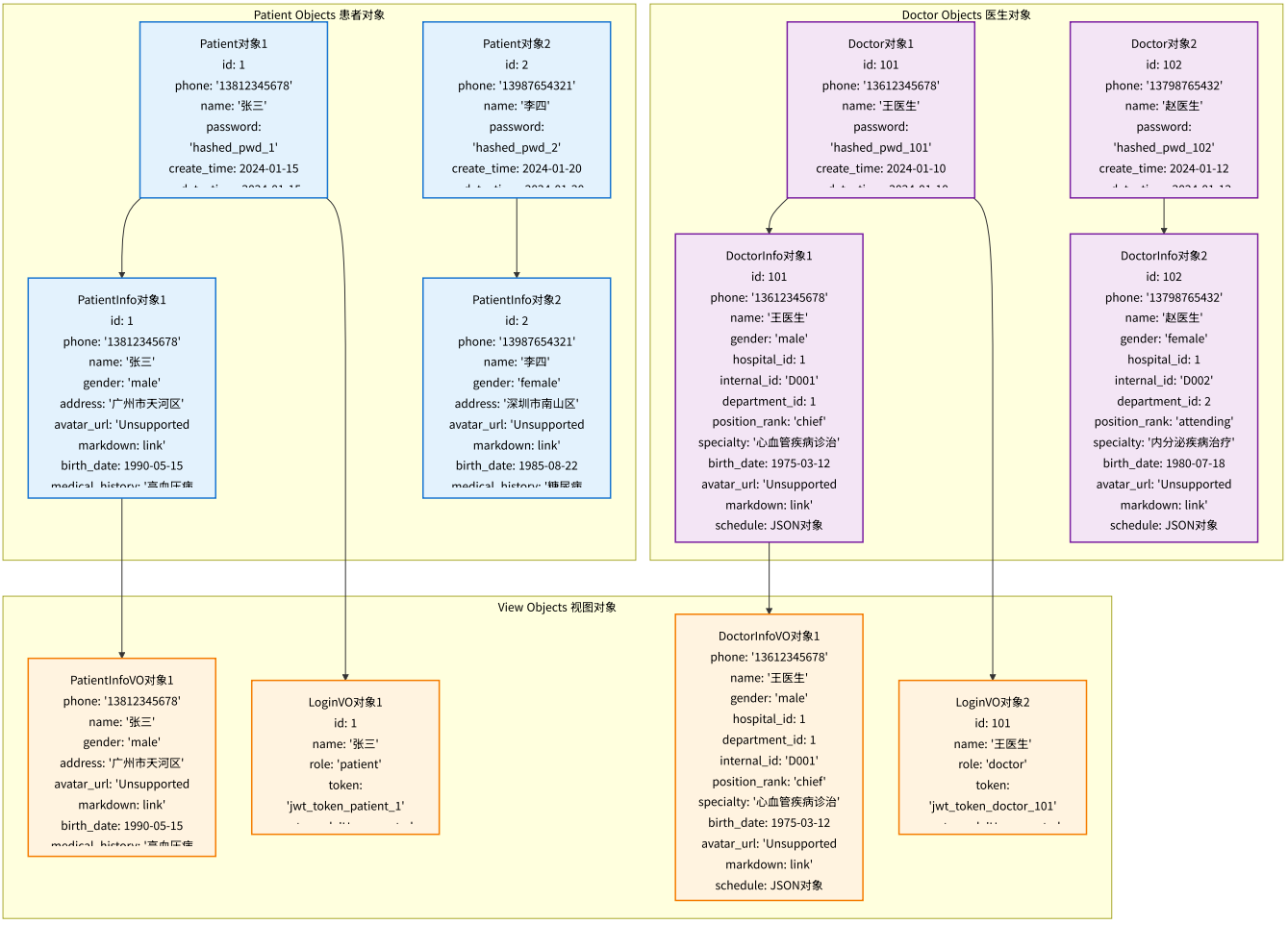
对象图

1. 用户管理对象图



业务对象说明

这个对象图展示了医疗系统中用户管理模块的运行时对象实例，包括患者和医生的基本信息对象、详细信息对象以及用于数据传输的视图对象。

关键技术实现

患者对象实例化 (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 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)
```

医生对象实例化 (models.py):

```
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)

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)
    avatar_url = db.Column(db.String(255))
    schedule = db.Column(db.JSON, nullable=True)
```

视图对象定义 (vo.py):

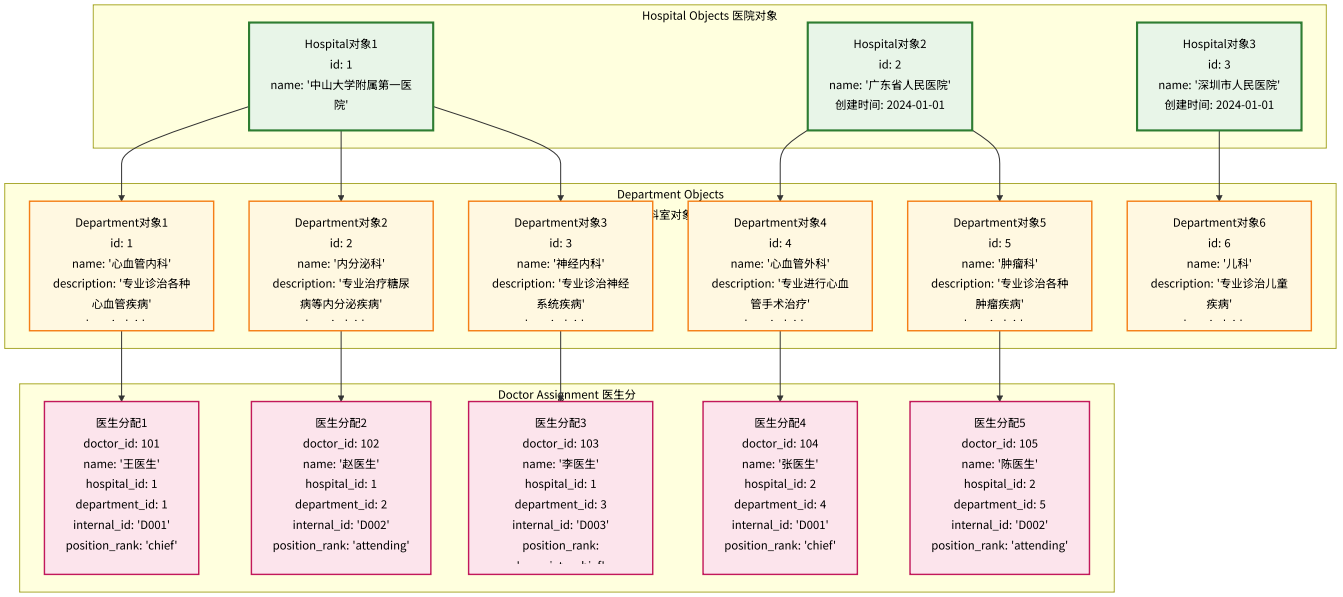
```
class LoginVO:
    def __init__(self, id, name, role, token, avatar_url):
        self.id = id
        self.name = name
        self.role = role
        self.token = token
        self.avatar_url = avatar_url
    def to_dict(self):
        return {"id": self.id, "name": self.name, "role": self.role,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

class PatientInfoVO:
    def __init__(self, phone, name, gender, address, birth_date, avatar_url,
medical_history):
        self.phone = phone
        self.name = name
        self.gender = gender
        self.address = address
        self.avatar_url = avatar_url
        self.birth_date = birth_date
        self.medical_history = medical_history

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
```

对象关系分析

1. 一对一关系: 每个PatientModel对象对应一个PatientInfoModel对象，通过外键关联
2. 对象转换: 数据库模型对象可以转换为VO对象用于API响应
3. 数据封装: VO对象隐藏了敏感信息（如密码），只传输必要的业务数据
4. 类型安全: 使用枚举类型确保gender和position_rank字段的数据一致性

2. 医疗机构管理对象图



业务对象说明

该对象图展示了医疗系统中医疗机构的层次化组织结构，包括医院、科室和医生分配的具体对象实例。

关键技术实现

医院对象模型 (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')
```

科室对象模型 (models.py):

```
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'),
    )
```

医生分配关系 (models.py):

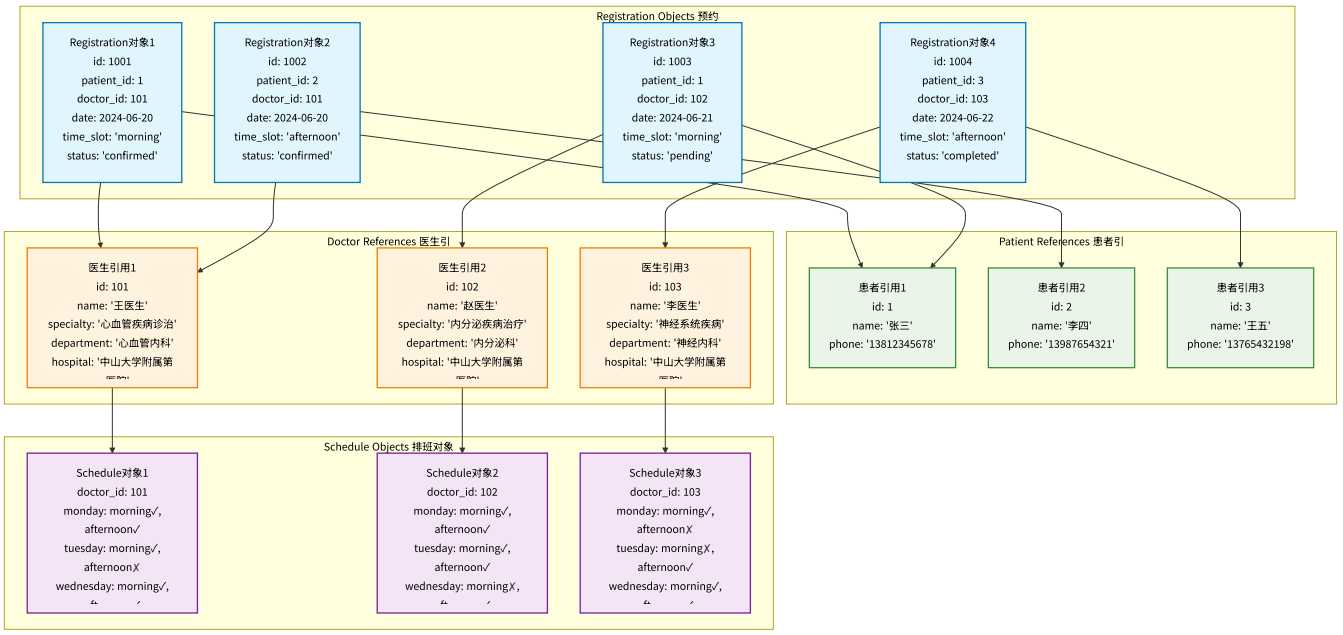
```
class DoctorInfoModel(db.Model):
    hospital_id = db.Column(db.Integer, db.ForeignKey('hospital.id'),
nullable=True)
    hospital = db.relationship('HospitalModel', backref='doctors')
    internal_id = db.Column(db.String(8), nullable=True)
    __table_args__ = (
        db.UniqueConstraint('hospital_id', 'internal_id',
name='uix_hospital_employee'),
```

```
    )
    department_id = db.Column(db.Integer, db.ForeignKey('department.id'))

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

对象关系分析

1. **层次化结构**: 医院→科室→医生的三级组织架构
2. **约束保证**: 同一医院内科室名称唯一，同一医院内医生工号唯一
3. **双向关联**: 通过backref建立双向关系，可以从医院查找科室，也可以从科室查找医院
4. **延迟加载**: 使用lazy='dynamic'优化查询性能，避免一次性加载所有关联数据

3. 预约挂号业务对象图



业务对象说明

这个对象图展示了预约挂号业务的核心对象实例，包括预约记录、医生排班、患者和医生引用对象的具体实例。

关键技术实现

预约记录对象 (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)
    patient = db.relationship('PatientInfoModel', backref='registrations')
    doctor_id = db.Column(db.Integer, db.ForeignKey('doctor_info.id'),
nullable=False)
    doctor = db.relationship('DoctorInfoModel', backref='registrations')
    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
        }
```

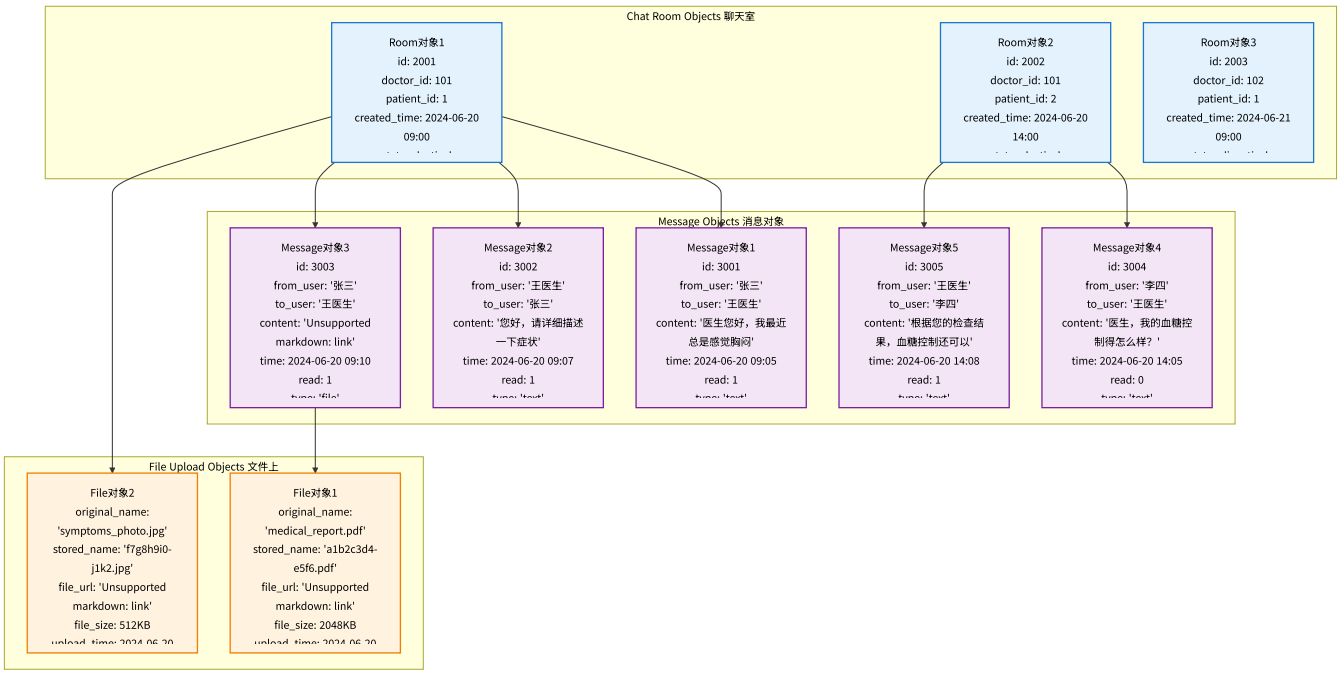
医生排班对象 (models.py):

```
class DoctorInfoModel(db.Model):
    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}
    })
```

对象关系分析

1. **多对一关系**: 多个预约记录可以关联同一个医生或患者
2. **时间管理**: 通过date和time_slot字段精确控制预约时间
3. **JSON存储**: 医生排班使用JSON格式存储复杂的时间表结构
4. **数据完整性**: 通过外键约束确保预约记录的患者和医生都是有效的

4. 消息通信对象图



业务对象说明

该对象图展示了医患在线咨询系统的实时通信对象，包括聊天室、消息记录和文件上传的具体对象实例。

关键技术实现

聊天室对象 (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)
    doctor = db.relationship('DoctorModel', backref='rooms')
    patient_id = db.Column(db.Integer, db.ForeignKey('patient.id'),
nullable=False)
    patient = db.relationship('PatientModel', backref='rooms')

    def to_dict(self):
```

```
        return {
            "doctor_id": self.doctor_id,
            "doctor_name": self.doctor_name,
            "patient_id": self.patient_id,
            "patient_name": self.patient_name
        }
```

消息对象 (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通信实现 (consultant.py):

```
@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()
```

文件上传对象 (blueprints/upload_file.py):

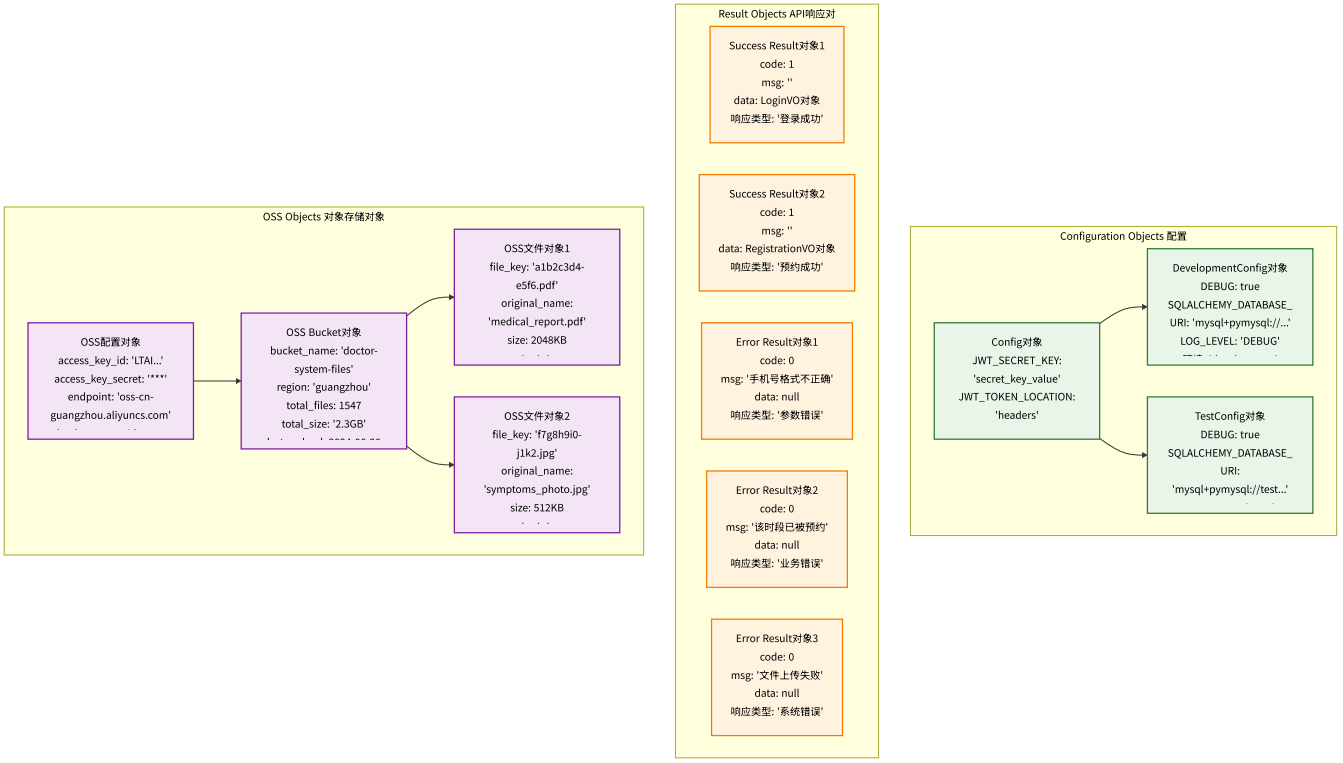
```
@bp.route('/picture', methods=['POST'])
def upload_picture():
    """上传头像"""
    file = request.files.get('file')
    if not file:
        return jsonify(Result.error("未提供文件").to_dict()), 400

    original_file_name = file.filename
    if not original_file_name.lower().endswith(('.png', '.jpg', '.jpeg',
'.gif')):
        return jsonify(Result.error("仅支持PNG、JPG、JPEG和GIF格式的图
片").to_dict()), 400
    suffix = original_file_name.split('.')[-1]
    file_name = f"{uuid.uuid4().hex}.{suffix}"

    try:
        file_url = upload_file(file, file_name)
        return jsonify(Result.success({"file_url": file_url}).to_dict()), 200
    except Exception as e:
        return jsonify(Result.error(f"服务器错误: {str(e)}").to_dict()), 500
```

对象关系分析

1. **会话管理**: 每个RoomModel对象代表一个医患咨询会话
2. **消息持久化**: 所有消息都保存到数据库，支持历史记录查询
3. **实时通信**: 使用WebSocket技术实现消息的实时推送

4. **多媒体支持**: 支持文本消息和文件上传，满足多样化的咨询需求

5. 系统配置和工具对象图



业务对象说明

这个对象图展示了系统底层支撑对象的具体实例，包括配置对象、API响应对象和OSS存储对象。

关键技术实现

配置对象层次 (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

class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = os.getenv('DEV_DB_URI',
'mysql+pymysql://root:lhh123456789@127.0.0.1:3306/doctor?charset=utf8')

class TestConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = os.getenv('TEST_DB_URI',
'mysql+pymysql://root:lhh123456789@127.0.0.1:3306/doctor?charset=utf8')
```

API响应对象 (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)

    def to_dict(self):
        return {
            "code": self.code,
            "msg": self.msg,
            "data": self.data
        }
```

OSS存储对象 (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')

    # 初始化 Bucket
    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
```

对象关系分析

1. **配置继承**: 不同环境的配置对象继承自基础配置，实现配置的层次化管理
2. **统一响应**: 所有API响应都通过Result对象封装，提供一致的成功/错误处理格式
3. **存储解耦**: OSS对象管理文件存储生命周期，从配置到上传再到访问URL生成，实现存储服务与业务逻辑的解耦