生命周期：

beforeCreate 组件实例刚刚被创建,属性都没有

created 实例已经创建完成，属性已经绑定

beforeMount 模板编译之前

mounted 模板编译之后，代替之前 ready *

beforeUpdate 组件更新之前

updated 组件更新完毕 *

beforeDestroy 组件销毁前

destroyed 组件销毁后

组件通信：

vm.$emit()

vm.$on();

父组件和子组件:

子组件想要拿到父组件数据:

通过 props

之前，子组件可以更改父组件信息，可以是同步 sync

现在，不允许直接给父级的数据，做赋值操作

Event.$emit(事件名称, 数据)

Event.$on(事件名称,function(data){

//data

}.bind(this));

计算属性：

计算属性在处理一些复杂逻辑时是很有用的。

我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

```
var vm = new Vue({ el: '#app', data: { message: 'Runoob!' }, computed: { //
计算属性的 getter reversedMessage: function () { // `this` 指向 vm 实例
return this.message.split('').reverse().join('') } } })
```

```
methods: { reversedMessage2: function () { return
this.message.split('').reverse().join('') } }
```

computed 属性默认只有 getter ，不过在需要时你也可以提供一个 setter

1. 布局

```
<router-link to="/home">主页</router-link>


<router-view></router-view>
```

2. 路由具体写法

```
//组件
var Home={
    template:'<h3>我是主页</h3>'
};
var News={
    template:'<h3>我是新闻</h3>'
};

//配置路由
const routes=[
    {path:'/home', componet:Home},
    {path:'/news', componet:News},
];

//生成路由实例
const router=new VueRouter({
    routes
```

```
});

//最后挂到 vue 上
new Vue({
    router,
    el:'#box'
});
```

3. 重定向

之前 router.rediect 废弃了

```
{path:'*', redirect:'/home'}
```

路由嵌套:

/user/username

```
const routes=[
    {path:'/home', component:Home},
    {
        path:'/user',
        component:User,
        children:[  //核心
            {path:'username', component:UserDetail}
        ]
    },
    {path:'*', redirect:'/home'}  //404
];
```

/user/strive/age/10

:id

:username

:age

路由实例方法:

router.push({path:'home'}); //直接添加一个路由,表现切换路由，本质往历史记录里面添加一个

router.replace({path:'news'}) //替换路由，不会往历史记录里面添加

axios

```
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

执行多个并发请求

```
function getUserAccount() {
  return axios.get('/user/12345');
}

function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}

axios.all([getUserAccount(), getUserPermissions()])
  .then(axios.spread(function (acct, perms) {
    // 两个请求现在都执行完成
  }));
```

Ajax：

```
if(window.XMLHttpRequest){
        var xhr=new XMLHttpRequest();
}else{
        var xhr=new ActiveXObject("Microsoft.XMLHTTP");
};
```

open(method, url, asyn)

xhr.send(); 将请求发送到服务器(get 请求)

xhr.send(string)：仅用于 post 请求

onreadystatechange 事件

当 readyState 为 4 且 status 为 200 时，表示响应已就绪

Jqajax：

```
$.ajax({
   url: "发送的请求地址",
   type:"请求方式",
   data:"要发送的数据",
   dataType: "服务器返回的数据类型",
   async:boolean
});
```