虚拟树组件

一.定义树中Props

组件的基本应用，传入数据即可根据数据渲染树组件

```
<z-tree :data="data"></z-tree>
```

> 定义树中所需的基本`props`属性

```
// 树的数据类型
export type Key = string | number
export interface TreeOption {
  key?: Key
  label?: string
  isLeaf?: boolean
  children?: TreeOption[]
  [k: string]: unknown
}
export const treeProps = {
  defaultExpandedKeys: {
    // 1.要展开的key
    type: Array as PropType<Key[]>,
    default: () => []
  },
  keyField: {
    // 2.key字段的别名
    type: String,
    default: 'key'
  },
  labelField: {
    // 3.label字段的别名
    type: String,
    default: 'label'
  },
  childrenField: {
    // 4.children字段的别名
    type: String,
    default: 'children'
  },
  data: {
    // 5.所有数据
    type: Array as PropType<TreeOption[]>,
    default: () => []
  }
} as const
export type TreeProps =
Partial<ExtractPropTypes<typeof treeProps>>
```

> 我们需要对用户传入的数据进行格式化后在使用。格式化后的props是这个酱紫的~

```
export interface TreeNode extends
Required<TreeOption>{
  level: number // 层级
  children: Array<TreeNode> // 儿子数组
  rawNode: TreeOption // 原始的node
}
```

```
import { withInstall } from '@zi-
shui/utils/with-install'
import _Tree from './src/tree.vue'
const Tree = withInstall(_Tree) // 生成带有
install方法的组件
export default Tree // 导出tree组件
declare module 'vue' {
  export interface GlobalComponents {
    ZTree: typeof Tree
  }
}
export * from './src/tree'
```

二.数据格式化和组件渲染

1.创建渲染数据

```
function createData(level = 4, parentKey = ''):
TreeOption[] {
  if (!level) return []
  const arr = new Array(6 - level).fill(0)
  return arr.map((_, idx: number) => {
    const key = parentKey + level + idx
    return {
      label: createLabel(level),
      key,
      children: createData(level - 1, key)
    }
  })
}
function createLabel(level: number): string {
  if (level === 4) return '道生一'
  if (level === 3) return '一生二'
  if (level === 2) return '二生三'
  if (level === 1) return '三生万物'
  return ''
}
const data = ref<TreeOption[]>(createData())
```

2.封装获取属性方法

```
<script lang="ts" setup>
import { createNamespace } from '@zi-
shui/utils/create'
import { computed} from 'vue'
import { treeProps, TreeOption, TreeNode, Key }
from './tree'
const bem = createNamespace('tree')
defineOptions({
  name: 'ZTree'
})
const props = defineProps(treeProps)
// 1) 封装获取方法
function createTreeOptions(keyField: string,
childrenField: string,labelField:string) {
  return {
    getKey<T>(node:T):Key {
      return node[keyField]
    },
    getChildren<T>(node:T):T[] {
      return node[childrenField]
    },
    getLabel<T>(node:T):string{
      return node[labelField]
    }
  }
}
const treeOptions =
createTreeOptions(props.keyField,
props.childrenField,props.labelField)
</script>
```

3.数据格式化

```
function createTree(data: TreeOption[], parent: TreeNode | null = null) {
  function traversal(data: TreeOption[], parent: TreeNode | null): TreeNode[] {
    return data.map(node => {
      let children = treeOptions.getChildren(node) || [] // 获得所有的孩子
      const childrenLen = children.length || 0
      const treeNode: TreeNode = {
        key: treeOptions.getKey(node),
        label: treeOptions.getLabel(node),
        level: parent ? parent.level + 1 : 0,
        isLeaf: node.isLeaf ?? childrenLen == 0,
        children: [],
        rawNode: node
      }
      if (childrenLen > 0) {
        treeNode.children = traversal(children, treeNode)
      }
      return treeNode
    })
  }
  const result: TreeNode[] = traversal(data, parent)
  return result
}
const tree = ref<Array<TreeNode>>([]) // tree数据列表
watch(
  () => props.data, // 监控data变化，重新创建树
  (data: TreeOption[]) => {
    tree.value = createTree(data)
    console.log(tree.value)
  },
  { immediate: true }
)
```

4.根据expandedKeys拍平数组

```
const expandedKeySet = ref<Set<Key>>(new Set(props.defaultExpandedKeys))
const flattenTree = computed(() => {
  const expandedKeys = expandedKeySet.value // 需要展开的key
  const flattenNodes: TreeNode[] = []
  const nodes = tree.value || []
  const stack: TreeNode[] = [] // 存放节点的
  for (let i = nodes.length - 1; i >= 0; --i) {
    stack.push(nodes[i]) // 节点2 节点1
  }
  while (stack.length) {
    const node = stack.pop(); // 拿到节点1
    if (!node) continue
    flattenNodes.push(node); // 将节点1入队列
    if (expandedKeys.has(node.key)) { // 如果需要展开
      const children = node.children
      if (children) {
        const length = children.length; // 将节点1的儿子  child3 child2 child1入栈
        for (let i = length - 1; i >= 0; --i) {
          stack.push(children[i])
        }
      }
    }
  }
  return flattenNodes
})
```

5.渲染Tree组件

```
<div :class="bem.b()">
    <z-tree-node
      v-for="node in flattenTree"
      :node="node"
      :expanded="isExpanded(node)"
    ></z-tree-node>
</div>
<script>
// ...
function isExpanded(node: TreeNode): boolean {
  return expandedKeySet.value.has(node.key)
}
</script>
```

三.抽离TreeNode组件

> tree-node组件中需要使用展开图标，这里采用tsx编写内置图标组件

```
import { h, defineComponent } from 'vue'
export default defineComponent({
  name: 'Switcher',
  render () {
    return (
      <svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 32 32">
        <path d="M12 8l10 8l-10 8z" />
      </svg>
    )
  }
})
```

> 需要再`vite`中配置`@vitejs/plugin-vue-jsx`插件，才可以正确解析`tsx`语法

1.定义TreeNodeProps

```
export const treeNodeProps = {
  node:{
    type: Object as PropType<TreeNode>,
    required:true
  },
  expanded:{
    type:Boolean,
    default:false
  }
} as const
export type TreeNodeProps =
Partial<ExtractPropTypes<typeof treeNodeProps>>
```

```
<template>
  <div :class="bem.b()">
    <div :class="bem.e('content')">
      <span
        :class="[
          bem.e('expand-icon'),
          bem.is('leaf', node.isLeaf),
          { expanded: !node.isLeaf && expanded }
        ]"
      >
        <z-icon>
          <Switcher></Switcher>
        </z-icon>
      </span>
      <span> {{ node.label }}</span>
    </div>
  </div>
</template>

<script lang="ts" setup>
import { createNamespace } from '@zi-
shui/utils/create'
import { PropType, Ref, ref, toRef, watch } from
'vue'
import { TreeNode, treeNodeProps } from './tree'
import ZIcon from '@zi-shui/components/icon'
import Switcher from './icon/Switcher'
const bem = createNamespace('tree-node')
const props = defineProps(treeNodeProps)
</script>
```

2.编写基本样式

```
@use 'mixins/mixins' as *;
@use 'common/var' as *;
@include b('tree') {
  display: inline-block;
  width: 100%;
}
@include b('tree-node') {
  padding: 5px 0;
  font-size:14px;
  &:hover {
    background-color:#f5f7fa;
  }
  @include e(expand-icon){
    display: inline-block;
    cursor: pointer;
    transform: rotate(0deg);
    transition: transform 0.1s ease-in-out;
    &.expanded{
      transform:rotate(90deg);
    }
    &.is-leaf{
      fill:transparent;
      cursor: default;
    }
  }
}
```

3.展开收缩功能

> 定义触发切换的事件

```
export const treeNodeEvents = {
  toggle:(node:TreeNode)=> node
}
```

```
<div
      :class="bem.e('content')"
>
      <span
        :class="[
          bem.e('expand-icon'),
          bem.is('leaf', node.isLeaf),
          { expanded: !node.isLeaf && expanded }
        ]"
      >
        <z-icon>
          <Switcher
@click.stop="handleExpandIconClick(node)">
</Switcher>
        </z-icon>
      </span>
      <span> {{ node.label }}</span>
    </div>
<script>
const emit = defineEmits(treeNodeEvents)
const handleExpandIconClick = (node:TreeNode)=>{
  emit('toggle',node); // 触发toggle事件
}
</script>
```

> 组件监听toggle事件

```
<z-tree-node
      v-for="node in flattenTree"
      :node="node"
      :expanded="isExpanded(node)"
      @toggle="toggleExpand"
></z-tree-node>
<script>
function collapse(node: TreeNode) {
  expandedKeySet.value.delete(node.key)
}
function expand(node: TreeNode) {
  const keySet = expandedKeySet.value
  keySet.add(node.key)
}
function toggleExpand(node: TreeNode) {
  const expandedKeys = expandedKeySet.value
  if (expandedKeys.has(node.key)) {
    collapse(node)
  } else {
    expand(node)
  }
}
</script>
```

4.增加indent值

> 根据层级实现缩进

```
<div :class="bem.e('content')" :style="
{paddingLeft:`${(node.level)*16 +'px'}`}">
  <span>
    <z-icon
@click.stop="handleExpandIconClick(node)">
      <Switcher></Switcher>
    </z-icon>
  </span>
  <span> {{ node.label }}</span>
</div>
```

四.树组件异步加载

1.构建异步数据

```
<script>
function createData() {
  return [
    {
      label: nextLabel(),
      key: 1,
      isLeaf: false
    },
    {
      label: nextLabel(),
      key: 2,
      isLeaf: false
    }
  ]
}
function nextLabel(currentLabel?: string):
string {
  if (!currentLabel) return 'Out of Tao, One is
born'
  if (currentLabel === 'Out of Tao, One is
born') return 'Out of One, Two'
  if (currentLabel === 'Out of One, Two') return
'Out of Two, Three'
  if (currentLabel === 'Out of Two, Three') {
    return 'Out of Three, the created universe'
  }
  if (currentLabel === 'Out of Three, the
created universe') {
    return 'Out of Tao, One is born'
  }
  return ''
}

const data = ref<TreeOption[]>(createData())
const handleLoad = (node: TreeOption) => {
  // 每次实现懒加载时，会触发此方法，将当前点击的node传入
  return new Promise<TreeOption[]>((resolve, reject) => {
    setTimeout(() => {
      resolve([
        {
          label: nextLabel(node.label),
          key: node.key + nextLabel(node.label),
          isLeaf: false
        }
      ])
    },2000)
  })
}
</script>
<template>
  <z-tree :data="data" :on-load="handleLoad">
</z-tree>
</template>
```

```
{
  onLoad:Function as PropType<(node:TreeOption)=>Promise<TreeOption[]>>
}
```

2.实现触发加载

```
const loadingKeysRef = ref(new Set<Key>()) // 存储正在加载的key
function triggerLoading(node){
  if (!node.children.length && !node.isLeaf) { // 需要异步加载
    const loadingKeys = loadingKeysRef.value
    const { onLoad } = props // 有onLoad方法
    if (!loadingKeys.has(node.key)) { // 防止重复加载
      loadingKeys.add(node.key) // 添加为正在加载
      if (onLoad) { // 调用用户提供的加载方法

 onLoad(node.rawNode).then((children:TreeOption[]) => {
          node.rawNode.children = children;
          node.children = createTree(children,node); // 格式化后绑定children属性
          loadingKeys.delete(node.key); // 加载完毕移除key
        })
      }
    }
  }
}
function expand(node: TreeNode) {
  const keySet = expandedKeySet.value
  keySet.add(node.key)
  triggerLoading(node);// 展开时触发加载逻辑
}
function toggleExpand(node: TreeNode) {
  const expandedKeys = expandedKeySet.value
  if (expandedKeys.has(node.key) &&
!loadingKeysRef.value.has(node.key)) {
    collapse(node) // 如果现在是正在加载中，则不进行
收起操作
  } else {
    expand(node)
  }
}
```

3.loading图标实现

```
import { h, defineComponent } from 'vue'

export default defineComponent({
  name: 'Loading',
  render () {
    return (
      <svg
      xmlns="http://www.w3.org/2000/svg"
      viewBox="0 0 1024 1024"
      class="loading"
    >
      <path
        d="M512 1024c-69.1 0-136.2-13.5-199.3-
40.2C251.7 958 197 921 150 874c-47-47-84-101.7-
109.8-162.7C13.5 648.2 0 581.1 0 512c0-19.9
16.1-36 36-36s36 16.1 36 36c0 59.4 11.6 117 34.6
171.3c22.2 52.4 53.9 99.5 94.3 139.9c40.4 40.4
87.5 72.2 139.9 94.3C395 940.4 452.6 952 512
952c59.4 0 117-11.6 171.3-34.6c52.4-22.2 99.5-
53.9 139.9-94.3c40.4-40.4 72.2-87.5 94.3-
139.9C940.4 629 952 571.4 952 512c0-59.4-11.6-
117-34.6-171.3a440.45 440.45 0 0 0-94.3-
139.9a437.71 437.71 0 0 0-139.9-94.3C629 83.6
571.4 72 512 72c-19.9 0-36-16.1-36-36s16.1-36
36-36c69.1 0 136.2 13.5 199.3 40.2C772.3 66 827
103 874 150c47 47 83.9 101.8 109.7 162.7c26.7
63.1 40.2 130.2 40.2 199.3s-13.5 136.2-40.2
199.3C958 772.3 921 827 874 874c-47 47-101.8
83.9-162.7 109.7c-63.1 26.8-130.2 40.3-199.3
40.3z"
        fill="currentColor"
      ></path>
    </svg>
    )
```

```
  }
})
```

> 将loadingKeys列表传入

```
<z-tree-node
      v-for="node in flattenTree"
      :node="node"
      :expanded="isExpanded(node)"
      @toggle="toggleExpand"
      :loadingKeys="loadingKeysRef"
></z-tree-node>
```

```
export const treeNodeProps = {
  loadingKeys:{
    type:Object as PropType<Set<Key>>
  }
} as const
```

> 根据传入的loadingKeys判断是否需要显示loading图标

```
<z-icon>
  <Switcher v-if="!isLoading"></Switcher>
  <Loading v-else></Loading>
</z-icon>
<script>
const isLoading = computed(() => {
  return props.loadingKeys?.has(props.node.key)
})
</script>
```

五.实现禁用、多选节点

1.计算选中列表

```
<z-tree :data="data" selectable v-
model:selected-keys="value"></z-tree>
```

```
export const treeProps = {
  multiple: Boolean,
  selectable:{
    type:Boolean,
    default:true,
  },
  selectedKeys: Array as PropType<Key[]>,
} as const
export const treeEvents = {
  'update:selectedKeys':(keys:Key[])=> keys
}
```

```
const emit = defineEmits(treeEvents)
const selectedKeys = ref<Key[]>([]) // 选中的key
列表
watch(
  () => props.selectedKeys, // 监控selectedKeys
  value => {
    if (value != undefined) {
      selectedKeys.value = value
    }
  },
  { immediate: true }
)
function handleSelect(node: TreeNode) {
  let keys = Array.from(selectedKeys.value);
  if (!props.selectable) {
    // 如果不支持选中
    return
  }
  if (props.multiple) { // 支持多选
     const index = keys.findIndex(key=>key ===
node.key)
     if(index > -1){
      keys.splice(index,1);
     }else{
      keys.push(node.key)
     }
  } else {
    if (keys.includes(node.key)) {
      // 如果选中的包含则清空
      keys = []
    } else {
      keys = [node.key]
    }
  }
  emit('update:selectedKeys',keys)
}
```

```
<z-tree-node
    :selectKeys="selectedKeys"
    @select="handleSelect"
></z-tree-node>
```

```
export const treeNodeProps = {
  // ...
  selectKeys:{
    type:Array as PropType<Key[]>
  }
} as const
export const treeNodeEvents = {
   select:(node:TreeNode)=> node,
}
```

2.实现选中状态

```
<div :class="
[bem.b(),bem.is('selected',isSelected)]">
    <div
      :class="bem.e('content')"
      :style="{ paddingLeft: `${node.level * 16
+ 'px'}` }"
    >
      <!-- ... -->
      </span>
      <span @click="handleContentClick(node)"
:class="bem.e('label')"> {{ node.label }}</span>
    </div>
</div>
```

```
const isSelected = computed(() => { // 判断是否选
中
  return
props.selectKeys?.includes(props.node.key)
})
const handleContentClick =(node: TreeNode)=>{ //
内容点击触发选择
  emit('select', node)
}
```

```
@include b('tree-node') {
  @include when(selected){
    background-color:#e7f5ee;
  }
  @include e(content){
    display: flex;
  }
  @include e(label){
    cursor: pointer;
    flex:1
  }
}
```

3.禁用节点

```
const data = ref<TreeOption[]>([
  {
    key: '0',
    label: '0',
    children: [
      {
        key: '0-0',
        label: '0-0'
      },
      {
        disabled: true,
        key: '0-1',
        label: '0-1',
        children: [
          {
            label: '0-1-0',
            key: '0-1-0'
          },
          {
            label: '0-1-1',
            key: '0-1-1'
          }
        ]
      }
    ]
  }
])
```

```
const treeNode: TreeNode = {
  key: treeOptions.getKey(node),
  label: treeOptions.getLabel(node),
  level: parent ? parent.level + 1 : 0,
  isLeaf: node.isLeaf ?? childrenLen == 0,
  children: [],
  disabled:!!node.disabled, // 添加disabled属性
  rawNode: node
}
```

```
<div :class="
[bem.b(),bem.is('selected',isSelected),bem.is('d
isabled',node.disabled)]">
  <span @click="handleContentClick(node)"
:class="[bem.e('label')]"> {{ node.label }}
</span>
</div>
```

```
@include b('tree-node') {
  &:not(.is-disabled){
    .z-tree-node__label{
      cursor: pointer;
      flex:1
    }
  }
  &.is-disabled{
    .z-tree-node__label{
      cursor: not-allowed;
      flex:1;
      color:#cdcdcd;
    }
  }
}
```

> 在选中节点时判断，节点是否为禁用状态

```
const handleContentClick =(node: TreeNode)=>{
  if(node.disabled) return;
  emit('select', node)
}
```

4.自定义节点内容

```
// 创建上下文对象，提供注入实现
export interface TreeContext {
  slots: SetupContext['slots'] // 插槽属性
}
export const treeInjectionKey:
InjectionKey<TreeContext> = Symbol()
```

```
provide(treeInjectionKey, {
  slots: useSlots() // 提供slots属性
});
```

```
<span @click="handleContentClick(node)"
:class="bem.e('label')">
    <ZTreeNodeContent :node="node">
</ZTreeNodeContent>
</span>
```

```
export const treeNodeContentProps = {
  node: {
    type: Object as PropType<TreeNode>,
    required: true
  }
} as const
```

```
import { defineComponent, inject } from "vue"
import { treeInjectionKey, treeNodeContentProps
} from "./tree"

export default defineComponent({
  name:'ZTreeNodeContent',
  props: treeNodeContentProps,
  setup(props) {
    const tree = inject(treeInjectionKey)
    return () => {
      const node = props.node
      return tree?.slots.default
        ? tree.slots.default({ node })
        : node?.label
    }
  }
})
```

六.可选择的树

1.基本功能

```
<z-tree
    :data="data"
    multiple
    selectable
    v-model:selected-keys="value"
    show-checkbox
>
</z-tree>
```

```
export const treeProps = {
  showCheckbox: {
    type: Boolean,
    default: false
  },
  defaultCheckedKeys:{
    type: Array as PropType<Key[]>,
    default: () => []
  }
} as const
```

> 在父组件中将属性传递下去~

```
<z-tree-node
      v-for="node in flattenTree"
      :node="node"
      :expanded="isExpanded(node)"
      @toggle="toggleExpand"
      :loadingKeys="loadingKeysRef"
      :selectKeys="selectedKeys"
      @select="handleSelect"
      :show-checkbox="showCheckbox"
></z-tree-node>
```

```
export const treeNodeProps = {
  showCheckbox:{
    type:Boolean,
    required:true
  },
  checked:Boolean,
  disabled:Boolean,
  indeterminate:Boolean
} as const
```

```
 <div :class="
[bem.b(),bem.is('selected',isSelected),,bem.is('
disabled',node.disabled)]">
    <div
      :class="bem.e('content')"
      :style="{ paddingLeft: `${node.level * 16
+ 'px'}` }"
    >
      <span>
        <input type="checkbox" v-
if="showCheckbox">
      </span>
      <span @click="handleContentClick(node)"
:class="bem.e('label')">
        <ZTreeNodeContent :node="node">
</ZTreeNodeContent>
      </span>
    </div>
</div>
```

2.封装checkbox组件

```
import { ExtractPropTypes, PropType } from
"vue";
export const checkboxProps = {
  modelValue:{
    type:[Boolean,Number,String] as
PropType<boolean | number | string>
  },
  label:{
    type:[Boolean,Number,String] as
PropType<boolean | number | string>
  },
  indeterminate: Boolean,
  disabled: Boolean,
} as const

export type CheckboxProps =
Partial<ExtractPropTypes<typeof checkboxProps>>
export const checkboxEmits = {
  change: (value:boolean) => typeof value ===
'boolean',
  'update:modelValue':(value:boolean | number |
string ) => value
}
export type CheckboxEmits = typeof checkboxEmits
```

```
<template>
  <label :class="bem.b()">
    <span :class="bem.e('input')">
      <input
        type="checkbox"
        :value="label"
        :disabled="disabled"
        v-model="model"
        :checked="isChecked"
        @change="handleChange"
        ref="inputRef"
      />
    </span>
    <!-- 没有默认 有label -->
    <span v-if="$slots.default || label"
:class="bem.e('label')">
      <slot></slot>
      <template v-if="!$slots.default">{{ label
}}</template>
    </span>
  </label>
</template>

<script lang="ts" setup>
import { createNamespace } from '@zi-
shui/utils/create'
import { computed, onMounted, ref, watch } from
'vue'
import { checkboxEmits, CheckboxProps,
checkboxProps } from './checkbox'
const bem = createNamespace('checkbox')
defineOptions({
  name: 'ZCheckbox'
```

```
})
const emit = defineEmits(checkboxEmits)
const props = defineProps(checkboxProps)
const useModel = (props: CheckboxProps) => {
  const model = computed<string | number |
boolean>({
    get() {
      return props.modelValue!
    },
    set(val) {
      emit('update:modelValue', val)
    }
  })
  return model
}
const useCheckboxStatus = (props: CheckboxProps,
model) => {
  const isChecked = computed(() => {
    const value = model.value
    return value
  })
  return isChecked
}
const useEvent = () => {
  // checkbox修改事件
  function handleChange(e: Event) {
    const target = e.target as HTMLInputElement
    const value = target.checked ? true : false
// 获取checked属性，触发修改逻辑
    emit('change', value)
  }
  return handleChange
}
function useCheckbox(props: CheckboxProps) {
  // 1.实现用于双向绑定的model属性
  const model = useModel(props)
  const isChecked = useCheckboxStatus(props,
model)
  const handleChange = useEvent()
  return {
    model,
    isChecked,
    handleChange
  }
}
const { model, isChecked, handleChange } =
useCheckbox(props)

const inputRef = ref<HTMLInputElement>()
function indeterminate(val) {
  inputRef.value!.indeterminate = val
}
watch(() => props.indeterminate, indeterminate)
onMounted(() => {
  // 默认加载完毕后
  indeterminate(props.indeterminate)
})
</script>
```

3.指定tree-node中属性

```
<z-tree-node
      v-for="node in flattenTree"
      :node="node"
      :expanded="isExpanded(node)"
      @toggle="toggleExpand"
      :loadingKeys="loadingKeysRef"
      :selectKeys="selectedKeys"
      @select="handleSelect"
      :show-checkbox="showCheckbox"
      :checked="isChecked(node)"
      :disabled="isDisabled(node)"
      :indeterminate="isIndeterminate(node)"
></z-tree-node>

<script>
// 稍后更新选中集合
const checkedKeySet = ref<Set<Key>>(new
Set(props.defaultCheckedKeys))
function isChecked(node:TreeNode){
   return checkedKeySet.value.has(node.key)
}
function isDisabled(node:TreeNode){
  return !!node.disabled
}
// 稍后更新半选集合
const indeterminateKeySet = ref<Set<Key>>(new
Set())
function isIndeterminate(node:TreeNode){
  return true
}
</script>
```

4.checkbox接收属性

```
<Checkbox
  :indeterminate="indeterminate"
  :model-value="checked"
  :disabled="disabled"
  @change="handleCheckChange"
></Checkbox>
```

```
const handleCheckChange = (value: boolean) => {
   emit('check',props.node,value)
}
```

```
export const treeNodeEvents = {
  toggle:(node:TreeNode)=> node,
  select:(node:TreeNode)=> node,
  check: (node: TreeNode,value:boolean) => node
}
```

```
function toggleCheckbox(node: TreeNode,
isChecked: boolean) {
  toggle(node,isChecked); // 控制孩子切换
  updateCheckedKeys(node);
}
```

> 这里需要构建panrentKey

```
const treeNode: TreeNode = {
        key: treeOptions.getKey(node),
        label: treeOptions.getLabel(node),
        level: parent ? parent.level + 1 : 0,
        isLeaf: node.isLeaf ?? childrenLen == 0,
        children: [],
        disabled:!!node.disabled, // 添加disabled
属性
        rawNode: node,
        parentKey:parent?.key!,
}
```

```
function toggle(node:TreeNode,isChecked:boolean)
{
    let checkKeys = checkedKeySet.value
    if(isChecked){
      indeterminateKeySet.value.delete(node.key)
    }
    checkKeys[isChecked ? 'add' : 'delete']
(node.key)
    const children = node.children
    if (children) {
      children.forEach(childNode => {
        if (!childNode.disabled) {
          toggle(childNode,isChecked);
        }
      })
    }
}
function updateCheckedKeys(node:TreeNode){
    if(node.parentKey){ //有父key 存在
       let parentNode =
flattenTree.value.find(item=> item.key ===
node.parentKey);// 找到父节点
       if(parentNode){
            let allChecked = true;
            let hasChecked = false;
            let nodes = parentNode.children; //
获取孩子节点
```

```
              for(let node of nodes){

 if(checkedKeySet.value.has(node.key)){ // 孩子被
选中
                  hasChecked = true;
                }else
if(indeterminateKeySet.value.has(node.key)){ //
孩子是半选
                  allChecked = false;
                  hasChecked = true;
                }else{
                  allChecked = false;
                }
              }

              if(allChecked){

 checkedKeySet.value.add(parentNode.key)

 indeterminateKeySet.value.delete(parentNode.key
)
              }else if(hasChecked){

 indeterminateKeySet.value.add(parentNode.key)

 checkedKeySet.value.delete(parentNode.key)
              }else{
```

```
 checkedKeySet.value.delete(parentNode.key)

 indeterminateKeySet.value.delete(parentNode.key
)
                }
                updateCheckedKeys(parentNode); // 自
己搞定再看父级
         }
     }
}
```

七.虚拟滚动组件

```
import { withInstall } from '@zi-shui/utils/withInstall'
import _vertual from './src/virtual'

const VirtualList = withInstall(_vertual) // 生成带有install方法的组件

export default VirtualList // 导出Icon组件

declare module 'vue' {
  export interface GlobalComponents {
    ZVirtualList: typeof VirtualList
  }
}
```

```
 <z-virtual-list :items="flattenTree">
      <template #default="{ node,idx }">
        <z-tree-node
          :node="node"
          :idx="node.key"
          :expanded="isExpanded(node)"
          :loadingKeys="loadingKeysRef"
          :selectKeys="selectedKeys"
          :show-checkbox="showCheckbox"
          :checked="isChecked(node)"
          :disabled="isDisabled(node)"
          :indeterminate="isIndeterminate(node)"
          @check="toggleCheckbox"
           @toggle="toggleExpand"
          @select="handleSelect"
        ></z-tree-node>
      </template>
 </z-virtual-list>
```

```
import { createNamespace } from "@zi-
shui/utils/create";
import { computed, defineComponent, h,
onMounted, reactive, ref } from "vue";

export default defineComponent({
  name:'ZVirtualList',
  props:{
    size:{
      type:Number,
      default:30
    },
    remain:{
      type:Number,
      default:8
    },
    items:{
      type:Array,
      default:()=>[]
    }
  },
  setup(props,{slots}){

    const bem = createNamespace('vl');

    const wrapperRef = ref<HTMLElement>()
    const barRef = ref<HTMLElement>();

    const state = reactive({
        start:0, // 从哪里开始
        end:props.remain
    });
    const offset = ref(0);

    const prev = computed(()=>{
      return Math.min(state.start,props.remain)

    })
    const next = computed(()=>{
      return
Math.min(props.remain,props.items.length -
state.end)
    })
    const visibleData = computed(()=>{
      return props.items.slice(state.start -
prev.value,state.end + next.value);
    })

    const handleScroll = ()=>{
      // 算出来 当前滚过去几个了，当前从第几个显示
      let scrollTop =
 wrapperRef.value!.scrollTop;
      state.start = Math.floor(scrollTop /
props.size);
      state.end = state.start + props.remain;
      offset.value =state.start * props.size -
props.size * prev.value; // 让可视区域调整
    }
    onMounted(()=>{
      wrapperRef.value!.style.height =
props.size * props.remain  + 'px';
      barRef.value!.style.height =
props.items.length * props.size + 'px'
    })
    return ()=>{
      return <div class={bem.b()} ref=
{wrapperRef} onScroll={handleScroll} >
        <div class={bem.e('scroll-bar')} ref=
{barRef}></div>
```

```
        <div class={bem.e('scroll-list')} style=
{{transform:`translate3d(0,${offset.value}px,0)`
,color:'red'}}>
        {
          visibleData.value.map( (node,idx) =>
slots.default?.({node,idx}))
        }
         </div>
      </div>
    }
  },
})
```

```
@use 'mixins/mixins' as *;
@include b('vl') {
  overflow-y:scroll;
  position: relative;
  border:2px solid #ddd;
  @include e('scroll-list') {
    position: absolute;
    top:0;
    left:0;
    width: 100%;
  }
}
```