链滴

思源笔记折腾记录 - 做一个白板 - 更多的视图

作者：leolee

原文链接：https://ld246.com/article/1670305177796

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

一、前情提要：

之前实现了一个简单的白板工具，并且让它能够实现显示子文档了。

思源笔记折腾记录 - 做一个白板 - 保存数据和显示模式 - 链滴 (ld246.com)

就像这样：

不过只有子文档视图显然不给力啊。

所以我们来给它增加一些视图。

二、实现过程：

1、实现sql适配器

之前我们的子文档视图适配器是这样获取数据的：

```
获取卡片列表: async () => {
        return await 核心api.sql(
            {
                stmt: `select * from blocks where path like '%${id}%' and type='d'`
            }
        )
    },
```

很容易想到，那把不同的sql喂给这个适配器就能够实现不同的视图了嘛。

所以先来实现一个通用的sql适配器算了。

这次我们使用类class来试一下：

```
import { 核心api,获取地址参数 } from ".."
```

```
let 白板id = 获取地址参数().id
let mode = 获取地址参数().mode

if (!白板id) {
    白板id = '20200812220555-lj3enxa'
}
if (!mode) {
    mode = "子文档视图"
}

export class 基础块适配器{
    获取卡片内容数据=async (卡片id) => {
        return await 核心api.exportPreview(
            { id: 卡片id }
        )
    }
    获取卡片几何数据= async (卡片id) => {
        let 原始数据 = await 核心api.sql(
            {
                stmt: `select * from attributes where  name = 'custom-whiteBoardData-${白板id}' an
block_id='${卡片id}'`
            }
        )
        if (原始数据 && 原始数据[0]) {
            let json数据 = JSON.parse(原始数据[0].value)
            return json数据.find(
                item => { return item.mode === mode }
            )
        }
    }
    保存卡片几何数据= async (卡片id, 显示数据) => {
        let 原始数据 = await 核心api.sql(
            {
                stmt: `select * from attributes where  name = 'custom-whiteBoardData-${白板id}' an
block_id='${卡片id}'`
            }
        )
        if (原始数据 && 原始数据[0]) {
            let json数据 = JSON.parse(原始数据[0].value)

            json数据.find(
                item => { return  item.mode === mode }
            ).data = 显示数据
            let obj = {}
            obj.id =卡片id
            obj.attrs ={}
            obj.attrs[`custom-whiteBoardData-${白板id}`]=JSON.stringify(json数据)
            await 核心api.setBlockAttrs(obj)
        } else {
            let obj = {}
            obj.id =卡片id
            obj.attrs ={}
            obj.attrs[`custom-whiteBoardData-${白板id}`]=JSON.stringify([{
                mode: mode,
```

```
            data: 显示数据
        }])
        await 核心api.setBlockAttrs(obj)

      }
    }

}
```

这是一个最基本的思源块适配器，实现了保存几何信息到块属性的功能。

然后我们继承并扩展一下它：

```
import { 核心api,获取地址参数 } from ".."
import {基础块适配器} from './baseBlock'
let 白板id = 获取地址参数().id
let mode = 获取地址参数().mode

if (!白板id) {
    白板id = '20200812220555-lj3enxa'
}
if (!mode) {
    mode = "子文档视图"
}
export class 基础sql适配器 extends 基础块适配器{
    constructor(sql){
        super()
        this.sql =sql
    }
    获取卡片列表=async () => {
        return await 核心api.sql(
            {
                stmt: this.sql
            }
        )
    }
}
```

extends 后面的是基类，前面的基础sql适配器继承了这个类的各种方法，所以只需要再改一下获取片列表的方法就行了。

然后基于这个基础sql类，就可以更简单地实现之前的子文档适配器了：

```
import { 基础sql适配器 } from "./baseSQL"
import { 获取地址参数 } from ".."
let id = 获取地址参数().id||'20200812220555-lj3enxa'
let 子文档视图适配器 = new 基础sql适配器(
    `select * from blocks where type = 'd' and path like '%${id}%'`
)
export default 子文档视图适配器
```

好啦，这回子文档视图适配器就简单多了。

2、搜索适配器

然后来实现更多种类的适配器，比如说：

```
import { 基础块适配器 } from "./baseBlock";
import {核心api,获取地址参数} from '..'
import 'http://127.0.0.1:6806/stage/protyle/js/lute/lute.min.js'
let lute =Lute.New()
lute.SetTextMark(true)
lute.SetProtyleWYSIWYG(true)
lute.SetBlockRef(true)
lute.SetFileAnnotationRef(true)
lute.SetKramdownIAL(true)
lute.SetTag(true)
lute.SetSuperBlock(true)
lute.SetImgPathAllowSpace(true)
lute.SetGitConflict(true)
lute.SetMark(true)
lute.SetSup(true)
lute.SetSub(true)
lute.SetInlineMathAllowDigitAfterOpenMarker(true)
lute.SetFootnotes(false)
lute.SetToC(false)
lute.SetIndentCodeBlock(false)
lute.SetParagraphBeginningSpace(true)
lute.SetAutoSpace(false)
lute.SetHeadingID(false)
lute.SetSetext(false)
lute.SetYamlFrontMatter(false)
lute.SetLinkRef(false)
lute.SetCodeSyntaxHighlight(false)
lute.SetSanitize(true)

export class 搜索适配器 extends  基础块适配器{
    constructor(关键词){
        super()
        if(!关键词,关键词=获取地址参数().search)
        this.关键词 = 关键词
    }
    获取卡片列表=async()=>{
        return (await 核心api.fullTextSearchBlock(
            {query:this.关键词}
        )).blocks
    }
    获取卡片内容数据=async(卡片id)=>{
        let data = await 核心api.getDoc(
            {
                id:卡片id,
                mode:0,
                size:102400
            }
        )
    //使用lute来生成html
        return {html:lute.Md2HTML(lute.BlockDOM2StdMd(data.content))}
    }
}
```

这就是一个搜索适配器，作用是把思源的关键词搜索结果显示成一个白板视图：

这样就显示了搜索思源笔记的结果。

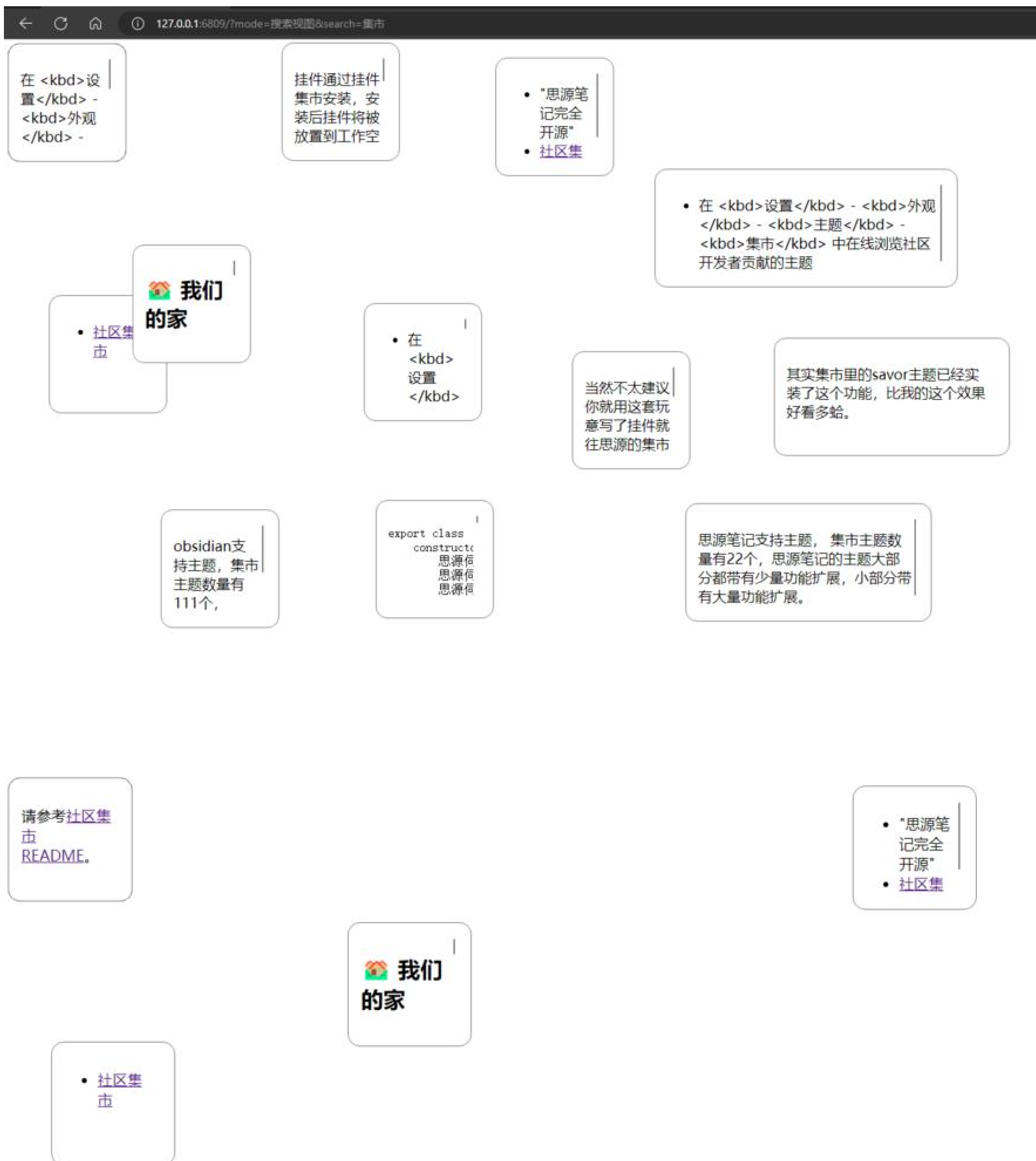
因为几何信息是存储在块上的，所以就算搜索结果有变化，已经在白板上放好位置的块也不会乱跑啦。

不过这里有个小问题，在不同的关键词搜索结果里，同一个块会停留在同一个位置，所以我们来重载下它的读写数据方法：

```
  获取卡片几何数据= async (卡片id) => {
     let 原始数据 = await 核心api.sql(
        {
           stmt: `select * from attributes where  name = 'custom-whiteBoardData-${白板id}' an
block_id='${卡片id}'`
        }
     )
     if (原始数据 && 原始数据[0]) {
        let json数据 = JSON.parse(原始数据[0].value)
        let 关键词匹配结果= json数据.find(
           item => { return item.mode === '搜索视图'&&item.search===this.关键词 }
        )
        if(关键词匹配结果){
           return 关键词匹配结果
        }
        else {
           return json数据.find(
              item =>{return item.mode==='搜索视图'}
           )
        }
     }
  }
  保存卡片几何数据=async (卡片id,显示数据)=>{
     let 原始数据 = await 核心api.sql(
        {
```

```
            stmt: `select * from attributes where  name = 'custom-whiteBoardData-${白板id}' an
block_id='${卡片id}'`
        }
    )
    let json数据
    if (原始数据 && 原始数据[0]) {
        json数据 = JSON.parse(原始数据[0].value)
        let 旧数据= json数据.find(
            item => { return  item.mode === '搜索视图'&&item.search===this.关键词 }
        )
        if(旧数据){
            旧数据.data = 显示数据
        }
        else json数据.push({
            mode:'搜索视图',
            search:this.关键词,
            data:显示数据
        })
    }
    else json数据=[{mode:'搜索视图',search:this.关键词,data:显示数据}]
    let obj = {}
    obj.id =卡片id
    obj.attrs ={}
    obj.attrs[`custom-whiteBoardData-${白板id}`]=JSON.stringify(json数据)
    await 核心api.setBlockAttrs(obj)
}
```

好了，这样就把每个关键词的搜索结果分开了。



可以看到这回每个搜素关键词白板里各个块的位置就不再一样了。

3、超链接适配器：

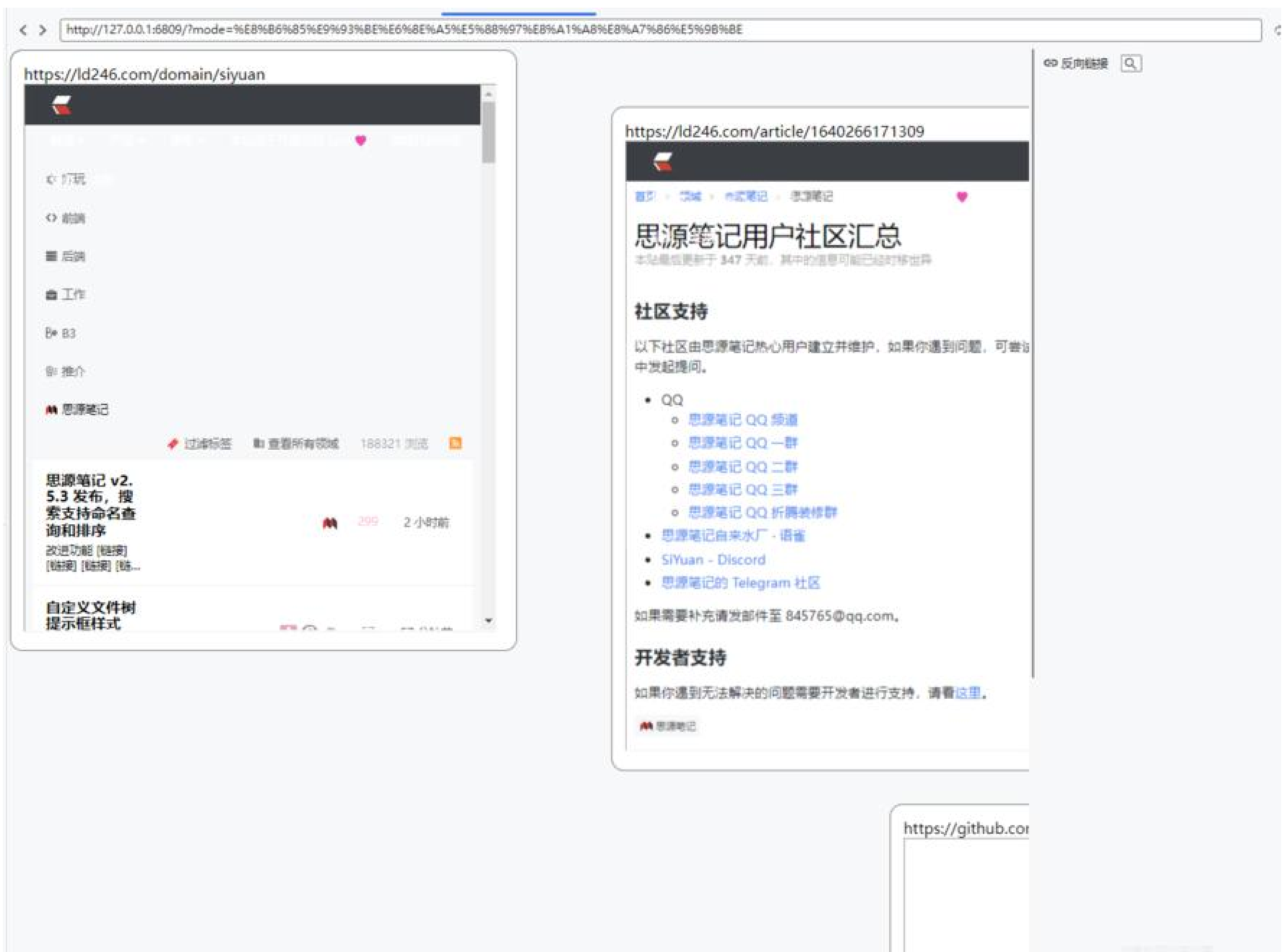
来，我们来搞一个新的，显示一个文档里面所有的超链接，基于sql适配器改一下获取数据的结果就以了。

首先是获取搜索结果，：

```
import { 基础sql适配器 } from "./baseSQL"
import { 获取地址参数 } from ".."
let id = 获取地址参数().id||'20200812220555-lj3enxa'
let sql = `select * from spans where root_id ='${id}' and type like '%textmark%a%'`
export  class 超链接适配器 extends 基础sql适配器{
        constructor(){
          super(sql)
        }
        获取卡片内容数据=async (卡片id) => {
           let 卡片数据 = this.卡片列表.find(
              item=>{return item.id === 卡片id}
           )
           let 链接 = 卡片数据.markdown.replace(`[${卡片数据.content}]`,'')
           console.log(链接)

           链接 = 链接.substring(1,链接.length-1)
           return {html:`<iframe src='${链接}' style="width:100%;height:calc(100% - 10px);"></iframe>`}
        }
}
```

啊，也就是把文档里面所有的超链接显示成iframe展示到白板上，这样好像比较适合做啥网址收集什的?



原文链接：思源笔记折腾记录 - 做一个白板 - 更多的视图

上面那个浏览器页签一样的也是在思源里打开的，它的实现参考这里：

思源笔记折腾记录 - 思源内部打开网页 - 链滴 (ld246.com)

不过其实这个适配器还没有完成，因为它的修改还没有保存的方式，不过可以自己思考一下，它能够哪里存，想出来的可以在评论里说一下，不止有一种方法的。

好了，其实还有一个地方没有说，适配器需要显式的引入和声明：

whiteBoard\src\data\adapter.js

```
import { 获取地址参数, 核心api } from "./index.js";
import 子文档视图适配器 from './adapters/childDoc.js'
import { 搜索适配器 } from "./adapters/search.js";
import { 超链接适配器 } from "./adapters/hyperLinks.js";
let {  id,mode,search } = 获取地址参数()
if (!mode) {
    mode = "子文档视图"
}
if(!id){
    id = '20200812220555-lj3enxa'
}



let 适配器列表 = {
    子文档视图:子文档视图适配器,
    搜索视图:new 搜索适配器(search),
    超链接列表视图:new 超链接适配器()
}
let 当前数据适配器 = 适配器列表[mode]
export default 当前数据适配器
```

就像上面那样。

还可以实现更多的适配器来显示更多的白板布局，就是提供一下数据获取和保存的方法而已，不过就多说这个了。

代码片段的仓库在这里

leolee9086/snippets (github.com)

viteWdigets的仓库在这里

leolee9086/viteWidgets (github.com)