【原子权限梳理】

简述

```
; ;
; "";
```

什么是原子权限?

原子(Atom)就是不能再分解的最小颗粒(粒度);
原子权限(Atomic Authority): 最底层，最基本的权限;
所谓的权限，是对文件/文件夹的可操作性的描述；原子权限之上还有有"外链权限","密级管控";
我们对一个文件的可操作性是由各级权限筛选组合之后的一个结果;
例如:
  场景：在企业空间中，用户A创建文件a并将预览权限授权给用户B，同时生成一条外链,关闭预览权限，将外链分享给用户B;
  结果：用户B在企业空间中可以预览文件a，在外链中无法预览文件a;
  分析：在这两个场景下，用户B对文件a的原子权限不同，导致对文件a的可操作性不同

12 种原子权限:可预览，可下载，可上传，可新建，可创建外链(外链可上传 + 外链可下载)，可删除，可重命名，可复制，可移动，可评论，可见列表

如何区分组合原子权限?

经需求文档、接口文档和代码分析,得出以下结论:
原子权限只有可操作和不可操作两种，可以用二进制来表示，0代表false，1代表true;
12中原子权限就用12位二进制的数表示，每一位表示一个原子权限，每一位的数值对应是否可操作;

例如:

可预览的二进制数表示为：001，可下载的二进制数表示为：010，可上传的二进制数表示为：100
我们可以对其自由组合，
同时满足：可预览,可下载,可上传的数值表示为 001 + 010 + 100 = 111
同时满足：可预览,可下载,可上传的数值表示为 001 + 100 = 101
考虑到原子权限组合时可相互覆盖，此处不应该是 + 运算；应修改为"按位或运算符 |"
同时满足：可预览,可下载,可上传的数值表示为 001 | 010 | 100 = 111
同时满足：可预览,可上传的数值表示为 001 | 100 = 101
如果权限叠加(原子 + 继承关系) 101 | 100 = 101

反之，我们看到组合为010时,可以推断出这个组合仅有下载权限(010)
以上就是权限掩码和权限组合

补充：权限的常见操作

1. 权限的验证：

(basic_auth_value& atom_auth_value) =atom_auth_value

按位与运算符 & ：对每一位执行与(AND)操作，对应位置均为1返回1，否则返回0
```
  A = 1 0 1 0
  B = 1 1 1 1
A&B = 1 0 1 0
```

2. 权限的分配

basic_auth_value = basic_auth_value| atom_auth_value

按位或运算符 | ：对每一位执行或（OR）操作，对应位置均为0返回0，否则返回1

```
  A = 1 0 1 0
  B = 1 1 1 1
A&B = 1 1 1 1
```

3. 权限的删除(求补、与运算)

basic_auth_value=basic_auth_value& (~atom_auth_value)

> *tips:basic_auth_value➡ 用户(基本)权限atom_auth_value➡原子权限*

补：如何判断权限？

通过设置掩码 0b1
能够判断一个用户的是否具备某个权限。
某个原子权限相对预览权限的左移位数为 n(n≥1)
使用用户权限与左移n位的掩码进行按位与运算，得到的结果真值，即为是否具备某个原子权限。

例如：

用户对文件夹的基本权限为:可下载，对应的掩码为:111000011101；我们想要判断是否具有预览权限
通过按位与运算:
```
  A = 1 1 1 0 0 0 0 1 1 1 0 1
  B = 0 0 0 0 0 0 0 0 0 0 0 1 // n = 1
A&B = 0 0 0 0 0 0 0 0 0 0 0 1 // 对应的第一位值为真，具有预览的原子权限
```

权限掩码定义/权限组合

原子权限

最基本的权限定义

ID	名称	允许掩码	允许十进制值	描述
1001	preview_op	000000000001	1	可预览
1002	upload_op	000000000010	2	可上传
1003	download_op	000000000100	4	可下载文件
1004	create_upload_delivery	000000001000	8	外链（可上传）
1005	create_download_delivery	000000010000	16	外链（可下载）
1006	create	000000100000	32	创建目录
1007	delete	000001000000	64	删除

1008	rename	000010000000	128	重命名
1009	move	000100000000	256	移动
1010	copy	001000000000	512	复制
1011	list	010000000000	1024	列出所有
1012	comment	100000000000	2048	评论

基本权限

原子权限复合形成，将外链部分的进行统一

基本权限由原子权限组合而成
对多种原子权限进行按位或操作，得到一个基本权限。

ID	名称	允许掩码	允许十进制	拒绝十进制
1	可预览	000000000001	1	0
2	可下载	000000000100	4	0
3	可上传	000000000010	2	0
4	可新建	000000100000	32	0
5	可创建外链	000000011000	24	0
6	可删除	000001000000	64	0
7	可重命名	000010000000	128	0
8	可复制	001000000000	512	0
9	可移动	000100000000	256	0
10	可评论	100000000000	2048	0
11	可见列表	010000000000	1024	0

例如：可创建外链，为原子权限“外链（可上传）和“外链（可下载）”按位或操作得到

预置组合权限

基本权限复合形成；预置内容，elbe版好像是可编辑,新增的

ID	名称	允许掩码	允许十进制	拒绝十进制
3001	预览	110000000001	3073	0
3002	上传	110000111010	3130	0
3003	下载	111000011101	3613	0
3004	上传/下载	111000111111	3647	0
3005	编辑	111111111111	4095	0
3006	禁止访问	0	0	4095
3007	可见列表	010000000000	1024	3071

权限逻辑分析

- 复合权限
- 权限逻辑：就近原则、继承原则

复合权限

原子权限继承关系，通过权限组合能够衍生角色

标准权限和原子权限的对照表

标准权限	可预览	可上传	可新建	可下载	可创建外链	可删除	可重命名	可移动	可复制	可评论	可见列表
预览	○									○	○
上传		○	○		○					○	○
下载	○			○	○				○	○	○
上传/下载	○	○	○	○	○				○	○	○
编辑	○	○	○	○	○	○	○	○	○	○	○

禁止访问	×	×	×	×	×	×	×	×	×	×	×
可见列表	×	×	×	×	×	×	×	×	×	×	○

○：允许操作 ×：禁止操作

标准权限页面功能对照

标准权限	涵盖功能
预览	预览文件、查看属性、详情、备注、查看标签、列出文件列表、列出历史版本、收藏、搜索、添加评论
上传	上传、新建、添加备注、添加标签、添加评论、外链分享、列出文件列表、列出历史版本、收藏、搜索
下载	预览文件、下载文件、打包下载、下载历史版本、外链分享、复制、添加评论、列出文件列表、收藏、搜索
上传/下载	预览文件、上传、新建、添加备注、添加标签、下载文件、打包下载、下载历史版本、外链分享、复制、添加评论、列出
编辑	预览文件、上传、新建、添加备注、添加标签、删除/还原、重命名、下载文件、打包下载、下载历史版本、外链分享、移动/复制、添加评论、列出文件列表、收藏、搜索
可见列表	列出文件列表、查看属性、详情、列出历史版本、收藏、搜索
禁止访问	无任何功能

自定义权限勾选逻辑

编号	原子权限	允许态特殊逻辑	拒绝态特殊逻辑
1	可预览	勾选允许后，11号也自动勾选允许	勾选拒绝后，2、8-10自动勾选
2	可下载	勾选允许后，1、11号也自动勾选允许	勾选拒绝后，8、9自动勾选
3	可上传	勾选允许后，11号也自动勾选允许	无
4	可新建	勾选允许后，11号也自动勾选允许	无
5	可创建外链	勾选允许后，1	-3、11号自动勾选允许无
6	可删除	勾选允许后，11号也自动勾选允许	勾选拒绝后，9号自动勾选
7	可重命名	勾选允许后，11号也自动勾选允许	无
8	可复制	勾选允许后，1、2、11号权限自动勾选	勾选拒绝后，则9号自动勾选
9	可移动	勾选允许后，1、2、6、8、11号自动勾选	无
10	可评论	勾选允许后，1、11号也自动勾选允	勾选拒绝后，则1号被勾选
11	可见列表	无	勾选拒绝后，1-10号自动勾选

- 如，勾选可预览的拒绝，可下载、可复制、可移动、可评论的拒绝会同时勾选

原子权限对应功能

编号	原子权限	对应动作	对应终端
1	可预览	预览文件	all
		查看备注	all
		查看标签	web、PC、mac
2	可下载	下载文件	all
		打包下载	web
		建立下载同步	PC、mac
		下载历史版本	all
3	可上传	上传文件	all
		上传文件夹	PC、mac
		添加备注	Web、PC、mac
		添加标签	Web、PC、mac
4	可新建	新建文件夹	all

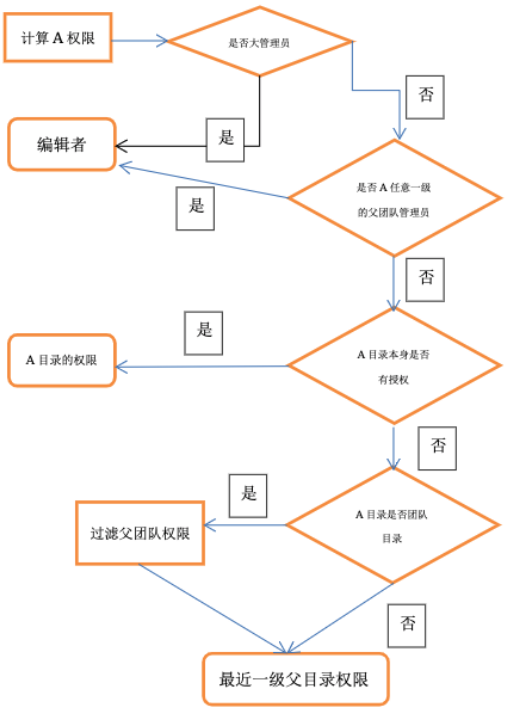
		新建文件	all
5	可创建外链	创建外链、修改外链设置	all
		删除外链	all
6	可删除删除文件/	文件夹	all
		彻底删除	web、PC、mac
		还原已删除	web、PC、mac
		设置定期清理	web、PC、mac
7	可重命名	重命名文件/文件夹	all
8	可复制	复制文件/文件夹	all
9	可移动	移动文件/文件夹	all
10	可评论	查看评论、发表（回复）评论	all
11	可见列表	列出文件列表	all
		查看属性、详情	all
		列出历史版本	all
		收藏	all
		可被搜索（仅可按文件名搜索）	all

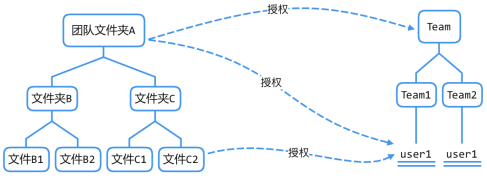
权限逻辑

就近原则、继承原则

1. 用户对该目录具有管理权限，则权限为编辑：大管理员或任意一级父团队管理员；
2. 用户在该目录下有授权，以该权限为准；
3. 如果该目录为团队下子目录，则其只受该团队本身及其下的授权影响，比如：/A、/A/B为团队，/A/B/C为目录，用户在/A的权限不影响用户/A/B/C的权限计算；
4. 在不满足1和2的条件下，用户对某目录的权限为其就近一级父目录权限，同时满足3条件，只能计算到就近一级团队目录为止

- 逻辑1. 目录就近授权：权限取最近一级目录的授
  - 授权方式：A赋予user1预览权限，C2赋予user1下载权限；
  - 授权结果：user1在C2的权限为：下载权限；
- 逻辑2. 目录继承权限
  - 授权方式：在A目录授权给user1预览权限
  - 授权结果：user1在C目录权限为预览权限
- 逻辑3：团队用户就近授权：user>团队>父团队（如继承）>所有者，父子团队默认不继承
  - 授权方式：在A目录授权给Team1（团队不继承）预览权限
  - 授权结果：user1在A目录没有权限
  - 授权方式：在A目录授权给Team1（团队继承）预览权限
  - 授权结果：user1在A目录预览权限
  - 授权方式：在A目录授权给Team1（团队继承）预览权限，授权给Team2下载权限
  - 授权结果：user1在A目录权限为下载权限
  - 授权方式：在A目录授权给Team1（团队继承）预览权限，授权给Team2下载权限，授权给user1编辑权限
  - 授权结果：user1在A目录权限为编辑权限
- 逻辑4：同一用户属于不同团队，两个团队同级目录被授权，权限合并
  - 授权方式：在A目录授权Team1下载外链者权限，授权给Team2上传者权限
  - 授权结果：user1在A目录有上传者、下载外链者2个权限的所有操作能力，权限取并集
- 逻辑5：外链中所有的目录必须分享者有外链权限才展示
  - 授权方式：
    - 在A目录授权user1下载外链者权限，在C1目录授权user1预览权限；
    - user1在A目录创建外链，勾选允许预览+允许下载
  - 授权结果：
    - 第三者打开外链后，A目录下的文件可以下载预览。
    - 看不到C1目录，因为user1对C1目录没有外链权限。
    - 看不到B1目录，因为B1目录为团队文件夹，不继承父目录权限





授权流程分析

原子权限的定义

* 对文件/文件夹原子权限的分析，均在密级允许的前提下进行
* 不考虑外链的场景
* 操作流程：点击"分享"按钮 - 选择分享方式 - 选择用户/组 - 设置原子权限(企业空间的文件夹允许继承)

"分享"按钮的展示

以企业空间为例，企业空间:/list/ent，个人空间:/list/self

右键菜单选择

代码逻辑梳理

右键 ➞ 菜单显示的流程：

1. mixin/space : setPosition() 设置"右键菜单"的定位和显示. 在组件<file-list>中透传
2. file-item.vue: setPos() 调用"右键菜单"
3. mixin/context-menu: "右键菜单"的内容列表
4. util/operations-filter.js previewVisible()判断显示隐藏
5. static/js/util/util.js getCSSActionByCode方法，通过权限掩码转换出对应的权限
6. 根据接口`/v2/metadata_page/databox`返回的`access_mode`字段,展示对应掩码的按钮

通过权限掩码转换出对应的权限

```
export function getCSSActionByCode(code) {
  let rs = ''
  const returnData = {}
  Object.keys(G_CUSTOM_AUTH_CODE).forEach((o) => {
    if ((code & G_CUSTOM_AUTH_CODE[o]) ===
G_CUSTOM_AUTH_CODE[o]) {
      if (rs !== '') {
        rs += `:${o}`
      } else {
        rs = o
      }
    }
  })
  returnData.authCSS = rs
  Object.keys(G_STANDARD_AUTH_CODE).forEach((i) =>
{
    if (code === G_STANDARD_AUTH_CODE[i]) {
      returnData.authType = i
    }
    if (!returnData.authType) {
      returnData.authType = 'custom'
    }
  })
  return returnData
}

:
G_CUSTOM_AUTH_CODE
G_STANDARD_AUTH_CODE
G_STANDARD_AUTH_CODE_FILE  ()
...
```

思考、优化建议

1. 从交互上来看,@contextmenu.prevent 仅控制主视图区的右键事件，没有覆盖到所有区域，是否需要覆盖 ???
2. 掩码权限交由本地处理有一定的问题，后续迭代的成本会增加，类似的权限内容建议完全由接口返回

菜单按钮属性的判断

```
// static/js/mixin/context-menu.js
menuData: {
  contextMenu: {
    folder: [
        // ...
        // ""
        {
      vm: this,
          //
      name: Util.isClient() ? '' : '',
      // type
      ref: 'download',
          //
      get show() {
        const { vm } = this
        const cur = vm.selectedData
        if (vm.selectedData === null) return false
// false
        const { access_mode: accessMode, path } =
cur // access_mode
        const routerName = window.Router.
currentRoute.name
        if (routerName === 'favorite' && vm.
selectedData.path_type === 'share_in' && /^\/([^/]
+)$/.test(path)) {
                // ,,
          return false
        }
                 // ifHasAuth, ""
        return Util.ifHasAuth(accessMode,
'download')
      },
      operateType: 3, // ,
    }
        // ...
        ]
  }
}
```

判断是否有某种权限

```
export const ifHasAuth = (accessMode, authname) =>
{
  const action = getCSSActionByCode(accessMode).
authCSS
  return action.split(':').includes(authname)
}
```

hover列表

代码逻辑梳理

获取➡渲染逻辑:

1. 'mixin/space.js'中, 请求接口`/metadata_page/databox`, 对获取到的数据进行分页缓存等操作
2. 引入到组件`components/page/ent.vue`中
3. 父子组件关系: ent ➡ file-list ➡ file-item ➡ file-item-operation
4. 在ent中,使用slot传参,展示操作按钮组件
5. 通过变量access_mode来进行筛选展示按钮

分析:

思考、优化建议

- 需要熟悉 slot 相关的内容
- 需要mixin 本身作为一种优化策略来使用, 但是当体积庞大时阅读起来也比较麻烦

目前`static/js/mixin/space.js`中的变量`switchMark`, 好像没有实际的意义,必须为 true 否则无法展示列表; 对于这种无意义的变量可以优化;

由代码分析,不管怎么变化,switchMark均为 true, 无其他情况, 这个变量无意义

`components/page/ent.vue`是最外层的组件，可以视为父组件，引用的最外层就是这里
1.通过`import FileList from '../common/file-list/index.vue'`的方式引入组件'<file-list>'

分享按钮的展示逻辑

```
// components/page/ent.vue
// import ENT_OPERATE from 'js/mixin/file-
operation'
<template
  slot="operation"
  slot-scope="{data:{ index, currentData,
haveSecureDelivery, haveFavorite, operationState,
otherData}}"
  // , `components/common/file-list/file-item.vue`
>
  <file-item-operation
    v-if="operationState"
    ...
  />
</template>
```

2.组件`<file-list>`的显示依赖变量 switchMark && ! showSearchHistory

components/common/file-list/index.vue

```
<component
  :is="fileItemName" // FileItem
  v-for="(currentData, index) in fileList.content"
  :current-data="currentData"
  ...
></component>
```

变量switchMark，由static/js/mixin/space.js定义(固定的值，恒为true)；
变量showSearchHistory(展示搜索历史)，在`store/modules/m-get-searchrouter.js`中控制，搜索时为true；
3.操作按钮的显示依赖`components/common/file-list/file-item.vue`中的`operationState`
operationState 为true时展示操作按钮；
4.子组件中状态以及内容的展示都依赖于 currentData 变量

在这个动态组件中,引用了`components/common/file-list/file-item.vue`;
* 在父组件中，将 currentData 等值传入子组件，然后利用 slot 传参，在父组件中调用;
currentData 是由 fileList.content 控制，推断 fileList 是当前页面展示的列表内容;
向前查找，在`components/page/ent.vue`中，绑定了 `fileList` 并传入;
fileList 是 mixin 中的变量. 请求接口 `/metadata_page/databox` 获取到;

父组件 `components/page/ent.vue` 利用了插槽的传参功能，将子组件的参数逐传到父组件中，在父组件中进行判断;
插槽中的判断条件 v-if="operationState" 在子组件 `components/common/file-list/file-item.vue` 中进行查看

变量 switchMark

```
data() {
  return {
    // ...
    switchMark: true
    // ...
  }
}
// mothods
changeListMode() { //
  // ...
  if (this.isListMode) {
    // safariBUG
    this.switchMark = false
    // ...
    this.$nextTick(() => {
      this.switchMark = true
    })
  } else {
    // ...
  }
}
```

- 计算属性执行频率过高；每次hover 的时候，会执行 20 次，整个列表都执行了一遍

变量 switchMark

```
// operationState,
operationState() {
  console.log(111)
  if (!this.isListMode) return this.isShowOperation
  return (this.selectedData.length > 1)
    ? !this.currentData.checked && this.
isShowOperation
    : this.isShowOperation
}
```

- 无意义的代码，需要删除

无实际意义的函数

```
// static/js/mixin/space.js
changeData(metaData = {}) {
  // ...  ...
  return metaData
}
' ' '
```

插槽属性透传

```
//
<slot
  v-if="list.type === 'file' && !list.showRename"
  :data="{currentData,
          index,
          haveFavorite,
          haveSecureDelivery,
          operationState,
          otherData: $data}"
  name="operation"
/>
```

按钮 hover 时的渲染逻辑

file-item-operation 组件的渲染

```
// components/common/file-list/file-item-operation.
vue
// operationstatic/js/mixin/file-operation.js
<template v-for="(operateIcon, index) in operation
[type]">
  <span
    v-if="operateIcon.show"
    :key="index"
    :ref="operateIcon.ref"
  >
        //  mixin/file-operation
    {{ operateIcon.ref == 'link' ? $t('') : '' }}
  </span>
</template>

type() { //  file folder ...
  if (this.$route.name === 'recent') {
    if (this.currentData.fileType === 4) return
'imglist'
    if (this.currentData.fileType === 7) return
'delivery'
  }
  return this.currentData.isDir ? 'folder' : 'file'
},
```

渲染判断条件，数据处理的方法，掩码权限分析

依赖

```
operationState() {
  // computed
  if (!this.isListMode) return this.isShowOperation
  <!-- isListMode: truelist, falseicon -->
  return (this.selectedData.length > 1)
    ? !this.currentData.checked && this.
isShowOperation
    : this.isShowOperation
},
isShowOperation() {
  //
  if (this.hoverNeid && this.hoverNeid === this.
currentData.neid) {
    <!--  hoverid  list_id   showOperation -->
    return this.showOperation
  }
  return false
},
```

```
showOperation false, ;
, 1,  true

fileList mixin ; `static/js/mixin/space.js`
getMeta(obj) {
  // ...
  return _$.postData('getMetadata', params)
},
getData(obj = {}, cb, add) {
  // ...
  this.getMeta(obj).then((response) => { //
    // ... id
    Promise.all(requestArr).then((rets) => {
      const metadata = res.data
      const extraMeta = rets[0] && rets[0].data
      res.data = this.changeData(metadata,
extraMeta) //
      //
      store.commit('SET_TEMP_EXTRA_META',
extraMeta)
      //  fileList ,
      this.fileList = this.decorateData(res.data,
add, obj.type)
      //
      RefreshHeader(store, { fileList: this.
fileList })
      store.commit('CANCEL_ISLOADING')
      //  ...
    }).then(() => {
      // ...
    }).catch((error) => {
      // ...
    })
  }).catch((error) => {
    // ...
  })
},

{
  // ...
  name: '',
  get show() {
    return shareVisible(currentData) //
  }
},
, `static/js/util/util.js`
if (accessMode === 4095) return 'edit'
if (accessMode === 1599) return 'upload:download:
delivery'
if (accessMode === 1575) return 'upload:download'
if (accessMode === 1565) return 'download:delivery'
if (accessMode === 1541) return 'download'
if (accessMode === 1082) return 'upload:delivery'
if (accessMode === 1058) return 'upload'
if (accessMode === 1025) return 'preview'
if ((accessMode & 1) === 1) { action.push
('preview') }
if ((accessMode & 2) === 2) { action.push
('upload0') }
if ((accessMode & 4) === 4) { action.push
('download0') }
if ((accessMode & 24) === 24) { action.push
('delivery') }
if ((accessMode & 256) === 256) { action.push
('move') }
if ((accessMode & 512) === 512) { action.push
('copy') }
if ((accessMode & 128) === 128) { action.push
('rename') }
if ((accessMode & 64) === 64) { action.push
('delete') }
```

```
if ((accessMode & 32) === 32) { action.push
('create') }
```

勾选单个文件/文件夹

代码逻辑梳理

获取➡渲染逻辑:

1. 在全局`main.vue`中引入组件`box-header`; 在`box-header`组件中引入组件`header-button-group`
2. 在组件`header-button-group`中控制了按钮的显示和隐藏(forbidden/showd等变量)
3. 父子组件关系:box-header ➡ header-button-group
4. 通过变量headerAuth对按钮信息进行循环渲染

按钮组件的引入

组件引入，通过掩码权限来判断展示的内容

全局引入顶部菜单

```
/**
 * `components/main/main.vue``box-header`
 */
import BoxHeader from './box-header.vue'
<box-header
  @showInvite="showInvite"
  @setCmd="setCmd"
/>
/**
 * `components/main/box-header.vue``header-button-
group`
 */
import HeadButtonGroup from 'components/common
/header-button-group.vue'
<HeadButtonGroup @setCmd="setCmd" />
/**
 * `components/common/header-button-group.vue`
 */
```

按钮组件的渲染逻辑

思考、优化建议

- 无效的嵌套层级过多
  * 如下面的代码,div[id="header-auth-button"]有多个父级,且父级内无其他子元素,我们也可以对父级的 DOM 和变量进行合并;
  - 从逻辑方面考虑：代码冗余，无任何意义；(疑问：需要做结构预留 ???)
  - 从性能反面考虑：减少 DOM 渲染的开销；(vDom就是为了减少DOM开销而存在的)
- 变量的职责重复
  * 如下面的代码,div[class="header-button-group"]和div[class="header-manager-group"] v-show 的条件一样;
  - 从逻辑方面考虑，因为他们是父子关系，所以根本不需要再判断一遍;
  - 从性能方面考虑，每多一个渲染条件绑定，就会多一个renderWatcher，当冗余的绑定过多，renderWatcher 积少成多也是会对性能有一定损耗的;

嵌套层级及变量问题

```
<template>
  <div
    v-show="isShowHeaderButton"
    class="header-button-group"
  >
    <div
      v-if="!forbidden"
      v-show="isShowHeaderButton"
      class="header-manager-group"
    >
      <div
        v-show="headAuthShow"
        id="header-auth-button"
        class="button-area"
      >
                // ...
      </div>
    </div>
  </div>
</template>
```

- computed计算属性的嵌套层级太深了，依赖关系太复杂，有很多重复的依赖关系，不利于代码的迭代
  例如:
  - `isShowHeaderButton` 依赖`forbidden`，但是在div[class="header-manager-group"]中v-if v-show 同时判断了这两个变量，这使他们的依赖关系无意义，判断一个即可
  - `isShowHeaderButton` 和 `headAuthShow` 都依赖于`historySearch`,和`selectedLen`，同时`isShowHeaderButton`中也依赖于`headAuthShow`，逻辑严重重复

header-button-group 组件

```
// , "",
<template>
  <div v-show="isShowHeaderButton" class="header-
button-group">
    <div v-if="!forbidden" v-show="
isShowHeaderButton" class="header-manager-group">
      <div v-show="headAuthShow" id="header-auth-
button" class="button-area">
        <div class="fileAttribute file-operate"
@click="clickHandler">
          <span
            v-for="({name, ref, className,
hasicon, show, operateType, editClass}, index) in
headerData"
            v-show="show"
            :key="index"
            :ref="ref"
            class="lv-btn default-btn"
            :data-cmd="ref"
            :data-operate-type="operateType"
            :class="[className, {'hover': hover}]"
            @mouseenter="toggleMenu(ref, true,
$event)"
            @mouseleave="toggleMenu(ref, false,
$event)"
          >
            <!-- ... -->
          </span>
        </div>
      </div>
      <!-- ... -->
    </div>
  </div>
</template>
```

渲染的逻辑

获取要渲染的按钮列表

```
// import ENT_HEAD_BUTTON from 'js/mixin/header-
button' `headerAuth`
headerData() {
  // , : [, , ...]
  if (this.headerAuth) {
    return this.headerAuth[this.headerType]
  }
  return []
},
headerAuth() {
  // `static/js/mixin/header-button.js`
  return store.state.HeaderButtonState.
headerButtonData.headerAuth
},
headerType() {
  //
},
```

判断显示隐藏的逻辑

```
isShowHeaderButton() {
  if (this.isShow) {
    return false
  }
  if (this.historySearch) {
    return false
  }
  if (!this.forbidden) {
    //
    if (this.selectedLen > 0) {
      return true
    }
    //
    if ((this.isCreateFolder && !this.
headAuthShow) || this.isCreateFolderDisable) {
      return true
    }
    //
    if ((!this.headAuthShow && this.isUpload) ||
this.isUploadDisable) {
      return true
    }
    //
  }
  return false
}
```

- 代码简化，多个 if 判断的输出结果相同时，可以简写，写好注释即可

变量 isShowHeaderButton

```
// isShowHeaderButton
isShowHeaderButton() {
  if (this.isShow) {
    return false
  }
  if (this.historySearch) {
    return false
  }
  if (!this.forbidden) {
    //
    if (this.selectedLen > 0) {
      return true
    }
        // ...
  }
  return false
}
// isShowHeaderButton
forbidden() {
  /**
   *  true: false:
   * `store/modules/m-get-headercache.js`
   */
  return store.state.HeaderButtonState.forbidden
},
selectedLen() {
  /**
   * , :
   * `store/modules/m-get-headercache.js`
   */
  return store.getters.selectedDataLength
},
```

依赖 forbidden 的取值

```
,  forbidden  false
forbidden(state) {
  return isAdmin() && state.lenovoData.customize.
forbidden_admin_manage_doc === 'true'
  // : '/user/initInfo' customize
},
isAdmin, , `static/js/util/util.js`:
// : '/account/funs/get' userInfo
export function isAdmin() {
  if (isEntVersion()) {
    const { entRole } = store.state.lenovoData.
userInfo
    return entRole.indexOf('super_admin') !== -1
  }
  return store.getters.userInfo && store.getters.
userInfo.role === 'admin'
}

`static/js/mixin/space.js`, `SetupHeader`
SetupHeader, `store/modules/m-get-headercache.js`,
;
const HeaderButtonMutations = {
  [TYPES.CACHE_HEADER_DATA](state, data) {
    const { vm } = data
    Object.keys(data).forEach((i) => {
      //  store  forbidden
      if (data[i]) {
        state.HeaderButtonState[i] = data[i]
      }
    })
  },
}
 (super_admin)(forbidden_admin_manage_doc),
forbidden true,  false;
```

依赖 isShow/historySearch 的取值

```
isShow() {
  /**
   * , : ; , ;
   * `store/modules/m-get-headercache.js`
   */
  return true/false
},
```

变量 headAuthShow

```
headAuthShow() {
  // isShowHeaderButton,  this.$route.name ===
'recent' ;
  return (!this.historySearch && !(this.
selectedLen < 1)) || this.$route.name === 'recent'
},
```

此部分的逻辑，可以看这里:参考文档

定义用户原子权限

复制引用地址

通过地址快速访问，权限由用户的网盘账号权限决定；最后的显示和操作依然是根据原子掩码值来判断权限

新建外链

通过连接分享，可以设置上传/下载/预览功能

代码逻辑梳理

创建流程:
1. 点击"分享"按钮之后，选择"外链"
2. 在弹窗中选择对应的"原子权限"
3. 勾选对应的权限,以字符串参数的形式传给接口

表单数据都存在`this.linkShareFormData`中

相关接口:
delivery/create/databox
metadata_page/databox

生成外链的按钮

```
<input
  v-show="linkStatus === 'create'"
  id="link-create"
  type="button"
  :class="['lv-btn','primary-btn',
pwVerified?'':'create-btn-disable']"
  name="create"
  :value="$t('')"
  :disabled="showInApprovalWarn || needAuth || !
pwVerified"
  @click="createLink"
>
```

可勾选的内容

```
<span
  v-show="linkShareFormData.UPLOAD_SHOW"
  class="item-wrapper auth-row upload"
  :class="{'disabled': linkShareFormData.
UPLOAD_DISABLE || showInApprovalWarn}"
>
  // ...
  <input
    id="common-delivery-upload"
    v-model="linkShareFormData.UPLOAD"
    type="checkbox"
    name="common-delivery-upload"
    :disabled="showInApprovalWarn"
  >
  <span
    class="box-checkbox-inner"
    :class="(
      (linkShareFormData.UPLOAD && !
linkShareFormData.IS_SNAPSHOT)?
        (linkShareFormData.UPLOAD_DISABLE ||
showInApprovalWarn)?
          'checkbox-disabled':'checkbox-checked':
''
    )"
    @click="checkOrNot('UPLOAD',linkShareFormData.
UPLOAD_DISABLE)"
  />
  <label for="common-delivery-upload">{{ $t('') }}<
/label>
  // ...
</span>
```

思考、优化建议

无意义的变量赋值，如下面函数中的 res

```
return new Promise((resolve, reject) => {
  postData('createLink', deliveryCreateParams).then
((paramRes) => {
    console.log(paramRes);
    let res = paramRes
    res = {
      URL_VALUE: res.data.url,
      EOD_ID_VALUE: res.data.eod_id > -1 ? res.
data.eod.id : null,
      LINK_ID: true,
      LINK_ID_VALUE: res.data.id,
      DELIVERY_ID: true,
      PASSWORD: res.data.password || '',
      DELIVERY_ID_VALUE: res.data.code,
      REMAIN_DOWNLOAD: res.data.remain_download >
-1,
      INFO_REMAIN_DOWNLOAD_VALUE: res.data.
remain_download > -1 ? res.data.remain_download :
null,
    }
    resolve(res)
  }).catch((err) => {
    reject(err)
  })
})
```

对结构的思考:

前端仅仅是UI层，不应该处理权限相关的内容，我看到很多权限掩码在前端进行定义并使用；这部分内容可以提取出来交由后端保存，前端通过接口的形式访问；这样的可读性和可维护性更强；

多层三目运算嵌套使用，使可维护性和可读性都比较差

三目运算符

```
// , ;
<span
  class="box-checkbox-inner"
  :class="(
    (linkShareFormData.UPLOAD && !
linkShareFormData.IS_SNAPSHOT)?
      (linkShareFormData.UPLOAD_DISABLE ||
showInApprovalWarn)?
        'checkbox-disabled':'checkbox-checked': ''
  )"
  @click="checkOrNot('UPLOAD',linkShareFormData.
UPLOAD_DISABLE)"
/>
```

创建外链的函数

```
createLink(cb) {
  if (!this.validate()) {
    return false
  }
  const linkShareCreateNeedParams = {
    path: this.metaData.path,
    path_type: this.metaData.path_type,
    linkShareSubmitFormDataParams: this.
submitFormDataIntegrate(),
  }
  if (this.linkShareFormData.IS_APPROVAL) {
    this.$emit('setCmd', { name: 'reqApproval',
operateType: '1' }, {
      nsid: this.metaData.nsid, path: this.
metaData.path, type: 'delivery', linkshareparams:
linkShareCreateNeedParams,
    })
    return false
  }
  // ...
  deliveryCreateMethods(linkShareCreateNeedParams).
then((res) => {
    // ...
  }).catch((err) => {
    // ...
  })
},
```

接口参数的原子权限组合

```
// formData.mode():   r(read)    w(write)    p
(preview)
formData.mode = ''
if (this.linkShareFormData.DOWNLOAD) {
  formData.mode += 'r'
}
if (this.linkShareFormData.UPLOAD) {
  formData.mode += 'w'
}
if (this.linkShareFormData.PREVIEW) {
  formData.mode += 'p'
}
// ...
```

文件夹授权

哪些文件结构支持授权(共享)？

类型 \ 空间	企业空间	个人空间
文件	不支持	支持
目录	支持	支持
团队	支持	无

谁可以进行授权?

大管理员可以对企业空间内所有目录进行授权;
团队管理员可以对他所管理的团队及其子目录进行授权;
普通用户（包含团队管理员）可以对个人空间内目录（文件）进行共享（授权）;

代码逻辑梳理　　思考、优化建议

弹窗的展示逻辑:
1. 点击文件夹授权(header/右键/列表按钮); 触发 $emit ('getTarget'); `mixin/bus.js动态on、emit`
2. 在setCmd中处理点击按钮时携带的信息; `store/modules/m-get-cmd.js`
3. operationHandler方法判断事件类型, 授权/添加/删除等等; `mixin/space.js`
4. dialogHandler方法处理弹窗的逻辑; `mixin/space.js`
原子权限的选择:
1. 授权弹窗的右侧列表
2. 点击权限说明, 查看原子权限的组合及描述; `authorize/authPanel.vue` 前端控制的固定内容;
3. 点击右侧列表的`class="share-auth-area"`按钮, 触发 `toggleAuthList`方法, 展示下拉框`class="share-auth-list"`
4. 可以选择对应的权限/也可以选择自定义权限 (都允许选择继承)
5. 点击自定义按钮, 渲染`autoDefineDialog.vue`组件, 勾选原子权限

触发的接口:/auth/batch_create/databox; 字段allowed_mask, 是对应的十进制权限掩码

数据的渲染:
1. 弹窗引用权限组件`authorize`
{
id: 'auth',
component: 'authorize',
prop: {title: '授权管理', dialogSize: 'layout-small',
titleHover: '',},
},
* 在`authorize.vue`组件中, 调用`resolveAuthOptions`方法获取可选权限
2. 在`authorize.vue`组件中, 调用授权组件`operateMemberAuth.vue`
3. 在`operateMemberAuth.vue`组件中, 对变量`authOptions`进行循环渲染
`自定义按钮` + `允许继承`
点击`允许继承`, 子用户将继承权限
点击`自定义按钮`, 渲染`autoDefineDialog.vue`;
在'checkboxAllowChange'中,定义了勾选的交互

中间页面、动态绑定事件

`components/common/header-button-group.vue`
`clickHandler`, ``;
//
Bus.$emit('getTarget', { name, operateType, e }, this.selectedData)

//
Bus.$on('getTarget', (data, dialoginfo) => {
  if (dialoginfo) {
    this.setCmd(data, dialoginfo)
  } else {
    this.setCmd(data)
  }
})

// Bus.vue
import Vue from 'vue'
export default new Vue()

`store/modules/m-get-cmd.js`中定义了一个mutations:setCmd 获取dialog的属性信息
`static/js/mixin/space.js`将方法`setCmd`暴露
`static/js/mixin/space.js`中的operationHandler方法: 判断事件的类型, 授权/删除/添加等多种类型, 需要在这里筛选出授权弹窗
`static/js/mixin/space.js`中的dialogHandler方法: 弹窗逻辑展示

授权弹窗的条件判断

```
if (name === 'auth') {
  let tempDialogTitle = ''
  if (data) {
    if (data.path_type === 'ent') {
      tempDialogTitle = ''
    } else {
      tempDialogTitle = ''
    }
  } else {
    tempDialogTitle = this.dialogTitle
  }
  // ...
}
this.showDialog = hideDialog !== true //
```

控制列表的显示和隐藏

```
// toggleAuthList,
// ...
this.authListSwitch.splice(index, 1, isShow)
// 1.  this.authListSwitch[index];
// 2. false,true
```

解析权限列表内容

```
resolveAuthOptions(res) {
  // ...
  const cssaction = this.fileInfo.isDir
    ? ['preview', 'upload', 'download', 'upload:
download', 'edit', 'deny', 'list']
    : ['preview', 'download', 'edit', 'deny',
'list']
  const defaultOptions = cssaction.map((item) => ({
      privilege_name: this.$t(getAuthOptionsName
(item)),
      privilege_id: getPrivilegeIdByKey(item),
      allowed_mask: getAllowedMaskByKey(item),
      denied_mask: getDeniedMaskByKey(item),
    }))
    console.log(1, this.authOptions)
    this.authOptions = [...autoDefineOptions, ...
defaultOptions]


  { "privilege_name": "","privilege_id": 3001,"
allowed_mask": 3073,"denied_mask": 0 }
  {...}
  {...}
  {...}
},
```

审批设置

对"文件/文件夹"添加审批权限，就是对原子权限添加一些额外的管控；

管理员进行文件夹授权

对"文件/文件夹"添加审批权限，就是对原子权限添加一些额外的管控；

代码逻辑梳理　　思考、优化建议

流程：管理控制台-选择团队-已授出文件夹-添加文件夹授权(选择文件夹/选择用户/选择权限)

操作一(新增授权):
1. 点击左侧列表中的"用户&团队管理"
2. 选择某个团队, 点击上面tab按钮"已授出文件"
3. 点击按钮"添加文件夹授权"
4. 先选择要分享的文件夹, 点击添加
5. 选择用户/用户组, 选择赋予用户/用户组的权限 [和文件夹授权一样, 可自定义可继承]
6. 点击确定完成授权 → 请求接口: "auth/batch_create/databox" 进行授权

操作二(修改授权):
1. 点击左侧列表中的"用户&团队管理"
2. 选择某个团队, 点击上面tab按钮"已授出文件"
3. 请求接口: /auth/list_subset_resource?team_id={xxx} 展示
4. 点击列表中的操作按钮"设置"
5. 选择用户/用户组, 选择赋予用户/用户组的权限 [和文件夹授权一样, 可自定义可继承]
6. 点击确定完成授权 → 请求接口: "/auth/batch_create/databox" 进行授权

展示左侧团队列表

```
// web/www/js/lenovodata/SDK/TeamManager.js
/**
  *
  @param {object} params
  @param {function} params.callback
  */
exports.getTeamListByUser = function(params){
  var uri = URL_PREFIX + '/list_by_user/';
  Util.ajax_json_get_asyncFalse(uri, function(xhr,
textStatus){
    var retVal = Util.
ajax_json_process_normal_result(xhr, textStatus);
    if(params.callback){
      params.callback(retVal);
    }
  });
};
```

展示已授权的文件夹

```
// web/www/js/lenovodata/SDK/AuthManager.js
/**
 * id
 @param {object} params
  @param {function} params.callback -
  @param {string} params.team_id - Id
  */
exports.list_subset_resource = function (params) {
  var uri = URL_PREFIX + "/list_subset_resource?
team_id=" + params.team_id;
  if(window.nsid) {
    uri += '&nsid='+window.nsid
  }
  Util.ajax_json_get(uri, function (xhr,
textStatus) {
    var retVal = Util.
ajax_json_process_normal_result(xhr, textStatus);
    if (params.callback) {
      params.callback(retVal);
    }
  });
};
```

渲染权限选项的列表流程

1. 定义文件夹权限

```
/**
 *
 */
var FOLDER_AUTHITEM = {
    'preview':{name:_(""),privilege_id:3001,
allowed_mask:3073,denied_mask:0},
    'upload':{name:_(""),privilege_id:3002,
allowed_mask:3130,denied_mask:0},
    'download':{name:_(""),privilege_id:3003,
allowed_mask:3613,denied_mask:0},
    'upload:download':{name:_("/"),privilege_id:
3004,allowed_mask:3647,denied_mask:0},
    'edit' :{name:_(""),privilege_id:3005,
allowed_mask:4095,denied_mask:0},
    'deny':{name:_(""),privilege_id:3006,
allowed_mask:0,denied_mask:4095},
    'list':{name:_(""),privilege_id:3007,
allowed_mask:1024,denied_mask:3071}
};
```

将文件夹权限赋值到 AUTHITEM 上

```
function DialogListCombox(node,options){
  this.node = $(node);
  // jq
  this.option = $.extend({isTeam:false,
initval:'preview'},options);
  //
  if(options.is_share){
    AUTHITEM = options.isfolder?FOLDER_AUTHITEM:
FOLDER_SHARE;
  }else{
    AUTHITEM = FOLDER_AUTHITEM;
  }
  this.render();
}
```

遍历 AUTHITEM 渲染权限选项(ul-li)

```
for(var i in AUTHITEM){
  var item = "<li class='dialog_combox-item'
privilege_id={{privilege_id}} allowed_mask=
{{allowed_mask}} denied_mask={{denied_mask}}>
{{name}}</li>";
  item = $(Mustache.render(item,AUTHITEM[i]));
  datalist.append(item);
}
var authType = "";
//
if(Util.getAuthTypeByAuthID(this.option.initval)){
  //
  authType = Util.getCSSActionByCode(Util.
getAuthCodeByAuthID(this.option.initval)).authType;
}else {
  authType = 'custom';
}
//
var selectAuth = Util.getAuthName(authType);
self.node.find(".dialog_combox-text").html(_
(selectAuth));
datalist.append(Mustache.render('<li class="
dialog_combox-item-advanced-setting"><a"/>'+_("")
+'</a></li>',{inherit:this.option.inherit}));
```

选择自定义权限组合时

渲染自定义高级设置弹窗

```
// web/www/js/lenovodata/component/dialog
/dialog_listCustom.js
DialogHigher.prototype = {
  init : function(){
    //  +
  },
  render : function(dialog_higher_box){
    var self =
this;
    //object
    var custom_object = self.custom;
    //
    var higher_radio_title = {
      "preview" : _(""),
      "download" : _(""),
      "upload" : _(""),
      "create" : _(""),
      "delivery" : _(""),
      "delete" : _(""),
      "rename" : _(""),
      "copy" : _(""),
      "move" : _(""),
      "comment" : _(""),
      "list" : _("")
    }
    //
    var li_box = new Array();

    //
    for( o in custom_object){
      var allow = false;
      var deny =
false;
      if(custom_object[o] == "YES"){
        allow = true;
        deny = false;
      }else if(custom_object[o] == "NO"){
        allow = false;
        deny = true;
      }else{
        allow = false;
        deny = false;
      }
      var li = $(Mustache.render(self.default_.
template, {'allow':allow,'deny':deny,'title':
higher_radio_title[o],'name': o}));
      li_box.push
(li);
    }
    // ...
    //
  },
  events : function(){
    //
  },
  power : function(this_radio_name,
this_radio_value){
    //
  }
}
```

"掩码拒绝"的使用

根据`标准权限和原子权限的对照表`和代码分析，掩码的允许和拒绝计算的方式都是相同的；（由预置权限中的"禁止"掩码|计算得出）

目前前端判断的条件均是使用 掩码的允许十进制（通常是access_mode/allowed_mask这两个字段）. 未使用到掩码的拒绝十进制；

在我们的代码中，使用到拒绝码的场景有：1. 判断是否为基本权限 2. 作为接口中的参数 ；公有项目，elbe项目均未使用；私有工程中页面的展示和组件渲染过程中未使用到拒绝掩码，不需要太深入了解；

补充：我觉得原子掩码的禁止无实际的意义；如果使用禁止掩码，我们的判断逻辑就是：允许/不允许/拒绝，这显然不符合逻辑，我们仅需要判断：允许/不允许 即可

思考体会

从代码和文档分析，原子权限的设计是比较巧妙的，通过二进制来表示，通过十进制来传递解析；唯一感觉到不足的地方是，太乱了；

我们应当遵循一个原则，如非必要，勿增实体；

既然所有的权限都是统一的，我们就可以将所有的原子权限集中定义，管理，使用，这样的逻辑更加清晰，目前的情况是工程下面有多重定义，梳理的时候很容易乱；

需要做的工作：写一个权限中心，将原子权限统一管理输出（参考axios对接口的管控，找一个模型对原子权限进行重构）；

鉴权方法参考

```
/*
 * : ES6 Set
 * @accessOpArray :  => Set(6) {'preview', 'download', 'delivery', 'copy', 'list', ...}
 * @ATOM_AUTH_CODE  "-"
 * @accessMode ; e.g. 4095 3613
 * &:
 */
function accessOpArray(accessMode) {
  const userAtomAuth = new Set()
  Object.keys(ATOM_AUTH_CODE).forEach((key) => {
    if (accessMode & ATOM_AUTH_CODE[key]) {
      userAtomAuth.add(key)
    }
  })
  return userAtomAuth
}

console.log(accessOpArray(3613).has('download'))


/*
 * : function-curry
 */
function authCodeWrapper(type) {
  return function (accessMode) {
    const userAtomAuth = new Set()
    Object.keys(ATOM_AUTH_CODE).forEach((key) => {
      if (accessMode & ATOM_AUTH_CODE[key]) {
        userAtomAuth.add(key)
      }
    })
    return userAtomAuth.has(type)
  }
}

// utils,
const hasDownloadAuth = authCodeWrapper('download')
const hasUploadAuth = authCodeWrapper('upload')
// ...

// , boolean
hasDownloadAuth(3613) // => true
hasUploadAuth(3613) // => false
```

参考：

https://zhuanlan.zhihu.com/p/352025616
https://www.cnblogs.com/jasonlny/p/3526671.html