

# 存客宝前后端对接

## 存客宝前后端对接

- 前端使用 `fetch` 封装请求，管理 Token 与统一错误处理；
- 后端以 RESTful API 提供服务，大量接口集中在 `/v1/backend/{module}/...` 路径；
- 接口模块包括：系统配置、用户认证、通用对象管理（如联系人、订单等）；
- 前端封装了请求 Hook（如 `useApi`，`usePaginatedApi`，`useSubmit`）来处理通用数据流；
- 强调了分页加载、表单提交、登录后保存用户信息、异常提示、Token刷新等流程。

## 前后端对接的衔接方式与具体步骤

### 通用对接流程（用于所有模块）

1. 定义接口结构：后端提供接口路径与参数说明；
2. 封装请求工具：前端封装统一 `request.ts` 请求函数，支持 token、错误拦截、缓存等；
3. 定义API函数：如 `getUserInfo`，`list(objname)`；
4. 在组件中调用API：通过 `useApi`，`useSubmit` 等 Hook 调用；
5. 响应渲染数据：前端处理响应数据并渲染页面；
6. 错误统一提示：如 Toast 弹窗显示失败原因。

## 流程图一：整体系统数据流流程

1 sequenceDiagram

2 participant U as 用户

3 participant F as 前端组件

4 participant R as 请求工具(request.ts)

5 participant B as 后端接口

6 U->>F: 发起操作 (如点击登录)

7 F->>R: 调用封装 API 函数

8 R->>B: 发送请求 (带Token)

9 B-->>R: 返回响应 (成功/失败)

10 R-->>F: 返回数据或错误

11 F-->>U: 渲染结果或提示错误

视图 | ?

```
sequenceDiagram
    participant U as 用户
    participant F as 前端组件
    participant R as 请求工具(request.ts)
    participant B as 后端接口
    U->>F: 发起操作 (如点击登录)
    F->>R: 调用封装 API 函数
    R->>B: 发送请求 (带Token)
    B-->>R: 返回响应 (成功/失败)
    R-->>F: 返回数据或错误
    F-->>U: 渲染结果或提示错误
```

**说明：**这个流程适用于所有模块，从用户操作出发，到后端响应，再返回给用户界面。

✔ 流程图二：接口行为流程

视图 | ?





输入代码即可展示图表

视图 | ?

输入代码即可展示图表

---

## 这张图帮助你 做到

-  一眼看清每一步谁负责什么（前端/后端）
  -  新手也知道怎么“对”起来：哪边先做、怎么调试
  -  每个环节都能单独复用 & 测试
  -  哪怕有多人协作也不乱——流程统一、职责清晰
- 

## 存客宝前后端接口完整对接流程（可视化）

☰ 视图 | ⓘ

输入代码即可展示图表

---

## 每一步详细解释（谁负责？做什么？）

步骤	说明	
1. 用户操作	点击/滑动/输入等交互	用户
2. 组件收集数据	onClick/onSubmit 收集用户输入	前端
3. Hook调用接口	使用 useApi 或 useSubmit	前端
4. 封装API调用	在 api/*.ts 中编写接口请求函数	前端
5. 封装请求中转	所有请求统一走 request.ts：加 token、异常提示、重定向	前端
6. 发出 fetch 请求	使用原生 fetch 请求，带上 headers 和 body	前端
7. 后端接收接口	Controller 接收参数并验证	后端
8. 调用 Service	实际执行业务逻辑，比如查询数据库	后端
9. 返回统一格式	{ code: 0, message: "成功", data: {...} }	后端
10. 前端处理结果	根据 code 统一处理 toast、跳转、缓存等	前端
11. 渲染页面	将结果展示给用户（如列表、表单）	前端

## 对接顺序建议（新人友好）

### 前端 → 先封装接口（mock接口也行）

1. 定义接口函数：`getUserInfo()`，`list(contact)`
2. 在组件中写 Hook 和 UI
3. 使用 request.ts 模拟返回数据（可配合 Mock 工具）

### 后端 → 提供接口文档 & 开发 Controller

1. 先写清楚接口文档（路径、参数、响应格式）
2. 开发接口并用 postman 测试返回结构
3. 确认响应结构与前端预期一致

### 联调阶段

- 前端调接口发现失败 => 检查 token / 参数名
- 后端返回结构前端不识别 => 对照 `{code, message, data}` 格式
- 异常 toast 不出现 => 看 request.ts 是否走了 catch 分支

为了帮助你基于核心代码与模块理解“存客宝”系统前后端对接的完整流程，我绘制了一张结构更清晰、从核心文件出发的可视化流程图，并补充了关键说明。

## 存客宝核心文件级的前后端对接流程图



## 对应关键文件与角色

步骤	文件/模块	描述
用户操作	页面组件	例如点击“登录”、“提交”、“查询”等按钮
API Hook	useApi.ts / useSubmit.ts	统一封装加载状态、错误提示和执行逻辑
API 函数	auth.ts, commonobject.ts, config.ts 等	每个模块一个 API 文件，组织清晰
请求封装	request.ts	自动添加 token、处理错误、封装响应
后端接口	Controller	路径如 /v1/backend/obj/list，接受请求
后端逻辑	Service	执行业务逻辑、访问数据库等
统一响应	所有接口	返回 {code, message, data} 结构，方便前端解析
数据渲染	组件中调用 setData()	将数据渲染为表格、表单、图表等 UI

## 示例对接方式：对象列表（list()）

### 1. 前端组件中调用

代码块

```
1  const { data, loading, fetchData } = usePaginatedApi(list, 10);
2  useEffect(() => {
3    fetchData(1, { keyword: "" }, "contact"); // 第1页, 查询联系人
4  }, []);
```

## 2. API 定义 (commonobject.ts)

代码块

```
1  export function list(data: any, objname: string) {
2    return request({
3      url:
4        /v1/backend/${objname}/list
5      ,
6      method: "post",
7      data
8    });
9  }
```

## 3. 后端接口 (Java 示例)

代码块

```
1  @PostMapping("/v1/backend/{objname}/list")
2  public Result<?> list(@PathVariable String objname, @RequestBody QueryParams
    params) {
3      List<?> data = objectService.queryList(objname, params);
4      return Result.success(data);
5  }
```

## ✓ 小结：可视化背后的对接原则

- **解耦**：所有请求和接口都通过统一 `request.ts` 处理，便于维护与扩展；
- **模块化**：每类功能（如用户、对象、配置）拆成单独 `api/xxx.ts` 文件；
- **统一结构**：前后端都使用 `{ code, message, data }` 结构，提升容错能力；
- **可复用**：前端组件只需调用 Hook 和 API，不需要关心底层实现；
- **强兼容**：适配新对象/新功能，只需添加一个 `objname`，不用写新接口。

✓ 登录流程图（飞书 Mermaid 文本绘图可用）

输入代码即可展示图表

下面是登录界面的完整流程图，基于你“存客宝”项目的架构逻辑，从用户点击登录按钮开始，到后端返回 token 并完成登录流程结束。

该流程图已经适配飞书文本绘图，✅ 无特殊符号冲突，✅ 新手也可直接参考执行。

## 🧠 补充说明：关键代码 & 接口

### ✅ 前端 login 函数 (login.ts)

代码块

```
1 export function login(data: any) {  
2   return request({  
3     url: "/v1/backend/user/login",  
4     method: "post",  
5     data  
6   });  
7 }
```

### ✅ 前端处理响应

代码块

```
1 const res = await login({ phone, code });  
2 if (res.code === 0) {  
3   localStorage.setItem("token", res.data.token);  
4   navigate("/home");  
5 }
```

## ✓ 后端接口逻辑（伪代码）

代码块

```
1  @PostMapping("/v1/backend/user/login")
2  public Result login(@RequestBody LoginDto dto) {
3      if (!smsService.verify(dto.getPhone(), dto.getCode())) return
        Result.fail("验证码错误");
4      User user = userService.getOrCreate(dto.getPhone());
5      String token = jwtService.createToken(user);
6      return Result.success(Map.of("token", token, "userInfo", user));
7  }
```

## ✓ 登录流程的可视化价值

模块	作用
表单校验	避免无效请求
请求封装	统一处理 token、loading、异常等
后端逻辑	解耦登录流程 + 用户注册
状态保存	登录后立即设置 token、跳转首页、初始化全局状态

下面是你要的「**场景获客创建计划流程图**」，结合“存客宝”的业务逻辑与模块结构，设计出一套适合你当前团队使用、并能直接落地执行的**计划创建流程图**。该流程涵盖：

- 用户创建计划的操作步骤
- 前端与后端之间的交互对接
- 各节点逻辑处理与数据流向

## ✓ 场景获客 - 创建计划流程图



输入代码即可展示图表

### 🧠 对应模块与说明

阶段	描述	文件 / 模块
表单填写	用户输入计划信息	CreatePlanForm.tsx
表单校验	使用 yup 或前端字段规则校验	前端组件
保存接口	savePlan(data) API 函数调用	plan.ts
封装处理	request.ts 拼 token + 发请求	通用请求工具
后端逻辑	接收 & 验证 & 落库	/v1/backend/plan/save
统一响应	返回 planId、状态码等	Result 封装
结果处理	弹出 toast / 跳转详情页	组件逻辑

### ✅ 示例：前端 API 封装

代码块

```
1 export function savePlan(data: any) {
2   return request({
3     url: "/v1/backend/plan/save",
4     method: "post",
5     data
6   });
7 }
```

## ✓ 示例：后端 Java 接口伪代码

代码块

```
1  @PostMapping("/v1/backend/plan/save")
2  public Result<?> savePlan(@RequestBody PlanDto plan) {
3      if (!authService.checkUser()) return Result.fail("无权限");
4      Long id = planService.save(plan);
5      return Result.success(Map.of("planId", id));
6  }
```

---