南京大学自然语言处理组编程规范

南京大学自然语言处理组

version 2.0

2012 年 11 月 21 号

目　　录

前　　言 1
1 命名规范 2
1.1 通用命名规范 2
1.1.1 基本规则 2
1.1.2 命名的选择 2
1.1.3 命名的长度 2
1.1.4 慎用缩写 3
1.2 语言特有的命名 3
1.2.1 Java 3
1.2.2 C# 3
2 注释规范 5
2.1 实现注释规范 5
2.1.1 注释内容 5
2.1.2 注释格式 5
2.1.2.1 块注释 5
2.1.2.2 短注释 5
2.1.2.3 行末注释 6
2.1.2.4 TODO 注释 7
2.2 文件注释规范 7
2.2.1 注释内容 7
2.2.2 注释格式 8
2.3 文档注释规范 8
2.3.1 注释内容 8
2.3.1.1 类注释 8
2.3.1.2 方法注释 9
2.3.2 注释格式 9
2.3.2.1 Java 文档注释 10
2.3.2.2 C# 文档注释 10

3 项目规范 11
3.1 项目管理 11
3.1.1 项目管理平台 11
3.1.2 目录和文件 11
3.1.3 编码 11
3.1.4 数据 11
3.2 代码管理 11
3.2.1 版本控制 11
3.2.2 代码测试 12
3.2.3 代码审查 12
3.3 文档管理 12
3.3.1 设计文档 12
3.3.2 使用文档 12
3.3.3 更新记录 12
3.3.4 Readme 13
3.3.5 版权信息 13
4 版权协议 14
5 附录 15
5.1 Java 文档注释 15
5.1.1 部分标签示例 15
5.1.2 程序示例 17
5.1.3 HTML 文档 17
5.2 C# 文档注释 18
5.2.1 部分标记示例 18
5.2.2 程序示例 28
5.2.3 xml 文档 29
5.3 redmine 使用介绍 29
5.3.1 新建项目 29
5.3.2 讨论区使用 30
5.3.3 wiki 使用 31
5.3.4 版本库使用 31
5.3.5 文件使用 31

5.3.6 代码评审 32
5.4 git 使用介绍 32
5.4.1 使用命令介绍 32
5.4.2 git 使用配置 35
5.4.3 git 使用操作 36
5.4.3.1 本地操作 36
5.4.3.2 远程操作 36
5.4.3.3 从服务器上克隆项目 38
致 谢 39
参考文献 40

插 图 目 录

2.1 文件注释 8

5.1 java 文件注释 17
5.2 html 文件一 17
5.3 html 文件二 18
5.4 CSharp 文件注释一 28
5.5 CSharp 文件注释二 28
5.6 xml 示例 -1 29
5.7 xml 示例 -2 29
5.8 xml 文件 29
5.9 redmine-project 示例 29
5.10 redmine-forum 示例 1 30
5.11 redmine-forum 示例 2 30
5.12 redmine-wiki 示例 31
5.13 redmine-file 示例 31
5.14 redmine-codereview 示例 32
5.15 redmine-git 示例 38

前　言

南京大学自然语言处理组（NJUNLP）在数十年的发展过程中积累了大量的与科研相关的代码、程序和工具，包含着每一届同学和每一位老师的辛勤工作的成果。但是，由于受到个人工作风格和人员流动等因素的影响，这些工作的传承和重用逐渐遇到了困难。为了更好的对这些工作进行管理和维护，我们制定了这一份编程规范。希望能够通过规范化的手段控制代码生成、文档管理、项目管理等过程，以及包括版权协议在内的诸多其他方面。

从开始编制到完成初稿历时超过一个月的时间，我们参考了大量的相关规范，针对组内的实际情况进行了组合、删减和优化，并给出了相应的示例；力求让这份规范能够符合 NLP 组的实际情况，并尽量减少实际执行过程中的负担。尽管删改多次，其中难免会有一些疏漏。如有任何不清楚、不恰当之处，请各位老师同学随时指出。我们一同管理和维护这份规范，为 NLP 组的发展尽一份力量。请各位老师同学仔细阅读这份规范，并在工作中认真遵守规范中的相关约定。愿我们的工作不断进步。

本文档给出了南京大学自然语言处理研究组的开发过程中的需要遵守的命名规范、注释规范、项目管理规范以及组内项目所遵守的版权协议（包括但不限于开源协议）。本文的示例程序中默认使用的是 GPL 协议。附录中详细说明了 Java 和 C# 中对文档注释的语言支持。

请严格执行本文档中的约定（少量可选的约定在文档中标记为 optional）

黄书剑

2012 年 10 月 7 日

1 命名规范

从建立项目到书写每一行代码，无时无刻不需要进行各种命名。命名规范给出了各类命名任务的基本命名规则，其目的是使程序具有更好的可读性，从而更易于理解、重用和维护。下面首先给出不受限于语言的通用命名规范，再针对部分语言中的特殊情况(如 Java 中的包，C# 中的名空间等）给出相应的规范。

1.1 通用命名规范

1.1.1 基本规则

常用的命名单位包括类、接口、变量、常量、方法等。命名应采用大小写混合的方式，类名和接口名首字母大写，变量名和方法名首字母小写；命名由多个单词构成时，后续单词首字母大写。变量名不应该以下划线或 $ 开头。常量的命名采用全部字母大写，单词间用下划线隔开。

示例如下：

(1)	类名	Raster	ImageSprite
(2)	接口名	Storing	RasterDelegate
(3)	方法名	getMax()	runFast()
(4)	变量名	myWidth	fruitShape
(5)	常量名	MIN_WIDTH	MAX_WIDTH

1.1.2 命名的选择

命名应在不使用注释的情况下也能较好的体现出作者的意图，应该告诉代码阅读人员，它们为什么会存在，做什么事情。类、接口、变量、常量的命名应使用名词，方法的命名应使用动词。名称必须是可读的，这样会方便讨论。

示例如下：

(1)	消逝的时间	elapsedTime
(2)	生成时间戳	generateTimeStamp

1.1.3 命名的长度

避免使用单字命名。在我们阅读或者修改代码的过程中，有时候需要查找变量名或者常量名，然而单字母名称和数字常量很难在一个代码文件中找到，这也造成了相当程度的不便。

若变量或常量可能在代码中多处用到，则应赋予其便于搜索的名称，建议命名长度

与作用域大小相对应, 即作用域越大命名越长。同时，避免命名超长 (原则上不应该超过 20 个字符)。

示例如下:

(1) 变量名　　realDaysPerYear

(2) 常量名　　WORK_DAYS_PER_WEEK

1.1.4 慎用缩写

请尽量少使用意义不明确的缩写，如果要用到缩写，尽量按照大家公认的缩写名称来使用，并在使用时用注释进行说明。

示例如下:

(1) 缩写规则　　NO. 代表 number

(2) 缩写规则　　ID. 代表 identification

1.2 语言特有的命名

1.2.1 Java

(1) 包

包的命名应全部使用小写的 ASCII 字母。由于互联网上的域名是不会重复的，所以一般采用在互联网上的域名作为包的惟一前缀。NLP 组的包命名前缀规范是 edu.nju.nlp.* 。对应的代码目录结构是 edu->nju->nlp->*。包名的后续部分根据不同机构各自内部的命名规范而不尽相同，NLP 组的包后缀部分命名应同样遵守有意义、可理解的原则，具体要求可参照 1.1 节相关内容。

示例如下:

(a) com.sun.eng

(b) edu.nju.nlp.keyword

1.2.2 C#

(1) 名空间

名空间命名包括：CompanyName.ProjectName.Feature。其中，CompanyName：公司名称；Project Name：该项目缩写；Feature：实现的功能。每个单词的首字母需要大写。需要注意的是名空间和类不能使用同样的名字。NLP 组的名空间中 CompanyName 为 NJUNLP。项目名和特征名的命名应同样遵守有意义、可理解的原则，具体要求可参照 1.1 节相关内容。

示例如下：

(a) Microsoft.office.ui

(b) NJUNLP.Keyword.Extrator

2 注释规范

类、方法、变量等命名意义明确的代码本身为理解代码打下了良好的基础，但实际工作中仍需要在代码中加入额外的解释和说明成分，这些成分应采用注释来完成。适当的注释对保证代码可读性、项目可持续发展至关重要。代码中的注释分为实现注释、文档注释和文件注释三种类型。其中实现注释用于直接解释代码内容，文档注释用于具体说明代码的结构、文件注释用于说明版权，作者等信息。注释规范中约定了需要进行注释的内容和注释使用的格式。注释中不应包括诸如制表符和回退符之类的特殊字符。

2.1 实现注释规范

2.1.1 注释内容

程序关键逻辑，特别对于实现代码中重要的地方，应采用实现注释进行说明

2.1.2 注释格式

实现注释的格式在不同语言中有所不同。例如：在 perl、python 等脚本语言中采用 # 标记；在 C++，Java 和 C# 等语言中可以使用//或/* */两种格式。下面详细说明 C++、Java 和 C# 中的实现注释风格。脚本语言中的注释可参照短注释和行末注释。

C++、Java 和 C# 中的实现注释风格包括：块注释、短注释，行末注释，TODO 注释。

2.1.2.1 块注释

块注释之首应该有一个空行，用于把块注释和代码分割开来. 块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

示例如下：

```
/*
  Here is a block comment.
*/
```

2.1.2.2 短注释

短注释分为单行注释、尾端注释。它们可以在一行内对需要解释的代码给出注释。单行注释之前有一个空行并且需要与其后的代码具有一样的缩进层级。尾端注释需要与所要描述的代码位于同一行。

(1) 单行注释, 示例如下:

```
if (condition) {
    /* Handle the condition. */
                ...
}
```

(2) 尾端注释, 示例如下:

```
if (a == 2) {
    return TRUE;                /* special case */
}
else {
    return isPrime(a);          /* works only for odd a */
}
```

2.1.2.3 行末注释

注释界定符"//", 可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本; 然而, 它可以用来注释掉连续多行的代码段。

示例如下:

```
if (foo > 1) {
    // Do a double-flip.
        ...
}
else {
    return false;           // Explain why here.
}
//if (bar > 1) {
        //
        //     // Do a triple-flip.
        //     ...
        //}
        //else {
        //     return false;
        //}
```

2.1.2.4 TODO 注释

对于那些临时的，短期的解决方案或者不够完美的代码使用 TODO 注释。TODO 注释格式：[TODO :]

示例如下：

```
Public SomeClass(){
        //TODO:Add Constructor Logic here
}
```

2.2 文件注释规范

2.2.1 注释内容

在每个文件的开头应加入文件注释，用于说明版权、许可版本、作者、版本等信息。具体内容如下：

版权：如 Copyright © 2012 Nanjing university

许可版本：项目的使用许可信息，如 GPL，LGPL，BSD，Apache2.0；NLP 组的使用许可信息参见本文档第 4 部分

作者：文件的原始作者信息以及联系方式

版本：初始版本信息

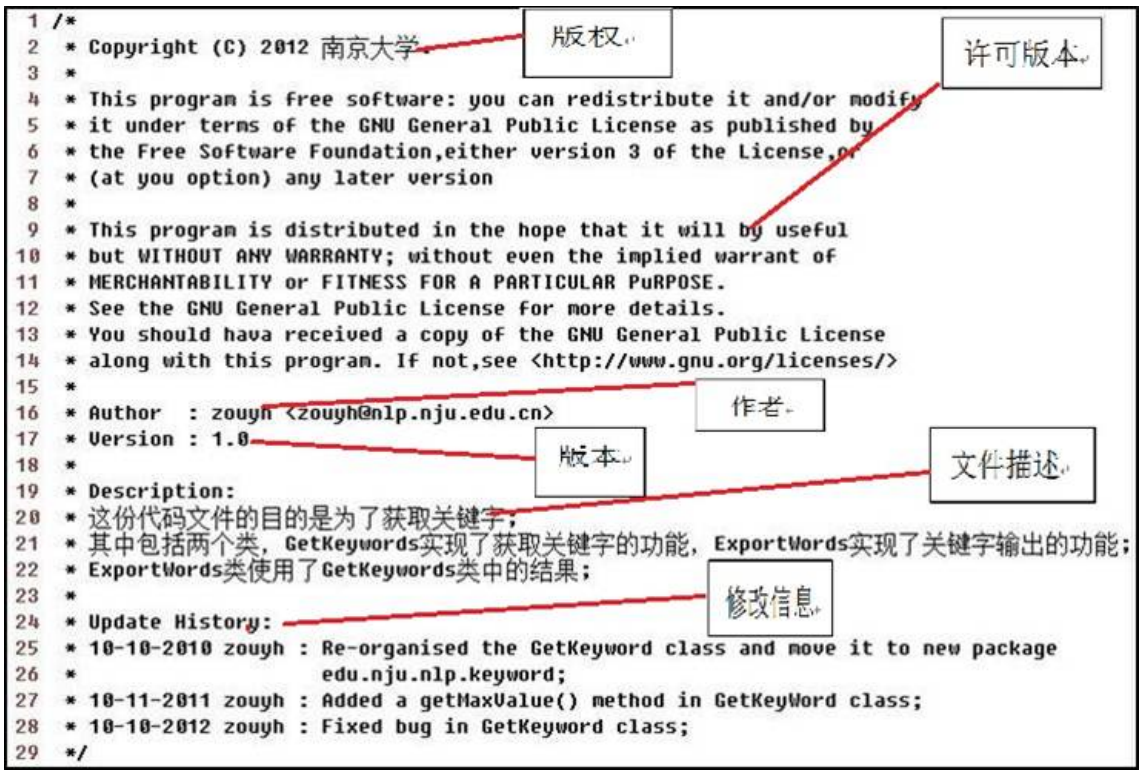
文件描述：简要介绍文件的内容，文件中类的功能等，如果文件中包含多个类，还应描述这些类的相互关系

修改信息：记录文件修改的信息，包括修改日期，修改人，修改内容等

2.2.2 注释格式

这里 java，c，c++，c# 等语言以/*开头, 以*/结尾。具体格式见下面。

至于 python，perl，ruby 等语言的文件注释只要包含上述文件注释内容，符合语言语法即可。示例如下:




```
/*
 * Copyright (C) 2012 南京大学
 *
 * This program is free software: you can redistribute it and/or modify
 * it under terms of the GNU General Public License as published by
 * the Free Software Foundation,either version 3 of the License,or
 * (at you option) any later version
 *
 * This program is distributed in the hope that it will by useful
 * but WITHOUT ANY WARRANTY; without even the implied warrant of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PuRPOSE.
 * See the GNU General Public License for more details.
 * You should hava received a copy of the GNU General Public License
 * along with this program. If not,see <http://www.gnu.org/licenses/>
 *
 * Author  : zouyh <zouyh@nlp.nju.edu.cn>
 * Version : 1.0
 *
 * Description:
 * 这份代码文件的目的是为了获取关键字;
 * 其中包括两个类，GetKeywords实现了获取关键字的功能，ExportWords实现了关键字输出的功能;
 * ExportWords类使用了GetKeywords类中的结果;
 *
 * Update History:
 * 10-10-2010 zouyh : Re-organised the GetKeyword class and move it to new package
 *                    edu.nju.nlp.keyword;
 * 10-11-2011 zouyh : Added a getMaxValue() method in GetKeyWord class;
 * 10-10-2012 zouyh : Fixed bug in GetKeyword class;
 */
```

图 2.1　文件注释

2.3　文档注释规范

文档注释用于标明与文件的相关版权作者等信息和对代码的结构和内容进行系统性的解释和说明。文档注释规范约定了 NLP 组的文档注释应该包含的内容，并对文档注释应该使用的格式进行了说明。

2.3.1 注释内容

下面从类、方法两个层次分别说明文档注释的内容。

2.3.1.1 类注释

每个类的定义之前应该加入类注释，用于对该类进行简单的介绍，并描述关于类的功能和用法。如果该类有与其他类进行交互的重要功能函数或接口函数，也应该在类注释中进行简要说明。

示例如下:

```
/** Description of MyClass
  * The function of MyClass is Blah Blah
  * The method of MyClass is Balh Blah
  */
public class Myclass {
}
```

2.3.1.2 方法注释

在关键方法（例如实现主要功能的方法、被多个类调用的方法、可能被重用的方法等）的定义之前应该加入方法注释.

用于描述以下内容:

(1) 方法的功能和实现，包括该方法的主要功能，方法中使用的关键算法等信息。如有需要，还应该说明该方法可能产生的异常等;

(2) 参数描述，包括需要传入的参数的意义、要求等

(3) 返回值描述，返回值的意义、可能发生的异常情况等

示例如下:

```
/** Description of myMethod(int a,double b)
  * The function of myMethod is Blah Blah
  * @param  a   Description of a
  * @param  b   Description of b
  * @return c   Description of c
  */
public object myMethod(int a ,double b){
    object c;
    // Blah Blah Blah...
    return c;
}
```

2.3.2 注释格式

文档注释应使用对应程序设计语言所提供的注释机制进行。例如，perl 和 python 中使用 # 符号，C++ 中使用//或/* */等。

特别的，Java 和 C# 中对文档注释进行了特殊的支持。在 Java 和 C# 中，如果采用给定的格式进行文档注释，可以利用辅助工具将这些注释直接转化为 html 或者 xml 表

示的文档，从而更加直观的得到对于代码信息的描述。因此，我们规定在 Java 和 C# 的程序中，应该使用其内建的格式进行文档注释。下面简要介绍这两种语言中内建的文档注释的支持。

2.3.2.1 Java 文档注释

Java 中的文档注释内容由/**...*/界定。文档注释中可以通过插入标签来描述指定内容。标签使用如下形式描述：

```
@label description
```

其中 label 指的是具体的标签名，而 description 是对该标签的具体描述。Java 中支持的标签包括 version，author 等 (详细描述和示例见附录 5.1)。文档注释可以通过 Javadoc 工具转换成 HTML 文件。

2.3.2.2 C# 文档注释

C# 中的文档注释内容由///界定。C# 提供的机制是可以使用含有 XML 文本的特殊注释语法为代码编写文档。XML 标记可以使用如下形式描述：

```
<label>description </label>
```

其中 label 指的是具体标记名称，description 是对该标记的具体描述。C# 中支持的标签包括 version，author 等 (详细描述和示例见附录 5.2)。

C# 的文档注释可用 Microsoft Visual Studio 生成对应的 XML 文件。具体方法如下所示：1) 在“解决方案资源管理器”中选定一个项目，然后在“项目”菜单中单击“属性”;2) 单击“生成”选项卡;3) 在“生成”页上，选择“XML 文档文件”。默认情况下，在指定的输出路径（如“bin\\Debug\\Projectname.XML”）下创建该文件。

3 项目规范

项目规范包含项目管理、代码管理、文档管理等多个方面内容，我们使用 redmine 来管理自己的项目，使用 git 来管理自己的代码。

3.1 项目管理

3.1.1 项目管理平台

NLP 组使用基于开源工具 Redmine 的项目管理平台，可以用于管理代码、文档并提供实用信息反馈、版本更新，并提供讨论区，代码评审等功能。组内同学定期必须往 redmine 上传代码，文档等。nlp 关于 redmine 的使用请参考附录 5.3。

3.1.2 目录和文件

代码和文档是项目的主要部分，因此代码和文档在项目中应该单独存放和管理。建议在每个项目中都包含 src 和 doc 两个目录，对应存放该项目的代码和文档文件。此外，部分项目还应该包含 lib 文件夹用于存放该项目所使用的工具包和程序库，data 文件夹用以存放该项目相关的数据。需要特别注意的是，编译结果或可执行文件应单独存放，如 Java 中应选择用 bin 文件夹单独存放编译得到的二进制文件等

3.1.3 编码

需要注意的是，由于 NLP 组的工作涉及到大量的多种语言相互转换，文件编码往往成为许多 bug 的罪魁祸首。因此，我们要求在所有文件（包括源文件和数据文件）的管理中，默认使用 utf-8 编码。如有特殊情况需要使用其他编码的，应予以显式说明。

3.1.4 数据

项目的数据使用应该清晰明确，大规模的数据应按照版本单独存储，在项目中只需要注明使用数据的版本即可，尽量少使用重复的数据副本。在项目保存和提交过程中应随项目保留小规模的测试数据，用以验证程序的正确性。

3.2 代码管理

3.2.1 版本控制

NLP 组统一使用开源的 git 工具进行代码的版本控制。关于 git 的使用请参考附录 5.4

3.2.2 代码测试

任何一个项目都需要进行充分的测试，这包括最基本的项目能否正确的运行，到速度、效率上面的追求。

测试代码的最好方法是给代码写测试。写出来的测试可以反复的执行。当你修改了实现，可以通过再次运行测试来检查是否引入了任何 bug。这种方法可以将你从调试中拯救出来，并且可以帮助你设计更好的代码。

NLP 组的代码应该都经过较为完善的测试过程以保证正确性。

3.2.3 代码审查

代码审查即 code review。代码审查（code review）可以检验代码设计的合理性（如实现方法，数据结构，设计模式，扩展性考虑等)，是否存在大量重复代码等等。在代码审查中，参与审查的人员可以充分了解代码的设计和实现，方便后期的维护，也减少了项目风险。NLP 组的重要项目代码应尽量保证进行过代码审查。

3.3 文档管理

软件开发过程中，文档有着非常重要的作用。通过文档代码阅读者能够迅速的理解程序设计思想，从而为程序改进或者二次开发帮助极大。这不仅节省时间同时也有利于该项目的可持续发展。完整的项目中必须包含设计文档，使用文档，更新记录、Readme 以及版权信息。项目中的文档应包含下面几个方面的内容:

3.3.1 设计文档

(1) 设计目的

描述项目实现的功能以及包含的主要代码文件、每个代码文件之间的关系;

(2) 设计思路

描述项目中使用的主要算法；输入输出接口说明, 以及在项目开发过程中使用到的参考文献;

3.3.2 使用文档

使用文档记录介绍了项目的使用方法。

3.3.3 更新记录

里面记录了项目升级时的性能提升、新增的功能或者修正了上一个版本出现 bug 等等。格式：[version][date]

示例如下:

```
1.05        10-10-2010
* 修改了基本数据结构，优化了速度
* 增加了中文识别的功能
* 修改了版本出现的1.0bug
```

3.3.4 Readme

列出了项目开发过程中做出贡献的作者以及联系方式。

3.3.5 版权信息

版权信息的使用方法参见本文档的第 4 部分。

4 版权协议

版权协议目前先使用闭源协议，请在每一个头文件中添加下面的协议：

```
/*
* Copyright (c) Nanjing University. All rights reserved.
* The code is owned by Natural Language Processing Group, Nanjing University.
* If other people want to use the code, please contact Professor Chen (chenjj@nju.edu.cn).
*/
```

5 附录

5.1 Java 文档注释

5.1.1 部分标签示例

(1) @version

该标签的格式如下:

```
@version version-information
```

其中,”version-information”可以是任何你认为适合包含在版本说明中的重要信息。如果 Javadoc 命令使用了”-version”标记, 那么就可以从生成的 HTML 文档中提取出版本信息。

(2) @author

该标签的格式如下:

```
@author author-information
```

其中,”author-information”可以包括电子邮件地址或者其他任何适宜的信息。如果 Javadoc 命令行使用了”-author”标记，那么就可以从生成的 HTML 文档中提取出作者信息。可以使用多个标签，以便列出所有的作者，但是它们必须连续放置。全部作者信息会合并到同一段落，置于生成的 HTML 中。

(3) @throws

该标签的格式如下:

```
@throws fully-qualified-class-name description
```

“异常”它们是由于某个方法调用失败而抛出的对象。尽管在调用一个方法时，只出现一个异常对象，但是某个特殊方法可能会出产生任意多个不同类型的异常，所有这些异常都需要进行说明。其中 fully-qualified-class-name 给出一个异常类的无歧义的名字，而该异常类常在别处定义。description 告诉你为什么此特殊类型的异常会在方法调用中出现

(4) @param

该标签的格式如下:

```
@param parameter-name description
```

其中，”parameter-name”是方法的参数列表中的标识符，description 是可延续数行的文本，终止于新的文档标签出现之前。可以使用任意多个这种标签，每个参数都有一个这样的标签。

(5) @return

该标签的格式如下：

```
@return descrption
```

其中，description 用来描述返回值的含义，可以延续数行。

(6) @see

该标签的格式如下：

```
@see class name
@see fully-qualified-classname
@see fully-qualified-classname\#method-name
```

@see 标签允许用户引用其它类的文档。Javadoc 会在其生成的 HTML 文件中，通过 @see 标签链接到其它文档。

(7) @since(optional)

该标签允许你指定程序代码最早使用的版本，可以在 HTML Java 文档中看到它被用来指定所用的 JDK 版本的情况

(8) @deprecated(optional)

该标签用于指出一些旧特性已由改进的新特性所取代，建议用户不要再使用这些旧特性，因为在不久的将来它们很可能会被删除。如果使用一个标记为 @deprecated 的方法，则会引起编译器的警告

若想了解更多有关文档注释和 Javadoc 的详细资料，参见 Javadoc 的主页：http://www.oracle.com/technetwork/Java/Javase/documentation/index-jsp-135444.htm

5.1.2 程序示例

这里我们假设使用开源协议中的 GPL 协议，遵守命名规范和文档注释要求。

示例如下：

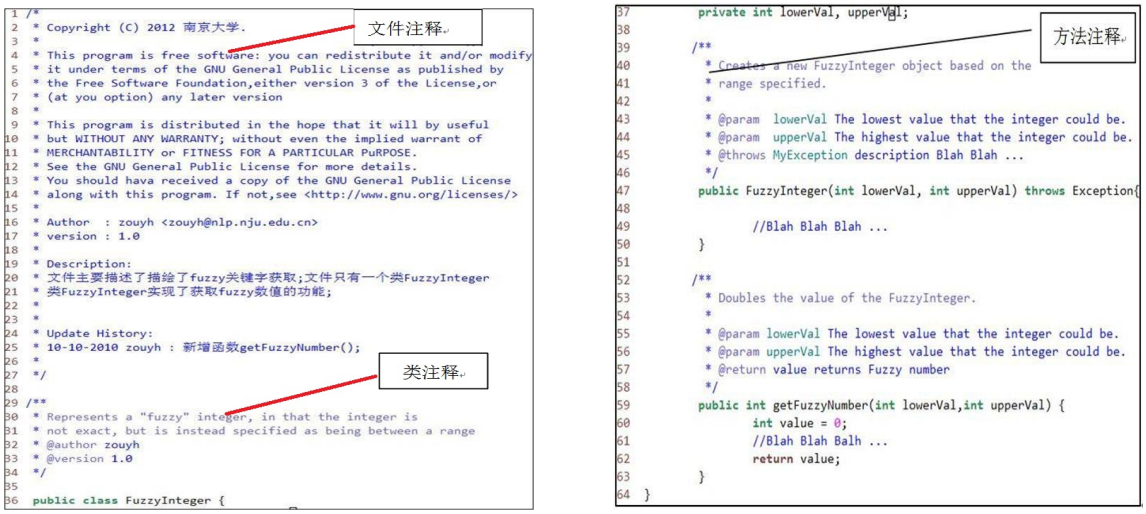


图 5.1 java 文件注释

5.1.3 HTML 文档

根据 5.1.2Java 文档注释生成的 HTML 文档

示例如下：

图 5.2 html 文件一

图 5.3　html 文件二

5.2　C# 文档注释

5.2.1　部分标记示例

(1) <summary>

该标记格式如下:

```
<summary>description </summary>
```

参数 description: 对象的摘要。<summary> 标记应当用于描述类型或类型成员。使用 <remarks> 添加针对某个类型说明的补充信息。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
public class TestClass{
    /// <summary>doWork is a method in the TestClass class.
    /// <para>Here's how you could make a second paragraph
    ///  in a description. <see cref="System.Console.WriteLie
    ///  (System.String)"/>
    ///  for information about output statements.</para>
    /// <seealso cref="TestClass.Main"/>
    /// </summary>
    public static void doWork(int number){
    }
    /// text for Main
    static void Main(){
    }
```

```
}
```

(2) <remarks>

该标记的格式如下:

```
<remarks>description </remarks>
```

参数 description： 成员的说明。<remarks> 标记用于添加有关某个类型的信息，从而补充由 <summary> 所指定的信息。此信息显示在对象浏览器中。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// <summary>
/// You may have some primary information about this class.
/// </summary>
/// <remarks>
/// You may have some additional information about this class.
/// </remarks>
public class TestClass{
/// text for Main
    static void Main(){
    }
}
```

(3) <param>

该标记的格式如下:

```
<param name='name'>description </param>
```

参数 name： 方法参数名。将此名称用双引号括起来 (" ")。参数 Description: 参数说明。<param> 标记应当用于方法声明的注释中，以描述方法的一个参数。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
public class TestClass{
/// <param name="int1">Used to indicate status.</param>
    public static void doWork(int int1){
```

```
    }
/// text for Main
    static void Main(){
    }
}
```

(4) <typeparamref>

该标记的格式如下:

```
<typeparamref name="name">description </typeparamref>
```

参数 name 类型参数的名称。将此名称用双引号括起来 (" ")。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// comment for class
public class TestClass{
    /// <summary>
    /// Creates a new array of arbitrary type <typeparamref
    ///name="T"/>
    /// </summary>
    /// <typeparam name="T">The element type of the array
    ///</typeparam>
    public static T[] mkArray<T>(int n){
        return new T[n];
    }
}
```

(5) <exception>

该标记的格式如下:

```
<exception cref="member">description </exception>
```

参数 cref = "member": 对可从当前编译环境中获取的异常的引用。编译器检查到给定异常存在后，将 member 转换为输出 XML 中的规范化元素名。必须将 member 括在双引号 (" ") 中。<exception> 标记使您可以指定哪些异常可被引发。此标记可用在方法、属性、事件和索引器的定义中。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// comment for class
public class EClass : System.Exception{
// class definition...
}
/// comment for class
class TestClass{
    /// <exception cref="System.Exception">Thrown when...
    /// </exception>
    public void doSomething(){
        try{
        }
    catch (EClass){
    }
}
}
```

(6) <returns>

该标记的格式如下：

```
<returns>description</returns>
```

参数 description：返回值的说明。<returns> 标记应当用于方法声明的注释，以描述返回值。

示例如下：

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
public class TestClass{
/// <returns>Returns zero.</returns>
    public static int getZero(){
        return 0;
    }
    /// text for Main
    static void Main(){
    }
}
```

(7) <value>

该标记的格式如下:

```
<value>property-description </value>
```

参数 property-description: 属性的说明。<value> 标记可以描述属性所代表的值。请注意，当在 Visual Studio .NET 开发环境中通过代码向导添加属性时，它将会为新属性添加 <summary> 标记。然后，应该手动添加 <value> 标记以描述该属性所表示的值。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class Employee
public class Employee{
    private string aName;
    /// <summary>The Name property represents the
    /// employee's name.</summary>
    /// <value>The Name property gets/sets the aName data
    ///  member.</value>
    public string Name{
        get{
                return aName;
            }
        set{
                aName = value;
            }
        }
}
/// text for class MainClass
public class MainClass{
/// text for Main
    static void Main(){
    }
}
```

(8) <c>(optional)

该标记的格式如下:

```
<c>text</c>
```

参数 text: 希望将其指示为代码的文本。<c> 标记提供了一种将说明中的文本标记为代码的方法。使用 <code> 将多行指示为代码。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
public class TestClass{
/// <summary><c>doWork</c> is a method in the
/// <c>TestClass</c> class.
    /// </summary>
    public static void doWork(int int1){
    }
    /// text for Main
    static void Main(){
    }
}
```

(9) <code>(optional)

该标记的格式如下:

```
<code>content</code> (optional)
```

参数 content: 希望将其标记为代码的文本。<code> 标记提供了一种将多行指示为代码的方法。

(10) <example>(optional)

该标记的格式如下:

```
<example>description</example>
```

参数 description: 代码示例的说明。使用 <example> 标记可以指定使用方法或其他库成员的示例。这通常涉及使用 <code> 标记。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
```

```
public class TestClass{
/// <summary>
/// The getZero method.
/// </summary>
/// <example> This sample shows how to call the
///  getZero method.
/// <code>
/// class TestClass
/// {
///     static int Main()
///     {
///         return getZero();
///     }
/// }
/// </code>
/// </example>
    public static int getZero(){
        return 0;
    }
}
```

(11) <include>(optional)

该标记的格式如下:

```
<include file='filename' path='tagpath[@name="id"]' />
```

参数 filename: 包含文档的文件名。该文件名可用路径加以限定。将 filename 括在单引号 (' ') 中。tagpath: filename 中指向标记 name 的标记路径。将此路径括在单引号中 (' ')。name: 注释前边的标记中的名称说明符; name 具有一个 id。id: 位于注释之前的标记的 ID。将此 ID 括在双引号中 (" ")。<include> 标记可以引用描述源代码中类型和成员的另一文件中的注释。这是除了将文档注释直接置于源代码文件中之外的另一种可选方法。<include> 标记使用 XML XPath 语法。有关自定义 <include> 使用的方法，请参见 XPath 文档。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[
```

```
/// @name="test"]/*' />
class Test{
    static void Main(){
    }
}
/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[
///  @name="test2"]/*' />
class Test2{
    public void test(){
    }
}
```

(12) <paramref>(optional)

该标记的格式如下:

```
<paramref name="name"/>
```

参数 name : 要引用的参数名。将此名称用双引号括起来 (" ")。<paramref> 标记提供了指示代码注释中的某个单词（例如在 <summary> 或 <remarks> 块中）引用某个参数的方式。可以处理 XML 文件来以不同的方式格式化此单词，比如将其设置为粗体或斜体。

示例如下:

```
// compile with: /doc:DocFileName.xml
/// text for class TestClass
public class TestClass{
/// <summary>doWork is a method in the TestClass class.
/// The <paramref name="int1"/> parameter takes a
///  number.
/// </summary>
    public static void doWork(int number){
    }
    /// text for Main
    static void Main(){
    }
}
```

(13) <permission>(optional)

该标记的格式如下:

```
<permission cref="member">description </permission>
```

参数 cref = "member" 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 member 转换为输出 XML 中的规范化元素名。必须将 member 括在双引号 (" ") 中。

description: 对成员的访问的说明。<permission> 标记使得你可以将成员的访问记入文档。使用 PermissionSet 类可以指定对成员的访问。

示例如下:

```
// compile with: /doc:DocFileName.xml
class TestClass{
/// <permission cref="System.Security.PermissionSet">
/// Everyone can access this method.</permission>
    public static void test(){
    }
    static void Main(){
    }
}
```

(14) <see>(optional)

该标记的格式如下:

```
<see cref="member"/>
```

参数 cref = "member" 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查给定的代码元素是否存在，并将 member 传递给输出 XML 中的元素名称。应将 member 放在双引号 (" ") 中。

示例如下:

```
// compile with: /doc:DocFileName.xml
// the following cref shows how to specify the reference,
// such that,
// the compiler will resolve the reference
/// <summary cref="C{T}">
```

```
/// </summary>
class A {
}
// the following cref shows another way to specify the reference,
// such that, the compiler will resolve the reference
// <summary cref="C &lt; T &gt;">
// the following cref shows how to hard-code the reference
/// <summary cref="T:C`1">
/// </summary>
class B{
}
/// <summary cref="A">
/// </summary>
/// <typeparam name="T"></typeparam>
class C<T>{
}
```

(15) <typeparam> (optional)

该标记的格式如下:

```
<typeparam name="name">description </typeparam>
```

参数 name: 类型参数的名称。将此名称用双引号括起来 (" ")。description: 类型参数的说明。在泛型类型或方法声明的注释中应该使用 <typeparam> 标记描述类型参数。为泛型类型或方法的每个类型参数添加标记。=

5.2.2 程序示例

这里我们假设继续使用开源协议中的 GPL 协议。示例程序是按照 C# 命名规范和 C# 文档注释要求写的。

示例如下：

```
/*
 * Copyright 2012-2014 南京大学 All Rights Reserved.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under terms of the GNU General Public License as published by
 * the Free Software Foundation,either version 3 of the License,or
 * (at you option) any later version
 *
 * This program is distributed in the hope that it will by useful
 * but WITHOUT ANY WARRANTY; without even the implied warrant of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PuRPOSE.
 * See the GNU General Public License for more details.
 * You should hava received a copy of the GNU General Public License
 * along with this program. If not,see <http://www.gnu.org/licenses/>
 *
 * Author  : zouyh <zouyh@nju.nlp.cn>
 * version : 1.0
 *
 * Description:
 * 描绘了抽象类SomeClass。
 * SomeClass类提供了SomeMethod()方法和SomeOtherMethod()方法
 * 文件中只有一个类。
 *
 * Update History:
 * 10-10-2010 zouyh : 新增函数someOtherMethod();
 *
 */
// XMLsample.cs
// compile with: /doc:XMLsample.xml
using System;

/// <summary>
/// Class level summary documentation goes here.</summary>
/// <remarks>
/// Longer comments can be associated with a type or member
/// through the remarks tag</remarks>
public class SomeClass
```

文件注释

类注释

图 5.4　CSharp 文件注释一

```
{
    /// <summary>
    /// Store for the name property</summary>
    private string myName = null;

    /// <summary>
    /// The class constructor. </summary>
    public SomeClass()
    {
        // TODO: Add Constructor Logic here
    }
    /// <summary>
    /// Name property </summary>
    /// <value>
    /// A value tag is used to describe the property value</value>
    public string Name
    {
       get
       {
          if ( myName == null )
          {
             throw new Exception("Name is null");
          }

          return myName;
       }
    }

    /// <summary>
    /// Description for someMethod.</summary>
    /// <param name="s"> Parameter description for s goes here</param>
    /// <seealso cref="String">
    /// You can use the cref attribute on any tag to reference a type or member
    /// and the compiler will check that the reference exists. </seealso>
    public void someMethod(string s)
    {
    }
    /// <summary>
    /// Some other method. </summary>
    /// <returns>
    /// Return results are described through the returns tag.</returns>
    /// <seealso cref="someMethod(string)">
    /// Notice the use of the cref attribute to reference a specific method </seealso>
    public int someOtherMethod()
    {
       return 0;
    }

    /// <summary>
    /// The entry point for the application.
    /// </summary>
    /// <param name="args"> A list of command line arguments</param>
    public static int Main(String[] args)
    {
       // TODO: Add code to start application here

        return 0;
    }
}
```

方法注释

方法注释

图 5.5　CSharp 文件注释二

5.2.3 xml 文档

示例如下

```xml
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>ConsoleApplication1</name>
    </assembly>
    <members>
        <member name="T:SomeClass">
            <summary>
            Class level summary documentation goes here.</summary>
            <remarks>
            Longer comments can be associated with a type or member
            through the remarks tag</remarks>
        </member>
        <member name="F:SomeClass.myName">
            <summary>
            Store for the name property</summary>
        </member>
        <member name="M:SomeClass.#ctor">
            <summary>
            The class constructor. </summary>
        </member>
        <member name="M:SomeClass.SomeMethod(System.String)">
            <summary>
            Description for SomeMethod.</summary>
            <param name="s"> Parameter description for s goes here</param>
            <seealso cref="T:System.String">
            You can use the cref attribute on any tag to reference a type or member
            and the compiler will check that the reference exists. </seealso>
        </member>
```

图 5.6 xml 示例 -1

```xml
        <member name="M:SomeClass.SomeOtherMethod">
            <summary>
            Some other method. </summary>
            <returns>
            Return results are described through the returns tag.</returns>
            <seealso cref="M:SomeClass.SomeMethod(System.String)">
            Notice the use of the cref attribute to reference a specific method </seealso>
        </member>
        <member name="M:SomeClass.Main(System.String[])">
            <summary>
            The entry point for the application.
            </summary>
            <param name="args"> A list of command line arguments</param>
        </member>
        <member name="P:SomeClass.Name">
            <summary>
            Name property </summary>
            <value>
            A value tag is used to describe the property value</value>
        </member>
    </members>
</doc>
```

图 5.7 xml 示例 -2

图 5.8 xml 文件

5.3 redmine 使用介绍

Redmine 是用 Ruby 开发的基于 web 的项目管理软件，是用 Ruby On Rails 框架开发的一套跨平台项目管理系统。redmine 使用介绍如下所示:

5.3.1 新建项目

点击这里可以创建项目。如下所示，请注意在项目描述里面必须添加项目在服务器上的绝对路径。

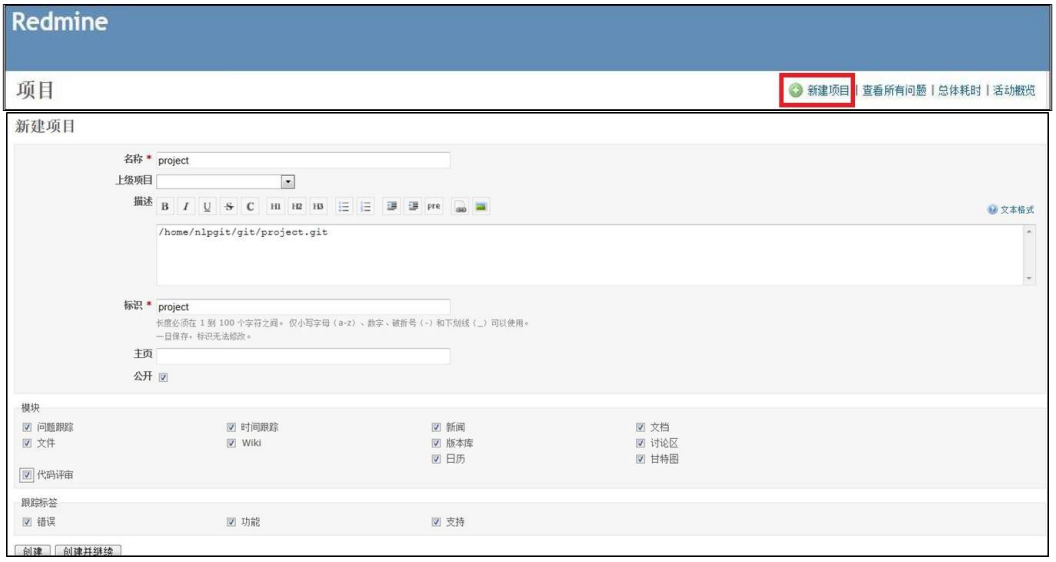


图 5.9 redmine-project 示例

5.3.2 讨论区使用

讨论区给项目成员之间提供一个交流的平台。这里可以讨论跟项目相关的问题

图 5.10 redmine-forum 示例1

讨论区列表页面显示的内容：

(1) 话题的总数

(2) 留言的总数

(3) 最后留言的链接

发起一个话题

点击右上角的“新帖”链接，进入新建帖子的页面，输入主题和内容，点击创建按钮，一个新的话题就发起了。

图 5.11 redmine-forum 示例2

话题里有两个可选选项：

(1) 置顶

如果选中，表示该话题将会在讨论区列表置顶，并加粗显示

(2) 锁定

如果选中，表示该贴不允许用户跟贴

5.3.3 wiki 使用

WiKi 是一个供多人协同写作的系统, 这里可以写项目的功能，特色，更新进度。

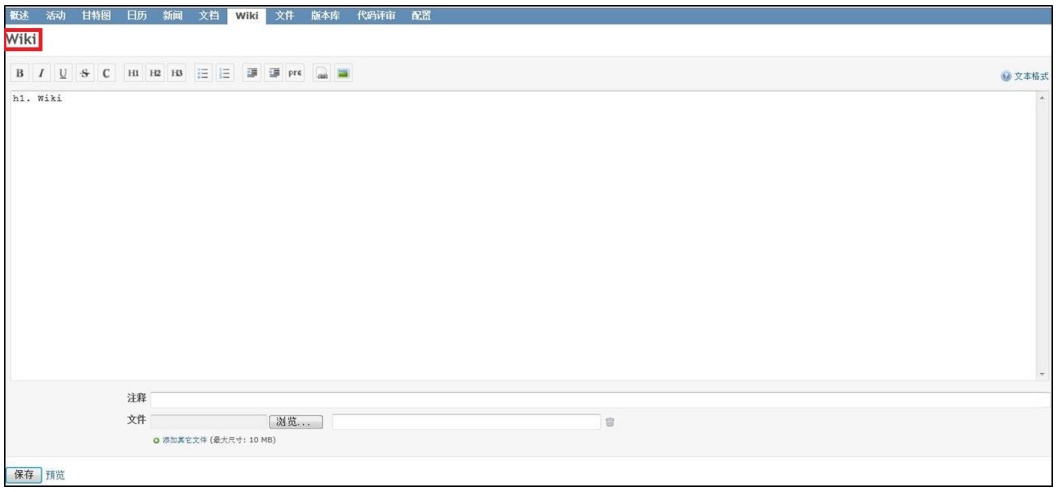


图 5.12 redmine-wiki 示例

5.3.4 版本库使用

参见 git 一节介绍

5.3.5 文件使用

图 5.13 redmine-file 示例

这里可以上传项目需要用到的文件比如设计文档，使用文档，更新记录和 readme 文件。

5.3.6 代码评审

代码评审，可以用来在同一项目的小组成员之间互相检查代码，熟悉项目，减少项目风险。点击代码评审后出现上图，接着就可以审阅代码了。

图 5.14 redmine-codereview 示例

5.4 git 使用介绍

git 是开源版本控制系统，具有简单的设计、对非线性开发模式的强力支持（允许上千个并行开发的分支)、完全分布式、有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量)，下面的文档将会 git 配置，git 管理本地代码，git 管理服务器代码等方面展开介绍。

下面的例子中，所有标注为红色的部分需要在自己的机器上做出相应切换，例如下面例子中的，longsail是邹远航电脑的用户名，在操作的时候需要换成自己的电脑用户名，mypassword换成自己设置的密码，project换成自己的项目名称。其它一些切换，在下面也会有相应的说明。

Git 客户端可以在 http://114.212.189.55:2012 里下载到，登录到 redmine 后，点击 git-client 项目，然后点击版本库，然后点击 Git-1.8.0-preview20121022.rar，就可以下载了。

在 http://114.212.189.55:2012 首页有名称为 test 的项目，在 test 目录下面已经有了一个 kmeans 项目，大家想做测试的话，可以将项目建立在这个目录下面或者直接对这个 kmeans 进行操作。

5.4.1 使用命令介绍

在 git 的配置以及使用中将会涉及到 linux 和 git 命令操作，下面将解释这些命令的意思。

(1) linux 命令介绍

$cd directory

/* 切换到目录 directory */

$ls

/* 查看当前目录下的文件 */

$mkdir directory

/* 创建新的目录 directory */

$rm file

/* 删除文件 file */

$ssh-keygen

/* 生成公钥 */

$ssh user@server

/* 用户 user 以 ssh 方式登录到服务器 server */

$scp /.ssh/zouyh_rsa.pub user@server:/home/nlpgit

/* 将用户 zouyh 机器上的公钥以 user 的用户名上传到服务器 server（114.212.189.18）的/home/nlpgit 文件夹下面 */

$cat zouyh_rsa.pub » /.ssh/authorized_keys

/* 将用户 zouyh 机器上产生的公钥拷贝到服务器上的 authorized_keys 文件中
这样我们以后就可以跟服务器通信了
*/

(2) git 命令介绍

a) git 一般操作命令

$git init

/* 从当前目录初始化项目 */

$git config –global user.email ”email@email.com”

/* 配置个人邮箱这里建议使用组内邮箱 */

$git config –global user.name username

/* 配置个人用户名这里建议使用自己名字的缩写 */

$git config –global core.editor notepad

/* 设置默认使用的文本编辑器为记事本如果不设置默认的编辑器是 vim */

$git –bare init

/* 在服务器端初始化自己的项目 */

$git status

/* 查看哪些文件当前处于什么状态 *

$git add README

/* 使用命令 git add 开始跟踪一个新文件 README */

$git commit -m ’initial commit’

/* 提交更新，‘initial commit’是提交的说明 */

$git commit –amend

/* 撤消之前的提交操作 */

$git log

/* 按提交时间列出所有的更新，最近的更新排在最上面 */

b) git 特殊操作命令

$git clone /opt/git/project.git

/* 克隆本地仓库项目目录是 /opt/git/project.git */

$git remote add local_proj /opt/git/project.git */

/* 添加一个本地仓库 local_proj 到现有 Git 工程, 工程目录是 /opt/git/project.git */

$git clone ssh:user@server:project.git

/* 通过 SSH 克隆一个 Git 仓库，user 是用户名，server 是服务器的 ip 地址 */

$git remote add origin user@server:project.git

/* 添加远程仓库到服务器下的目录 project.git 中 */

$git push origin master

/* 推送数据到远程仓库 */

*/

5.4.2 git 使用配置

(1) git 客户端用户名和邮箱，文本编辑器设置

以潘林林的配置为例，命令中 global 前面是两个 -。在自己机器配置时候红色部分”panll@nlp.nju.edu.cn”、panll替换成自己的邮箱和用户名。

```
$git config --global user.email "panll@nlp.nju.edu.cn"
$git config --global user.name panll
```

可选的配置，如果对 vim 或者 emacs 不熟悉的，可以使用记事本或者其他自己熟悉的文本编辑工具，这里假设配置为记事本（notepad)

```
$git config --global core.editor notepad
```

如果仅仅使用 git 管理本地项目，配置可以到此结束了。由于我们组使用 redmine 平台，涉及到网络操作，需要进行进一步的配置，如下所示：

(2) 生成公钥

```
$ssh-keygen
```

注意：ssh-keygen 这条命令执行后出现下面的提示，红色部分是需要自己输入的

Generating public/private rsa key pair. Enter file in which to save the key (/c/Users/longsail/.ssh/id_rsa):/c/Users/longsail/.ssh/id_rsa
Enter passphrase (empty for no passphrase):mypasswd
Enter same passphrase again:mypasswd

补充说明：

(a) /c/Users/longsail/.ssh/id_rsa

这里是本机 ssh 公钥保存的位置，注意将 longsail 替换为自己本机的名称。

(b) mypassword

是以后你跟服务器通信的密码，需要自己设置

(3) 上传公钥到服务器

为了便于管理公钥，请将/c/Users/longsail/.ssh 目录下面的 id_rsa.pub 文件重新命名, 例如，邹远航的公钥修改为zouyh_rsa.pub, 红色部分替换为自己的名字。

下面的命令结束后要求输入密码，第一个密码输入自己设置的密码，第二个密码输入 nlpgit。

```
$scp ~/.ssh/zouyh_rsa.pub nlpgit@114.212.189.55:/home/nlpgit
$ssh nlpgit@114.212.189.55
```

将密钥拷贝到服务器的 /.ssh/authorized_keys 文件中。

```
$cat zouyh_rsa.pub >> ~/.ssh/authorized_keys
$exit
```

git 本机的配置完了，就可以进行网络操作了

5.4.3 git 使用操作

我们可以用 git 来管理本地项目和服务器上的项目，在下面的介绍中，将详细介绍怎样利用 git 在服务器上建立项目以及对项目进行操作，从本地克隆项目和从服务器上克隆项目的操作例如提交更新，上传到远程服务器命令是相似的，就不作详细介绍了。

5.4.3.1 本地操作

git 可以用来管理自己在本机上的代码，文档等等, 使用的是本地协议 (Local protocol)，远程仓库在该协议中就是硬盘上的另一个目录。常见于团队每一个成员都对一个共享的文件系统 (例如 NFS) 拥有访问权，抑或比较少见的多人共用同一台电脑的时候或者仅仅是自己管理自己的项目。

如果使用一个共享的文件系统，就可以在一个本地仓库里克隆，推送和获取。要从这样的仓库里克隆或者将其作为远程仓库添加现有工程里，可以用指向该仓库的路径作为 URL。

克隆一个本地仓库，可以用如下命令完成, 假设你要管理 c 盘 git 目录下的 project:

$git clone /c/git/project.git

其它共同操作参考 git 共同操作。

5.4.3.2 远程操作

这里我们以在服务器上建立项目和从服务器上克隆项目为例，在服务器上建立项目需要跟 redmine 结合起来，主要包含三个操作，服务器端操作、本地机器操作、redmine 上面的操作。从服务器上克隆项目相对简单，我们将结合一个命令讲下。

(a) 服务器端操作

红色部分project.git 请替换成自己的项目名称，比如 zouyh.git(这里的命名方式不是规范的，仅仅是为了操作方便)

```
$ssh nlpgit@114.212.189.55
```

```
$cd git
$mkdir project.git
$cd project.git
$git --bare init
$exit
```

(b) 客户端操作

从当前目录初始化, 这里以新建一个空的文件目录为例，并且在新建的文件目录里面添加一个 readme 文件，上传到服务器。(在空目录中执行上传操作会报错，为了上传成功添加一个 readme 文件，仅仅是为了操作方便)。

```
$mkdir project
$cd project
$git init
$git add .
$vim readme
$git add readme
$git commit -m 'initial commit'
$git remote add origin nlpgit@114.212.189.55:/home/nlpgit/git
 /project.git
$git push origin master
```

完成上述操作后将会出现下面的输入密码的提示,mypasswd 替换为自己设置的密码,

Enter passphrase for key '/c/Users/longsail/.ssh/id_rsa':mypasswd

(c) redmine 上的操作

在 redmine 上新建项目 project

点击配置 -» 版本库

点击新建版本库，在库路径中填写：/home/nlpgit/git/project.git,然后点击创建

然后点击版本库

这样你的项目就建立成功了，如下图所示：

图 5.15　redmine-git 示例

5.4.3.3　从服务器上克隆项目

这里我们以付强在服务器上建立的项目 fq.git 为例，如果想将服务器上名称为 fq.git 克隆到自己的主机，可以执行下面的操作。

```
$git clone nlpgit@114.212.189.55:/home/nlpgit/git/project.git
```

其它共同操作参考 git 共同操作。

致　　谢

本文档的最初版本由黄书剑和邹远航合作完成。在编写过程中，黄书剑给出了文档的基本思路和框架结构，邹远航收集整理了大量的资料并完成了初稿的写作。

参 考 文 献

[1] ROBERT. Clean Code. Prentice Hall, 2010.
[2] BRUCE. Thinking in Java. Prentice Hall, 2006.
[3] SCOTT. Pro Git Apress 2009.