目录

[1. 目的 2](#_Toc92372369)

[2. 总则 2](#_Toc92372370)

[3. 适用范围 3](#_Toc92372371)

[4. 生效日期 3](#_Toc92372372)

[5. 定义 3](#_Toc92372373)

[5.1. 项目追踪表 3](#_Toc92372374)

[5.2. 需求分析报告 3](#_Toc92372375)

[5.3. 开发方案 3](#_Toc92372376)

[5.4. 工作说明书 3](#_Toc92372377)

[5.5. 测试报告 3](#_Toc92372378)

[5.6. 上线报告 4](#_Toc92372379)

[5.7. 验收报告 4](#_Toc92372380)

[6. 项目管理 4](#_Toc92372381)

[6.1. 项目生命周期 4](#_Toc92372382)

[6.2. 项目生命周期实施要求 4](#_Toc92372383)

[6.3. 项目生命周期实施流程 5](#_Toc92372384)

[7. 程序质量管理 7](#_Toc92372385)

[7.1. 前端相关规范 7](#_Toc92372386)

[7.1.1. HTML 7](#_Toc92372387)

[7.1.2. 图片 7](#_Toc92372388)

[7.1.3. 命名规范 7](#_Toc92372389)

[7.1.4. JavaScript 8](#_Toc92372390)

[7.2. 后端相关规范 8](#_Toc92372391)

[7.2.1. 命名规范 8](#_Toc92372392)

[7.2.2. 常量定义 9](#_Toc92372393)

[7.2.3. 注释规范 9](#_Toc92372394)

[7.2.4. 数据库SQL 10](#_Toc92372395)

[7.2.5. 压力测试 10](#_Toc92372396)

[7.2.6. 发现并解决慢调用 10](#_Toc92372397)

[8. 系统权限管理 11](#_Toc92372398)

[8.1. 功能权限 11](#_Toc92372399)

[8.2. 数据权限 11](#_Toc92372400)

[8.3. 权限授予与变更 11](#_Toc92372401)

[9. 数据交互管理 11](#_Toc92372402)

[9.1. 内部交互 11](#_Toc92372403)

[9.2. 外部交互 12](#_Toc92372404)

[10. 迭代发布管理 12](#_Toc92372405)

[10.1. 代码提交 12](#_Toc92372406)

[10.2. 代码迭代 12](#_Toc92372407)

[10.3. 程序版本 12](#_Toc92372408)

[10.4. 程序发布 13](#_Toc92372409)

[11. 应用监控管理 13](#_Toc92372410)

# 目的

规范公司数字化建设过程中的系统与架构设计管理规范，从项目管理、程序质量、系统权限、数据交互、迭代发布、应用监控等方面规范项目团队各类行为活动；以确保项目的实施过程有章可循，成本、质量、风险得到有效的管控，促进系统建设工作得到健康、有序的推进。

# 总则

以“合理性、高效性和可落地性”为原则，制定本规范。

# 适用范围

统一中投数字化中心

# 生效日期

2022年6月

# 定义（可以增加人岗事的定义，如项目经理，产品经理等）

## 项目追踪表

《项目追踪表》项目经理用来记录和追踪所有项目的推进进程以及项目推进过程中，各环节所涉及到人、事、物的活动计划；是软件项目生命周期管理的必备文件。

## 需求分析报告

《需求分析报告》是产品经理/主管对用户单位所提的新需求、需求变更进行分析后所生成分析报告，是产品团队与需求单位间的契约性文件。

## 开发方案

《开发方案》是产品经理/主管在开发经理/主管的协助下， 以《需求分析报告》为蓝本，结合开发团队的技术实现能力，而制定的用于指导开发团队进行实施的方案文件，它是产品团队与开发团队间的契约性文件。

## 工作说明书

《工作说明书》是开发经理/主管结合开发团队人力资源及工作任务情况所拟定的开发计划书，是对《开发方案》所述开发内容的工作分解，它是开发经理/主管用来管控开发人员有序且保质保量完成开发任务的纲领性文件。

## 测试报告

测试报告是产品经理/主管携开发团队、维护团队与用户单位共同商讨制定的，用于指导并规范全体验证人员在功能发布到测试环境的前期、中期、后期 必须完成的各类活动事项；是指导和评判测试工作是否达标的标准性文件。

## 上线报告

上线报告是产品经理/主管携开发团队、维护团队 协助用户单位共同商讨制定的，用于指导并规范全体上线人员在功能发布到生产环境的前期、中期、后期 必须完成的各类活动事项；是指导和评判上线工作是否达标的标准性文件。

## 验收报告

验收报告是项目上线后，在项目经理的组织下，由产品经理/主管起草与执行，是用于评判项目是否达标的结论性文件，该文件之结论直接影响项目款项的结案与支付工作。

# 项目管理

## 项目生命周期

软件项目管理中，项目生命周期依先后顺序大体划分为：项目调研、需求分析、方案设计、开发、测试、上线、验收和运维，八个关键阶段。

## 项目生命周期实施要求

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **阶段** | **目标** | **责任人** | **执行人** | **协作人** | **交付物** |
| **项目调研** | 对代表性的用户进行走访、调查与研究；从需求之目标、真伪、合理、效益及可行性等方面进行甄别，并与相关协同单位达成一致意见后，将需求整理成文并提报立案申请 | 用户单位负责人 | 用户单位需求经理 | 产品经理/主管 | 《立案签呈》/《资讯作业委托单》 |
| **需求分析** | 以项目调研阶段《立案签呈》/《资讯作业委托单》中所述之需求说明为方向，对需求进行分析与解读，形成用户单位与产品团队间的契约性文件《需求分析报告》 | 项目经理 | 产品经理/主管 | 用户单位需求经理 | 《需求分析报告》  《项目追踪表》 |
| **方案设计** | 以需求分析阶段《需求分析报告》为实施方案之宗旨，对报告中的需求内容逐一进行分解并制定相应的实施方案，形成产品团队与开发团队间的契约性文件《开发方案》 | 项目经理 | 产品经理/主管 | 开发经理/主管 | 《开发方案》  《项目追踪表》 |
| **开发** | 围绕方案设计阶段《开发方案》对开发团队的人、事、物进行任务分配，形成指导开发团队日常工作的《工作说明书》 | 项目经理 | 开发经理/主管 |  | 《工作说明书》  源代码  《项目追踪表》 |
| **测试** | 拟定测试验证计划，模拟业务场景，从功能 和 性能层面对开发成果进行测试验证及优化改善，确保符合《需求分析书》、《开发方案》中所述之内容要求，并形成《测试报告》 | 项目经理 | 产品经理/主管 | 用户单位需求经理  开发经理/主管  运维经理/主管 | 《测试报告》  《项目追踪表》 |
| **上线** | 开发成果测试验证通过后，在用户单位的统筹协调之下，协助拟定上线计划 并 配合用户单位做好开发成果的发布及发布后各项功能的验证工作，确保上线工作的顺利开展，上线完成后形成《上线报告》 | 用户单位负责人 | 用户单位需求经理 | 产品经理/主管  开发经理/主管  运维经理/主管 | 《上线报告》  《项目追踪表》 |
| **验收** | 项目上线并稳定运行一段时间后，组织用户单位进行项目验收工作，完成项目交付，并形成《验收报告》 | 项目经理 | 产品经理/主管 | 用户单位需求经理  开发经理/主管 | 《验收报告》  《项目追踪表》 |
| **运维** | 确保项目稳定可靠的为用户提供服务，定期形成《运维报告》 | 项目经理 | 运维经理/主管 | 产品经理/主管  开发经理/主管  用户单位需求经理 | 《运维报告》 |

交付物模板：



这些附件名字有些紧凑，可能会影响读者直观识得文件名称，建议重新附加。

## 项目生命周期实施流程

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **阶段** | **流程** | | | | **交付物** |
| **用户单位执行人** | **用户单位责任人** | **IT单位责任人** | **IT单位执行人** |
| **项目调研** | 调研  申请立案 | 审批 | 审批 |  | 《立案签呈》/《资讯作业委托单》 |
| **需求分析** | 确认  确认 |  |  | 需求分析 | 《需求分析报告》、《项目追踪表》 |
| **方案设计** | 确认 |  | 确认 | 方案设计 | 《开发方案》、《项目追踪表》 |
| **开发** |  |  | 确认 | 任务执行  任务分解 | 源代码、《项目追踪表》  《工作说明书》 |
| **测试** | 执行测试  确认 |  | 确认 | 执行测试  测试计划 | 《测试报告》、《项目追踪表》 |
| **上线** | 执行上线  上线计划 | 确认 | 确认 | 执行上线  确认 | 《上线报告》、《项目追踪表》 |
| **验收** | 执行验收  确认 | 确认 | 确认 | 验收计划 | 《验收报告》、《项目追踪表》 |
| **运维** |  |  | 审批 | 运维 | 《运维报告》 |

# 程序质量管理

## 前端相关规范

### HTML(建议加上样例)

1. 标签
   1. 所有具有开始标签和结束标签的元素都要写上起止标签，某些允许省略开始标签或结束标签的元素亦都要写上。
   2. 空元素 标签都不加”/”字符。

推荐：

<div>

<h1>我是h1标题</h1>

<p>我是一段文字</p>

</div>

<br>

不推荐：

<div>

<h1>我是h1标题</h1>

<p>我是一段文字

</div>

<br/>

1. 书写风格
   1. HTML标签名、类名、标签属性和大部分属性统一用小写。

推荐：<div class=”demo”></div>

不推荐：<div CLASS=”DEMO”></div>

* 1. HTML文本、CDATA、JavaScript、meta标签某些属性等内容可大小写混合。

例：

1. 优先使用 IE 最新版本和Chrome Frame

<meta http-equiy=”X-UA-Compatible” content=”IE-edge,chrome=1” />

1. HTML文本内容

<h1>I AM WHAT I AM</h1>

1. JavaScript 内容

<script type=”text/javascript”>

let demoName=”demoName”;

</script>

1. CDATA 内容

<script>

<!CDATA[...]]

</script>

1. 类型属性
   1. 不需要为CSS、JS指定类型属性，HTML默认已包含。

推荐：

<link rel=”stylesheet” href=””>

<script src=””></script>

不推荐：

<link rel=”stylesheet” type=”text/css” href=””>

<script type=”text/javascript” src=””></script>

1. 元素属性
   1. 元素属性值使用双引号语法。
   2. 元素属性值可以写上的都写上。

推荐：

<input type=”text”>

<input type=”radio” name=”name” checked=”checked”>

不推荐

<input type=text>

<input type=’text’>

<input type=”radio” name=”name” checked>

1. 特殊字符引用
   1. 在HTML中不能使用小于号”<”和大于号”>”，浏览器会将它们作为标签解析，若要正确显示。在HTML源代码中使用字符实体。

推荐：

<a href=”#”>more&gt;&gt;</a>

不推荐：

<a href=”#”>more>></a>

1. 代码嵌套
   1. 元素嵌套规范，每个块状元素独立一行，内联元素可选。

推荐：

<div>

<h1></h1>

<p></p>

</div>

<p><span></span><span></span></p>

不推荐：

<div>

<h1></h1><p></p>

</div>

<p>

<span></span>

<span></span>

</p>

段落元素与标题元素只能嵌套内联元素

推荐：

<h1><span></span></h1>

<p><span></span></p>

不推荐：

<h1><div></div></h1>

<p><div></div></p>

### 图片(是否需要增加图片在不同场景使用的分辨率建议值)

1. 内容图：内容图多以商品图等照片类图片形式存在，颜色较为丰富，文件体积较大，优先考虑JPEG格式，条件允许的话优先考虑WebP格式，尽量不适用PNG格式。
2. 背景图：背景图多为图标等颜色比较简单、文件体积不大、起装饰作用的图片
   1. 优先考虑使用PNG格式；
   2. 图像颜色比较简单的,如块色块线条图标,优先考虑使用PNG8格式，避免使用JPEG格式；
   3. 图像颜色丰富而且图片文件不太大的(40kb以下)或有半透明效果的优先考虑使用PNG24格式；
   4. 图像颜色丰富而且文件比较大的（40kb-200kb）优先考虑JPEG格式；
   5. 条件允许的，优先使用WebP代替PNG和JPEG格式。

3 图片质量：

上线的图片都应该经过压缩，压缩后的图片不应该出现肉眼可感知的失真区域

60质量的JPEG格式图片必须大于60的相比，肉眼已看不出的区别，因此保存JPEG图的时候，质量一般控制在60，若保真度要求高的图片可适量提高到80，图片大小控制在200KB以内

4 图片压缩

4.1 工具压缩

Tinypng 免费、批量速度快

Squoosj 在线图片压缩

Compressor 支持JPG、PNG、SVG、GIF

4.2 webpack

使用image-webpack-loader项目打包时进行压缩

团队约定：

国内普通家庭的宽带基本能达到8mbs，实际速率大约为500-900KB/s,全国3G/4G用户占有比超过了50%，为了保证图片能更好的加载展示给用户看：建议

PC平台单张的图片的大小不应大于200kb

移动平台单张的图片的大小不应大于100kb

单一色调，小图标建议使用目前主流方案iconfont

当一个页面使用多张图片时，建议使用雪碧图，减少请求次数

### 命名规范(建议加上样例)

1. 项目命名：全部采用小写方式 ，以中划线分隔。

正例：mall-management-system

反例：mall\_management-system / mallMAnagementStstem

1. 目录命名：全部采用小写方式，以中划线分隔，有复数结构时，要采用复数命名法，缩写不用算法。

正例：scripts / styles / components / images / utils / layouts /demo-styles / demo-scripts / img / src

反例：script / style / demo\_scripts / demoStyles / imgs /docs

1. JS、CSS、HTML、PNG、LESS、SASS文件命名：全部采用小写，以中划线分隔。

正例：render-dom.js / signup.css / index.html / company-logo.png

反例：renderDom.js / UserManagement.html

1. 命名严谨性：代码中的命名严禁使用拼音与英文混合的方式，更不允许使用中文方式。说明：正确的英文拼音和语法可以让阅读者易于理解，避免歧义，纯拼音也要避免采用。
2. React&JSX书写规范

基本规则

1. 每个文件只包含一个React类组件
   1. 但是多个函数式组件可以放到一个文件中
2. 一般使用JSX语法

命名规范

1. 组件文件扩展名
   1. 如果使用JavaScript，则文件扩展名为.js;如果使用TypeScipt，则文件扩展名为.tsx
2. 组件文件名
   1. 如果是组件文件，则使用PascalCase,如MyComponent.js
   2. 如果组件是一个目录，则组件入口命名index，如index.js
3. 引用命名
   1. React组件使用PascalCase，组件实例使用CamelCase

推荐：

import ReservationCard from ‘./ReservationCard’;

const reservationCard = <ReservationCard />

不推荐：

import reservationCard from ‘./ReservationCard’;

const ReservationCard = <ReservationCard />

1. 组件命名
2. 使用文件名为组件名字，例如：ReservationCard .js应高包含ReservationCard 的引用，然而对于文件夹的根组件，使用index.js 作为文件名，使用文件的名字作为组件的名字

推荐：

import Footer from ‘./Footer’

不推荐：

import Footer from ‘./Footer/index’

import Footer from ‘./Footer/Footer

### JavaScript(是否有什么工具进行统一格式化以保持代码风格一致性，增加可阅读性)

1. 命名
   1. 采用小驼峰命名lowerCamelCase，代码中均不能以下划线开始命名也不能以下划线或美元符号结束。
   2. 方法名、参数名、成员变量、局部变量统一使用lowerCamelCase风格，必须遵从驼峰形式，其中方法命名必须是动词或动词+名词形式

正例：saveShopCarData / openShopCarInfoDialog

反例：save/open/show/go

* 1. 常量命名全部大写，单词间用下划线隔开，力求语义表达式完整清楚

正例：MAX\_STOCK\_COUNT

反例：MAX\_COUNT

1. 优先使用ES6 及ES7语法糖和函数。以利于简化程序，并让代码更加灵活和可复用。

注：强烈推荐使用ES6、ES7的新语法，比如：箭头函数、await/async，解构，lei，const，for...of等。

1. 下列关键字后必须有大括号（即使代码块的内容只有一行）：

if,else,for,while,do,switch,try,catch,finally,with

1. undefined判断：永远不要使用undefined进行判断；使用typeof 和字符串’undefined’对变量进行判断。
2. 条件判断和循环判断最多建议三层。

条件判断能使用三目运算符和逻辑运算符解决的，就要使用条件判断，但是谨记不要写太长的三目运算符。如果超过三层请抽成函数，并写清楚注释。

1. 慎用console.log。因console.log大量使用会有性能问题，在非webpack项目中谨慎使用log功能。

7 前端建议使用Vs Code进行开发，prettier插件自动格式化代码

## 后端相关规范

### 命名规范

1. 代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

反例： \_name/\_\_name/$Object/name\_/name$/Object$

1. 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

反例： DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

正例： alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文

1. 类名使用UpperCamelCase风格，必须遵从驼峰形式，但以下情形例外：（模型的相关命名） Model/ DTO / VO等。

正例：MarcoPolo / User / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / user / XMLService / TCPUDPDeal / TAPromotion

1. 方法名、参数名、成员变量、局部变量都统一使用lowerCamelCase风格，必须遵从驼峰形式。

正例： localValue / getHttpMessage() / inputUserId

1. 抽象类命名使用Abstract或Base开头；异常类命名使用Exception结尾；测试类命名以它要测试的类的名称开始，以Test结尾。
2. 中括号是数组类型的一部分，数组定义如下：String[] args;

反例：请勿使用String args[]的方式来定义

1. 包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。

正例： 应用工具类包名为com.pec.crc.util、类名为MessageUtils（此规则参考spring的框架结构）

1. 接口和实现类的命名有两套规则：
   1. 对于Service和DAO类，基于SOA的理念，暴露出来的服务一定是接口，内部的实现类用Impl的后缀与接口区别。

正例：CacheServiceImpl实现CacheService接口。

* 1. 如果是形容能力的接口名称，取对应的形容词做接口名（通常是–able的形式）。 正例：AbstractTranslator实现 Translatable。

1. 各层方法命名规范：

Controller、Service、Dao/Mapper层方法命名规范

获取单个对象的方法用get做前缀。

获取多个对象的方法用list、find、obtain做前缀。

获取统计值的方法用count做前缀。

插入的方法用save（推荐）或insert做前缀。

删除的方法用remove（推荐）或delete做前缀。

修改的方法用update做前缀。

### 常量定义

1. 不允许出现任何魔法值（即未经定义的常量）直接出现在代码中。

反例： String key="Id#taobao\_"+tradeId； cache.put(key, value);

1. long或者Long初始赋值时，必须使用大写的L，不能是小写的l，小写容易跟数字1混淆，造成误解。
2. 不要使用一个常量类维护所有常量，应该按常量功能进行归类，分开维护。如：缓存相关的常量放在类：CacheConsts下；系统配置相关的常量放在类：ConfigConsts下。

说明：大而全的常量类，非得使用查找功能才能定位到修改的常量，不利于理解和维护。

### 注释规范(建议加上样例)

1. 类、类属性、类方法的注释必须使用Javadoc规范，使用/\*\*内容\*/格式，不得使用//xxx方式。
2. 所有的抽象方法（包括接口中的方法）必须要用Javadoc注释、除了返回值、参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。
3. 所有的类都必须添加创建者信息。
4. 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/\* \*/注释，注意与代码对齐。
5. 代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。
6. 注释掉的代码尽量要配合说明，而不是简单的注释掉。

说明：代码被注释掉有两种可能性：1）后续会恢复此段代码逻辑。2）永久不用。前者如果没有备注信息，难以知晓注释动机。后者建议直接删掉。

### 数据库SQL(建议加上样例)

1. 依据业务实际需要，可以考虑读写分离，分库分表机制。
2. 结合业务逻辑，常需要用来作为搜索、排序 等用途的字段，应考虑建立索引。
3. SQL书写前，应先观察分库分表情况 及 索引情况，尽可能让所书写的SQL用上 分库分表键 及 筛选效率较高的索引。
4. 在多表联合查询、更新等情况下，关联条件需要使用索引，且需要更多的使用相同字段类型的列来关联(例如int与int类型的列做关联条件)。
5. 结合业务逻辑，针对web端显示等情况，尽可能多的使用分页查询，避免一次性查询过多的数据。
6. 做写入或者delete等操作时，可以考虑分批次执行，避免一次性执行过多的数据。
7. 字段的种类尽可能用int 或者tinyint类型。另外字段尽可能用NOT NULL。
8. 尽量避免在 where 子句中使用!=或<>操作符，否则可能使mysql放弃使用索引而进行全表扫描。
9. 尽量避免在 where 子句中使用 or 来连接条件。
10. 慎用in 和 not in 。
11. 避免使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。
12. Update 语句，如果只更改 1、2 个字段，不要 Update 全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志。
13. SQL书写后，应观察该SQL的执行计划，借助执行计划反馈出来的信息，不断优化，提升SQL执行效率。
14. 结合实际业务场景，定期转移生产环境中的历史数据。

### 压力测试(建议加上样例)

为保障系统上线后的稳定运行，按当前及未来一个时期业务量的发展情况，设定压力测试范围，对各接口服务进行压力测试，通过压力测试（压测工具 如：阿里云性能测试PTS 等），对相关的软硬件及网络资源做出合理评估，压力测试通过后，需将测试结论附加到《测试报告》中。

### 发现并解决慢调用(建议增加监控频率以便及时发现慢调用的发生)

借助监控服务（如：阿里云监控产品 等）及时发现慢调用，并及时进行排查、优化慢调用。

# 系统权限管理(建议加上样例、图例)

系统权限管理，一般是指根据系统设置的安全规则或安全策略，用户可以访问且只能访问授权范围内的资源（包含页面、操作、数据、图片、文件等），避免因权限控制异常、操作不当等引发资安等风险问题，如：越权操作、数据泄露等。通常情况下，我们将权限划分为：功能权限 及 数据权限。

## 功能权限

用户能否打开或操作某个功能界面，取决于该用户 或 该用户所对应的角色 是否被授予了该功能界面所对应的菜单之权限。换言之，用户登录系统成功后，所能看到的所有功能菜单，即为该用户所拥有的全部功能的集合。

## 数据权限

用户仅有“功能权限”，若没有“数据权限”，则不足以支撑用户的正常操作，需要对用户进行“数据权限”的授权之后，方可正常使用该功能。以“订单管理”功能画面为例：A用户被授予了“订单管理”功能菜单，但该A用户未被授予任何数据权限；此时，A用户可以正常打开该功能界面，但是无法在该界面对任何组织的订单数据进行操作（如：查询订单、新增订单、修改订单）；若成功向权限管理员申请到“XX营业所”的数据权限后，则A用户就可以在“订单管理”界面中对“XX营业所”的订单进行相应的操作了。

## 权限授予与变更

“功能权限”及“数据权限”应依据用户实际工作之需求在OA系统中提报“账号申请单”，经各级主管审批同意后，由相应“系统权限管理员”或“权限管理接口服务”进行权限授予与变更。

# 数据交互管理(建议增加数据权限管理和交互监控管理，以保障数据交互的安全性和稳定性)

数据交互分为内部交互、外部交互。

## 内部交互(建议加上样例、图例，以及安全机制要求等)

1. Api服务模式
2. 若交互双方均支持阿里云环境，推荐DTS+ADB模式
3. dblink模式（不推荐）

## 外部交互

api服务（注意：必要的数据脱敏与权限管控）

# 迭代发布管理(建议增加相关章节对应的图例增强本规范的理解性)

## 代码提交

1. 代码更新迭代通过gitlab上传下载
2. gitlab每个工程分别对应三个主要分支dev(开发环境)、test(测试环境)、master(正式/生产环境)
   1. dev和test分支代码提交代码权限可下放各开发人员。
   2. Master分支代码管理权限给项目leader或代理人，负责代码合并

## 代码迭代

1. 开发代码在dev环境进行，功能开发，内部测试完毕后推送至test分支发布测试环境
2. 若测试有问题且dev环境正在开发其他模块代码，从test远程获取分支做调整，无误后提交test并合并到dev分支。
3. test分支测试无误后，由test分支推送至master分支。
4. 每次代码合并之后必须打对应的tag。

## 程序版本

1. 标准的版本号必须（MUST）采用 X.Y.Z 的格式，其中 X、Y 和 Z 为非负的整数，且禁止（MUST NOT）在数字前方补零。X 是主版本号、Y 是次版本号、而 Z 为修订号。每个元素必须（MUST）以数值来递增。例如：1.9.1 -> 1.10.0 -> 1.11.0。
2. 标记版本号的软件发行后，禁止（MUST NOT）改变该版本软件的内容。任何修改都必须（MUST）以新版本发行。
3. 主版本号为零（0.y.z）的软件处于开发初始阶段，一切都可能随时被改变。这样的公共 API 不应该被视为稳定版。
4. 1.0.0的版本号用于界定公共 API 的形成。这一版本之后所有的版本号更新都基于公共 API 及其修改内容。
5. 修订号 Z（x.y.Z | x > 0）必须（MUST）在只做了向下兼容的修正时才递增。这里的修正指的是针对不正确结果而进行的内部修改。
6. 次版本号 Y（x.Y.z | x > 0）必须（MUST）在有向下兼容的新功能出现时递增。在任何公共 API 的功能被标记为弃用时也必须（MUST）递增。也可以（MAY）在内部程序有大量新功能或改进被加入时递增，其中可以（MAY）包括修订级别的改变。每当次版本号递增时，修订号必须（MUST）归零。
7. 主版本号 X（X.y.z | X > 0）必须（MUST）在有任何不兼容的修改被加入公共 API 时递增。其中可以（MAY）包括次版本号及修订级别的改变。每当主版本号递增时，次版本号和修订号必须（MUST）归零。

## 程序发布

程序发布在流程上须符合本规范中项目管理章节所述之条款进行管控；程序发布时，通过打tag来区分当前的版本和之前运行正常的版本，tag命名可根据项目发布次数或者应用程序内部版本命名 如:v.0.0.1 依次类推v.0.0.2……

1. 针对旧版本的修复版命名

如:v0.0.1.1

1. 新功能上线 命名v0.0.2……
2. 发布使用kenkis，选择所需发布的tag进行。

# 应用监控管理(建议增加图例增强本规范的理解性，增加监控频率和责任人)

应用监控体系建设纬度及具体措施如下：

1. 基础设施维度监控，如cpu，内存，硬盘，网络，服务器存活性等；
2. 应用维度监控，如程序进程存活，程序端口存活，应用url可用性、延迟等；
3. 中间件维度监控，如jvm性能监控，redis性能监控，mysql性能监控，mq性能监控等；
4. 应用链路维度监控，如各服务间的接口调用性能，接口调用次数，接口相互间调用链路等；
5. 日志体系，如应用日志格化输出，应用日志采集，应用日志访问等。

目前，云上系统比较方便的集成多种类型的监控策略，例如，arms、redis监控、数据库监控等、日志系统等。IDC部署程序则需要自行开发监控系统或整合第三方开源监控软件进行多纬度、多策略的监控。