七、文章详情

创建组件并配置路由

1、创建 `views/article/index.vue` 组件

```
<template>
  <div class="article-container">文章详情</div>
</template>

<script>
export default {
  name: 'ArticleIndex',
  components: {},
  props: {
    articleId: {
      type: [Number, String],
      required: true
    }
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

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

2、然后将该页面配置到根级路由

```
{
  path: '/article/:articleId',
  name: 'article',
  component: () => import('@/views/article'),
  // 将路由动态参数映射到组件的 props 中，更推荐这种做法
  props: true
}
```

> 官方文档：路由 props 传参

页面布局

使用到的 Vant 中的组件：

- NavBar 导航栏

- Loading 加载
- Cell 单元格
- Button 按钮
- Image 图片
- Divider 分割线
- Icon 图标

```
<template>
  <div class="article-container">
    <!-- 导航栏 -->
    <van-nav-bar
      class="page-nav-bar"
      left-arrow
      title="黑马头条"
    ></van-nav-bar>
    <!-- /导航栏 -->

    <div class="main-wrap">
      <!-- 加载中 -->
      <div class="loading-wrap">
        <van-loading
          color="#3296fa"
          vertical
        >加载中</van-loading>
      </div>
      <!-- /加载中 -->

      <!-- 加载完成-文章详情 -->
      <div class="article-detail">
        <!-- 文章标题 -->
        <h1 class="article-title">这是文章标题</h1>
        <!-- /文章标题 -->

        <!-- 用户信息 -->
        <van-cell class="user-info" center :border="false">
          <van-image
            class="avatar"
            slot="icon"
            round
            fit="cover"
            src="https://img.yzcdn.cn/vant/cat.jpeg"
          />
          <div slot="title" class="user-name">黑马头条号</div>
          <div slot="label" class="publish-date">14小时前</div>
          <van-button
            class="follow-btn"
            type="info"
            color="#3296fa"
            round
            size="small"
            icon="plus"
          >关注</van-button>
          <!-- <van-button
            class="follow-btn"
            round
            size="small"
          >已关注</van-button> -->
```

```
        </van-cell>
        <!-- /用户信息 -->

        <!-- 文章内容 -->
        <div class="article-content">这是文章内容</div>
        <van-divider>正文结束</van-divider>
      </div>
      <!-- /加载完成-文章详情 -->

      <!-- 加载失败：404 -->
      <div class="error-wrap">
        <van-icon name="failure" />
        <p class="text">该资源不存在或已删除！</p>
      </div>
      <!-- /加载失败：404 -->

      <!-- 加载失败：其它未知错误（例如网络原因或服务端异常） -->
      <div class="error-wrap">
        <van-icon name="failure" />
        <p class="text">内容加载失败！</p>
        <van-button class="retry-btn">点击重试</van-button>
      </div>
      <!-- /加载失败：其它未知错误（例如网络原因或服务端异常） -->
    </div>

    <!-- 底部区域 -->
    <div class="article-bottom">
      <van-button
        class="comment-btn"
        type="default"
        round
        size="small"
      >写评论</van-button>
      <van-icon
        name="comment-o"
        info="123"
        color="#777"
      />
      <van-icon
        color="#777"
        name="star-o"
      />
      <van-icon
        color="#777"
        name="good-job-o"
      />
      <van-icon name="share" color="#777777"></van-icon>
    </div>
    <!-- /底部区域 -->
  </div>
</template>

<script>
export default {
  name: 'ArticleIndex',
  components: {},
  props: {
    articleId: {
```

```
      type: [Number, String],
      required: true
    }
  },
  data () {
    return {}
  },
  computed: {},
  watch: {},
  created () {},
  mounted () {},
  methods: {}
}
</script>

<style scoped lang="less">
.article-container {
  .main-wrap {
    position: fixed;
    left: 0;
    right: 0;
    top: 92px;
    bottom: 88px;
    overflow-y: scroll;
    background-color: #fff;
  }
  .article-detail {
    .article-title {
      font-size: 40px;
      padding: 50px 32px;
      margin: 0;
      color: #3a3a3a;
    }

    .user-info {
      padding: 0 32px;
      .avatar {
        width: 70px;
        height: 70px;
        margin-right: 17px;
      }
      .van-cell__label {
        margin-top: 0;
      }
      .user-name {
        font-size: 24px;
        color: #3a3a3a;
      }
      .publish-date {
        font-size: 23px;
        color: #b7b7b7;
      }
      .follow-btn {
        width: 170px;
        height: 58px;
      }
    }
```

```
    .article-content {
      padding: 55px 32px;
      /deep/ p {
        text-align: justify;
      }
    }
  }

  .loading-wrap {
    padding: 200px 32px;
    display: flex;
    align-items: center;
    justify-content: center;
    background-color: #fff;
  }

  .error-wrap {
    padding: 200px 32px;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    background-color: #fff;
    .van-icon {
      font-size: 122px;
      color: #b4b4b4;
    }
    .text {
      font-size: 30px;
      color: #666666;
      margin: 33px 0 46px;
    }
    .retry-btn {
      width: 280px;
      height: 70px;
      line-height: 70px;
      border: 1px solid #c3c3c3;
      font-size: 30px;
      color: #666666;
    }
  }

  .article-bottom {
    position: fixed;
    left: 0;
    right: 0;
    bottom: 0;
    display: flex;
    justify-content: space-around;
    align-items: center;
    box-sizing: border-box;
    height: 88px;
    border-top: 1px solid #d8d8d8;
    background-color: #fff;
    .comment-btn {
      width: 282px;
      height: 46px;
      border: 2px solid #eeeeee;
```

```
      font-size: 30px;
      line-height: 46px;
      color: #a7a7a7;
    }
    .van-icon {
      font-size: 40px;
      .van-info {
        font-size: 16px;
        background-color: #e22829;
      }
    }
  }
}
</style>
```

关于后端返回数据中的大数字问题

之所以请求文章详情返回 404 是因为我们请求发送的文章 ID （article.art_id）不正确。

JavaScript 能够准确表示的整数范围在 `-2^53` 到 `2^53` 之间（不含两个端点），超过这个范围，无法精确表示这个值，这使得 JavaScript 不适合进行科学和金融方面的精确计算。

```
Math.pow(2, 53) // 9007199254740992

9007199254740992  // 9007199254740992
9007199254740993  // 9007199254740992

Math.pow(2, 53) === Math.pow(2, 53) + 1
// true
```

上面代码中，超出 2 的 53 次方之后，一个数就不精确了。
ES6 引入了 `Number.MAX_SAFE_INTEGER` 和 `Number.MIN_SAFE_INTEGER` 这两个常量，用来表示这个范围的上下限。

```
Number.MAX_SAFE_INTEGER === Math.pow(2, 53) - 1
// true
Number.MAX_SAFE_INTEGER === 9007199254740991
// true

Number.MIN_SAFE_INTEGER === -Number.MAX_SAFE_INTEGER
// true
Number.MIN_SAFE_INTEGER === -9007199254740991
// true
```

上面代码中，可以看到 JavaScript 能够精确表示的极限。

后端返回的数据一般都是 **JSON 格式的字符串。**

```
'{ "id": 9007199254740995, "name": "Jack", "age": 18 }'
```

如果这个字符不做任何处理，你能方便的获取到字符串中的指定数据吗？非常麻烦。所以我们要把它转换为 JavaScript 对象来使用就很方便了。

幸运的是 axios 为了方便我们使用数据，它会在内部使用 `JSON.parse()` 把后端返回的数据转为 JavaScript 对象。

```
// { id: 9007199254740996, name: 'Jack', age: 18 }
JSON.parse('{ "id": 9007199254740995, "name": "Jack", "age": 18 }')
```

可以看到，超出安全整数范围的 id 无法精确表示，这个问题并不是 axios 的错。

了解了什么是大整数的概念，接下来的问题是如何解决?

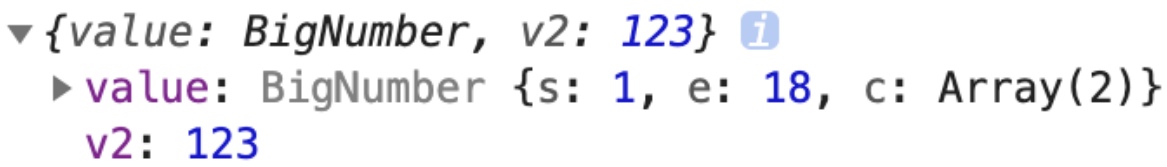
json-bigint 是一个第三方包，它可以帮我们很好的处理这个问题。

使用它的第一步就是把它安装到你的项目中。

```
npm i json-bigint
```

下面是使用它的一个简单示例。

```
const jsonStr = '{ "art_id": 1245953273786007552 }'

console.log(JSON.parse(jsonStr)) // 1245953273786007600
// JSON.stringify()

// JSONBig 可以处理数据中超出 JavaScript 安全整数范围的问题
console.log(JSONBig.parse(jsonStr)) // 把 JSON 格式的字符串转为 JavaScript 对象

// 使用的时候需要把 BigNumber 类型的数据转为字符串来使用
console.log(JSONBig.parse(jsonStr).art_id.toString()) // 1245953273786007552

console.log(JSON.stringify(JSONBig.parse(jsonStr)))

console.log(JSONBig.stringify(JSONBig.parse(jsonStr))) // 把 JavaScript 对象 转为 JSON 格式的字符串转
```



> json-bigint 会把超出 JS 安全整数范围的数字转为一个 BigNumber 类型的对象，对象数据是它内部的一个算法处理之后的，我们要做的就是在使用的时候转为字符串来使用。

通过 Axios 请求得到的数据都是 Axios 处理（JSON.parse）之后的，我们应该在 Axios 执行处理之前手动使用 json-bigint 来解析处理。Axios 提供了自定义处理原始后端返回数据的 API：`transformResponse` 。

```
import axios from 'axios'

import jsonBig from 'json-bigint'

var json = '{ "value" : 9223372036854775807, "v2": 123 }'

console.log(jsonBig.parse(json))

const request = axios.create({
```

```
  baseURL: 'http://ttapi.research.itcast.cn/', // 接口基础路径

  // transformResponse 允许自定义原始的响应数据（字符串）
  transformResponse: [function (data) {
    try {
      // 如果转换成功则返回转换的数据结果
      return jsonBig.parse(data)
    } catch (err) {
      // 如果转换失败，则包装为统一数据格式并返回
      return {
        data
      }
    }
  }]
})

export default request
```

> 扩展：ES2020 BigInt
>
> ES2020 引入了一种新的数据类型 BigInt（大整数），来解决这个问题。BigInt 只用来表示整数，没有位数的限制，任何位数的整数都可以精确表示。
>
> 参考链接：
>
> - https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/BigInt
> - http://es6.ruanyifeng.com/#docs/number#BigInt-%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B

展示文章详情

思路：

- 找到数据接口
- 封装请求方法
- 请求获取数据
- 模板绑定

一、请求并展示文章详情

1、在 `api/article.js` 中新增封装接口方法

```
/**
 * 根据 id 获取指定文章
 */
export const getArticleById = articleId => {
  return request({
    method: 'GET',
    url: `/app/v1_0/articles/${articleId}`
  })
}
```

2、在组件中调用获取文章详情

```
+ import { getArticleById } from '@/api/article'

export default {
  name: 'ArticlePage',
  components: {},
  props: {
    articleId: {
      type: String,
      required: true
    }
  },
  data () {
    return {
+      article: {} // 文章详情
    }
  },
  computed: {},
  watch: {},
  created () {
+    this.loadArticle()
  },
  mounted () {},
  methods: {
+++    async loadArticle () {
      try {
        const { data } = await getArticleById(this.articleId)
        this.article = data.data
      } catch (err) {
        console.log(err)
      }
    }
  }
}
```

3、模板绑定

处理内容加载状态

需求：

- 加载中，显示 loading
- 加载成功，显示文章详情
- 加载失败，显示错误提示
  - 如果 404，提示资源不存在
  - 其它的，提示加载失败，用户可以点击重试重新加载

关于文章正文的样式

文章正文包括各种数据：段落、标题、列表、链接、图片、视频等资源。

- 将 github-markdown-css 样式文件下载到项目中
- 配置不要转换样式文件中的字号

图片点击预览

一、ImagePreview 图片预览 的使用

二、处理图片点击预览

思路：

1、从文章内容中获取到所有的 img DOM 节点

2、获取文章内容中所有的图片地址

3、遍历所有 img 节点，给每个节点注册点击事件

4、在 img 点击事件处理函数中，调用 ImagePreview 预览

关注用户

思路：

- 给按钮注册点击事件
- 在事件处理函数中
    - 如果已关注，则取消关注
    - 如果没有关注，则添加关注

下面是具体实现。

视图处理

功能处理

- 找到数据接口
- 封装请求方法
- 请求调用
- 视图更新

1、在 `api/user.js` 中添加封装请求方法

```
/**
 * 添加关注
 */
export const addFollow = userId => {
  return request({
    method: 'POST',
    url: '/app/v1_0/user/followings',
    data: {
      target: userId
    }
  })
}
```

```
/**
 * 取消关注
 */
export const deleteFollow = userId => {
  return request({
    method: 'DELETE',
    url: `/app/v1_0/user/followings/${userId}`
  })
}
```

2、给关注/取消关注按钮注册点击事件

3、在事件处理函数中

```
import { addFollow, deleteFollow } from '@/api/user'
```

```
async onFollow () {
  // 开启按钮的 loading 状态
  this.isFollowLoading = true

  try {
    // 如果已关注，则取消关注
    const authorId = this.article.aut_id
    if (this.article.is_followed) {
      await deleteFollow(authorId)
    } else {
      // 否则添加关注
      await addFollow(authorId)
    }

    // 更新视图
    this.article.is_followed = !this.article.is_followed
  } catch (err) {
    console.log(err)
    this.$toast.fail('操作失败')
  }

  // 关闭按钮的 loading 状态
  this.isFollowLoading = false
}
```

最后测试。

loading 效果

两个作用：

- 交互反馈
- 防止网络慢用户多次点击按钮导致重复触发点击事件

组件封装

文章收藏

> 该功能和关注用户的处理思路几乎一样，建议由学员自己编写。

封装组件

处理视图

功能处理

思路：

- 给收藏按钮注册点击事件
- 如果已经收藏了，则取消收藏
- 如果没有收藏，则添加收藏

下面是具体实现。

1、在 `api/article.js` 添加封装数据接口

```
/**
 * 收藏文章
 */
export const addCollect = target => {
  return request({
    method: 'POST',
    url: '/app/v1_0/article/collections',
    data: {
      target
    }
  })
}

/**
 * 取消收藏文章
 */
export const deleteCollect = target => {
  return request({
    method: 'DELETE',
    url: `/app/v1_0/article/collections/${target}`
  })
}
```

2、给收藏按钮注册点击事件

3、处理函数

```
async onCollect () {
  // 这里 loading 不仅仅是为了交互提示，更重要的是请求期间禁用背景点击功能，防止用户不断的操作界面发出请求
  this.$toast.loading({
    duration: 0, // 持续展示 toast
    message: '操作中...',
    forbidClick: true // 是否禁止背景点击
```

```
  })

  try {
    // 如果已收藏，则取消收藏
    if (this.article.is_collected) {
      await deleteCollect(this.articleId)
      // this.article.is_collected = false
      this.$toast.success('取消收藏')
    } else {
      // 添加收藏
      await addCollect(this.articleId)
      // this.article.is_collected = true
      this.$toast.success('收藏成功')
    }
    this.article.is_collected = !this.article.is_collected
  } catch (err) {
    console.log(err)
    this.$toast.fail('操作失败')
  }
}
```

文章点赞

> 该功能和关注用户的处理思路几乎一样，建议由学员自己编写。

article 中的 `attitude` 表示用户对文章的态度

- `-1` 无态度
- `0` 不喜欢
- `1` 已点赞

思路：

- 给点赞按钮注册点击事件
- 如果已经点赞，则请求取消点赞
- 如果没有点赞，则请求点赞

1、添加封装数据接口

```
/**
 * 点赞
 */
export const addLike = articleId => {
  return request({
    method: 'POST',
    url: '/app/v1_0/article/likings',
    data: {
      target: articleId
    }
  })
}

/**
 * 取消点赞
 */
export const deleteLike = articleId => {
```

```
  return request({
    method: 'DELETE',
    url: `/app/v1_0/article/likings/${articleId}`
  })
}
```

2、给点赞按钮注册点击事件

3、处理函数

```
async onLike () {
  // 两个作用：1、交互提示 2、防止网络慢用户连续不断的点击按钮请求
  this.$toast.loading({
    duration: 0, // 持续展示 toast
    message: '操作中...',
    forbidClick: true // 是否禁止背景点击
  })

  try {
    // 如果已经点赞，则取消点赞
    if (this.article.attitude === 1) {
      await deleteLike(this.articleId)
      this.article.attitude = -1
      this.$toast.success('取消点赞')
    } else {
      // 否则添加点赞
      await addLike(this.articleId)
      this.article.attitude = 1
      this.$toast.success('点赞成功')
    }
  } catch (err) {
    console.log(err)
    this.$toast.fail('操作失败')
  }
}
```