|  |
| --- |
| IBG [质量管理部][标准过程文档] |
| 文档编号:IBG-H07-SPD-20160408-01 |

****

配置管理

代码管理规范

|  |  |
| --- | --- |
| 作者 | 董佳玲 |
| 版本号 | V1.2 |
| 更新日期 | 2016-04-08 |
| 安全级别 | 内部公开 |

修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 属性 | 内容简述 | 作者 | 版本号 |
| 2016-03-31 | 创建 | 创建文档 | 董佳玲 | V1.0 |
| 2016-04-01 | 修改 | 根据董兵/余晟/张建荣/马荣磊提出的建议修改 | 董佳玲 | V1.1 |
| 2016-04-08 | 修改 | 根据前端黄凯等意见增加部分说明 | 董佳玲 | V1.2 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**备注：属性选项：创建、追加、修改、删除、评审(如果是评审，请在内容简述中注明，是否批准发布)。**

目录

1 目的 4

2 适用范围 4

3 代码库管理 4

3.1 概述 4

3.2 命名规则 4

3.3 目录结构 5

3.4 创建流程 5

3.5 权限管理 5

3.5.1 权限群组 5

3.5.2 权限申请 6

3.5.3 权限变更 6

3.5.4 权限分类、级别与对应角色 6

3.6 使用规范 8

4 分支管理 9

4.1 概述 9

4.2 分支策略说明 9

4.3 分支使用规范 10

4.4 管理职责划分 11

5 标识规范 11

5.1 版本号标识 11

5.2 基线标识 12

5.2.1 缩写定义 12

5.2.2 基线标识规范 13

6 流程图 13

6.1 流程说明 13

6.2 图示 14

# 目的

为了规范代码管理，保障各产线产品代码的完整性、一致性和可维护性，特制订本规范。

# 适用范围

本规范适用于基础事业群各产线代码管理。

# 代码库管理

### 3.1 概述

代码库指使用版本管理工具（如GIT）所创建的用于存放产品开发过程的源代码以及相关内容等、可以进行代码版本控制的库。每一个完整的独立开发、部署及发布的产品，可创建一个独立的代码库。涉及保密信息的产品，需单独创建代码库，避免机密信息泄露。

### 3.2 命名规则

**代码库:【产品线代号-产品代号-应用类型-特殊标识】**

说明：

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **项目** | **说明** | **是否必填** |
| 1 | 产品线代号 | 产品线的名称代号，代号固定，如下表。 | 必填 |
| 2 | 产品代号 | 产品线包含的各产品或应用的名称代号，由开发经理定义，需使用英文简称，并确保统一。 | 必填 |
| 3 | 应用类型 | 包含Web、App(IOS/Android) 、API、客户端、服务类、Job程序等这些类型，需使用英文简称，并确保统一。 | 必填 |
| 4 | 特殊标识 | 一般用于区分同产品或应用的不同用途的代码库等情况。 | 选填 |

产品线代号表：

|  |  |
| --- | --- |
| **产品线代号** | **对应中文名称** |
| Base | 架构产品线 |
| CC | CCTalk产品线 |
| Class | 网校产品线 |
| Common | 公共业务产品线 |
| Community | 社区产品线 |
| Portal | 门户产品线 |
| Tools | 工具产品线 |

举例：

工具线沪江小D词典代码库：Tools-Dict-Web

### 3.3 目录结构

详细目录结构如下：代码库目录结构固定，必要时可依据实际调整。APP端的目录结构可按实际开发要求确定，无需依据下表。

|  |  |
| --- | --- |
| **代码库名称** | **一级目录** |
| **代码库** |  |
|  | src 源码 |
|  | lib 第三方类库 |
|  | doc 文档(可选，一般可存放代码编写要求、说明等文档。需求、设计等文档不建议存放在此，请存放到专用文档库) |
|  | build 构建输出 |
|  | deploy 可部署文件 |

### 3.4 创建流程

代码管理工具使用git，git服务器一般只用于存放代码库，如有其余库需存放，需在库名称中清楚标识库类型。创建代码库的权限可开放给配置管理员、开发经理。创建代码库前，需确保已提前创建了相应的群组。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **步骤** | **步骤说明** | **角色** |
| 1 | 申请新增代码库 | 一般立项后，在OA系统发起申请流程，需说明代码库名称、代码库用途及使用人员列表。 | 开发经理 |
| 2 | 创建代码库 | 正式用代码库：申请批准后，按要求创建，需创建在相应的各产品线等群组空间下，不可创建在个人空间下。 | 配置管理员 |
| 机密代码库：申请批准后，按要求创建，需创建在相应的机密群组空间下。 | 配置管理员 |
| 研发性项目的代码库：申请批准后，按要求创建，可创建在个人空间，等到能被正式使用转移到相应的群组空间下。 | 开发经理 |
| 个人私有库：申请批准后，按要求创建，创建在个人空间下，配置管理员需定期进行确认并清理。 | 开发经理 |
| 3 | 创建master、develop分支 | 建好库后，创建master分支，从master分支创建develop分支，同时设定master、develop分支为protected branches。 | 该库创建人 |

### 3.5 权限管理

#### 3.5.1 权限群组

需提前创建及设定好各群组，用于创建代码库及辅助设定权限。

* 1. 产品线群组：一般各产品线需分别创建对应的群组，由产品线负责人申请。
  2. 横线群组：一般横线可按需要创建对应的群组，由横线负责人申请。
  3. 产品群组：如有需要，产品线下包含的各产品或应用可申请创建单独的群组，由开发经理申请。
  4. 机密群组：保密的产品需申请创建单独的群组，由该产品负责人申请。

#### 3.5.2 权限申请

1. 各人员参考权限说明，确认需要的权限；
2. 申请人员统一在OA系统提交申请，申请时需注明代码库链接（链接中包含代码库名称及所属群组）、申请原因等。另如已了解权限类型，可注明下，如是申请多人权限，可注明下使用人员。



备注：此申请单也可用于申请创建代码库，具体申请要求参阅上文。

1. 申请通过后，正式及机密的代码库由配置管理员统一设定权限，其余库可由开发经理设定权限。

#### 3.5.3 权限变更

需定期检查各群组及各库成员，如有人员变动，需及时变更人员权限；如有人员离职，需及时回收所有权限。

#### 3.5.4 权限分类、级别与对应角色

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **权限分类** | **权限设置方法** | **权限级别** | **拥有权限说明** | **角色** |
| 单个代码库 | 加入project members | Guest | / | / |
| Reporter | 可下载（clone）代码 拉取（pull）代码 | 其他人员 |
| Developer | 可下载（clone）代码 拉取（pull）代码 创建新分支 合并（merge）代码 增加新标签 推送（push）到非保护分支 | 开发人员 |
| Master | 可下载（clone）代码 拉取（pull）代码 创建新分支 合并（merge）代码 增加新标签 修改或删除标签 设置保护分支 推送（push）到保护和非保护分支 管理project members成员及权限级别 配置该代码库hooks 编辑代码库设置 | 开发经理指定的feature分支负责人 |
| Owner | 如上master级别的所有权限 修改代码库的公开程度 将代码库转移到其他群组空间下 删除该代码库 | 开发经理或配置管理员 |
| 群组空间下所有代码库 | 加入group members | Guest | / | / |
| Reporter | 查看群组 | 其他人员 |
| Developer | 查看群组 | 公用的开发人员（需经常使用群组空间下所有代码库） |
| Master | 群组空间下所有代码库的如下权限： project members中master级别的所有权限 查看群组、编辑群组 创建代码库到该群组空间下：代码库创建限制数目设置为多少，就可创建多少库 | 产品线或横线等负责人 各开发经理 配置管理员 |
| Owner | 群组空间下所有代码库的如下权限： project members中owner级别的所有权限 查看群组、编辑群组 创建代码库到该群组空间下 管理群组成员及权限级别 删除群组 新建git账号，设定代码库创建限制数目 | hjgitowner（GitLab的管理员账号） |

备注：机密代码库可按实际要求变更各角色。

### 3.6 使用规范

1. 所有产品开发过程的源代码均需存放在对应的代码库，不可随意存放，避免丢失。
2. 建库后需制定合适的分支策略，依据分支策略规划分支及做开发。
3. 开发过程中需及时获取（fetch）或拉取（pull）远程代码库的代码，并及时将本地的代码推送（push）到远程代码库，防止代码丢失。
4. 初次提交代码（commit）时请注意正确配置用户信息，以方便别人和自己查看代码提交记录。
5. 提交代码前，需先检查代码是否有遗漏，并在本地自行验证，确保代码正确性，不要提交自己都没有明白的代码以及有明显逻辑错误或bug的代码。
6. 合并代码（merge）时有冲突的文件，需手动或使用工具解决完所有冲突后再提交。
7. 推送（push）代码到git服务器之前，需先检查配置及远程服务器URL是否正确。
8. 审核（review）代码的时候如看到有删除的代码，需确定下是否是写代码的同事自己删除的，并确认是否是代码回溯。
9. 尽量避免代码回溯，避免代码丢失风险。
10. 提交（commit）代码时需按要求填写注释，便于查询及追溯。在满足基本规则约束(字符限制/前缀描述)的前提下，要表达的核心是让其他人看得懂此次提交做了什么。

注释规则：**前缀<英文冒号><英文空格><标题>**

说明：

|  |  |  |
| --- | --- | --- |
| **前缀** | **说明** | **示例** |
| add | 添加新特性 | add: 增加redis客户端统一实现 |
| update | 更新特性 | update: 更新redis客户端的连接参数 |
| fix | 解决bug | fix: 解决redis客户端连接超时的bug |
| refactor | 重构 | refactor: 重构redis客户端实现 |
| remove | 移除 | remove: 移除掉无用的redis客户端文件 |

要求：

* + 1. 禁止无意义的标题；

举例：

错误示例：

fix bug

save

release

add

......

正确示例：

add: 新增了对调用者的IP限制功能

fix: 修正了3月12日用户提出的图片无法显示的问题

refactor: 改用LinQ让代码更简洁

* + 1. 限制标题字数在50个字符内；
    2. 不要用句号结束标题行；
    3. 尽量多提交（commit），每次提交的粒度要细，如果一次提交既有bug修复又有新特性, 应该分两次提交。

# 分支管理

### 4.1 概述

本分支策略为不同的分支分配了非常明确的角色，并且定义了使用场景和用法。除了用于需求开发的分支，它还使用独立的分支进行发布前的准备、记录以及后期维护。使开发和发布工作职责明确、有序进行、易于维护、避免混乱、便于责任追溯。

### 4.2 分支策略说明

|  |  |  |
| --- | --- | --- |
| **分支类型** | **分支命名方式** | **分支使用说明** |
| 主分支 | master | master分支始终和最新的生产发布的代码保持一致。master分支禁止做任何开发用，只用于生产发布后合并对应的代码及打生产标签用。  对于App端，首次生产发布后，合并对应的代码到master分支并打生产标签，后续可按实际需要继续使用该生产标签进行生产发布。 |
| 合并分支（或称集成分支） | develop | develop分支是保存当前最新开发成果的分支。develop分支从master分支派生，不做开发用，只用于合并feature分支的代码及打合并测试标签用。 |
| feature分支（需求开发分支等） | feature/需求编号-创建日期 | feature分支通常是在开发一项新的软件功能的时候使用。feature分支从develop分支创建，最终必须合并回develop分支或者被抛弃掉（例如实验性且效果不好的代码变更）。feature分支有即将生产发布要求时，需合并回develop分支。 |
| release分支（发布分支） | release/版本号 | release分支是为发布新版本而设计的。release分支从develop分支派生，不做开发用。成功的派生了release分支后，需赋予版本号，注意所有在即将生产发布的版本之外的业务需求一定不能混到release分支（避免引入一些不可控的缺陷）。release分支生产发布后，必须合并回master和develop分支。 |
| hotfix分支（紧急生产修复分支） | hotfix/生产bug编号-创建日期 | 当生产环境中的软件遇到了异常情况或者发现了严重到必须立即修复的软件缺陷时，从master分支上指定的“生产标签”派生hotfix分支来组织代码的紧急修复工作。hotfix分支生产发布后，必须合并回master和develop分支。 |

注意事项：

* 如使用“git工作流”操作， “feature/”、“release/”、“hotfix/”前缀为自带，无需重复输入；
* feature分支名称：需求编号需写全称，例如HJCLASS-6007；在无需求编号的情况下，可使用需求名称替代，需求名称使用英文简称；
* hotfix分支名称：bug编号需写全称，例如[CTSEVER-1414](http://jira.yeshj.com/browse/CTSEVER-1414)；在无bug编号的情况下，可使用bug名称替代，bug名称使用英文简称；
* release分支名称：版本号是指计划生产发布用的版本的版本号；
* 分支名称的日期格式：均为YYMMDD，年份取后两位。

### 4.3 分支使用规范

* 1. 代码库分支数目：总的分支数目确保不要过多，避免分支过多导致混乱及占用服务器资源等。
  2. feature分支创建与使用要求：

1. 需从develop分支上稳定的节点处创建，避免引入不稳定的代码(建议从与最近的已生产发布的代码一致的节点处创建)。
2. 每个feature分支必须由开发经理指定负责人，只有feature负责人可合并feature分支的代码到develop分支，其余人不可合并。
3. feature分支有即将生产发布要求并测试通过后，才可合并，不可任意合并。
   1. 分支合并要求：每次代码合并时，合并人员需在本地合并完成并自行验证通过后，才可push到远程代码库。
   2. 分支对应提测要求：
4. feature分支：feature分支开发完成，实现了必需的软件功能；feature分支在即将生产发布的版本中要求包含的，或在近期内要求生产发布的。尽量避免出现一个feature分支测试完毕后，却长期无生产发布要求的情况。同时提测的feature分支建议不要过多，需合理安排每批生产发布的需求，并合理规划分支。
5. develop分支：有即将生产发布要求的所有feature分支均测试通过，并全部合并到develop分支；合并完成后由开发经理确认，确保代码合并没有遗漏、冲突均已解决等。
   1. 分支对应测试环境：
6. 模块测试、合并测试、验证测试环境分别对应不同的分支。模块测试对应feature分支，合并测试对应develop分支，验证测试对应release分支。
7. 各测试环境的版本需从相应的分支制作。
   1. 分支删除规定：
8. master和develop分支禁止删除。
9. feature分支开发的需求或功能生产发布后，确认无问题便可删除，也可定期批量删除旧的feature分支。
10. release或hotfix分支在每次生产发布完成，代码合并回master和develop分支后，确认无问题便可删除。需尽量确保代码库当前在用的release分支只有一个，hotfix也一样。
    1. 标签删除规定：master分支禁止删除标签，如有误打可由配置管理员删除；develop分支不建议删除标签，如有需要可由开发经理删除。

### 4.4 管理职责划分

| **角色** | **职责** |
| --- | --- |
| 开发经理 | 1. 如有初始源代码，导入初始源代码及打初始基线标签； 2. release、hotfix分支创建； 3. release或hotfix分支的代码合并到master和develop分支； 4. develop分支打标签； 5. 确认feature分支的代码合并、同步或删除； 6. 删除hotfix、release分支； 7. 协助代码分支及标签正确性检查。 |
| feature分支负责人（由开发经理指定） | 1. 创建feature分支； 2. feature分支的代码合并到develop分支； 3. feature分支同步最新的生产发布的代码（开发与测试按实际确定是否执行）； 4. feature分支打标签； 5. 删除feature分支； 6. 删除feature分支标签。 |
| 开发人员 | 1. 在feature分支做开发； 2. 修复bug等 |
| 配置管理员 | 1. master分支打标签； 2. 代码分支及标签正确性检查。 |

# 标识规范

### 5.1 版本号标识

**产品线版本号：【产品线代号\_Vx.y】**

**产品版本号：****【产品线代号\_产品代号\_Vx.y.z.n】**

说明：

x/y/z/n均为阿拉伯数字，主版本号从1开始，其他版本号以0为起始数字，均每次增加1。

| **序号** | **项目** | **说明** | **是否必填** |
| --- | --- | --- | --- |
| 1 | 产品线代号 | 产品线的名称代号（参见上文代码库命名规则中定义） | 必填，App或客户端可选填 |
| 2 | 产品代号 | 产品线包含的各产品或应用的名称代号，由开发经理定义，需使用英文简称，并确保统一。 | 必填，App或客户端可选填 |
| 3 | V | Version，版本 | 必填，App或客户端可选填 |
| 4 | x | 主版本号：产品整体发生全局变化时，加 1 | 必填 |
| 5 | y | 子版本号：每个大迭代（功能模块发生大变更等）时，加1 | 必填 |
| 6 | z | 修正版本号：每个小迭代时, 加1 | 必填 |
| 7 | n | 构建版本号：build版本号，每build一次加1，持续递增 | 选填 |

举例：

工具线的某版本号，Tools\_Dict\_V1.0.2.35

备注：

* 产品线版本号一般在产品线属于一整套解决方案时可用。
* 产品的模块没有单独的版本号，包含在对应的产品版本号。
* 进入开发阶段后，开发经理提供版本计划，需经过项目经理/测试确认。版本计划需提供计划的版本号、对应的需求编号或功能名称、预计转测试时间、计划生产发布时间。版本计划需定期更新，测试经理依据版本计划，定期将版本号维护到jira系统。新增需求、bug，或提测需选择对应的版本号。

### 5.2 基线标识

配置管理员根据配置管理计划，对源代码及其基线进行分阶段管理。源代码建立基线时使用对应的标签标识。标签中的版本号、模块标识号为必选项，其余为可选项。

#### 5.2.1 缩写定义

|  |  |  |
| --- | --- | --- |
| **缩写** | **英文对照** | **中文对照** |
| TInit | Tag-for-Initialization | 初始标签 |
| TMT | Tag-for-Module-Testing | 模块测试标签 |
| TMMT | Tag- for-Merge-Module-Testing | 合并测试标签 |
| TP | Tag-for- Production | 生产标签 |
| THP | Tag-for-Hotfix-of-Production | hotfix生产标签 |

#### 5.2.2 基线标识规范

| **基线类型** | **对应标签标识** | **标签类型** | **说明** |
| --- | --- | --- | --- |
| 初始基线 | TInit-打标日期 | 初始代码标签 | 创建代码库后，如有初始源代码，导入代码库后需打初始代码标签。 |
| 测试阶段基线 | TMT-模块标识号-打标日期 | 模块测试标签 | 模块标识号指【模块对应的feature分支编号-构建次数】，是指feature分支模块测试时对应的模块标识号，仅供feature分支使用。 |
| TMMT**-**版本号**-**打标日期 | 合并测试标签 | 版本号是指develop分支合并测试时对应的版本号，仅供develop分支使用。 |
| 生产基线 | TP-版本号-上线日期 | 标准生产标签 | 当每次生产发布完成后，将对应release分支的代码合并到master和develop分支，对master分支打生产标签。  版本号是指生产发布用的版本的版本号，仅供master分支使用。 |
| THP-版本号-上线日期 | hotfix生产标签 | 当每次hotfix生产发布完成后，将对应hotfix分支的代码合并到master和develop分支，对master分支打hotfix生产标签。  版本号是指生产发布用的版本的版本号，仅供master分支使用。 |

# 流程图

### 6.1 流程说明

| **序号** | **流程说明** | **详细事项** | **角色** |
| --- | --- | --- | --- |
| 1 | 申请新增代码库 | 开发经理在OA申请创建对应代码库。 | 开发经理 |
| 2 | 创建代码库 | 配置管理员新建代码库，按要求设定代码库权限。 | 配置管理员 |
| 3 | 创建master、develop分支 | 建好库后，配置管理员创建master分支，从master分支创建develop分支；设定master、develop分支为protected branches。通知开发组使用。 | 配置管理员 |
| 4 | 创建feature分支  完成开发及自测 | 当有新需求待开发时，由开发经理或feature分支负责人从develop创建feature分支，通知开发人员在对应分支上开发。  当feature分支的开发活动完成后，开发人员自行验证通过。 | 开发经理或feature分支负责人  开发人员 |
| 5 | 模块测试 | feature分支负责人从master分支同步最新的已生产发布的代码（开发与测试按实际确定是否执行），然后给feature分支打模块测试标签并提测。测试组安排发版、测试并反馈结果： 如测试通过，feature分支负责人将即将准备生产发布的feature分支合并到develop分支；  如测试未通过，通知开发人员在对应的feature分支修改代码。修改完成后feature分支负责人再给feature分支打模块测试标签并提测。 | feature分支负责人  测试组 |
| 6 | 合并测试 | 当develop分支上的代码已经包含了即将生产发布的版本所计划包含的所有软件功能时，开发经理给develop分支打“合并测试标签”并提测。测试组发版、安排测试并反馈结果： 如测试通过，开发经理创建release分支；  如测试不通过，通知开发人员回到对应的feature分支修改代码，feature分支负责人修改完成后再次合并到develop分支。开发经理确认无误后，再给develop分支打合并测试标签并提测。 | 开发经理  测试组 |
| 7 | 创建release分支 | 合并测试通过后，开发经理创建release分支，并加入“protected branches”后，进行验证提测。测试组发版、安排测试并反馈结果： 如验证通过后，开发经理可以准备安排生产发布；  如验证未通过，开发经理删除release分支，通知开发人员回到对应的feature分支修改代码。修复完成后，feature分支负责人将代码合并到develop分支。开发经理确认无误后，再给develop分支打合并测试标签并提测。合并测试通过后，开发经理新建release分支并再次走验证流程。 | 开发经理  测试组 |
| 8 | 生产发布 | 验证测试通过后，开发组及运维安排生产发布，测试组进行线上验证。 | 开发组  运维组  测试组 |
| 9 | 源代码合并及打标签 | 当生产发布完成并线上验证通过后，开发经理需将该生产版本对应的源代码，从release分支合并到master和develop分支，配置管理员对master分支打生产标签。 | 开发经理  配置管理员 |
| 10 | 创建hotfix分支 | 当生产环境有异常或紧急缺陷的时候，开发经理从master分支上指定的“生产发布标签”派生hotfix分支来进行紧急修复工作。开发人员修复完成后，测试组安排紧急发版、测试，测试通过后，各组配合安排紧急生产发布。  生产发布完成并验证通过后，开发经理将hotfix分支的代码合并回master和develop分支，配置管理员给master分支打hotfix生产标签。 | 开发经理  测试组  运维组  配置管理员 |

备注：对于App端，首次生产发布后，合并对应的代码到master分支并打生产标签，后续可按实际需要继续使用该生产标签进行生产发布。

### 6.2 图示

上述流程说明的前3步请参阅上文配置库部分，不包含在图示中。

注意事项如下：

1. 如使用“git工作流”，点击“完成发布版本”操作，会将release分支的代码同时合并到master和develop分支（如release与develop分支代码完全一致，将不会合并任何修改到develop分支）。
2. 如在develop分支进行合并测试时，发现某需求或功能影响较大，无法一批上线，可由开发经理安排代码回退，重新计划此次的上线需求，重新合并代码及提测，此情况需尽量避免。
3. feature分支合入条件如下：
4. 开发活动完成，已实现必须的软件功能；
5. 有即将生产发布要求；
6. 同步过最新的已生产发布的代码（开发与测试按实际确定是否执行）；
7. 模块测试通过；
8. 分支负责人本地合并完成后验证通过。

分5种主要情况分别以流程图方式说明，从上至下按时间推进（具体流程图见下文）：

1. 图1：模块测试不通过情况；
2. 图2：合并测试不通过情况；
3. 图3：验证测试不通过情况；
4. 图4：生产环境有异常或紧急缺陷情况；
5. 图5：需要同步代码情况。

流程图中使用到的图例说明如下：

| **序号** | **图标** | **说明** |
| --- | --- | --- |
| 1 |  | 代码修改 |
| 2 |  | 模块测试提测点 |
| 3 |  | 合并测试提测点 |
| 4 | M | 合并代码 |
| 5 | b | 各类bug修复 |
| 6 | TP | 生产标签 |
| 7 | THP | hotfix生产标签 |
| 8 | TMMT | 合并测试标签 |
| 9 | TMT | 模块测标签 |
| 10 | Brh | hotfix分支 |
| 11 | brr | release分支 |
| 12 | brf | feature分支 |
| 13 | 数字1，2，…… | 代表第几个 |

**图1：模块测试不通过情况**



**图2：合并测试不通过情况**

备注：合并测试时，发现bug均需回对应的feature分支修改，如发现的bug无法确定是哪个feature分支导致，或是由代码合并后导致，可选择一个feature分支或从develop新建一个分支修改。

**图3：验证测试不通过情况**

备注：验证测试时，发现bug均需回对应的feature分支修改，如发现的bug无法确定是哪个feature分支导致，或是由代码合并后导致，可选择一个feature分支或从develop新建一个分支修改。

**图4：生产环境有异常或紧急缺陷情况**



**图5：需要同步代码情况**

