RestCloud-API开发与管理规范V3.0

文档编号	V3.0
文档密级	商密 （本文档只授权给谷云公司服务的企业内部传阅， 未经谷云许可不得对外传阅）

RestCloudAPI开发与管理规范V3.0

RestCloud
谷云科技(广州)有限责任公司
http://www.restcloud.cn

修订历史记录

版本	日期	AMD	修订者	说明
1.0	2020-02-12	A	RestCloud	发布文档
2.0	2020-05-10	M	RestCloud	修改文档
3.0	2020-10-20	M	RestCloud	修改2.5版本控制规范说明

（A-添加　M-修改，D-删除）

目 录

1. 前言
1.1 编写目的
1.2 阅读应具备知识
2. 开发规范
2.1 协议规范
2.2 Method规范
2.3 安全性和幂等性
2.4 接口路径规范
2.5 版本控制规范
2.6 请求参数规范
2.7 返回数据规范
2.8 Java Bean注解规范
2.9 数据大小约定
2.10 API超时时间约定
3. API接入流程
4. API开发商约束
4.1 用户帐号注册
4.2 应用系统注册
4.3 API发布
4.4 API版本变更
4.5 API联调测试
4.6 调用外部业务系统的API
4.7 API编排流程约束
5. 管理规范建议
5.1 API需求管理
5.2 API发布
5.3 API管理责职
5.4 API性能监控
5.5 API文档索引
5.6 API迭代
5.7 API下线

1. 前言

1.1 编写目的

不同开发人员对于API的地址名称，输入参数，返回参数等都要自己的格式，导致每个开发人员定义的API规范都不一致，这种不一致在平台开发时可能会产生一系列的冲突和未知问题。

为解决不同开发人员不同的API规范，以本文档统一平台API的开发设计规范与管理规范，提高使用本平台的开发人员之间的交流与协作开发的效率。

1.2 阅读应具备知识

在阅读本文档之前，我们建议您具备以下条件，这有助于您阅读并理解文档内容。

1. 对API的整体概念有所了解
2. 开发和实施过API
3. 了解http请求过程及原理
4. 使用过RestCloud相关产品
5. 使用过前后端分离的开发模式

2. 开发规范

2.1 协议规范

为了确保不同业务系统之间以及前后端的的数据交互的快捷性，通讯协议统一约定如下：

1. 对内调用的API接口统一使用 HTTP协议
2. 对外互联网发布的API建议使用HTTPS协议也可以使用HTTP
3. 只有旧业务系统已有接口可以使用WebService报文
4. 新的API接口必须使用标准的HTTP报文并使用JSON作为统一的数据传送标准
5. 如无特殊情况禁止在新开发的API接口中使用XML格式传送数据，统一使用JSON格式作为数据包或者form键值对模式数据包。

2.2 Method规范

1. 在接口路径规范阅读之前必须先定义好Method规范，虽然Restful风格建议使用get、post、delete、put来区分和操作不同的资源，但是在实际项目开发的过程中全部使用Restful风格往往会造成开发效率的降低，而且在资源较多时API命名将给开发人员造成很大的困扰。

2. 我们建议在Http Method标准上只允许使用get和post方法，禁止使用delete和put方法，很多企业和安全扫描工具不允许开放delete和put方法，有些web防火墙默认会拦截delete和put方法的请求。
3. 使用了delete和put方法后tomcat安装后还需要进行配置才能支持此两种方法，delete和put方法在键值对提交模式下后端servlet接收参数必须要自己从流中接收开发难度增大。
4. 前端使用delete和put方法时没有get和post快捷。
5. 我们建议在API开发时只允许使用get和post方法，delete和update则放在url中。
6. 更新用户使用：POST /appid/users/update
7. 删除用户使用：POST /appid/users/delete

2.3 安全性和幂等性

安全性：不会改变资源状态，可以理解为只读的，输入相同参数的情况下总是能查出相同的内容

幂等性：执行1次和执行N次，对资源状态改变的效果是等价的，适于用参与事务的API必须要具有幂等性的API

.	安全性	幂等性
GET	底	√
POST	高	参与事务时必须支持

POST提交的API如果参与到API服务编排平台中时如果要支持补偿或事务时则必须支持幂等性，否则API服务编排平台在进行事务补偿执行时有可能会发生多次重复数据写入的可能。

2.4 接口路径规范

API接口路径规范的原则是能从路径中清析的查看到此API所属的业务领域或应用，appid必须先在RestCloud平台中建立。

1. 路径中必须包含所属的appid应用编号示例：/appid/api/delete
2. 路径中的变量参数应尽量放在最后示例：

/appid/users/{id} 正确

错误: /appid/{id}/users

在RestCloud平台中路径变量越靠后性能越好，越靠前则所能匹配到的url会越多，系统需要一层一层分析比对才能最终确定所调用的API，在如果路径中没有{}变量的情况下，url的长短并不影响性能，但是过长的url不便于阅读。

3. 路径中的所有字母建议使用小写，词语之间的分隔可以使用/路径或者_下划线，而不要在URL中使用驼峰格式，因为URL是区分大小写的，所以全部小写是比较好的策略。

正确示例：/appid/user/username_info

错误示例：/appid/user/usernameInfo

4. 定义自定义路径部分时，使用名词的复数形式定义一个资源，如若有动词词性在url中考虑以下划线区分，在使用动词为前/后缀时，常见动词有：add、delete、update、query、get、send、save、detail、list等.

5. 基本操作示例

GET	/users	获取用户列表
GET	/users/{userId}	查看某个具体的用户信息
POST	/users/update	新建或更新一个用户
POST	/users/delete	删除某一个用户信息
GET	/users/search	搜索用户

6. 避免层级过深的URI

/在url中表达层级，用于按实体关联关系进行对象导航，一般根据id导航。过深的导航容易导致url膨胀，不易维护，如 GET /zoos/1/areas/3/animals/4，可以尽量使用查询参数代替路径中的实体导航，如GET /animals?zoo=1&area=3；

7. 对资源访问时的URL约定

服务器端的资源组合实体建议在uri中通过父实体的id导航访问，组合实体不是主实体，它的生命周期完全依赖父实体，无法独立存在，在实现上通常是对数据库表中某些列的抽象，不直接对应表，也无id。一个常见的例子是 User — Address，Address是对User表中zipCode/country/city三个字段的简单抽象，无法独立于User存在，则获取address的api的url为：

GET /user/addresses

2.5 版本控制规范

RestCloud平台支持在header头中传入version作为版本的标识，通过版本标识调用不同版本API接口。不传入版本号系统默认调用最新版本API。

2.6 请求参数规范

请求参数字段，尽可能与数据库表字段、对象属性名等保持一致，因为保持一致最省事，最简单，对于敏感数据或安全性较高的API则可以与数据库字段或对像不一致。

分页参数约定：

当前页统一使用：pageNo参数

每页显示数统一使用：pageSize参数

搜索关键字统一使用：keywords参数

排序：?sort=age,desc

传入参数共分为 3 种类型

1. 地址栏参数

restful 地址栏参数 /api/v1/product/00123 00123为产品编号，获取产品为 00123 的信息，其他示示例用法：

get 方式的查询字串，此种方式主要用于过滤查询，如下：

?limit=10：指定返回记录的最大数量

?pageNo=2&pageSize =100：指定第几页，以及每页的记录数。

?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。

?producy_type=1：指定筛选条件

2. 请求 body 数据

主要用于提交新建数据等，get请求应使用url参数，而非json 格式，post请求数据为RequestBody时请使用标准的JSON数据作为请求格式如下：

```
{
“userName”:”admin”, //认证token
“age”:”123”
}
```

3. 请求头

用于存放请求格式信息、版本号、token 密钥、语言等信息。请求头根据项目需求添加配置参数。如：请求数据格式，accept=‘application/json’等。如有需要，请求头可根据项目需求要求传入用户token、唯一验签码等加密数据。

4. 请求示例

对于Body请求的参数API接口必须提供JSON请求示例和返回示例，否则API的调用者很难知道Body参数的格式。

2.7 返回数据规范

统一规范返回数据为json格式，返回数据应包含：返回状态码、返回状态信息、具体数据等。

格式规范如下：

```
{
 "state":true, //表示成功,false表示失败
```

"msg":"success",
“ercode”:500 //如果有错误时返回错误状态码
"data": {
//json格式的具体数据
}
}

API执行过程中禁止发生了错误但给2xx响应，客户端可能会缓存成功的http请求；
正确设置http状态码，不要自定义；
Response body 提供
1) 错误的代码（可协助用于进行日志/问题追查的编码）；
2) 错误的描述文本（展示给用户的提示信息）；
3) API 可能抛出两类异常：业务异常和非业务异常。业务异常由自己的业务代码抛出，表示一个用例的前置条件不满足、业务规则冲突等，比如参数校验不通过、权限校验失败。非业务类异常表示不在预期内的问题，通常由类库、框架抛出，或由于自己的代码逻辑错误导致，比如数据库连接失败、空指针异常、除0错误等等，业务类异常统一设置 HTTP 响应状态码为：500；

2.8 Java Bean注解规范

在使用Java Bean进行API开发时需要注意以下规范
必须标注appId所属应用
beanId是所有应用的全局唯一id所以如果Class名称在多个应用中出现相同的class名称时会造成beanId重复，所以beanId建议命名为: appId.className这种类型如：

```
@BeanConfig(appId = "example",beanId = "example.DemoUserRest",beanName = "Rest示例服务",beanType = BeanType.CONTROLLER,autowired = true,singleton = true,hotMode = true)
@RestConfig(url = "/rest/example/users")
public class DemoUserRest {
}
}
```

其中方法的返回类型建议使用统一的DocAndView类型，由DocAndView类型统一转换为json再输出到前端，错误时使用：
ViewUtil.getErrorView(“错误原因”);

成功提示时使用：

```
return ViewUtil.getSuccessView("成功消息");
```

其他Json格式统一使用ViewUtil.getJsonView()方法返因json字符串

```
public DocAndView getById() throws Exception{
    return ViewUtil.getJsonView(userObj);
}
```

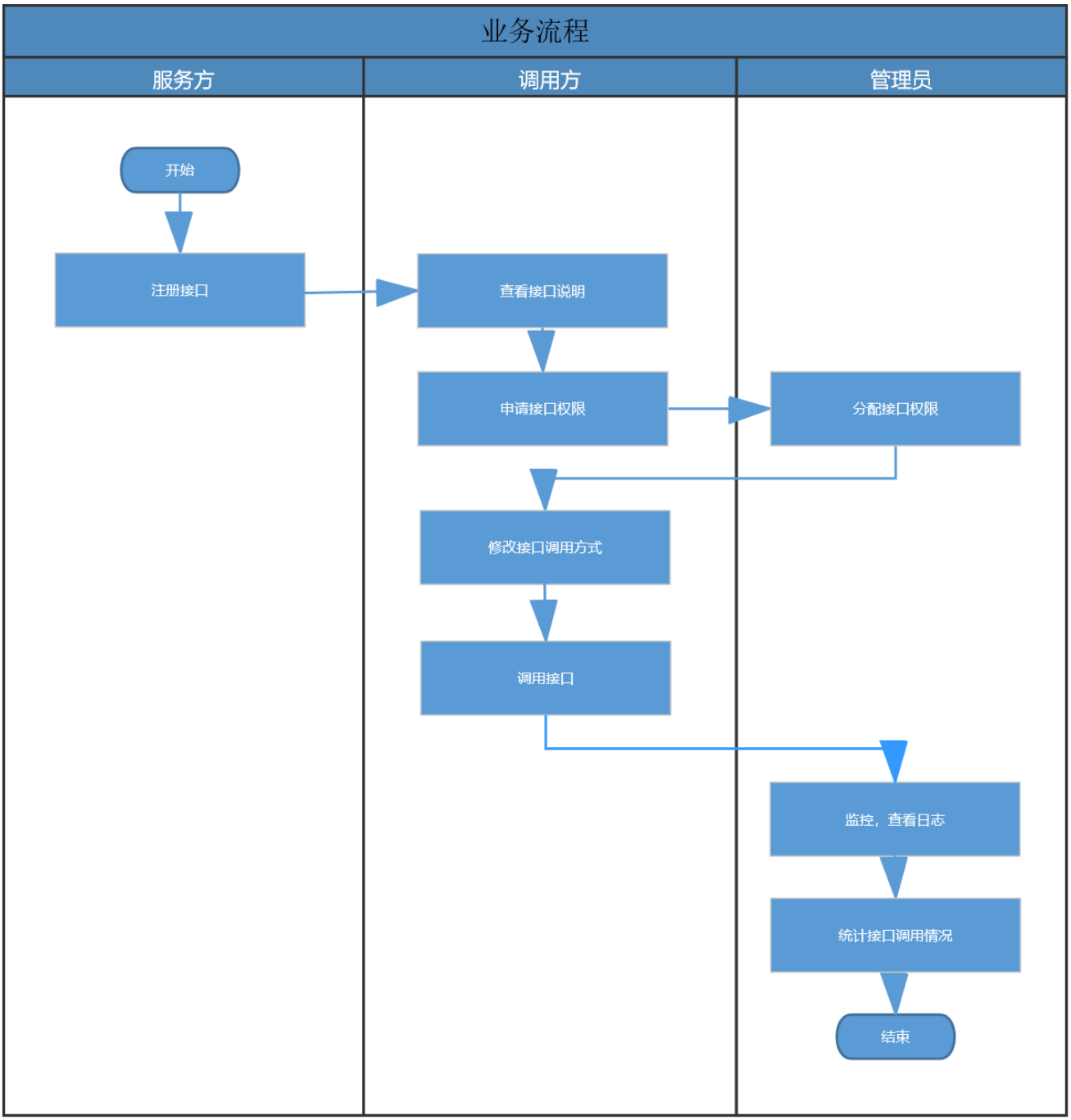
2.9 数据大小约定

单个API的请求及响应数据严格上不建议超过10M，超过10M的数据会严重影响网络以及JSON的序列化性能，如果超过10M的数据建议使用分页传输。

2.10 API超时时间约定

API的响应时间一般不建议超过180S即3分钟，如果时间太久会影响API网关性能，同时也容易出现问题，如果超过3分钟以上的API链接时间可以使用MQ等方式进行数据传输。

2.11 API接入流程



3. API开发商约束

3.1 用户帐号注册

1. 第三方开发商首先向API集成开发平台管理员申请一个平台的帐号和密码，开发商可以使用此帐号和密码登录API集成开发平台对API进行注册以及调用第三方API，使用此帐号可以获取平台的token信息。
2. 每个开发商原则上只发放一个帐号，如果同一开发商有多个业务系统时可以为每个业务系统发放一个帐号。

3.2 应用系统注册

1. 开发商必须在API集成平台中注册相应的业务系统名称和appId，appId的名称应与API集成平台的管理员进行确认，由管理员创建后告知开发商appId。

3.3 API发布

1. 新的业务系统或模块必须按照API规范发布相应的Restful的标准API接口
2. API必须提供详细的输入输出参数和参数示例并在API集成平台中形成API文档
3. 所有API必须安照相应业务功能分类在应用下进行分类注册和发布
4. 每个API或者应必须提供一个明确的管理员，当API出现故障时联第一时间联系此管理员
5. API注册时后端IP地址一律不得写死到每个API的配置中，必须配置成为一个公共变量，当后端系统的IP发生变更时只需要修改变更即可批量修改所有API的后端的IP地址。
6. API在测试环境中可以设置为调试状态，当发布到生产环境时必须把调试状态取消，否则调试状态的API将产生大量的日志影响系统性能。

3.4 API版本变更

1. API的版本变更规定，如果有大的版本变更建议保留原版本的API，然后发布一个新的API版本接口
2. 如果仍需变更原版本的API则必须取得API消费者的同意方可进行版本变更
3. API的下线必须经得所有API消费端和API集成平台的管理员同意后方可下线API

3.5 API联调测试

1. 业务系统开发商必须配合API消费者或调用端基于API集成平台进行联调测试，联调测试的结果以API网关的监控数据为准
2. API集成开发平台定期提供API监控数据，所有API的调用者和发布者均可以看到API的相应运行数据
3. API的测试和联调应在测试环境中进行
4. API上线发布到生产环境时如果要进行测试的必须按规定填写测试报文，产生的测试数据必须通知业务人员进行及时删除

3.6 调用外部业务系统的API

1. 业务系统需要调用其他业务系统的API时禁止点对点进行直接调用，必须从API集成平台中进行调用，如果对方的API没有注册到集成平台时可以要求对方注册到API集成平台后再进行消费调用

3.7 API编排流程约束

1. API编排流程中的API的HOST必须配置为变量，否则在后端API发生变更时很难找到那些API流程使用什么host，造成运维难度增加

2. API编排流程在测试环境可以设为调试状态，在生产环境严格禁止使用调试状态，否则流程产生大量的日志数据会影响性能和错误排查

3. API编排中的流程如果要调用外部业务系统的API建议先在API网关中注册，API编排流程调用注册后的API，这样可以统一管理所有编排流程中使用到的API接口

4. API编排流程的命名一定要能体现此业务场景的名称，禁止使用test、测试等字样

4. API管理规范建议

良好的API即需要在上线前做好一系列的准备工作，又要在上线后保持向前兼容，因此应具备一套覆盖整个生命周期的完善的管理方案，以此规范API的整个生命周期各个阶段的工作要求，保证API的良好质量。

4.1 API需求管理

1. 需求管理

规范API需求管理，明确各阶段的人员职责、工作内容、处理流程，可以保证有效地、高质量地完成需求工作。

1.1 明确成员职责

需求开发人员负责编写业务需求报告

项目负责人负责需求评估和审核

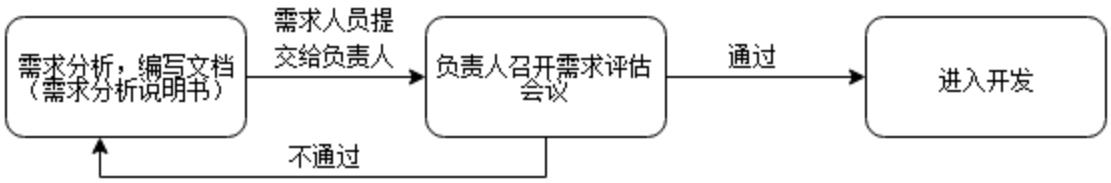
技术开发人员负责需求实现

1.2 规范需求开发流程

建议按照以下流程完成需求开发工作

需求分析：需求人员根据业务功能对需求进行可行性研究以及分析调研，编写需求分析说明书提交给项目负责人。

需求评估：项目负责人需组织需求评估会议，召集技术开发人员和需求开发人员，由需求开发人员讲解需求，技术开发人员对不清楚的问题进行提问，共同讨论需求的实现方式，实现难度等内容。最终由项目负责人总结评估，若评估部通过，则回到“需求分析”阶段重走流程；若评估通过，由项目负责人确定审核通过并提交需求分析说明书给技术开发人员，进入开发阶段。



2. 需求文档的模板制定

u 版本信息：

- 版本号
- 修订日期
- 修订人
- 修改内容

u 文档说明
- 文档简介
- 文档读者

u 目录

u API简介
- 功能定位
- 需求说明

u 详细功能说明
- 功能列表
- 流程图
- 用例图

u 非功能需求
- 性能需求
- 环境需求

u 项目规划
- 时间规划
- 版本规划

u 附录
- 技术文档
- API详细设计说明书

4.2 API发布

1. 明确成员职责

技术负责人负责发布的各个流程的审核

开发人员负责开发环境

测试人员负责测试环境

运维人员负责正式部署环境

数据库操作均由DBA统一负责

2. 规范流程

在已开发完成的API正式发布前建议按照以下流程进行上线前检查和审批。

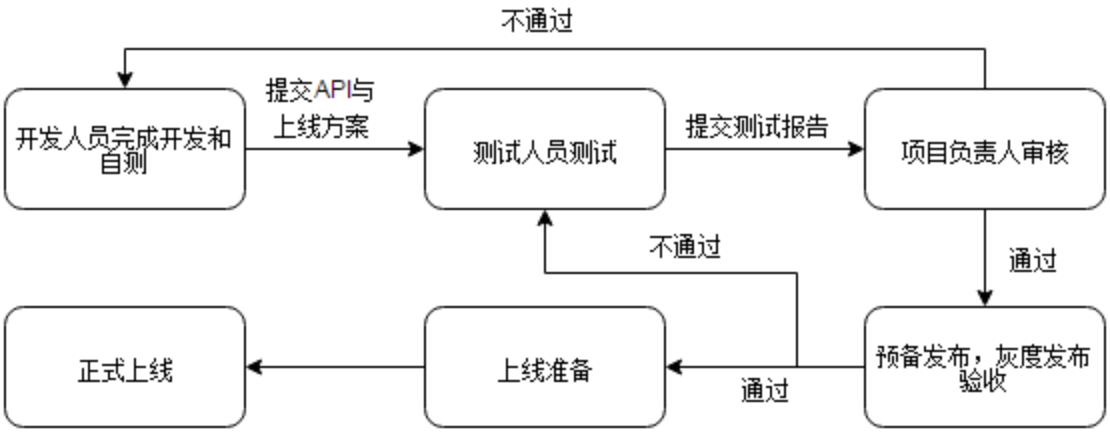
提交测试：开发人员开发完成后应先在开发环境中对API自测，通过之后方可提交给项目技术负责人审核，与API一并提交的还要有上线方案（须包括API与外部应用的关联说明，可能包含的数据库脚本，合理的上线时间以及失败的回滚步骤等说明内容），审核通过之后提交给测试人员。

测试：测试人员根据提交的上线方案内容制定测试方案，编写测试用例。通过平台编写测试计划对API进行自动化测试（涉及数据库操作则交由DBA操作），记录各种测试数据结果以及测试过程中出现的问题，汇总后编写测试结果报告交由技术负责人，若允许上线则进入下一部；若不能通过则重新交给开发人员迭代开发，再从“提交测试”重走流程。

预备发布： API通过测试并达到上线标准时，技术负责人发起通知，让相关开发人员，运维人员等准备灰度发布，通过平台灰度发布策略或选择发布范围，进行灰度发布对API验收测试。若有紧急BUG则按照上线方案的失败预案处理，而后从“测试”重走流程。若只是不影响功能的BUG可留到下个版本解决，准备正式上线。

上线准备：在确认API具备正式上线标准后，技术负责人应在上线前发起发布会议，召集相关开发人员，运维人员，DBA讨论发布事宜，根据上线方案对项目的部署发布，操作步骤，数据库的执行脚本以及失败后的回滚操作进行讨论，而后生成发布会议总结。由技术负责人将会议总结连同上线方案提交给运维人员和DBA。

正式上线：API发布后，运维人员负责日常运维工作并通知测试人员对API进行正式运行环境的线上测试。测试结果以及处理参考“测试”流程。



4.3 API管理责职

API的管理职责是API管理中的一个重要部分，通过明确定义职责，指定API的管理员，划分管理人员的职责范畴，可以使项目的所有事务管理得到统一的监督管理，解决职责不明确带来的一系列问题。

1. 明确API分类：

API建议按以下标准分为两类

l 应用下的API：在API开发平台下的应用中所包含的API，以下称为内部API。
l 独立的API：包含注册到网关的API以及通过ESB编排组成的大型API，为方便指认，以下称为外部API（不特指第三方API）。

2. 明确管理人员：

在涉及API的管理职责过程中，仅建议由项目技术负责人和开发人员参与。

3. 明确管理职责：

明确指定API的管理人员，可以方便在项目上线期间API的管理，譬如API的预警提示信息应该发给对应的管理人员，这样有利于快速解决API运行时出现的问题。

l 内部API负责人：该类API由于大部分功能较为单一，并且是由项目内部开发人员开发，开发人员对其有着清楚的了解，所以建议由API对象的开发人员负责管理。
l 外部API负责人：该类API大多数是独立于应用存在的，仅为某一业务功能所引入的第三方API或是通过ESB编排服务平台编排的大型API，所以建议根据API的业务功能范围来划分到不同项目，由项目技术负责人负责或者指定其他负责人员。

4.4 API性能监控

运维人员定时通过平台监控中心对发布的API有计划地监控其服务的性能和可用性，及时发现问题并处理，无法解决的则提交给技术负责人，由其判断如何处理。无法线上解决的则由技术负责人汇总成报告纳入版本迭代需求，能解决的则由技术负责人通知相关人员负责。

4.5 API文档索引

为方便开发人员及运维人员在API发布之后对API的管理和应用，建

议尽可能为

发布的API或支持的API提供一份列表，列表格式可参考：

API名称	API地址	API简介
mox的脚本测试示例 V1.0	/rest/mox458/queryByIdAndName	通过ID和名称查询

在列表定位到API之后，可通过API的名称和地址在平台里查找获取API的具体信息。

4.6 API迭代

API的迭代规范即API的版本控制规范，对于API在开发中的版本控制在2.5中已有说明，可通过URL的格式规范来实现。

而在API发布之后，运维人员在搜集API运行数据提交给技术负责人之后，技术负责人应根据API的运行情况制定相应的迭代计划交由开发人员进行迭代开发，然后按3.1重新走API发布的流程对API进行迭代。

4.7 API下线

1. 明确成员职责

运维人员负责项目相应数据的留存

DAB负责数据库的备份与删除

项目负责人负责所有文件的留存

2. 规范流程

下线准备：根据设置的项目下线时间进入下线流程，由运维人员通知API对应负责人，确定是否按照之前约定的下线时间正常下线，如果不是，需要告知正确的下线时间并重新备注。如果时间没问题，进入下线流程。

数据备份：运维人员搜集API运行期间的性能监控数据与运行数据的，编写成API运行的情况概括文档。DBA将API相关的数据库数据和日志文件做好备份，

下线：运维人员将API从平台下线，DBA删除API在正式运行环境中数据库里的数据和日志文件。之后运维人员将API运行情况概括文档、DBA所备份的数据和文件打包提交给API负责人。负责人将API的源代码一起打包之后，将文件存到固定的文件备份专用空间