7. 商品详情

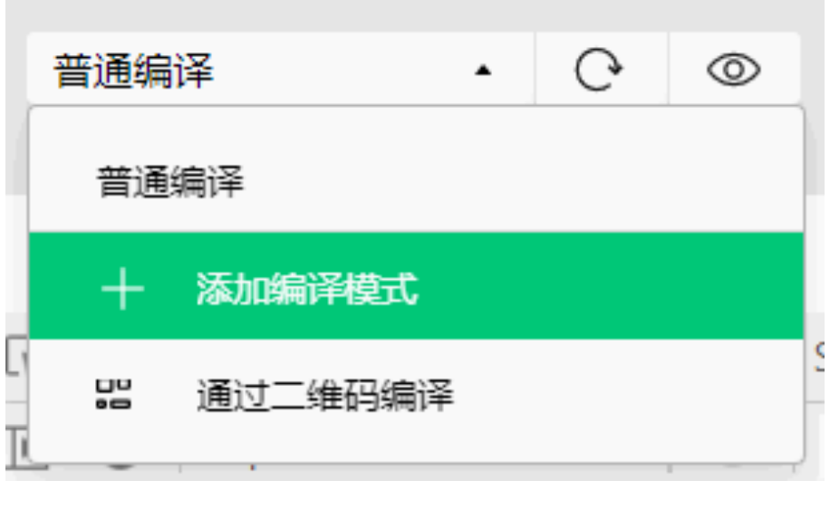
7.0 创建 goodsdetail 分支

运行如下的命令，基于 master 分支在本地创建 goodsdetail 子分支，用来开发商品详情页相关的功能：

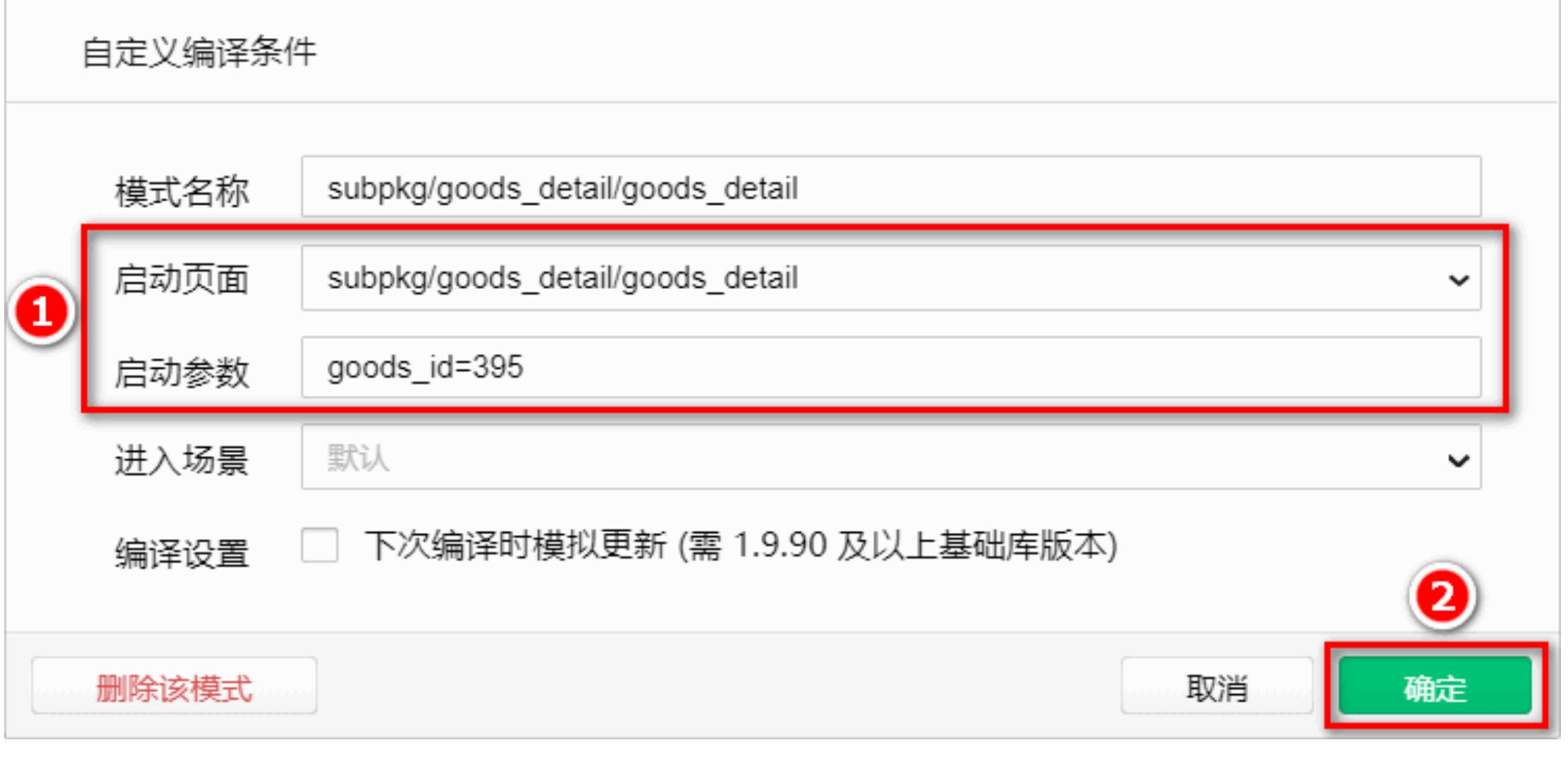
```
git checkout -b goodsdetail
```

7.1 添加商品详情页的编译模式

1. 在微信开发者工具中，点击工具栏上的编译模式下拉菜单，选择 `添加编译模式` 选项：



2. 勾选 `启动页面` 的路径，并填写了 `启动参数` 之后，点击 `确定` 按钮，添加详情页面的编译模式：


自定义编译条件

模式名称 subpkg/goods_detail/goods_detail

启动页面 subpkg/goods_detail/goods_detail

启动参数 goods_id=395

进入场景 默认

编译设置 ☐ 下次编译时模拟更新 (需 1.9.90 及以上基础库版本)

7.2 获取商品详情数据

1. 在 `data` 中定义商品详情的数据节点：

```
data() {
  return {
    // 商品详情对象
    goods_info: {}
  }
}
```

2. 在 `onLoad` 中获取商品的 Id，并调用请求商品详情的方法：

```
onLoad(options) {
  // 获取商品 Id
  const goods_id = options.goods_id
  // 调用请求商品详情数据的方法
  this.getGoodsDetail(goods_id)
}
```

3. 在 `methods` 中声明 `getGoodsDetail` 方法：

```
methods: {
  // 定义请求商品详情数据的方法
  async getGoodsDetail(goods_id) {
    const { data: res } = await uni.$http.get('/api/public/v1/goods/detail', { goods_id })
    if (res.meta.status !== 200) return uni.$showMsg()
    // 为 data 中的数据赋值
    this.goods_info = res.message
  }
}
```

7.3 渲染商品详情页的 UI 结构

7.3.1 渲染轮播图区域

1. 使用 `v-for` 指令，循环渲染如下的轮播图 UI 结构：

```
<!-- 轮播图区域 -->
<swiper :indicator-dots="true" :autoplay="true" :interval="3000" :duration="1000" :circular="true">
  <swiper-item v-for="(item, i) in goods_info.pics" :key="i">
    <image :src="item.pics_big"></image>
  </swiper-item>
</swiper>
```

2. 美化轮播图的样式：

```
swiper {
  height: 750rpx;

  image {
    width: 100%;
    height: 100%;
  }
}
```

7.3.2 实现轮播图预览效果

1. 为轮播图中的 `image` 图片绑定 `click` 事件处理函数：

```
<swiper-item v-for="(item, i) in goods_info.pics" :key="i">
  <!-- 把当前点击的图片的索引，传递到 preview() 处理函数中 -->
  <image :src="item.pics_big" @click="preview(i)"></image>
</swiper-item>
```

2. 在 `methods` 中定义 `preview` 事件处理函数：

```
// 实现轮播图的预览效果
preview(i) {
  // 调用 uni.previewImage() 方法预览图片
  uni.previewImage({
    // 预览时，默认显示图片的索引
    current: i,
    // 所有图片 url 地址的数组
    urls: this.goods_info.pics.map(x => x.pics_big)
  })
}
```

7.3.3 渲染商品信息区域

1. 定义商品信息区域的 UI 结构如下：

```
<!-- 商品信息区域 -->
<view class="goods-info-box">
  <!-- 商品价格 -->
  <view class="price">￥{{goods_info.goods_price}}</view>
  <!-- 信息主体区域 -->
  <view class="goods-info-body">
    <!-- 商品名称 -->
    <view class="goods-name">{{goods_info.goods_name}}</view>
    <!-- 收藏 -->
    <view class="favi">
      <uni-icons type="star" size="18" color="gray"></uni-icons>
      <text>收藏</text>
    </view>
  </view>
  <!-- 运费 -->
  <view class="yf">快递：免运费</view>
</view>
```

2. 美化商品信息区域的样式：

```
// 商品信息区域的样式
.goods-info-box {
  padding: 10px;
  padding-right: 0;

  .price {
    color: #c00000;
    font-size: 18px;
    margin: 10px 0;
  }

  .goods-info-body {
    display: flex;
    justify-content: space-between;

    .goods-name {
      font-size: 13px;
      padding-right: 10px;
    }
    // 收藏区域
    .favi {
      width: 120px;
      font-size: 12px;
      display: flex;
      flex-direction: column;
      justify-content: center;
      align-items: center;
      border-left: 1px solid #efefef;
      color: gray;
    }
  }

  // 运费
  .yf {
    margin: 10px 0;
    font-size: 12px;
    color: gray;
  }
}
```

7.3.4 渲染商品详情信息

1. 在页面结构中，使用 `rich-text` 组件，将带有 HTML 标签的内容，渲染为小程序的页面结构：

```
<!-- 商品详情信息 -->
<rich-text :nodes="goods_info.goods_introduce"></rich-text>
```

2. 修改 `getGoodsDetail` 方法，从而解决图片底部 `空白间隙` 的问题：

```
 // 定义请求商品详情数据的方法
async getGoodsDetail(goods_id) {

  const { data: res } = await uni.$http.get('/api/public/v1/goods/detail', { goods_id })
  if (res.meta.status !== 200) return uni.$showMsg()

  // 使用字符串的 replace() 方法，为 img 标签添加行内的 style 样式，从而解决图片底部空白间隙的问题
  res.message.goods_introduce = res.message.goods_introduce.replace(/<img /g, '<img style="display:block;"
  this.goods_info = res.message
}
```

3. 解决 `.webp` 格式图片在 `ios` 设备上无法正常显示的问题：

```
// 定义请求商品详情数据的方法
async getGoodsDetail(goods_id) {
  const { data: res } = await uni.$http.get('/api/public/v1/goods/detail', { goods_id })
  if (res.meta.status !== 200) return uni.$showMsg()

  // 使用字符串的 replace() 方法，将 webp 的后缀名替换为 jpg 的后缀名
  res.message.goods_introduce = res.message.goods_introduce.replace(/<img /g, '<img style="display:block;"
  this.goods_info = res.message
}
```

7.3.5 解决商品价格闪烁的问题

1. 导致问题的原因：在商品详情数据请求回来之前，data 中 `goods_info` 的值为 `{}`，因此初次渲染页面时，会导致 `商品价格、商品名称` 等闪烁的问题。

2. 解决方案：判断 `goods_info.goods_name` 属性的值是否存在，从而使用 `v-if` 指令控制页面的显示与隐藏：

```
<template>
  <view v-if="goods_info.goods_name">
   <!-- 省略其它代码 -->
  </view>
</template>
```

7.4 渲染详情页底部的商品导航区域

7.4.1 渲染商品导航区域的 UI 结构

> 基于 uni-ui 提供的 GoodsNav 组件来实现商品导航区域的效果

1. 在 data 中，通过 `options` 和 `buttonGroup` 两个数组，来声明商品导航组件的按钮配置对象：

```
data() {
  return {
    // 商品详情对象
    goods_info: {},
    // 左侧按钮组的配置对象
    options: [{
      icon: 'shop',
      text: '店铺'
    }, {
      icon: 'cart',
      text: '购物车',
      info: 2
    }],
    // 右侧按钮组的配置对象
    buttonGroup: [{
        text: '加入购物车',
        backgroundColor: '#ff0000',
        color: '#fff'
      },
      {
        text: '立即购买',
        backgroundColor: '#ffa200',
        color: '#fff'
      }
    ]
  }
}
```

2. 在页面中使用 `uni-goods-nav` 商品导航组件：

```
<!-- 商品导航组件 -->
<view class="goods_nav">
  <!-- fill 控制右侧按钮的样式 -->
  <!-- options 左侧按钮的配置项 -->
  <!-- buttonGroup 右侧按钮的配置项 -->
  <!-- click 左侧按钮的点击事件处理函数 -->
  <!-- buttonClick 右侧按钮的点击事件处理函数 -->
  <uni-goods-nav :fill="true" :options="options" :buttonGroup="buttonGroup" @click="onClick" @buttonClick="
</view>
```

3. 美化商品导航组件，使之固定在页面最底部：

```
.goods-detail-container {
  // 给页面外层的容器，添加 50px 的内padding，
  // 防止页面内容被底部的商品导航组件遮盖
  padding-bottom: 50px;
}

.goods_nav {
  // 为商品导航组件添加固定定位
  position: fixed;
  bottom: 0;
  left: 0;
  width: 100%;
}
```

7.4.2 点击跳转到购物车页面

1. 点击商品导航组件左侧的按钮，会触发 `uni-goods-nav` 的 `@click` 事件处理函数，事件对象 `e` 中会包含当前点击的按钮相关的信息：

```
// 左侧按钮的点击事件处理函数
onClick(e) {
  console.log(e)
}
```

打印的按钮信息对象如下：

```
▼{index: 1, content: {…}}
  ▶content: {icon: "cart", text: "购物车", info: 2}
   index: 1
  ▶__proto__: Object
```

2. 根据 `e.content.text` 的值，来决定进一步的操作：

```
// 左侧按钮的点击事件处理函数
onClick(e) {
  if (e.content.text === '购物车') {
    // 切换到购物车页面
    uni.switchTab({
      url: '/pages/cart/cart'
    })
  }
}
```