系统字典设计-含前后端

这段时间设计了系统字典功能, 特定写一遍文字记录一下.

后端

表结构设计

> 先贴上 sql 可直接执行

```
create table sys_dictionary
(
    id           int auto_increment
        primary key,
    name         varchar(64)                          not null,
    dict_type    int                                  not null comment '字典类型',
    sort         int          default 0               not null comment '排序',
    default_flag tinyint(1)   default 0               not null comment '默认',
    remark       varchar(128) default ''              not null comment '备注',
    meta         varchar(128) default ''              not null comment '元数据
',
    del          tinyint(1)   default 0               not null comment '删除标记',
    create_time  timestamp    default CURRENT_TIMESTAMP not null
comment '创建时间',
    update_time  timestamp    default CURRENT_TIMESTAMP not null on
update CURRENT_TIMESTAMP comment '更新时间'
)
    comment '系统字典表' engine = InnoDB;
```

笔者对上面字段进行说明

1. 字段类型主要是为了将字段进行归类
2. 排序为了字段类型在页面上显示顺序

3. 备注主要对这个字段值进行解释
4. 元数据这个字段提供给前端显示使用, 比如字体颜色/字体/图标 登录
5. 删除标记主要解决因为这里删除是为了用户不能再使用这个字典生成新的数据,之前的使用这个字典数据不受影响

> 这里你会发现大多数字典表 value 这里并没有出现, 这个是因为在实际中并没有很大的作用. 为了设计而设计一个 value 后面只会使程序变得不易维护所以我们这里字典值采用 id 作为 value 值

dictType 为了后端方便可以建立一个 enum 类进行维护.

接口设计

字典列表接口

`/dict/list`

参数	是否必填	描述
dictType	否	字典类型, 不传返回全部

其他修改删除接口都多比较简单, 这里就不举例了

前端
--

前端这里使用的 vue3 技术, 下面主要是字典组件的封装

pinia 状态管理字典

```
export const useDictionaryStore = defineStore('widgetDictionaryStore', () => {
```

```
  const allDistTree = ref<Record<number,
DictionaryItemResponse[]>>({});

  /**
   * 处理字典返回值
   * @param dictList
   */
  function handleRes(dictList: DictionaryItemResponse[]) {
    return dictList.map(item => {
      if (item.meta) {
        // @ts-ignore
        item.meta = JSON.parse(item.meta);
      }
      return item;
    });
  }

  /**
   * 加载字典
   * @param dictType
   */
  async function loadDistData(dictType: null | WidgetDictType = null) {
    const data =  await getAllDictionary({ dictType })
    if (dictType) {
      // 局部更新
      allDistTree.value[dictType] = handleRes(data);
    } else {
      // 全局更新
      allDistTree.value = groupBy(handleRes(data), (item) =>
item.dictType);
    }
  }

  function getDictItemInner(dictType: WidgetDictType, value: number) {
    const searchDictItems =  allDistTree.value[dictType] || [];
    const item = searchDictItems.find(item => item.id === value)
    if (item) {
      return item;
    }
  }

  /**
   * 按字典类型获取字典列表
   * @param dictType
   */
  async function getDistList(dictType: WidgetDictType) {
    const dictItem = allDistTree.value[dictType];
    if (dictItem) {
```

```
      return dictItem;
    }
    await loadDistData(dictType);
    return allDistTree.value[dictType];
  }

  /**
   * 获取字典项
   * @param dictType
   * @param value
   */
  async function getDictItem(dictType: WidgetDictType, value: number) {
    const dictItem = getDictItemInner(dictType, value)
    if (!dictItem) {
      return dictItem;
    }
    // 重新加载
    await loadDistData(dictType)
    return getDictItemInner(dictType, value);
  }

  return {
    getDictItem,
    loadDistData,
    getDistList,
    allDistTree
  }
});

```

问:上面 用 Record<number, DictionaryItemResponse[]> 方式来管理字典是为什么?

答: 是因为字典的使用方式 99% 多少通过类型获取字典列表的方式使用, 所以使用这种存储结构会使按字典类型获取字典列表检索的复杂度降低到 0(1) .

字典项显示组件

字典显示组件，是字典组件最基础的组件之一

```
<script lang="ts" setup>
```

```
import type { DictionaryNameProps } from
'@/components/Distionary/index';
import { ref, watch } from 'vue'
import { useWidgetDictionaryStore } from
'@/stores/WidgetDictionaryStore'


const props = defineProps();
const dict = ref();

const dictionaryStore = useDictionaryStore()
// 监控字典值变动
watch(() => props.distValue, async (newValue) => {
  dict.value = await dictionaryStore.getDictItem(props.distType,
newValue);
})

</script>
<template>
  <slot name="default" v-if="dict" :dict="dict">
    <span>{{dict.name}}</span>
  </slot>
  <slot name="noFound" v-else></slot>
</template>

```

这个是字典项基础组件主要提供给字典单项显示功能,

1. 使用者可以自定义字典渲染样式, 我们提供了 default 插槽
2. 当字典值没有找到时我们提供了noFound 插槽

属性

属性名	说明	类型	默认值
dictType	字典类型，必传	Number	-
value	字典值，必传		-

Slots 插槽

插槽名	参数	**说明**
default	dict： 字典项	字典渲染
notFound	–	当字典 value 值在字典中不存在渲染

使用

```
<DictionaryName :dict-type="..">
   <template #default="{dict}">
       <div>{{ dict.name }}</div>
   </template>
</DictionaryName>
```

字典列表组件

提供字典列表显示, 是字典组件最基础的组件之一。

```
<script lang="js" setup>
import { ref, onMounted } from 'vue'
import { useDictionaryStore } from "@/stores";
const props = defineProps({
  dictType: {
    type: Number,
  },
  setDictList: {
    type: Function,
    default: (val) => {},
  }
})
const dictList = ref([])
const dictionaryStore = useDictionaryStore()
function addDictItem(data) {
  dictList.value.push(data);
  props.setDictList(dictList.value)
}
async function loadDictList() {
```

```
  dictList.value = await dictionaryStore.getDistList(props.dictType, true)
  props.setDictList(dictList.value);
}

defineExpose({
  loadDictList,
  addDictItem,
})

onMounted(async () => {
  await loadDictList();
})
</script>

<template>
  <slot v-for="(dict, index) in dictList"
        :key="dict.value"
        :dict="dict"
        :index="index">
    <span>{{dict.name}}</span>
  </slot>
</template>
```

属性

属性名	说明	类型	默认值
dictType	字典类型，必传	Number	-
setDictList	组件会调用这个方法设置数据, 外部需要这个字典列表可以使用这个属性	funcation(val: DictList) => void	-

Slots 插槽

插槽名	参数	**说明**
default	dict，index，key	列表项渲染插槽

Exposes

名称	**说明**	**类型**
loadDictList	重新加载当前字典列表	() => void
addDictItem	向当前字典添加当前项	(dict: DictItem) => void

使用

```
<template>
  <DictionaryName
      :dict-type=""
      :value="">
    <template #default="{dict}">
      <div>
        <el-tag>{{dict.name}}</el-tag>
      </div>
    </template>
  </DictionaryName>
</template>
```

字典下拉列表组件

因为字典选择大多数会出现在 select 中, 我们这里选用 el-select 利用上面提供的组件封装一下字典下拉列表组件, 这个组件支持单选多选自动创建字典

```
<script lang="js" setup>
import { watchEffect, ref, computed, watch, onMounted } from 'vue';
import DictionaryListDisplay from '../DictionaryListDisplay/index.vue'
import { SaveDictionary } from "@/api/dictionaryModuleApi";
const props = defineProps({
  className: {
    type: Array,
    default: ''
  },
  style: {
```

```
    type: Object,
    default: () => ({})
  },
  dictType: {
    type: Number,
  },
  selectProps: {
    type: Object,
    default: () => ({})
  },
  disableAutoCreate: {
    type: Boolean,
    default: false
  },
  modelValue: {
    default: () => {}
  },
  createDict: {
    type: Function,
    default: null
  }
});


const emit = defineEmits([
  'update:modelValue',
   'change',
   'focus',
   'blur'
])

const multipleEnvSingle = computed(() => props.selectProps.multiple &&
!(props.modelValue instanceof Array))


function calcSelect(val) {
  return multipleEnvSingle.value ? [val].filter(item => item !== '') : val
}

const selectValue = ref(calcSelect(props.modelValue));
watch(() => props.modelValue, (newVal) => {
  selectValue.value = calcSelect(newVal);
})

const selectRef = ref();

onMounted(() => {
  console.log('selectValue', selectValue.value)
```

```
})
const allDict = ref([]);
async function handleAutoCreate(val) {
  const lastVal = val[val.length - 1]
  const index = allDict.value.findIndex(({name, value}) =>
      lastVal.toString() === value.toString() || name === lastVal.toString())
  if (index === -1) {
    const data = props.createDict
        ? props.createDict(lastVal)
        : await SaveDictionary({
            dictType: props.dictType,
            name: lastVal,
        });
    await dictionaryListDisplayRef.value.addDictItem(data);
    setTimeout(() => {
      val[val.length - 1] = data.value
      change(val)
    })
  }
  return val;
}
function change(val) {
  if (val && val.length && multipleEnvSingle.value) {
    if (val instanceof Array && val.length > 1) {
      val[0] = val.pop()
    }
    val = val[0]
  } else if (multipleEnvSingle.value) {
    val = ""
  }
  selectRef.value.blur()
  selectRef.value.focus()
  emit('change', val)
  emit('update:modelValue', val)
}
async function onChange(val) {
  if (val && val.length
      && !props.disableAutoCreate && props.selectProps.allowCreate) {
    val = await handleAutoCreate(val)
  }
  change(val)
}
async function onFocus() {
  emit('focus')
  await dictionaryListDisplayRef.value.loadDictList()
}
async function onBlur() {
  // 这里需要延迟是因为需要时间去创建这个字典
```

```
  setTimeout(() => {
    emit('blur')
  }, 200)
}
const dictionaryListDisplayRef = ref();

defineExpose({
  dictionaryListDisplayRef,
  selectRef,
})
</script>
<template>
  <el-select ref="selectRef"
             :style="style"
             :class="className"
             @focus="onFocus"
             @blur="onBlur"
             v-bind="selectProps"
             :model-value="selectValue"
             @change="onChange">
    <DictionaryListDisplay ref="dictionaryListDisplayRef"
                           :dict-type="dictType"
                           :set-dict-list="(val) => allDict = val">
      <template #default="{dict}">
        <slot name="option" :dict="dict">
          <el-option :label="dict.name" :key="dict.id" :value="dict.value"
/>
        </slot>
      </template>
    </DictionaryListDisplay>

    <!--region el-select 的插槽-->
    <template #tag>
      <slot name="tag" />
    </template>
    <template #empty>
      <slot name="empty" />
    </template>
    <template #loading>
      <slot name="loading" />
    </template>
    <!--endregion-->
  </el-select>
</template>

```
```

这里组件封装时候也继承了原来 el-select 全部的数据而且并使用 el-select 原
```

有的属性判断是否是多选, 用户无需额外传递新的参数进来. 又因为我这边有要求单选要显示 tag 样式所以这块组件单独封装 value 值为数组, 如果你不需要可以直接将值绑定进来

属性

属性名	说明	类型	默认值
className	class 样式名称	string	–
style	style 对象样式	object	–
dictType	字典类型，必传	Number	–
selectProps	el-select 全部属性	[Select Attributes](http://cxyroad.com/ "https://element-plus.org/zh-CN/component/select.html#select-attributes")	–
disableAutoCreate	是否禁止自动创建字典如果禁止将不会自动请求并创建字典，createDict 指定函数也不会执行	bool	false
modelValue v-modal	当前选择值	– array	

* string | – |
| createDict | 创建字典的函数 | funcation(name: string) => void | |

Slots 插槽

插槽名	参数	**说明**
option	dict	下拉列表项渲染插槽
tag		select 组件自定义标签内容
empty		无选项时的列表
loading		select 组件自定义 loading内容

事件

事件名	**说明**	**类型**
blur	在组件 select 失去焦点时触发	

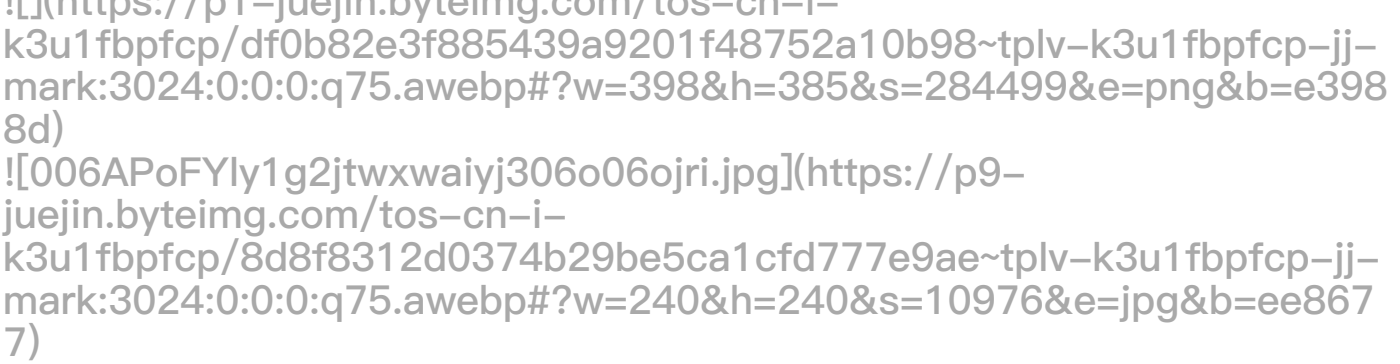
| focus | 在组件 select 获得焦点时触发 | |
| change | select 值发生变化触发 | |

Exposes

名称	**说明**	**类型**
dictionaryListDisplayRef	DictionaryListDisplay 实例	
selectRef	[el-select 实例](http://cxyroad.com/ "https://element-plus.org/zh-CN/component/select.html#select-exposes")	

最后
--

如果想一起交流请加我号: -xiaou- 加后发送 技术群, 拉你进一个无广告无推销的技术交流群.



原文链接: https://juejin.cn/post/7354150320425271306