

Table of Contents

home	1.1
入门	1.2
面向对象	1.3
调试	1.4
开源库	1.5
寻找库	1.5.1
编译安装	1.5.2
示例	1.5.3
解释器	1.6
安装	1.6.1
环境介绍	1.6.2
H2XS	1.7
示例	1.7.1
xs文件编码说明	1.7.2
xs编译工具	1.7.3
经验	1.8
checklist	1.8.1
内存	1.8.2
杂项	1.8.3
性能分析	1.8.4
apache	1.8.5
DBIx	1.8.6

- [1. Perl-Programming-experience](#)

1. Perl-Programming-experience

perl简单易用（语言表达），但是要用好（编程经验），也是一件难事。本章节主要介绍perl有关的一系列技术门路，文章内容偏向于只表达关键信息，并不对语言表达做过多介绍，也不扩散介绍其它技术。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 10:55:01

- 1. 学习
 - 1.1. 基础
 - 1.2. 详细文档
 - 1.3. 学习心得
- 2. 关键点
 - 2.1. 运行平台
 - 2.2. 了解perl基础语法
- 3. 练习
 - 3.1. 习题1
 - 3.2. 习题2

1. 学习

1.1. 基础

<http://www.runoob.com/perl/perl-tutorial.html>

1.2. 详细文档

<https://perldoc.perl.org/>

1.3. 学习心得

在面对海量资料时，首先要做的是“心平气和”。静下心来寻找文档的starting page，一般都会系统性的引导阅读。

2. 关键点

• 2.1. 运行平台

<http://www.perl.org/get.html>

支持Windows、Unix/Linux、Mac OSX

• 2.2. 了解perl基础语法

至少包括这些关键词或运算符的用法：use、require、sub、my、local、our、state、\$、%、@、qw、

\$@、\$_、@_、=>、.、&、\、die、if、elsif、unless、last、next、for、foreach、each、while、ref、scalar、eval、undef、delete、return、exists、defined、map、eq、ne、ge、le、lt、gt、==、!=、>=、<=、<、>、shift、push、pop、keys、=~、!~、m//、s//、//=、||=、+、-、*、/、++、--、**、__PACKAGE__、bless、package、use base、constant

3. 练习

3.1. 习题1

任意构造一个hash结构，将其每个key的值，放到一个数组，并按照字符串顺序排序，输出。

编写的脚本支持一个参数：指定排序是倒序还是顺序。

3.2. 习题2

任意构造一个hash，至少具备以下几个key：

key1 => "123d456e",

key2 => "23d45e",

key3 => "789d456e"

要求对hash每一个key做遍历，编写正则匹配，要求正确匹配上key1和key3，过滤掉key2。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:00:55

- 1. 为何用面向对象
- 2. perl中的面向对象
- 3. 扩展
 - 3.1. 建议

1. 为何用面向对象

仅从perl语言来看，在大型工程应用中，perl官方社区推荐的最佳实践是使用其面向对象编程。

2. perl中的面向对象

要初步掌握perl面向对象的编程特性，请仔细阅读：<https://perldoc.perl.org/perlobj.html>

3. 扩展

面向对象编程是软件开发中的一种编码实现方式，不只是perl语言。仅仅学习perl语言的面向对象的一些语法或关键词，并不能写好面向对象的代码。

3.1. 建议

- 面向对象的哲学核心在于抽象总结。对抽象总结，有很多种方法，如果您没有任何这类抽象总结的经验，有这样一个简单的建议：
- 对于多个事物，如果同属于某个集合，可以抽象一个集合类。例如身份证、账户余额、姓名，都属于某个客户，那就抽象一个客户类。
- 对于多个事物，如果有共同点，可以提取一个公共类。例如服务器需要接收数据，客户端需要发送数据，而在这个行为上有一个共同点，即客户端和服务器都要处理相同的数据（通常把这个叫做通信协议），因此可以抽象一个协议类。
- 对于任何时候，优先思考封装，例如需要设定一个永久变量，优先考虑作为类成员，而不是全局变量。
- 对于任何时候，优先思考局部化，即将大问题逐步分解。例如写一个服务器时，需要响应请求并处理请求，在编码时，应该将处理请求的整个过程作一个完整推演，并作为一个持续性编码任务，然后再逐个编码请求内容的处理细节，如果处理过程中有需要做数据存取，则又将数据存取作为持续性编码任务，通常这种思维叫做分层。非局部化的思维是：直接编码接受请求并处理请求。上述例子中就是把请求响应过程做逐步分解，而之所以要分层编码，是要编码者对各层严格定义输入输出，简化各个处理逻辑。
- 对接口编程，而不是对实现编程：如管理配置文件模块设计时，需要解析配置文件的内容，得到固定格式的配置文件结构表示，而配置文件内容有好几种形式（json、ini、...）。此时对接口编程的做法是，设计一个配置解析类，提供解析的接口，管理模块只调用该类的解析接口；对实现编程的做法是，管理模块里区分配置文件类型，调用不同的配置文件解析类。总而言之，当上层模块出现一类调用需求时（本例中是配置解析），需要进一步抽象为统一的接口，由接口内部实现具体的处理分类。
- 优先考虑对象组合，而不是继承：有一个通讯接口类，现在要增加另一种通讯协议，使用该通讯接口传递数据。继承的做法是：新建一个通讯协议类，继承自通讯接口类，再实现通讯协议；对象组合的做法是：新建一个通讯协议类，使用通讯接口类实例化一个对象，作为通讯协议类的类对象成员。

- 面向对象设计是一件伤神但是又很快乐的事情，往往一不留神，设计不合理就会引来诸多维护难题。已经有人总结了若干最佳实践，书名叫《设计模式》，其将事物分围三个大类：创建、结构、行为。各个事物始于创建，若干事物在特定问题里组成了结构，而后这些事物会发生行为。值得一提的是，如果没有太多面向对象编码经验去看这本书，会理解地云里雾里。但是建议顺着思路做一些思考，以便在未来面向对象编码的道路上能对自己有一些约束力。当完成一些设计后，针对遇到的设计问题，将《设计模式》当成手册来查询，以寻找最佳设计。如此反复提升。

- 学会画图和看图，在表达面向对象设计时，常用uml建模：

<http://www.uml.org.cn/oobject/201211231.asp>

<http://www.cnblogs.com/wolf-sun/p/3420120.html>

上述链接介绍了大部分uml知识，但要想真正掌握，还是要靠练习，即多看、多画。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-16 20:50:25

- 1. 常规调试
 - 1.1. perl -d 脚本文件
 - 1.2. 在代码中打印信息
 - 1.3. 使用die抛出异常
- 2. 高级调试
 - 2.1. 动态调试---gdb
 - 2.2. 动态调试----远程
- 3. 参考网址

perl调试详细文档: <https://perldoc.perl.org/perldebug.html>

1. 常规调试

• 1.1. perl -d 脚本文件

常用调试命令:

n----- 执行到当前作用域下一条语句

s----- 单步执行

b 函数名----- 给某函数打断点

b 行数----- 给某一行打断点

f 文件绝对路径----- 跳到某个源文件

c ----- 执行到断点或结束

c 行数 ----- 执行到某一行

p 变量名或表达式 ----- 输出结果

x 变量名 ----- 输出变量详细信息 (通常使用这个命令打印变量)

• 1.2. 在代码中打印信息

```
use Data::Dumper;
```

```
print Dumper(表达式或变量); 即可打印信息
```

打印堆栈

```
use Carp;
```

```
print Carp::longmess();
```

或

```
print caller(0);
```

• 1.3. 使用die抛出异常

perl -MDevel::SimpleTrace 脚本文件

当脚本有die抛出时，可捕获到出错堆栈

2. 高级调试

• 2.1. 动态调试----gdb

出问题时，当perl进程正在运行，重启可能破坏现场，那么就可以借助gdb对perl进行调试：

gdb -p 进程pid

执行perl语句：call Perl_eval_pv(Perl_get_context(), "my \$stack = caller(0); print(\$stack);", 0)

上述语句会打印perl当前调用堆栈

• 2.2. 动态调试-----远程

当需要对一个fork的子进程某个逻辑做调试时，可以使用远程调试

```
call (void*)Perl_eval_pv((void*)Perl_get_context(),"eval{require Enbugger;warn  
q(stopping);$ENV{PERLDB_OPTS}='RemotePort=localhost:4000';Enbugger->stop.};print STDERR $@",0)
```

需要开一个远程终端监听localhost:4000端口

3. 参考网址

<https://www.slideshare.net/hiro31/inspect-runningperl>

<https://metacpan.org/pod/App%3a%3aStacktrace>

<https://github.com/ahiguti/bulkdbg>

<http://search.cpan.org/~jjore/Enbugger-2.016/lib/Enbugger.pod>

http://search.cpan.org/~shay/mod_perl-2.0.10/docs/devel/debug/c.pod

<http://search.cpan.org/~stas/Debug-FaultAutoBT-0.02/FaultAutoBT.pm>

<http://search.cpan.org/~jjore/Internals-CountObjects-0.05/lib/Internals/CountObjects.pm>

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:03:02

对于开源库的建议：

不完全信赖开源库，也不完全闭门造车。取舍在于各个方案对于实现目标的性价比优劣。

比如，如果是一个简单的功能，闭门造车需要2天，且问题较少。那就没有必要一定找个开源库来使用。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-16 20:41:22

- 1. 寻找库
 - 1.1. step1: 搜索关键词
 - 1.2. step2: 查看描述，选择一个查看使用说明
 - 1.3. step3: 查看接口的使用介绍是否符合需求
 - 1.4. step4: 下载源码

1. 寻找库

perl有一个庞大的开源社区，里面有形形色色的package: <http://search.cpan.org/>

以搜索ping功能有关的package为例：

• 1.1. step1: 搜索关键词



• 1.2. step2: 查看描述，选择一个查看使用说明



• 1.3. step3: 查看接口的使用介绍是否符合需求

SYNOPSIS

```
use Net::Ping;

$p = Net::Ping->new();
print "$host is alive.\n" if $p->ping($host);
$p->close();

$p = Net::Ping->new("icmp");
$p->bind($my_addr); # Specify source interface of pings
foreach $host (@host_array)
{
    print "$host is ";
    print "NOT " unless $p->ping($host, 2);
    print "reachable.\n";
    sleep(1);
}
$p->close();

$p = Net::Ping->new("tcp", 2);
# Try connecting to the www port instead of the echo port
$p->port_number(scalar(getservbyname("http", "tcp")));
while ($stop_time > time())
{
```

• 1.4. step4: 下载源码



Download:

[Net-Ping-2.66.tar.gz](#)

[Dependencies](#)

[Annotate this POD](#)

[Related Modules](#)

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-17 09:05:38

- 1. 在线安装
- 2. 离线安装
 - 2.1. 安装到其它环境
 - 2.2. 自制安装脚本
- 3. 库依赖
- 4. 库依赖批量安装

1. 在线安装

参考: <http://search.cpan.org/~andk/CPAN-2.16/lib/CPAN.pm>

2. 离线安装

从cpan下载的开源库，有两种方式编译：**Makefile.PL**或**Build.PL**

参考: <http://search.cpan.org/~andk/CPAN-2.16/lib/CPAN.pm>

perl Makefile.PL perl Build.PL

```
make                ./Build
make test           ./Build test
make install        ./Build install
```

更多编译参数，参考：

<http://search.cpan.org/~leont/Module-Build-0.4224/lib/Module/Build.pm>

<http://search.cpan.org/~bingos/ExtUtils-MakeMaker-7.34/lib/ExtUtils/MakeMaker.pm>

2.1. 安装到其它环境

如果有这种场景：当前环境只是编译环境，而实际需要將开源库安装到另一个运行环境才能使用。

可以通过指定**Makefile.PL**或**Build.PL**的参数来实现。

perl Build.PL --destdir=\$DestDir

perl Makefile.PL INSTALL_BASE=\$INSTALL_BASE

2.2. 自制安装脚本

针对离线安装，自制cpan_module_install.sh，仅供参考：

```
#!/bin/bash

function log_cmd()
{
    cmd=$1
```

```

echo "*****"
echo $cmd
$cmd || (echo "$1 failed" && exit 1)
}

function install_init()
{
    ModuleName=$1
    if [ ! -d "/var" ]; then
        log_cmd "mkdir /var"
    fi
    log_cmd "tar zxvf $ModuleName -C /var"
    log_cmd "cd /var/${ModuleName%.tar.gz}"
}

function install_clean()
{
    ModuleName=$1
    cd -
    log_cmd "rm -rf /var/${ModuleName%.tar.gz}"
    echo "install success: ${DestDir:=/}"
}

function build()
{
    DestDir=$1
    if [ -z $DestDir ]; then
        log_cmd "perl Build.PL --install_base=/usr --install_path arch=/usr/lib/perl5"
    else
        log_cmd "perl Build.PL --install_base=/usr --destdir=$DestDir --install_path arch=/usr/lib/perl5"
    fi
    log_cmd "./Build"

    if [ -z $DestDir ]; then
        log_cmd "./Build test"
    fi

    log_cmd "./Build install"
}

function makefile()
{
    DestDir=$1
    INSTALL_BASE=/usr
    if [ ! -z $DestDir ]; then
        INSTALL_BASE=$DestDir/usr
    fi
    log_cmd "perl Makefile.PL INSTALL_BASE=$INSTALL_BASE INSTALLARCHLIB=\${INSTALL_BASE}/lib/perl5 INSTALLSITEARCH=\${INSTALL_BASE}/lib/perl5 INSTALLVENDORARCH=\${INSTALL_BASE}/lib/perl5"

    log_cmd "make"
    if [ -z $DestDir ]; then
        log_cmd "make test"
    fi
    log_cmd "make install UNINST=1"
}

#####
function print_help()
{
    echo "Usage: $0 -s aaa.tar.gz"
    echo "    $0 -s aaa.tar.gz -d /var/install_tmpdir"
    echo "    -s the package file"
    echo "    -d specify the folder to do a temp install"
}

function main()
{
    if [ $# -eq 0 ]; then
        print_help
        exit 1
    fi
}

```

```

while getopts "s:d:h" arg #选项后面的冒号表示该选项需要参数
do
    case $arg in
        s)
            ModuleName=$OPTARG
            ;;
        d)
            DestDir=$OPTARG
            ;;
        h)
            print_help
            exit 1
            ;;
        ?) #当有不认识的选项的时候arg为?
            print_help
            exit 1
            ;;
    esac
done
if [ -z $ModuleName ]; then
    print_help
    exit 1
fi

install_init $ModuleName || exit 1
if [ -f "Build.PL" ]; then
    build $DestDir
elif [ -f "Makefile.PL" ]; then
    makefile $DestDir
else
    echo "该安装包不正确, 缺少Build.PL和Makefile.PL"
fi
install_clean $ModuleName
}

main $*

```

执行./cpan_module_install.sh -h即可查看使用方法

./cpan_module_install.sh -s 源码包 -d 目标路径

3. 库依赖

离线安装时，单个下载的源码库，会依赖其它库，在make test时会报错：

```

cd /var/JSON-XS-3.04
*****
perl Makefile.PL INSTALL_BASE=/usr INSTALLARCHLIB=$(INSTALL_BASE)/lib/
Can't locate Canary/Stability.pm in @INC (@INC contains: /etc/perl /usr
rl/5.14 /usr/share/perl/5.14 /usr/local/lib/site_perl .) at Makefile.P
BEGIN failed--compilation aborted at Makefile.PL line 4.
perl Makefile.PL INSTALL_BASE=/usr INSTALLARCHLIB=$(INSTALL_BASE)/lib/
ed
*****
make

```

这说明缺少Canary::Stability，需要到cpan继续搜索Canary::Stability，然后按照离线方式安装后，再安装当前模块，如果还有错误，则继续安装。

PS：如果使用cpan_module_install.sh，建议首次不指定-d参数，这样会执行make test检查错误。

4. 库依赖批量安装

当库依赖比较多时，而每次都是人工安装到当前环境，但是如果要安装到其它环境，即`cpan_module_install.sh`指定`-d`参数时，又需要重新执行一遍安装，比较麻烦。这里再提供一个自制脚本批量安装`install.sh`：

```
#!/bin/bash
#说明：该脚本用于安装有依赖的组件包，第一个参数给一个配置文件，文件内容是按照依赖顺序列出的要安装的cpan模块，如
#   a.tar.gz
#   b.tar.gz
#   ...
#如果某个包的安装选择提示，可以增加一个选择文件，文件名为：包文件名.sel，如a.tar.gz.sel
#文件内容每一行都是一个选择，编译时，会按照这些选择逐个输入，如：
#   y
#   n
#   y
#   yes
#   ...
config=$1
dest_dir=$2
if [ -z $config ]; then
    echo "Usage: $0 config_file installdir"
    exit 1
fi

for pkg in `cat $config`; do
    echo -e "\033[32m install $pkg \033[0m"
    if [ -z $dest_dir ]; then
        if [ -f $pkg.sel ]; then
            RES=`cat $pkg.sel | ./cpan_module_install.sh -s $pkg 2>&1| grep 'Result: '`
        else
            RES=`./cpan_module_install.sh -s $pkg 2>&1| grep 'Result: '`
        fi
        if [ "${RES}" == "Result: FAIL" ]; then
            echo -e "\033[31m x $RES \033[0m"
            exit 2
        fi
        echo $RES
    else
        if [ -f $pkg.sel ]; then
            cat $pkg.sel | ./cpan_module_install.sh -s $pkg -d $dest_dir
        else
            ./cpan_module_install.sh -s $pkg -d $dest_dir
        fi
    fi
done
```

该脚本第一个参数是一个配置文件，按照顺序记录依赖库的安装顺序，每个依赖库源码包一行，例如：

module.list

```
Canary-Stability-2012.tar.gz
Types-Serialiser-1.0.tar.gz
JSON-XS-3.04.tar.gz
```

执行安装：

`./install.sh module.list /home/myroot`

将会把所有模块安装到`/home/myroot`，如果不指定第二个参数，则在当前系统安装。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:07:26

暂时不写这个，时间太少了^_^

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-16 20:41:22

perl是一门解释性语言，perl解释器应用yacc实现编译，可搜索相关说明。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-16 20:41:22

- 1. 解释器安装
 - 1.1. 下载perl解释器源码
 - 1.2. 解压后查看INSTALL（耐心阅读）
 - 1.3. 根据参数说明，指定配置参数
 - 1.4. 查看环境已有的解释器编译参数

1. 解释器安装

• 1.1. 下载perl解释器源码

<http://www.cpan.org/src/README.html>

• 1.2. 解压后查看INSTALL（耐心阅读）

• 1.3. 根据参数说明，指定配置参数

```
./Configure -Dinstallusrbinperl -Dusethreads -Duselargefiles -Dccflags="-DDEBIAN -
D_FORTIFY_SOURCE=2 -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-
security -Dldflags= -Wl,-z,relro -Dlddflags=-shared -Wl,-z,relro" -Dcccdlflags=-fPIC -Darchname=x86_64-
linux-gnu -Dprefix=/usr -Dprivlib=/usr/share/perl/5.14 -Darchlib=/usr/lib/perl/5.14 -Dvendorprefix=/usr -
Dvendorlib=/usr/share/perl5 -Dvendorarch=/usr/lib/perl5 -Dsiteprefix=/usr/local -
Dsitelib=/usr/local/share/perl/5.14.2 -Dsitearch=/usr/local/lib/perl/5.14.2 -Dman1dir=/usr/share/man/man1 -
Dman3dir=/usr/share/man/man3 -Dsiteman1dir=/usr/local/man/man1 -Dsiteman3dir=/usr/local/man/man3 -
Duse64bitint -Dman1ext=1 -Dman3ext=3perl -Dpager=/usr/bin/sensible-pager -Uafs -Ud_csh -Ud_ualarm -
Uusesfio -Uusenm -Ui_libutil -DDEBUGGING=-g -Doptimize=-O2 -Duseshrplib -Dlibperl=libperl.so.5.14.2 -
des
```

make install

或者

make install DESTDIR=/home/myroot（安装到临时目录）

• 1.4. 查看环境已有的解释器编译参数

perl -V:config_args

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:09:03

解释器搜索package的规则：按照@INC指定的路径搜索

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-16 20:41:22

在某些场景下，需要在perl调用一些底层c代码，比如为了提高性能。

perl提供h2xs工具，方便封装c代码为perl接口。大致原理是：h2xs以c代码提供了perl的内部使用的数据结构，并能与c语言支持的类型进行转换，在封装c代码时，允许c代码中使用这些结构，从而实现对接perl接口调用的输入输出。

参考网址：

<https://perldoc.perl.org/index-internals.html>

<https://perldoc.perl.org/perlguts.html>

<https://perldoc.perl.org/h2xs.html>

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:09:32

- 1. H2XS示例
 - 1.1. step1, 执行: `h2xs -n Plibtest`
 - 1.2. step2, 修改`Plibtest.xs`文件实现接口
 - 1.3. step3, 编译
 - 1.4. step4, 调用接口

1. H2XS示例

• 1.1. step1, 执行: `h2xs -n Plibtest`

```
Defaulting to backwards compatibility with perl 5.14.2
If you intend this module to be compatible with earlier perl versions, please
specify a minimum perl version with the -b option.

Writing Plibtest/ppport.h
Writing Plibtest/lib/Plibtest.pm
Writing Plibtest/Plibtest.xs
Writing Plibtest/fallback/const-c.inc
Writing Plibtest/fallback/const-xs.inc
Writing Plibtest/Makefile.PL
Writing Plibtest/README
Writing Plibtest/t/Plibtest.t
Writing Plibtest/Changes
Writing Plibtest/MANIFEST
```

• 1.2. step2, 修改`Plibtest.xs`文件实现接口

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include "ppport.h"

#include "const-c.inc"
void echo(const char *str)
{
    printf("%s\n", str);
}

MODULE = Plibtest      PACKAGE = Plibtest
void
echo(str)
    char *str
PPCODE:
    echo(str);

INCLUDE: const-xs.inc
```

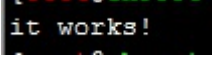
• 1.3. step3, 编译

```
cd Plibtest
perl Makefile.PL
make
```

```
make install
```

• 1.4. step4, 调用接口

```
perl -e 'use Plibtest; Plibtest::echo("it works!");'
```

结果: 

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-17 09:11:26

- 1. 原理简介
- 2. xs语句
 - 2.1. 返回一个值
 - 2.2. 返回数组
- 3. 使用perl数据结构
 - 3.1. 接口的参数输入SV
 - 3.1.1. 获取字符串
 - 3.1.2. 获取结构体
 - 3.1.3. 使用建议
- 4. 返回值
 - 4.1. 返回undef
 - 4.2. 返回SV标量
 - 4.3. 返回HV
- 5. 其它操作
 - 5.1. hash中嵌入hash
 - 5.2. 减少内存占用
- 6. 扩展

1. 原理简介

.xs文件分为两部分：

1. xs语句：用于声明封装perl接口
2. c代码：用于编写c代码逻辑，提供给xs语句做接口声明

在编译时，xs文件会被展开xs语句，最终被转换为.c文件。再经编译为so库

本章节并不覆盖h2xs有关的所有内容，只介绍一些常用示例

2. xs语句

xs语句指定封装的接口属性，放在“MODULE = Plibtest PACKAGE = Plibtest”之后

• 2.1. 返回一个值

接口支持一些常规类型，如char*，即perl语句中传入的标量。详细参考前面章节提到的文档说明。

```
I32
testfuncxxx(c, str, i, sv)
    char c
    char *str
    I32 i
    SV *sv
CODE:
    int ret = otherfunc(c, str, i, sv);
    if (ret > 10000) {
        XSRETURN_UNDEF;
    }
    RETVAL = ret;
OUTPUT:
```


说明：本例中XSRETURN_UNDEF表示返回undef值

• 2.2. 返回数组

```
void
_load(file)
    char *file
PREINIT:
    char *perror = PERIOR_NO;
    HV *hv      = NULL;
PCODE:
    EXTEND(SP, 0);
    hv = ini_load(file, &perror);
    if (hv == NULL) {
        logerr("ini_load", "NULL");
        mXPUSHs(&PL_sv_undef);
    } else {
        mXPUSHs(newRV((SV *)hv));
    }
    if (perror) {
        logerr("_load", "perror");
        mXPUSHs(newSVpv(perror, 0));
    }
}
```

说明：本例中mXPUSHs(&PL_sv_undef)表示返回一个undef值（不能把NULL当做undef，否则会出core）。

newRV((SV *)hv)是返回hash引用，不能直接返回HV。

3. 使用perl数据结构

常用结构：SV、HV、AV

3.1. 接口的参数输入SV

• 3.1.1. 获取字符串

```
char * str = SvPV(sv, len);
```

• 3.1.2. 获取结构体

```
struct xxx *st = (struct xxx *)SvPVbyte(sv, len);
```

```
inline static stcs_h *sv_to_stcs_h(SV *h_sv)
{
    if (h_sv) {
        STRLEN len = 0;
        stcs_h **h = NULL;
        if (SvPOK(h_sv) || SvPOKp(h_sv)) {
            h = (stcs_h **)SvPVbyte(h_sv, len);
            if (h && len == sizeof(stcs_h *)) {
                return *h;
            }
        }
    }
}
```

```

    }
    logerrf("SvPVbyte failed", "len=%ld", len);
}

//logerrf("SvPOK failed");
}
return NULL;
}

```

• 3.1.3. 使用建议

- 尽量使用SvPVbyte，这个只针对内存字节数而言的，而SvPV可能将字符串中的特殊字符进行转义。
- 判断需要加上SvPOK(h_sv) || SvPOKp(h_sv)，以确认是sv标量，另，不能只判断SvPOK，perl中sv标量具有public和private属性。在运行环境中，标量可能呈现private属性，此时调用接口，SvPOK就判断为false，而SvPOKp为true。

4. 返回值

• 4.1. 返回undef

- 在xs语句中XSRETURN_UNDEF
- SV *svundef = newSV(0); RETVAL=sv_undef

• 4.2. 返回SV标量

```

SV *sv = newSVpv(str, 0);

RETVAL = sv;

```

• 4.3. 返回HV

```

HV *hash = (HV *)sv_2mortal((SV *)newHV());
RETVAL = hash;

```

5. 其它操作

• 5.1. hash中嵌入hash

```

...hash 是HV *变量
HV *hsec = (HV *)sv_2mortal((SV *)newHV());

if (!hsec) {
    return NULL;
}

SV *rv = newRV((SV *)hsec);
const char * key = "test";

```

```
if (!hv_store(hash, key, strlen(key), rv, 0)) {  
    SvREFCNT_dec(rv);  
    hv_undef(hsec);  
    return NULL;  
}
```

• 5.2. 减少内存占用

```
...sv是SV*变量  
RETVAL = newRV((SV *)sv_2mortal(sv))
```

注：如果直接返回`hv`或者`sv`，在超过作用域后，内存不会立即释放，这会导致内存占用。使用`sv_2mortal`是延迟释放`sv`的内存占用，在超过作用域时，会自动释放。

6. 扩展

perl支持直接在perl语言中操作结构体，即`pack/unpack`，详见：<https://perldoc.perl.org/functions/pack.html>

有时候，利用这个特性，能和`h2xs`更好配合使用

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:18:56

- 1. h2xs_module_install

1. h2xs_module_install

```
#!/bin/bash

function log_cmd()
{
    cmd=$1
    echo "*****"
    echo $cmd
    $cmd || (echo "$1 failed" && exit 1)
}

function makefile()
{
    DestDir=$1
    INSTALL_BASE=/usr
    if [ ! -z $DestDir ]; then
        INSTALL_BASE=$DestDir/usr
    fi
    log_cmd "perl Makefile.PL INSTALL_BASE=$INSTALL_BASE INSTALLARCHLIB=\${INSTALL_BASE}/lib/perl5 INSTALLSITEARCH=\${INSTALL_BASE}/lib/perl5 INSTALLVENDORARCH=\${INSTALL_BASE}/lib/perl5"

    log_cmd "make"
    if [ -z $DestDir ]; then
        log_cmd "make test"
    fi
    log_cmd "make install UNINST=1"
}

#####
function print_help()
{
    echo "Usage: $0 -s ./libaaa"
    echo "    $0 -s ./libaaa -d /var/install_tmpdir"
    echo "    -s the package path"
    echo "    -d specify the folder to do a temp install"
}

function main()
{
    if [ $# -eq 0 ]; then
        print_help
        exit 1
    fi

    local ModuleName=
    local DestDir=
    while getopts "s:d:h" arg #选项后面的冒号表示该选项需要参数
    do
        case $arg in
            s)
                ModuleName=$OPTARG
                ;;
            d)
                DestDir=$OPTARG
                ;;
            h)
                print_help
                exit 1
                ;;
            ?) #当有不认识的选项的时候arg为?
                print_help
                exit 1
                ;;
        esac
    done
    if [ -z $ModuleName ]; then
        print_help
    fi
}
```

```
        exit 1
    fi

    cd $ModuleName
    if [ -f "Makefile.PL" ]; then
        makefile $DestDir
    else
        echo "该安装包不正确，缺少Makefile.PL"
    fi
    cd -
}

main $*
```

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-17 09:19:21

使用\$1等变量，不要加{}，直接用\$1, \$2 ...

[原因]

\$1,\$2,...等变量是perl内置变量，只读。\${ 1 }，1会被解析为"1"，\${ 1 }会被当成symbolic reference，在use strict时，运行时会错误。

\${1}是正确的，但是代码格式化工具可能会将其格式化为\${ 1 }，因此本条checklist规定，非特殊场景，\$1,\$2...等特殊变量不能加括号{}。

详见：<http://perldoc.perl.org/perlref.html>

条件判断，需要注意空字符串""，字符0，undef，以及引用类型的变量

[原因]

perl中空字符串为假，字符0为假，undef为假，引用类型为真

每个模块必须加上：

use strict;

use warnings;

使用undef参数作为除数，会产生除0错误

[原因]

Perl里面undef在数字上下文环境中会被当做0处理。

建议判断正则表达式中模式串为"的情况

1) 使用模式串前先判断模式串是否为空串，对为空串的情况进行明确处理。确保m//模式匹配串不是空串。

[原因]

perl里面如果发现模式串为空串，会默认用最后一次匹配成功的模式串代替。所以，会让程序的行为变得很诡异。

perl多线程模型

1) perl多线程模型结合lock函数会导致死锁，禁用perl多线程。

[原因]

1.perl的多线程结合lock本身实现存在缺陷会导致线程死锁

2.线程中创建进程可能导致子进程死锁。

[解决办法]

用多进程模型代替多线程模型

取hash结构的键值，需要先判断该键值是否存在

原因说明：

当键值不存在的时候，会向hash结构中插入一个空值

使用die语句，需要在末尾加上换行符“\n”

原因说明：

die语句，会输出出错代码的文件名以及代码行，

使用反引号执行命令且开启了perl的去污染模式，要注意命令中的变量需要去污染，

原因说明：

命令中如果有变量，用反引号，perl认为引入不安全的字符，会直接的报错

去污染: `($username) = $username =~ m/(.*)/ if $username;`

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-16 20:41:22

- 1. 检测内存占用
 - 1.1. 获取所有perl对象的内存占用情况
 - 1.2. 测试大hash结构的内存占用问题
- 2. 解决内存占用
 - 2.1. 利用COW（copy-on-write）机制解决代码内存占用问题
 - 2.2. 修改perl内存管理，解决大hash内存占用问题
 - 2.2.1. 方法一：尝试修改perl的编译参数，变更内存申请接口
 - 2.2.2. 方法二：尝试引入gperftool，优化小内存管理

perl解释器为了提高性能，在内存方面有这样一个规则：申请的内存，尤其是hash结构，内存不会释放回操作系统。

会算作perl内存池资源。如果perl代码中加载了某个大的hash结构，会发现该进程的内存增长了很多，且不会减少。

1. 检测内存占用

• 1.1. 获取所有perl对象的内存占用情况

```
use Devel::Gladiator qw(walk_arena arena_ref_counts arena_table);
use Data::Dumper;
use Devel::Size qw(size total_size);
open STDOUT, ">/sf/data/local/gwl/x/vtppdaemon.arena.$$txt";
eval { $self->handle_requests(); };
my $all = walk_arena();
my $total = 0;
foreach my $sv ( @$all ) {
    eval {
        my $x = total_size($sv);
        my $xx = Dumper($sv);
        print "live object: $x, $xx\n" if $x > 2048;
        $total += $x;
    };
}
print "total_size:$total\n";
```

• 1.2. 测试大hash结构的内存占用问题

读取大hash后，即使解除引用，内存依然不会减少

创建大hash文件：[perl/jing-yan/create-big-json.pl](#)

测试读取大hash文件：[perl/jing-yan/perl-memory-test.pl](#)

2. 解决内存占用

perl内存占用主要有两个方面：储存代码， 储存数据结构。

• 2.1. 利用COW（copy-on-write）机制解决代码内存占用问题

一个大项目（使用perl），通常有数十万行代码，根据经验：1万行占用2-3MB左右，那么20万行则40-60MB。如果后端有20个进程

则内存占用是800-1200MB，可想而知，随着进程数增多，内存占用叠加增长。

考虑代码占用的内存是不变的，所以可以首先启动一个进程，加载所有代码，其它所有perl进程都从这个进程fork，则代码的内存占用理论上只有40-60MB。

技术分析：linux下，COW机制原理是，父进程申请的内存，操作系统的内存管理建立的是父进程虚拟内存（页表）和物理内存页（页框）的关系，当fork子进程时，操作系统仍然为子进程建立映射，因此父子进程共同引用相同页框，只有在修改时，内核才会复制一个页框承载修改的内存。页框是内核维护的，所以无论父子进程最终是否在相同进程组或是否还维持父子关系，对COW机制都毫无影响。

详情：</perl/jing-yan/perl-memory>

• 2.2. 修改perl内存管理，解决大hash内存占用问题

大hash问题在于每个hash子项，perl都是申请的小块内存，内核提供两种申请内存的方式，brk和mmap，小内存使用brk。

而brk申请的内存，在调用free时，是要看内存栈顶是否有内存被占用，如果有，则不会释放内存（归还操作系统），因此内存并没有归还操作系统。

2.2.1. 方法一：尝试修改perl的编译参数，变更内存申请接口

参考[解释器-安装](#)中，编译参数Uusemymalloc

2.2.2. 方法二：尝试引入gperftool，优化小内存管理

编译解释器时需要连接tcmalloc：

```
-Uusemymalloc -Dusetheads -Duselargefiles -Dlibs="-ltcmalloc_minimal -lnsl -ldl -lm -lcrypt -lutil -  
lpthread -lc -lunwind" -Dccflags="-DDEBIAN -D_FORTIFY_SOURCE=2 -g -O2 -fstack-protector --  
param=ssp-buffer-size=4 -Wformat -Werror=format-security -fno-builtin-malloc -fno-builtin-calloc -fno-  
builtin-realloc -fno-builtin-free" -Dldflags="-Wl,-z,relro" -Dldldflags="-shared -Wl,-z,relro" -Dcccdlflags=-fPIC  
-Darchname="x86_64-linux-gnu" -Dprefix=/usr -Dprivlib=/usr/share/perl/5.14 -Darchlib=/usr/lib/perl/5.14 -  
Dvendorprefix=/usr -Dvendorlib=/usr/share/perl5 -Dvendorarch=/usr/lib/perl5 -Dsiteprefix=/usr/local -  
Dsitelib=/usr/local/share/perl/5.14.2 -Dsitearch=/usr/local/lib/perl/5.14.2 -Dman1dir=/usr/share/man/man1 -  
Dman3dir=/usr/share/man/man3 -Dsiteman1dir=/usr/local/man/man1 -Dsiteman3dir=/usr/local/man/man3 -  
Duse64bitint -Dman1ext=1 -Dman3ext=3perl -Dpager=/usr/bin/sensible-pager -Uafs -Ud_csh -Ud_ualarm -  
Uusesfio -Uusenm -Uj_libutil -DDEBDEBUGGING=-g -Doptimize=-O2 -Duseshrplib -Dlibperl=libperl.so.5.14.2 -  
des
```

注：上述两种方法，笔者尚未验证成功，方法一是官方提供的内存优化方案（perl6解决了内存问题）。方法二主要是tcmalloc未起到小内存管理的作用，需要调brk申请的内存阈值和tcmalloc的小内存阈值。以及修改perl申请内存的尺寸。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:20:58

查看package的版本: `perl -e 'use 包;print $包s::VERSION."\n";'`

```
#!/usr/bin/perl

use Crypt::OpenSSL::RSA;

use Crypt::OpenSSL::Bignum;

my $n = Crypt::OpenSSL::Bignum->new_from_hex(@ARGV[0]);
my $e = Crypt::OpenSSL::Bignum->new_from_hex("010001");

my $rsa_pubkey = Crypt::OpenSSL::RSA->new_key_from_parameters($n, $e);

print "Public Key : \n".$rsa_pubkey->get_public_key_x509_string();

print "Private Key : \n".$rsa_pubkey->get_private_key_string();
```

```
#!/usr/bin/perl

use Crypt::OpenSSL::X509;

my $x509 = Crypt::OpenSSL::X509->new_from_file('a.crt');

print $x509->pubkey()."\n";
```

python调用perl: <http://search.cpan.org/dist/pyperl/perlmodule.pod>

perl调用python: <https://metacpan.org/pod/distribution/Inline-Python/Python.pod>

文件锁

```
my $fd;

if (!open($fd, "$lock_file")) {

    vtp_throw("open $lock_file failed:$!");

}

if (!flock($fd, LOCK_EX | LOCK_NB)) {

    flock($fd, LOCK_EX);

    flock($fd, LOCK_UN);

    close($fd);
```

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-16 20:41:22

详情查看 `Devel::NYTProf`

对任何需要分析的脚本执行: `perl -d:NYTProf` 脚本

在当前目录会生成 `nytprof.out`, 记录了性能数据

使用 `nytprofhtml` 即可得到 `nytprof` 文件夹, 其以 `html` 展示性能数据

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook 该文件修订时间: 2018-04-17 09:24:09

- 1. 配置mod_perl

apache是后端接入层服务器，通常perl代码由perl解释器执行，解释器进程和apache是两个独立的进程。如果要实现apache响应请求时执行perl代码，按照常规思路：

1. apache与解释器进程进行跨进程通信。但后端的perl进程需要做到高并发响应能力，因为所有请求都会阻塞在apache与解释器进程之间。
2. apache调用解释器进程执行perl代码。但这中间涉及：启动新进程（解释器进程），以及解析代码，甚至初始化一些数据结构。带来的消耗是很大的。

mod_perl就能很好的解决apache执行perl代码的问题，mod_perl以apache模块方式（动态库）封装了perl解释器。允许在apache进程中调用perl解释器。并且在apache启动时，就调用了一系列初始化行为（包括解析perl代码）。

1. 配置mod_perl

apache配置中添加

```
PerlRequire /usr/share/vtp-manager/startup.pl
<Location /vapi/>
    SetHandler perl-script
    PerlHandler REST
</Location>
```

在apache父进程初始化时，会调用startup.pl；当收到请求时，会调用REST::handler函数，参数是请求对象。

参考：<http://perl.apache.org/docs/1.0/guide/>

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间： 2018-04-17 09:24:30

- 1. DBIx
 - 1.1. 建立数据表对应的class
 - 1.2. 用DBIx::Class::Schema连接数据库
 - 1.3. 执行数据库语句（例如优化数据库配置）
 - 1.4. 构造查询器
 - 1.5. 构造查询条件
 - 1.6. 示例
- 2. 友情链接

1. DBIx

DBIx是perl操作数据库的一个开源库，支持多种数据库，例如sqlite。DBIx由多个类组成，使用过程：

假设当前工作目录是rootdir

• 1.1. 建立数据表对应的class

建立rootdir/SchemaExample/Result/User.pm

```
package SchemaExample::Result::User;
use strict;
use warnings;
use base qw/DBIx::Class::Core/;
__PACKAGE__->table('user');
__PACKAGE__->add_columns(
    id => {data_type => 'INTEGER'},
    name => {data_type => 'TEXT', is_nullable => 0}
);
__PACKAGE__->set_primary_key('id');
1
```

• 1.2. 用DBIx::Class::Schema连接数据库

```
package SchemaExample;
use strict;
use warnings;
use base qw/DBIx::Class::Schema/;
__PACKAGE__->load_classes(qw/Result::User/);
1

#参数参看开源库文档说明
my $schema = SchemaExample->connect("dbi:SQLite:/home/db/test.db",
    undef, undef, undef,
    {
        on_connect_do => [
            'PRAGMA cache_size = 20000',
            'pragma page_size = 4096',
            'pragma synchronous = 0',
            'pragma journal_mode = OFF',
            'pragma temp_store = 2',
        ]
    }
);
```

• 1.3. 执行数据库语句（例如优化数据库配置）


```
$schema->storage->dbh_do(
    sub {
        my ($storage, $dbh, @args) = @_;
        $dbh->do("CREATE TABLE if not exists user (id INTEGER PRIMARY KEY, name TEXT NOT NULL)");
    }
);
```

• 1.4. 构造查询器

```
my $rs = $schema->resultset('user');
```

• 1.5. 构造查询条件

```
my $new_rs = $rs->search_rs(条件, 选项);
```

• 1.6. 示例

```
my $hash = {
    name => 'test1'
};
my $cd = $rs->create($hash);
my $id = $cd->id;
my $name = $cd->name;
```

```
my $hash = {
    name => 'test2'
};
my $cond = {
    name => 'test1'
};
my $new_rs = $rs->search_rs($cond);
$new_rs->update($hash);
```

```
my $cond = {
    name => 'test1'
};
my $new_rs = $rs->search_rs($cond);
my $cd = $new_rs->first;
my $id = $cd->id;
```

```
my $cond = {
    id => [-and => {'>=', 1}, {'<=', 100}]
};
my $option = {
    rows => 10,
    pages => 2
};
my $new_rs = $rs->search_rs($cond, $option);
my @cds = $new_rs->all();
foreach my $cd (@cds) {
    my $name = $cd->name;
    ...
}
```

注: `search_rs`只是封装了查询结构, 并不会执行查询, 因此一个`new_rs`可以被保存, 需要查询时再执行`$new_rs->all`等操作。

2. 友情链接

如要深入了解**DBIx**, 请仔细阅读以下内容:

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class.pod>

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class/Relationship.pm>

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class/Relationship/Base.pm#condition>

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class/Schema.pm#resultset>

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class/Schema.pm>

<http://search.cpan.org/~ribasushi/DBIx-Class-0.082841/lib/DBIx/Class/ResultSet.pm#join>

<http://search.cpan.org/~ilmari/SQL-Abstract-1.84/lib/SQL/Abstract.pm>

编码技巧: 在写查询语句时, 最好实测, 例如查询**sqlite3**数据库, 可以使用**sqlite3**命令行构造数据, 并按照查询预想查询结果。

再对比**DBIx**的查询结果, 以便检查**DBIx**的代码是否写的正确。

Copyright (C) Gong Weilin 2017 all right reserved, powered by Gitbook该文件修订时间: 2018-04-17 09:26:09