四、首页—文章列表

页面布局

头部导航栏

1、使用导航栏组件

2、在导航栏组件中插入按钮

- 文本
- 图标

3、样式调整

- 宽高
- 背景色
- 边框
- 文本大小
- 图标大小

```
<template>
  <div class="home-container">
    <!-- 导航栏 -->
    <van-nav-bar class="page-nav-bar">
      <van-button
        class="search-btn"
        slot="title"
        type="info"
```

```
        size="small"
        round
        icon="search"
      >搜索</van-button>
    </van-nav-bar>
    <!-- /导航栏 -->
  </div>
</template>

<script>
export default {
  name: 'HomeIndex',
  components: {},
  props: {},
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

<style scoped lang="less">
.home-container {
  .van-nav-bar__title {
    max-width: unset;
  }
  .search-btn {
    width: 555px;
    height: 64px;
    background-color: #5babfb;
    border: none;
    font-size: 28px;
    .van-icon {
      font-size: 32px;
    }
  }
}
</style>
```

频道列表

使用 Tab 标签页组件

参考：Tab 标签页组件

样式调整

（1）基础样式调整

- 标签项
  - 右边框
  - 下边框
  - 宽高
  - 文字大小
  - 文字颜色
- 底部条
  - 宽高
  - 颜色
  - 位置

（2）处理汉堡按钮

1、使用插槽插入内容

2、样式调整

- 定位
- 内容居中
- 宽高
- 背景色、透明度
- 字体图标大小

3、使用伪元素设置渐变边框

- 定位
- 宽高
- 背景图
- 背景图填充模式

4、添加占位符充当内容区域

```
    通过 swipeable 属性可以开启滑动切换标签页
    -->
    <van-tabs class="channel-tabs" v-model="active" animated swipeable>
      <van-tab title="标签 1">内容 1</van-tab>
      <van-tab title="标签 2">内容 2</van-tab>
      <van-tab title="标签 3">内容 3</van-tab>
      <van-tab title="标签 4">内容 4</van-tab>
      <van-tab title="标签 4">内容 4</van-tab>
      <van-tab title="标签 4">内容 4</van-tab>
      <div slot="nav-right" class="placeholder"></div>
      <div slot="nav-right" class="hamburger-btn">
        <i class="toutiao toutiao-gengduo"></i>
      </div>
    </van-tabs>
    <!-- /频道列表 -->
  </div>
</template>

<script>
```

CSS 样式：

```
.placeholder {
  flex-shrink: 0;
  width: 66px;
  height: 82px;
}

.hamburger-btn {
  position: fixed;
  right: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  width: 66px;
  height: 82px;
  background-color: #fff;
```

```
  opacity: 0.902;
  i.toutiao {
    font-size: 33px;
  }
  &:before {
    content: "";
    position: absolute;
    left: 0;
    width: 1px;
    height: 100%;
    background-image: url(~@/assets/gradient-gray-line.png);
    background-size: contain;
  }
}
```

展示频道列表

思路:

1. 找数据接口
2. 把接口封装为请求方法
3. 在组件中请求获取数据
4. 模板绑定

1、封装数据请求接口

```
/**
 * 获取用户自己的信息
 */
export const getUserChannels = () => {
  return request({
    method: 'GET',
    url: '/app/v1_0/user/channels'
  })
}
```

2、请求获取数据

index.vue

```vue
<script>
import { getUserChannels } from '@/api/user'

export default {
  name: 'HomeIndex',
  components: {},
  props: {},
  data () {
    return {
      active: 0, // 控制被激活的标签项，其实就是索引
      channels: [] // 频道列表
    }
  },
  computed: {},
  watch: {},
  created () {
    this.loadChannels()
  },
  mounted () {},
  methods: {
    async loadChannels () {
      try {
        const { data } = await getUserChannels()
        this.channels = data.data.channels
      } catch (err) {
        this.$toast('获取用户频道失败')
      }
    }
  }
```

3、模板绑定

```vue
<!-- 文章频道列表 -->
<van-tabs v-model="active" animated swipeable>
  <van-tab
    :title="channel.name"
    v-for="channel in channels"
    :key="channel.id"
  >{{ channel.name }} 的内容</van-tab>
</van-tabs>
<!-- /文章频道列表 -->
```

文章列表

思路分析

你的思路可能是这样的：

1、找到数据接口

2、封装请求方法

3、在组件中请求获取数据，将数据存储到 data 中

4、模板绑定展示

根据不同的频道加载不同的文章列表，你的思路可能是这样的：

- 有一个 `list` 数组，用来存储文章列表
- 查看 `a` 频道：请求获取数据，让 `list = a` 频道文章
- 查看 `b` 频道：请求获取数据，让 `list = b` 频道文章
- 查看 `c` 频道：请求获取数据，让 `list = c` 频道文章
- ...

list: [列表数据]

查看 a 频道：请求获取数据，list = a 频道文章

查看 b 频道：请求获取数据，list = b 频道文章

查看 c 频道：请求获取数据，list = c 频道文章

思路没有问题，但是并不是我们想要的效果。

我们想要的效果是：**加载过的数据列表不要重新加载。**

实现思路也非常简单，就是我们准备**多个 list 数组，每个频道对应一个，查看哪个频道就把数据往哪个频道的列表数组中存放，这样的话就不会导致覆盖问题。**

可是有多少频道就得有多少频道文章数组，我们都一个一个声明的话会非常麻烦，所以这里的建议是利用组件来处理。

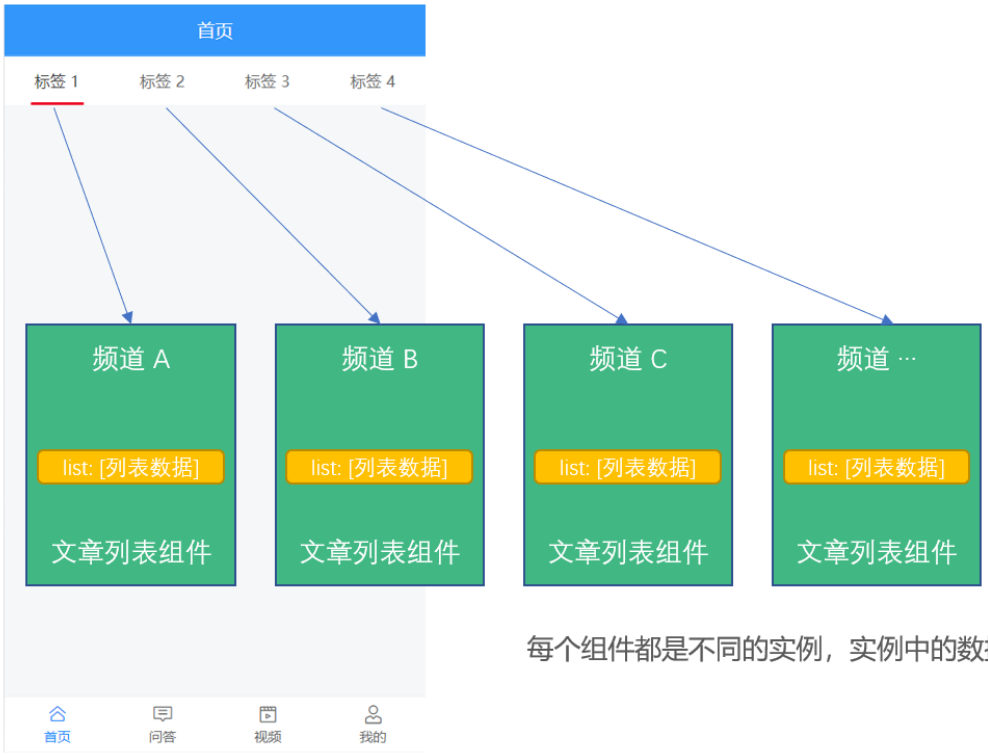
具体做法就是：

- 封装一个文章列表组件
- 然后在频道列表中把文章列表遍历出来

因为文章列表组件中请求获取文章列表数据需要频道 id，所以 频道 id 应该作为 props 参数传递给文章列表组件，为了方便，我们直接把频道对象传递给文章列表组件就可以了。

在文章列表中请求获取对应的列表数据，展示到列表中。

最后把组件在频道列表中遍历出来，就像下面这样。



每个组件都是不同的实例，实例中的数据都是独立的

1、创建 `src/views/home/components/article-list.vue`

```
<template>
  <div class="article-list">文章列表</div>
</template>

<script>
```

```vue
export default {
  name: 'ArticleList',
  components: {},
  props: {
    channel: {
      type: Object,
      required: true
    }
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

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

2、在 `home/index.vue` 中注册使用

3、最后测试。

答疑：

- 为什么标签内容是懒渲染的？
  - 因为这是 Tab 标签页组件本身支持的默认功能，如果不需要可以通过配置 `:lazy-render="false"` 来关闭这个效果。

使用 List 列表组件

List 列表组件：瀑布流滚动加载，用于展示长列表。

List 组件通过 loading 和 finished 两个变量控制加载状态，
当组件初始化或滚动到到底部时，会触发 load 事件并将 loading 设置成 true，此时可以发起异步操作并更新数据，数据更新完毕后，将 loading 设置成 false 即可。
若数据已全部加载完毕，则直接将 finished 设置成 true 即可。

- `load 事件`：
  - List 初始化后会触发一次 load 事件，用于加载第一屏的数据。
  - 如果一次请求加载的数据条数较少，导致列表内容无法铺满当前屏幕，List 会继续触发 load 事件，直到内容铺满屏幕或数据全部加载完成。
- `loading 属性`：控制加载中的 loading 状态
  - 非加载中，loading 为 false，此时会根据列表滚动位置判断是否触发 load 事件（列表内容不足一屏幕时，会直接触发）
  - 加载中，loading 为 true，表示正在发送异步请求，此时不会触发 load 事件
- `finished 属性`：控制加载结束的状态
  - 在每次请求完毕后，需要手动将 loading 设置为 false，表示本次加载结束
  - 所有数据加载结束，finished 为 true，此时不会触发 load 事件

```
<template>
  <div class="article-list">
    <!--
      List 列表组件：瀑布流滚动加载，用于展示长列表。

      List 组件通过 loading 和 finished 两个变量控制加载状态，
      当组件初始化或滚动到到底部时，会触发 load 事件并将 loading 自动设置成 true，此时可以发起异步操作并更新数据，
      数据更新完毕后，将 loading 设置成 false 即可。
      若数据已全部加载完毕，则直接将 finished 设置成 true 即可。

      - load 事件：
        + List 初始化后会触发一次 load 事件，用于加载第一屏的数据。
        + 如果一次请求加载的数据条数较少，导致列表内容无法铺满当前屏幕，List 会继续触发 load 事件，直到内容铺满屏幕或数据全部加载完成。

      - loading 属性：控制加载中的 loading 状态
        + 非加载中，loading 为 false，此时会根据列表滚动位置判断是否触发 load 事件（列表内容不足一屏幕时，会直接触发）
        + 加载中，loading 为 true，表示正在发送异步请求，此时不会触发 load 事件

      - finished 属性：控制加载结束的状态
        + 在每次请求完毕后，需要手动将 loading 设置为 false，表示本次加载结束
        + 所有数据加载结束，finished 为 true，此时不会触发 load 事件
```

```
      -->
    <van-list
      v-model="loading"
      :finished="finished"
      finished-text="没有更多了"
      @load="onLoad"
    >
      <van-cell v-for="item in list" :key="item" :title="item" />
    </van-list>
  </div>
</template>

<script>
export default {
  name: 'ArticleList',
  components: {},
  props: {
    channel: {
      type: Object,
      required: true
    }
  },
  data () {
    return {
      list: [], // 存储列表数据的数组
      loading: false, // 控制加载中 loading 状态
      finished: false // 控制数据加载结束的状态
    }
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {
    // 初始化或滚动到底部的时候会触发调用 onLoad
    onLoad () {
      console.log('onLoad')
      // 1. 请求获取数据
      // setTimeout 仅做示例，真实场景中一般为 ajax 请求
      setTimeout(() => {
        // 2. 把请求结果数据放到 list 数组中
        for (let i = 0; i < 10; i++) {
          // 0 + 1 = 1
          // 1 + 1 = 2
          // 2 + 1 = 3
          this.list.push(this.list.length + 1)
        }

        // 3. 本次数据加载结束之后要把加载状态设置为结束
        //     loading 关闭以后才能触发下一次的加载更多
        this.loading = false

        // 4. 判断数据是否全部加载完成
        if (this.list.length >= 40) {
          // 如果没有数据了，把 finished 设置为 true，之后不再触发加载更多
          this.finished = true
        }
      }, 1000)
```

```
    }
  }
}
</script>

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

让列表固定定位

```
.article-list {
  position: fixed;
  top: 180px;
  bottom: 100px;
  right: 0;
  left: 0;
    overflow-y: auto;
}
```

加载文章列表数据

实现思路：

- 找到数据接口
- 封装请求方法
- 请求获取数据
- 模板绑定

1、创建 `src/api/article.js` 封装获取文章列表数据的接口

```
/**
 * 文章接口模块
 */
import request from '@/utils/request'

/**
 * 获取频道的文章列表
 */
export const getArticles = params => {
  return request({
    method: 'GET',
    url: '/app/v1_1/articles',
    params
  })
}
```

> 注意：使用接口文档中最下面的 **频道新闻推荐_V1.1**

2、然后在首页文章列表组件 `onload` 的时候请求加载文章列表

```
<template>
  <div class="article-list">
      <!--
        loading 控制上拉加载更多的 loading 状态
```

```
        finished 控制数据是否加载结束
        load 事件：当触发上拉加载更多的时候会触发调用 load 事件

        List 初始化后会触发一次 load 事件，用于加载第一屏的数据
        如果一次请求加载的数据条数较少，导致列表内容无法铺满当前屏幕，List 会继续触发 load 事件，直到内容铺满屏幕或数据全部加载完成
      -->
      <van-list
        v-model="loading"
        :finished="finished"
        finished-text="没有更多了"
        :error.sync="error"
        error-text="请求失败，点击重新加载"
        @load="onLoad"
      >
        <van-cell
          v-for="(article, index) in list"
          :key="index"
          :title="article.title"
        />
      </van-list>
  </div>
</template>

<script>
import { getArticles } from '@/api/article'

export default {
  name: 'ArticleList',
  components: {},
  props: {
    channel: {
      type: Object,
      required: true
    }
  },
  data () {
    return {
      list: [], // 文章列表数据
      loading: false, // 上拉加载更多的 loading 状态
      finished: false, // 是否加载结束
      error: false, // 是否加载失败
      timestamp: null // 请求下一页数据的时间戳
    }
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {
    // 当触发上拉加载更多的时候调用该函数
    async onLoad () {
      try {
        // 1. 请求获取数据
        const { data } = await getArticles({
          channel_id: this.channel.id, // 频道 id
          timestamp: this.timestamp || Date.now(), // 时间戳，请求新的推荐数据传当前的时间戳，请求历史推荐传指定的时间戳
```

```
          with_top: 1 // 是否包含置顶，进入页面第一次请求时要包含置顶文章，1-包含置顶，0-不包含
        })

        // 2. 把数据添加到 list 数组中
        const { results } = data.data
        this.list.push(...results)

        // 3. 设置本次加载中 loading 状态结束
        this.loading = false

        // 4. 判断数据是否加载结束
        if (results.length) {
          // 更新获取下一页数据的时间戳
          this.timestamp = data.data.pre_timestamp
        } else {
          // 没有数据了，设置加载状态结束，不再触发上拉加载更多了
          this.finished = true
        }
      } catch (err) {
        console.log(err)
        this.loading = false // 关闭 loading 效果
        this.error = true // 开启错误提示
      }
    }
  }
}
</script>

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

最后测试。

下拉刷新

这里主要使用到 Vant 中的 PullRefresh 下拉刷新 组件。

思路：

- 注册下拉刷新事件（组件）的处理函数
- 发送请求获取文章列表数据
- 把获取到的数据添加到当前频道的文章列表的顶部
- 提示用户刷新成功！

下拉刷新时会触发组件的 `refresh` 事件，在事件的回调函数中可以进行同步或异步操作，操作完成后将 `v-model` 设置为 `false`，表示加载完成。

```
// 当触发下拉刷新的时候调用该函数
async onRefresh () {
  try {
    // 1. 请求获取数据
    const { data } = await getArticles({
      channel_id: this.channel.id, // 频道 id
      timestamp: Date.now(), // 下拉刷新每次都应该获取最新数据
      with_top: 1 // 是否包含置顶，进入页面第一次请求时要包含置顶文章，1-包含置顶，0-不包含
    })

    // 2. 将数据追加到列表的顶部
    const { results } = data.data
    this.list.unshift(...results)

    // 3. 关闭下拉刷新的 loading 状态
    this.isRefreshLoading = false

    // 提示成功
    this.refreshSuccessText = `刷新成功，更新了${results.length}条数据`
  } catch (err) {
```

```
    console.log(err)
    this.isRefreshLoading = false // 关闭下拉刷新的 loading 状态
    this.$toast('刷新失败')
  }
}
```

文章列表项

准备组件

在我们项目中有好几个页面中都有这个文章列表项内容，如果我们在每个页面中都写一次的话不仅效率低而且维护起来也麻烦。所以最好的办法就是我们把它封装为一个组件，然后在需要使用的组件中加载使用即可。

1、创建 `src/components/article-item/index.vue` 组件

```
<template>
  <div class="article-item">文章列表项</div>
</template>

<script>
export default {
  name: 'ArticleItem',
  components: {},
  props: {
    article: {
      type: Object,
      required: true
    }
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

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

2、在文章列表组件中注册使用文章列表项组件

```
          @load="onLoad"
        >
          <article-item
            v-for="(article, index) in list"
            :key="index"
            :article="article"
          />
        </van-list>
      </van-pull-refresh>
    </div>
  </template>

  <script>
  import { getArticles } from '@/api/article'
  import ArticleItem from '@/components/article-item'

  export default {
    name: 'ArticleList',
    components: {
      ArticleItem
    },
    props: {
      channel: {
```

展示列表项内容

- 使用 Cell 单元格组件
- 展示标题
- 展示底部信息

```
<template>
  <van-cell
    class="article-item"
  >
    <div slot="title" class="title">{{ article.title }}</div>
    <div slot="label">
      <div v-if="article.cover.type === 3" class="cover-wrap">
        <div
          class="cover-item"
          v-for="(img, index) in article.cover.images"
          :key="index"
        >
          <van-image
            width="100"
            height="100"
            :src="img"
          />
        </div>
      </div>
      <div>
        <span>{{ article.aut_name }}</span>
        <span>{{ article.comm_count }}评论</span>
        <span>{{ article.pubdate }}</span>
      </div>
    </div>
    <van-image
```

```
      v-if="article.cover.type === 1"
      slot="default"
      width="100"
      height="100"
      :src="article.cover.images[0]"
    />
  </van-cell>
</template>

<script>
export default {
  name: 'ArticleItem',
  components: {},
  props: {
    article: {
      type: Object,
      required: true
    }
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

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

样式调整

- 文章标题
  - 字号
  - 颜色
  - 多行文字省略
- 单图封面
  - 封面容器
    - 去除 flex: 1，固定宽高
    - 左内边距
  - 封面图
    - 宽高
    - 填充模式：cover
- 底部文本信息
  - 字号
  - 颜色
  - 间距
- 多图封面
  - 外层容器
    - flex 容器

        - 上下外边距
    - 图片容器
        - 平均分配容器空间：flex: 1;
        - 固定高度
        - 容器项间距
    - 图片
        - 宽高
        - 填充模式

以下代码仅供参考。

```
<template>
  <van-cell
    class="article-item"
  >
    <div slot="title" class="title van-multi-ellipsis--l2">{{ article.title }}
</div>
    <div slot="label">
      <div v-if="article.cover.type === 3" class="cover-wrap">
        <div
          class="cover-item"
          v-for="(img, index) in article.cover.images"
          :key="index"
        >
          <van-image
            class="cover-item-img"
            fit="cover"
            :src="img"
          />
        </div>
      </div>
      <div class="label-info-wrap">
        <span>{{ article.aut_name }}</span>
        <span>{{ article.comm_count }}评论</span>
        <span>{{ article.pubdate }}</span>
      </div>
    </div>
    <van-image
      v-if="article.cover.type === 1"
      slot="default"
      class="right-cover"
      fit="cover"
      :src="article.cover.images[0]"
    />
  </van-cell>
</template>

<script>
export default {
  name: 'ArticleItem',
  components: {},
  props: {
    article: {
      type: Object,
      required: true
    }
```

```
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

<style scoped lang="less">
.article-item {
  .title {
    font-size: 32px;
    color: #3a3a3a;
  }

  .van-cell__value {
    flex: unset;
    width: 232px;
    height: 146px;
    padding-left: 25px;
  }

  .right-cover {
    width: 232px;
    height: 146px;
  }

  .label-info-wrap span {
    font-size: 22px;
    color: #b4b4b4;
    margin-right: 25px;
  }

  .cover-wrap {
    display: flex;
    padding: 30px 0;
    .cover-item {
      flex: 1;
      height: 146px;
      &:not(:last-child) {
        padding-right: 4px;
      }
      .cover-item-img {
        width: 100%;
        height: 146px;
      }
    }
  }
}
</style>
```

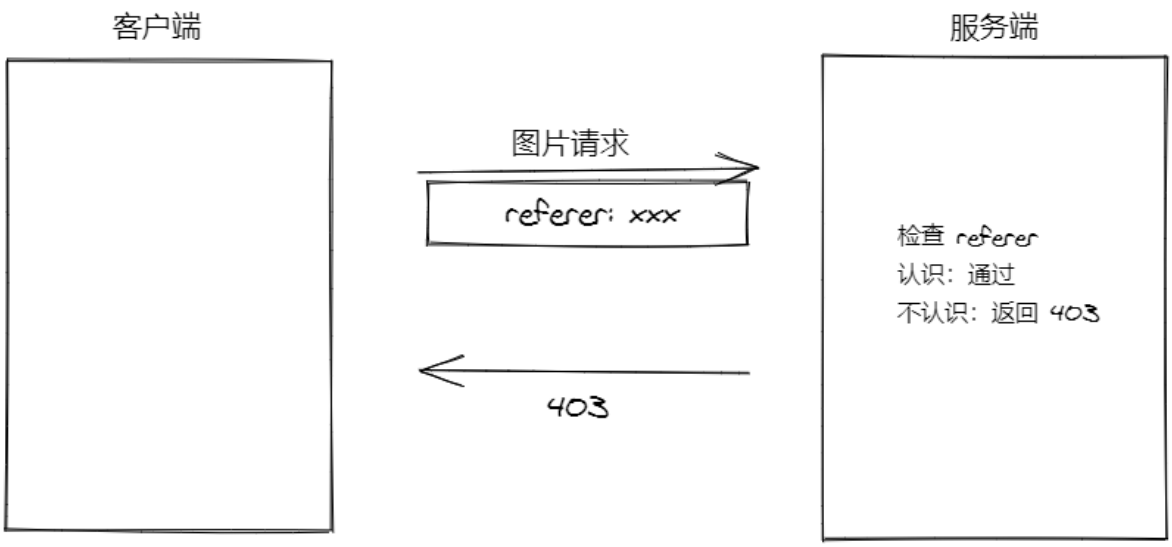
关于第三方图片资源403问题

为什么文章列表数据中的好多图片资源请求失败返回 403?

> 这是因为我们项目的接口数据是后端通过爬虫抓取的第三方平台内容，而第三方平台对图片资源做了防盗链保护处理。

第三方平台怎么处理图片资源保护的?

> 服务端一般使用 `Referer` 请求头识别访问来源，然后处理资源访问。



Referer 是什么东西?

> 扩展参考：http://www.ruanyifeng.com/blog/2019/06/http-referer.html

> `Referer` 是 `HTTP` 请求头的一部分，当浏览器向 `Web` 服务器发送请求的时候，一般会带上 `Referer`，它包含了当前请求资源的来源页面的地址。服务端一般使用 `Referer` 请求头识别访问来源，可能会以此进行统计分析、日志记录以及缓存优化等。

> 需要注意的是 referer 实际上是 "referrer" 误拼写。参见 HTTP referer on Wikipedia （HTTP referer 在维基百科上的条目）来获取更详细的信息。

怎么解决?

> 不要发送 `referrer` ，对方服务端就不知道你从哪来的了，姑且认为是你是自己人吧。

如何设置不发送 referrer?

用 `<a>`、`<area>`、`<img>`、`<iframe>`、`<script>` 或者 `<link>` 元素上的 `referrerpolicy` 属性为其设置独立的请求策略，例如：

```
<img src="http://……" referrerPolicy="no-referrer">
```

或者直接在 HTMl 页面头中通过 meta 属性全局配置：

```
<meta name="referrer" content="no-referrer" />
```

处理相对时间

推荐两个第三方库：

- Moment.js
- Day.js

两者都是专门用于处理时间的 JavaScript 库，功能差不多，因为 Day.js 的设计就是参考的 Moment.js。但是 Day.js 相比 Moment.js 的包体积要更小一些，因为它采用了插件化的处理方式。

Day.js 是一个轻量的处理时间和日期的 JavaScript 库，和 Moment.js 的 API 设计保持完全一样，如果您曾经用过 Moment.js, 那么您已经知道如何使用 Day.js 。

- Day.js 可以运行在浏览器和 Node.js 中。
- 和 Moment.js 相同的 API 和用法
- 不可变数据 (Immutable)
- 支持链式操作 (Chainable)
- 国际化 I18n
- 仅 2kb 大小的微型库
- 全浏览器兼容

1、安装

```
npm i dayjs
```

2、创建 `utils/dayjs.js`

```
import Vue from 'vue'
import dayjs from 'dayjs'

// 加载中文语言包
import 'dayjs/locale/zh-cn'

import relativeTime from 'dayjs/plugin/relativeTime'

// 配置使用处理相对时间的插件
dayjs.extend(relativeTime)

// 配置使用中文语言包
dayjs.locale('zh-cn')

// 全局过滤器：处理相对时间
Vue.filter('relativeTime', value => {
  return dayjs().to(dayjs(value))
})
```

3、在 `main.js` 中加载初始化

```
import './utils/dayjs'
```

4、使用

使用过滤器：

```
<p>{{ 日期数据 | relativeTime }}</p>
```