To-do list项目讲解与功能展示

项目功能

任何情况下的功能:

- 用户**注册**
- 用户**登录**

以下功能登录后使用:

- To-do**上传**
- To-do**修改**
- To-do**删除**
- To-do**筛选**查询(按截止时间,完成情况,升序/倒序排序)
- To-do按索引**查询**

新增功能:**jwt鉴权中间件**

*本项目使用**json**文件进行数据持久化

用户注册

路由

```
r.POST("/register", useregister) //注册
```

功能代码

```
func useregister(c *gin.Context) {
    var user USER
    if err := c.BindJSON(&user); err != nil {
       c.JSON(400, ErrInvalidUSERFormat)
       return
    }

    if len(user.Password) <= 6 { //密码长度过短提示重新设置
       c.JSON(400, ErrInvalidPassword)
       return
    }

    existingUsers, err := loadUsersFromFile()
    if err != nil {
       c.JSON(500, ErrReadUserData)
       return
    }

    // 检查是否已经存在相同的用户名
    for _, existingUser := range existingUsers {
       if existingUser.Username == user.Username {
          c.JSON(400, ErrRegister)
          return
```

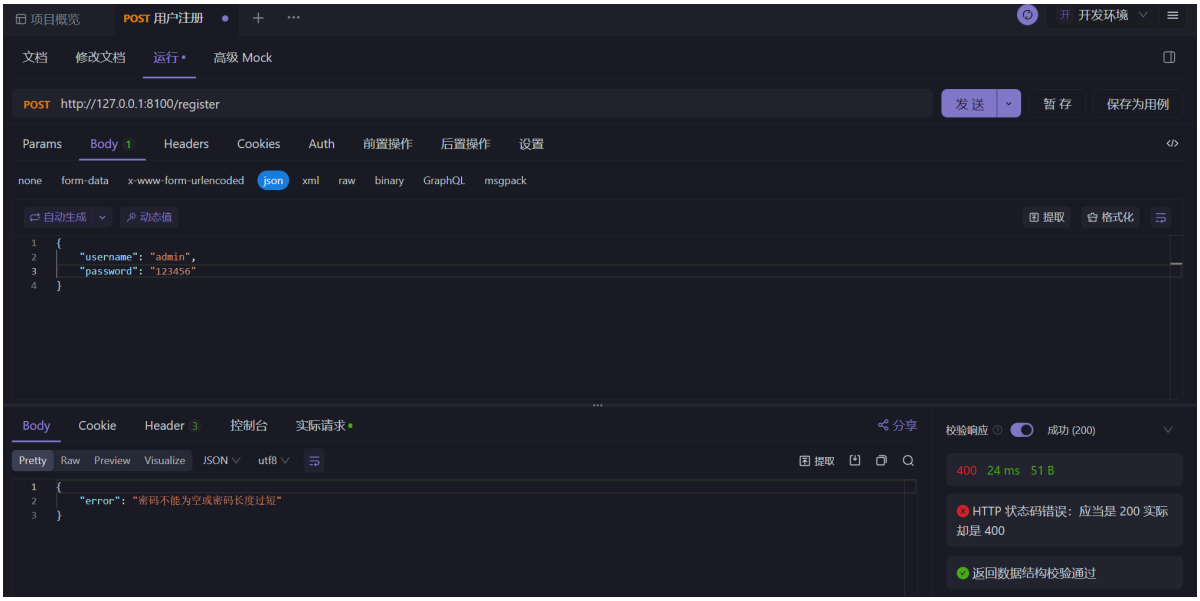
```
        }
    }

    existingUsers = append(existingUsers, user)
    err = saveUsersToFile(existingUsers)
    if err != nil {
        c.JSON(500, ErrSaveUserData)
        return
    }

    c.JSON(200, UserRegisterSuccess)
}
```

代码逻辑

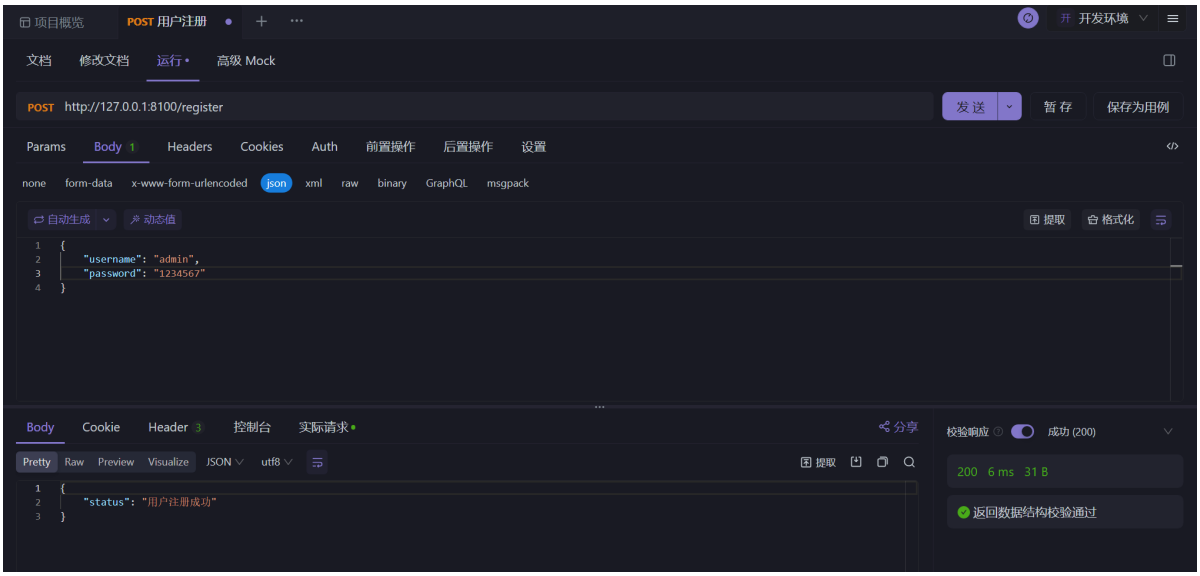
1. 绑定json数据至user结构体
2. 判断密码长度
3. 读取用户文件并查重用户名
4. 保存用户名至文件

Apifox测试

尝试注册,失败,密码长度过短

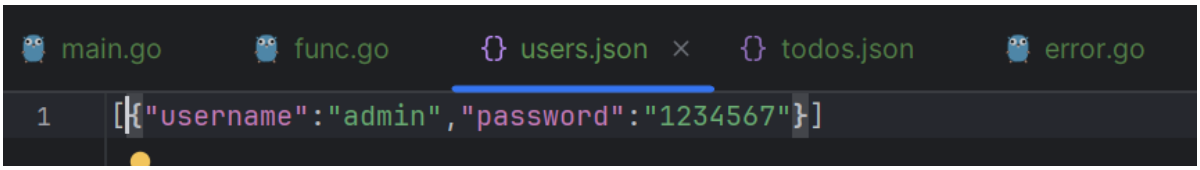


注册成功



注册失败,用户名已被注册

用户名在user.json中存储成功



用户登录

路由

```
r.POST("/login", userlogin)      //登录
```

功能代码

```
func userlogin(c *gin.Context) {
    var user USER
    if err := c.BindJSON(&user); err != nil {
        c.JSON(400, ErrInvalidUSERFormat)
        return
    }
```

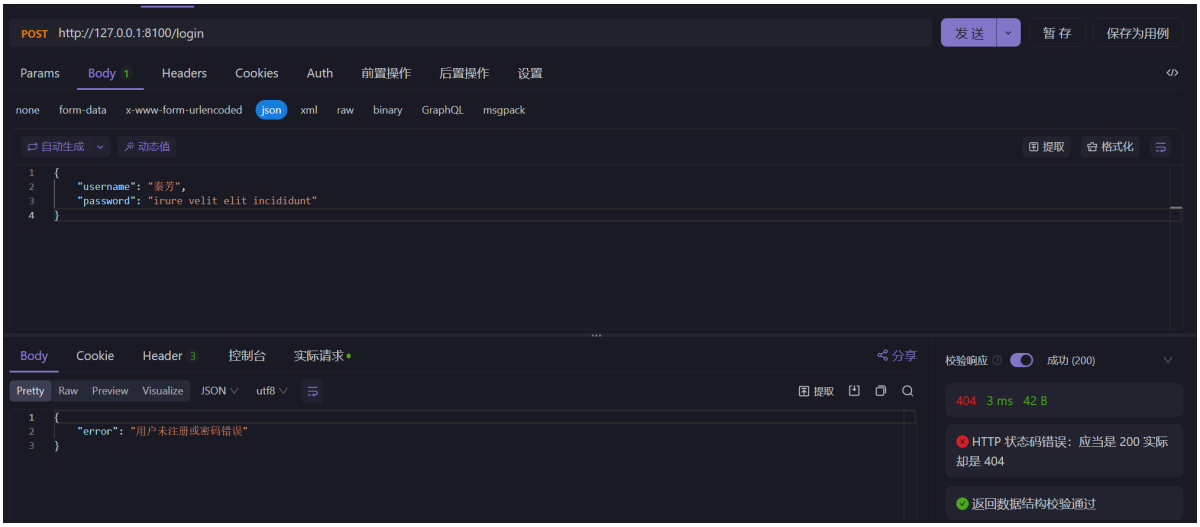
```
    existingUsers, err := loadUsersFromFile()
    if err != nil {
        c.JSON(500, ErrReadUserData)
        return
    }

    var foundUser USER
    for _, existingUser := range existingUsers {
        if existingUser.Username == user.Username && existingUser.Password ==
user.Password {
            foundUser = existingUser
            break
        }
    }

    if foundUser.Username != "" {
        currentUser = foundUser.Username // 设置全局变量为当前用户的用户名
        c.JSON(200, gin.H{"status": "用户登录成功", "username": foundUser.Username})
    } else {
        c.JSON(404, ErrUserlogin)
    }
}
```

代码逻辑

1. 绑定json数据至user结构体
2. 读取用户文件
3. 遍历判断用户名密码是否符合
4. 返回登录状态,用户名和token

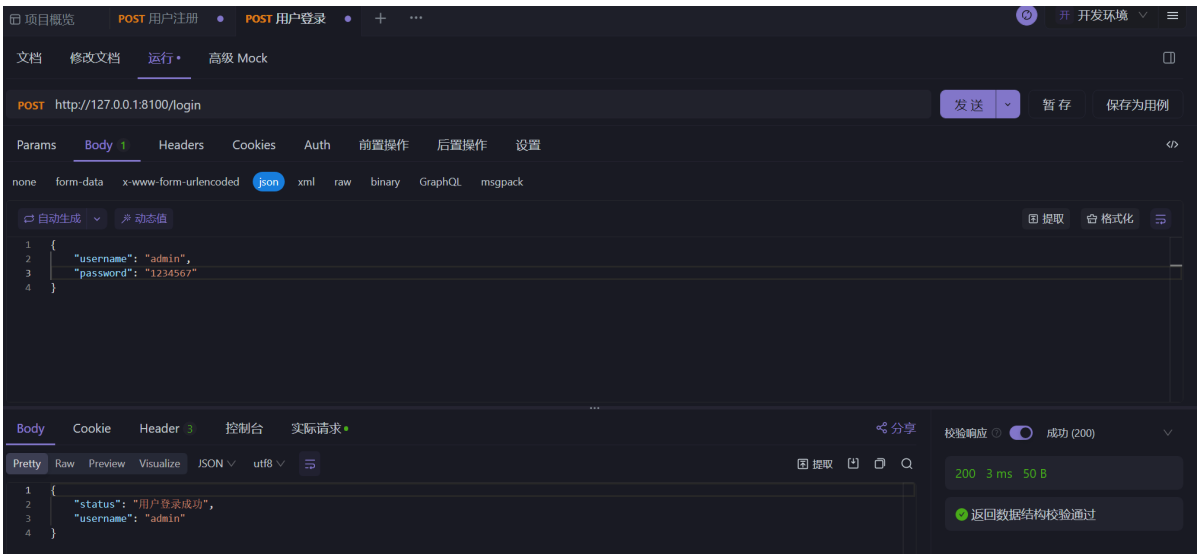
Apifox测试

用户名未注册



用户名密码不匹配

登录成功



数据上传

路由

```
authGroup.POST("/todo", TodoCreation)          //增
```

功能代码

```
func TodoCreation(c *gin.Context) {
    currentUser := currentUser
    if currentUser == "" {
        c.JSON(401, ErrUser)
        return
    }

    var todo TODO
    if err := c.BindJSON(&todo); err != nil {
        c.JSON(400, ErrInvalidTODOFormat)
        return
    }

    if todo.Deadline == 0 {
```

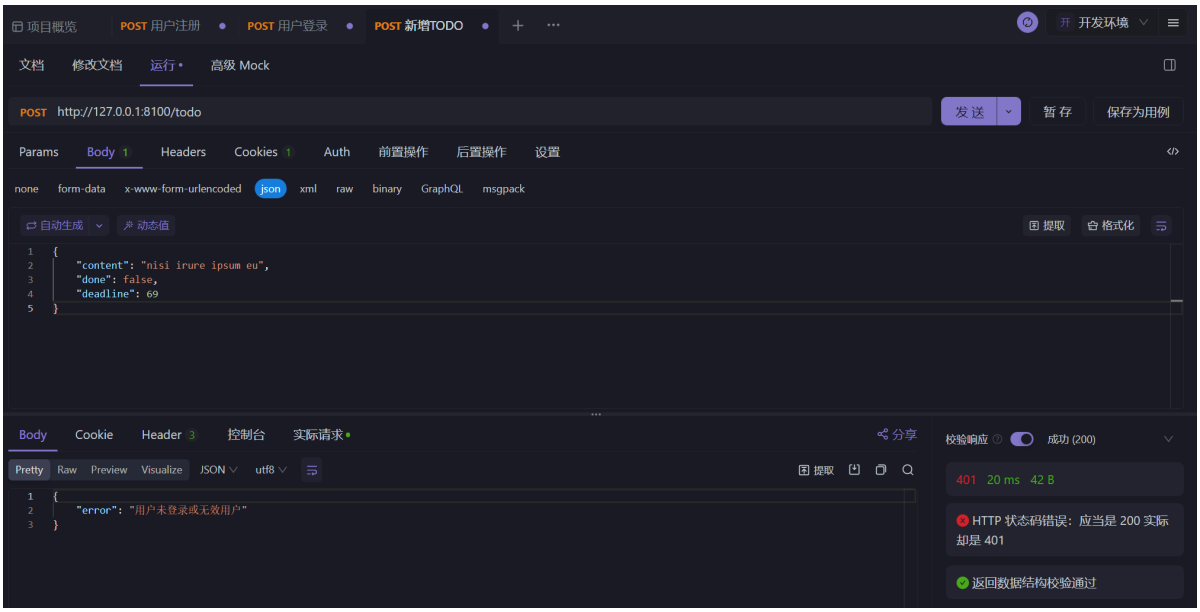
```
        defaultDeadline := time.Now().Add(time.Hour * 24 * 7)
        todo.Deadline = UnixTimestamp(defaultDeadline.Unix())
    } else if int64(todo.Deadline) < time.Now().Unix() {
        c.JSON(400, ErrInvalidDeadline)
        return
    }

    existingTodos, err := loadTodosFromFile()
    if err != nil {
        c.JSON(500, ErrReadTODOData)
        return
    }

    // 计算用户的索引
    userIndex := 1
    for _, t := range existingTodos {
        if t.Username == currentUser {
            userIndex++
        }
    }

    // 为新的 Todo 分配 index 序号
    todo.Index = userIndex
    todo.Username = currentUser

    existingTodos = append(existingTodos, todo)

    err = saveTodosToFile(existingTodos)
    if err != nil {
        c.JSON(500, ErrSaveTODOData)
        return
    }
    c.JSON(200, TodoSubmitSuccess)
}
```

代码逻辑

1. 获取当前用户
2. 尝试绑定json数据到todo结构体变量
3. 解析传入的截止时间(必须在当前时间之后)
4. 如果todo数据格式正确,则计算已经存储的,属于当前用户的todo的个数,并将其值+1作为新的index传入
5. 结构体已经被全部赋值,使用`json`库的`marshal/unmarshal`和`ioutil`库的`writefile/readfile`来进行文件读取与保存实现功能

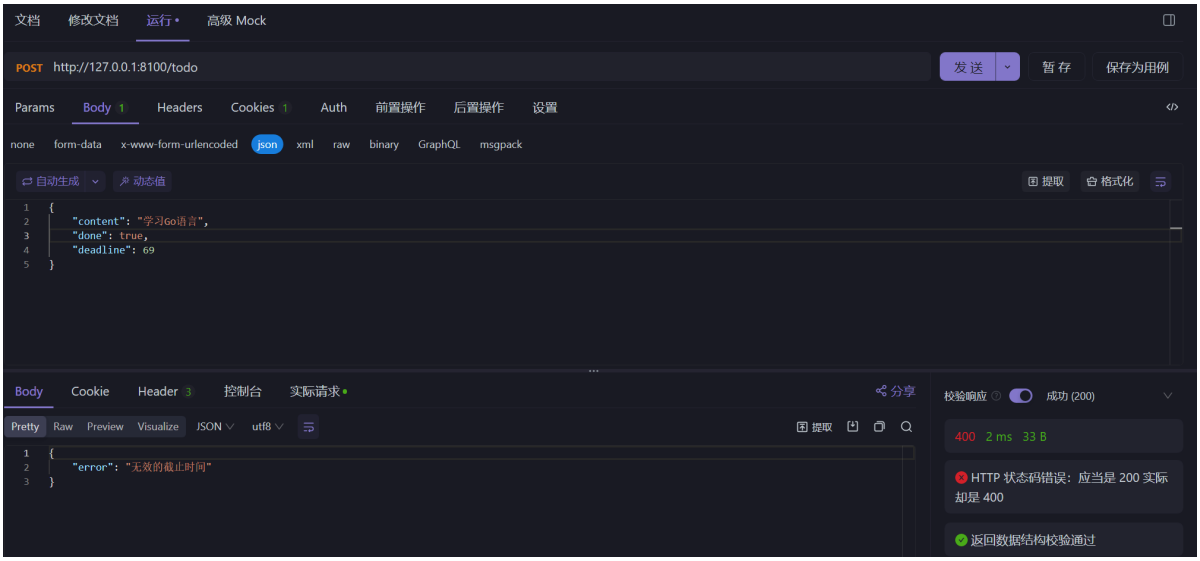
Apifox测试

用户未登录,操作无效

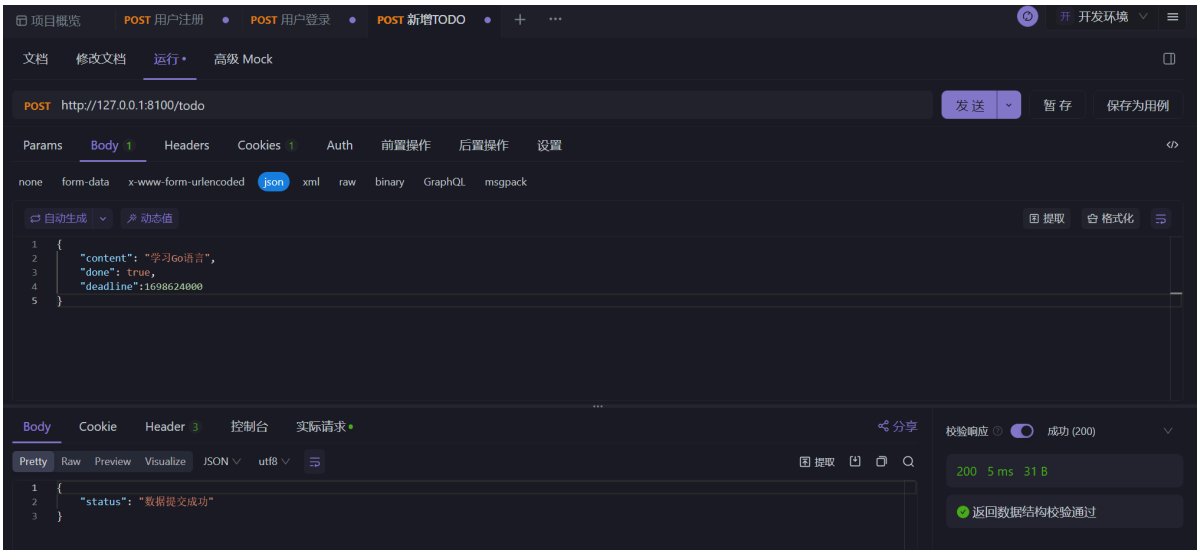


进行用户登录

截止时间无效



数据提交成功



数据存储

数据修改

路由

```
authGroup.PUT("/todo/:index", TodoUpdate)      //改
```

功能代码

```
func TodoUpdate(c *gin.Context) {
    currentUser := currentUser
    if currentUser == "" {
       c.JSON(401, ErrUser)
       return
    }
    indexToUpdate, err := strconv.Atoi(c.Param("index"))
    indexToUpdate -= 1
    if err != nil || indexToUpdate < 0 {
       c.JSON(404, ErrTODOIndexNotExist)
       return
    }

    var todo TODO
    if err := c.BindJSON(&todo); err != nil {
       c.JSON(400, ErrInvalidTODOFormat)
       return
    }

    existingTodos, err := loadTodosFromFile()
    if err != nil {
       c.JSON(500, ErrReadTODOData)
       return
    }

    // 遍历待办事项列表，找到与当前用户匹配的待办事项并匹配索引
```

```
    for index, existingTodo := range existingTodos {
        if existingTodo.Username == currentUser && index == indexToUpdate {
            // 更新待办事项内容
            existingTodo.Content = todo.Content
            existingTodo.Done = todo.Done
            existingTodo.Deadline = todo.Deadline

            // 更新待办事项回到列表
            existingTodos[index] = existingTodo

            err = saveTodosToFile(existingTodos)
            if err != nil {
                c.JSON(500, ErrSaveTODOData)
                return
            }

            c.JSON(200, gin.H{"status": "修改成功"})
            return
        }
    }

    // 如果没有匹配的待办事项，返回错误
    c.JSON(404, ErrTODOIndexNotExist)
}
```

代码逻辑

1.获取当前用户名

2.获取index参数并处理

3.遍历待办事项列表，找到与当前用户匹配的待办事项并匹配索引

4.更新待办事项内容

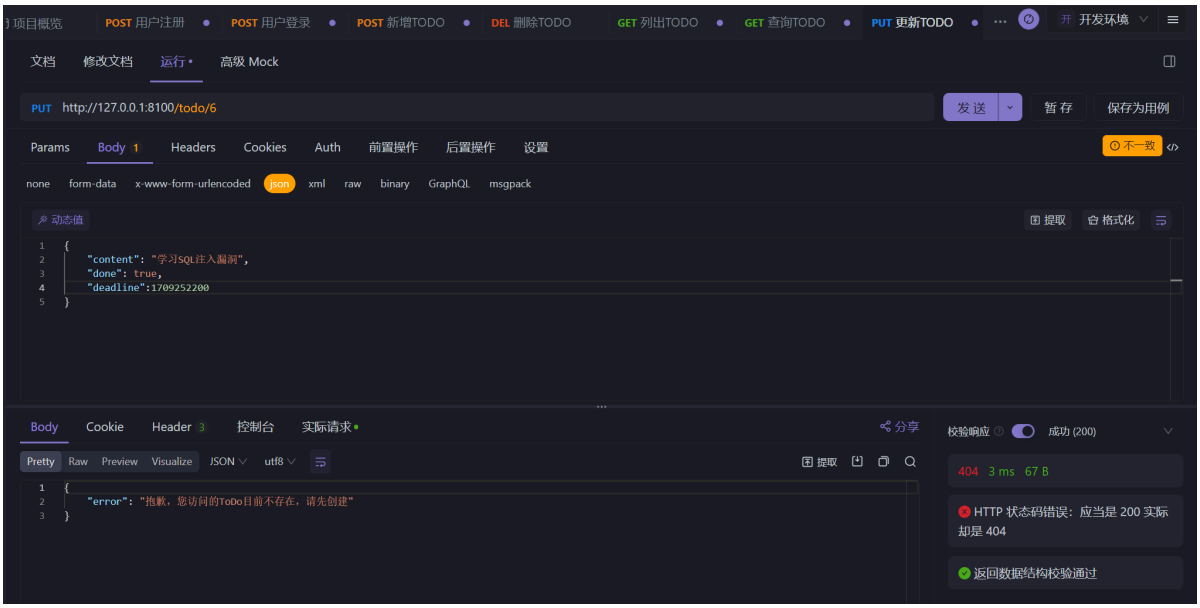
5.返回

Apifox测试

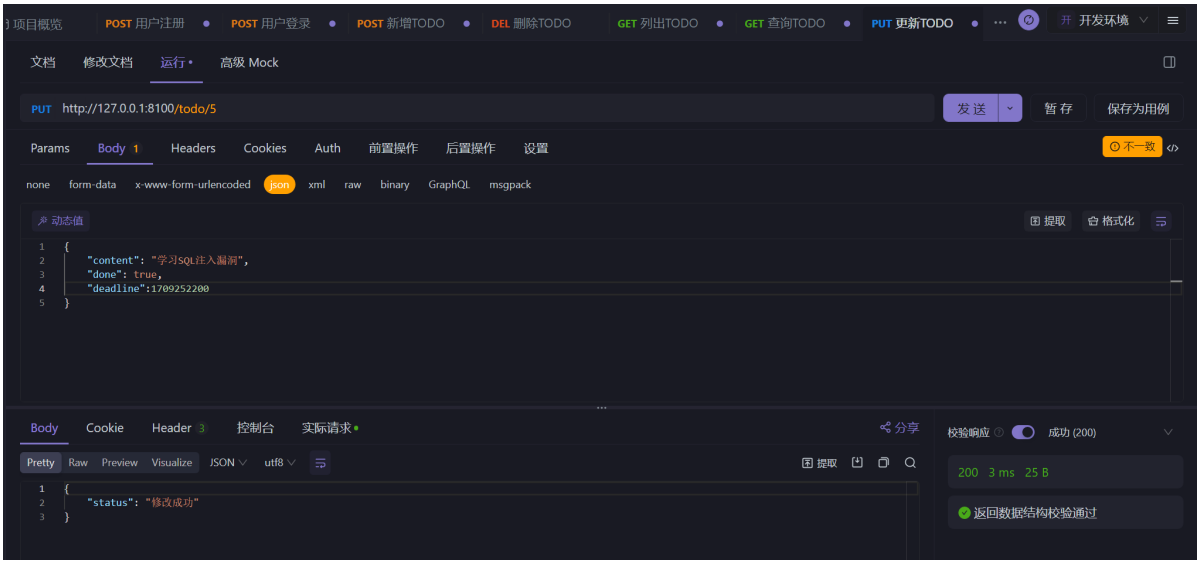
当前todos.json文件内容如下:

```
[{"username":"admin","index":1,"content":"学习Go语
言","done":true,"deadline":1698624000},{"username":"admin","index":2,"content":"学
习PHP反序列化漏洞","done":true,"deadline":1701302400},
{"username":"admin","index":3,"content":"学习文件包含漏
洞","done":true,"deadline":1703894400},{"username":"admin","index":4,"content":"学
习变量覆盖漏洞","done":true,"deadline":1706572800},
{"username":"admin","index":5,"content":"学习SQL注入漏
洞","done":true,"deadline":1709251200}]
```

todo不存在错误



更新成功



修改后:

```
[{"username":"admin","index":1,"content":"学习Go语
言","done":true,"deadline":1698624000},{"username":"admin","index":2,"content":"学
习PHP反序列化漏洞","done":true,"deadline":1701302400},
{"username":"admin","index":3,"content":"学习文件包含漏
洞","done":true,"deadline":1703894400},{"username":"admin","index":4,"content":"学
习变量覆盖漏洞","done":true,"deadline":1706572800},
{"username":"admin","index":5,"content":"学习SQL注入漏
洞","done":true,"deadline":1709252200}]
```

修改成功.

数据删除

路由

```
authGroup.DELETE("/todo/:index", TodoDeletion) // 删(不改动序号)
```

功能代码

```go
func TodoDeletion(c *gin.Context) {
    currentUser := currentUser
    if currentUser == "" {
       c.JSON(401, gin.H{"status": "用户未登录或无效的用户"})
       return
    }
    indexToDelete, err := strconv.Atoi(c.Param("index"))
    indexToDelete-=1
    if err != nil || indexToDelete < 0 {
       c.JSON(404, ErrTODOIndexNotExist)
       return
    }

    existingTodos, err := loadTodosFromFile()
    if err != nil {
       c.JSON(500, ErrTODONotFound)
       return
    }

    var deletedContent string // 用于保存被删除的待办事项内容

    // 遍历待办事项列表，找到与当前用户匹配的待办事项并匹配索引
    for index, todo := range existingTodos {
       if todo.Username == currentUser && index == indexToDelete {
          // 检查是否已经在已删除的待办事项列表中
          if todo.Content == "此Todo已被删除" {
             c.JSON(400, TodoDeleteSuccess)
             return
          }
          // 保存被删除的待办事项内容
          deletedContent = todo.Content

          // 标记待办事项为已删除
          todo.Content = "此Todo已被删除"
          todo.Done = true

          // 更新待办事项回到列表
          existingTodos[index] = todo
          err = saveTodosToFile(existingTodos)
          if err != nil {
             c.JSON(500, ErrSaveTODOData)
             return
          }

          // 在 JSON 响应中包括被删除的内容
          c.JSON(200, gin.H{"status": "删除成功", "被删除的内容是": deletedContent})
          return
       }
    }

    // 如果没有匹配的待办事项，返回错误
    c.JSON(404, ErrTODOIndexNotExist)
}
```

代码逻辑

1.获取当前用户名

2.获取index参数并处理

3.遍历待办事项列表，找到与当前用户匹配的待办事项并匹配索引

4.检查是否已经在已删除的待办事项列表中,若否则保存被删除的待办事项内容,标记待办事项为已删除,并更新待办事项回到列表

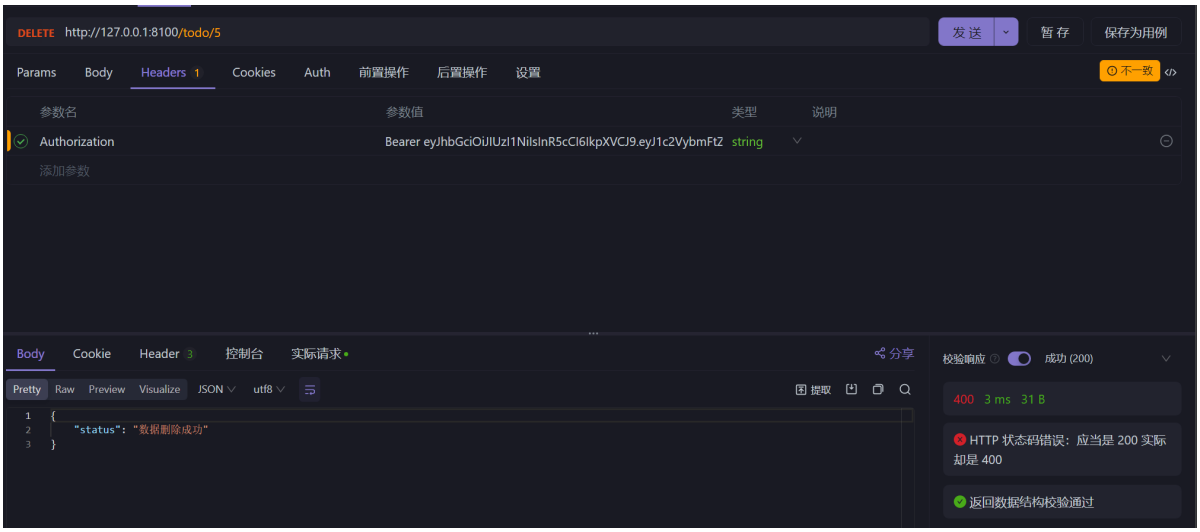
5.在json响应中返回被删除的内容和删除状态码.

Apifox测试

删除不存在的数据(失败)

删除成功,显示删除内容

重复删除,显示提示



数据筛选查询

路由

```
authGroup.GET("/todo", ListTodos)                    // 查(使用条件筛选)
```

功能代码

```
func ListTodos(c *gin.Context) {
    // 从请求上下文中获取当前用户的用户名
    currentUser := currentUser
    if currentUser == "" {
       c.JSON(401, ErrUser)
       return
    }

    existingTodos, err := loadTodosFromFile()
    if err != nil {
       c.JSON(500, ErrReadTODOData)
       return
    }

    // 获取查询参数
    deadline := c.DefaultQuery("deadline", "0") // 默认值设置为 "0"，表示不进行筛选
    reverse := c.DefaultQuery("reverse", "false")
    finished := c.DefaultQuery("finished", "")

    // 转换 reverse 字符串为布尔值
    reverseSort := (reverse != "true")

    // 根据 finished 参数过滤待办事项
    filteredTodos := []TODOWithOriginalIndex{} // 使用新的结构体保存待办事项和原始索引
    for index, todo := range existingTodos {
       // 检查索引是否在 deletedTodoIndexes 中，如果在就跳过
       if todo.Content == "此Todo已被删除" {
          continue
       }

       // 只返回属于当前用户的待办事项
       if todo.Username != currentUser {
```

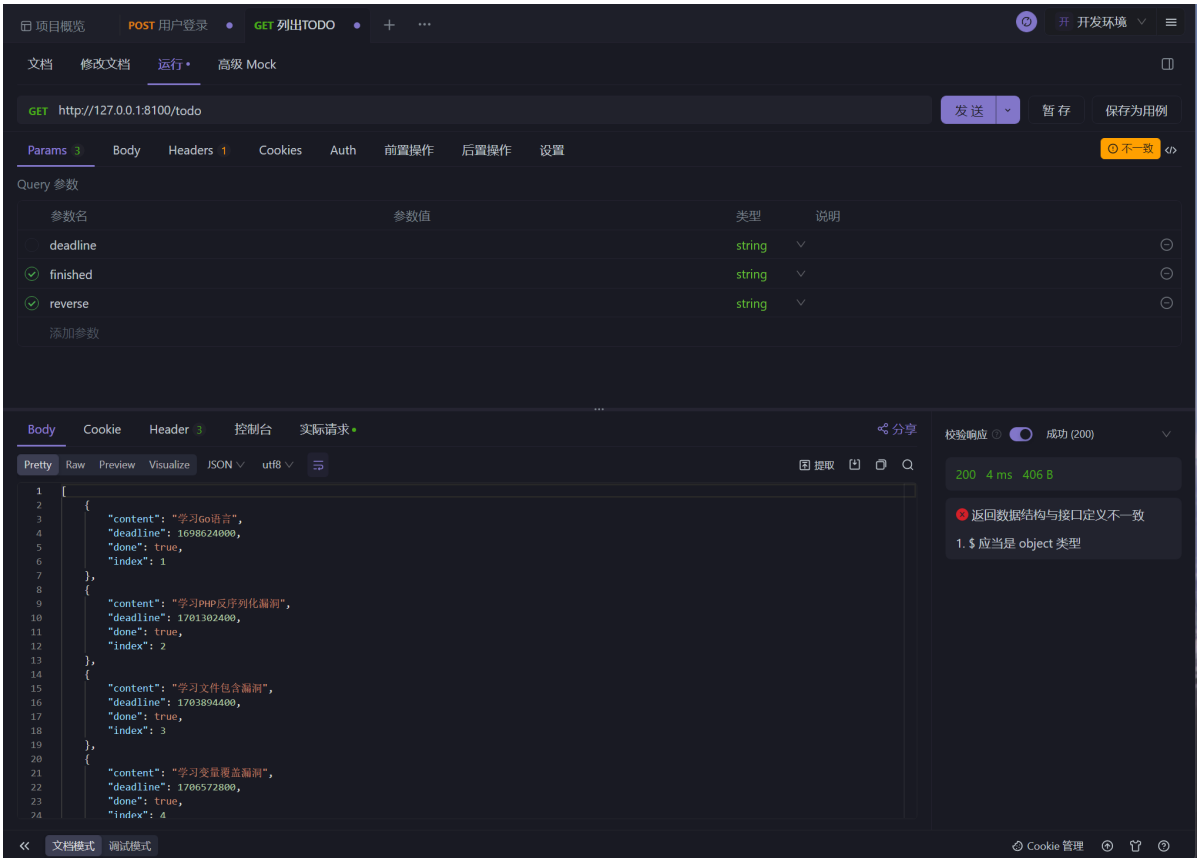
```
            continue
        }

        // 直接将查询参数转换为整数
        queryDeadline, err := strconv.ParseInt(deadline, 10, 64)
        if err != nil {
            c.JSON(400, ErrInvalidDeadline)
            return
        }

        // 检查截止日期是否符合筛选条件
        if (finished == "true" && todo.Done) || (finished == "false" &&
!todo.Done) || finished == "" {
            if queryDeadline == 0 || (queryDeadline != 0 && int64(todo.Deadline) <=
queryDeadline) {
                // 保存待办事项和原始索引
                filteredTodos = append(filteredTodos, TODOWithOriginalIndex{todo,
index})
            }
        }
    }

    // 根据 reverseSort 参数排序
    if reverseSort {
        sort.Slice(filteredTodos, func(i, j int) bool {
            return int64(filteredTodos[i].Todo.Deadline) <
int64(filteredTodos[j].Todo.Deadline)
        })
    } else {
        sort.Slice(filteredTodos, func(i, j int) bool {
            return int64(filteredTodos[i].Todo.Deadline) >
int64(filteredTodos[j].Todo.Deadline)
        })
    }

    // 返回结果
    todosWithIndex := []map[string]interface{}{}

    for _, todo := range filteredTodos {
        todoWithIndex := map[string]interface{}{
            "index":    todo.Index + 1,
            "content":  todo.Todo.Content,
            "done":     todo.Todo.Done,
            "deadline": todo.Todo.Deadline, // 已经是 int64 格式
        }
        todosWithIndex = append(todosWithIndex, todoWithIndex)
    }

    c.JSON(200, todosWithIndex)
}
```

代码逻辑

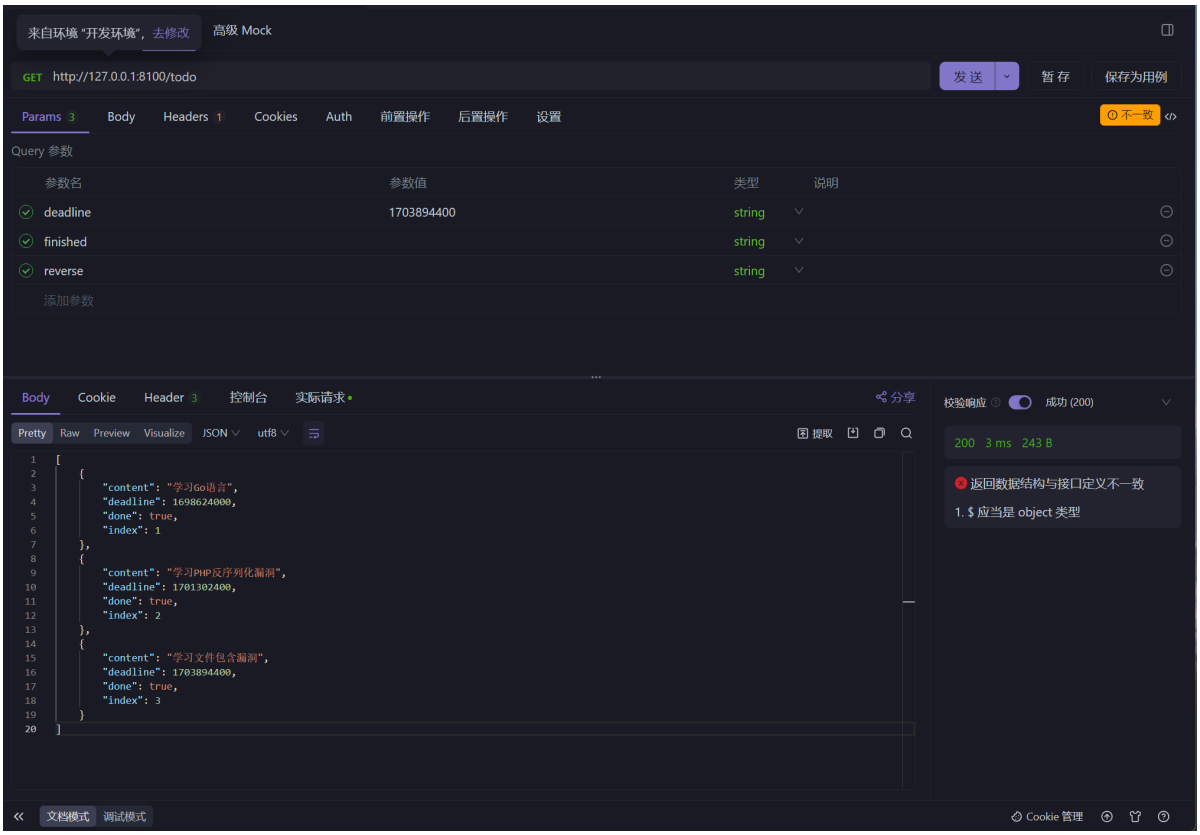
1. 获取当前用户名
2. 从文件中读取todo,将筛选参数赋值(未传入按照默认值赋值)
3. 进行筛选,默认按照截止时间从小到大升序排列,相同则按照index从小到大排列;若传入筛选参数deadline,则筛选出deadline之前的所有todo(包含等于deadline的情况);若传入done参数,则按照done属性为true/false筛选;传入reverse参数,false为升序排列,true为降序排列.
4. 保存原有的todo序号输出.

Apifox测试

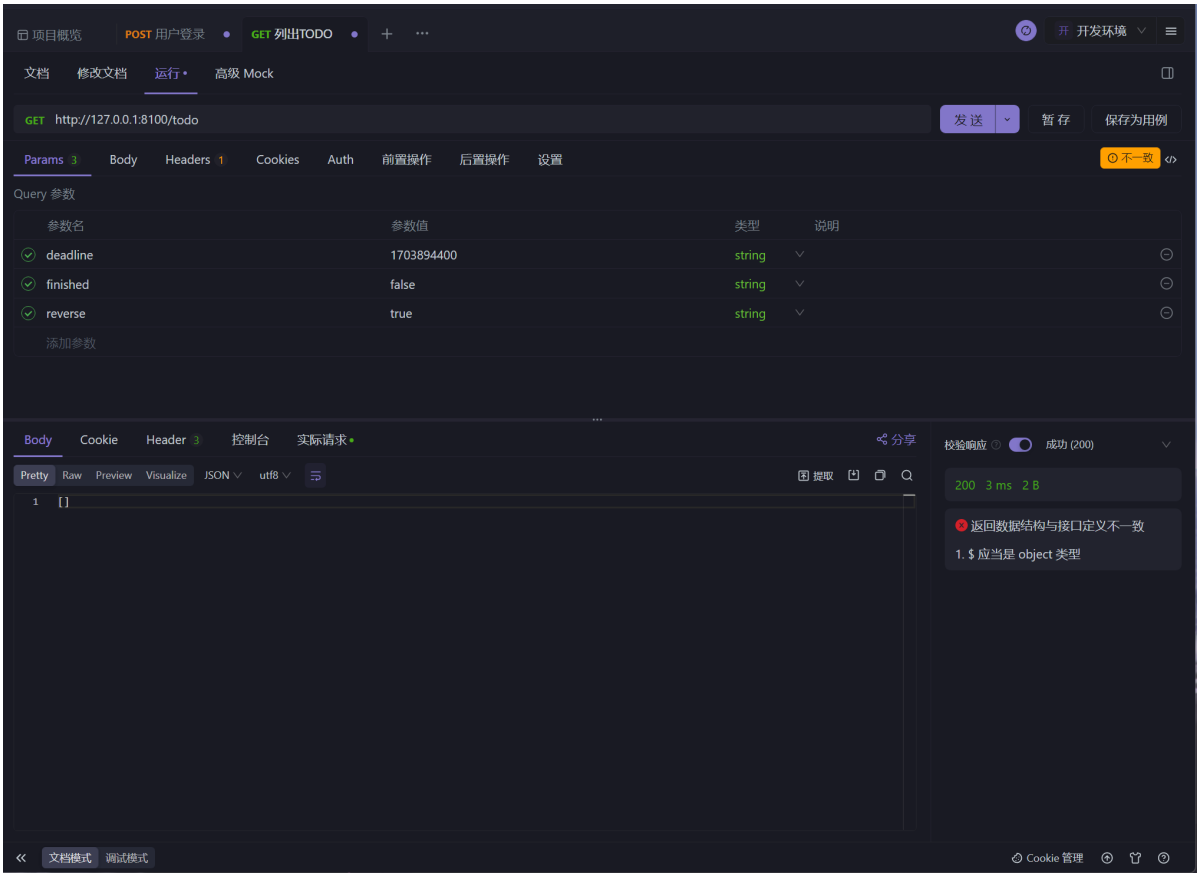
未输入参数,默认全部(不能勾选deadline参数)



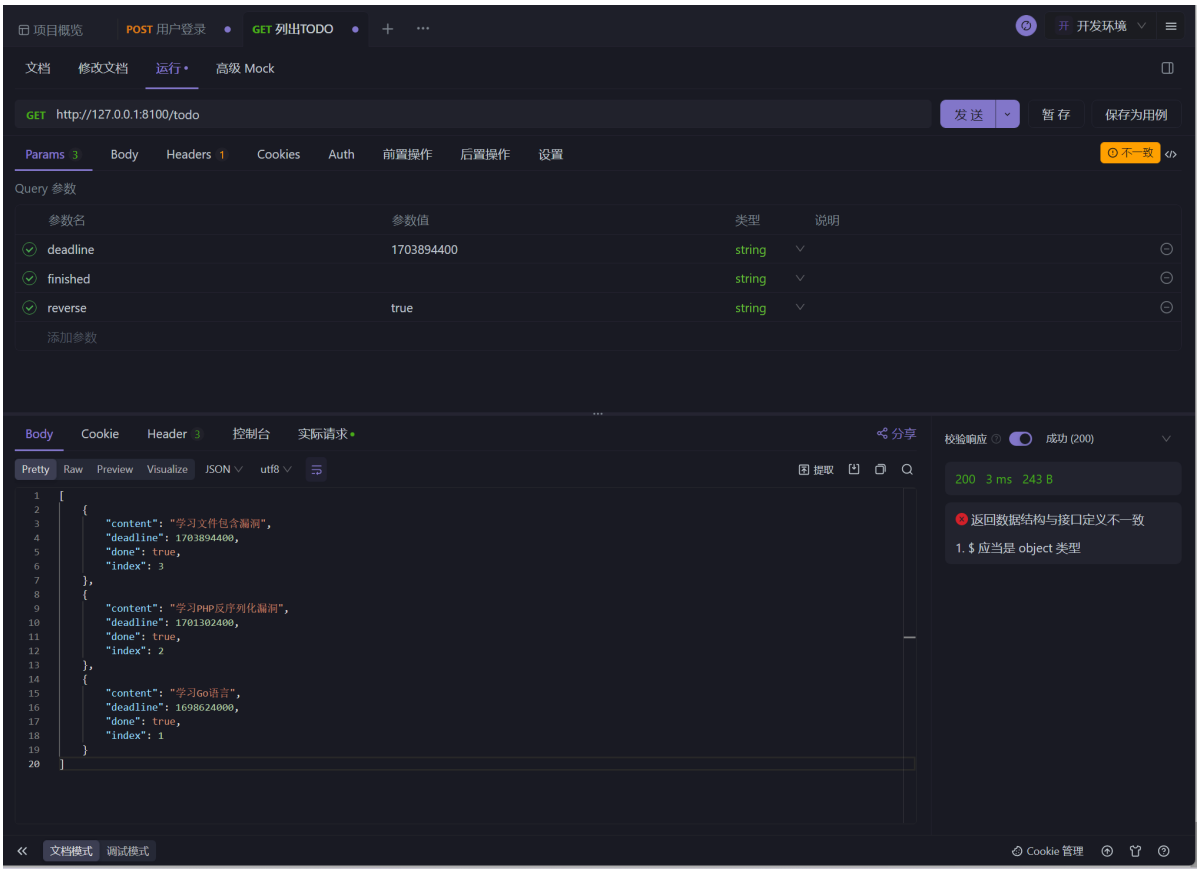
设置deadline,筛选出deadline之前的数据



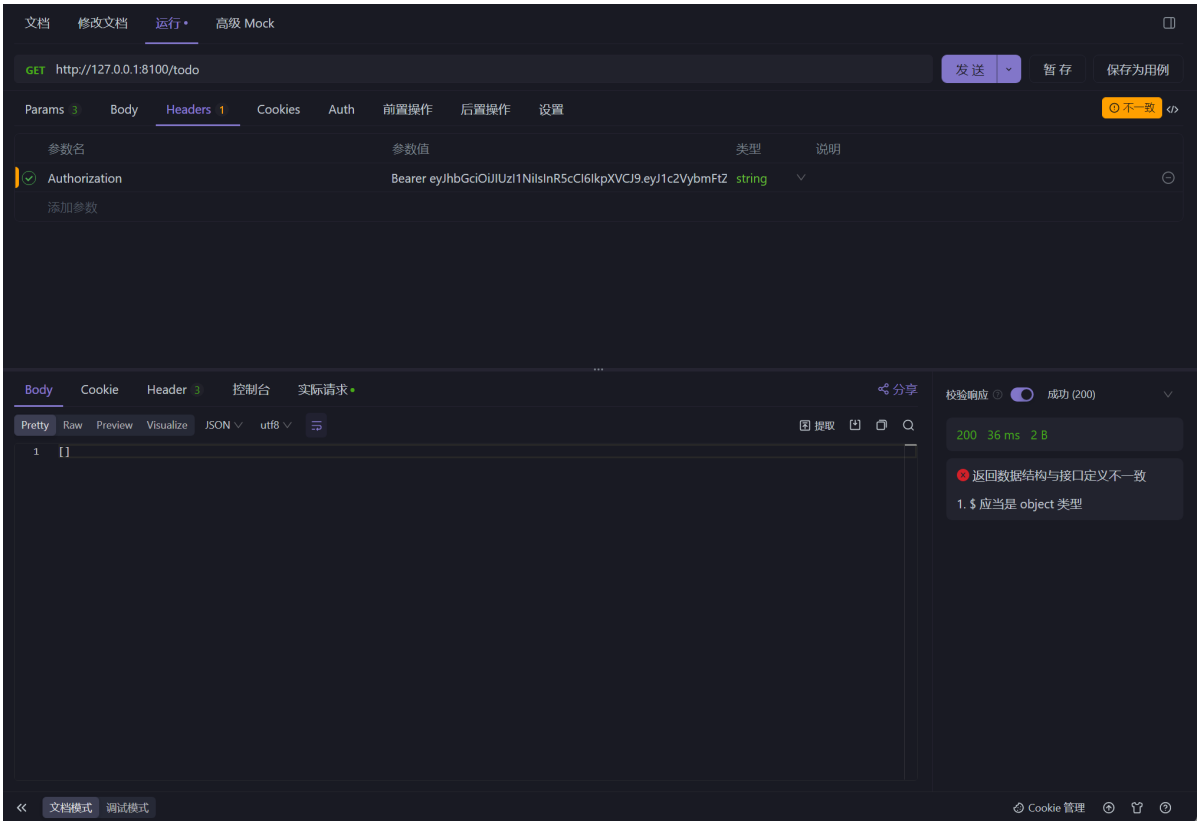
设置finished数据,筛选done值为false的数据(这里没有)



reverse为true,降序输出数据



切换到另一位用户,没有筛选出原用户的todo,实现用户数据隔离.(这里使用的是guest用户的token)



数据索引查询

路由

```
authGroup.GET("/todo/:index", GetTodo)          // 获取单个 todo 信息
```

功能代码

```go
func GetTodo(c *gin.Context) {
    currentUser := currentUser
    if currentUser == "" {
       c.JSON(401, gin.H{"status": "用户未登录或无效的用户"})
       return
    }
    indexToGet, err := strconv.Atoi(c.Param("index"))
    if err != nil || indexToGet < 0 {
       c.JSON(404, ErrTODOIndexNotExist)
       return
    }
    filteredTodo := []TODO{}
    existingTodos, err := loadTodosFromFile()
    if err != nil {
       c.JSON(500, ErrReadTODOData)
       return
    }
    for _, todo := range existingTodos {
       // 检查索引是否在 deletedTodoIndexes 中，如果在就跳过
       if todo.Content == "此Todo已被删除" {
          continue
       }

       // 只返回属于当前用户的待办事项
       if todo.Username != currentUser {
          continue
       }

       if todo.Username == currentUser && todo.Index == indexToGet {
          filteredTodo = append(filteredTodo, todo)
          c.JSON(200, filteredTodo)
          return
       }

    }
    c.JSON(404, ErrTODONotFound)
}
```

代码逻辑

1.获取当前用户名

2.获取index参数并处理

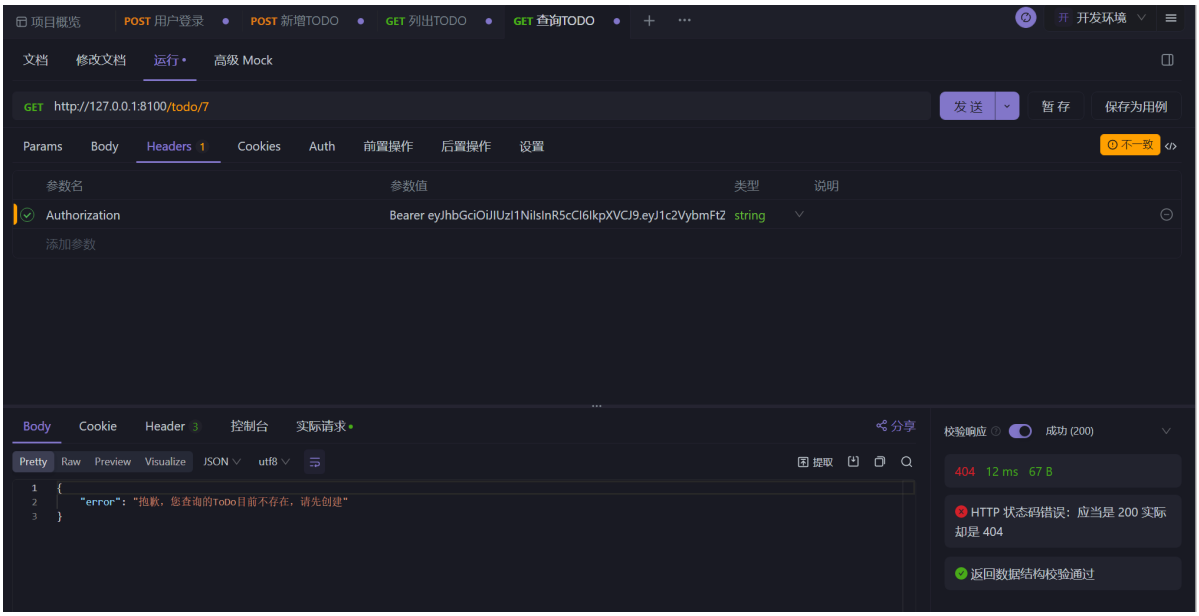
3.遍历待办事项列表，找到与当前用户匹配的待办事项并匹配索引

4.获取待办事项

5.返回

Apifox测试

查询的参数不存在报错



查询成功,返回信息

JWT鉴权

功能代码

生成一个token

```
func createToken(username string) (string, error) {
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
       "username": username,
    })
    tokenString, err := token.SignedString(jwtKey)
    if err != nil {
       return "", err
    }
    return tokenString, nil
}
```

JWT中间件获取登录状态

```
func JWTMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
       tokenString := c.GetHeader("Authorization")
       print(tokenString)
       print("123456")
       if tokenString == "" {
          c.JSON(401, gin.H{"error": "未提供令牌"})
          c.Abort()
          return
       }

       // 去掉 "Bearer " 部分
       tokenString = strings.Replace(tokenString, "Bearer ", "", 1)

       token, err := jwt.Parse(tokenString, func(token *jwt.Token) (interface{},
error) {
          return jwtKey, nil
       })

       if err != nil {
          c.JSON(401, gin.H{"error": "无效的令牌"})
          c.Abort()
          return
       }

       claims, ok := token.Claims.(jwt.MapClaims)
       if !ok || !token.Valid {
          c.JSON(401, gin.H{"error": "无效的令牌"})
          c.Abort()
          return
       }

       username, ok := claims["username"].(string)
       if !ok {
          c.JSON(401, gin.H{"error": "无效的令牌"})
          c.Abort()
          return
       }

       currentUser = username
       c.Next()
    }
}
```

这个JWT中间件实现了一个简单的鉴权功能.

总结

项目进行了一些改进

- **JWT鉴权**
- **json文件存储**

- **数据排序与筛选**
- **用户注册功能**
- **用户登录功能**
- **字段名增加了截止时间**
- **实现了删除后序号不改变的问题**