Blog Archive 2025

2025 年博客文章合集

目录

Typst Syntax

2025-05-27

Raw Blocks

This is an inline raw block `class T`.

Examples

Inline Raw Blocks

Block Definition

This is an inline raw block `class T`.

This is an inline raw block `class T`.

This is a long inline raw block `class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {};`.

Js syntax highlight are handled by syntect:

```
class T {};
```

Typst syntax hightlight are specially handled internally:

```
#let f(x) = x;
```

Custom Blocks

Hello?

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英

H3 $\Delta = 2$

$$ax^2 + bx + c = 0$$

aaa

```
f("foo")

def foo do
  "bar"
end
```

New Blog Debut

2025-08-23

使用新的技术栈创建了新的博客网站。

技术栈：

- Astro
- Typst
- Tailwind CSS

Hello

2025-08-24

Hello World!

这是一篇测试用文章，用来测试评论功能。

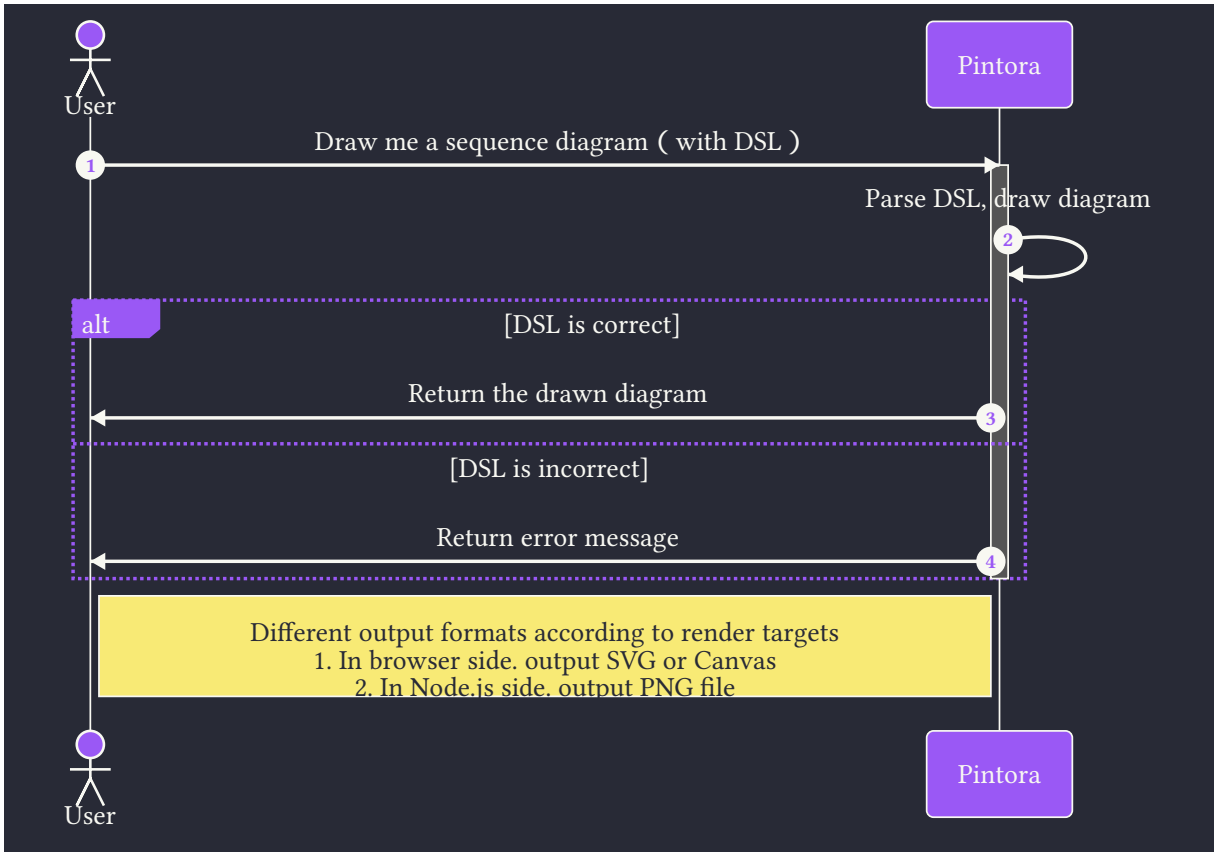

Figure 1: Sequence Diagram Example

使用 Cloudflare D1 与 Astro 为博客添加评论功能

2025-08-30

自从博客部署到 Cloudflare 后，我就一直在寻找一个合适的评论系统。自我建立博客网站一开始，我的博客使用的是 Disqus，但是 Disqus 在国内访问速度很慢，而且有时候还会被墙，所以我就放弃了。

后来了解了 Valine，一直想要尝试一下，但因为太懒了没有尝试。

直到最近，我了解到了 Cloudflare D1，以及 Astro 的 Drizzle ORM，所以我就想要尝试一下，为我的博客添加评论功能。

技术栈

- Cloudflare D1
- Astro
- Drizzle ORM
- Bun

实现步骤

创建数据库以及表

在项目中可以通过 wrangler 命令行工具创建数据库以及表。

```
bunx wrangler d1 create wonderland
```

然后我在项目中放置了 migrations/0001_init.sql 数据库迁移脚本。

```
DROP TABLE IF EXISTS comments;
CREATE TABLE
  IF NOT EXISTS comments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    post_slug TEXT NOT NULL, -- 文章唯一标识/slug
    author_name TEXT NOT NULL,
    author_email TEXT NOT NULL,
    author_url TEXT NOT NULL,
    content TEXT NOT NULL,
    created_at INTEGER NOT NULL DEFAULT (unixepoch ()),
    parent_id INTEGER,
    ip_hash TEXT
  );

CREATE INDEX IF NOT EXISTS idx_comments_post_created ON comments (post_slug, created_at DESC);

INSERT INTO comments (post_slug, author_name, author_email, author_url, content) VALUES ('test-slug', 'John Doe',
'john.doe@example.com', 'https://example.com', 'This is a test comment');
INSERT INTO comments (post_slug, author_name, author_email, author_url, content) VALUES ('test-slug', 'John Doe',
'john.doe@example.com', 'https://example.com', 'This is a test comment');
```

然后通过 wrangler d1 execute 命令执行数据库迁移脚本。

```
bunx wrangler d1 execute wonderland --local --file=./migrations/0001_init.sql
```

D1 是一个非常神奇的数据库，它的命令可以分 本地 和 远程 执行，只需要在命令中添加--local/--remote 参数即可，这大大方便了开发和调试，意味着你本地可以随便折腾，不用担心对生产环境造成影响。

在编译 NBIS 时强制使用小端序

2025-09-14

在编译 NBIS 时，按照正常情况来说，源码自带 Makefile 会根据系统环境自动判断是否使用小端序。但是在实际使用中发现，这个配置在我的系统上不生效。

环境

系统：Ubuntu 22.04 WSL2

问题

使用 NBIS 工具包是为了对 WSQ 文件进行解析。WSQ 数据文件是以大端序来排列字节的。一开始我并不知道大小端序的问题，直接编译生成链接库然后调用解析。正常情况下，大端序数据在进行内存拷贝的时候会自动进行大小端序的转换。但是当我把数据从 WSQ 文件中读取出来之后，直接进行内存拷贝，发现数据并没有被正确转换，导致标记字节反序：

```
// 数据中的大端序标记（程序应当读取到的正确标记）
0xFF 0xA0
// 转换后内存中的小端序标记（程序读取到的错误标记）
0xA0 0xFF
```

查看源码后发现，在读取数据的函数 getc_ushort 中，程序会根据一个__NBISLE__宏来判断是否要对两个字节进行交换：

```
#ifdef __NBISLE__
  swap_short_bytes(shrt_dat);
#endif
```

但是这个宏明明由 Makefile 生成，却没有生效。我试过了各种通过 make 命令传入参数的方式，但都没有起效。

解决方案

最终我决定直接改源码，在源码中直接写死__NBISLE__宏：

```
#ifndef __NBISLE__
#define __NBISLE__ 1
#endif
```

这样可以绝对确保程序在读取的时候一定能够读取到__NBISLE__宏。

经过一番 AI 和搜索，最终确定需要改的位置为：

- commonnbis/include/defs.h 中加入宏定义
- imgtools/include/ioutil.h 中引用上面的头文件

这样修改完编译之后，程序终于能够正确读取到数据中的大端序标记了。

Make NBIS's wsq_decode_mem Thread Safe

2025-09-15

Recently, I'm working on a project to decode WSQ files which requires to use NBIS's wsq_decode_mem function. But I found that the function is not thread-safe. After coding with AI, I made several changes to the source code to make it thread-safe.

Problem

The function is not thread-safe because it uses several global static variables to store the state of the decoder. When my program is running in parallel, the variables will be overwritten by different threads, causing the decoder to crash.

Solution

Modify source code to use thread-local variables.

In imgtools/include/wsq.h insert,

```
#if defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 201112L)
#define WSQ_TLS _Thread_local
#elif defined(__GNUC__) || defined(__clang__)
#define WSQ_TLS __thread
#else
#define WSQ_TLS
#warning "No thread-local storage; WSQ decode won't be thread-safe."
#endif
```

In imgtools/include/wsq.h, replace the global static variables with thread-local variables,

```
extern int debug;
extern WSQ_TLS QUANT_VALS quant_vals;
extern WSQ_TLS W_TREE w_tree[];
extern WSQ_TLS Q_TREE q_tree[];
extern WSQ_TLS DTT_TABLE dtt_table;
extern WSQ_TLS DQT_TABLE dqt_table;
extern WSQ_TLS DHT_TABLE dht_table[];
extern WSQ_TLS FRM_HEADER_WSQ frm_header_wsq;

/* hifilt/lofilt are read-only constants, so no need to change TLS */
extern float hifilt[];
extern float lofilt[];
```

In imgtools/src/lib/wsq/globals.c, replace the global static definitions with thread-local variables,

```
#ifdef TARGET_OS
  WSQ_TLS QUANT_VALS quant_vals;

  WSQ_TLS W_TREE w_tree[W_TREELEN];

  WSQ_TLS Q_TREE q_tree[Q_TREELEN];

  WSQ_TLS DTT_TABLE dtt_table;

  WSQ_TLS DQT_TABLE dqt_table;

  WSQ_TLS DHT_TABLE dht_table[MAX_DHT_TABLES];

  WSQ_TLS FRM_HEADER_WSQ frm_header_wsq;
#else
  WSQ_TLS QUANT_VALS quant_vals = {};

  WSQ_TLS W_TREE w_tree[W_TREELEN] = {};

  WSQ_TLS Q_TREE q_tree[Q_TREELEN] = {};

  WSQ_TLS DTT_TABLE dtt_table = {};

  WSQ_TLS DQT_TABLE dqt_table = {};

  WSQ_TLS DHT_TABLE dht_table[MAX_DHT_TABLES] = {};

  WSQ_TLS FRM_HEADER_WSQ frm_header_wsq = {};
#endif
```

In imgtools/src/lib/wsq/decoder.c, replace static variables with thread-local variables in nextbits_wsq and getc_nextbits_wsq,

```
static WSQ_TLS unsigned char code;  /*next byte of data*/
static WSQ_TLS unsigned char code2;  /*stuffed byte of data*/
unsigned short bits, tbits;  /*bits of current data byte requested*/
int bits_needed;    /*additional bits required to finish request*/

                    /*used to "mask out" n number of
                      bits from data stream*/
static unsigned char bit_mask[9] = {0x00,0x01,0x03,0x07,0x0f,
                                    0x1f,0x3f,0x7f,0xff};
```

bit_mask is a read-only constant, so no need to change TLS.

The code and code2 variables are used to store the next byte of data and the stuffed byte of data, respectively. After changing TLS, the variables will be stored in the thread-local storage, so they will not be overwritten by different threads.

Result

Multi-thread problem solved :-)

After modifying these variables, the wsq_decode_mem function is now thread-safe, so I built it and used it in my project with multiple threads, turned out to be working fine.