一周编码心得分享

能很直观的看清整个文件的代码逻辑

一个文件的代码量最好控制在100行以内

```
setup() {
  const formRef = ref();
  const formData = ref<T>({}); // 表单数据对象
  const validate = (fieldName?: string)=>{...} // 校验表单
  const { renderComplex } = useComplex(formData) // 复杂一点的组件
  const { renderCustomInfo } = useCustomInfo(formData, validate) // // 加载自定义信息
  const submit = () => {...} // 提交表单

  return () => <Form ref={formRef} model={formData.value}>
    <FormItem label='简单输入框' name='name' rules={[{required: true}]}>
      <Input v-model={[formData.value.name, 'value']}/>
    </FormItem>
    <FormItem label='复制一点的组件' name='age' rules={[{required: true}]}>
      { renderComplex() }
    </FormItem>
    {/* 好多存在关联关系的表单项 */}
    { renderCustomInfo() }
  </Form>;
},
```

拆解Dom为多个render函数，防止Dom标签缩进太深

当你发现一行代码的缩进超过5个tab时，就要考虑拆解出来了。
可以增强代码的可读性。也有助于梳理代码逻辑

```
// 按权重配置奖品数量 - 优惠券,到店核销券
const renderCouponPrizeNum = () => <FormItem label='每人中奖上限: '>...</FormItem>;
// 按权重配置奖品数量 - 实物
const renderMaterialPrizeNum = () => <FormItem label='每人中奖上限'>...</FormItem>;
// 按权重配置奖品数量
const renderPrizeNum = () => {
  switch (formDataRef.value.type) {
    case PRIZE_TYPE.coupon:
    case PRIZE_TYPE.writeOff:
      return renderCouponPrizeNum();
    default:
      return renderMaterialPrizeNum();
  }
};
```

避免使用魔鬼字符，用常量替换

```
//员工组件结构器回显
orgNodes.value = userIds.value.map((t) => ({ id: t, nodeType: NodeTypeEmployee }));
```

```
// 由组件统一导出
export const NodeTypeEmployee = 'NodeTypeEmployee';

// 使用时引入常量
import { NodeTypeEmployee } from '@/components/mk-organization-picker';

orgNodes.value = userIds.value.map((t) => ({ id: t, nodeType: NodeTypeEmployee }));
```

看一个复制文本的工具类

mk-manage-traffic 实现方式

```
export function copyText(e: MouseEvent, text: string|undefined, successMsg?:string, errorMsg?: string) {
  const copyAttr = 'data-clipboard-text';
  if (e.currentTarget) {
    const el = e.currentTarget as HTMLButtonElement;
    const hasCopyAttr = el.hasAttribute(copyAttr);
    el.setAttribute(copyAttr, text === undefined ? '' : text);

    const clipboard = new Clipboard(e.currentTarget as any);
    clipboard.on('success', () => {
      message.success(successMsg || '复制成功');
      clipboard.destroy();
    });
    clipboard.on('error', () => {
      clipboard.destroy(); // 释放内存
    });

    // 触发首次复制失败的问题
    if (!hasCopyAttr) {
      el.click();
    }
  }
}
```

qw-client-mk-common 实现方式

```
/**
 * 复制文本
 */
export function copyText(e: Event, text: string|undefined) {
  const clipboard = new Clipboard(e.target as any, {
    text: () => text || '',
  });
  clipboard.on('success', () => {
    message.success('复制成功');
    clipboard.destroy();
  });
  (clipboard as any).onClick(e);
}
```

qw-mobile-mk 中的实现方式

```
app.config.globalProperties.$clipboard =
  (className: string, successMsg = '复制成功', errorMsg = '复制失败') => {
  const clipboard = new Clipboard(className);
  console.log('clipboard: ', className, clipboard);
  clipboard.on('success', (e) => {
    e.clearSelection();
    Toast(successMsg);
    console.log('successMsg: ', successMsg);
    // 释放内存
    clipboard.destroy();
  });
  clipboard.on('error', () => {
    Toast(errorMsg);
    console.log('errorMsg: ', errorMsg);
    // 释放内存
    clipboard.destroy();
  });
};
```

```
<pre
  class="flex-1 mk-fs-14 copyTip cursor"
  :data-clipboard-text="item.content"
  @click="handleCopy(item.content)"
  >{{ item.content }}</pre>
```