用 SVN 分支管理多版本

2010-02-22 梁军

1. 目的

为了在多个版本中并行开发，提高开发效率，保证各个版本和各个环境（开发、测试、主干）的独立，避免相互影响，减少最终发布时合并主干出现冲突的概率，降低冲突处理的难度，特编写该文档；

2. 原则

多个版本（开发版本，测试版本，发布版本)；多次合并。

3. Svn 目录结构

采用类似下面的目录结构：

```
project
  |
  +-- trunk
  +      |
  +      +----- main.js  (3.0版本的最新文件）
  +      +----- common.js
  +
  +-- branches
  +      |
  +      +-- r1.0
  +      +      |
  +      +      +---- main.js （1.x 版本的最新文件）
  +      +      +---- common.js
  +      +
  +      +-- r2.0
  +             |
  +             +---- main.js （2.x 版本的最新文件）
  +             +---- common.js
```

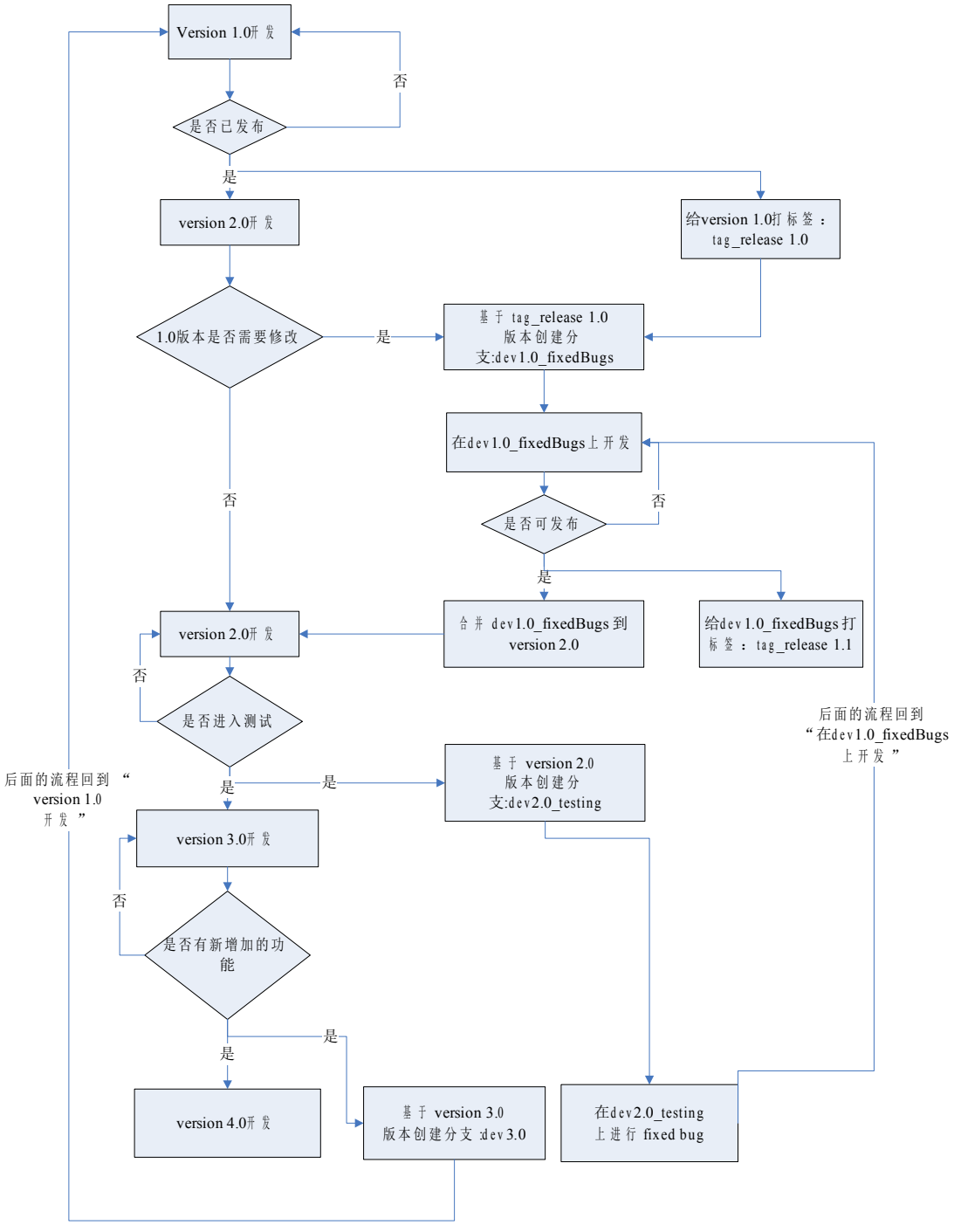
```
+
+-- tags   (此目录只读)
      |
      +-- r1.0
      +     |
      +     +---- main.js （1.0版本的发布文件）
      +     +---- common.js
      +
      +-- r1.1
      +     |
      +     +---- main.js （1.1版本的发布文件）
      +     +---- common.js
      +
      +-- r1.2
      +     |
      +     +---- main.js （1.2版本的发布文件）
      +     +---- common.js
      +
      +-- r1.3
      +     |
      +     +---- main.js （1.3版本的发布文件）
      +     +---- common.js
      +
      +-- r2.0
      +     |
      +     +---- main.js （2.0版本的发布文件）
      +     +---- common.js
      +
      +-- r2.1
            |
            +---- main.js （2.1版本的发布文件）
            +---- common.js
```

trunk：主干，是日常开发进行的地方。

branches：分支。一些阶段性的 release 版本，这些版本是可以继续进行开发和维护的，则放在 branches 目录中，里面的版本全部基于 trunk 基础上建立的。

tags：表示标签存放的目录，一般为只读写，存储阶段行发布版本，一般是基于分支上建立。

4. 流程



4.1. 选定一个非常稳定的版本作为一个基础版本，也就是主干版本。要求该版本必须是稳定版本。(假设这个版本是1.0)，则当前目录结构为：

```
project
|
+-- trunk
+       |
+       +----- main.js  (1.0版本的最新文件）
+       +----- common.js
+--branches
+--tags
```

4.2. 1.0版本开发完成并已发布，则直接在 tags 里面打个标签，并且新需求在主干上开发，此时主干版本变成2.0。则目录结果变成:

```
project
|
+-- trunk
+       |
+       +----- main.js  (2.0版本的最新文件）
+       +----- common.js
+--branches
+--tags
+       +-----tag_release 1.0 (从主干复制过来，该版本和外网一致)
```

4.3. 如果发现外网版本1.0,有 bug，或者有个新的需求，要基于在外网版本上开发，因为主干版本已和外网不一样，因此不能够在主干上直接开发，此时，在 tag_release 1.0 基础上建立一个分支，进行 fixed bug 或小功能需求开发， 则目录结果为：

```
project
|
+-- trunk
+       |
+       +----- main.js  (2.0版本的最新文件）
+       +----- common.js
+--branches
+       +dev_1.0_fixedBug (从 tag_release 1.0复制过来)
```

```
+--tags
+       +-----tag_release 1.0
```

4.4. 在 dev_1.0_fixedBug 版本上进行 fixed bug，或者在这个基础上进行部分小功能开发，在主干 truck 进行2.0开发；

4.5. dev_1.0_fixedBug 版本开发完毕，并正式上线，然后基于 dev_1.0_fixedBug 的基础上在 tags 里面打上标签 tag，此时目录结果为:

```
project
  |
  +-- trunk
  +       |
  +       +----- main.js  (2.0版本的最新文件）
  +       +----- common.js
  +--branches
  +       +dev_1.0_fixedBug
  +--tags
  +       +-----tag_release 1.0
  +       +-----tag_release 1.1 （从 dev_1.0_fixedBug 复制过来）
```

4.6. 根据需要选择性地把 dev_1.0_fixedBug 这个分支合并到主干 truck。

合并原则：低版本合并到高版本；换言之，低版本里面修改的 bug，一定要合并到高版本中。这里的合并可以根据具体来定，如果是 bug fixed，则一定要合并到主干 truck 中，但如果是小需求修改，并且不计划在后续版本中实现，则可以不合并相应代码。至于合并时间间隔问题，最长不能够超过一个礼拜，否则会引起冲突严重，加大合并难度。

4.7. 当主干上的2.0开发已经结束，进入测试阶段，此时可以规划下一个版本 3.0，则在当前主干 truck 上建立一个分支，目录结果为：

```
project
  |
  +-- trunk
  +       |
  +       +----- main.js  (3.0版本的最新文件）
  +       +----- common.js
```

```
+--branches
+       +dev_1.0_fixedBug
+       +dev_2.0_testing (从原来主干上2.0的版本基础上复制)
+--tags
+       +-----tag_release 1.0
+       +-----tag_release 1.1
```

4.8. 此时在主干上开发3.0，在 dev_2.0_testing 上 fixed bugs。后面步骤类似4.4--4.7的步骤；

4.9. 如果在2.0 fixed bugs 和3.0版本同时进行阶段，想规划下一个版本4.0，则可以考虑在3.0基础上再做一个分支，为了减少在4.0版本上的合并，要求4.0版本的需求是属于新增功能，不能够对原来的文件有过多的修改，否则会引起代码冲突严重。

```
project
  |
  +-- trunk
  +       |
  +       +----- main.js
  +       +----- common.js
  +       +-----dialog.js (因为新增功能而增加的文件4.0)
  +--branches
  +       +dev_1.0_fixedBug
  +       +dev_2.0_testing
  +       +dev_3.0 (从原来主干上3.0的版本基础上复制)

  +--tags
 +       +-----tag_release 1.0
 +       +-----tag_release 1.1
```

4.10. 2.0 测试，3.0开发、4.0开发多个版本同时进行。

5. 优点和不足

优点主要有：

1、多个版本相互独立，互不影响

2、通过分支与主干的合并，这样主干永远是最新、最高版本，并且都在后面的测试中，保证了质量。

不足：

当版本比较多时候，低版本上的 bug 修改，需要合并到高版本，版本越多，合并的次数就越多。

6. 分支合并：

6.1. 使用 BCompare 手工合并；

6.2. 使用 svn 合并功能合并

7. 建议

7.1. 如果项目的周期比较长，和主干进行合并的次数也应该加大，以降低处理冲突的难度。

7.2. 要求开发人员养成增加注释(log)习惯，方便后期对代码修改的跟踪。建议 svn 配置成强制添加注释，方可以提交代码。注释主要有下面类型：

7.2.1.新增功能：用功能需求作为注释，可以概括为简单的一句话。

7.2.2.fixed bug: 用 bug 的标题、url 作为注释；

7.2.3. 合并：必须要描述清楚合并来源。eg: merge from

dev_1.0_fixedBug branch.

7.3. 每次发布后，必须要把对应的版本进行打标签，也就是在 tags 创建对应的 tag.

7.4. 在接受到新的需求或者 bug fixed 时候，要先确定该功能需求，应该在那个版本上开发，是否需要建立新的分支。

7.5. 为了降低合并的冲突，在低版本修复 bug 时，开发员修复一个 bug，就合并到对应的高版本，以减少后期合并的冲突发生。