**整型数据**

**Nginx使用的标准数据类型**

nginx主要使用3个标准数据类型：size\_t, u\_char, off\_t

**Nginx自定义整数类型**

定义在core/ngx\_config.h中

ngx\_int\_t：有符号整数

ngx\_uint\_t：无符号整数

ngx\_flag\_t：标志整数类型

typedef intptr\_t ngx\_flag\_t;

**无效值**

定义在core/ngx\_conf\_file.h中

#define NGX\_CONF\_UNSET -1

nginx使用NGX\_CONF\_UNSET代表无效值，包括指针，数值等，如果值无效，均为NGX\_CONF\_UNSET

**错误处理**

**错误码定义**

在nginx中我们需要抛出异常，nginx定义了如下异常码

// 定义在 core/ngx\_core.h 中

#define NGX\_OK 0 # 执行成功，无错误

#define NGX\_ERROR -1 # 执行失败

#define NGX\_BUSY -3 # 后台服务正忙

#define NGX\_DECLINED -5 # 执行成功，但为做处理

...

**内存池**

当一次http请求时，nginx会创建一个内存池，我们可以在里面任意的申请内存而不必担心释放问题，请求结束后nginx会自动销毁

**结构定义**

Nginx使用ngx\_pool\_t表示内存池

定义在core/ngx\_core.h中

struct ngx\_pool\_s {

    ...

    ngx\_pool\_cleanup\_t \*cleanup;    // 释放内存池的清理动作

    ngx\_log\_t \*log;     // 管理的日志对象

}

**操作函数**

void\* ngx\_pcalloc(ngx\_pool\_t \*pool, size\_t size); // 从内存池中分配size大小的内存

ngx\_int\_t ngx\_pfree(ngx\_pool\_t \*pool, void \*p); // 从内存池中释放释放内存

**清理机制**

Nginx会自动清理ngx\_pool\_t中申请的内存，但不会清理我们申请的其他系统资源（如：文件句柄）

Nginx使用ngx\_pool\_cleanup\_t结构封装清理动作

定义在/core/ngx\_palloc.h中

typedef struct ngx\_pool\_cleanup\_s ngx\_pool\_cleanup\_t;

struct ngx\_pool\_cleanup\_s {

    ngx\_pool\_cleanup\_pt handle;     // 清理动作函数

    void \*data;                     // 清理动作所需数据

    ngx\_pool\_cleanup\_t \*next;       // 下一个清理动作

}

操作：

// 向内存池申请一个ngx\_pool\_cleanup\_t，其data的大小为size

ngx\_pool\_cleanup\_t \*ngx\_pool\_cleanup\_add(ngx\_pool\_t \*p, size\_t size);

**字符串**

**Nginx自定义字符串**

// 定义在 ngx\_string.h 中

typedef struct {

    size\_t len;         // 字符串长度

    u\_char \*data;       // 字符串内容

}

**操作**

// 创建字符串

#define ngx\_string(str) { sizeof(str) - 1, (u\_char \*) str }

// 空字符串

#define ngx\_null\_string { 0, NULL }

// 字符串赋值，str 为 ngx 字符串，text 文c字符串

#define ngx\_str\_set(str, text) ...

// 设为空

#define ngx\_str\_null(str) ...

**时间与日期**

**时间结构**

typedef struct {

    time\_t sec;         // 自 epoch 以来的秒数

    ngx\_uint\_t msec;    // 毫秒数

    ngx\_int\_t gmtoff;   // gmt 时区偏移量

} ngx\_time\_t;

**时间操作**

// 获取当前时间毫秒数，ngx\_cached\_time 是当前时间的缓存

#define ngx\_time() ngx\_cached\_time->sec

// 获取当前时间结构

#define ngx\_timeofday (ngx\_time\_t \*) ngx\_cached\_time

**日期结构**

// 定义在 os/unix/ngx\_time.h 中

typedef struct tm ngx\_tm\_t;

struct tm{

    int tm\_mday;    // 日份

    ...

}

**日期操作**

// 定义在 core/ngx\_times.h

// ngx\_time\_t 结构转为 ngx\_tm\_t 结构

void ngx\_gmtime(time\_t t, ngx\_tm\_t \*tp);

// ngx\_time\_t 结构转为 日期字符串

u\_char \*ngx\_http\_time(u\_char \*buf, time\_t t);

**运行日志**

**结构**

// 定义在 core/ngx\_core.h

typedef struct ngx\_log\_s ngx\_log\_t;

**操作**

// 记录日志（不对等级进行过滤）

void ngx\_log\_error\_core(

    ngx\_uint\_t level,   // 日志等级

    ngx\_log\_t \*log,     // 日志结构

    ngx\_err\_r err,      // 系统调用返回的错误码，如果没有，则写 0

    const char \*fmt,    // 记录的文本

    ...

)

// 记录日志，推荐使用，参数同上，如果level小于或等于log->log\_level，则记录日志

#define ngx\_log\_error(...) ...

**日志级别**

#define NGX\_LOG\_ERR 4       // 错误，常用级别

#define NGX\_LOG\_WARN 5      // 警告，常用级别