括三个子项目：demo，review，raw。它们的功能描述如下

（1）demo 示例演示了一个基本的 CGI 程序交互过程，同时展示了如何将图片文件集成到 CGI 程序中。

（2）review 示例是一个简单的留言板程序

（3）raw 示例展示了如何通过 cgimain()直接写一个 CGI 程序。本例意在展示通过类似传统 cgilib 的方式编写过程。

对比 demo 和 review 会发现它们具有类似的目录结构和文件内容，它们的功能描述如下：

（1）cgimain.c　CGI 程序的主启动程序文件

（2）cspmap.map 和 romap.map CSP/是将静态文件映射到虚目录的配置文件

（3）csp/ 和 cspsrc/ 分别用于存放生 CSP 源程序文件和翻译后生成的 C 源文件所在的目录；

（4）cspsrc/csp_maplist.c 是 CSP 文件生成的到 C 程序的文件引用列表，需要将其加入项目文件中

（5）cspsrc/csp_maplist.c 是静态文件生成的到 C 程序的文件引用列表，需要将其加入项目文件中

（6）cspsrc/csp_p 是 CSP 源文件翻译成 C 程序所在的目录，不必将其中的文件加入项目文件中，会被间接的引用

（7）win32/ 和 unix/ 分别指 Windows 和 Unix 环境下的项目文件所在的目录

（8）win32/domap.bat 和 unix/domap.unix 分别是指 Windows 和 Unix 环境下用于将 CSP 文件及静态文件翻译成 C 程序的脚本工具

## 4.4 有用的 project-map

在 Windows VC++开发环境中，每次修改 CSP 文件后都要到相应 win32 目录下运行 domap.bat 以将 CSP 文件翻译成 C 文件，这是一个重复而繁琐的过程。我们可以使用 project-map 工具动态地定位要执行 domap.bat，以简化操作步骤。

下面介绍下如何将 project-map 添加到 VC++ 的工具栏上，以快速翻译 CSP 文件：

（1） 从菜单打开“Tools→Customize…”打开话框，并切换到“Tools”选项卡

（2） 在“Menu Contents:”最后新建一个菜单项，名字取作“project-map”

（3） 在“Command:”中输入 project-map 所在的路径（或点击右边的按钮选择），如“E:\eybuild\bin\project-map.bat”

（4） 在“Arguments:”中输入“"$(WkspDir)"”，注意$(WkspDir)两边加上双引号

（5） 选中“Use Output Window”复选框

（6） 切换到“Commands”选项卡，从“Category:”中选择“Tools”，在对话框的右边将第 8 个工具图标拖到工具栏适到的位置。

（7） 这样每次翻译 CSP 项目中的 CSP 文件，直接单击该按钮即可。project-map 能根据当前项目所在的目录定位 domap.bat 所在的路径并执行之。

## 4.5 CGI 程序的入口 cgimain()

最终生成的可执行脚本的入口是 cgimain()，它相当于应用程序的 main()主口。该函数必须由用户定义，用户可以通过它来控制后文脚本的处理行为。

一般地，cgimain()要完成如下功能：

（1） 设置调试环境，如调试时预置CGI环境变量，设置程序异常中断等

（2） 设置模块的主页面，即当不指定任何请求页面时返回的页面。未指定主页时，默认地为/main.csp。主页的设置方法是，设置环境变量 MAIN_PAGE，例如：putenv("MAIN_PAGE=/demo/main.csp");

（3） 启动 eybuild 环境，响应客户端请求。即调用函数 eyBuildExec()，基函数原型为：

```
extern int eyBuildExec
    (
    void *      cgitab,     /* cgi pages table */
    void *      filetab,     /* rom files table */
    void *      ebenv       /* ebuild environment, NULL is to set */
    );
```

（4） 下面是 cgimain()的模板，应用时根据需要对局部（一般将必须将 xxx 换成你项目中的名字，默认的为空）稍微改变即可。

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdLib.h>
4  #include <ebrequest.h>
5  #include <ebrespond.h>
6
```

```
 7  /* set debug break in VC++ */
 8  #define EXCEPTION_DEBUG          0
 9  int cgimain()
10  {
11      extern EB_CSP_MAPLIST    csp_maplist[];
12      extern EB_ROM_MAPLIST    rom_maplist[];
13  #if EXCEPTION_DEBUG    /* for debug */
14      int                      v=0, *p1=&v, *p2=NULL;
15      *p2 = 0;
16  #endif
17
18      ebSetDebug("GET", "cgi=/demo/main.csp"); /* set debug environment */
19      ebSetMainPage("/demo/main.csp");    /* set main-page */
20
21  #ifdef HAVE_FASTCGI
22      BEGIN_FASTCGI_LOOP();
23  #endif /* HAVE_FASTCGI */
24
25      /* respond require */
26      eyBuildExec(csp_maplist, rom_maplist, NULL);
27
28  #ifdef HAVE_FASTCGI
29      END_FASTCGI_LOOP();
30  #endif /* HAVE_FASTCGI */
31
32      return OK;
33  }
34
```

**说明：**

(1) 1~5 行是头文件包含

(2) 11~12 行，声明了外部定义的 maplist。它们通常在 DONEMAP 配置环境中指定，并被动态生成。

(3) 7~8/13~16 行，用于设置异常中断调试，一般不使用

(4) 18 行，设置调试环境。手工设置 HTTP 的 QUERY_STRING 和 REQUEST_METHOD。

(5) 19 行，设置默认主页文件的文件名，通过虚目录指定。

(6) 21~23 和 28~30 行，用于支持 FastCGI，只有 HAVE_FASTCGI 被定义时才会启用 FastCGI。

(7) 26 行，由 eyBuild 响应请求。

当然通过 eybuild SDK 也可手工地编写小型 CGI 应用程序。 eyBuild 开发环境自带了一个用 eybuild SDK 开发的一个这样的示例程序，见$(EYBUILD_BASE) /project/raw 目录。

## 4.6 在项目中添加 FastCGI 组件

在 eybuild 开发环境中支持 FastCGI 相应容易。因为你不必修改任何源文件，只需要在项目的配置文件中稍做修改即可。下面分别就 Windows 和 Linux 的 Apache 环境下如何支持 FastCGI 进行讲述。

**在 Linux 环境下添加 FastCGI 组件：**

（1） 确保Linux环境中安装了最新版的FastCGI，见附录A
（2） 打开项目的 Makefile
（3） 找到 # HAVE_FASTCGI= TRUE 所在行，将最前面的#去掉
（4） 找到 FASTCGI_LIB = /usr/local/lib/libfcgi.a 所在的行，修改 FASTCGI_LIB 指向 libfcgi.a 所在的正确位置。
（5） 执行 make all 重编译项目

**在 Windows 环境下添加 FastCGI 组件：**

（1） 确保 Windows 环境中安装了最新版的 FastCGI 或直接使用 eybuild 自带的 fastcgi 开发环境，见附录 A
（2） 像添加 EYBUILD_BASE 的步骤一样，添加环境变量 FASTCGI_BASE，如令 FASTCGI_BASE=D:/eybuild/fastcgi。
（3） 从菜单“Build→Configuration…”打开项目配置对话框
（4） 点击“Add…”按钮依据“Debug”和“Release”配置添加两个新项目配置，“DebugFastCGI”和“ReleaseFastCGI”
（5） 从菜单“Project→Setting…”打开项目设定对话框，从“Setting For:”下拉框中选择“Multiple Configurations”打开选择对话框，在对话框的复选列表中选中选中“DebugFastCGI”和“ReleaseFastCGI”（因为它们 FastCGI 的配置相同）。
（6） 打开“C/C++”选项卡，在“Category:”中选择“Preprocessor”。在预定义宏“Preprocessor definitions”中添加“,HAVE_FASTCGI”；在添加头文件搜索路径“Additional include directories:”中添加头文件搜索路径，结果类似：“$(EYBUILD_BASE)/include,$(FASTCGI_BASE)/include”。
（7） 打开“C/C++”选项卡，在“Category:”中选择“Code Generation”。在 Use run-time library 下拉框中选择“Multithreaded DLL”。
（8） 打开“Link”选项卡，在“Category:”中选择“General”。修改输出的文件“Output file name”，如设置为 D:\website\fcgi-bin\demo.fcgi。修改被引用库“Object/library modules”所在的路径，如改为 $(EYBUILD_BASE)/lib/fastcgi/eybuild.lib $(FASTCGI_BASE)/lib/libfcgi.lib（注意 eybuild.lib 是 fastcgi 子目录中的 eybuild.lib）。
（9） 保存退出项目设置“Project Settings”对话框。
（10） 在工具栏上任意处，单击右键打开“Build”工具条，在选择项目的活动配置“Select Active Configurations”中选择“Win32 DebugFastCGI”或“Win32 ReleaseFastCGI”。

（11） 从菜单“Build→Rebuild All”重编译当前项目即可。

## 4.7 CSP 页面内置对象

“CSP 页面内置对象”是指变量作用域仅在 CSP 文件页面内有效的系统内定变量和常量。通过这些变量和常量可以方便地对页面行为进行控制。在 CSP 页面内部可以自由引用它们，但一般不修改它们。分别描述如下：

1. void * __ebfp;
   __ebfp 是页面输出缓冲控制的描述符，当直接调用 ebprintf 时必须用该变量作为第一个参数。

2. void * __handle;
   __handle 是当通过 @require 宏指令标记子页面输出时，传给子页面的参数。无参数值入时，该参数默认值为 NULL。

3. char * __page_name;
   __page_name 是指当前页面名称，如被初始成：char * __page_name="main.csp";

4. char * __page_path;
   __page_path 是指当前页面的虚目录名，如被初始成：char * __page_path="/demo/";

5. char * __page_fullname;
   __page_path 是指当前页面的全路径名（指相对虚目录），
   如被初始成：char *__page_fullname="/demo/main.csp";

## 4.8 什么是虚目录

什么是虚目录，为什么要用虚目录？

虚拟是指在跟在 CGI/ROM 前缀后面的目录名，它用于区分同名的 CSP 或 ROM 文件。如<% =cgiPrefix(NULL) %>/demo/demo.csp 和 <% =cgiPrefix(NULL) %>/rt/demo.csp，其中的/demo/和/rt/就是两个虚目录。因为在最后生成的 CGI 文件中并不存在目录，所以称它为虚目录。同时虚目录还可以有效地隐藏源文件目录的一些细节。

## 4.9 MAP 文件

在开发工具的安装目录结构中我们注意到，demo 目录下有个 cspmap.map 文件。这个就是一个 map 文件。它是项目的配置文件，用于指示哪些目录或源文件需要被翻译，翻译后它们将使用什么名字的虚目录（见上一节）。

MAP 文件的语法格式：

1. 解析时以行为单位，一行最多不能多于 255 字符

2. 解析时空行和每行开头的空格、制表位被忽略
3. 以“!”为开头的行被视为注释行
4. 映射行格式：
   （1）文件名==> 虚目录
   （2）目录名==> 虚目录
5. 虚目录可以省略（但“==>”不可以省略），省略时被默认为“/”
6. 映射行若以目录名为开头，目录名最后必须以“\”或“/”结束
7. 文件名可以包括通配置符“*”和“?”
8. “目录名==> 虚目录”表示，下面的若干行是以该目录为搜索路径，并指定的文件映射到这个虚目录下

例 1．映射文件

| | | |
|---|---|---|
| 1 | ! This is comment | 注释语句 |
| 2 | d:/demo/csp/X.csp ==> X/ | 映射文件到虚目录“X/”下 |
| 3 | d:/demo/csp/Y.csp ==> /Y/ | 映射文件到虚目录“/Y/”下 |
| 4 | d:/demo/csp/Z.csp ==> /Z_ | 映射到根“/”并加下前缀 Z_ |
| 5 | d:/demo/csp/AA.csp ==> / | 映射到根“/” |
| 6 | d:/demo/csp/BB.csp ==> | 省略时同结果上 |

这样，我们可以在 CSP 文件中这样引用它们了：<% cgiPrefix(NULL) %>/

| | | |
|---|---|---|
| 1 | | 备注 |
| 2 | <% cgiPrefix(NULL) %>X/X.csp | 对应上面的第二行 |
| 3 | <% cgiPrefix(NULL) %>/Y/Y.csp | |
| 4 | <% cgiPrefix(NULL) %>/Z_Z.csp | |
| 5 | <% cgiPrefix(NULL) %>/AA.csp | |
| 6 | <% cgiPrefix(NULL) %>/BB.csp | |

例 2．使用配符“*”或“?”

| | | |
|---|---|---|
| 1 | d:/demo/*.csp ==> /demo/ | 映射扩展名为“.csp”的文件 |
| 2 | d:/test/*.html ==> /demo/ | 映射到同一目录 |

假设：
在目录 d:/demo/ 中存在文件 X.jpg、Y.csp
在目录 d:/test/ 中存在 X.html、X.html
那么我们就可以这样用如下方法引用它们了，好像它们来自同一真实目录一样：

| | | |
|---|---|---|
| 1 | | 备注 |
| 2 | <% romPrefix(NULL) %>/demo/X.jpg | 作为 ROM 类型文件引用 |

| | | |
|---|---|---|
| 3 | <% romPrefix(NULL) %>/demo/Y.csp | |
| 4 | <% romPrefix(NULL) %>/demo/X.html | |
| 5 | <% romPrefix(NULL) %>/demo/Y.html | |

例 3．映射目录到虚目录

| | | |
|---|---|---|
| 1 | d:/demo/ ==> /demo/ | **指明目录映射** |
| 2 | X.html | 列举文件 |
| 3 | Y.css | 列举文件 |
| 4 | d:/test/ ==> /demo/ | **指明目录映射** |
| 5 | A.txt | 列举文件 |
| 6 | B.doc | 列举文件 |

引用方法跟上面的例子一样。

## 4.10 WEB2BIN

工具 WEB2BIN 用于把 map 文件中所列的文件，全部翻译成 C 代码。将它与其它 C 源程序一起，用 C 编译器编译到最终的可执行脚本文件中去。当通过 URL 获取同名的文件时，脚本程序再把它原样地输出给请求的客户端程序。

这个工具很有用，是因为我们可以用它将不可执行的文件（如 JPEG、GIF、CSS 等）翻译成 C 代码，并在编译到最终的脚本文件中去。将它作为模块的一部分，当我们需要移动它们时，只移动一个可执行的脚本文件。

WEB2BIN 的命令行格式如下：

web2bin [map-file] [map-list-name] [output]

| | |
|---|---|
| map-file | 必须的，它用于指示从哪儿读取 MAP 文件 |
| map-list-name | 可选的，生成的列表名称，即生成的 C 代码的列表 |
| output | 可选的，将结果输出到该文件 |

如果不指明 map-list-name 和 output，那么将会依据 map-file 的文件自动生成为它们生成一个合法的字名

## 4.11 CSP2BIN

顾名思义，CSP2BIN 就是用于把 map 文件中所列的文件，全部翻译成 C 代码的工具。

CSP2BIN 的命令行格式如下：

csp2bin [map-file] [map-list-name] [dest dir]

| | |
|---|---|
| map-file | 必须的，它用于指示从哪儿读取 MAP 文件 |
| map-list-name | 可选的，生成的列表名称，即生成的 C 代码的列表 |
| dest dir | 指出翻译的结果存放在哪个文件夹中 |

## 4.12 DONEMAP

一般可以直接使用 CSP2BIN 和 WEB2BIN 等工具。为了简单操作，我们也可以使用 DONEMAP 工具为我们快速配置翻译环境。配置 DONEMAP 工具，即指定如下五个变量的值：

| | | |
|---|---|---|
| 1 | DEST_DIR | 结果存放的目录 |
| 2 | CSP_MAPFILE | CSP MAP 文件名 |
| 3 | CSP_MAPLIST | CSP 文件列表名 |
| 4 | ROM_MAPFILE | ROM MAP 文件名 |
| 5 | ROM_MAPLIST | ROM 文件列表名 |

**说明：**

1. DEST_DIR 是必须的，它指示生成的文件存放的目录
2. CSP_MAPFILE、ROM_MAPFILE 之一可以为空，它表示没有 CSP 文件或 ROM 文件需要翻译
3. 通常，可以修改下面的脚本文件（以 Windows 下的批处理文件 domap.bat 为例，表格第二行中所列的五变量）来配置翻译环境
4. REM 表示注释，SET 用于设置环境变量
5. 注意：等号“=”前后不要随意加空格，空格不会自动删除

win32/domap.bat 模板文件

| | |
|---|---|
| 1 | @echo off |
| 2 | REM ############################################################<br>set DEST_DIR=destdir<br><br>set CSP_MAPFILE=cspmap.map<br>set CSP_MAPLIST=csp_maplist<br>set ROM_MAPFILE=<br>set ROM_MAPLIST=<br>REM ############################################################ |
| 3 | if not defined EYBUILD_BASE (<br>  echo Not defined "EYBUILD_BASE".<br>  pause & goto :EOF |

```
)

REM convert file
call %EYBUILD_BASE%/BIN/DONEMAP.BAT "%CD%" %~dp0 %1
echo on
```

unix/domap.unix 模板文件

```
#!/bin/bash
# Copyright(C) eyBuild Group, 2005, 2006. All Rights Reserved.
#
# modification history
# --------------------
# 01b, 2006-3-23 newzy    add get current directory
# 01a, 2006-2-15 newzy    create
#

############################################################
DEST_DIR=../cspsrc

CSP_MAPFILE=../cspmap.map
CSP_MAPLIST=csp_maplist
ROM_MAPFILE=../rommap.map
ROM_MAPLIST=rom_maplist
############################################################

# get current directory
[ ${0:0:1} = / ] && c_dir=${0%/*} || c_dir=$PWD/${0%/*}

# REM convert file
$EYBUILD_BASE/bin/donemap.unix $DEST_DIR $CSP_MAPFILE $CSP_MAPLIST
$ROM_MAPFILE $ROM_MAPLIST $c_dir
```

# 第5章 建立工程并生成 CGI 可执行文件

## 5.1 建立工程的一般步骤

创建一个工程并生成 CGI 文件通常需要如下几步：

1. 编辑 CSP 源程序及其相关的 C 程序源文件和 ROM 文件（如: 图片、CSS 文件、JavaScript 文件等）
2. 配置翻译器（map 文件和 DONEMAP 环境）
3. 将 CSP 文件和 ROM 文件翻译成 C 源程序文件
4. 用 C 编译器第一步编写的 C 程序文件和第三步生成的 C 程序文件编译成目标文件。用链接器将目标文件与库 eyBuildLib 一起链接，生成可执行的 CGI 文件。

**说明：**

1. 一般地，CSP 源程序可以由 WEB 程序员先编辑好 HTML 模板，再由 CGI 程序员在模板上嵌入 C 程序程序完成。
2. 可以根据平台需要可自由选择 32 位 C 程序编译器和链接器，如 gcc，VC++等均可。需要说明的时，必须用与之平台对应的 eyBuildLib 库。

## 5.2 示例

在 Linux/Windows 环境下可直接运行 $(EYBUILD_BASE)/project 中所给出示例程序。或者直接复制其中的部分或全部到新目录，经改动后形式新的项目文件。

下面给出 Windows 环境下的一个示例，以演示如何建一个新的工程、生成可执行文件并 WEB 服务器下测试它。在 Linux 环境下具有类似的步骤或过程，请参阅 $(EYBUILD_BASE)/project/{project-name}/unix 目录下的 Makefile 和 domap.unix。

### 5.2.1 环境准备：

| | |
|---|---|
| 操作系统： | Windows 2000 或 Windows XP |
| Web Server： | 安装好 IIS4.0 以上 或 Apache 1.3 或 Apache 2.x |
| C 编译器： | 安装好 VC++ 6.0 |
| eyBuild： | 安装好eyBuild 00.06.00 或以上版本 |

### 5.2.2 创建源文件

（1）为工程新建立如下目录：

| | |
|---|---|
| demo | 根目录 |
| ├──csp | 存放 CSP 源文件 |
| ├──cspsrc | 存入 CSP 翻译生成的及其相关的 C 源文件 |
| ├──img | 存放图片文件 |

└──win32 存放项目文件

（2）在 /img/ 目录中创建一图片名为 face.gif，或者直接从将文件 $(EYBUILD_BASE)/project/demo/img/face.gif 拷贝到/img/目录中

（3）在 demo/csp/目录中建立两个 CSP 源文件，分别为 main.csp 和 test.csp，并分别为他们添加如下内容：

文件: demo/csp/main.csp

```
1 <html><body>
2 Hello world. <img src=”<% =romPrefix(NULL) %>/img/face.gif”>
3 <BR>
4 <form action="<% =cgiPrefix(NULL) %>/demo/test.csp" method="post">
5 请在输入: <input type="text" name="inputbox1"><br>
6 <input type="submit" name="testit" value="testit"><br>
7 </form>
8 </body></html>
```

文件: demo/csp/test.csp

```
1 <html><body>
2 This is in test.csp<BR>
3 你在文本框输入的内容为：<% =getParameter(“inputbox1”) %><BR>
4 [ <A href=”<% =cgiPrefix(NULL) %>/demo/main.csp”>返回</A> ] <BR>
  </body></html>
```

## 5.2.3 配置翻译器

（1）在 demo/目录中建立两个 MAP 文件，分别为 cspmap.map 和 rommap.map，并分别为他们添加如下内容（注意，仅第 5 行有区别）：

文件: demo/cspmap.map

```
1 # this file list how to map files to virtual directory
2
3 # source directory              virtual directory
4 # -----------------             -----------------
5 csp/*.csp                 ==>       /demo/
6
```

文件: demo/cspmap.map

```
1 # this file list how to map files to virtual directory
```

```
2
3  # source directory              virtual directory
4  # ------------------            ------------------
5  img/*.*                  ==>       /img/
6
```

（2）在demo/win32 目录中建立一个批处理文件domap.bat。方法是复制domap.bat的内容，必要要时可修改下表突出显示的内容（默认不必修改）：

```
2  REM ###########################################################
   set DEST_DIR=../spsrc

   set CSP_MAPFILE=../cspmap.map
   set CSP_MAPLIST=csp_maplist
   set ROM_MAPFILE=../rommap.map
   set ROM_MAPLIST=rom_maplist
   REM ###########################################################
```

## 5.2.4 翻译 CSP 文件和 ROM 文件

上一步已经配置好了翻译环境，现在就可以直接双击 demo.bat 进行翻译文件了。如果在 CSP 源文件中有错误，翻译器会输出错误的行号和错误的原因。下面是屏幕上输出的结果：

```
   ########## Build csp ##########
   No.    Local Path and Name                  Virtual Path and Name
   ======================                ======================
   001. csp/main.csp                    ==> /demo/main.csp
   002. csp/test.csp                    ==> /demo/test.csp
   2 files convert OK.

1  ########## Build romfile: ##########
   No.    Local Path and Name                  Virtual Path and Name
   ======================                ======================
   001. img/face.gif                    ==> /img/face.gif
   002. img/Thumbs.db                   ==> /img/Thumbs.db
   2 files convert OK.

   请按任意键继续. . .
```

再让我们看翻译的结果，并特别关注下目录 demo/cspsrc/下的目录结构：

```
DEMO/
│   cspmap.map
│   rommap.map
│
├──csp/
│       main.csp
│       test.csp
│
├──cspsrc/
│   │   csp_maplist.c
│   │   rom_maplist.c
│   │
│   ├──csp_p
│   │       _demo_main_csp.c
│   │       _demo_test_csp.c
│   │
│   └──usr
├──img/
│       face.gif
│
└──win32/
    └   demo.bat
```

容易发现：demo/cspsrc 目录中多了两个文件夹（csp_p/和 usr/）和两个 C 文件。这两个 C 文件的名字正是在 dome.bat 所指定的。

**说明：**

（1）csp_p 后面的_p 意思是说私有的（private），用户一般不用关心的部分。用户不能直接编译其中的 C 源程序文件，因为它们都是动态生成的，每次翻译后它们的内容将会改变。

（2）usr 目录，顾名思义它代表用户自己维护的目录。用户可以直接将与翻译结果相关的 C 源文件（如 CSP 中引用的用户自定义函数等）放在此目录中，此目录中的内容只有用户才可以改变它们，翻译器只是帮助创建这个目录。

## 5.2.5 创建项目编译环境

（1）打开 VC++，在 prj 目录中创建一个空的 Win32 Console Application 项目。项目名为 demo.dsp

（2）将/demo/cspsrc/csp_maplist.c 和/demo/cspsrc/rom_maplist.c 添加到工程中（注意，

不要将/demo/cspsrc/csp_p/中的文件添加到项目）

（3）添加头文件搜索路径，方法是：打开 Project → Setting →C/C++ 选项卡，在分类（Category）下拉框中选择 Preprocessor，并在 Additional include directories:中添加：$(EYBUILD_BASE)/include

（4）添加 eyBuildLib 库的搜索路径，方法是：打开 Project → Setting →Link 选项卡，在分类（Category）下拉框中选择 General，并在 Object/library modules:输入框最后（项与项之间用空格分隔）添加项$(EYBUILD_BASE)/lib/eybuild.lib

（5）最后，还必须手工添加一个 cgimain()。方法是，新建一个空的 C 文件/demo/cgimain.c，然后将 cgimain() 模板中的内容粘贴到该文件中去。

至此，就可以编译项目了。编译完毕，默认的会在 demo/win32/debug 目录生成一个可执行文件 demo.exe。

我们也可以用上面添加 eyBuildLib 库的搜索路径的方法，在同一选项页里的“Output file name:”输入框里改变量输出文件名字和路径（如改为：d:/cgi-bin/demo.cgi）。

## 5.2.6 运行结果

好了，至此我们就可以测试下上面的成果了（以 d:/cgi-bin/为例）：

（1） 在 IIS 或 Apache 下配置好了目录 d:/cgi-bin/具有执行执行权限，并重新启动

（2） 将编译结果结果（即 demo.cgi）生成到了 d:/cgi-bin/目录中，或拷贝到该目录。

（3） 打开IE 浏览器，在URL中输入地址：http://127.0.0.1/cgi-bin/demo.cgi

**说明：**

有关如何在 IIS 或 Apache 下运行脚本，请参阅相关手册。

# 第6章 调试

eyBuild 开发环境本身没有直接提供调试环境，但根据 C 编译器依然有目前有两种方法可以对生成的脚本程序进行调试：用 ebSetDebug()调试、异常中断调试。

## 6.1 用 ebSetDebug()调试

就是在 cgimain()中通过 ebSetDebug()函数预置 CGI 环境变量，然后逐步根跟踪调试，这种方法虽然具有一定的局限性（如调试文件上传），但已经能满足大多数需要。

## 6.2 异常中断调试

异常中断调试法是运行时调试 CGI 程序的最有效的方法（如调试文件上传，不以通过预置环境来调试）。原理是，故意构造异常（如让指针引用 0 地址中的变量），然后让操作系统和调试环境（如 VC++）捕捉该异常并暂停（如同设置了一个断点）。通过将修改寄存器的值，使程序能够进一步运行后，再单步跟踪程序进行调试。

下面给出在 Windows + VC++的环境下通过“异常中断”来调试 CGI 程序的示例，源程序可参考 $(EYBUILD_BASE)/project/demo/cgimain.c：

1. 在希望设置断点处手工添加能够导致程序异常的代码片断，类似如下程序:

```
#if 1/* for debug */
{int v=0, *p1=&v, *p2=NULL;

*p2 = 0;}
#endif
```

2. 编译并将 CGI 文件直接生成 cgi-bin 目录。注意，不是“拷贝到”，否则将无法关联到源程序。不要运行该 CGI 程序，因为程序执行会异常。

3. 从浏览器上打开该 CGI 程序，如 http://127.0.0.1/cgi-bin/demo.cgi。这时Windows 会弹出异常报告窗口了，如果你安装了 VC++，那么在异常报告的窗口左下角就会有个“调试”按钮。

4. 点击“调试”按钮用 VC++调试它，用 VC++打开后，断点就会停在“*p2 = 0;”所在行。

5. 从 View ->Debug Windows ->Registers 菜单打开寄存器窗口，这时你会发现其中的值，类似如下：

| | | | | | |
|---|---|---|---|---|---|
| EAX | = | 0012FD1C | EBX | = | 7FFD9000 |
| ECX | = | 00000000 | EDX | = | 00000001 |

| | | | | | |
|---|---|---|---|---|---|
| ESI | = | 00000000 | EDI | = | 0012FD20 |
| EIP | = | 0040130F | ESP | = | 0012FCC8 |

6. 容易发现第二行 ECX = 00000000 ，它就是指针 p2 的值(因为被我们初始化它为 NULL )。而 EAX 寄存器中的值就是变量 v 的地址，这一点可以通查看指针变量 p1 的值证实。

7. 为了让程序能正常运行，可以通过让指针 p2 也指向变量 v，方法是将 ECX 中的值修改成 EAX 中的值（即，先将光标定位到 ECX 的值处，再将值改为 EAX 寄存器中的值）。

8. 从调试工具栏中选择下一步。OK！ 程序可以单步向下执行了。

# 附录 A，如何安装和配置 FastCGI 环境

支持FastCGI的WEB服务器很多，这里选择Linux和Windows的Apache 2 环境为例，有关FastCGI的基础知识请参阅 FastCGI Developer's Kit和FastCGI Programmer's Guide。

在安装之前,请先打开FastCGI的官方网站(见http://www.fastcgi.com/dist/ )下载最新版的FastCGI 开发包 fcgi-2.4.0.tar.gz 和 FastCGI 的 Apache 模块 mod_fastcgi-2.4.2.tar.gz（Windows 环境下直接下载编译好的mod_fastcgi-2.4.2-AP20.dll）。

1、 在 Windows 环境下安装 FastCGI

（1） 解压开fcgi-2.4.0.tar.gz到一个目录，如放在E:\fcgi-2.4.0

（2） 从命令行切换到目录 E:\fcgi-2.4.0：

cd /d E:\fcgi-2.4.0

（3） 从命令行编译 FastCGI 工具包：

nmake –f Makefile.nt

（4） 这时，就会在 E:\fcgi-2.4.0\libfcgi\Release 目录生成两个文件“libfcgi.dll”和“libfcgi.lib”

（5） 将“libfcgi.dll”复制到 Windows 系统目录（如 system32 目录下），因为每个 FastCGI 应用程序执行时都会调用该动态库。

（6） 编译 FastCGI 应用程序时需要连接静态库“libfcgi.lib”，将它复制到一个特定的目录中，以方便所有项目引用。

（7） E:\fcgi-2.4.0\include 是 FastCGI 开发包的头文件，将它复制到一个特定的目录中，以方便所有项目引用。

（8） 将下载好的 mod_fastcgi-2.4.2-AP20.dll 复制到 Apache2 的 module 目录中

说明：$(EYBUILD_BASE)/fastcgi/ 目录就是按以上步骤定做好的 FastCGI，在 Windows 环境下设置好环境变量 FASTCGI_BASE 后可以直接使用。

2、 在 Linux 环境下安装 FastCGI

（1） 解压开fcgi-2.4.0.tar.gz：tar –xzf fcgi-2.4.0.tar.gz

（2） 执行配置和安装命令：

```
./configure
make install
```

（3） OK，安装完毕。这时头文件和库文件已经被复制到系统目录。

（4） 解压开mod_fastcgi-2.4.2.tar.gz：

```
$tar –xzf    mod_fastcgi-2.4.2.tar.gz
$cd mod_fastcgi-2.4.2
$cp Makefile.AP2 Makefile
$make
$make install
```

（5） 如果 Apache 的安装目录非“/usr/local/apache2”的话，则需要手工将“.libs/mod_fastcgi.so”复制到 Apache 的 module 目录中去。

3、 在 Apache 上配置 FastCGI

（1） 打开 Apache 配置文件“httpd.conf”

（2） 添加 LoadModule fastcgi_module modules/mod_fastcgi-2.4.2-AP20.dll

（3） 添加 AddHandler fastcgi-script .fcgi

（4） 添加 FastCGI 目录及权限：

```
ScriptAlias /fcgi-bin/ "F:/website/fcgi-bin/"
<Directory "F:/website/fcgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

（5） 重新启动 Apache

4、 测试 FastCGI 环境

（1） 在 Windows 或 Linux 环境下打开 $EYBUILD_BASE/project/目录中的项目文件，调整编译选并将文件生成到上面配置的目录中（如生成 demo.fcgi）

（2） 打开浏览器，输入地址：http://127.0.0/fcgi-bin/demo.fcgi

（3） 重复执行该地址，观察计数器是否在不断增加。