

# Xylem

## 内容创作平台及管理系统的设计与实现

---

GES<sup>1</sup> SYL<sup>1</sup> WJX<sup>1</sup> WPS<sup>1</sup> ZQY<sup>1</sup>

2023 年 11 月 8 日

<sup>1</sup> 中国农业大学烟台研究院

---

<sup>1</sup> 小组成员按拼音排序。

## 背景

Where to start?

## 总结

## 设计

战略设计

消化知识与细化概念

## 管理

## 总结

## 实现

技术选择

建立项目

数据与关系

功能与服务的实现

开发与发布

总结

# 背景

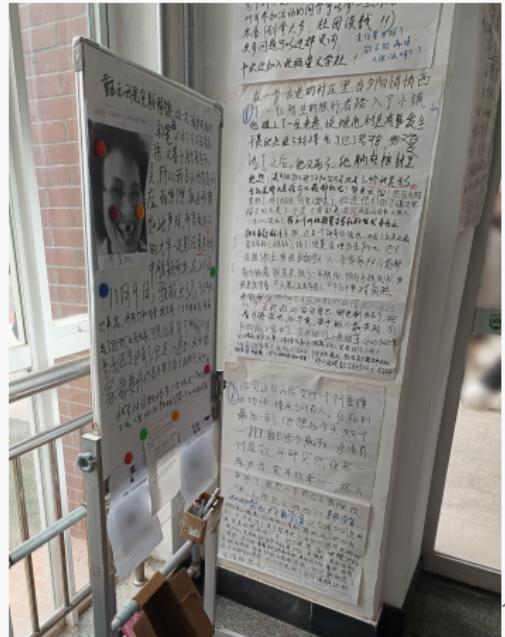
---

# 想法来源

想法的来源是周五（11月3日）在教学楼A区门口看到的小说接龙。

## 网络平台的接龙小说创作

知名度较高的包括 SCP 基金会、克苏鲁神话、银河系漫游指南、各类爱好者的 Wiki 等。甚至跑团也可以理解为通过接龙的形式在创作一部小说。



<sup>1</sup> 拍摄于 11 月 3 日下午两点，考虑隐私故对画面内的二维码以及行人进行模糊处理。

# 原型：赛博小说接龙



**目的：**设计一个轻量且易于部署的协同创作平台。

平台在形式上为从生产者向读者传播内容的载体；内容的展现形式遵循树状结构。而树木的骨架为运输水分的木质部，故起名为 Xylem 。

<sup>1</sup>以上是对内容的部分文段的展示。cr. CAU 北极星文学社

# 思路与前置知识

**开发思路：**产生想法后用最擅长的技术快速出活，反复迭代。

**涉及到的额外知识：**假设对课程内容有所了解

- MVC 架构
- 一些编程范式（例如面对对象、函数式等）
- 基本的 HTML 语法

课本之外的专业术语会在对应的页予以解释。

## 代码实例

```
defmodule Xylem.Plugs.Auth do
  @behavior Plug

  def init(opts), do: opts

  @doc """
  Invoked when used.
  """

  def call(conn, _opts) do
    conn
    |> fetch_token
    |> query_session
    |> Users.get_by_id
  end
end
```

为方便说明展示会列出部分代码，至多如上所示。

# 关于展示<sup>1</sup>

```
presentation = dict(  
    name="Research a MIS",  
    required_parts=[  
        "background", "analyse requirement", "designing",  
        "techstack", "envoirment", "comment"  
    ],  
    time=lambda t: t <= 5  
)  
  
presentation["name"].replace("Research", "Design")  
presentation["reference"] = dict(map(  
    lambda site: {site, presentation["required_parts"]},  
    ["zhihu", "tieba", "Discuz!", "Discord"]  
) # Research many MISS.  
presentation["required_parts"] = ["designing", "implementation"]
```

<sup>1</sup>用编程语言来描述内容，是技术展示中常见的方法

## 自然语言版本<sup>1</sup>

这段 Python 代码展示了一个项目展示的相关信息和一些操作。presentation是一个字典，包含项目展示的相关信息。

- "name"代指项目名称为"Research a MIS"
- "required\_parts"是一个包含字符串的列表，表示项目所需的不同部分
- "time"是一个函数，它接受参数t，并返回True如果t小于等于 5，否则返回False，用于检查展示是否在规定的时间内完成
- presentation["name"].replace("Research", "Design")这一行替换了之前的字符串，使之变成了"Design a MIS"
- presentation["reference"]是一个通过map函数构建的字典，将不同的站点名与项目所需的部分关联起来，赋值给该键
- 最后将字典中的presentation["required\_parts"]改成了[ "designing", "implementation" ]

需要注意的是，代码中的lambda函数和map操作可以使用更明确的方式来构建字典，以提高可读性。

<sup>1</sup>由 ChatGPT 提供技术支持。

# 总结

- 本展示大多以开发者的角度出发，开发一个叫做 Xylem 的系统
- 我假定你们对管理信息系统课程的内容有所了解
- 对于额外或超纲的内容将以脚注的方式呈现
- 除声明外，每节的内容均独立成文
- 每节最后都会对本节内容进行总结

# 设计

---

# 需求与核心功能分析

赛博接龙小说的问题：

- 世界观不连贯
- 捣蛋鬼来捣蛋（互联网平台的通病）<sup>1</sup>
- 前期作者的作品对后续作者的发挥带来限制
- 接龙创作的线性内容形式不利于从群体中选出最好的

**解决方案：**以内容为主；除了发起人声明外默认可以随意创作（从任何地方开始）。

---

<sup>1</sup>在互联网，没人知道你是一条狗。

# 用户

按照 Xylem 的设计理念，平台将大多数用户定义为内容生产者以及读者。其中，前者负责生产内容，而后者对内容进行评价。因此用户管理规范相比其他的系统有较大的差异。

其中包括用户的注册与版权问题，因为内网的情况前者不需要邮件，如果用户绑定邮件可以选择内容的版权许可证，否则默认公有领域。

## 版权争端

早期知乎的知识产权协议近乎霸王条款：「对于用户发表到知乎上的任何内容，用户同意知乎在全世界范围内具有免费的、永久性的、不可撤销的、排他性的和完全再许可权的权利和许可，以行使和保护与内容相关的全部著作权。」

后期知乎更新了著作权协议，可以在禁止转载到可以转载二者之间选择。

# 协同创作的实现

## *Scenario i: 接龙写作:*

以句子或段落为基本单位，用户均可对公开故事进行续写（可以编辑自己写的部分），并且可以对其他作品表态或评论。（类似于论坛但可以随意接龙，详见最后一节）

## *Scenario ii: 协同创作:*

在同一世界观下共同创作（类似于 Wiki，但也可以在前者的机制下实现）

## *Scenario iii: 跑团*

# 文字创作

在形式上与论坛有相同但是存在差别，故使用新的名称<sup>1</sup>，例如：

- 全站点的管理层 → 管理团队
- 楼主/PO 主 → 发起人
- 帖子 → 故事
- 跟帖 → 续写
- 草稿箱外每次提交的记录 → 记录

## DDD 的模式：通用语言 (Ubiquitous Language)

通用语言是领域驱动设计 (DDD) 开发中的使用的语言（包括术语以及描述方式），用于确保开发团队和业务团队之间使用相同的术语和概念，以便更好地理解和沟通领域需求，从而提高软件开发的质量和效率。

<sup>1</sup>部分内容参考了 FimTale 的用户手册

## 内容的格式与改动

内容的格式采用 Markdown 或 BBCode<sup>1</sup>，采用句子或分段为分割单位，选择什么取决于用户的要求。

Xylem 给予用户对内容的修改权力，同时也允许用户回滚到之前的某次修改。

对于表态的排名，对根据修改内容站之前内容的比重进行加权处理，以作为展现层的排名的依据。

当某内容被删除时，后续内容会根据被删除的原因而不可添加内容/设为不可见/标记“上游信息被删除，可能无法保证内容连贯”的措施等处理。

---

<sup>1</sup>以上均属于轻量级标记语言，使用简单的记号来对内容的格式进行标识。

# 内容管理

Xylem 的原则是「为所涉及的内容负责」，但考虑到可能有用户无法遵循该原则，因此需要引入管理机制。

Xylem 的内容管理权限将从维护者到发起人；从发起人到所发起的故事（主要为控制可见度）；用户则对其所发表的内容负责。

处理手段包括：

- 向用户发送警告
- 内容不能被表态或评论
- 删除内容
- 删除内容并冻结账号
- 封禁 IP，将日志上报学校（Xylem 在内网部署）

# 面向维护者的功能

面向维护者的功能主要包括控制台 (dashboard) 以及统计 (statistics)。

功能包括：

- 系统状态
- 服务列举与配置<sup>1</sup>
- 依赖配置与占用资源
- 用户统计
- 故事统计
- 推送设定

---

<sup>1</sup>参见最后一节「二次开发」。

# 总结

- 为了专注于创作，以内容作为第一出发点而非用户
- 内容以多种形式表达，主要是类似论坛的发帖回帖制，可能也会提供可以即时通信的聊天室
- 接龙形式的内容只有部分反响较好者可以在帖子/平台的时间线被所有人访问，但是所有未被删除的内容均可通过链接直接访问
- 除用户同时在平台以及社交网站声明外，所生产内容的版权默认属于公共领域，对于没有版权声明且可访问的内容，提供导出功能
- 内容管理的权限通过管理团队到对应故事的发起人、发起人到参与续写的用户的多层赋权实现
- 用户可以对平台内的公共领域的内容做出回应、评论，其能够影响展示的排名（存在不同的排名机制），也是被导出的一部分

# 实现

---

# 战术设计

**一句话：**关于如何实现软件系统的具体细节和技术方案的计划和决策。

包括但不限于架构选择、编程语言选择、代码规范确定、代码审查机制确定、测试规范等等。

## MVC 架构

MVC 架构 (Model-View-Controller) 是一种用于组织和分离应用程序代码的设计模式。在 MVC 中，模型 (Model) 负责管理应用程序的数据和业务逻辑<sup>1</sup>，视图 (View) 负责呈现数据和用户界面，而控制器 (Controller) 充当数据和用户界面之间的中介，接收用户输入并协调模型和视图之间的交互。该架构使代码更易于维护和扩展，因为不同部分之间的职责被清晰划分，允许独立地修改和更新每个部分而不会影响整个应用程序。

<sup>1</sup>对于业务逻辑应该放到哪里，不同的 MVC 框架有不同的处理方案

# 运行环境与技术栈

路线：快速开发、方便部署/跟随潮流、挑战新奇。

## 技术栈<sup>1</sup>

包括开发所用到的编程语言、框架、数据库、操作系统、开发工具、版本控制系统等。

可以以数据架构图的形式存在，也可以以组件首字母缩写表示。

选择的依据更多的取决于项目需求以及你所有的技能树。

Xylem 打算选什么？根据所会技术，选择

Python/Sanic/PicoCSS/Alpine.js。（运行环境：Ubuntu/Windows + SQLite + Redis(Optional)）

<sup>1</sup>把一项产品所用到的一系列相互关联的技术和工具堆起来就叫做技术栈。

## 他山之石：其他论坛的技术选择

- NodeBB 基于 Node.js 开发
- Discuz! 基于 PHP 开发
- discourse 基于 RoR 开发，只能在 Docker 上运行
- discord 基于 Elixir 开发，维护人少并发能力强，但函数式编程存在门槛

# 解 (chao) 剖 (xi) 开源信息系统的方法/流程

(假设已经做好背调)

- 本地部署，探索系统的功能
- 查看包括用户手册、开发文档、API 文档等的项目文档，了解基本信息
- 根据配置文件以及文件结构，了解项目的各部分
- 对数据模型进行研究，包括数据表以及各层次的数据
- 结合功能与数据，阅读源代码对功能的实现方式进行研究

# 项目的组织与配置

```
ROOT
+-config/
+-instance/
+-tests/
+-priv/ # Inspired by Phoenix
| +-assets/
| +-template/
| +-pics/
+-xylem/
| +-adapter/
| +-helpers/
| +-infra/
| +-services/
| +-xylem_core/
| +-xylem_domain/
| +-xylem_web/
+-run.bat
+-README.txt
```

项目采用了整洁架构（Clean Architecture）的变体。

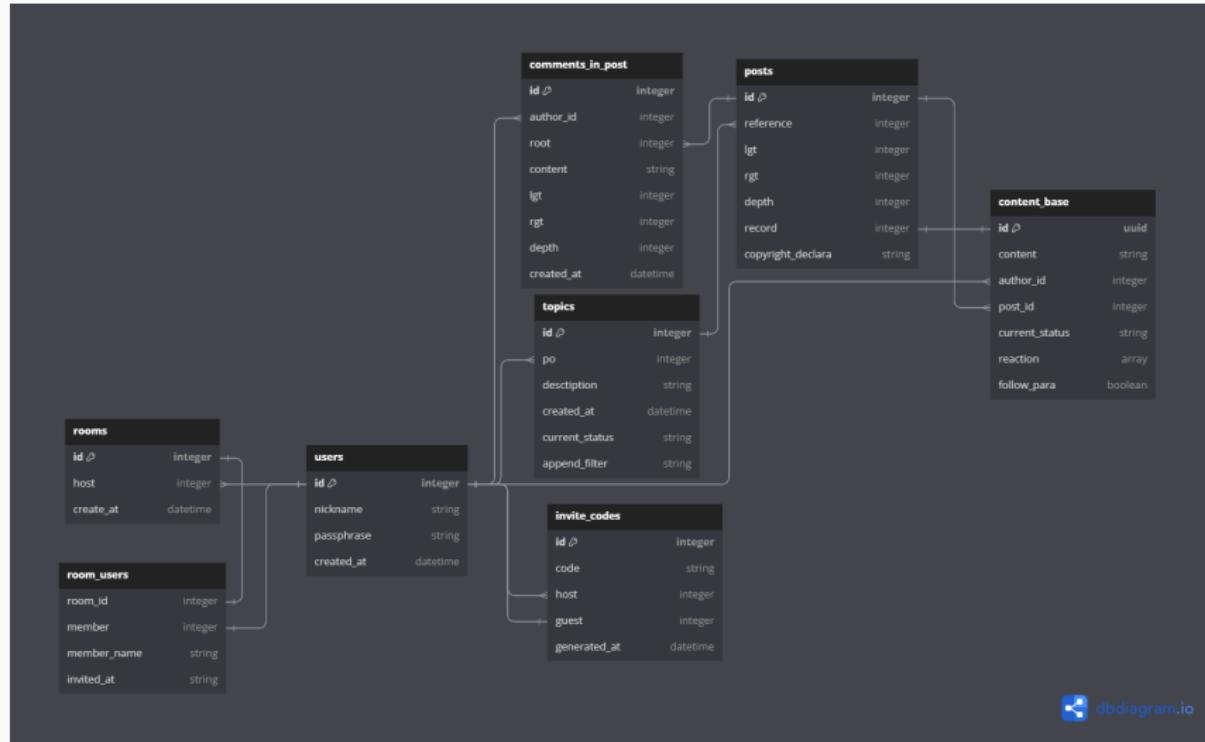
## 库、框架与架构的区别

库是一系列预先编写好的代码集合，供开发者在编程中调用。

框架是会介入编程人员工作的库，负责具体实现程序员书写的业务，能够接管程序的控制流。

架构是高度抽象的需求，是系统中的不变量。随着业务的抽象在不同的层级有不同的含义。

# 数据表关系



<sup>1</sup>跑团部分过于复杂，不予考虑

# 树形关系的实现

对内容的读写符合以下

规则：

查看某些有价值或令人  
忍俊不禁的 ➞ 添加 > 查  
看所有的

## 嵌套集

嵌套集是一种用于组织  
和表示层次结构数据的  
数据库模型。核心是使用  
两个字段来表示节点在  
树中的位置：左边界  
(lft) 和右边界 (rgt)。

-- Use comment as example.

```
CREATE TABLE comments(
    id INTEGER PRIMARY KEY,
    content TEXT,
    root_id INTEGER FOREIGN KEY,
    lft INTEGER,
    rgt INTEGER,
    create_at DATETIME,
    likes INTEGER,
    -- Optional
    depth INTEGER,
);
```

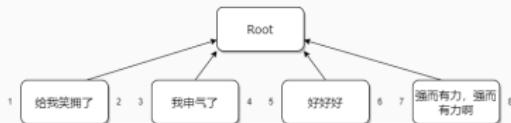
-- Insert comments.

```
INSERT comments VALUES
```

```
(1, "给我笑拥了", 1, 1, 2, NOW(), 0, 1)
(2, "我申气了", 1, 3, 4, NOW(), 0, 1)
(3, "好好好", 1, 5, 6, NOW(), 0, 1)
(4, "强而有力, 强而有力啊", 1, 7, 8, NOW(), 0,
→ 1);
```

# 树形关系的实现（简单查询）

例如：返回【一条】已知 id 的评论。

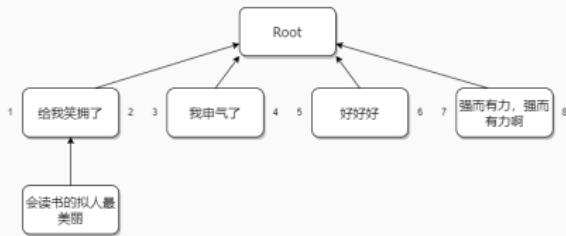


```
-- Read comment.  
SELECT ( comments.id,  
comments.root_id,  
comments.content,  
comments.create_at  
) FROM comments WHERE ( comments.root_id =  
AND comments.id = 4  
);
```

将返回：4, 1, " 强而有力，强而有力啊",  
2023/11/08T10:25:03

# 树形关系的实现（插入）

插入一条新评论<sup>1</sup>。



```

INSERT INTO comments VALUES(
    5, "会读书的拟人最美丽",
    1, 2, 3, NOW(),
    0, 2
);
    
```

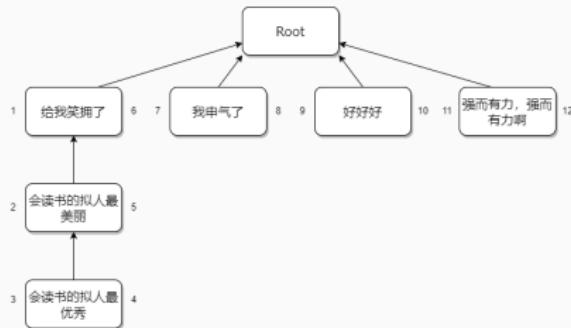
```

UPDATE comments SET
    lft = (
        CASE WHEN lft >= 3 THEN lft +
        → 2 ELSE lft END
    ), rgt = (
        CASE WHEN rgt >= 2 THEN rgt +
        → 2 ELSE rgt END
    )
WHERE root_id = 1;
    
```

<sup>1</sup>绘图只有一部分。

# 树形关系的实现（复杂查询）

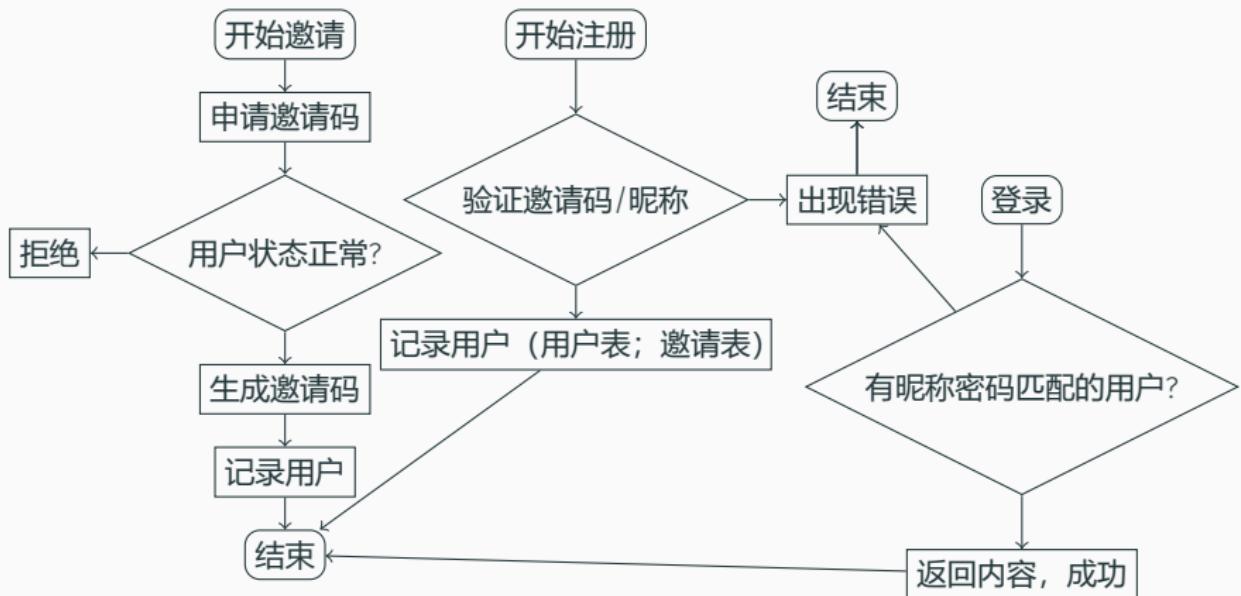
查询某评论下【所有】的子评论。



```
SELECT (
  comments.id,
  comments.root_id,
  comments.content,
  comments.create_at,
  comments.depth
) FROM comments WHERE (
  comments.lft >= 1 AND
  comments.rgt <=6
);
```

# 用户邀请、注册与登录

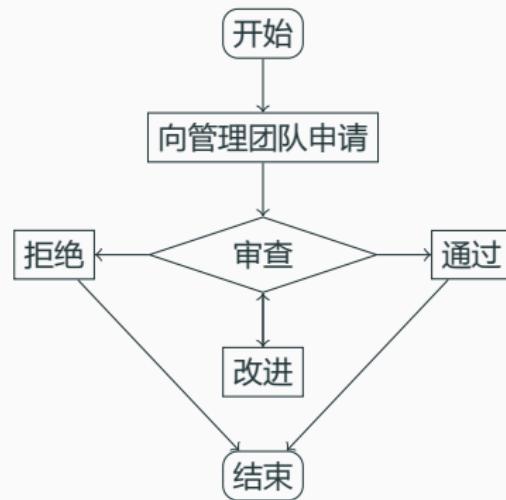
邀请与注册：



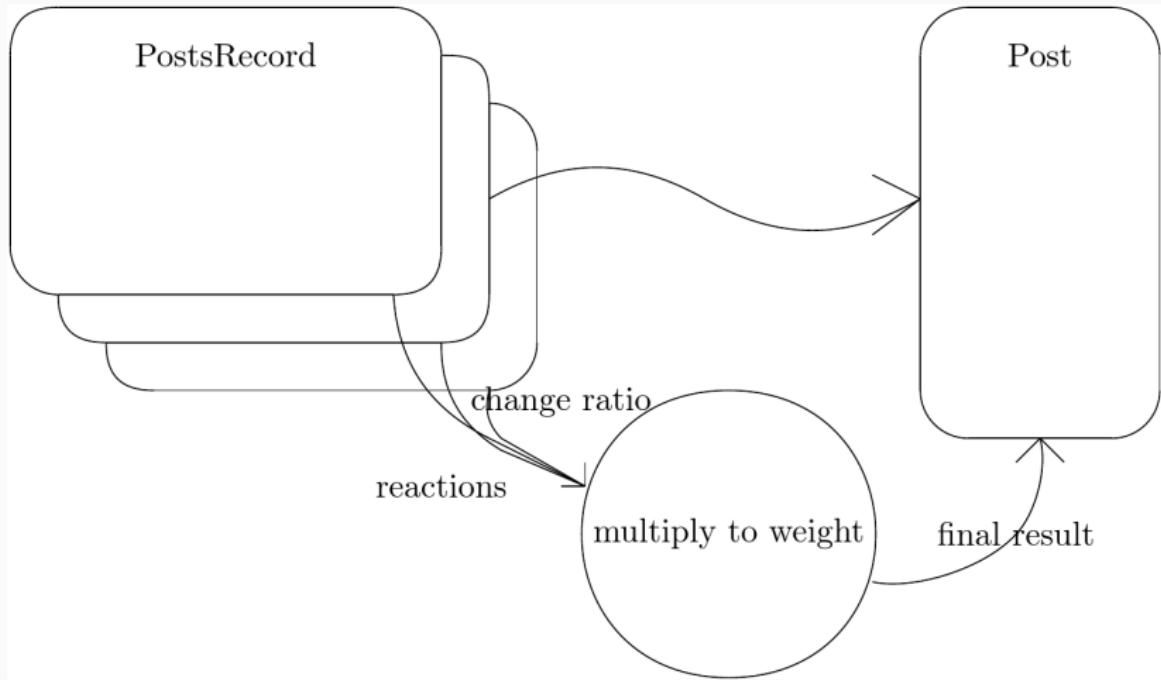
登录：

# 创建故事

欢迎旅行者，接下来等着你的居然是...



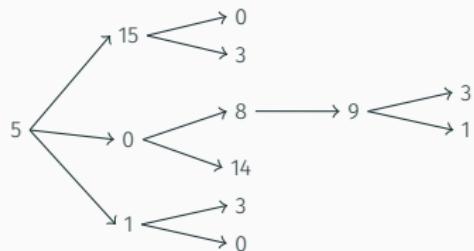
# 创建并修改内容



<sup>1</sup>由 TikZit 生成。

## 对内容的评价体系

我们将内容看成有  $n$  个节点的树  $T$ ，其中值就是对单个节点的评价。



对于没有子节点的节点（还没人回复的某内容） $T_i$  来讲，该讨论串（从最开始的内容到该回复的所有内容）的值就是可计算的。

对内容的评价就取决于该值与该节点的深度。

计算的方式可以是累加，也可以先通过算法进行修正再累加。

# 牛顿冷却定律：对评价的排序算法的实现

将内容的「热度」看成自然的温度，运用自然科学的方法解决问题：

$$T'(t) = -k\Delta T \quad (1)$$

$$T'(t) = -k(T - H) \quad (2)$$

$$\frac{dT}{dt} = -k(T - H) \quad (3)$$

$$\frac{dT}{T - H} = -kdt \quad (4)$$

取积分，得：

$$\int \frac{1}{T - H} dT = -k \int dt$$

## 牛顿冷却定律 (续)

可得：

$$\ln(T - H) = -kt + C$$

两边同时指数，得：

$$T - H = Ce^{-kt}$$

在时间  $t_0$  存在已知的  $T_0$ ，有：

$$T_0 - H = Ce^{-kt_0} \tag{5}$$

$$C = (T_0 - H)e^{kt_0} \tag{6}$$

## 牛顿冷却定律 (续)

将其代入，可得：

$$T = (T_0 - H)e^{k(t_0 - t)} + H$$

没有热度时  $H = 0$ ，有：

$$T = T_0 e^{-k(t-t_0)}$$

其中  $k$  为大于零的常数， $T_0$  为  $t_0$  时的温度。

# 牛顿冷却定律的实现

```
import math

def newtons_law_of_cool(
    T_prev: int | float, delta: float, k: float
) -> float:
    return T_prev * math.exp(-k * delta)

>>> print(newtons_law_of_cool(100, 0.5, 1))
60.653065971263345
```

# 类聊天室的设计

## 他山之石：TRPG Engine

TRPG Engine 是一款帮助用户更加方便跑团的即时通信软件。该项目的定位是帮助用户更快很好的进行沟通与交互的引擎，也让主持人将精力专注于剧情、场景上。在提供即时通信服务的基础上提供了一站式跑团（包括车卡、骰子等）的服务。

在前端界面上和 Discord 类似（都是基于 React 进行开发）。

TRPG Engine 的实时消息推送使用 WebSocket 实现。基于 Socket.io 框架进行开发，该服务被应用注册到中间件（Middleware）<sup>1</sup>。

由于源代码中数据库的迁移记录有近百条，数据库共计几十张表以及一千多行列。加之没有 Node.js 的开发经验，故放弃深入研究。

<sup>1</sup>中间件是计算机软件中的「传话者」，协同其他组件相互协调。

# 网络安全

业务上的安全：

- 用户登录解决方案 (Session Cookie)
- 用户的密码混淆
- 访问控制
- 发帖控制

基层设施的安全：

- 对来自客户的请求频次设置阈值 (反脚本)
- XSS、 CSRF
- 反向代理

# Pluggable Xylem: 二次开发与可插拔模块设计

项目将所有的功能都写到了 /xylem/services/ 内，并通过依赖注入<sup>1</sup>（演示见下）的方式被注册到了应用中。因此二次开发仅需要将功能封装成服务，注册到中间件或路由函数即可。

```
class UserAnalyseService(XyService):
    orinted: [xylem.xylem_core.user,
              ← xylem.xylem_core.content_base]

    def __fetch__(self):
        pass

    ...

```

```
# In handler module.

async def write(
    request: Request, service:
              ← UserAnalyseService
) -> HTTPResponse:
    ...

```

<sup>1</sup>依赖注入允许你将一个对象的依赖关系（通常是其他对象或服务）从外部传递给该对象，而不是在对象内部硬编码这些依赖关系。这使得代码更加松散耦合、可维护和可测试。

# 软件生命周期与交付

版本号按照 `{major}.{minor}.{micro}` 的格式。由于业务需求相对稳定，因此在正式版（v1.0.0）发布后主要以维护为主。

在开发完成部署后需要注意的方面：

- 生产环境与开发环境
- 应用的配置

## 部署

考虑到使用场景，最好的方法是把 Xylem 打包成一个可执行的应用文件。除此之外，额外需要的程序仅包括 SQLite。

当然，如果需要长期运行的话，也支持基于 Docker 的部署服务。

# 总结

- 从实现细节描述了调研系统的思路与方法
- 介绍了多种实现功能需要的算法
- 本节没什么要总结的

# 附录

---

# 媒体资源

- Slides:  $\text{\LaTeX}$  + Beamer + METROPOLIS
- **完颜慧德老师小说接龙的照片**: 拍摄
- **评论的节点图**: 使用免费软件 draw.io 绘制
- **关系图**: 使用 dbdiagram.io 的服务生成

**感谢大家收听!**