以下是帮助 AI 高效理解需求并将其转化为代码的 7 **大核心方法**，结合技术细节与工程实践，确保需求到代码的精准映射：

一、需求结构化：用 AI 可解析的「数据模型」定义需求

1. 字段级标准化描述

- **格式**：使用 **表格/JSON 结构化字段**，明确每个字段的 **名称、类型、枚举值、业务含义**（如数据库表结构）

```
{
"字段名": "status",
"类型": "int",
"枚举值": "10-刚获取, 20-初筛与测评, ...",
"说明": "候选人阶段状态，决定数据可见性与操作权限"
}
```

- **示例**：在技术需求中，将「阶段状态值」「技能测试状态」等枚举值按文档表格逐条列出，避免自然语言歧义。

2. 流程逻辑模块化

- **步骤拆分**：将复杂逻辑拆解为 **原子操作**（如"添加候选人"拆分为"文件解析→字段校验→数据库写入"）
- **状态机定义**：用流程图或伪代码描述状态转换规则（如"技能成绩≥240 + 口语≥B级 → 阶段自动转为预录取"）

二、技术细节显性化：明确「实现边界」与「依赖条件」

1. 技术栈强约束

- **明确框架/工具**：指定开发语言、框架、版本（如"前端：React 18 + Next.js 13，后端：Java 17 + Spring Boot 3.2"）
- **接口契约定义**：用 **OpenAPI/Swagger 格式** 描述 API 细节（请求方法、参数格式、响应码）

```
/api/candidate/assign:
post:
summary: 分配对接人
parameters:
- name: docking_person_id
type: long
required: true
description: 对接人用户ID（通过用户中心API校验合法性）
```

2. 数据库表结构前置

- DDL **脚本优先**：提供完整表结构（字段、约束、注释），如：

```
CREATE TABLE crm_candidate_record (
id BIGINT PRIMARY KEY COMMENT '主键ID',
phone VARCHAR(16) UNIQUE COMMENT '手机号（唯一索引，添加时查重）',
status INT COMMENT '阶段状态（10-80，见需求文档枚举值）'
);
```

三、示例引导：用「输入-输出对」降低理解成本

1. 业务场景示例

- **给出具体用例**：描述用户操作路径及预期结果（如"业务员登录后，列表页仅显示 creator=当前用户ID 的数据"）
- **异常场景覆盖**：明确边界条件（如"上传重复手机号时，返回错误码 400，提示'手机号已注册'"）

2. 代码模板引导

- **提供骨架代码**：针对常见功能（如分页、权限过滤）给出代码框架，减少 AI 自由发挥空间

```
// 列表页数据查询（伪代码）
public List<Candidate> listCandidates(String role, Long userId) {
// 权限过滤逻辑
String permissionCondition = switch(role) {
case "业务员" -> "creator = " + userId;
case "服务专家" -> "creator = " + userId + " OR status >= 40";
default -> ""; // 管理员无限制
};
// 组合查询条件（关键词、时间范围等）
String sql = "SELECT * FROM crm_candidate_record WHERE " + permissionCondition +
otherConditions;
return jdbcTemplate.query(sql, Candidate::map);
}
```

四、分阶段拆解：从「需求」到「伪代码」再到「可执行代码」

1. 先定义「非技术化逻辑」

- **用自然语言描述核心流程**（不涉及代码），如：“当用户点击‘安排测评’按钮时，系统需校验当前阶段是否为‘刚获取’，若是则弹出课程选择框，提交后更新技能测试状态为‘待安排’，并记录测评时间。”

2. 转化为「技术伪代码」

- **用半代码化语言明确交互**：

```
if (currentStatus == 10) { // 刚获取阶段
showModal(测评课程选择);
onSubmit(selectedCourse) {
callAPI("/api/skill-test/schedule", {
candidateId,
courseId,
status: 10 // 待安排
});
updateStageTo初筛与测评(); // 触发状态转换规则
}
} else {
showToast("当前阶段无法安排测评");
}
```

3. 细化为「可执行代码片段」

- **指定具体组件/函数**：如“`showModal` 调用 Ant Design 的 Modal 组件，`callAPI` 使用 Axios 封装的请求函数”

五、约束条件显性化：明确「禁止行为」与「必选功能」

1. 强制约束说明

- **必选功能**：用 “**必须**”“**禁止**” 等关键词明确要求（如“必须实现手机号查重功能，禁止跳过该校验直接保存”）
- **性能指标**：指定响应时间（如“列表页查询接口响应时间需≤500ms”）、并发量（如“支持 100 并发分配操作”）

2. 兼容性说明

- **浏览器/环境适配**：明确支持的浏览器版本（如“支持 Chrome 100+、Firefox 98+”）、数据库版本（如“MySQL 8.0 及以上”）

六、交互迭代：通过「多轮对话」渐进式澄清需求

1. 第一轮：需求概述

- **提供全景图**：简要描述系统目标（如“开发候选人管理系统，实现从简历录入到就业的全流程管理”）
- **关键输出**：让 AI 先输出 **功能模块列表** 或 **ER 图**，验证是否理解核心实体（候选人、对接人、跟进记录）

2. 第二轮：细节填充

- **针对模糊点追问**：如“‘服务专家可查看正式录取后的数据’，这里的‘正式录取’对应状态码是多少？请确认是否与文档中的 status=40 一致”
- **要求输出中间产物**：让 AI 先生成 **数据库 ER 图** 或 **组件架构图**，确认数据关系与模块划分

3. 第三轮：边界校验

- **提供反例测试**：如“若用户同时属于业务员和服务专家（双重角色），数据权限如何处理？”
- **要求代码自测逻辑**：让 AI 在生成代码时同步添加 **单元测试用例**（如权限校验函数的测试场景）

七、验证与校准：用「测试用例」确保代码符合预期

1. 提前定义验收标准

- **功能测试用例**：按 “**输入-操作-预期输出**” 格式编写（如：

- 输入：角色=业务员，创建候选人A
- 操作：登录后查询列表页
- 预期：仅显示候选人A的数据）

2. 要求代码包含注释与文档

- **强制注释规范**：关键逻辑必须添加注释（如状态转换规则、权限过滤条件）
- **生成接口文档**：让 AI 在输出代码时同步生成 **Swagger 文档** 或 **Postman Collection**，确保接口可调试

总结：AI 高效实现需求的核心原则

1. **结构化优先**：用表格、JSON、DDL 等机器友好格式替代纯文本描述
2. **细节无歧义**：枚举值、状态码、接口参数必须与需求文档完全对齐
3. **分阶引导**：从业务逻辑→伪代码→可执行代码，逐步降低 AI 的理解难度
4. **验证闭环**：通过测试用例、中间产物校验（ER图、组件图）确保需求落地 通过以上方法，可将需求转化为 AI 能准确解析的"工程指令"，大幅减少沟通成本与代码返工，尤其适用于复杂业务系统（如 CRM、项目管理平台）的开发。