

# 豆厂的程序员如何抱AI大腿的

**Copilot + Chatgpt = 150%**

# 目录



GitHub Copilot



ChatGPT



# Github Copilot

--- happy code

# 一、Xmind model 设计到建表

```
db > migrate --help 20230527071210_create_comments.rb > ...
You: 3 小时前 [1 author (You)]
1 class CreateComments < ActiveRecord::Migration[6.1]
2 # 评论Comment
3 # attrs
4 # content
5 # state:enum
6 # pending
7 # published
8 # banned
9 # deleted_at:datetime
10 # comments_count:integer
11 # approved_comments_count:integer
12 # likes_count
13 # associations
14 # belongs_to
15 # root
16 # 根评论
17 # class_name: Comment
18 # parent
19 # 上级评论 (回复谁)
20 # class_name: Comment
21 # creator
22 # polymorphic
23 # commentable
24 # polymorphic
25 # has_many
26 # children
27 def change
28   create_table :comments do |t|
29     t.text :content
30     t.string :state
31     t.datetime :deleted_at
32     t.integer :comments_count
33     t.integer :approved_comments_count
34     t.integer :likes_count
35     t.references :root, null: true, foreign_key: { to_table: :comments }
36     t.references :parent, null: true, foreign_key: { to_table: :comments }
37     t.references :creator, polymorphic: true, null: false, index: true
38     t.references :commentable, polymorphic: true, null: false, index: true
39
40     t.timestamps
41   end
42 end
43 end
44
```

## 二、API CRUD

```
# 商品分类列表接口
# GET /api/v1/categories
# params: title_cont
desc "商品分类列表", {
  | success: API::Entities::Category
}
params do
  | optional :title_cont, type: String, desc: "分类名称"
end
paginate
get do
  | authorize_and_response_collection
end
```

# 三、组件封装

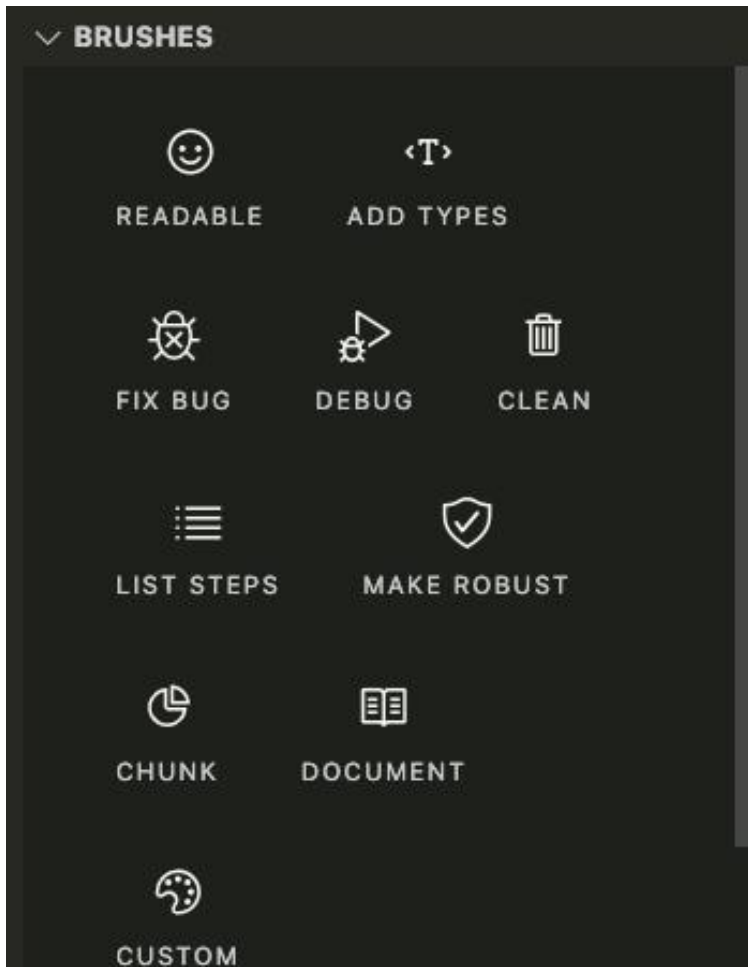
```
> apple_pay > request.rb
1 # frozen_string_literal: true
2
3 # ApplePay::Request组件, 包含get、post等基础方法, 使用restClient组件, 并对返回结果进行了封装
4 module ApplePay
5   class Request
6     include Singleton
7
8     def get(url, headers = {})
9       request(:get, url, headers)
10    end
11
12    def post(url, payload = {}, headers = {})
13      request(:post, url, headers, payload)
14    end
15
16    def put(url, payload = {}, headers = {})
17      request(:put, url, headers, payload)
18    end
19
20    def delete(url, headers = {})
21      request(:delete, url, headers)
22    end
23
24    def request(method, url, headers = {}, payload = {})
25      RestClient::Request.execute(
26        method: method,
27        url: url,
28        headers: headers,
29        payload: payload,
30        timeout: 10,
31        open_timeout: 10
32      )
33    rescue RestClient::ExceptionWithResponse => e
34    end
35  end
36 end
```

## 四、多语言翻译

```
category.rb orders.rb order.rb ! grape_api.zh-CN.yml ! grape_api.en.yml ! grape_api.zh-TW.yml ...
config > locales > ! grape_api.en.yml > ...
1 ---
2 en:
3   grape:
4     connection:
       timeout: Connection timeout
     users:
       account_deleted: Account deleted
       invalid_mobile: Invalid mobile
       failed_to_auth_wechat_info: Failed to auth wechat info
       already_a_member: Already a member
       provide_correct_code: Provide correct code
       already_bound_to_wechat: Already bound to wechat
       be_bound_to_another_member_account: Be bound to another member account
       phone_number_taken: Phone number taken
       become_member_first: Become member first
       failed_to_login: Failed to login
     student_application:
       application_exists: Application exists
       already_an_overseas_student: Already an overseas student
     shopping_cart_items:
       record_not_unique: Record not unique
       already_an_overseas_student: Already an overseas student
     auction_activities:
       can_not_create_bid: Can not create bid
       already_submitted_automatic_bid: Already submitted automatic bid
     orders:
       receiver_info_is_required: Receiver info is required
       can_not_pay: Can not pay
       can_not_create_duplicated_order: Can not create duplicated order
       coupon_and_skus_all_give: and
       gift_notice_text: "Order amount has reached $#{amount}, you can choose the following gifts %{coupon_
       all_give_text: Gifts can be selected
       gift_notice_notice_text: "Buy $#{amount} and get the following gifts. Your order is still $#{still
```

## 五、Copilot 其他姿势

1. 给代码写注释
2. 让他给你解释同事写的代码
3. 让他给你找bug
4. 让他给你生成测试代码（仅js）
- 5....







**Chatgpt**

# 一、i18n 多语言翻译

Model: GPT-4



我将会给你两个列表数据，这两个数据每一行的格式是 `key: value`。我需要你把第二个列表里的 key: value 补全，需求是第一个列表里有的key、但第二个列表没有的 key，把第一个列表里的 value 翻译成英文，补充到第二个列表里，把补充后的第二个列表输出。

### 第一个列表

```
push_message: 消息中心
sku_in_stock_notice: 到货通知统计
category: 商品分类
address: 地址
default_freight_template: 默认运费模版
exceptional_freight_template: 特殊省份运费模版
groupon_activity: 拼团活动
custom_page: 自定义页面
custom_list: 自定义产品列表
page: 独立页面
member: 会员
integral_record: 积分记录
page_popup: 弹窗管理
user_suggestion: 用户反馈
coupon_template: 优惠券模板
random_reduction_coupon_template_rule: 随机满减分配比例
preferential_pool: 优惠池
coupon_package: 优惠券礼包
coupon: 优惠券
gift_card_coupon: 礼品券
```



Below is the second list with the missing translations added:

yaml

Copy code

```
category: Category
address: Address
default_freight_template: Default Freight Template
exceptional_freight_template: Exceptional Freight Template
logistic: Logistic
groupon_activity: Groupon Activity
custom_page: Custom Page
custom_list: Custom List
page: Page
member: Member
integral_record: Integral Record
page_popup: Page Popup
user_suggestion: User Suggestion
coupon_template: Coupon Template
random_reduction_coupon_template_rule: Random Reduction Assignment Ratio
preferential_pool: Preferential Pool
```

## 二、生成 fake data



2023/4/26 16:43:50

好的，以下是 10 条虚假的商品评价内容：

1. 这个商品真的很棒！质量非常好，价格也很实惠。我肯定会向我的朋友们推荐这个品牌。
2. 我对这个商品感到非常失望。它的质量非常差，完全不符合我的期望。我不会再买这个品牌的任何东西了。
3. 这个商品非常实用。我真的很喜欢它的设计，使用起来很方便。我准备再为家人买几个。
4. 这个商品送货非常快，质量也不错。售后服务也很好，他们的客服帮了我很多。
5. 我对这个商品的外观和手感非常满意。这是一件高质量的产品，可以为我的日常生活提供很好的帮助。
6. 这个商品非常棒！它的功能非常强大，价格也很适中。我感到非常满意。
7. 我无法理解为什么这个商品能够卖得这么贵。它的质量并不比其他相似的商品更好。我不建议购买。
8. 这个商品真的超出了我的预期。它有一个很好的外观和质地，我认为它的价格非常合理。
9. 我对这个商品没有什么特别的想法。它的质量很一般，但也不算太糟糕。我可能会考虑购买其他品牌的产品。
10. 这个商品的性价比非常高。它的价格很便宜，但是它的质量却比预期要好。我很喜欢这次购物体验。

2023/4/26 16:42:54



帮我生成10个商品的评价内容的假数据

## 二、生成 fake data

2023/4/26 16:44:06



帮我写一个rails的rake task 将以上评价内容写入 comments 表的 content 字段



2023/4/26 16:46:06

好的，以下是一个简单的 Rails Rake 任务 (task)，它可以上面生成的 10 条虚假评价内容写入 comments 表的 content 字段中：

1. 在命令行中输入 `rails g task import_comments`，创建一个名为 `import_comments` 的 Rake 任务。
2. 打开 `lib/tasks/import_comments.rake` 文件，编写以下代码：

ruby 复制代码

```
namespace :import do
  desc "Import fake comments to the comments table"
  task :comments => :environment do
    comments = [
      "这个商品真的很棒！质量非常好，价格也很实惠。我肯定会向我的朋友们推荐这个品牌。",
      "我对这个商品感到非常失望。它的质量非常差，完全不符合我的期望。我不会再买这个品牌的任何东西了。",
      "这个商品非常实用。我真的很喜欢它的设计，使用起来很方便。我准备再为家人买几个。",
      "这个商品送货非常快，质量也不错。售后服务也很好，他们的客服帮了我很多。",
      "我对这个商品的外观和手感都非常满意。这是一件高质量的产品，可以为我的日常生活提供很好的帮助。",
      "这个商品非常棒！它的功能非常强大，价格也很适中。我感到非常满意。",
      "我无法理解为什么这个商品能够卖得这么贵。它的质量并不比其他相似的商品更好。我不建议购买。",
      "这个商品真的超出了我的预期。它有一个很好的外观和质地，我认为它的价格非常合理。",
      "我对这个商品没有什么特别的想法。它的质量很一般，但也不算太糟糕。我可能会考虑购买其他品牌的产品。",
      "这个商品的性价比非常高。它的价格很便宜，但是它的质量却比预期要好。我很满意这次购物体验。"
    ]

    comments.each do |content|
      Comment.create(content: content)
    end

    puts "#{comments.size} comments have been created successfully!"
  end
end
```

3. 在命令行中输入 `rake import:comments` 运行任务，将虚假评价内容导入 comments 表中。

这样就可以将上面生成的虚假评价内容写入 comments 表的 content 字段中了。



# 三、编写测试用例

现在我们要做一个NBA官方商城，请针对购物车模块编写测试用例，需要站在功能测试、接口测试、性能角度进行分析。



2023/4/19 16:05:43

## 1. 功能测试:

- 测试购物车模块能否将选定的商品添加到购物车;
- 测试购物车模块能否正确显示商品信息, 如名称、价格、数量等;
- 测试购物车模块能否计算并展示商品总价;
- 测试购物车模块能否修改商品数量或删除某个商品;
- 测试购物车模块能否进行结算并生成订单。

## 2. 接口测试:

- 测试购物车模块与后端接口之间的数据传递是否正确;
- 测试购物车模块能否正常调用后端接口;
- 测试购物车模块对后端接口返回的错误码和异常情况的处理是否正确。

## 3. 性能测试:

- 测试购物车模块在大量用户请求下的响应时间是否满足要求;
- 测试购物车模块在高并发情况下的性能表现;
- 测试购物车模块的负载能力, 即在多少并发用户同时使用时会出现性能问题。

## 4. 安全测试:

- 测试购物车模块是否存在安全漏洞, 如SQL注入、XSS攻击等;
- 测试购物车模块是否采取了足够的安全措施, 如访问控制、输入过滤、加密等;
- 测试购物车模块是否符合相关安全标准和法规。

2023/4/19 16:06:19



2023/4/19 16:06:50

好的, 以下是一些可能的购物车模块功能测试用例:

### 1. 测试添加商品到购物车:

- 点击某个商品的“加入购物车”按钮后, 检查购物车中是否出现该商品;
- 检查商品名称、价格、数量等信息是否与实际添加的一致;
- 尝试添加库存不足的商品, 检查是否提示用户无法添加。

### 2. 测试修改购物车中的商品数量:

- 在购物车页面中修改某个商品的数量, 检查总价是否正确计算, 并且是否正确反映在商品列表中;
- 修改商品数量为0, 则该商品应该从购物车中移除;
- 对于单个商品, 最大数量 (通常是99) 是否得到限制。

### 3. 测试删除购物车中的商品:

- 在购物车页面中删除某个商品, 检查该商品是否被成功移除;
- 检查移除后总价是否正确计算;
- 检查清空购物车是否能够成功执行。

### 4. 测试购物车结算和生成订单:

- 点击结算按钮, 检查是否跳转至支付页面并正确展示订单信息;
- 当选择了快速方式, 检查是否正确计算运费;
- 确认订单信息后, 提交订单并检查是否成功生成订单, 并且从库存中扣除相应商品数。

### 5. 测试购物车优惠券功能:

- 尝试使用不同种类的优惠券, 检查是否正确计算折扣;
- 尝试使用已经失效或不可用的优惠券, 检查是否提示用户无法使用;
- 检查在结算之前和之后使用优惠券的效果是否一致。

### 6. 测试购物车在多语言环境下的显示:

- 切换语言环境为其他语言, 检查购物车页面中商品信息、数量、总价等是否正确翻译。

以上仅是一些示例, 根据实际需求和情况, 可能还需要对不同操作的错误处理、异常情况和边界条件进行测试。

2023/4/19 18:06:17

我需要更详细的功能测试用例

# 四、Cron/正则 表达式



2023/4/26 16:31:53

这是一个每 8 小时执行一次的 cron 表达式：

```
0 */8 * * *
```

解释：

- 0 表示分钟为 0。
- \*/8 表示每 8 小时触发一次任务，即小时数取 0、8、
- \* 表示每天都执行该任务。
- \* 表示每月都执行该任务。
- \* 表示每周都执行该任务。

因此，该表达式可以实现每 8 小时运行一次任务的效果。



2023/4/26 16:55:53

中国大陆的身份证号码是由 18 位数字或 17 位数字和一个字母组成的。以下是一个可以用于验证身份证号码的正则表达式：

```
^\A\d{6}(19|20)\d{2}(\0[1-9]|1[012])\0[1-9]|[12]\d|3[01])\d{3}[0-9xX]\z/
```

ruby 复制代码

解释：

- \A 表示匹配字符串的开始位置。
- \d{6} 表示匹配前 6 个数字，即地址编码。
- (19|20)\d{2} 表示出生年份，必须是以 19 或 20 开头的四位数字。
- (\0[1-9]|1[012]) 表示出生月份，必须是 01 到 12 之间的数字。
- (\0[1-9]|[12]\d|3[01]) 表示出生日期，必须是 01 到 31 之间的数字。
- \d{3} 表示顺序码，一般是 001 到 999。
- [0-9xX] 表示校验码，最后一位可能是数字或者大小写字母 X。
- \z 表示匹配字符串的结束位置。

因此，如果需要验证一个字符串是否为有效的身份证号码，只需要使用上面的正则表达式进行匹配即可。

2023/4/26 16:31:38



帮我生成一个 cron 的表达式，每八个小时

2023/4/26

给我生成一个身份证的验证的正则

谢谢大家