Perl CGI编程 Perl 进程管理

Perl 包和模块

Perl 中每个包有一个单独的符号表，定义语法为：

```
package mypack;
```

此语句定义一个名为 **mypack** 的包，在此后定义的所有变量和子程序的名字都存贮在该包关联的符号表中，直到遇到另一个 **package** 语句为止。

每个符号表有其自己的一组变量、子程序名，各组名字是不相关的，因此可以在不同的包中使用相同的变量名，而代表的是不同的变量。

从一个包中访问另外一个包的变量，可通过" 包名 + 双冒号(::) + 变量名 " 的方式指定。

存贮变量和子程序的名字的默认符号表是与名为 **main**的包相关联的。如果在程序里定义了其它的包，当你想切换回去使用默认的符号表，可以重新指定main包：

```
package main;
```

这样，接下来的程序就好象从没定义过包一样，变量和子程序的名字象通常那样存贮。

以下实例中文件有 main 和 Foo 包。 特殊变量 **__PACKAGE__** 用于输出包名：

实例

```
#!/usr/bin/perl
# main 包
$i = 1;
print "包名 : " , __PACKAGE__ , " $i\n";
package Foo;
# Foo 包
$i = 10;
print "包名 : " , __PACKAGE__ , " $i\n";
package main;
# 重新指定 main 包
$i = 100;
print "包名 : " , __PACKAGE__ , " $i\n";
print "包名: " , __PACKAGE__ , " $Foo::i\n";
1;
```

执行以上程序，输出结果为:

```
包名 : main 1
包名 : Foo 10
包名 : main 100
包名: main 10
```

BEGIN 和 END 模块

Perl语言提供了两个关键字：BEGIN，END。它们可以分别包含一组脚本，用于程序体运行前或者运行后的执行。

语法格式如下：

```
BEGIN { ... }
END { ... }
BEGIN { ... }
END { ... }
```

- 每个 **BEGIN** 模块在 Perl 脚本载入和编译后但在其他语句执行前执行。
- 每个 **END** 语句块在解释器退出前执行。
- **BEGIN** 和 **END** 语句块在创建 Perl 模块时特别有用。

如果你还不大理解，我们可以看个实例：

实例

```
#!/usr/bin/perl
package Foo;
print "Begin 和 Block 实例\n";
BEGIN {
print "这是 BEGIN 语句块\n"
}
END {
print "这是 END 语句块\n"
}
1;
```

执行以上程序，输出结果为:

```
这是 BEGIN 语句块
Begin 和 Block 实例
这是 END 语句块
```

什么是 Perl 模块？

Perl5 中用Perl包来创建模块。

Perl 模块是一个可重复使用的包，模块的名字与包名相同，定义的文件后缀为 **.pm**。

以下我们定义了一个模块 Foo.pm，代码如下所示：

实例

```
#!/usr/bin/perl
package Foo;
sub bar {
print "Hello $_[0]\n"
```

```
}
sub blat {
print "World $_[0]\n"
}
1;
```

Perl 中关于模块需要注意以下几点：

- 函数 **require** 和 **use** 将载入一个模块。
- **@INC** 是 Perl 内置的一个特殊数组，它包含指向库例程所在位置的目录路径。
- **require** 和 **use** 函数调用 **eval** 函数来执行代码。
- 末尾 **1;** 执行返回 TRUE，这是必须的，否则返回错误。

Require 和 Use 函数

模块可以通过 **require** 函数来调用，如下所示：

实例

```
#!/usr/bin/perl
require Foo;
Foo::bar( "a" );
Foo::blat( "b" );
```

也可以通过 use 函数来引用：

实例

```
#!/usr/bin/perl
use Foo;
bar( "a" );
blat( "b" );
```

我们注意到 require 引用需要使用包名指定函数，而 use 不需要，二者的主要区别在于：

- 1、require用于载入module或perl程序(.pm后缀可以省略，但.pl必须有)
- 2、Perl use语句是编译时引入的，require是运行时引入的
- 3、Perl use引入模块的同时，也引入了模块的子模块。而require则不能引入，要在重新声明
- 4、USE是在当前默认的@INC里面去寻找,一旦模块不在@INC中的话,用USE是不可以引入的，但是require可以指定路径
- 5、USE引用模块时，如果模块名称中包含::双冒号，该双冒号将作为路径分隔符，相当于Unix下的/或者Windows下的\。如：

```
use MyDirectory::MyModule
```

通过添加以下语句 use 模块就可以从模块中导出列表符号：

```
require Exporter;
@ISA = qw(Exporter);
```

@EXPORT数组包含默认导出的变量和函数的名字：

```
package Module;

require Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(bar blat);  # 默认导出的符号

sub bar { print "Hello $_[0]\n" }
sub blat { print "World $_[0]\n" }
sub splat { print "Not $_[0]\n" }  # Not exported!

1;
```

创建 Perl 模块

通过 Perl 分发自带的工具 h2xs 可以很简单的创建一个 Perl 模块。

你可以在命令行模式键入 h2xs 来看看它的参数列表。

h2xs 语法格式：

```
$ h2xs -AX -n  ModuleName
```

参数说明：

- **-A** 忽略 autoload 机制
- **-X** 忽略 XS 元素
- **-n** 指定扩展模块的名字

例如，如果你的模块在 **Person.pm** 文件中，使用以下命令：

```
$ h2xs -AX -n Person
```

执行以上程序将输出：

```
Writing Person/lib/Person.pm
Writing Person/Makefile.PL
Writing Person/README
Writing Person/t/Person.t
```

```
Writing Person/Changes
Writing Person/MANIFEST
```

Person 目录下你可以看到新增加的目录及文件说明:

- README ：这个文件包含一些安装信息，模块依赖性，版权信息等。
- Changes ：这个文件作为你的项目的修改日志（changelog）文件。
- Makefile.PL ：这是标准的 Perl Makefile 构造器。用于创建 Makefile.PL 文件来编译该模块。
- MANIFEST ：本文件用于自动构建 tar.gz 类型的模块版本分发。这样你就可以把你的模块拿到 CPAN 发布或者分发给其他人。它包含了你在这个项目中所有文件的列表。
- Person.pm ：这是主模块文件，包含你的 mod_perl 句柄代码（handler code）。
- Person.t ：针对该模块的一些测试脚本。默认情况下它只是检查模块的载入，你可以添加一些新的测试单元。
- t/ ：测试文件
- lib/ ：实际源码存放的目录

你可以使用 tar (Linux 上) 命令来将以上目录打包为 Person.tar.gz。

安装 Perl 模块

我们可以对刚才压缩的 **Person.tar.gz** 文件进行解压安装，执行步骤如下：

```
tar xvfz Person.tar.gz
cd Person
perl Makefile.PL
make
make install
```

首先运行 "perl Makefile.PL" 在当前目录生成 Makefile；
然后运行 "make" 编译并创建所需的库文件；
之后用 "make test" 测试编译结果是否正确；最后运行 "make install" 将库文件安装到系统目录，至此整个编译过程结束。

← Perl CGI编程 Perl 进程管理 →