

# 程序设计实践

## The Practice of Programming

### 接口(Interfaces)

# 接口 (界面)

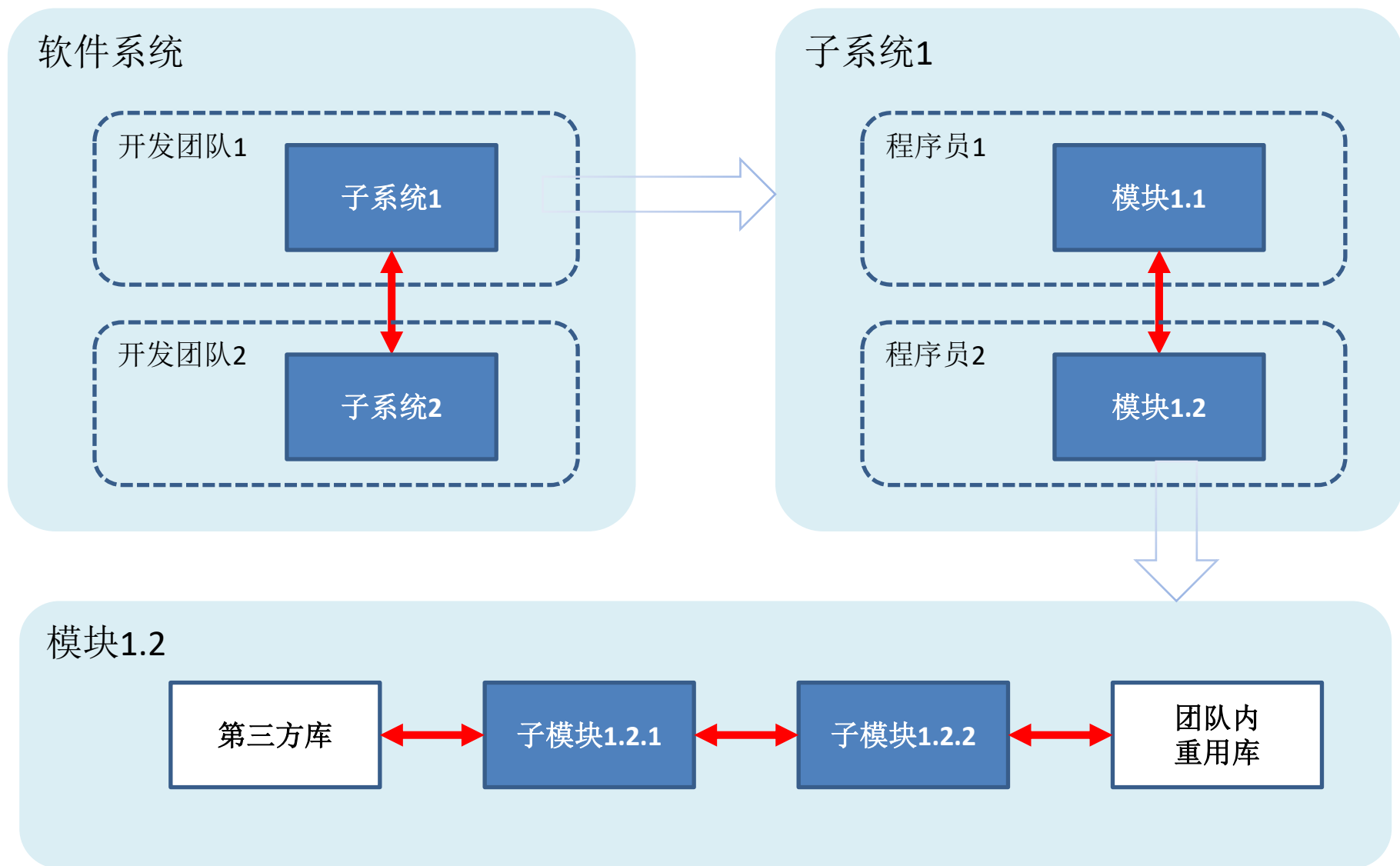
## 程序与程序之间的接口

库函数，共享数据，通信协议等

## 人与程序之间的接口

操作命令，配置文件，错误信息，日志  
图形用户界面(GUI)  
语音，视频

# 程序与程序之间的接口



# 库函数接口

hello\_world.c

```
#include <stdio.h>

void main()
{
    printf("Hello World!\n");
}
```

## 接口说明:

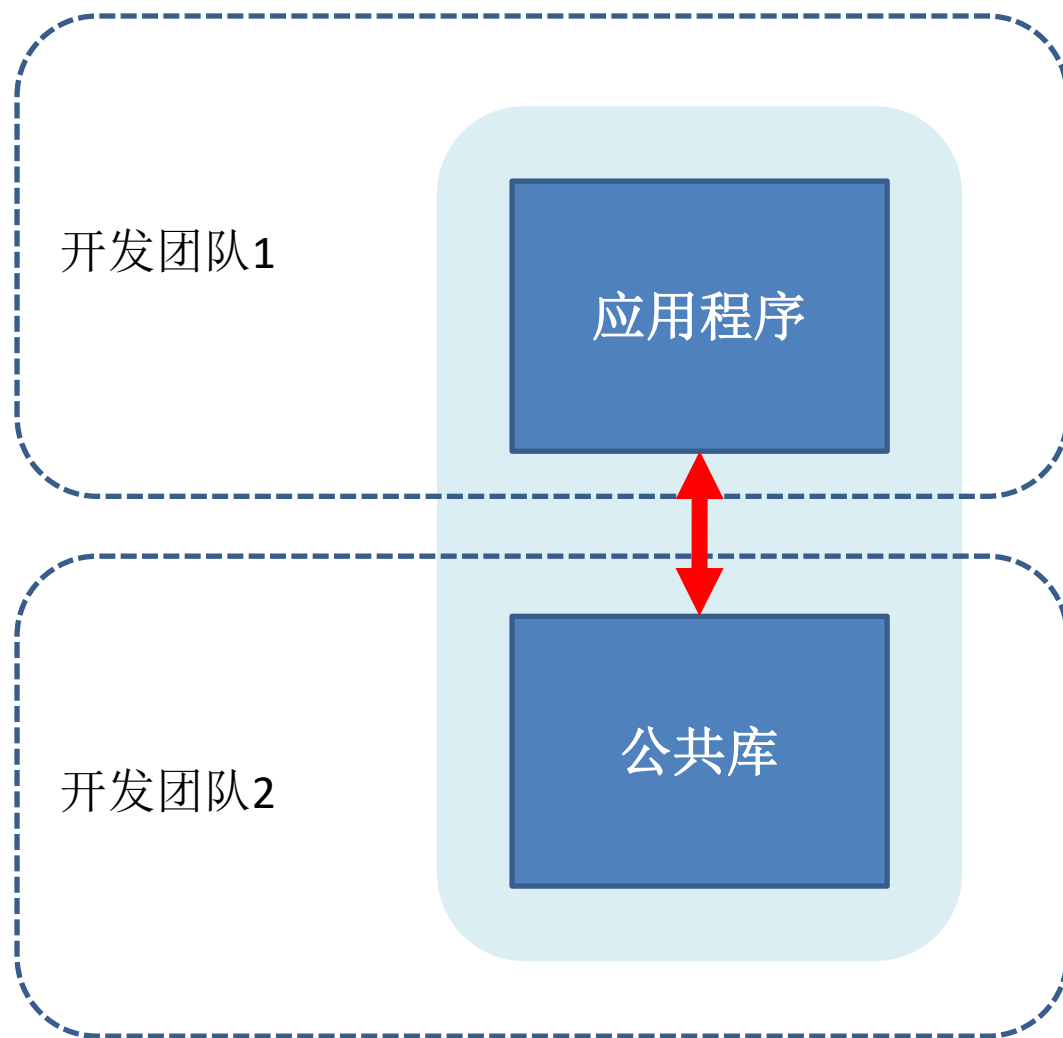
- 头文件
- 联机man
- 接口说明文档
- 编程指南

/usr/include/stdio.h (/usr/lib64/libc.a)

```
...
/* Write formatted output to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int printf (const char *__restrict __format, ...);
...
```

# 库函数接口



## 接口说明:

- 头文件
- 接口说明文档
- 编程指南

# 库函数接口 - 范例

```
QSORT(3)                                Linux Programmer's Manual                                QSORT(3)

NAME
    qsort - sorts an array

SYNOPSIS
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
            int(*compar)(const void *, const void *));

DESCRIPTION
    The qsort() function sorts an array with nmemb elements of size size. The base argument points to the start of the array.

    The contents of the array are sorted in ascending order according to a comparison function pointed to by compar, which is called with the objects being compared.

    The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

RETURN VALUE
    The qsort() function returns no value.

CONFORMING TO
    SVr4, 4.3BSD, C89, C99.

NOTES
    Library routines suitable for use as the compar argument include alphasort(3) and versionsort(3). To compare C strings, the comparison function should be strcmp(3) as shown in the example below.

EXAMPLE
    For one example of use, see the example under bsearch(3).

    Another example is the following program, which sorts the strings given in its command-line arguments:

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

static int
cmpstringp(const void *p1, const void *p2)
{
    /* The actual arguments to this function are "pointers to
       pointers to char", but strcmp(3) arguments are "pointers
       to char", hence the following cast plus dereference */

    return strcmp(* (char * const *) p1, * (char * const *) p2);
}
```

# 库函数接口 - 范例

## QDialog Class Reference

The QDialog class is the base class of dialog windows. [More...](#)

```
#include <qdialog.h>
```

Inherits [QWidget](#).

Inherited by [QColorDialog](#), [QErrorMessage](#), [QFileDialog](#), [QFontDialog](#), [QInputDialog](#), [QMessageBox](#), [QMotifDialog](#), [QProgressDialog](#), [QTabDialog](#), and [QWizard](#).

[List of all member functions.](#)

### Public Members

- explicit [QDialog](#) ( [QWidget](#) \* parent = 0, const char \* name = 0, bool modal = FALSE, WFlags f = 0 )
- [~QDialog](#) ()
- enum [DialogCode](#) { Rejected, Accepted }
- int [result](#) () const
- virtual void [show](#) ()
- void [setOrientation](#) ( Orientation orientation )
- Orientation [orientation](#) () const
- void [setExtension](#) ( [QWidget](#) \* extension )
- [QWidget](#) \* [extension](#) () const
- void [setSizeGripEnabled](#) ( bool )
- bool [isSizeGripEnabled](#) () const
- void [setModal](#) ( bool modal )
- bool [isModal](#) () const

### Public Slots

# 库函数接口 - 范例

## ØMQ/4.3.1 API Reference

[v4.3 master](#) | [v4.3 stable](#) | [v4.2 stable](#) | [v4.1 stable](#) | [v4.0 stable](#) | [v3.2 legacy](#)

- `zmq` - ØMQ lightweight messaging kernel
- `zmq_atomic_counter_dec` - decrement an atomic counter
- `zmq_atomic_counter_destroy` - destroy an atomic counter
- `zmq_atomic_counter_inc` - increment an atomic counter
- `zmq_atomic_counter_new` - create a new atomic counter
- `zmq_atomic_counter_set` - set atomic counter to new value
- `zmq_atomic_counter_value` - return value of atomic counter
- `zmq_bind` - accept incoming connections on a socket
- `zmq_close` - close ØMQ socket
- `zmq_connect` - create outgoing connection from socket
- `zmq_ctx_destroy` - terminate a ØMQ context
- `zmq_ctx_get` - get context options
- `zmq_ctx_new` - create new ØMQ context
- `zmq_ctx_set` - set context options
- `zmq_ctx_shutdown` - shutdown a ØMQ context
- `zmq_ctx_term` - terminate a ØMQ context
- `zmq_curve_keypair` - generate a new CURVE keypair
- `zmq_curve_public` - derive the public key from a private key
- `zmq_curve` - secure authentication and confidentiality
- `zmq_disconnect` - Disconnect a socket
- `zmq_errno` - retrieve value of errno for the calling thread
- `zmq_getsockopt` - get ØMQ socket options
- `zmq_gssapi` - secure authentication and confidentiality
- `zmq_has` - check a ZMQ capability



# 库函数接口 - 范例

网络SDK开发手册

隐藏

上一步

前进

主页

打印

选项(O)

目录(C)

索引(N)

搜索(S)

收藏夹(I)

前言

内容简介

设备支持的接口概况

接口定义

- SDK初始化
  - CLIENT\_Init
  - CLIENT\_Cleanup
  - CLIENT\_GetSDKVersion
  - CLIENT\_GetLastError
  - CLIENT\_SetAutoReconnect
  - CLIENT\_SetConnectTime
  - CLIENT\_SetNetworkParam
- 登陆设备
  - CLIENT\_LoginWithHighLevelSecurity
  - CLIENT\_Logout
- 实时监控
- 数据保存
- 视频抓图
  - CLIENT\_CapturePictureEx
  - CLIENT\_SetSnapRevCallBack
  - CLIENT\_SnapPictureEx
- 回放和下载
- 回放控制
- 语音对讲
- 云台控制

网络SDK开发手册

## CLIENT\_LoginWithHighLevelSecurity

登录设备

```
LLONG CLIENT_LoginWithHighLevelSecurity(  
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY *pstInParam,  
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY *pstOutParam,  
);
```

**Parameters**

[in] pstInParam  
设备登录输入参数

[out] pstOutParam  
设备登录输出参数

**Return Values**

成功返回非0(登录句柄)，失败返回0。

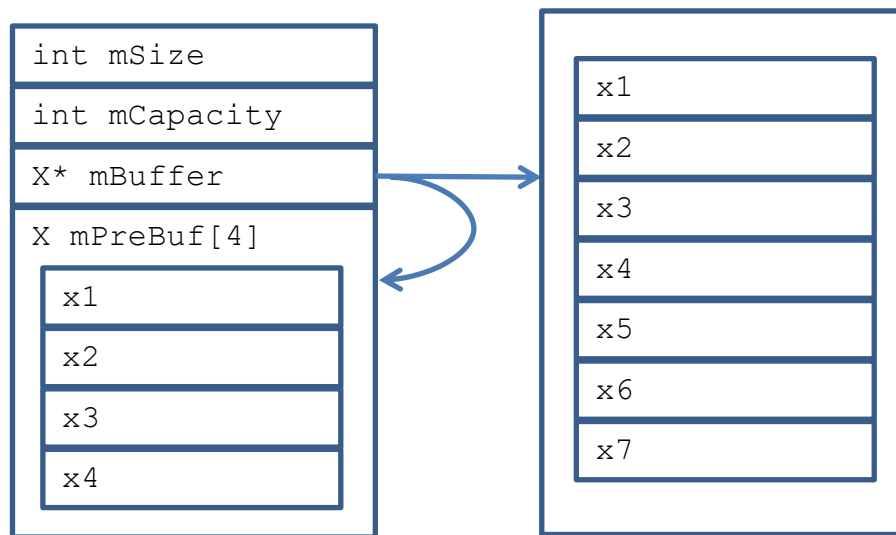
**Remarks**

一般在SDK初始化后调用，与设备进行各种业务功能的前提。

**See Also**

[CLIENT\\_Logout](#)

# 库函数接口 - 信息隐藏



```
void XArray::push(X& x) {
    if(mSize >= mCapacity)
        resize();
    mBuffer[mSize] = x;
    mSize++;
}
```

```
class XArray :
{
private:
    int mSize;
    int mCapacity;
    int mBuffer
#define PRE_BUF_SIZE 4
    X mPreBuf[PRE_BUF_SIZE];

    void resize();
public:
    XArray() {
        mSize = 0;
        mCapacity = PRE_BUF_SIZE;
        mBuffer = mPrebuf;
    }
    int size() { return mSize; }
    void push(X& x);
    X* getHead() { return mBuffer; }
    ...
};
```

库函数的接口应该隐藏复杂的内部实现细节，暴露给调用者的概念尽可能的少。

# 库函数接口 - 资源管理

```
char* getFileLine(FILE* f){ // 获取文件中下一个非空，非注释的行
    char* buffer = new char[BUF_SIZE];
    char* p;
    while((p = fgets(buffer, BUF_SIZE, f)) != NULL){
        while(isspace(*p)) p++; // 找到第一个非空白字符
        if(*p == 0 || *p == '#') continue; // 忽略空行或者#开头的注释
        return buffer;
    }
    return NULL;
}

...
char* p = getFileLine(f);
... // 处理p
delete p; // 需要删除p
```

```
char* getFileLine(FILE* f, char* buffer, int bufferSize){ // 获
    char* p;
    while((p = fgets(buffer, bufferSize, f)) != NULL){
        while(isspace(*p)) p++; // 找到第一个非空白字符
        if(*p == 0 || *p == '#') continue; // 忽略空行或者#开头的注释
        return buffer;
    }
    return NULL;
}

...
char buffer[1024];
char* p = getFileLine(f, buffer, sizeof(buffer));
... // 处理p, 不需要删除p
```

库函数不能偷偷申请资源，如果申请应该负责释放。应该让调用者自己管理资源。明确这个谁申请谁释放的原则，避免接口对资源申请释放权的误解。

# 库函数接口 - 错误处理

```
int validateIpAddress(const char* ipAddress){  
    ...  
    if(发现了某个错误)  
        exit(1)  
    ...  
}
```

```
int validateIpAddress(const char* ipAddress){  
    ...  
    if(发现了某个错误)  
        printf("ip address %s 有某个错误\n", ipAddress)  
    ...  
}
```

```
int validateIpAddress(const char* ipAddress){  
    ...  
    if(发现了某个错误)  
        MessageBox(hWin, "某某格式错误");  
    ...  
}
```

```
int validateIpAddress(const char* ipAddress){  
    ...  
    if(发现了某个错误)  
        return IP_ERROR_XXX;  
    if(发现了另一个错误)  
        return IP_ERROR_YYY;  
    ...  
}
```

库函数的接口应该安静地返回不同的返回值，让调用者根据返回值做不同的处理。而不是在实现中越俎代庖自己控制程序行为，自己输出错误信息。

# 库函数接口的缺点(1)

应用程序和库的代码在同一个进程内运行，共享相同的地址空间。如果程序运行出现的一些神秘的**BUG**，往往难以定位责任。

因此，库函数接口适合的应用场合：

- 可靠的库
- 个人项目
- 小团队内项目
- 合作良好的团队间项目

库函数接口**不适合**的应用场合：

- 使用不熟悉的团队的不可靠的库



# 库函数接口的缺点(2)

应用程序和库的代码在同一个进程内运行，如果应用程序对实时性要求较高，而库函数内可能有较长时间的阻塞操作，会引起严重的性能问题。

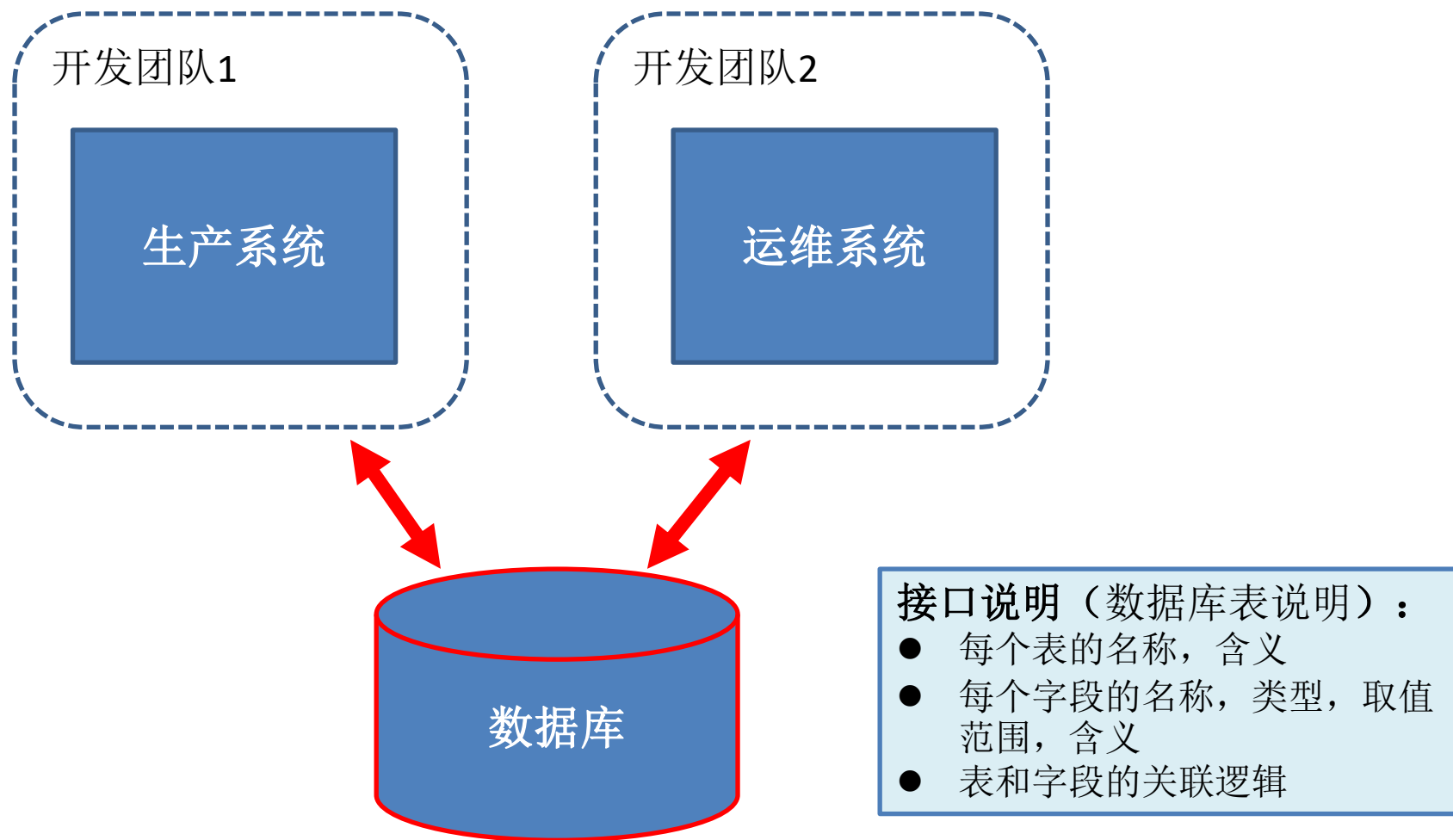
解决方案：

- 启动单独线程运行库函数
- 库函数提供异步接口

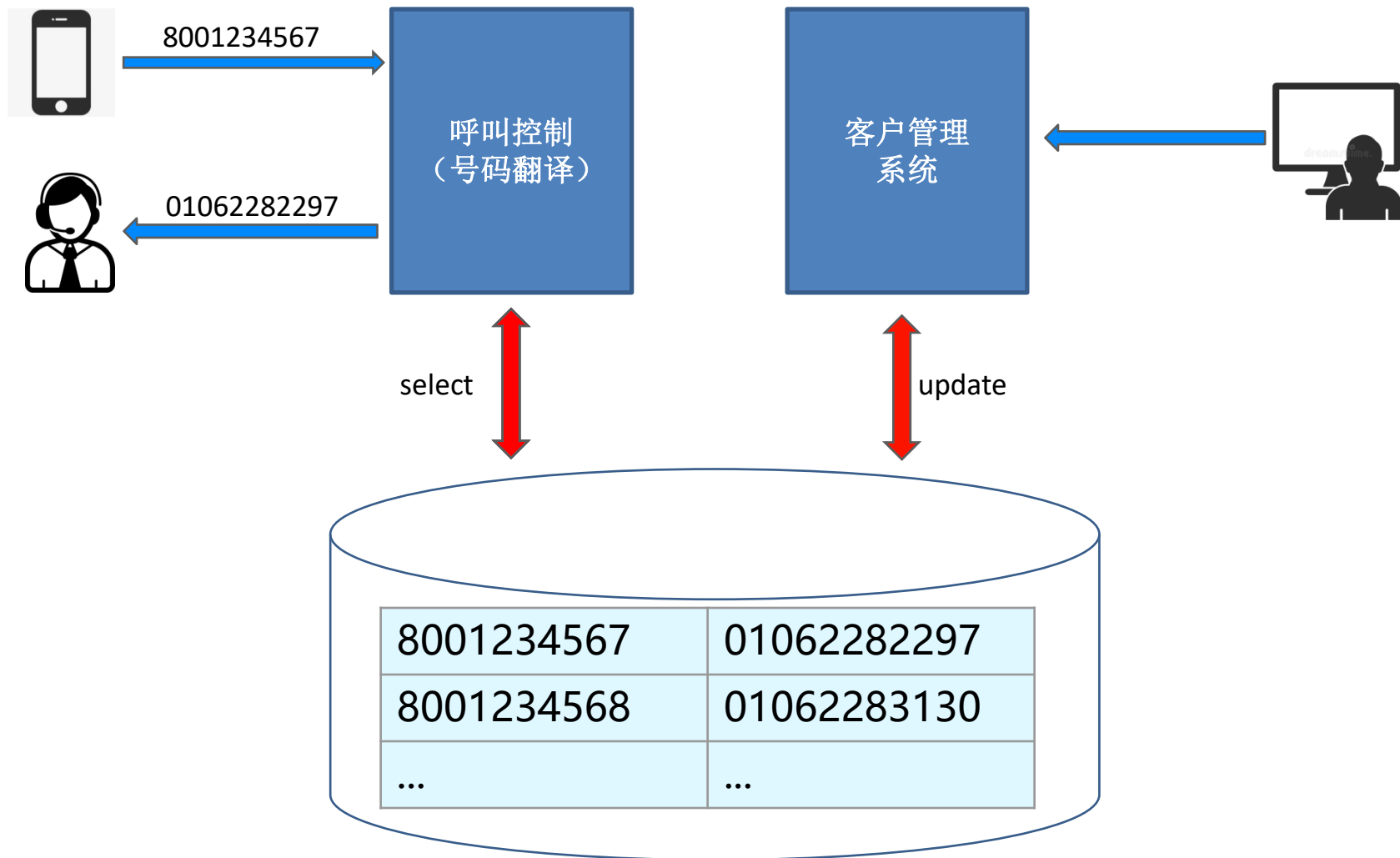
```
// 某设备开发SDK的登录接口，登录一个设备，登录成功才返回。  
LLONG CLIENT_Login(char *pchDVRIP, WORD wDVRPort  
    , char *pchUserName, char *pchPassword  
    , LPNET_DEVICEINFO lpDeviceInfo, int *error = 0  
);
```

```
// 某设备开发SDK的登录接口，调用之后立即返回。  
// 其中LPNET_DVR_USER_LOGIN_INFO含有回调函数指针，一旦登录成功调用回调函数。  
LONG NET_DVR_Login_V40(LPNET_DVR_USER_LOGIN_INFO pLoginInfo  
    , LPNET_DVR_DEVICEINFO_V40 lpDeviceInfo  
);
```

# 共享数据接口



# 共享数据接口 - 示例



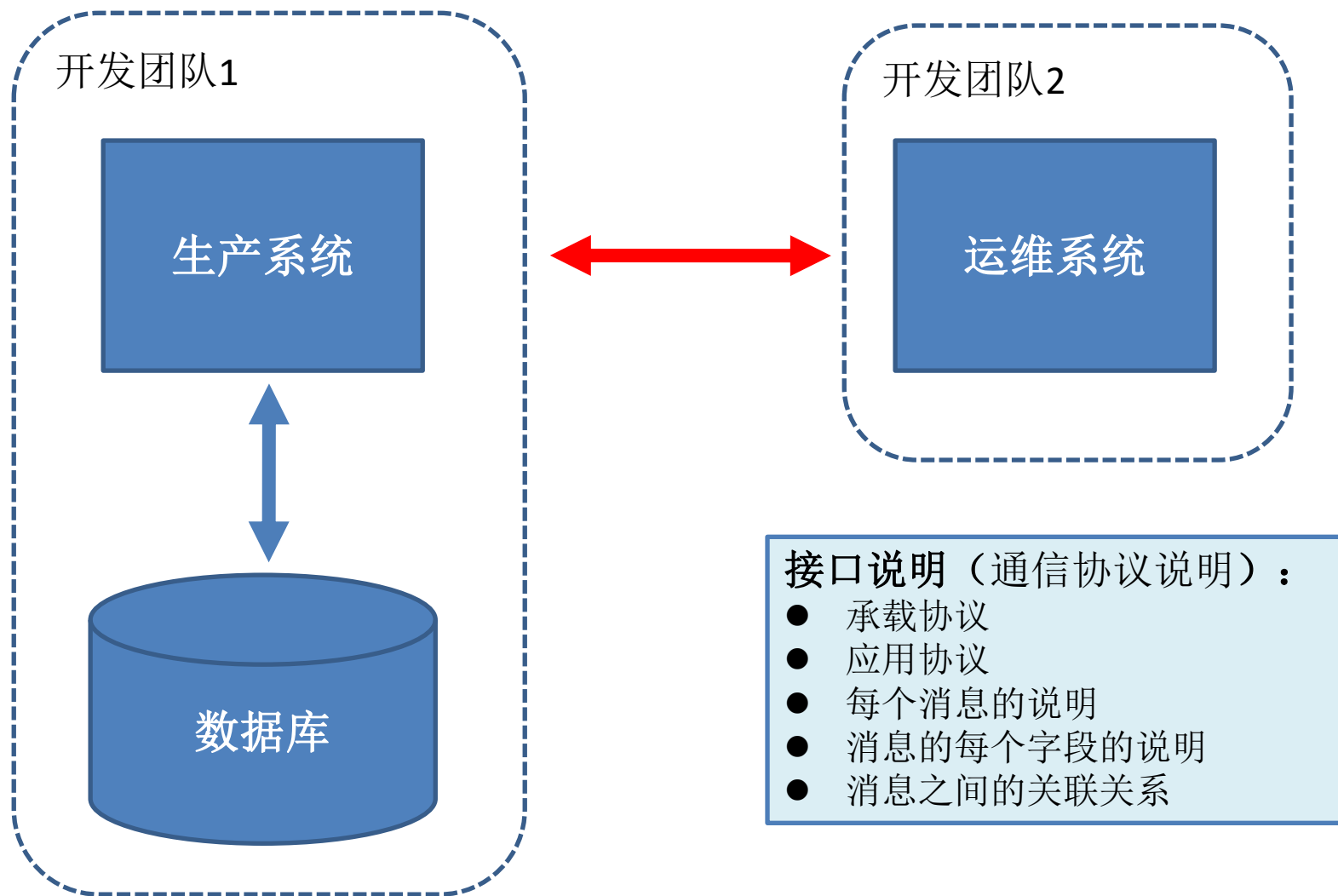


# 思考题

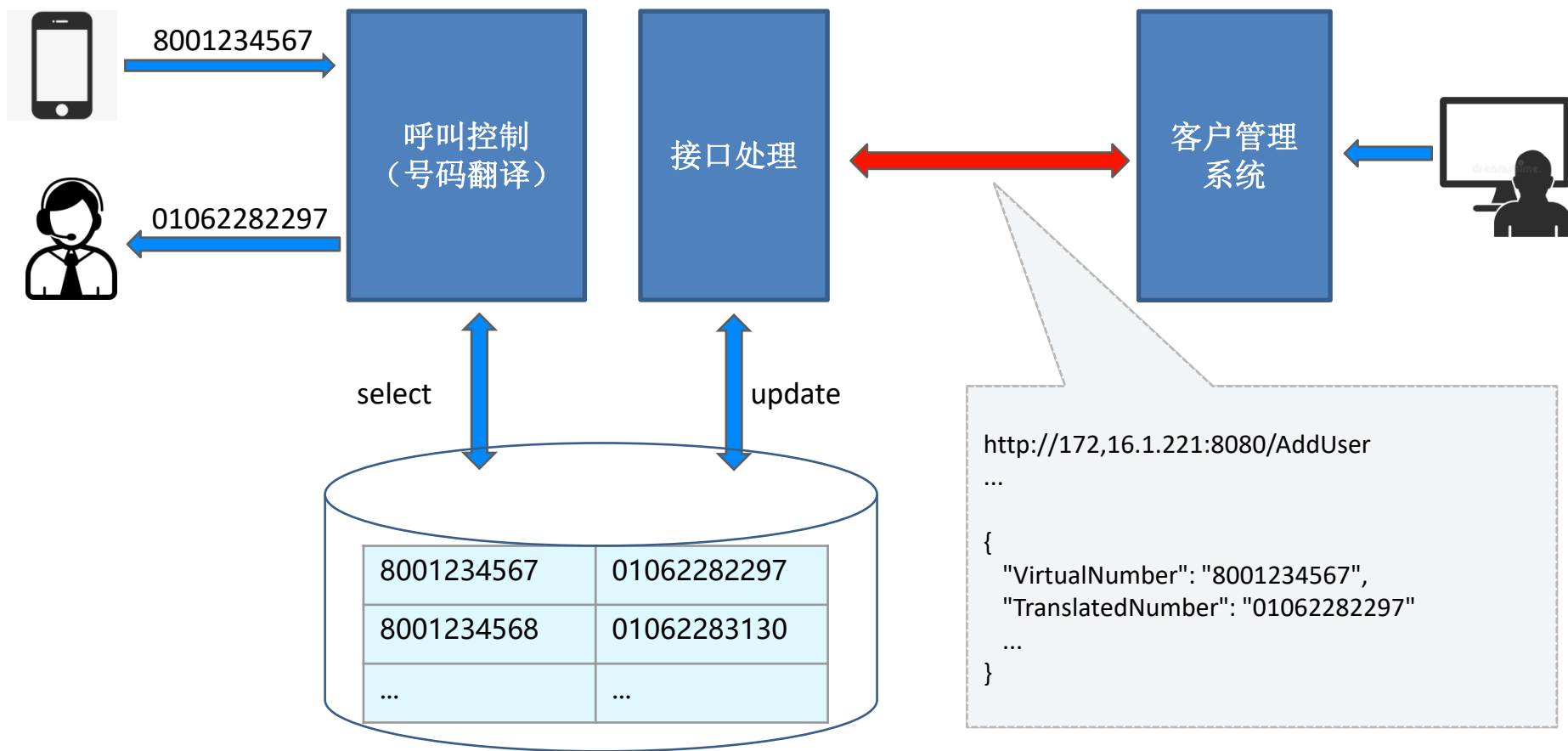
使用共享数据接口有哪些缺点？

如何规避这些缺点导致的问题？

# 通信协议接口



# 通信协议接口



# 通信协议接口 - 如何定义



定义通信协议  
接口的三个要点

选择数据传输承载协议

定义接口数据格式

定义接口双方的处理行为

# 通信协议接口 - 主要承载协议

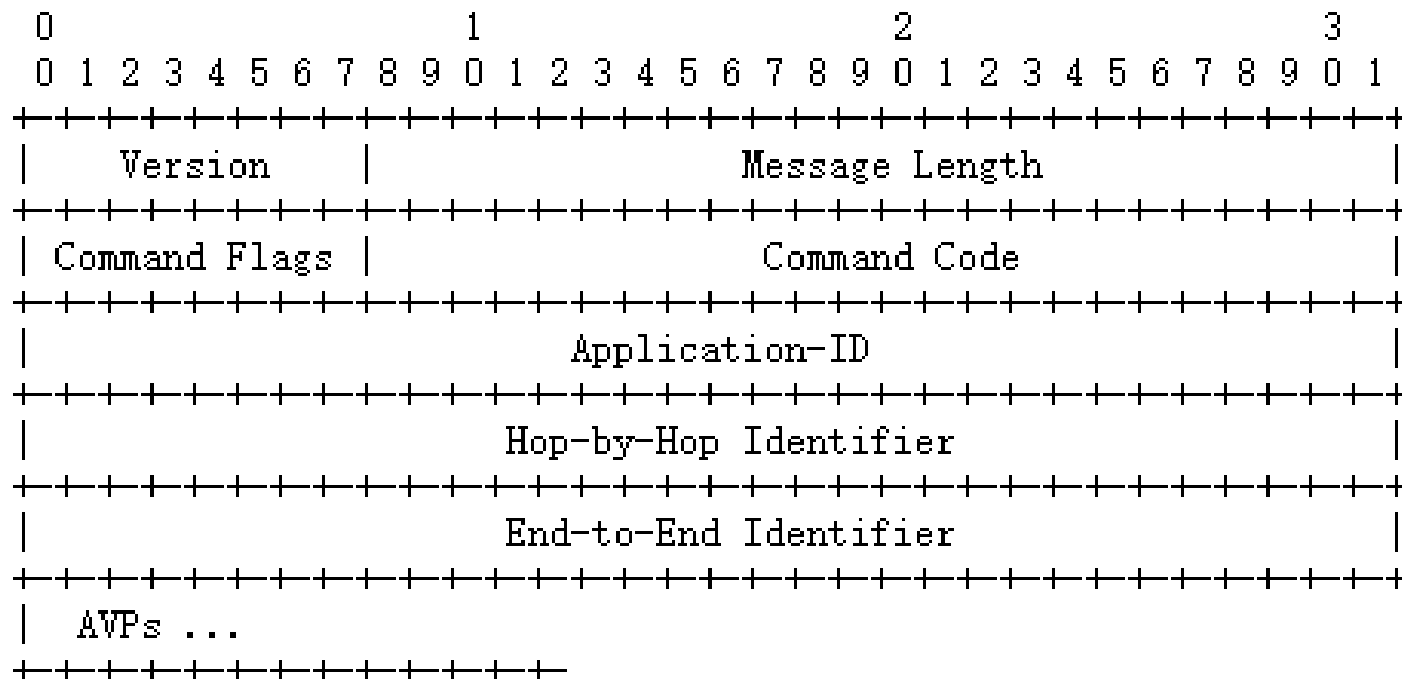
- 基于UDP, User Datagram Protocol
- 基于TCP, Transmission Control Protocol
- 基于更高层面的承载协议:
  - ◆ HTTP, 目前应用最广泛的一种无状态应答式接口承载协议。
  - ◆ WebSocket, 由HTTP衍生, 面向流数据。
  - ◆ RESTFUL, 基于HTTP, 一种特定的简约风格的接口设计。
  - ◆ MQTT, Message Queuing Telemetry Transport, 广泛应用于物联网领域的一种轻量级发布/订阅模式协议。
  - ◆ ZeroMQ, 一个基于消息队列的多线程网络库, 囊括了4中最通用的通信模型。
  - ◆ KAFKA, 基于消费者/生产者模式的实时消息中间件。

...

# 通信协议接口 - 数据格式定义方法

- 直接描述二进制格式，对消息的每一个参数的格式和长度进行描述。
- 特殊文本格式，使用**BNF**范式进行描述。
- 基于某种通用格式，使用这种通用格式的特定描述方法进行描述：
  - ◆ **ASN.1**: Abstract Syntax Notation 1，抽象文法记法1（ITU-T）
  - ◆ **ProtoBuf**: Protocol Buffer（Google）
  - ◆ **XML**: Extensible Markup Language，可扩展标记语言
  - ◆ **JSON**: JavaScript Object Notation
  - ...

# 通信协议接口 - 数据格式定义范例



二进制格式的Diameter协议  
(RFC 6733)

# 通信协议接口 - 数据格式定义范例

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147
```

```
v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

纯文本格式的SIP协议  
(RFC 3261)



# 通信协议接口 - 数据格式定义范例

```
{  
    "format": "pcm",  
    "rate": 16000,  
    "dev_pid": 1537,  
    "channel": 1,  
    "token": "xxx",  
    "cuid": "baidu_workshop",  
    "len": 4096,  
    "speech": "xxx", // xxx为 base64 (FILE_CONTENT)  
}
```

基于json的百度语音识别接口

# 通信协议接口 - 数据格式定义范例

```
GetRequest-PDU ::=
    [0]
        IMPLICIT SEQUENCE {
            request-id
                RequestID,

            error-status          -- always 0
                ErrorStatus,

            error-index           -- always 0
                ErrorIndex,

            variable-bindings
                VarBindList
        }
```

基于ASN1.1描述的SNMP协议  
(RFC 1157)

# 通信协议接口 - 数据格式定义范例

```
message Lane_PB {  
    required int32 laneID = 1;  
    optional LaneAttributes_PB laneAttributes=2;  
    optional AllowedManeuvers_PB maneuvers=3;  
    optional ConnectsToList_PB connectsTo=4;  
    optional SpeedLimits_PB speedLimits=5;  
    optional PointList_PB points =6;  
}
```

基于ProtoBuf描述的一个地图接口

# 通信协议接口 - 接口数据格式范例

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap
  xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
  <soap:Body>
    <cwmp:Request>
      <argument>value</argument>
    </cwmp:Request>
  </soap:Body>
</soap:Envelope>
```

基于XML的宽带设备管理协议片段  
(TR069)

# 通信协议接口 - 接口的兼容性

老版本接口定义:

```
message Lane_PB {  
    required int32 laneID = 1;  
    optional LaneAttributes_PB laneAttributes = 2;  
    optional AllowedManeuvers_PB maneuvers = 3;  
    optional ConnectsToList_PB connectsTo = 4;  
    optional SpeedLimits_PB speedLimits = 5;  
    optional PointList_PB points = 6;  
}
```

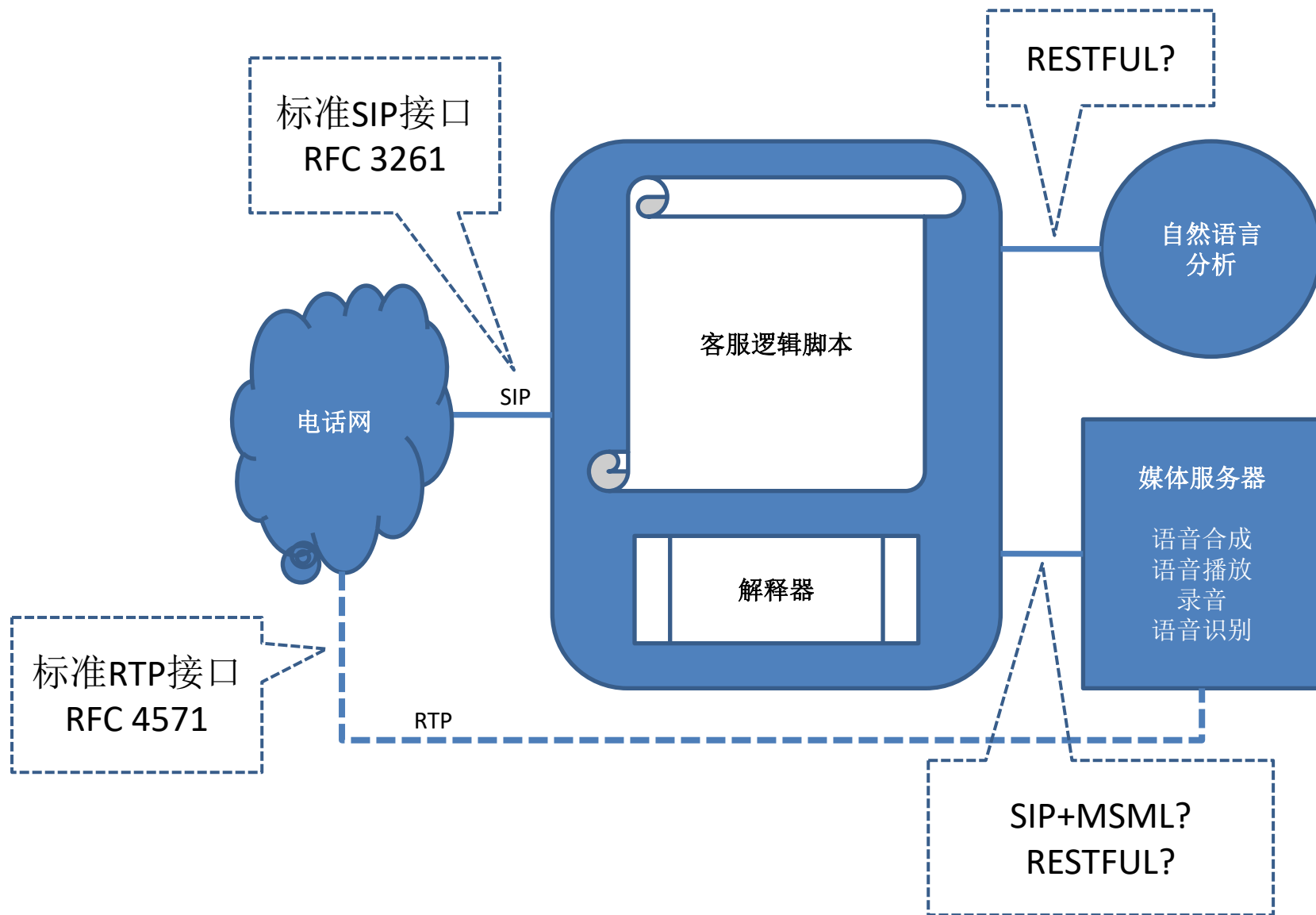
定义的不太好的新版本接口:

```
message Lane_PB {  
    required int32 laneID = 1;  
    optional int32 laneWidth = 2;  
    optional LaneAttributes_PB laneAttributes = 3;  
    optional AllowedManeuvers_PB maneuvers = 4;  
    optional ConnectsToList_PB connectsTo = 5;  
    optional SpeedLimits_PB speedLimits = 6;  
    optional PointList_PB points = 7;  
}
```

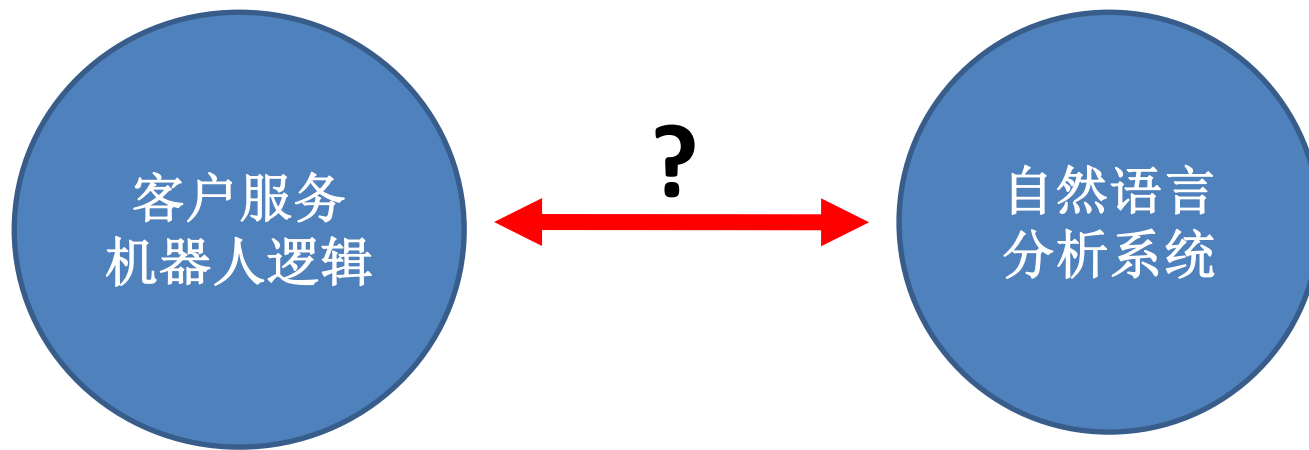
如果考虑接口兼容性, 新版本应该这样定义:

```
message Lane_PB {  
    required int32 laneID = 1;  
    optional LaneAttributes_PB laneAttributes = 2;  
    optional AllowedManeuvers_PB maneuvers = 3;  
    optional ConnectsToList_PB connectsTo = 4;  
    optional SpeedLimits_PB speedLimits = 5;  
    optional PointList_PB points = 6;  
    optional int32 laneWidth = 7;  
}
```

# 通信协议接口

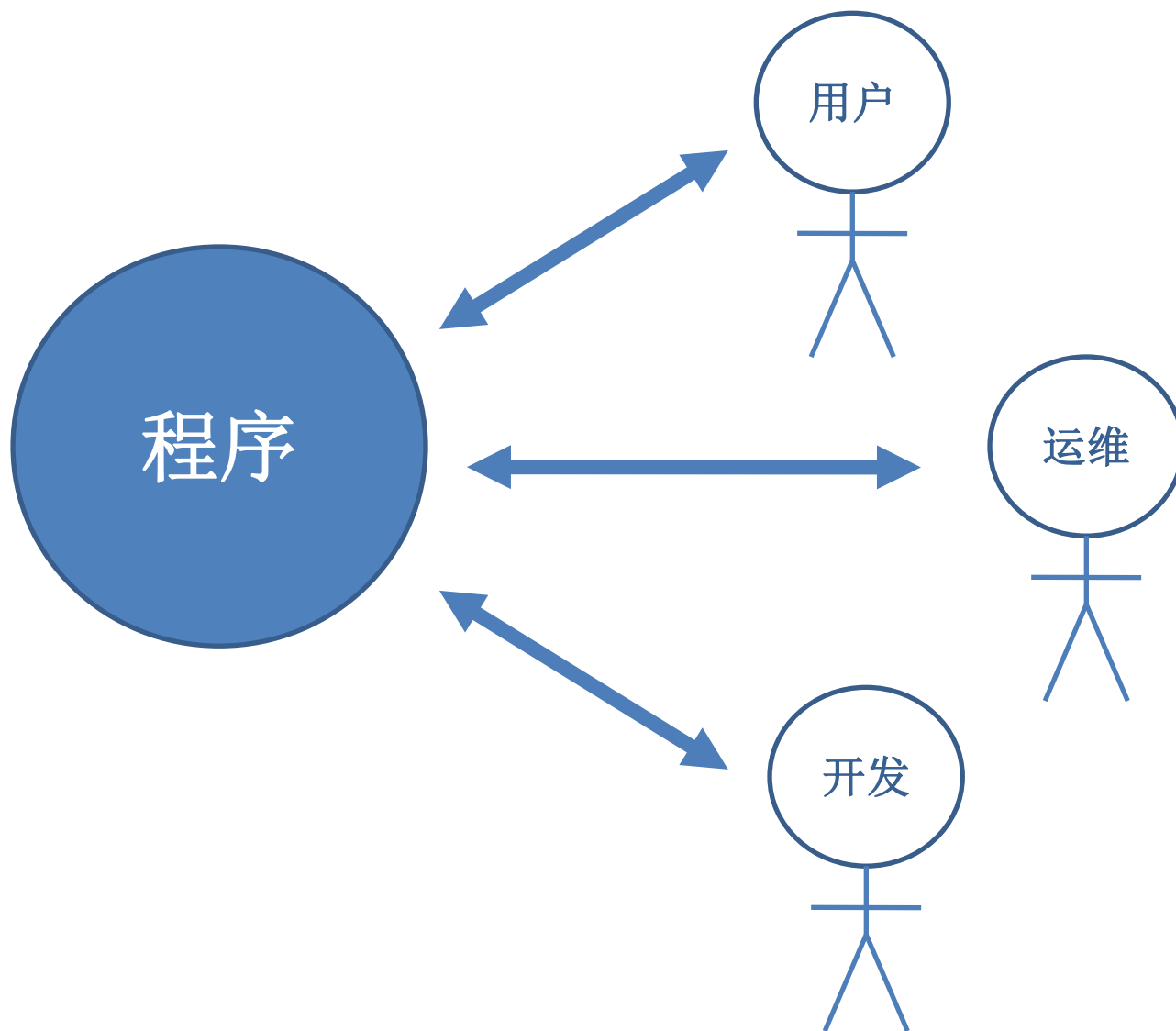


# 思考题



“自然语言分析”这个功能定义一个什么样的接口？

# 人与程序之间的接口





# 命令行接口 - 范例

```
命令提示符
C:\Windows\System32>dir b*.dll
驱动器 C 中的卷是 Windows7_OS
卷的序列号是 407B-E6E2

C:\Windows\System32 的目录

2021/01/13  15:57           71,680 BackgroundMediaPolicy.dll
2019/12/07  17:08           15,872 BamSettingsClient.dll
2021/03/11  12:59          92,160 BarcodeProvisioningPlugin.dll
2019/12/07  17:09        210,232 basecsp.dll
2019/12/07  17:08           71,680 basesrv.dll
2019/12/07  17:08          41,472 batmeter.dll
2019/12/07  17:08        261,632 bcastdvr.proxy.dll
2021/01/13  15:56          111,616 BcastDVRBroker.dll
2021/01/13  15:56        492,544 BcastDVRCClient.dll
2021/01/13  15:56        244,736 BcastDVRCommon.dll
2021/02/10  13:20    1,384,448 bcastdvruserservice.dll
2020/10/26  10:57          130,144 bcd.dll
2019/12/07  17:08           79,872 bcdprov.dll
2019/12/07  17:08           87,552 bcdsrv.dll
2021/03/11  12:59        361,056 BCP47Langs.dll
2021/03/11  12:59        175,624 BCP47mrm.dll
2021/06/09  20:28          146,248 bcrypt.dll
2021/06/09  20:28        529,952 bcryptprimitives.dll
2019/12/07  22:46          105,984 BdeHdCfgLib.dll
2019/12/07  22:46           50,688 bderepair.dll
2021/03/11  13:01        555,008 bdesvc.dll
2019/12/07  22:46           11,776 BdeSysprep.dll
2019/12/07  22:46           35,840 bdeui.dll
2021/07/07  15:25        887,296 BFE.DLL
```

Windows命令提示符

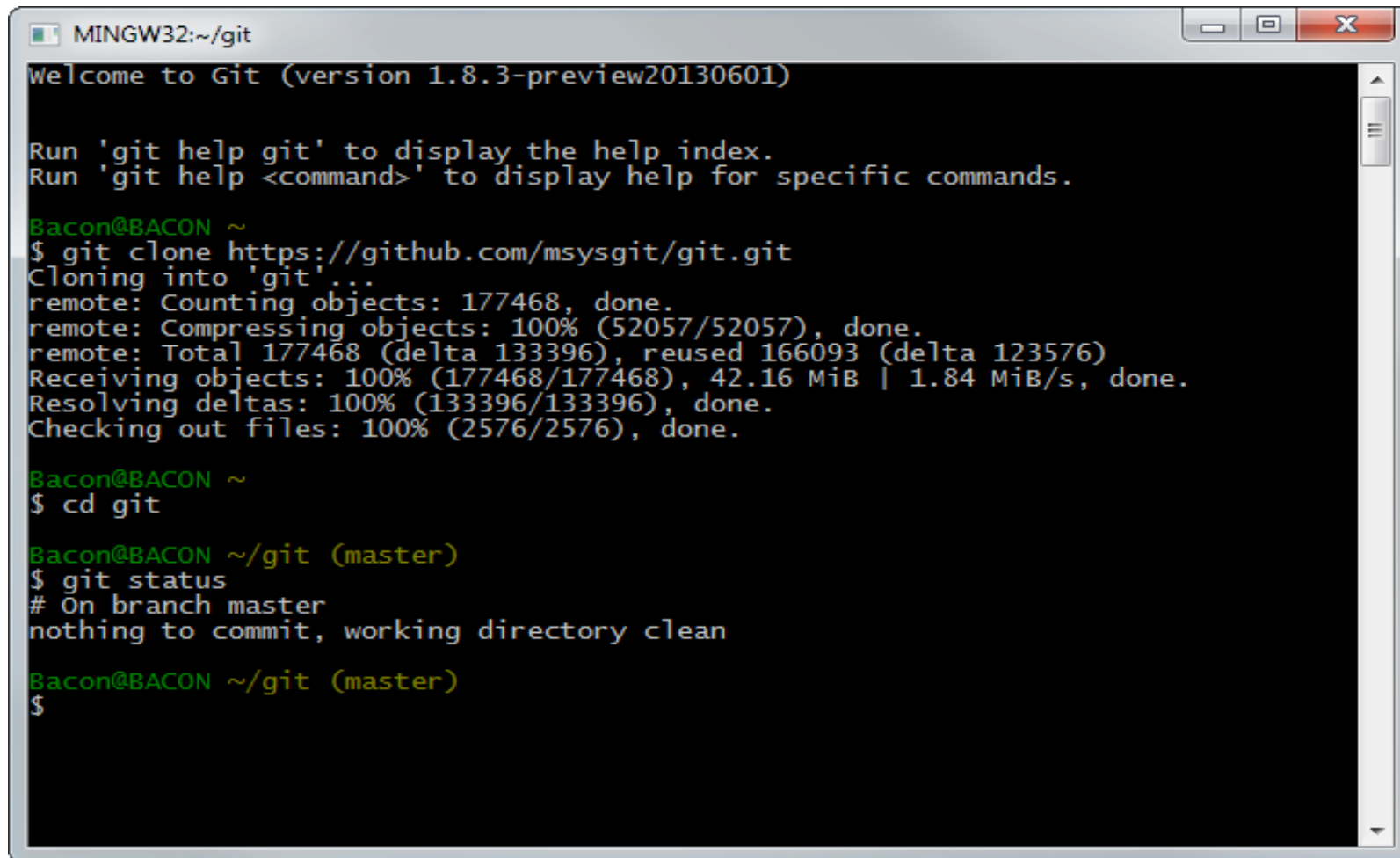
# 命令行接口 - 范例

bash - 快捷方式

```
none 104858620 84427936 20430684 81% /dev
none 104858620 84427936 20430684 81% /run
none 104858620 84427936 20430684 81% /run/lock
none 104858620 84427936 20430684 81% /run/shm
none 104858620 84427936 20430684 81% /run/user
tmpfs 104858620 84427936 20430684 81% /sys/fs/cgroup
C:\ 104858620 84427936 20430684 81% /mnt/c
D:\ 346963964 163129144 183834820 48% /mnt/d
mbzhang@UFO-PC:/mnt/c$ pwd
/mnt/c
mbzhang@UFO-PC:/mnt/c$ ls
20210529 MCLDownload swapfile.sys
ACoinfo MSOCache SymCache
Boot NsisLog.txt System Volume Information
bootmgr PerfLogs Tencent
BOOTNXT ProgramData tmp
BOOTSECT.BAK Program Files Users
Documents and Settings Program Files (x86) Windows
dpstech_sslvpn_bho.log Qt $WinREAgent
hiberfil.sys Recovery $WINRE_BACKUP_PARTITION.MARKER
inetpub $RECYCLE.BIN
Intel searchplugins
mbzhang@UFO-PC:/mnt/c$
```

Linux Bash

# 命令行接口 - 范例



```
MINGW32:~/git
Welcome to Git (version 1.8.3-preview20130601)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Bacon@BACON ~
$ git clone https://github.com/msysgit/git.git
Cloning into 'git'...
remote: Counting objects: 177468, done.
remote: Compressing objects: 100% (52057/52057), done.
remote: Total 177468 (delta 133396), reused 166093 (delta 123576)
Receiving objects: 100% (177468/177468), 42.16 MiB | 1.84 MiB/s, done.
Resolving deltas: 100% (133396/133396), done.
Checking out files: 100% (2576/2576), done.

Bacon@BACON ~
$ cd git

Bacon@BACON ~/git (master)
$ git status
# On branch master
nothing to commit, working directory clean

Bacon@BACON ~/git (master)
$
```

Git命令行终端

# 命令行接口 - 范例

```
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show tables;
```

```
+-----+  
| Tables_in_icoe_hbzhang |  
+-----+  
as_actionlog  
as_calllog  
as_forwardlog  
icoe_action_log  
icoe_asr_server  
icoe_caller_group  
icoe_holidays  
icoe_interface_log  
icoe_pool  
icoe_queue  
icoe_sip_addr  
icoe_st key
```

MYSQL命令行客户端

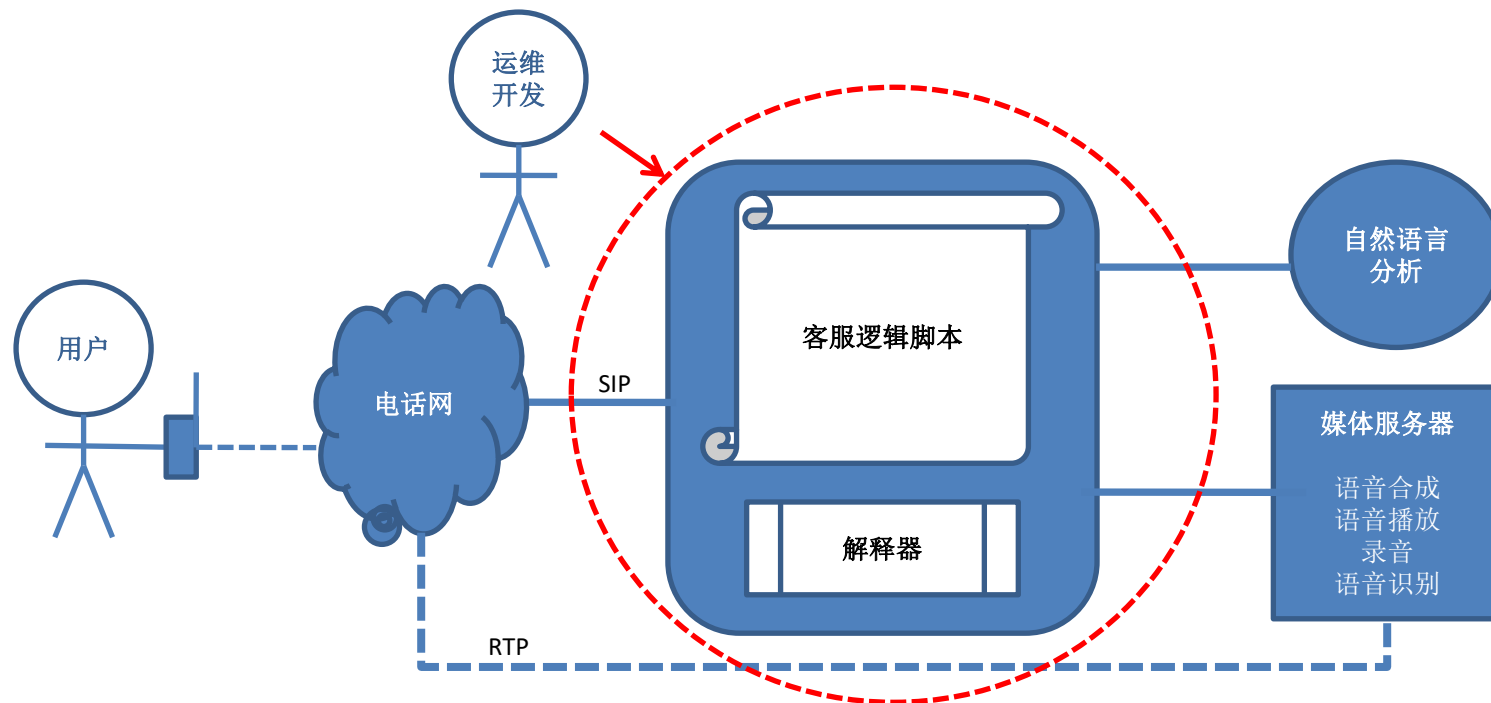
# 命令行接口 (CLI, Command-Line Interface)

## 命令行的优势:

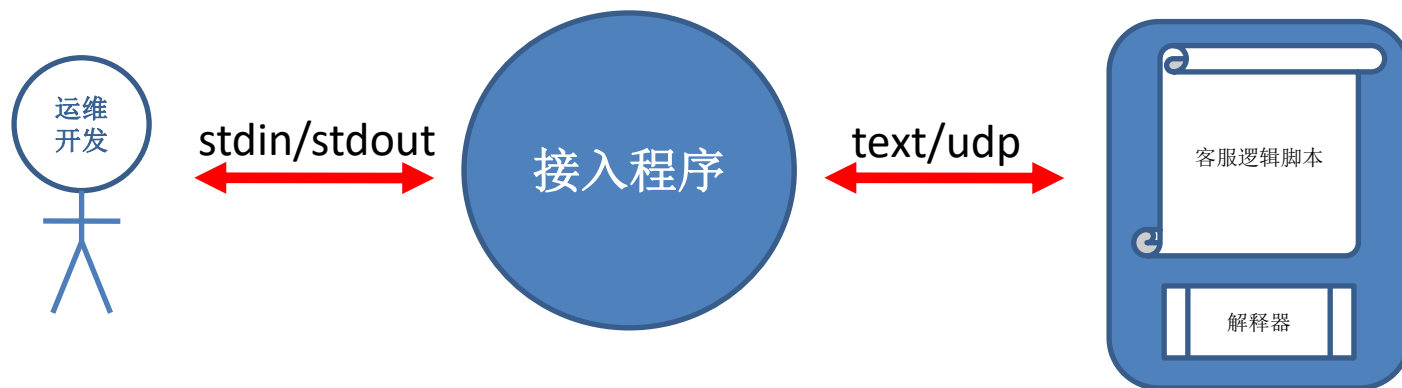
效率: 批处理, 复用, 低带宽占用, 低主机资源占用

稳定: 50年前的unix命令行现在仍然可用

低成本: 对于不经常用的, 面向小众 (开发运维) 的用户接口, 低成本提供大量强大功能



# 命令行接口



**about:** 查看版本

**ls:** 查看已经加载的客服逻辑脚本，查看脚本语法树在内存中的加载情况

**vm:** 查看解释器执行环境的情况，当前变量表，当前Step

**st:** 查看客服脚本的执行统计信息

**sip:** 查看与电话网之间的SIP链路情况，SIP消息的统计

**ms:** 查看与媒体服务器的连接情况，统计情况

**se:** 与自然语言分析系统的连接情况，统计情况

**db:** 数据库的连接情况

**log:** 日志开关的打开/关闭

**cfg:** 当前配置情况，重新加载配置

【开发者用户白盒测试的命令，也可以好好规划，永远留在程序里面】

# 配置文件

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended
symbolic-links=0
max_connections=300
lower_case_table_name=1
#default-character-set=gb2312
default-character-set=utf8
character-set-server=utf8

collation-server=utf8_general_ci

back_log=500
wait_timeout=1800
thread_concurrency=16
query_cache_type=1
query_cache_size=128M
query_cache_limit=4M
sort_buffer_size=2M
thread_cache_size=128
key_buffer_size = 1024M
bulk_insert_buffer_size = 64M
innodb_buffer_pool_size = 1024M
innodb_thread_concurrency = 8

[client]
#default-character-set=gb2312
default-character-set=utf8
default-character-set=utf8

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

MYSQL的配置文件/etc/my.cnf

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Networkd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:112::/run/uidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
landscape:x:110:115::/var/lib/landscape:/usr/sbin/nologin
```

Linux的/etc/passwd

# 配置文件 - 规划配置项

总的原则：提取不适合写死在程序中的数据作为配置文件中的配置项。

因为网络环境的变化而变化的数据，例如：

- 本地的IP地址，端口
- 目的地的IP地址，端口

因为主机资源情况变化的数据，例如：

- 线程池的大小
- 消息队列的最大长度限制

因为用户的偏好而变化的数据，例如：

- 字符编码，UTF-8还是GBK？
- 语言，中文还是英文？





# 配置文件 - 配置项生效

有些重要的配置，修改之后需要重启后生效

- 修改配置文件，重启程序

有的配置通过命令在线生效：

- 使用命令行单项修改配置，同步修改配置文件
- 修改配置文件，使用命令行整体重新加载

这两种配置需要谨慎区分，前者应该尽可能少。



# 配置文件 - 配置文件的格式

一个程序的多个配置文件的格式最好统一

- 一开始就规划好，选择一个合适的文件格式
- 其他人在后期的程序修改中，不要私自添加其它格式的配置文件

常用格式：

- 经典的ini格式，[xxx]分节，每节配置名字和取值
- XML格式
- JSON

配置文件一定要允许加注释

- 配置文件中的注释和程序中的注释一样重要
- #开头的注释行
- XML等格式本身支持的注释方式

# 错误信息 - 好的例子

cc编译器输出的错误信息

```
$ cc a.c
a.c: 在函数 'main' 中:
a.c:5: 错误: expected ';' before '}' token
```

ssh连接失败输出的错误信息

```
$ ssh test@172.16.1.226
ssh: connect to host 172.16.1.226 port 22: No route to host
```

git命令使用错误时的输出信息

```
$ git commi
git: 'commi' is not a git command. See 'git --help'.
```

```
The most similar commands are
    commit
    column
    config
```

# 错误信息 - 输出完整准确的错误信息

```
if(connect(...) < 0)
    printf("连接失败\n");
```

```
if(connect(...) < 0)
    printf("连接 %s:%d 失败\n", ip, port);
```

```
if(connect(...) < 0)
    printf("连接 %s:%d 失败, %d\n", ip, port, errno);
```

```
if(connect(...) < 0)
    printf("连接 %s:%d 失败, %d:%s\n"
        , ip, port, errno, strerror(errno));
```

```
if(connect(...) < 0)
    errorLog("连接 %s:%d 失败, %d:%s"
        , ip, port, errno, strerror(errno));
```

```
21-01-22 14:42:58: [xx]连接 172.16.1.221:27004 失败, 111:Connection refused
```

# 日志文件

/var/log/mysqld.log

```
210713 8:54:24 InnoDB: Started; log sequence number 4 2450114949
210713 8:54:24 [Note] Event Scheduler: Loaded 0 events
210713 8:54:24 [Note] /usr/libexec/mysqld: ready for connections.
Version: '5.1.73' socket: '/var/lib/mysql/mysql.sock' port: 3306 Source distribution
210908 09:58:45 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
210908 9:58:46 [Warning] '--default-character-set' is deprecated and will be removed in a future release.
Please use '--character-set-server' instead.
210908 9:58:46 InnoDB: Initializing buffer pool, size = 1.0G
210908 9:58:46 InnoDB: Completed initialization of buffer pool
InnoDB: The log sequence number in ibdata files does not match
InnoDB: the log sequence number in the ib_logfiles!
210908 9:58:46 InnoDB: Database was not shut down normally!
InnoDB: Starting crash recovery.
InnoDB: Reading tablespace information from the .ibd files...
InnoDB: Restoring possible half-written data pages from the doublewrite
InnoDB: buffer...
210908 9:58:47 InnoDB: Started; log sequence number 4 2450179499
210908 9:58:47 [Note] Event Scheduler: Loaded 0 events
210908 9:58:47 [Note] /usr/libexec/mysqld: ready for connections.
Version: '5.1.73' socket: '/var/lib/mysql/mysql.sock' port: 3306 Source distribution
```

日志是了解程序的过往运行状况的主要参考，  
是排错的主要依据。

# 日志文件 - 哪些内容要写入日志文件?

## 错误信息需要写入日志文件

错误信息，在输出到标准输出，或者输出到前端页面的同时，需要同时写入日志文件。

## 重要的程序运行信息需要写入日志文件

- 程序启动，关闭，
- 重要的通信链路连接断开，
- 重要的模块加载卸载

## 需要可控写入日志文件的内容

- 这些信息可能包括详细的业务运行信息，模块之间交互的接口信息
- 这些信息往往内容过多，可能需要有开关分级控制
- 开关可以使用命令行接口，实现不同的命令实现，例如：
  - trace db on 开启数据库sql语句的日志
  - trace db off 关闭数据库sql语句的日志
  - trace http on 开启http接口的原始消息写入日志
  - trace http off 关闭http接口的原始消息写入日志

# 日志文件 - 日志的要求

## 一般要求：

详略得当，文字简练，完整，准确。  
格式朴素，防止日志“内卷”

## 每行日志的要素：

时间戳（可以精确到毫秒甚至微秒）  
发生的模块标识  
事件简要说明  
事件参数的详细列表（例如记录模块间交互的接口消息的日志）

## 日志文件的大小控制：

日志文件不能累积太大，否则阅读，搜索，下载都是比较麻烦的事情  
根据程序特点，可以设置每天，或者每周，每月一个日志文件  
要有自动备份，删除的机制，不要让日志占满了所有存储空间

# 图形用户界面 (GUI)

GUI的设计  
包含两个  
重要层面

## 逻辑设计

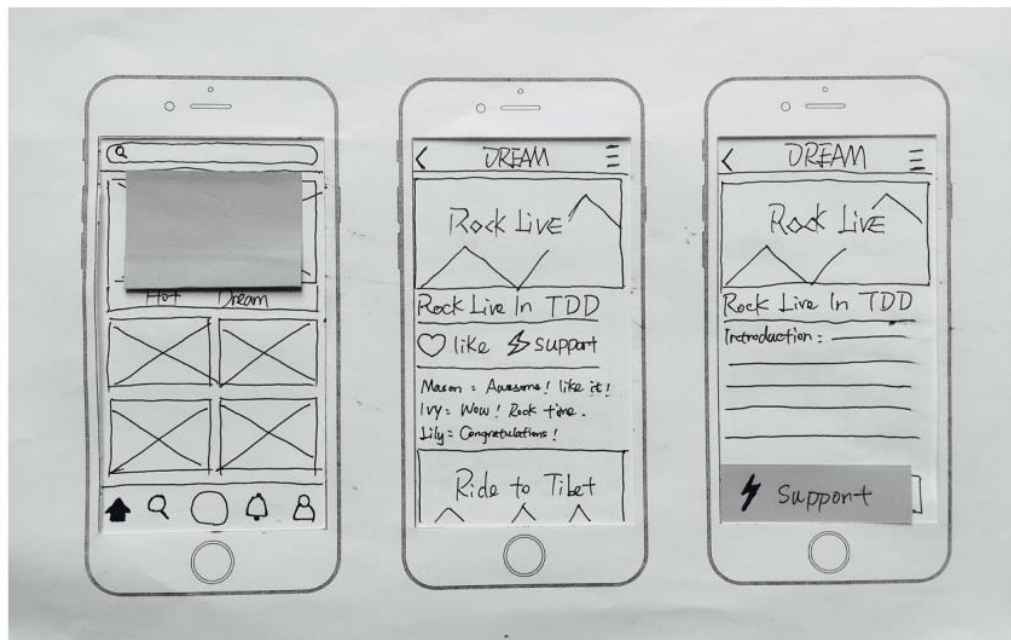
侧重于使用逻辑  
程序员主要负责  
例如主要针对HTML + JS

## 呈现设计

侧重于美观  
Artist主要负责  
例如主要针对CSS



# GUI的逻辑设计



```
<div id="dream" class="title">  
  ...  
</div>
```

有哪些页面？

页面之间的先后逻辑关系？

页面的分类？

有哪些控件？

各个控件应该放在哪一页？

控件之间的先后逻辑关系？

控件之间的位置逻辑关系？

控件的功能分类？

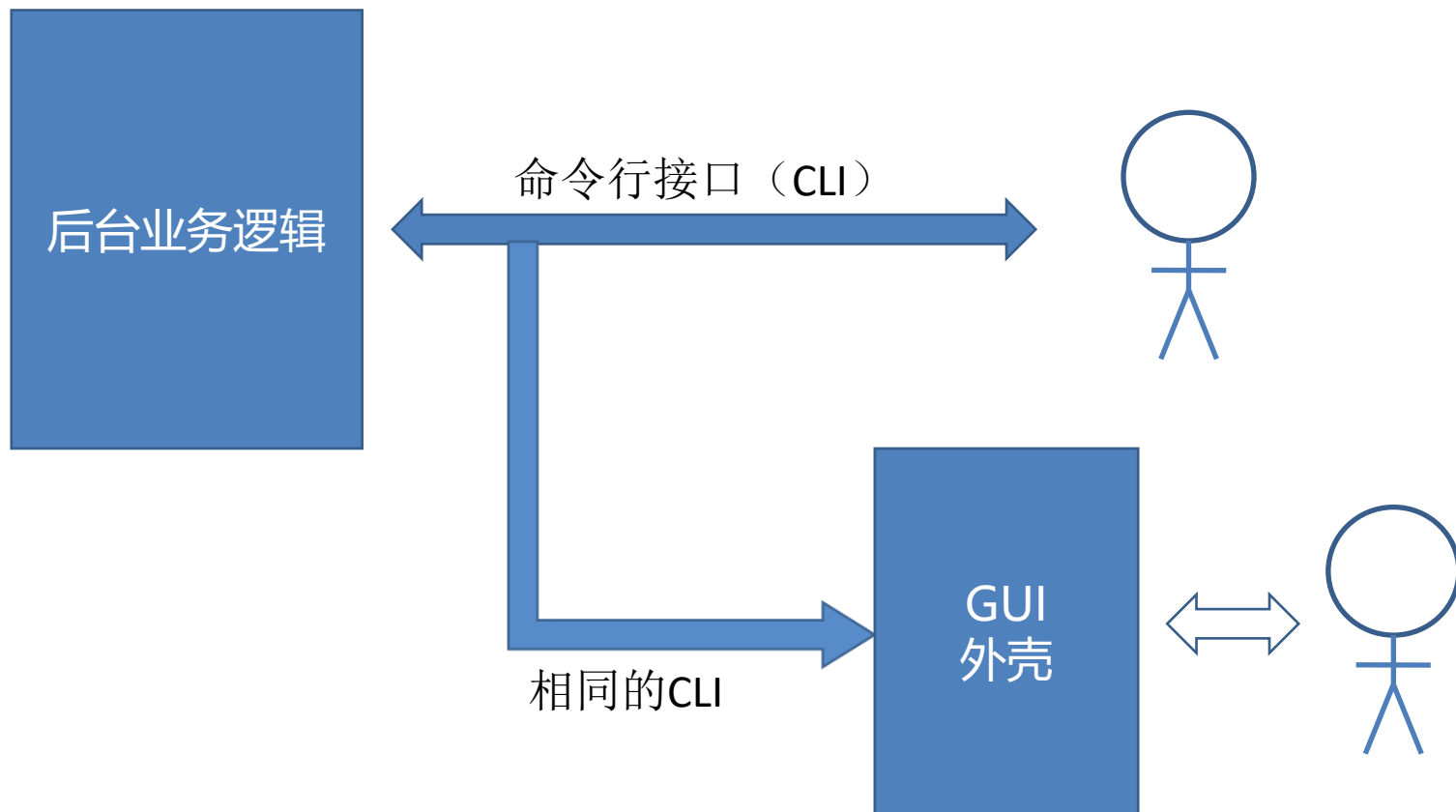
控件的风格分类？

# GUI的呈现设计

```
div.title
{
    font: bold 30px Arial,sans-serif;
    padding-top: 8px;
    padding-left: 10px;
    float: left;
}

#dream
{
    color: "#2233FF";
}
```

# 以CLI为基础的图形用户界面（GUI）



```
$ svn --help
usage: svn <subcommand> [options]
Subversion command-line client
Type 'svn help <subcommand>' for help on a specific
Type 'svn --version' to see the version
'svn --version --verbose' for more details
'svn --version --quiet' for a quiet version
```

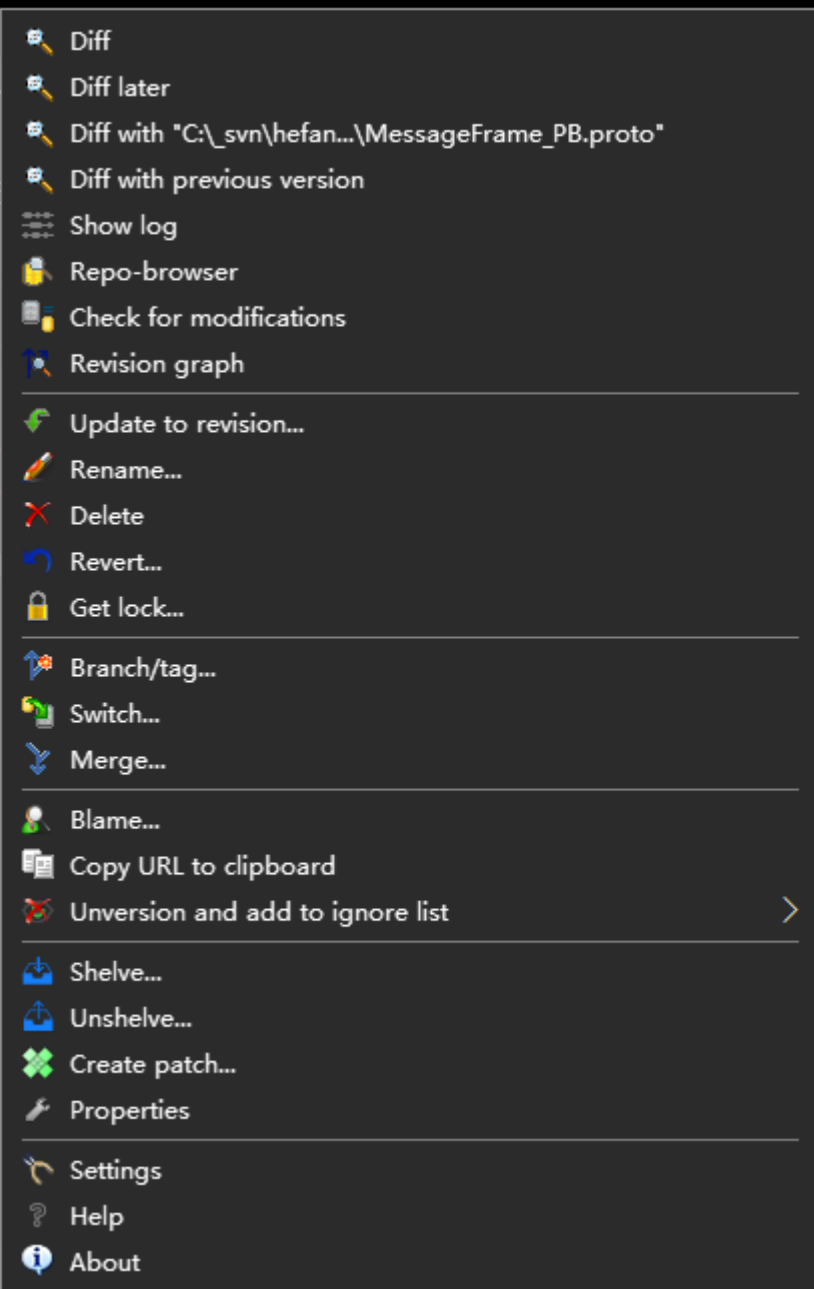
Most subcommands take file names as arguments. If no file names are given on the command, it recurses on the current directory.

Available subcommands:

```
add
auth
blame (praise, annotate)
cat
changelist (cl)
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
diff (di)
export
```



include > comserv	修改日期
..swp	2022/9/4 15:00
..un~	2022/9/4 15:00
check.h	2020/12/28 15:00



SVN的CLI



SVN的GUI

```
1 #include <stdio.h>
2 int main(int argc, const char* args[]){
3     int i, k;
4     printf("hello, world!\n");
5     printf("argc:%d\nargv:\n", argc);
6     for(i = 0; i < argc; i++){
7         printf("%d : %s\n", i, args[i]);
8     }
9     getchar();
10    return 0;
11 }
```

基于CLI的gdb



VSCode对gdb的封装

\$ g++ -o gdbtest

\$ gdb gdbtest

...

(gdb) b 6

Breakpoint 1 at

(gdb) r 111 222

Starting program

hello, world!

argc:4

argv:

Breakpoint 1, r

6 fo:

(gdb) p i

\$1 = 0

(gdb) p argc

\$2 = 4

(gdb)

The screenshot shows the VSCode IDE with the C++ Launch (GDB) configuration. The source code of 1.cpp is displayed, showing the main function. The GDB console output shows the program execution, including the output of the printf statements. The VSCode interface shows the source code, the GDB console, and the output window.

1.cpp

```
1 #include <stdio.h>
2 int main(int argc, char *args[]){
3     int i, k;
4     printf("hello, world!\n");
5     printf("argc:%d\nargv:\n",argc);
6     for(i = 0; i < argc; i++){
7         printf("%d : %s\n", i, args[i]);
8     }
9     getchar();
10    return 0;
11 }
```

调试 C++ Launch (GDB)

变量

Locals

i: 4194432 [int]

argc: 4 [int]

args: 0x690db8 [char \*\*]

监视

i: 4194432 [int]

&i: 0x61ff2c [int \*]

\*&i: 4194432 [int]

调用堆栈 已于 BREAKPOINT 暂停

Thread #1 已暂停

main(int argc, char \*\* args) 1.cpp 5

Thread #2 已暂停

Thread #3 已暂停

断点

1.cpp 5

1.cpp 7

输出

hello, world!

微软拼音 半 :

# GUI的分类

## 终端类型

PC桌面（Windows? Linux? Mac?）

移动终端（Android? IOS?）

各种嵌入式终端

## 呈现方式

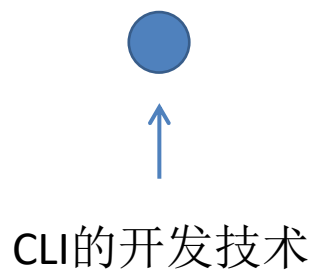
桌面应用？

移动终端APP？

小程序？

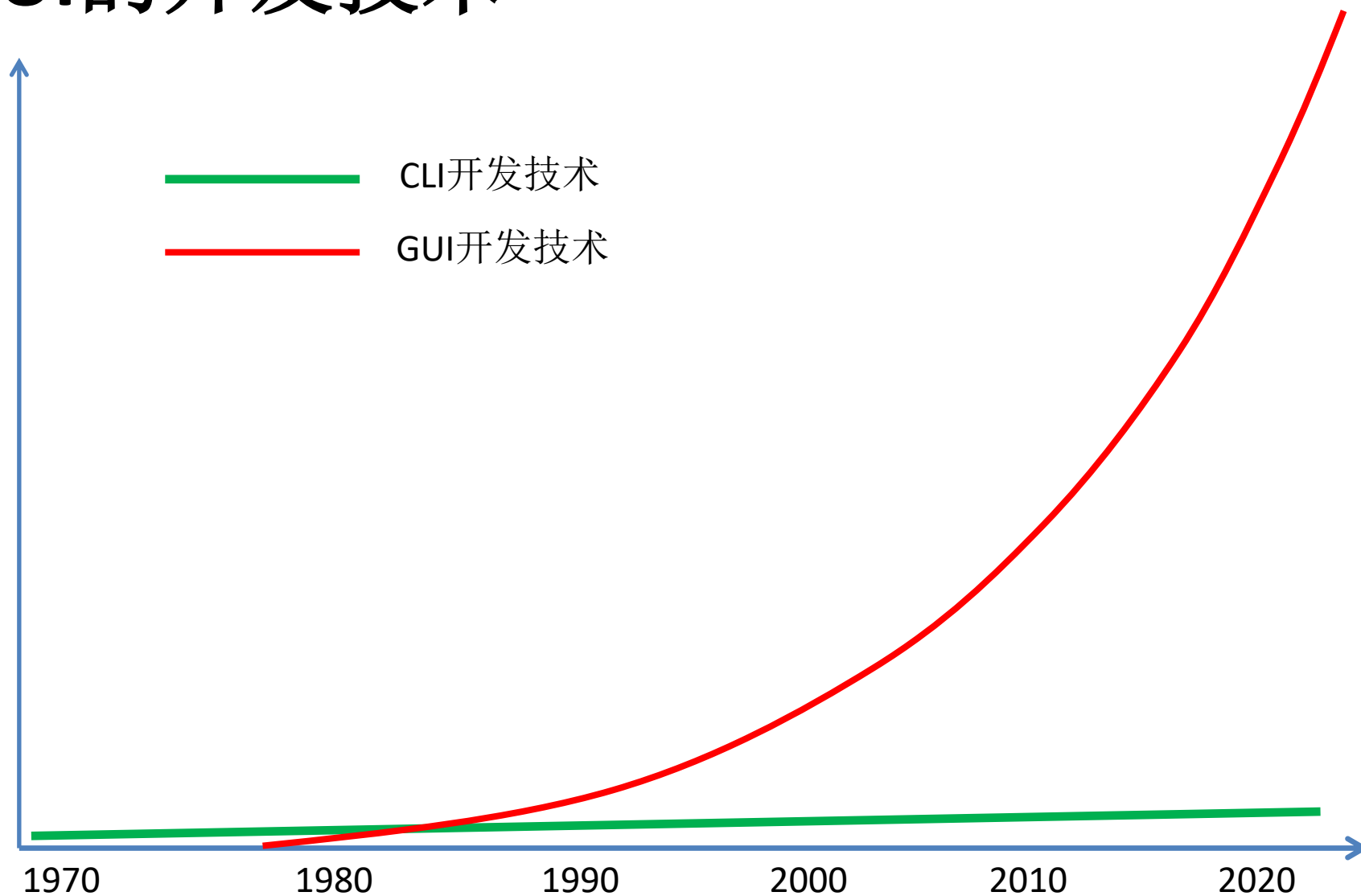
浏览器？

# GUI的开发技术



GUI的开发技术

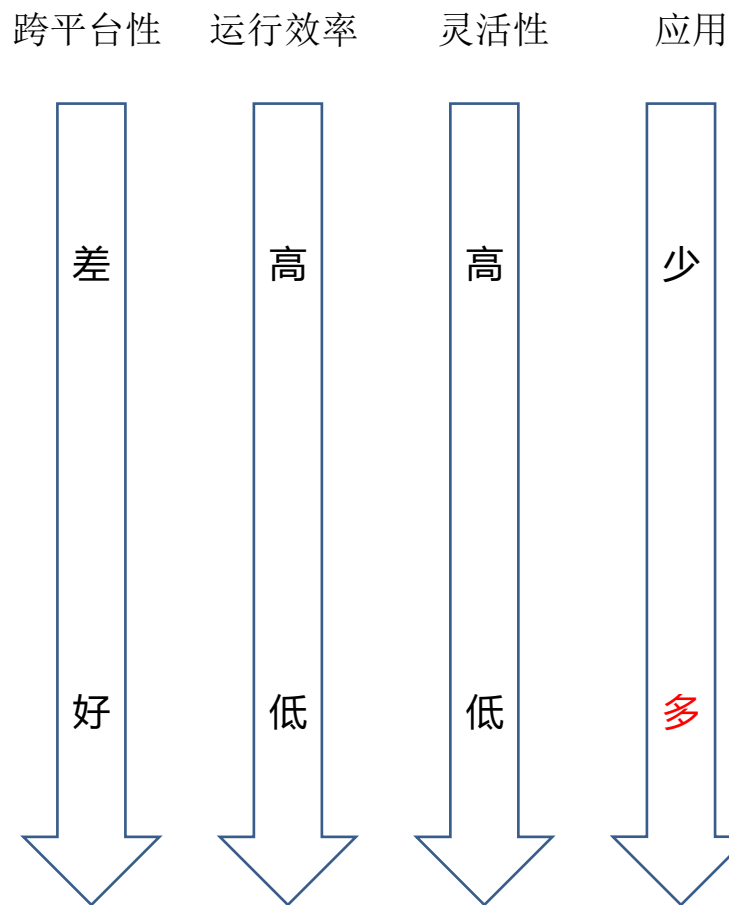
# GUI的开发技术





# GUI的开发技术

- ◆ 直接调用原生（Native）API
  - MFC, GTK
- ◆ 调用进一步封装的库
  - Qt, Swing, Unity 3D
- ◆ DSL + 解释器
  - QML/Qt, XAML/WFP
- ◆ **HTML + CSS + JS** + 浏览器引擎
  - Electron
- ◆ **HTML + CSS + JS** + 浏览器（WEB应用）



# HTML + CSS + JS

## HTML

```
<div id="hello">你好世界</div>
```



你好世界

## CSS

```
#hello  
{  
  color: green;  
  background-color: yellow;  
  border: 5px solid red;  
}
```



你好世界

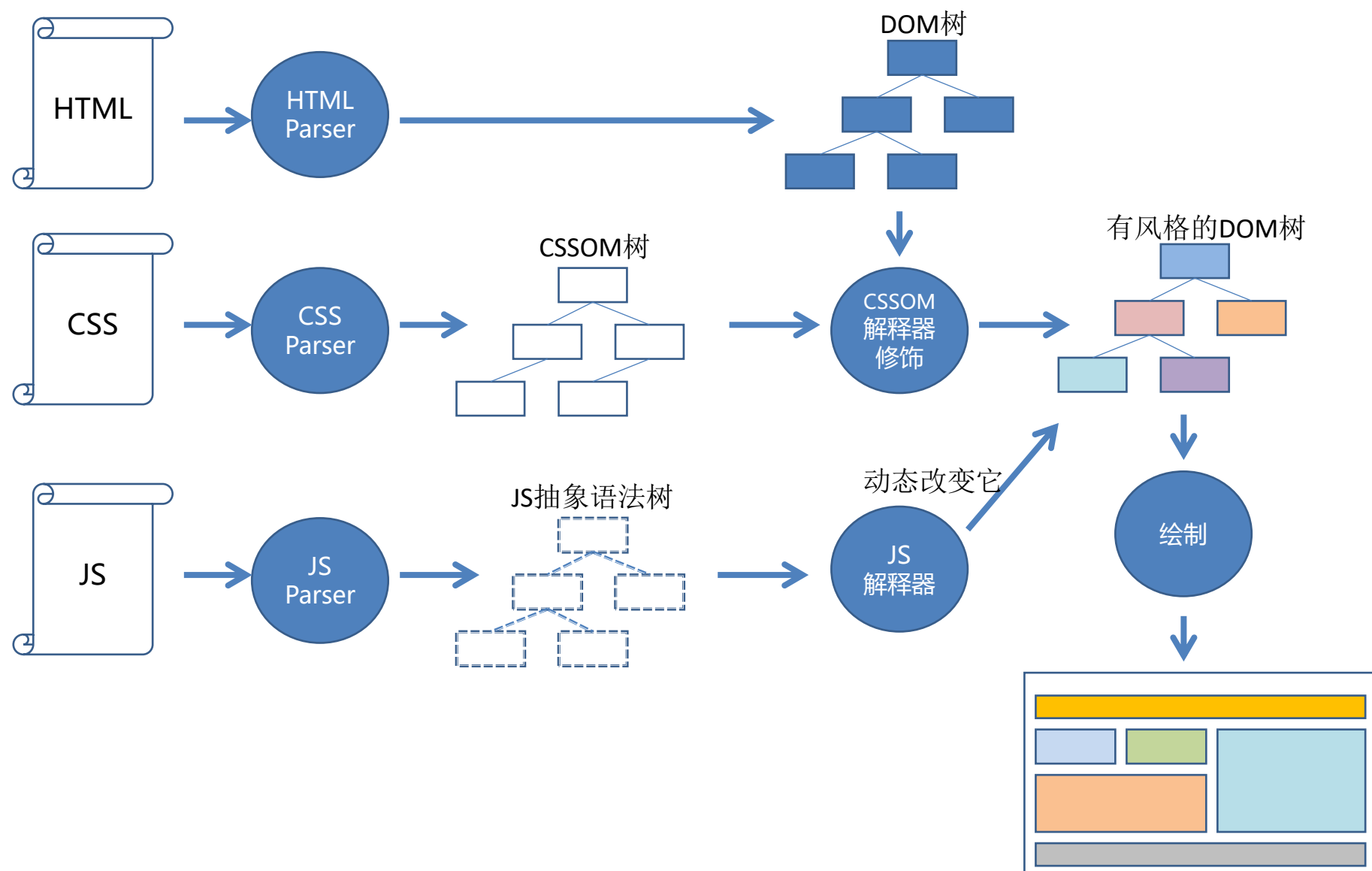
## JS

```
$("#hello").text("Hello World")
```

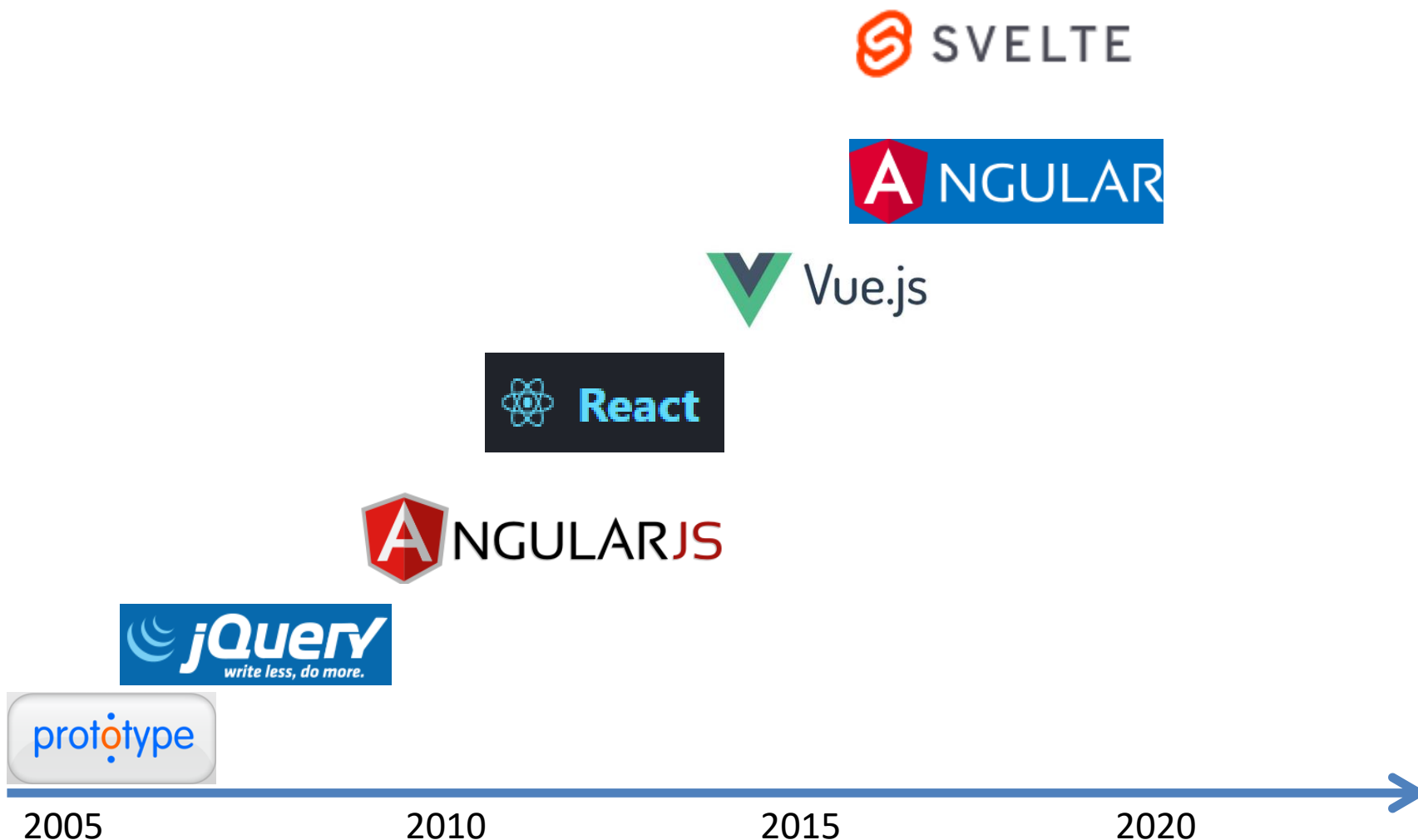


Hello World

# 浏览器引擎解释HTML + CSS + JS的过程



# 从JS库到前端框架



# JQuery.js

HTML:

```
<div id="hello">你好世界</div>
```

使用原生JS修改div元素的内容:

```
document.getElementById("hello").innerText = "Hello World"
```

JQuery.js片段:

```
var $ = function jQuery(selector) {  
    return new jQuery.fn.init(selector);  
}  
jQuery.fn = jQuery.prototype = {  
    init: function(selector) {  
        if(typeof selector === "string") {  
            match = rquickExpr.exec(selector);  
            elem = document.getElementById(match[2]);  
            this.length = 1;  
            this[0] = elem;  
            return this;  
        }  
        ...  
    }  
}  
  
jQuery.fn.init.prototype = jQuery.fn;  
  
jQuery.fn.extend({  
    text: function(value) {  
        ...  
    },  
    ...  
})
```



基于jQuery.js库, 可以这样修改div元素的内容:

```
$("#hello").text("Hello World")
```

语法糖



# 应用JS框架的一个例子

## 需求:

显示一句古诗，每个字都要有自己的单独装饰，其中某个字（例如“花”）的样式要与其它字不同。

下图所示:

江	流	宛	转	绕	芳	甸	月	照	花	林	皆	似	霰
---	---	---	---	---	---	---	---	---	---	---	---	---	---

# 不用JS框架的普通做法

## HTML

```
<div>
  <div class="normal">江</div>
  <div class="normal">流</div>
  <div class="normal">宛</div>
  <div class="normal">转</div>
  <div class="normal">绕</div>
  <div class="normal">芳</div>
  <div class="normal">甸</div>
  <div class="normal">月</div>
  <div class="normal">照</div>
  <div class="normal special">花
</div>
  <div class="normal">林</div>
  <div class="normal">皆</div>
  <div class="normal">似</div>
  <div class="normal">霰</div>
</div>
```

## CSS

```
.normal {
  float: left;
  width: 35px;
  height: 35px;
  margin: 2px;
  font-size: 25px;
  font-family: "宋体";
  border: 1px solid #888888;
  display: flex;
  justify-content: center;
  padding-top: 5px;
}

.special{
  background: #FFFF88;
}
```

后端直接根据数据生成正确的HTML，发送给前端，前端浏览器直接解释HTML以及CSS进行渲染

# 使用Vue.js

使用了Vue模板语法的HTML

```
<div id="vuetest">
  <div>
    <template v-for="(char) in sentence.split('')">
      <div class="normal" v-bind:class="{ 'special': char == special}">{{char}}</div>
    </template>
  </div>
</div>
<script>
new Vue({
  el: '#vuetest',
  data: { "sentence": "江流宛转绕芳甸月照花林皆似霰", "special": "花" }
})
</script>
```

- new Vue语句将数据传入
- template元素中使用v-for指令控制对于子元素div进行循环，循环条件为sentence.split('')拆分出来的一个数组。
- v-bind将class属性与{'special': char == special}绑定
- 双花括弧括起来的变量{{char}}实现插值

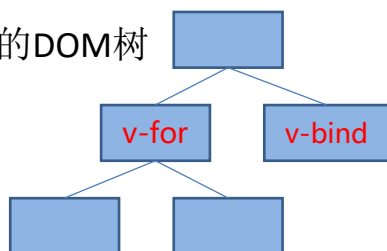


# Vue.js的工作原理

```
<div id="vuetest">
<div>
<template v-for="(char) in sentence.split('')">
  <div class="normal" v-bind:class="{ 'special': char == special}">{{char}}</div>
</template>
</div>
</div>
<script>
new Vue({
  el: '#vuetest',
  data: { "sentence": "江流宛转绕芳甸月照花林皆似霰", "special": "花" }
})
</script>
```

浏览器解析

含有Vue模板标签的DOM树



Vue.js解析

```
<div id="vuetest">
  <div>
    <div class="normal">江</div> flex
    <div class="normal">流</div> flex
    <div class="normal">宛</div> flex
    <div class="normal">转</div> flex
    <div class="normal">绕</div> flex
    <div class="normal">芳</div> flex
    <div class="normal">甸</div> flex
    <div class="normal">月</div> flex
    <div class="normal">照</div> flex
    <div class="normal special">花</div> flex
    <div class="normal">林</div> flex
    <div class="normal">皆</div> flex
    <div class="normal">似</div> flex
    <div class="normal">霰</div> flex
  </div>
```

浏览器绘制

江 流 宛 转 绕 芳 甸 月 照 花 林 皆 似 霰

# 使用AngularJS

## 含有Angular指令的HTML

```
<div ng-app=""  
  ng-init='data={"sentence":"江流宛转绕芳甸月照花林皆似霰","special":"花"}'  
  <div class="normal"  
    ng-repeat="char in data.sentence.split('')"  
    ng-class="char == data.special ? 'special' : ''">  
    {{char}}  
  </div>  
</div>
```

- ng-app指令指示这个div是AngularJS的所有者
- ng-init 指令初始化 AngularJS 应用程序变量。
- ng-repeat指令表示重复复制当前所在的div元素，以data.sentence.split('')返回的数组作为重复的条件。
- ng-class元素，将class与一个表达式绑定
- 双花括弧括起来的变量{{char}}实现插值

**Vue.js**与**AngularJS**的工作原理比较接近，都是通过JS库中实现了针对嵌入**DOM**的特殊指令标签的解释器，解释**v**-指令或者**ng**-指令为通用浏览器支持的**DOM**标签。

# 使用React + JSX

嵌入了JSX的HTML

```
<div id="reacttest"></div>
<script type="text/babel"> //调用babel前端翻译
  function Poem() {
    const data = {"sentence":"江流宛转绕芳甸月照花林皆似霰","special":"花"}
    const list = data.sentence.split('').map(
      char => (
        <div class={"normal " + (char == data.special ? "special" : "")}>
          {char}
        </div>
      )
    )
    return (<div>{list}</div>)
  }
  ReactDOM.render( <Poem/>, document.getElementById('reacttest') );
</script>
```

- 大写字母开头的Poem是一个React组件， ReactDOM.render在reacttest元素内更新Poem组件返回的元素。
- Poem组件通过JS数组的map方法返回一个div的列表。
- JSX代码会由babel翻译器翻译为普通JS代码。
- 单花括弧括起来的变量，例如{char} {list}，实现插值。

# JSX翻译器: babel

## JSX

```
function Poem() {  
  const data = {"sentence": "江流宛转绕芳甸月照花林皆似霰", "special": "花"}  
  const list = data.sentence.split('').map(char => (  
    <div class={"normal " + (char == data.special ? "special" : "")}>  
      {char}  
    </div>  
  ))  
  return <div>{list}</div>  
}  
ReactDOM.render(<Poem/>, document.getElementById('reacttest'));
```



## 基于React库的纯JS

```
function Poem() {  
  const data = {"sentence": "江流宛转绕芳甸月照花林皆似霰", "special": "花"};  
  const list = data.sentence.split('').map(char =>  
    React.createElement("div", {class: "normal " + (char == data.special ? "special" : "")}  
      , char  
    )  
  );  
  return React.createElement("div", null, list);  
}  
ReactDOM.render(React.createElement(Poem, null), document.getElementById('reacttest'));
```

# React的工作原理

HTML + 基于React的JSX



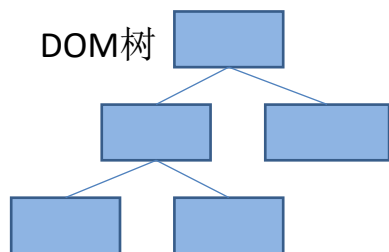
babel翻译

HTML + 基于React的纯JS



浏览器解析

DOM树



执行React库以及基于React库的JS应用逻辑，对DOM树进行更新

```
<div id="reacttest">
  <div>
    <div class="normal">江</div> flex
    <div class="normal">流</div> flex
    <div class="normal">宛</div> flex
    <div class="normal">转</div> flex
    <div class="normal">绕</div> flex
    <div class="normal">芳</div> flex
    <div class="normal">甸</div> flex
    <div class="normal">月</div> flex
    <div class="normal">照</div> flex
    <div class="normal special">花</div> flex
    <div class="normal">林</div> flex
    <div class="normal">皆</div> flex
    <div class="normal">似</div> flex
    <div class="normal">霰</div> flex
  </div>
</div>
```



浏览器绘制

江 流 宛 转 绕 芳 甸 月 照 花 林 皆 似 霰