## 创建项目

命令行工具 npm i -g @vue/cli version 3.x
创建 vue create vue-test
运行 npm run serve
创建 npm run build

## 数据绑定

插值绑定

```
{{title}}
```

属性绑定

```
<p :title="title">
```

## 条件和循环

```
v-if

v-for="item in items"
```

## 用户输入

```
<input type="text" v-model="text">
<button @click="addGood">添加商品</button>

 methods: {
    addGood() {
      if (this.text) {
        this.goods.push({
          id: this.goods.length + 1,
          text: this.text,
          price: 1000
        });
        this.text = "";
      }
    }
  }
```

组件通信

父->子组件 props

```
// 父组件
<cart title="title"/>

// 子组件
{
    props: ['title']
}
```

子->父 $emit

```
// Cart.vue
<button @click="$emit('add', c)">+</button>

// App.vue
<Cart ... @add="cartAdd($event)"></Cart>
cartAdd(item) {
    console.log(item.text + "加一个");
}
```

动态样式

```
<tr ... :class="{inactive:!c.active}">
```

计算属性

```
computed: {
   total() {
     //计算激活项总价
     return this.cart.reduce((sum, c) => {
       if (c.active) {
         sum += c.price * c.count;
       }
       return sum;
     }, 0);
   }
 }
```

监听数据变化 watch

```
data() {
    return {
        cart: JSON.parse(localStorage.getItem("cart")) || []
    };
},
watch: {
    cart(n) {
       localStorage.setItem("cart", JSON.stringify(n));
    }
}
```

深度监听

```
watch: {
    cart: {
        handler(n) {
           localStorage.setItem("cart", JSON.stringify(n));
        },
        deep: true
    }
}
```

组件设计：实现Form、FormItem、Input

Input

```
<template>
  <div>
    <input :type="type" :value="value" @input="onInput">
  </div>
</template>

<script>
export default {
  props: {
    type: {
      type: String,
      default: "text"
    },
    value: {
      type: String,
      default: ""
    }
  },
  methods: {
    onInput(e) {
      // 通知老爹发生了input事件
      this.$emit("input", e.target.value);
      // 通知FormItem校验
      this.$parent.$emit("validate");
    }
  }
};
</script>
```

FormItem

```
<template>
  <div>
    <label v-if="label">{{label}}</label>
    <!-- 插槽 -->
    <slot></slot>
    <!-- 校验错误信息 -->
    <p v-if="errorMessage" class="error">{{errorMessage}}</p>
  </div>
</template>

<script>
import Validator from "async-validator";
import { Promise } from "q";

export default {
  inject: ["form"],
  props: ["label", "prop"],
  data() {
    return {
      errorMessage: ""
    };
  },
  created() {
    this.$on("validate", this.validate);
  },
  methods: {
    validate() {
      return new Promise(resolve => {
        // 校验规则制定
        const descriptor = { [this.prop]: this.form.rules[this.prop] };
        // 创建校验器
        const validator = new Validator(descriptor);
        // 执行校验
        validator.validate(
          { [this.prop]: this.form.model[this.prop] },
          errors => {
            if (errors) {
              // 显示错误信息
              this.errorMessage = errors[0].message;
              resolve(false);
            } else {
              this.errorMessage = "";
              resolve(true);
          }
        }
      );
    });
  }
}
};
</script>

<style scoped>
.error {
  color: red;
}
</style>
```

Form

```
<template>
  <div>
    <slot></slot>
  </div>
</template>

<script>
export default {
  // provide返回对象可以跨层级传参给子孙
  provide() {
    return {
      form: this // 表单实例传递给后代
    };
  },
  props: {
    model: { type: Object, required: true },
    rules: { type: Object }
  },
  methods: {
    async validate(cb) {
      // 执行表单中所有表单项校验
      const tasks = this.$children
        .filter(item => item.prop)
        .map(item => item.validate());
      // tasks中任意false就校验失败
      const results = await Promise.all(tasks);
      if (results.some(valid => !valid)) {
        // 校验失败
        cb(false);
      } else {
        cb(true);
      }
    }
  }
};
</script>

<style lang="scss" scoped>
</style>
```