**Flask需求规格说明书**

**版本号：V1.0.0.200325**

\*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编写 | 马广洲 | 日期 | 2020年3月25 | 编写内容简介 | Flask-script插件的需求分析 |
|  | 张祥国 | 日期 | 2020年3月25 | 编写内容简介 | Flask-SQLAlchemy插件的需求分析 |
|  | 王康明 | 日期 | 2020年3月25 | 编写内容简介 | Flask-SQLAlchemy插件的需求分析 |
|  | 潘安佶 | 日期 | 2020年3月25日 | 编写内容简介 | Flask框架需求分析 |
|  | 常佳辉 | 日期 | 2020年3月25日 | 编写内容简介 | Flask框架需求分析 |
|  | 刘佳恒 | 日期 | 2020年3月25日 | 编写内容简介 | Flask-migrate插件的需求分析 |
|  | 牟秋宇 | 日期 | 2020年3月25日 | 编写内容简介 | Flask-migrate插件的需求分析 |
| 校对 |  |  |  |  |  |

**1 引言**

**1.1. 目的**

本文档是为了便于协调组内成员进行后期的工作，对项目进行跟踪和监控，对任务的进度进行安排与调控，在对于flask框架以及拓展插件进行分析后，对于flask框架系统的需求说明。

**1.2. 文档约定**

文档在编辑时，遵守IEEE发布的对软件需求说明书的文档约定，遵守其约定。

**1.3. 术语与缩略词**

|  |  |
| --- | --- |
| 缩写、术语及符号 | 解 释 |
| ORM | 对象关系映射 |
| SQLAlchemy | Python编程语言下的一款ORM框架 |
| RUCM | 限制性用例模型 |
| python shell | Python的命令行 |
| Script | Flask框架中插入外部脚本的插件 |
| RUCM | 限制性用例模型 |

**2 系统概述**

**2.1. 任务背景**

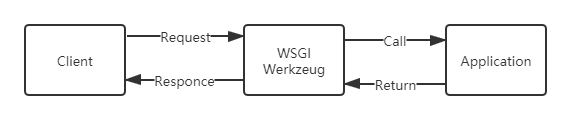
Flask是一个轻量级的可定制框架，使用Python语言编写，较其他同类型框架更为灵活、轻便、安全且容易上手。它可以很好地结合MVC模式进行开发，开发人员分工合作，小型团队在短时间内就可以完成功能丰富的中小型网站或Web服务的实现。另外，Flask还有很强的定制性，用户可以根据自己的需求来添加相应的功能，在保持核心功能简单的同时实现功能的丰富与扩展，其强大的插件库可以让用户实现个性化的网站定制，开发出功能强大的网站。

它具有以下主要特点：

1. 灵活、轻便、高效
2. 轻量级，使用模块化设计，不依赖更多的库
3. Restful风格的接口设计
4. 配置量少

因此，本次实验选用flask来作为分析目标。

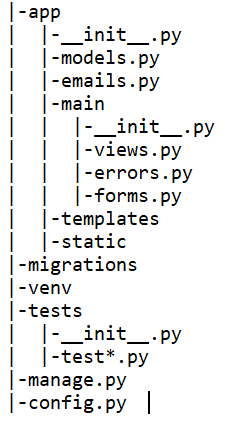
**2.2. 系统模型**



Flask的基本框架模式如上图所示。Flask是一个微小型框架，并且有两个主要依赖：路由、调试和Web服务器网关接口（WebServerGatewayInterface，WSGI）子系统由Werkzeug提供；模板系统由Jinja2提供。而像是对于数据库访问、验证Web表单和用户身份认证等一系列功能，Flask框架是不支持的。这些功能都是以扩展组件的方式进行实现，然后再与Flask框架集成。

flask框架是基于MVC设计的框架，框架在设计之初便考虑到了模块化的使用，为了达到这一目的，故基于独立的组件进行开发，使得开发的组件不完全依赖于flask。

**2.3 项目结构**



Flask的项目结构如图所示，其中可以分为两个部分，app中的源码和其他资源，具体的用途如下表所示。

\*

|  |  |
| --- | --- |
| 路径 | 用途 |
| app | 基本程序包 |
| app\models.py | 数据库模型 |
| app\main | 程序模块，根据业务区分，不同的模块可以各自拥有自己的单独目录 |
| app\templates | 用于存放Jinja2的html模板 |
| app\static | 存放静态文件 |
| migrations | 数据库迁移文件夹 |
| venv | 虚拟环境文件 |
| tests | 单元测试程序，可以包含多个文件 |
| manage.py | 主体启动文件 |
| config.py | 全局配置文件 |

**3 需求分析**

**3.1Flask**

**3.1.1 业务需求**

典型需求场景：使用Flask框架搭建Web应用程序

Flask是使用Python编写的轻量级Web应用框架，它具有简单的核心，使用extension增加框架内容，具有可定制的特点，可结合MVC模式进行开发、测试。适合开发人员分工合作，小型团队在短时间内完成功能丰富的中小型网站或Web服务的实现。

**3.1.2 功能需求**

开发人员可以利用flask框架快速开发出功能完备的中小型网站或Web服务。并且可以采用插件的方式增加框架功能。基于以上业务需求的分析与相应的业务要求，我们认为软件有如下的功能需求：

1. 定制路由规则
2. 处理静态资源
3. 渲染模板
4. 可操作数据库
5. 支持调试模式
6. 日志工具

**3.1.3 配置——创建并运行实例**

安装环境——安装python运行环境，以及安装相关依赖包，创建虚拟运行环境。

应用开发——定制路由规则，并实现处理函数，确定返回资源。

运行参数——指定服务运行时参数，包括调试模式、开发/生产环境等。

运行实例——指定运行应用文件，解释执行。

**3.1.4 非功能需求**

1. 高效性：

框架应当支持一定规模的并发请求，并且在单位时间内完成足够多的任务。

1. 扩展性：

Flask作为一款轻量级开发框架，框架的核心只提供一些基础功能，许多的其他功能需要使用者进行额外的扩展。因此框架应当能够支持使用者方便的进行扩展与修改，以实现更好的表现。

1. 代码重用：

Flask应当能够为使用者提供一定的代码重用功能，以此提高开发效率，并且有利于后期的开发与维护。

1. 鲁棒性：

系统应该在异常和危险情况下都能保持健壮的表现和稳定的性能，有健全的容错机制和方法。

**3.1.5 RUCM和用例图**

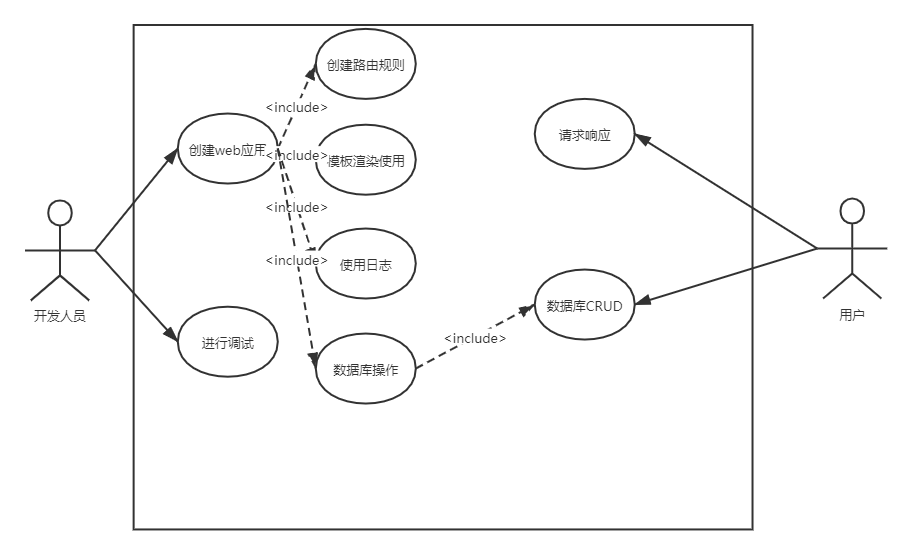


图 用例图

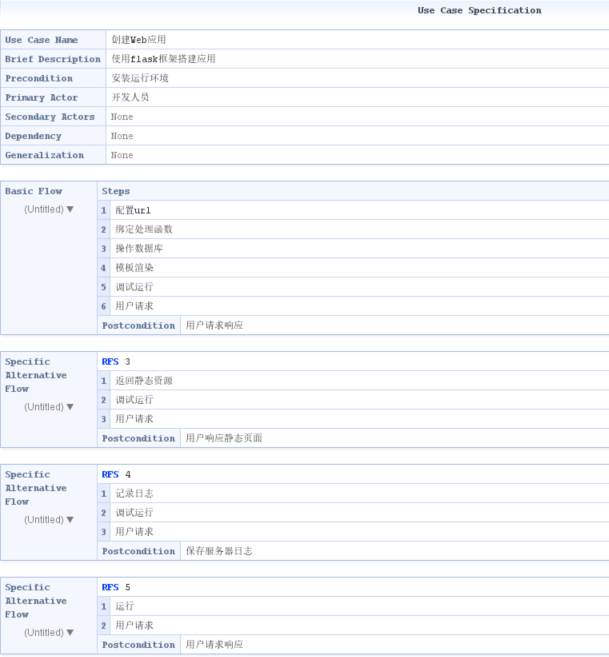


图 RUCM创建web应用

**3.2Flask-sqlalchemy**

**3.2.1 业务需求**

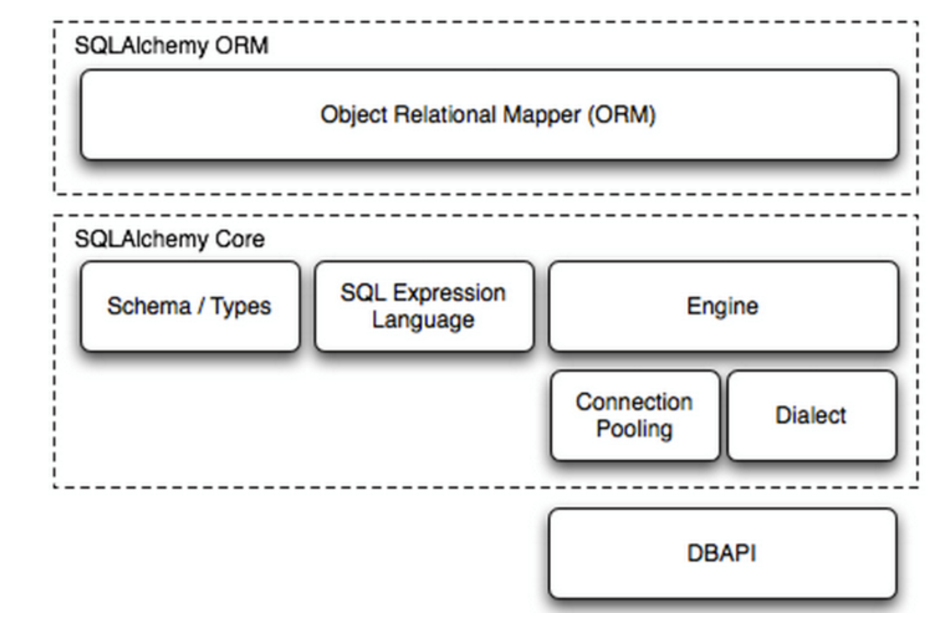
典型需求场景：在Flask应用中使用[SQLALchemy](http://true/)支持的扩展

对于一个Flask应用来说，对数据库的操作常常是不可避免的。如果通过复杂的SQL语句操作数据库，通常会大大增加开发人员的工作量。为了解决这一问题，对象关系映射器（Object Relational Mappers，ORM）在过去数年吸引了大量的目光，其作用是在关系型数据库和对象之间作一个映射，这样，在具体的操作数据库的时候，就不需要再使用复杂的SQL语句，只要像平时操作对象一样操作即可。

而SQLAlchemy是Python编程语言下的一款开源软件。提供了SQL工具包及对象关系映射（ORM）工具。因此，需要Flask-SQLAlchemy为Flask应用增加 SQLAlchemy 支持的扩展，从而简化在Flask 中 SQLAlchemy 的使用，提供有用的默认值和额外的助手来更简单地完成常见任务。

**3.2.2 功能需求**

Flask-SQLAlchemy是一个使用Python语言实现的ORM框架，该框架建立在数据库接口之上，使用关系对象映射进行数据库操作，简言之便是：将类和对象转换成SQL，然后使用数据API执行SQL并获取执行结果。



flask-SQLALchemy框架

* Engine，框架的引擎
* Connection Pooling ，数据库连接池
* Dialect，选择连接数据库的数据库 API种类
* Schema/Types，架构和类型
* SQL Exprression Language，SQL表达式语言

当用户需要开发flask应用时，可以调用本插件，搭建好一个完整的Flask应用。用户花费更多的时间在其他方面，而不必关心数据库操作底层的实现细节。基于以上业务需求的分析与相应的业务要求，flask-SQLALchemy应该具有如下的功能需求：

1） 除了ORM方式之外，还要支持执行原生的SQL语句

2） 连接数据库

3） 定义创建数据库表

4） 定义外键

5） 实现增、删、改、查操作功能

6） 数据库1NF、2NF、3NF、BCNF范式检测功能（扩展）

**3.3.3 配置——从Flask主配置中加载配置值**

声明模型——创建 SQLAlchemy 对象，这个对象提供一个名为Model 的类，用于作为声明模型时的 delarative基类

选择，插入，删除——向数据库插入记录，删除记录，查询记录，在视图中查询

绑定多个数据库——连接到多个数据库

信号支持——在更新提交到数据库之前以及之后得到通知

**3.3.4 非功能性需求**

高效性——高效进行数据库操作

高可扩展性——数据库表设计易于更改

灵活性——灵活简单，易于使用

安全——数据安全加密，防SQL注入攻击

**3.3.5 扩展功能需求分析**

数据库的范式设计是非常重要的，满足良好范式设计的数据库，它的数据冗余更小、结构更加合理，从而减小数据库的存储占用空间，也不容易发生插入异常、删除异常、更新异常等事件。

数据库设计一共有6大范式：1NF、2NF、3NF、BCNF、4NF、5NF，从左到右范式要求越来越严格，4NF和5NF应用情况比较少，属于非常专业的领域才需要用到，普通开发人员只需要满足BCNF或者3NF。

1NF：原子性（所有字段值都是不可分解的原子值）

2NF：消除部分依赖（所有的非主属性都完全依赖于关键字）

3NF：消除传递依赖（所有非主属性都不传递依赖于关键字）

BCNF：消除主属性的传递依赖（所有的属性都不传递依赖于关键字）

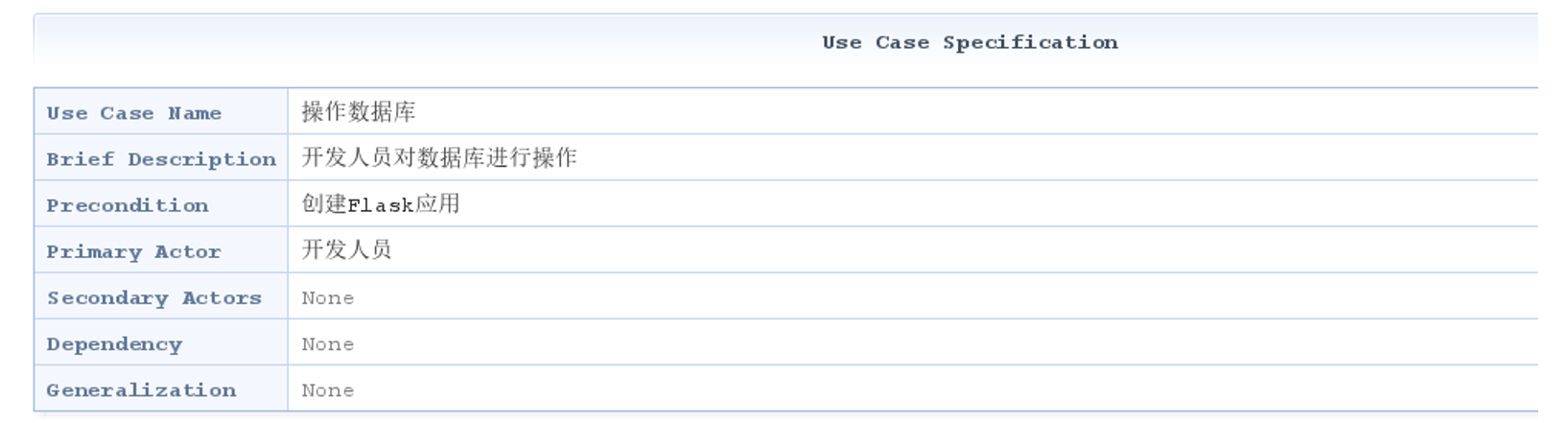
检查数据表是否符合这4个范式的设计，返回True或者False给开发人员，从而提示开发人员调整自己的面向对象的代码。

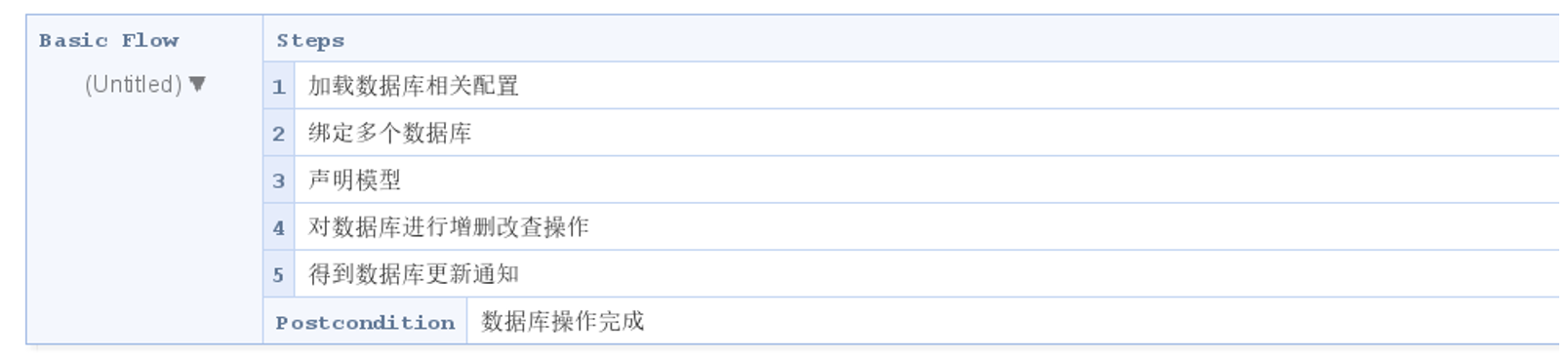
函数依赖是数据依赖的一种，它反映属性或属性组之间相依存，互相制约的关系，即反映现实世界的约束关系。

这里的函数依赖是指，假如表里有A、B、C三列，假如列A中所有相同的字段，在B中对应的行里的字段值都一样，则认为B依赖于A。

设置4个函数接口，用于检查是否分别满足1NF、2NF、3NF、BCNF。

**3.3.6 RUCM和用例图**





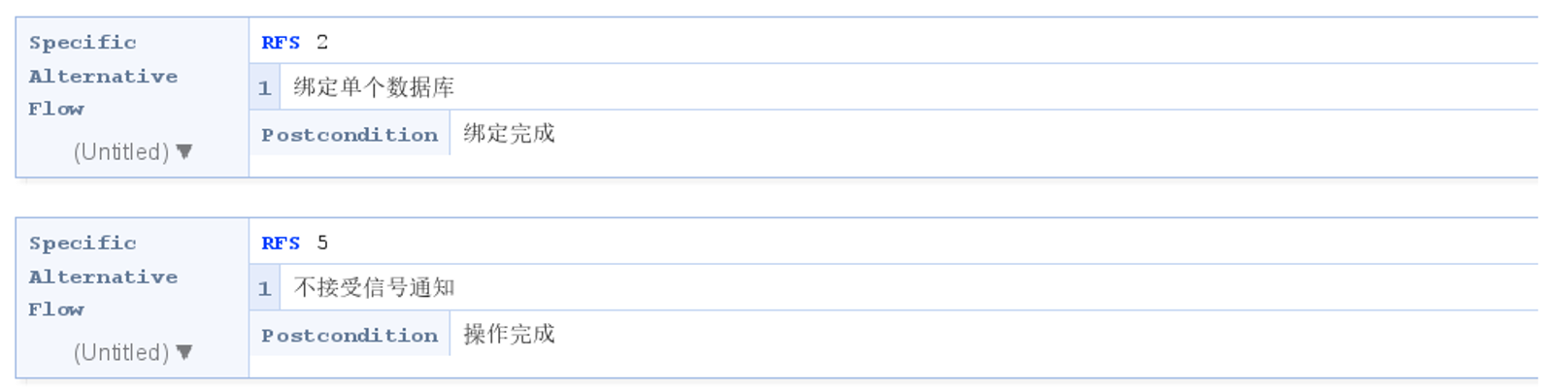


图 RUCM图

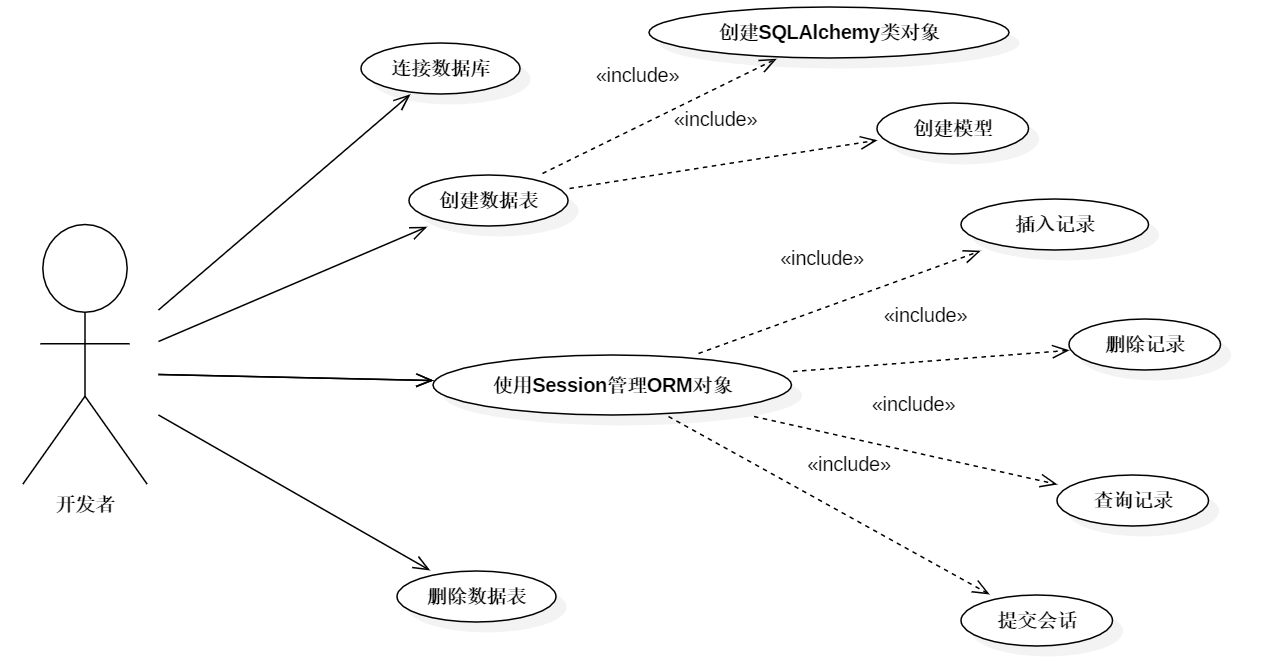


图 Flask-SqlAlchemy用例图

**3.3Flask-migrate**

**3.3.1 业务需求**

典型业务需求场景：更改数据库模型

在开发应用的过程中，会发现有时需要修改数据库模型，而且修改之后还要更新数据库。例如添加或删除列。在应用程序的早期开发过程中，如果不担心删除数据库中的所有数据，则对关系数据库中的表进行更改很容易。但是，一旦投入生产并将实际数据存储在关系数据库中，则在更改表时需要非常谨慎。如果更新表方式就是先删除旧表，这样做会丢失数据库中的全部数据。此时就需要一个扩展模块帮助开发人员执行数据库的迁移，包括数据库的版本控制。

**3.3.2 功能需求**

当开发人员执行数据库迁移时，可以调用本flask数据库迁移插件，通过本插件提供的组件，用户花费更多的时间在逻辑实现上，而不必关心底层的实现细节。基于以上业务需求的分析与相应的业务要求,我们认为软件有如下的功能需求。

1. 初始化对应用程序的迁移支持。
2. 手动创建迁移脚本。
3. 自动生成迁移脚本。
4. 数据库的升级，即把迁移中的改动应用到数据库中。
5. 数据库的降级，将改动删除。
6. 显示当前数据库修订版本
7. 显示迁移历史
8. 合并两个数据库的迁移，生成一个新的迁移脚本。

**3.3.3 配置**

1. 使用init命令初始化数据库，并建立名为migrations的文件夹
2. 使用migrate命令迁移数据库
3. 使用upgrade命令对迁移的数据库进行更新

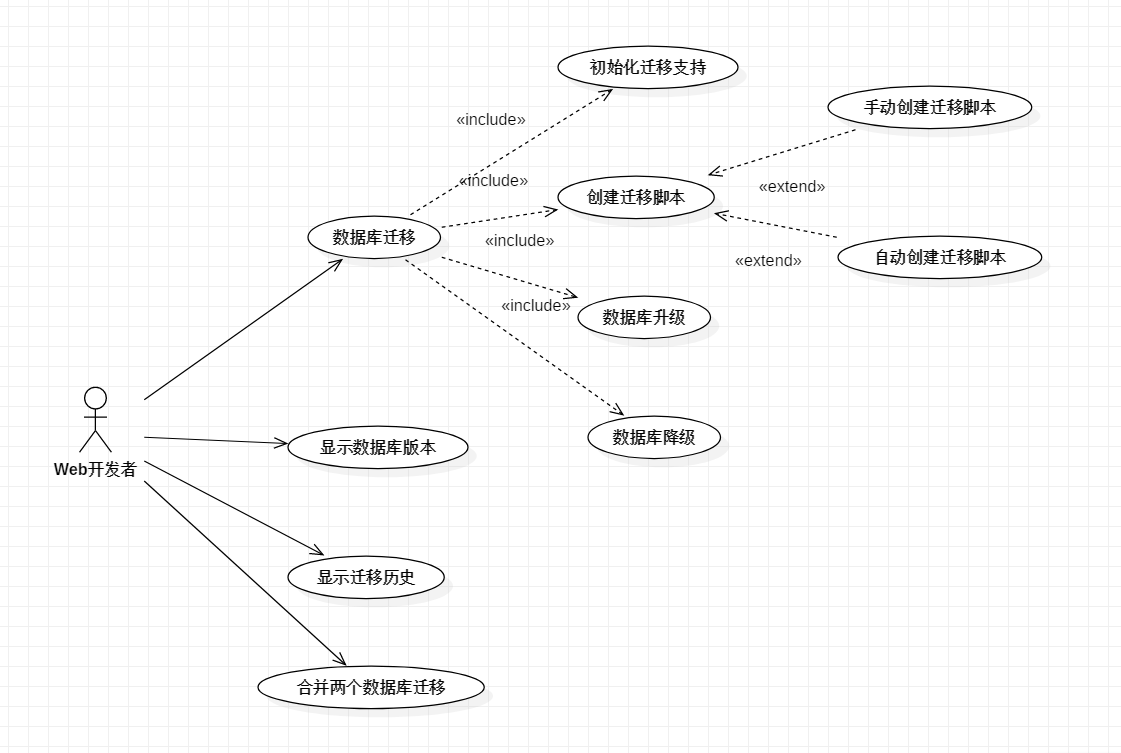
**3.3.4 非功能性需求**

1. 高效性：提高数据库迁移效率
2. 可扩展性：模块化

**3.3.5 扩展功能需求分析**

1. 创建数据库迁移脚本提升易用性：使用flask-script以命令行的形式完成数据库迁移的操作

**3.3.6 RUCM和用例图**



图Flask-migrate 用例图

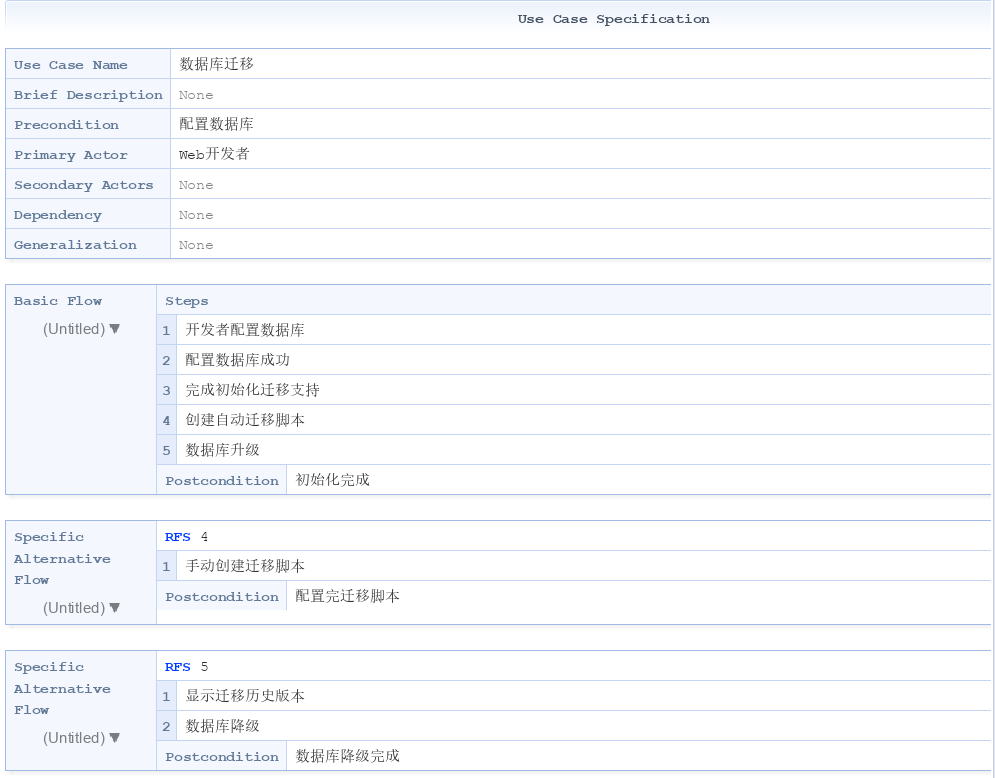
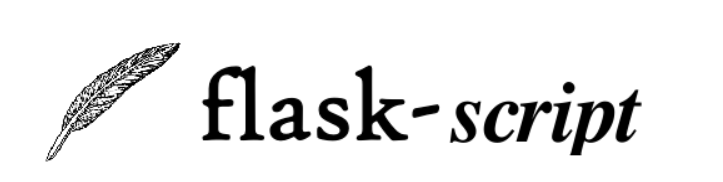


图 Flask-migrate RUCM图

**3.4Flask-script**



**3.4.1 业务需求**

典型需求场景：在Flask应用中使用Script支持的扩展

Flask Script扩展提供向Flask插入外部脚本的功能。Flask 的开发 Web 服务器支持很多启动设置选项，但只能在脚本中作为参数传给 app.run()函数。这种方式并不十分方便，传递设置选项的理想方式是使用命令行参数。

**3.4.2 功能需求**

当用户需要开发flask应用时，可以调用本插件，搭建好一个完整的Flask应用。需求分析如下：

1. 运行一个开发用的服务器，一个定制的Python shell，设置数据库的脚本，cronjobs，及其他运行在web应用之外的命令行任务；
2. 使得脚本和系统分开；
3. shell命令用于在程序的上下文中启动python shell 会话。你可以使用这个会话中运行维护任务或测试，还可调试异常；
4. runserver命令用来启动web服务器。运行python hello.py runserver将以调试模式启动服务器；
5. --host 参数用来告诉web服务器在哪个网络接口上监听来自客户端的连接，默认情况下，flask开发web服务器监听localhost上的连接，所以只接受来自服务器所在计算机发起的连接。

**3.4.3 配置——创建并运行命令**

1. 创建脚本——创建一个Python模板运行命令脚本，可起名为manager.py；
2. 追踪运行情况——文件中，必须有一个Manager实例，Manager类追踪所有在命令行中调用的命令和处理过程的调用运行情况；
3. 参数——Manager只有一个参数——Flask实例，也可以是一个函数或其他的返回Flask实例；
4. 接受命令——调用manager.run()启动Manager实例接收命令行中的命令

**3.4.4 非功能性需求**

1. 高可用性——高效进行工程运行操作
2. 灵活性——灵活简单，易于使用

**3.4.5 扩展功能需求分析**

可扩展性——可自定义命令脚本，共有三个需求，第一个是添加那些不需要传递参数的命令的；第二个用来添加那些需要传递参数的命令的，有几个参数就需要些使用几个参数器；第三个如果有一些命令是针对某一功能的，可以将这些命令放在同一个模块里。

**3.4.6 RUCM和用例图**

