FormItem组件

一.定义FormItem Props

```
import { ExtractPropTypes, PropType } from 'vue'
import type { RuleItem } from 'async-validator'
export type Arrayable<T> = T | T[]
export interface FormItemRule extends RuleItem {
  trigger?: Arrayable<string>
}
// form 表单校验时的状态
export const formItemValidateStates = ['',
'error', 'success'] as const
export type FormItemValidateState = typeof
formItemValidateStates[number]

export const formItemProps = {
  label: String,
  prop: String,
  rules: [Object, Array] as
PropType<Arrayable<FormItemRule>>,
  showMessage: {
    type: Boolean,
    default: true
  }
```

```
} as const
export type FormItem = ExtractPropTypes<typeof
formItemProps>
```

二.form-item结构实现

```
<template>
  <div :class="formItemClasses">
    <!-- label属性 -->
    <label :class="[bem.e('label')]">
      <slot name="label"> {{ label }} </slot>
    </label>
    <!-- content盒子 -->
    <div :class="[bem.e('content')]">
      <!-- input  -->
      <slot></slot>
      <!-- 错误信息 -->
      <slot v-if="shouldShowError" name="error">
        <div :class="[bem.e('error')]">
          {{ validateMessage }}
        </div>
      </slot>
    </div>
  </div>
```

```
</template>
<script lang="ts" setup>
import { createNamespace } from '@zi-
shui/utils/create'
import { computed, ref } from 'vue'
import { formItemProps, FormItemValidateState }
from './form-item'

const validateState = ref<FormItemValidateState>
('error') // 校验状态
const validateMessage = ref('校验失败') // 错误消息
const bem = createNamespace('form-item')
const formItemClasses = computed(() => [
  bem.b(),
  bem.is('success', validateState.value ===
'success'),
  bem.is('error', validateState.value ===
'error')
])
defineOptions({
  name: 'ZFormItem'
})
const props = defineProps(formItemProps)
const shouldShowError = computed(() => {
  // 当状态为失败并且需要显示错误消息时
  return validateState.value === 'error' &&
props.showMessage
```

```
})
</script>
```

三.Form-item入口编写

```
import _FormItem from './src/form-item.vue'
import { withInstall } from '@zi-
shui/utils/withInstall'

export const FormItem = withInstall(_FormItem)
export type FormItemInstance =
InstanceType<typeof FormItem>
```

四.Form-item组件试用

```
<script setup lang="ts">
import { ref } from 'vue'
const value = ref('')
</script>
<template>
  <z-form-item label="username">
    <z-input v-model="value"></z-input>
  </z-form-item>
</template>
```

五.组件功能实现

1).样式编写

```
@use 'mixins/mixins' as *;
@use 'common/var' as *;
@include b(form-item) {
  display: flex;
  @include e(label) {
    display: inline-flex;
    justify-content: flex-end;
    align-items: flex-start;
    padding-right: 12px;
    height: 32px;
    line-height: 32px;
  }
  @include e(content) {
    display: flex;
    flex: 1;
    flex-wrap: wrap;
  }
  @include e(error) {
    color: $color-danger;
    font-size: 12px;
  }
} ;
```

2).FormItem校验

```
<z-form-item
    label="用户名"
    prop="username"
    :rules="[
      { required: true, message: '用户名必须填写',
trigger: 'blur' },
      { min: 6, message: '用户名最少6位', trigger:
'change' }
    ]"
>
  <z-input v-model="value"></z-input>
</z-form-item>
```

> form/src/form-item.ts

```
// 生成formItemContext上下文, 增添校验方法
export interface FormItemContext extends
FormItemProps {
  validate(trigger: string, callback: (isValid:
boolean) => void)
}
// injectKey的生成
export const formItemContextKey:
InjectionKey<FormItemContext> = Symbol()
```

> 依赖注入

```
// 对规则进行格式化
const _rules = computed(() => {
  const rules: FormItemRule[] = props.rules
    ? Array.isArray(props.rules)
      ? props.rules
      : [props.rules]
    : []

  return rules
})
// 根据对应的trigger类型过滤规则
const getFilteredRule = (trigger: string) => {
  const rules = _rules.value
  return rules.filter(rule => {
    if (!rule.trigger || !trigger) return true
    if (Array.isArray(rule.trigger)) {
      return rule.trigger.includes(trigger)
    } else {
      return rule.trigger == trigger
    }
  })
}
// 暴露校验方法
const validate: FormItemContext['validate'] =
async trigger => {
```

```
  const rules = getFilteredRule(trigger)
  console.log('校验', rules)
}
const context: FormItemContext = {
  validate
}
provide(formItemContextKey, context)
```

> 在input组件中触发校验

```
const formItem = inject(formItemContextKey)
// 监控值的变化，触发blur事件
watch(
  () => props.modelValue,
  () => {
    formItem?.validate('change')
  }
)
const handleBlur = (event: FocusEvent) => {
  emit('blur', event)
  formItem?.validate?.('blur')
}
```

Form组件

一.定义Form Props

```ts
import { ExtractPropTypes, InjectionKey,
PropType, SetupContext } from 'vue'
import { FormItemRule, Arrayable } from './form-
item'

export const formProps = {
  model: Object, // 数据
  rules: {
    // 校验规则
    type: Object as PropType<Record<string,
Arrayable<FormItemRule>>>
  },
  showMessage: {
    // 显示消息
    type: Boolean,
    default: true
  }
} as const

// 表单属性
export type FormProps =
Partial<ExtractPropTypes<typeof formProps>>
export type FormContext = FormProps
// 表单上下文key, 包含所有props
export const formContextKey:
InjectionKey<FormContext> =
  Symbol('formContextKey')
```

二.Form结构实现

```
<template>
  <form :class="[bem.b()]">
    <slot></slot>
  </form>
</template>

<script lang="ts" setup>
import { createNamespace } from '@zi-
shui/utils/create'
import { provide, reactive, toRefs } from 'vue'
import { formContextKey, formProps } from
'./form'
const bem = createNamespace('form')
const props = defineProps(formProps)
defineOptions({
  name: 'ZForm'
})
provide(
  formContextKey,
  reactive({
    ...toRefs(props)
  })
)
```

```
</script>
```

三.Form入口实现

```
import _FormItem from './src/form-item.vue'
import _Form from './src/form.vue'
import { withInstall } from '@zi-
shui/utils/withInstall'

export const FormItem = withInstall(_FormItem)
export const Form = withInstall(_Form)

export type FormItemInstance =
InstanceType<typeof FormItem>
export type FormInstance = InstanceType<typeof
Form>
```

四,FormItem组件功能实现

1).根据表单属性提示错误

```
// 注入表单上下文
const formContext = inject(formContextKey)
const shouldShowError = computed(() => {
  // 当状态为失败并且需要显示错误消息时
  return (
    validateState.value === 'error' &&
    props.showMessage &&
    formContext?.showMessage
  )
})
```

2).合并表单规则

```
// 对规则进行格式化
const convertArray = (rules:
Arrayable<FormItemRule> | undefined) => {
  return rules ? (Array.isArray(rules) ? rules :
[rules]) : []
}
const _rules = computed(() => {
  const rules: FormItemRule[] =
convertArray(props.rules) // 转化自己的规则
  const formRules = formContext?.rules // 转化表
单的规则
  if (formRules && props.prop) {
    const _rules = formRules[props.prop]
```

```
    if (_rules) {
      // 取出表单规则 +
      rules.push(...convertArray(_rules))
    }
  }
  return rules
})
```

```
// form-item校验方法
const onValidationSucceeded = (state:
FormItemValidateState) => {
  validateState.value = state
}
const onValidationFailed = (err: Values) => {
  const { errors } = err
  validateState.value = !errors ? 'success' :
'error'
  validateMessage.value = errors ?
errors[0].message : ''
}
const validate: FormItemContext['validate'] =
async trigger => {
  const rules = getFilteredRule(trigger)
  const modelName = String(props.prop || '')
  const validator = new AsyncValidator({
    // 创造校验的骨架
    [modelName]: rules
```

```
  })
  const model = formContext?.model || ''
  return validator
    .validate({
      // 进行检验
      [modelName]: model[modelName]
    })
    .then(() => {
      onValidationSucceeded('success')
    })
    .catch((err: Values) => {
      onValidationFailed(err)
      return Promise.reject(err.fields)
    })
}
```

五.Form组件功能实现

> 声明`validate`事件，传递给`Formitem`中,这样在校验时子组件可以触发父组件的方法

```ts
export const formEmits = {
  validate: (prop: string, isValid: boolean,
message: string) =>
    typeof isValid === 'boolean' &&
isString(message) && isString(prop)
}
export type FormEmits = typeof formEmits
export type FormContext = FormProps & {
  emit: SetupContext<FormEmits>['emit']
  addField: (field: FormItemContext) => void
}
// 表单上下文key，包含所有props，暴露自己的方法
export const formContextKey:
InjectionKey<FormContext> =
  Symbol('formContextKey')
```

```ts
const emit = defineEmits(formEmits)
const fields: FormItemContext[] = []

const addField: FormContext['addField'] = field
=> {
  fields.push(field)
}

provide(
  formContextKey,
  reactive({
```

```
    ...toRefs(props),
    emit, // 可以监控所有formItem的错误信息
    addField
  })
)
```

1).FormItem共享上下文

```
onMounted(() => {
  if (props.prop) {
    // 如果有属性，就让其收集上下文
    formContext?.addField(context)
  }
})
```

2).触发父级检测方法

```
const onValidationSucceeded = (state:
FormItemValidateState) => {
  validateState.value = state
  formContext?.emit('validate', props.prop!,
true, '')
}
const onValidationFailed = (err: Values) => {
  const { errors } = err
  validateState.value = !errors ? 'success' :
'error'
  validateMessage.value = errors ?
errors[0].message : ''
  formContext?.emit('validate', props.prop!,
false, validateMessage.value)
}
```

3).暴露检测方法

```
// 为了使用ref可以调用validate方法
const validate = async (
  callback?: (valid: boolean, fields?: Values)
=> void
) => {
  let errors: Values = {}
  for (const field of fields) {
    try {
```

```
      await field.validate('')
    } catch (fields) {
      errors = {
        ...errors,
        ...(fields as Values)
      }
    }
  }
  if (Object.keys(errors).length === 0) return
callback?.(true)
  callback?.(false, errors)
}
defineExpose({
  validate
})
```

使用表单组件

```
<script setup lang="ts">
import { FormInstance } from '@zi-
shui/components/form'
import { Values } from 'async-validator'
import { reactive, ref } from 'vue'
const state = reactive({ username: '', password:
'' })
const form = ref<FormInstance>()
```

```
const submitForm = (form: FormInstance | undefined) => {
  if (!form) return
  form.validate((isVlalid: boolean, fields?: Values) => {
    console.log(isVlalid, fields)
  })
}
const validate = (prop: string, isValid: boolean, nesssage: string) => {
  console.log(prop, isValid, nesssage)
}
</script>
<template>
  <z-form ref="form" :show-message="true" :model="state" @validate="validate">
    <z-form-item
      label="用户名"
      prop="username"
      :rules="[
        { required: true, message: '用户名必须填写', trigger: 'blur' },
        { min: 6, message: '用户名最少6位', trigger: 'blur' }
      ]"
    >
```

```
      <z-input v-model="state.username"></z-
input>
    </z-form-item>

    <z-form-item
      label="密码"
      prop="password"
      :rules="[
        { required: true, message: '密码必须填写',
trigger: 'blur' },
        { min: 6, message: '密码最少6位', trigger:
'blur' }
      ]"
    >
      <z-input v-model="state.password"></z-
input>
    </z-form-item>
    <z-form-item>
      <z-button @click.stop="submitForm(form)">
提交</z-button>
    </z-form-item>
  </z-form>
</template>
```