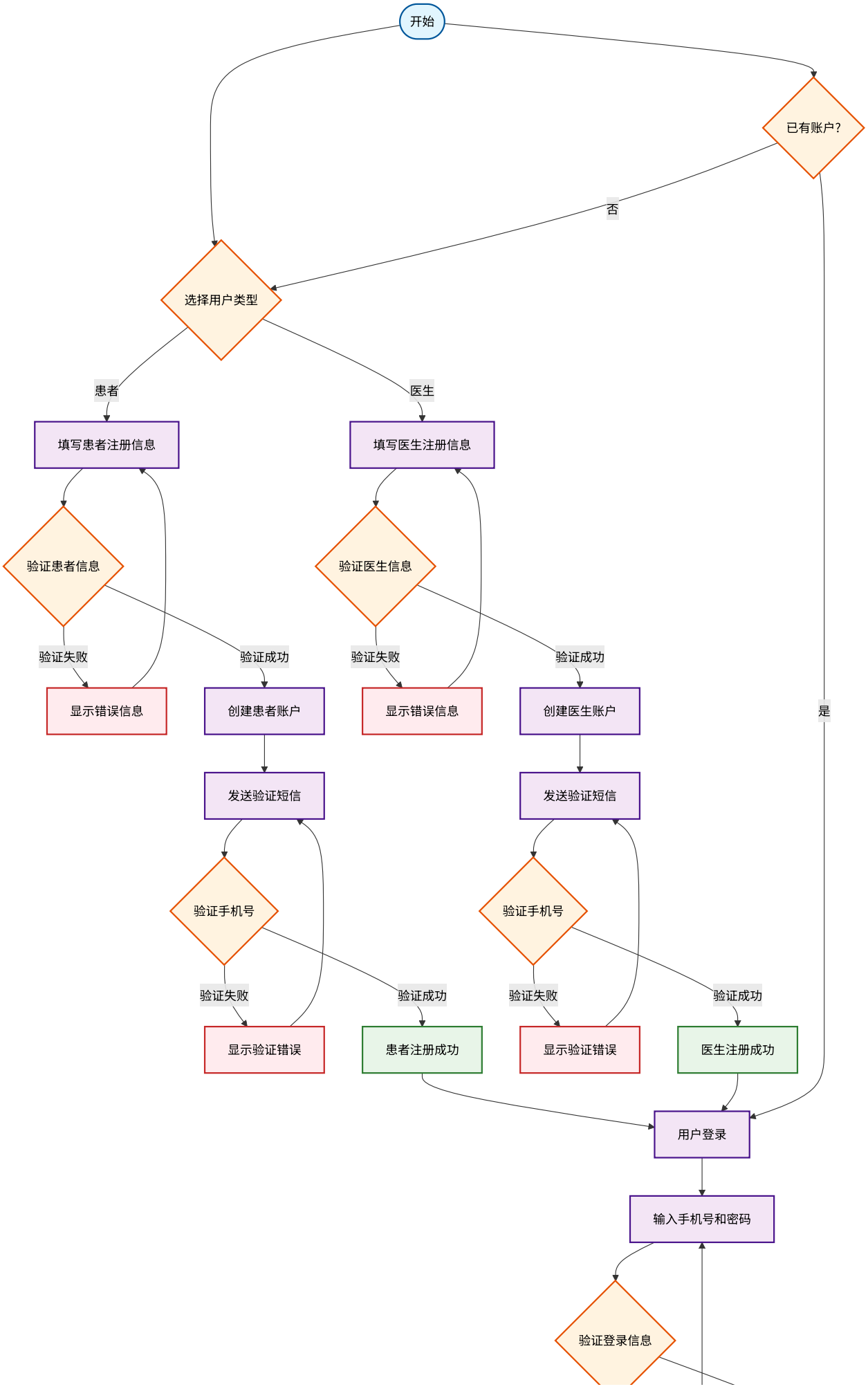
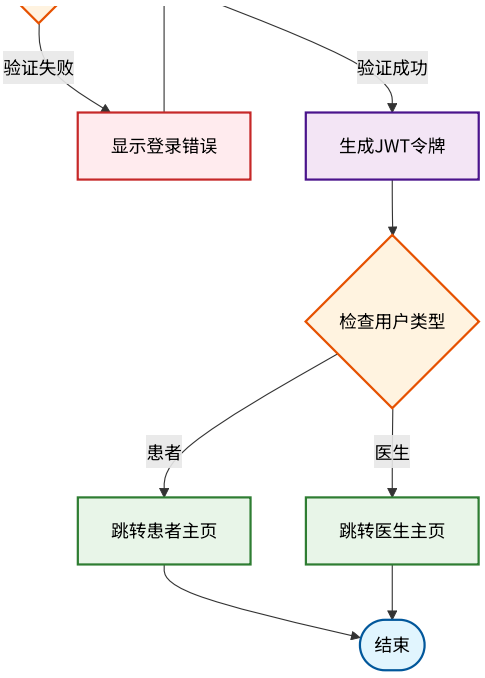
活动图

1. 用户注册和登录流程


开始
已有账户？
否
是
选择用户类型
患者
医生
填写患者注册信息
填写医生注册信息
验证患者信息
验证医生信息
验证失败
验证成功
验证失败
验证成功
显示错误信息
创建患者账户
显示错误信息
创建医生账户
发送验证短信
发送验证短信
验证手机号
验证手机号
验证失败
验证成功
验证失败
验证成功
显示验证错误
患者注册成功
显示验证错误
医生注册成功
用户登录
输入手机号和密码
验证登录信息



业务流程说明

这个流程图展示了医疗系统中患者和医生的注册登录完整流程。系统支持两种用户类型的注册，并通过手机号验证确保账户安全性。

关键技术实现

用户模型设计 (models.py):

```
class PatientModel(db.Model):
    __tablename__ = 'patient'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    phone = db.Column(db.String(11), nullable=False, unique=True)
    name = db.Column(db.String(50), nullable=False)
    password = db.Column(db.String(50), nullable=False)
    create_time = db.Column(db.DateTime, nullable=False, default=datetime.now)
    update_time = db.Column(db.DateTime, nullable=False, default=datetime.now,
onupdate=datetime.now)

class DoctorModel(db.Model):
    __tablename__ = 'doctor'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    phone = db.Column(db.String(11), nullable=False, unique=True)
    name = db.Column(db.String(50), nullable=False)
    password = db.Column(db.String(50), nullable=False)
    create_time = db.Column(db.DateTime, nullable=False, default=datetime.now)
```

```
    update_time = db.Column(db.DateTime, nullable=False, default=datetime.now,
onupdate=datetime.now)
```

手机号验证功能 (utils.py):

```
def validate_phone_number(phone):
    """简化版手机号验证（仅限国内）"""
    pattern = r'^1[3-9]\d{9}$'  # 严格的11位国内手机号
    return re.match(pattern, phone) is not None
```

统一响应格式 (utils.py):

```
class Result:
    def __init__(self, code, msg, data):
        self.code = code
        self.msg = msg
        self.data = data

    @staticmethod
    def success(data=None):
        return Result(1, "", data)

    @staticmethod
    def error(msg):
        return Result(0, msg, None)
```

登录视图对象 (vo.py):

```
class LoginVO:
    def __init__(self, id, name, role, token, avatar_url):
        self.id = id
        self.name = name
        self.role = role
        self.token = token
        self.avatar_url = avatar_url
    def to_dict(self):
        return {"id": self.id, "name": self.name, "role": self.role,
"avatar_url": self.avatar_url, "token": self.token}
```

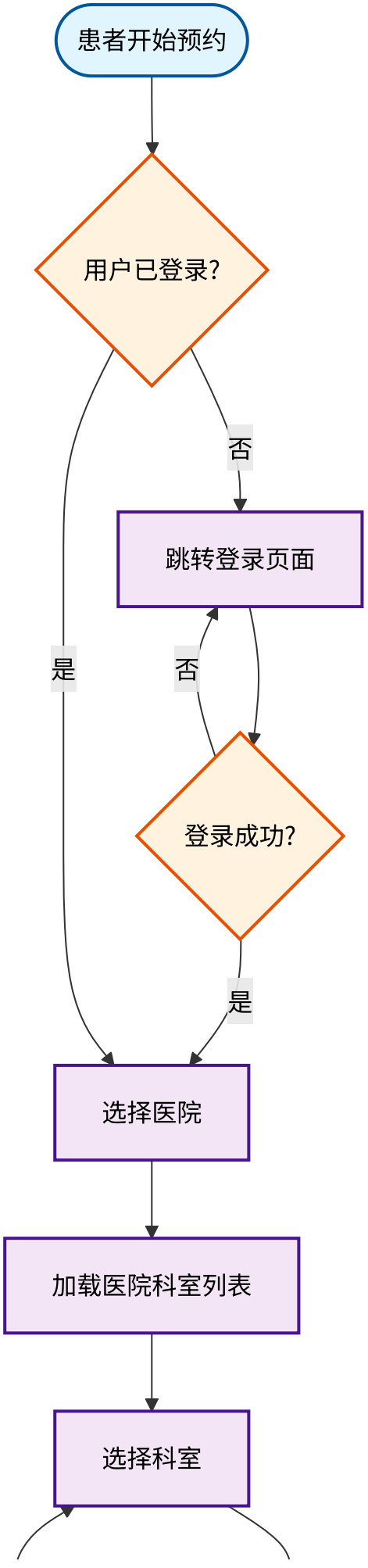
JWT配置 (config.py):

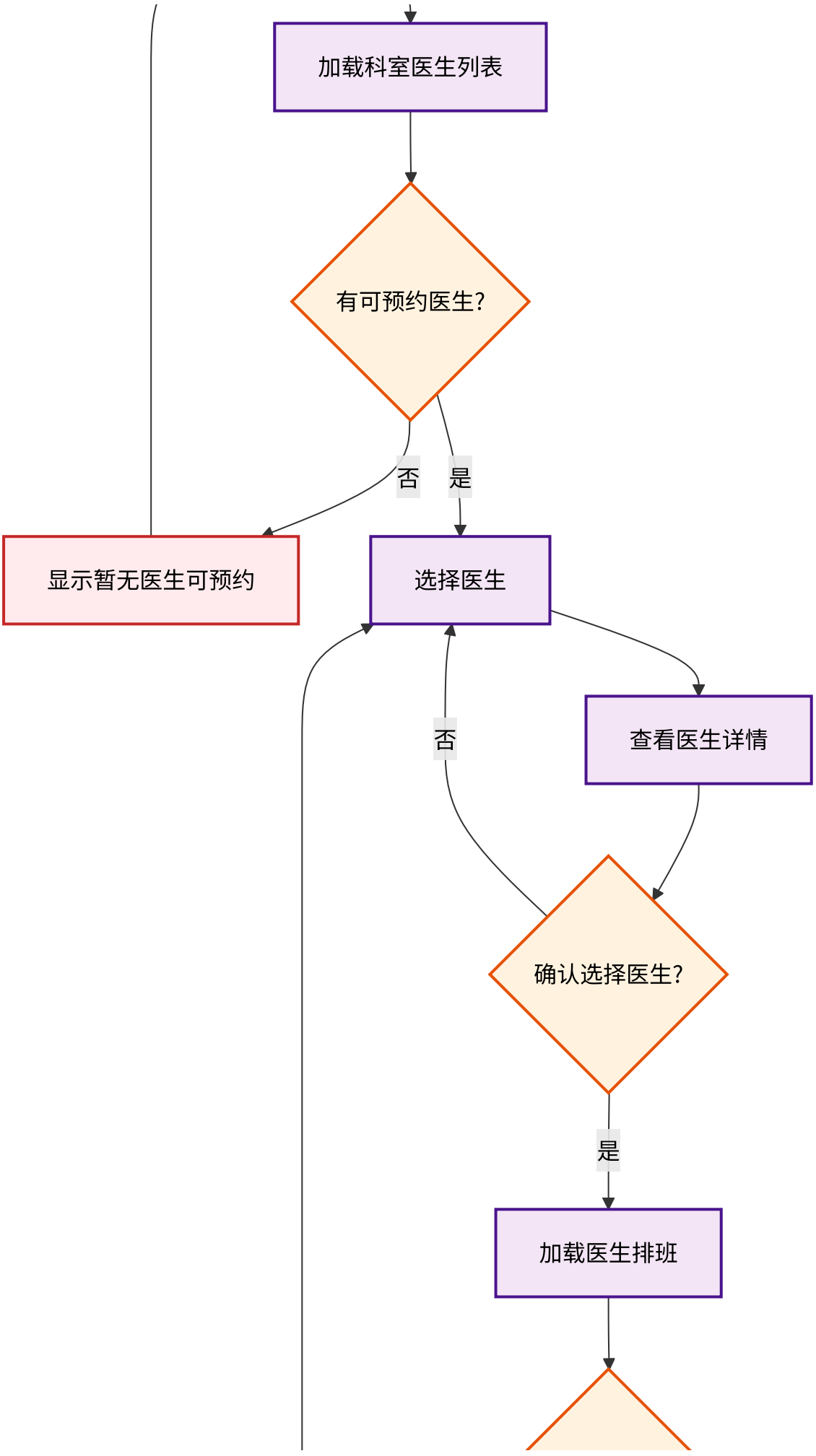
```
class Config:
    JWT_SECRET_KEY = os.getenv('JWT_SECRET_KEY', 'secret string')
    JWT_TOKEN_LOCATION = os.getenv('JWT_TOKEN_LOCATION', 'headers')
    JWT_ACCESS_TOKEN_EXPIRES = int(os.getenv('JWT_ACCESS_TOKEN_EXPIRES', 3600))
 # 1 hour
```

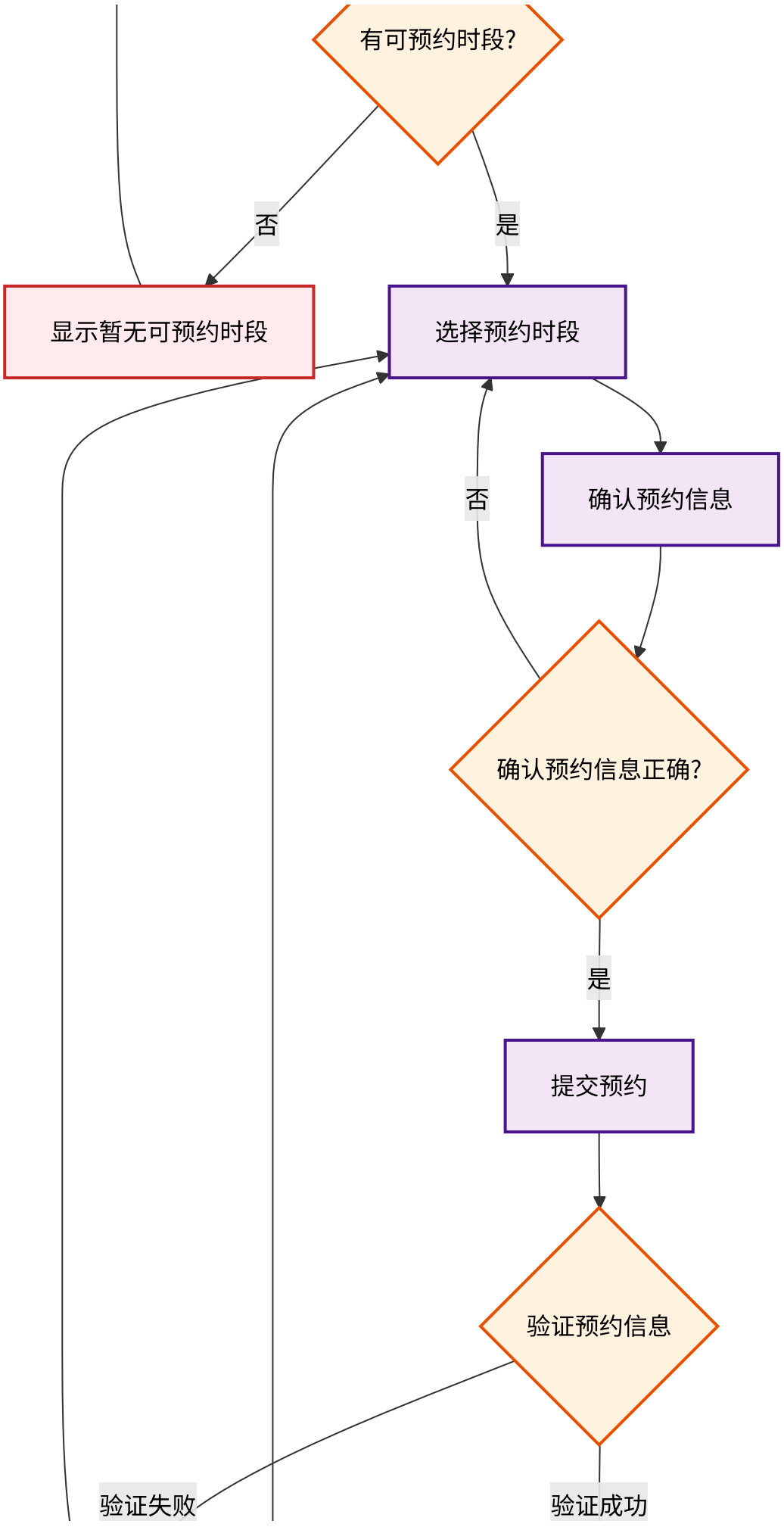
流程特点分析

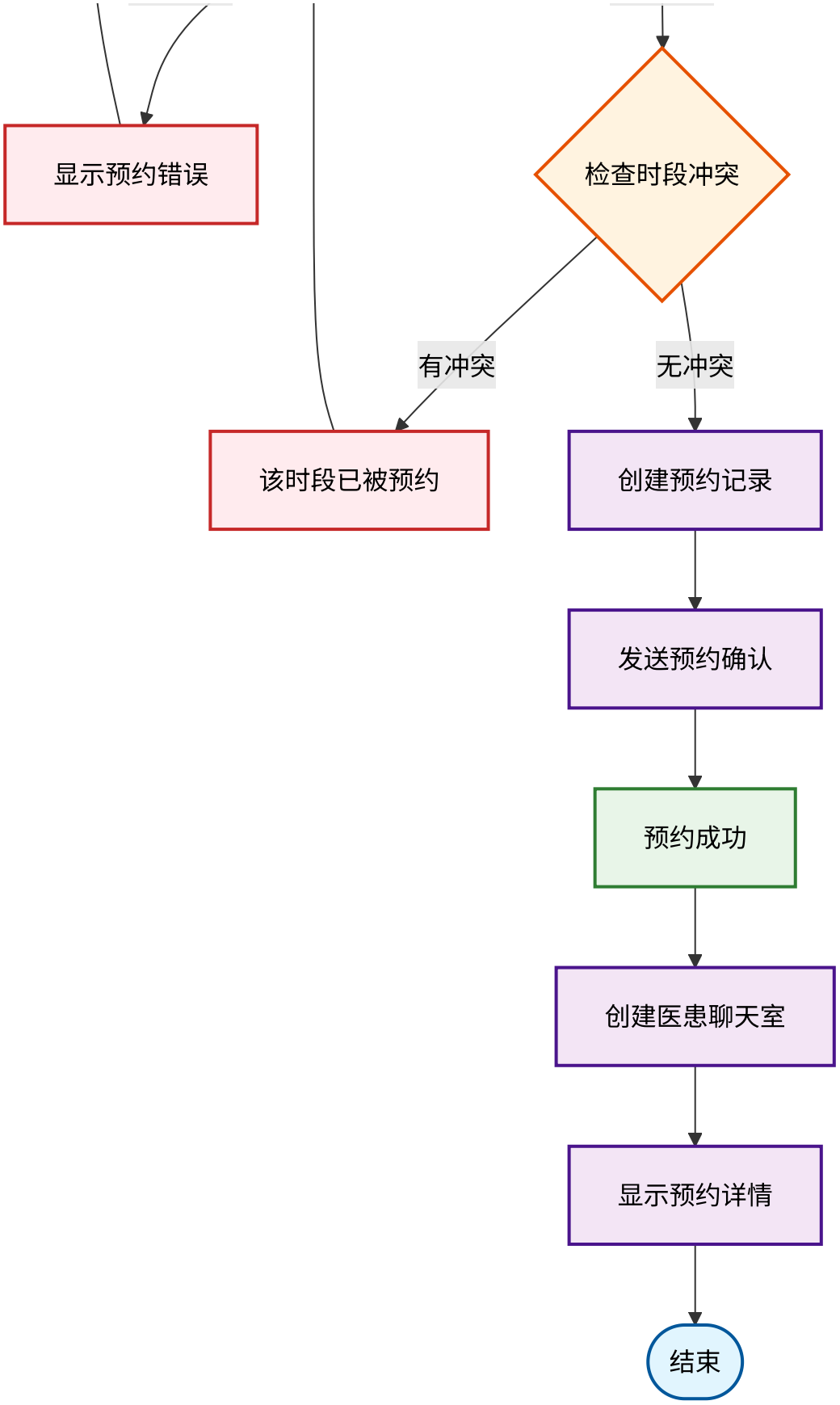
1. **双重验证机制**: 先验证基本信息格式，再通过SMS验证手机号真实性
2. **用户类型区分**: 患者和医生使用相同的验证流程但创建不同的账户类型
3. **安全性设计**: 密码存储、JWT令牌生成、手机号唯一性约束
4. **错误处理**: 每个验证环节都有对应的错误处理和用户提示

2. 患者预约挂号流程


患者开始预约
用户已登录?
否
跳转登录页面
是
否
登录成功?
是
选择医院
加载医院科室列表
选择科室


加载科室医生列表
有可预约医生?
否
是
显示暂无医生可预约
选择医生
否
查看医生详情
确认选择医生?
是
加载医生排班


有可预约时段?
否
是
显示暂无可预约时段
选择预约时段
确认预约信息
否
确认预约信息正确?
是
提交预约
验证预约信息
验证失败
验证成功


显示预约错误
检查时段冲突
有冲突
无冲突
该时段已被预约
创建预约记录
发送预约确认
预约成功
创建医患聊天室
显示预约详情
结束

业务流程说明

该流程实现了患者在线预约医生的完整业务逻辑，从医院选择到预约确认，确保预约时段不冲突并建立医患沟通渠道。

关键技术实现

医院和科室模型 (models.py):

```
class HospitalModel(db.Model):
    __tablename__ = 'hospital'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(100), unique=True, nullable=False)
    departments = db.relationship('DepartmentModel', backref='hospital',
lazy='dynamic')

class DepartmentModel(db.Model):
    __tablename__ = 'department'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(50), nullable=False)
    description = db.Column(db.Text)
    hospital_id = db.Column(db.Integer, db.ForeignKey('hospital.id'),
nullable=True)
    doctors = db.relationship('DoctorInfoModel', backref='department',
lazy='dynamic')
```

医生详细信息和排班 (models.py):

```
class DoctorInfoModel(db.Model):
    __tablename__ = 'doctor_info'
    id = db.Column(db.Integer, db.ForeignKey('doctor.id'),primary_key=True)
    hospital_id = db.Column(db.Integer, db.ForeignKey('hospital.id'),
nullable=True)
    department_id = db.Column(db.Integer, db.ForeignKey('department.id'))
    schedule = db.Column(db.JSON, nullable=True, default=lambda: {
        "monday": {"morning": False, "afternoon": False},
        "tuesday": {"morning": False, "afternoon": False},
        "wednesday": {"morning": False, "afternoon": False},
        "thursday": {"morning": False, "afternoon": False},
        "friday": {"morning": False, "afternoon": False}
    })
```

预约挂号模型 (models.py):

```
class RegistrationModel(db.Model):
    __tablename__ = 'registration'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    patient_id = db.Column(db.Integer, db.ForeignKey('patient_info.id'),
nullable=False)
    doctor_id = db.Column(db.Integer, db.ForeignKey('doctor_info.id'),
nullable=False)
    date = db.Column(db.Date, nullable=False, default=datetime.now)
    time_slot = db.Column(db.String(50), nullable=False)  # 时间段，例如
"morning", "afternoon"

    def to_dict(self):
        return {
            "patient": self.patient.name,
            "phone": self.patient.phone,
            "doctor": self.doctor.name,
            "hospital": self.doctor.hospital.name if self.doctor.hospital else
None,
            "department": self.doctor.department.name if self.doctor.department
else None,
            "date": self.date.isoformat(),
            "time_slot": self.time_slot
        }
```

聊天室模型 (models.py):

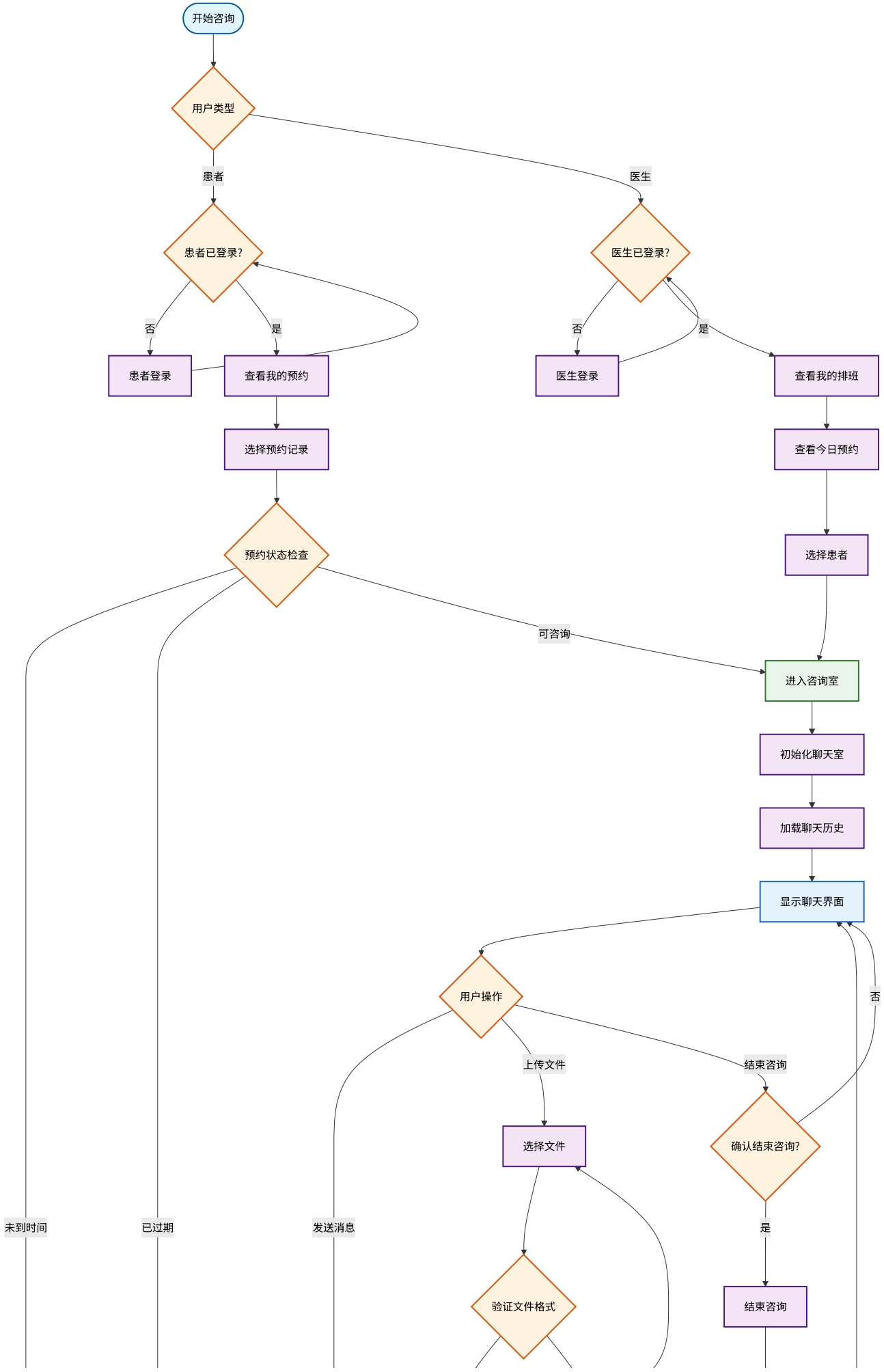
```
class RoomModel(db.Model):
    __tablename__ = 'room'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    doctor_id = db.Column(db.Integer, db.ForeignKey('doctor.id'),
nullable=False)
    patient_id = db.Column(db.Integer, db.ForeignKey('patient.id'),
nullable=False)
```

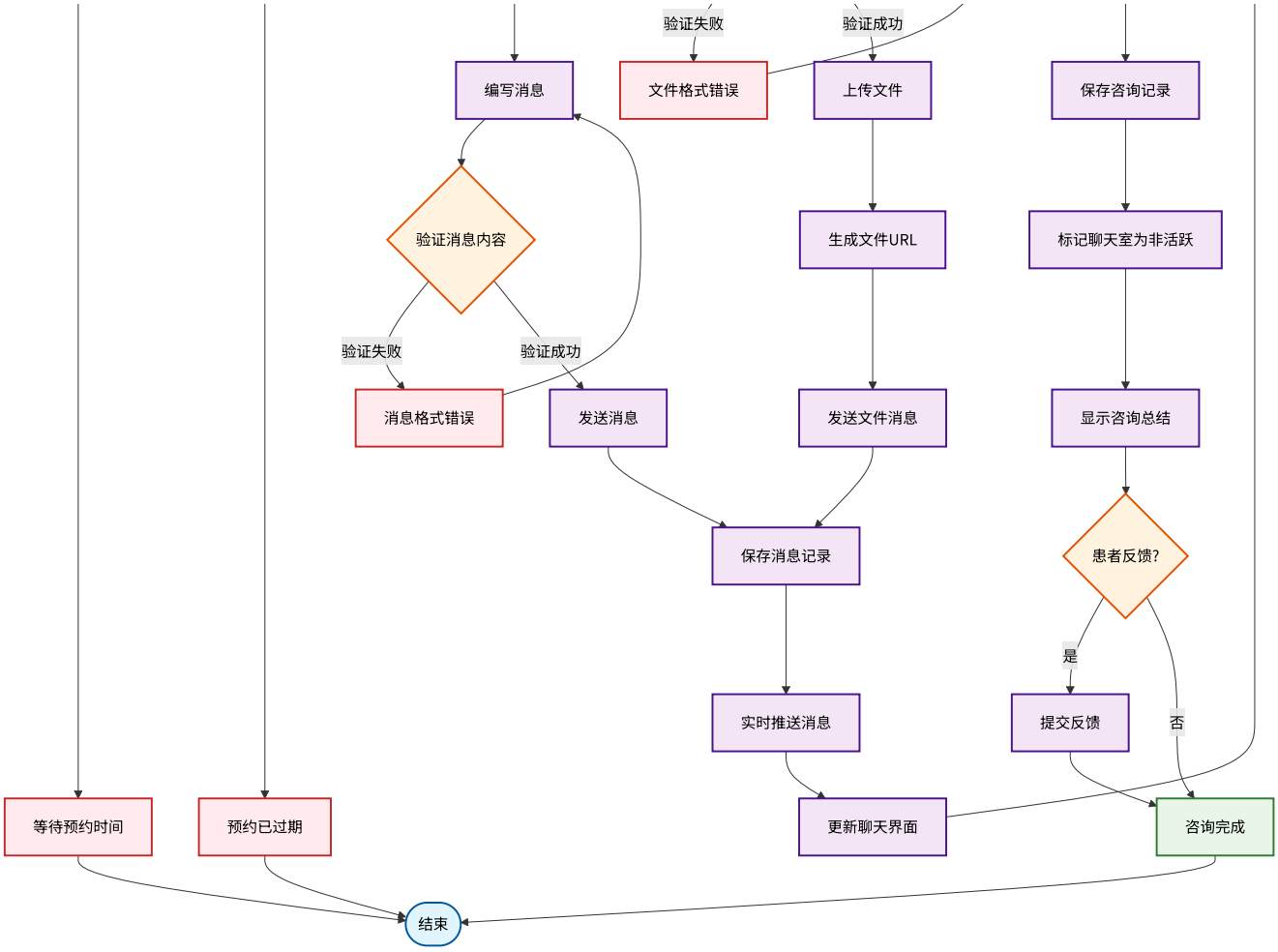
流程特点分析

1. **层次化选择**: 医院→科室→医生的三级选择结构，符合医疗机构组织架构
2. **实时可用性检查**: 动态加载医生排班信息，确保显示的时段真实可预约
3. **冲突检测机制**: 在提交预约前检查时段是否已被占用，避免重复预约

4. **自动化流程**: 预约成功后自动创建医患聊天室，为后续咨询做准备

3. 医患在线咨询流程


开始咨询
用户类型
患者
医生
患者已登录?
医生已登录?
否
是
否
是
患者登录
查看我的预约
医生登录
查看我的排班
选择预约记录
查看今日预约
预约状态检查
选择患者
可咨询
进入咨询室
初始化聊天室
加载聊天历史
显示聊天界面
用户操作
否
上传文件
结束咨询
选择文件
确认结束咨询?
未到时间
已过期
发送消息
是
验证文件格式
结束咨询



业务流程说明

这个流程实现了基于预约的实时医患在线咨询功能，支持文本消息和文件传输，并包含完整的咨询记录管理。

关键技术实现

消息模型 (models.py):

```
class MessageModel(db.Model):
    __tablename__ = 'message'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    from_user = db.Column(db.String(50), nullable=False)
    from_user_avatar = db.Column(db.String(255), nullable=True)
    to_user = db.Column(db.String(50), nullable=False)
    to_user_avatar = db.Column(db.String(255), nullable=True)
    content = db.Column(db.Text, nullable=False)
    time = db.Column(db.DateTime, nullable=False, default=datetime.now)
    read = db.Column(db.Integer, nullable=False, default=False)
```

```
    type = db.Column(db.String(50), nullable=False)

    def to_dict(self):
        return {
            "from_user": self.from_user,
            "from_user_avatar": self.from_user_avatar,
            "to_user": self.to_user,
            "to_user_avatar": self.to_user_avatar,
            "content": self.content,
            "time": self.time.isoformat(),
            "read": bool(self.read),
            "type": self.type
        }
```

WebSocket实时通信 (consultant.py):

```
from flask_socketio import join_room, emit
from exts import socketio, db

@socketio.on('connect')
def connect():
    print("Client connected")

@socketio.on('disconnect')
def disconnect():
    print("disconnected")

@socketio.on('sendMessage')
def message(data):
    from_user = data['from_user']
    from_user_avatar = data['from_user_avatar']
    to_user = data['to_user']
    to_user_avatar = data['to_user_avatar']
    content = data['content']
    read = 0
    type = data['type']
    try:
        message = MessageModel(from_user=from_user,
from_user_avatar=from_user_avatar,
                               to_user=to_user, to_user_avatar=to_user_avatar,
                               content=content, read=read, type=type)
        db.session.add(message)
        db.session.commit()
        emit('newMessage', message.to_dict(), broadcast=True)
```

```
    except Exception as e:
        print("Error saving message:", e)
        db.session.rollback()
```

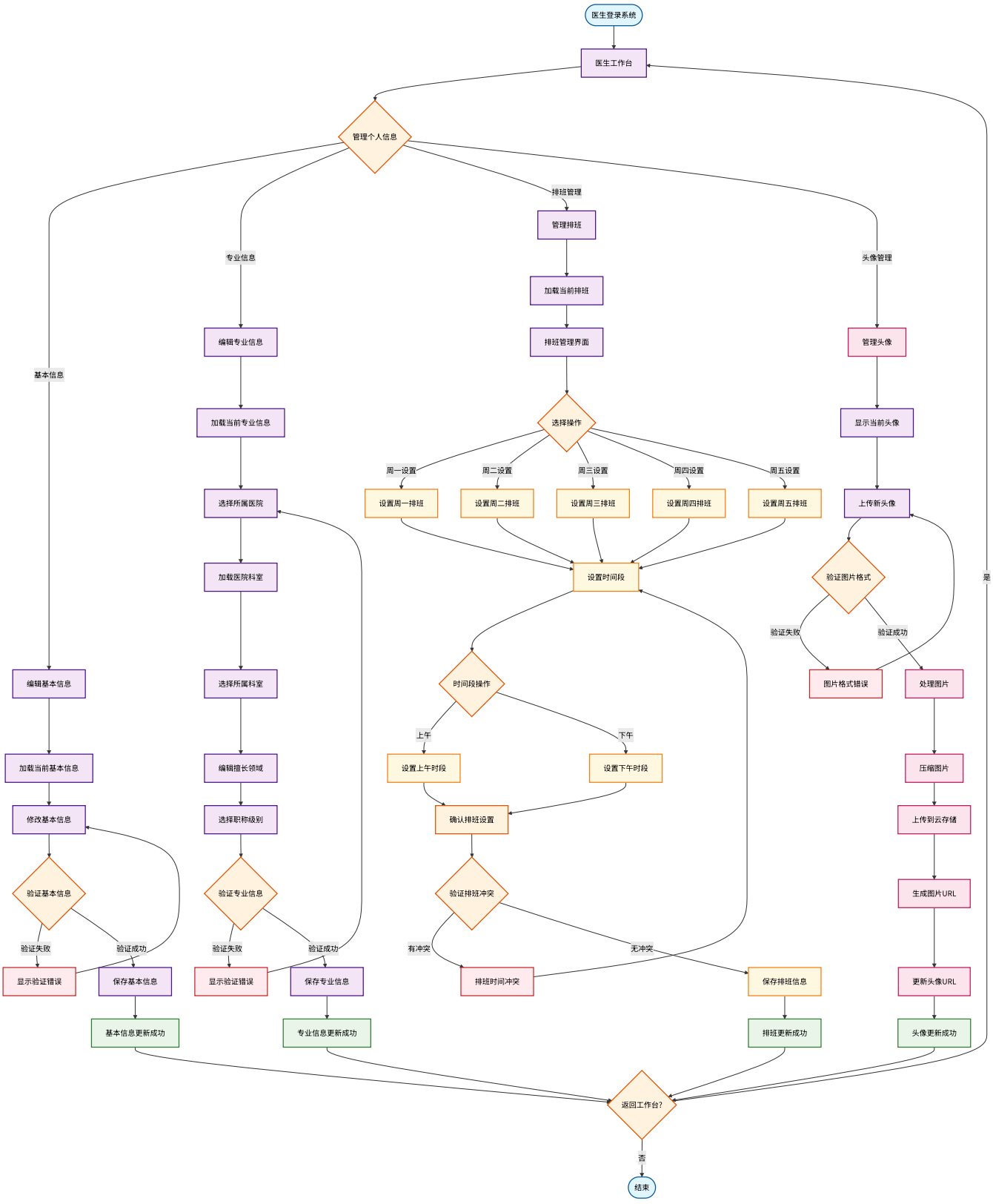
文件上传功能 (utils.py):

```
def upload_file(file_object, file_name):
    access_key_id = os.getenv('OSS_ACCESS_KEY_ID')
    access_key_secret = os.getenv('OSS_ACCESS_KEY_SECRET')
    endpoint = os.getenv('OSS_ENDPOINT')
    bucket_name = os.getenv('OSS_BUCKET_NAME')

    auth = oss2.Auth(access_key_id, access_key_secret)
    bucket = oss2.Bucket(auth, endpoint, bucket_name)

    try:
        bucket.put_object(file_name, file_object.stream)
    except oss2.exceptions.OssError as e:
        raise e
    except Exception as e:
        raise e

    file_url = f"https://{bucket_name}.{endpoint}/{file_name}"
    return file_url
```

流程特点分析

1. **预约制咨询**: 基于已有预约记录进行咨询，确保服务的有序性
2. **实时通信**: 使用WebSocket技术实现实时消息推送
3. **多媒体支持**: 支持文本消息和文件上传，满足医疗咨询的多样化需求
4. **数据持久化**: 所有聊天记录保存到数据库，便于后续查询和管理

4. 医生个人信息管理流程



业务流程说明

医生可以通过该流程管理个人基本信息、专业信息、工作排班和头像等，确保患者能够获取准确的医生信息。

关键技术实现

医生详细信息模型 (models.py):

```
class DoctorInfoModel(db.Model):
    __tablename__ = 'doctor_info'
    id = db.Column(db.Integer, db.ForeignKey('doctor.id'),primary_key=True)
    phone = db.Column(db.String(11), unique=True, nullable=False)
    name = db.Column(db.String(50), nullable=False)
    gender = db.Column(db.Enum('male', 'female'), nullable=True)
    hospital_id = db.Column(db.Integer, db.ForeignKey('hospital.id'),
nullable=True)
    internal_id = db.Column(db.String(8), nullable=True)
    department_id = db.Column(db.Integer, db.ForeignKey('department.id'))
    position_rank = db.Column(db.Enum(
        'resident',  # 住院医师
        'attending',  # 主治医师
        'associate_chief',  # 副主任医师
        'chief'  # 主任医师
    ), nullable=True)
    specialty = db.Column(db.Text)
    birth_date = db.Column(db.Date)
    avatar_url = db.Column(db.String(255))
    schedule = db.Column(db.JSON, nullable=True, default=lambda: {
        "monday": {"morning": False, "afternoon": False},
        "tuesday": {"morning": False, "afternoon": False},
        "wednesday": {"morning": False, "afternoon": False},
        "thursday": {"morning": False, "afternoon": False},
        "friday": {"morning": False, "afternoon": False}
    })

    def to_dict(self):
        return {
            "id": self.id,
            "phone": self.phone,
            "name": self.name,
```

```
            "gender": self.gender,
            "hospital_id": self.hospital_id,
            "hospital_name": self.hospital.name if self.hospital else None,
            "department_id": self.department_id,
            "department_name": self.department.name if self.department else
None,
            "position_rank": self.position_rank,
            "specialty": self.specialty,
            "avatar_url": self.avatar_url,
            "schedule": self.schedule,
        }
```

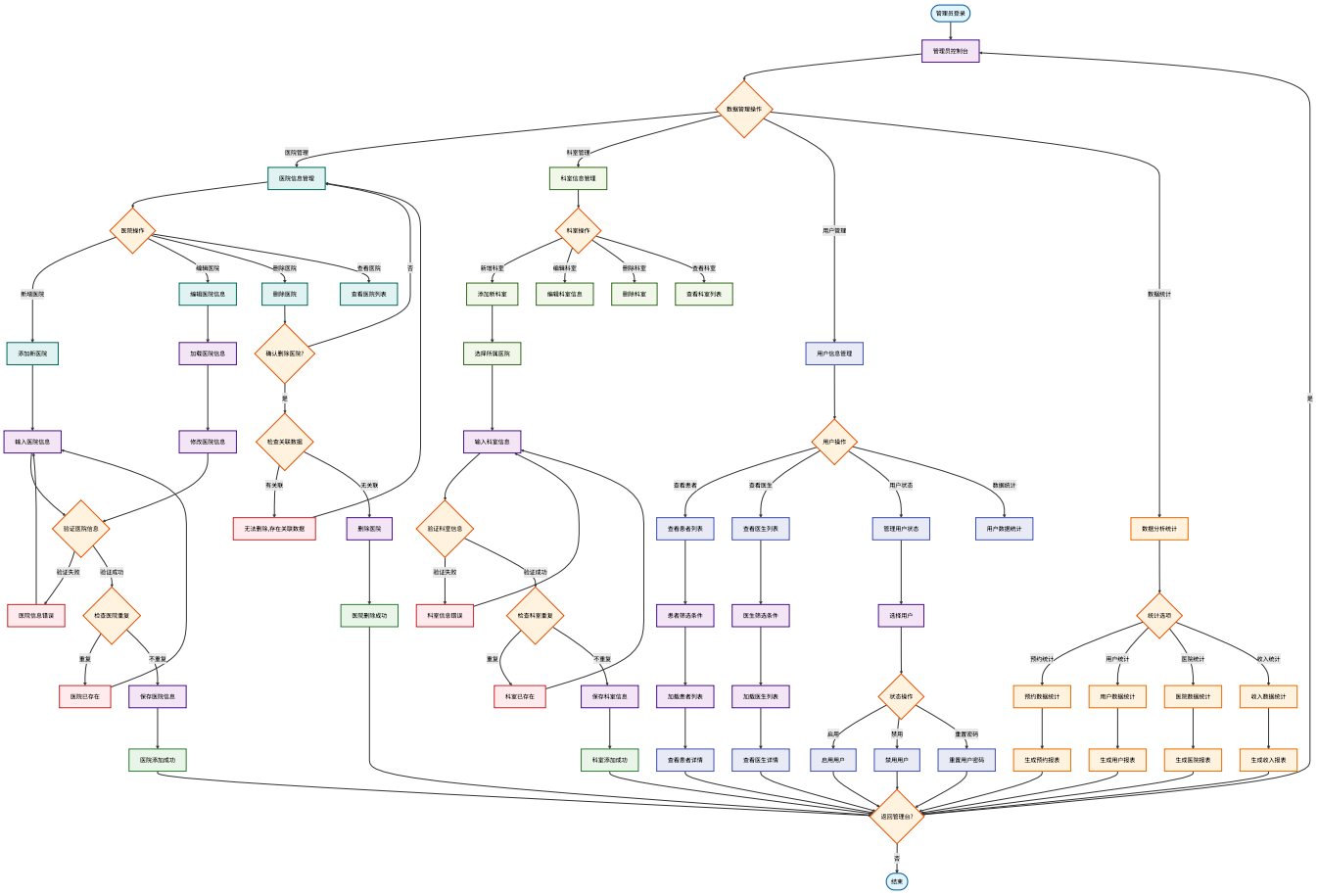
医生信息视图对象 (vo.py):

```
class DoctorInfoVO:
    def __init__(self, phone, name, gender, hospital, department, internal_id,
position_rank, specialty, birth_date, avatar_url, schedule):
        self.phone = phone
        self.name = name
        self.gender = gender
        self.hospital = hospital
        self.internal_id = internal_id
        self.department = department
        self.position_rank = position_rank
        self.specialty = specialty
        self.birth_date = birth_date
        self.avatar_url = avatar_url
        self.schedule = schedule
    def to_dict(self):
        return {"phone": self.phone,
                "name": self.name,
                "gender": self.gender,
                "hospital_id": self.hospital,
                "department_id": self.department,
                "internal_id": self.internal_id,
                "position_rank": self.position_rank,
                "specialty": self.specialty,
                "birth_date": self.birth_date,
                "avatar_url": self.avatar_url,
                "schedule": self.schedule}
```

流程特点分析

1. **分类管理**: 将医生信息分为基本信息、专业信息、排班信息和头像四个模块
2. **关联数据验证**: 医院和科室信息需要关联验证，确保数据一致性
3. **排班冲突检测**: 防止医生设置冲突的工作时间
4. **云存储集成**: 头像上传使用阿里云OSS服务，支持图片压缩和CDN加速

5. 系统管理员数据管理流程



业务流程说明

系统管理员通过该流程可以管理医院、科室等基础数据，监控用户状态，并生成各类统计报表，确保系统的正常运行。

关键技术实现

医院和科室的关联关系 (models.py):

```
class HospitalModel(db.Model):
    __tablename__ = 'hospital'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(100), unique=True, nullable=False)
    departments = db.relationship('DepartmentModel', backref='hospital',
lazy='dynamic')

class DepartmentModel(db.Model):
    __tablename__ = 'department'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(50), nullable=False)
    description = db.Column(db.Text)
    hospital_id = db.Column(db.Integer, db.ForeignKey('hospital.id'),
nullable=True)
    doctors = db.relationship('DoctorInfoModel', backref='department',
lazy='dynamic')
    __table_args__ = (
        db.UniqueConstraint('hospital_id', 'name',
name='uix_hospital_department'),
    )
```

唯一性约束确保数据完整性:

- 医院名称全局唯一：`name = db.Column(db.String(100), unique=True, nullable=False)`
- 同一医院内科室名称唯一：`db.UniqueConstraint('hospital_id', 'name', name='uix_hospital_department')`
- 同一医院内医生工号唯一: `db.UniqueConstraint('hospital_id', 'internal_id', name='uix_hospital_employee')`

患者和医生信息模型:

```python
class PatientInfoModel(db.Model):
    __tablename__ = 'patient_info'
    id = db.Column(db.Integer, db.ForeignKey('patient.id'), primary_key=True)
    phone = db.Column(db.String(11), nullable=False,unique=True)
    name = db.Column(db.String(50), nullable=False)
    gender = db.Column(db.Enum('male', 'female'), nullable=True)
    address = db.Column(db.String(255),nullable=True)
    avatar_url = db.Column(db.String(255),nullable=True)
    birth_date = db.Column(db.Date,nullable=True)
    medical_history = db.Column(db.Text,nullable=True)
```

流程特点分析

1. **层次化数据管理**: 医院→科室→医生的三级数据结构管理
2. **关联数据检查**: 删除操作前检查是否存在关联数据，避免数据不一致
3. **数据完整性保护**: 通过数据库约束确保关键字段的唯一性
4. **统计分析功能**: 支持多维度的数据统计和报表生成

系统安全特性

1. **权限控制**: 只有管理员角色才能访问数据管理功能
2. **操作审计**: 重要的数据修改操作应该记录操作日志
3. **数据备份**: 在删除操作前建议进行数据备份
4. **级联处理**: 删除医院时需要处理相关的科室和医生数据

这些活动图和对应的代码实现展示了一个完整的医疗管理系统的核心业务流程，每个流程都有对应的数据模型支撑，确保了系统的数据一致性和业务逻辑的完整性。