Tools Solutions

陈磊

2017 年 6 月 13 日

目录

1 代码质量

1.1 checkstyle

1.1.1 maven

```
<reporting>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-checkstyle-plugin</artifactId>
         <version>2.17</version>
         <configuration>
            <!--自定义 checkstyle 配置路径-->
            <configLocation>${project.basedir}/checkstyle.xml</
                ↪ configLocation>
         </configuration>
      </plugin>

      <!--使 checkstyle 结果可以直接跳转到代码行位置-->
      <plugin>
```

```
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jxr-plugin</artifactId>
            <version>2.3</version>
        </plugin>
    </plugins>
</reporting>
```

`mvn site` 执行后, 打开 `target/site/index.html`, 在 Project Reports -> Checkstyle 查看报告. 点击对应行数, 可查看具体代码位置.

checkstyle 常见问题包括:

1. 代码缩进不对.
2. 控制语句没加大括号.
3. `equals` 字符串应放在前面.
4. if-else 条件语句包含 return 的未简化 (可去除 if-else 直接 return).
5. 无用的包引入.
6. 命名问题: 包名、变量名、类名、函数名.
7. 字符串太长 (超出限制).
8. 缺注释.

checkstyle 修改快捷方法 (eclipse 或者 IDEA):

1. 变量名修改可用 Refactor 的快捷键. eclipse、IDEA 可用.
2. 缩进或控制语句缺大括号, 可配置格式化属性 (或导入模板), 快捷键针对工程中所有文件格式化, 比如 IDEA 的 Ctrl+Alt+L. eclipse, IDEA 可用.
3. eclipse 可安装 javadoc (不是这个名字), 对整个包快捷键添加注释.

1.2 findbugs

1.2.1 IDEA 插件

idea 编辑器可安装 FindBugs-IDEA 插件 (File->Settings->Plugins->Browse repo...). 安装重启后编辑器后, 底部会显示. 选择 src 文件夹, 点击 `Analyze Module Files` 可分析全部源文件.

1.2.2 maven

```
<reporting>
    <plugins>
```

```
      <!-- https://mvnrepository.com/artifact/org.codehaus.mojo/
          ↪ findbugs-maven-plugin -->
      <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>findbugs-maven-plugin</artifactId>
         <version>3.0.4</version>
      </plugin>
   </plugins>
</reporting>
```

`mvn site` 执行后在 Project Reports -> FindBugs 查看报告. `mvn site` 执行下载 FindBugs 相关依赖错误时, 可尝试用 VPN 解决.

1.3 sonar

1.3.1 IDEA 插件

idea 编辑器可安装 SonarLint 插件 (File->Settings->Plugins->Browse repo...). 安装重启后, 点击底部显示的 SonarLint, 并点击此 tab 里面的 Project Files, Scope 项选择 All projects files, 点击绿色运行图标进行分析. 在编写代码过程中, 可切换到 Current file 实时查看当前编写代码的问题.

1.3.2 maven

需配置数据库.

2 持续集成

2.1 travis-ci

2.1.1 travis-ci 使用流程

travis-ci 结合 github 使用, 每次提交自动执行编译或测试任务.

1. <travis-ci.org> 用 github 账户授权登录
2. travis-ci 上添加需要持续集成的 github repo
3. github repo 中添加 `.travis.yml` 配置, 用以配置 travis 所要执行的任务和环境
4. repo 更改完成, 提交, 可在 travis-ci 上实时查看任务执行过程

备注:

2.1.2 添加编译状态图标

```
[![Build Status](https://travis-ci.org/HereChen/Tools-Solutions.svg?
    ↪ branch=master)](https://travis-ci.org/HereChen/Tools-Solutions)
```

2.1.3 travis 配置文件样本

此库的 `.travis.yml` 样本

```
sudo: required
dist: trusty   # ubuntu 14.04

# install
before_install:
   - sudo apt-get update
   - sudo apt-get -y install texlive-full
   - sudo apt-get -y install pandoc

before_script:
   - chmod +x build.sh

script:
   - ./build.sh
```

3 版本控制

3.1 git 多账户

应用于多个代码托管站点多账户场景, 比如 github、gitlab、码云等[1][2]. 配置完成后方便多个账户同时使用. 以 github 和 gitlab 为例, 按照以下步骤完成配置. 除了以下提供的配置方法, Windows 10 上可以用 Bash On Ubuntu On Windows 额外添加一个 git 账户, 适用于只有两个账户的情况.

3.1.1 git 多账户配置

1. 生成公钥

生成公钥并添加到代码托管站点. 生成的时候, 输入不同的文件名保存.

[1] Multiple SSH Keys settings for different github account

[2] ssh hostname returns “Bad owner or permissions on ~/.ssh/config”

```
ssh-keygen -t rsa -C "chenlei.here@qq.com"
```

```
chmod 600 ~/.ssh/config
ssh-agent bash
ssh-add ~/.ssh/id_rsa_gitlab
ssh-add ~/.ssh/id_rsa_github
```

2. 添加 ssh 配置文件

   创建 config 文件 (`touch ~/.ssh/config`), 并添加以下内容.

```
# gitlab
Host gitlab
    HostName gitlab.com
    User HereChen
    IdentityFile ~/.ssh/id_rsa_gitlab

# github
Host github
    HostName github.com
    User HereChen
    IdentityFile ~/.ssh/id_rsa_github
```

3. 为项目设置单独的账户配置

```
git clone git@github.com:HereChen/Tools-Solutions.git
cd Tools-Solutions
git config user.name "HereChen"
git config user.email "chenlei.here@qq.com"
```

3.1.2 启动时添加 SSH

每次重新启动系统或者打开 Git 客户端, ssh-add 添加的信息失效, 可配置在启动时添加. Linux 可在 barshrc 文件 (`sudo vim ~/.bashrc`) 中添加以下内容.

```
# https://unix.stackexchange.com/questions/90853/how-can-i-run-ssh-add-
    ↪ automatically-without-password-prompt
if [ -z "$SSH_AUTH_SOCK" ] ; then
  eval `ssh-agent -s`
  ssh-add -D
  ssh-add ~/.ssh/id_rsa_oschina
  ssh-add ~/.ssh/id_rsa_github
```

```
  ssh-add -l
fi
```

3.1.3 git 一个库向两个托管服务器提交

3.2 git 提交信息格式化

1. 工具安装

```
npm install -g commitizen
npm install -g cz-conventional-changelog
```

2. 初始化

```
commitizen init cz-conventional-changelog --save --save-exact
```

3. 提交

```
git cz -a
```

提交时需要填写以下信息: 选择提交的类型 (新功能, bug 修复等等), 描述修改的范围, 简要描述修改的内容, 具体描述修改的内容.

4 文档处理工具

4.1 Pandoc

Pandoc 主页 `http://pandoc.org`. Pandoc 可以实现不同格式文档的相互转换, 支持的格式包括 markdown、HTML、LaTeX、docx、EPUB ODT 等等.

4.1.1 markdown 多个文档转换

```
pandoc src/markdown/*.md -o build/latex/content.tex
```

4.1.2 markdown 转 LaTeX, 用 listings 包替换 verbatim

```
pandoc --listings src/markdown/*.md -o build/latex/content.tex
```

4.2 LaTeX

4.2.1 Windows 上编译缓慢, 经常卡在 eu1lmr.fd

删除 `texlive\2016\texmf-var\fonts\cache` 文件夹下字体缓存, 重新生成字体缓存, 执行 `texlive\2016\bin\win32\fc-cache.exe`[3]. 可更改示例代码的路径, 编译前执行一次.

```
del /q D:\applications\texlive\2016\texmf-var\fonts\cache\*
D:\applications\texlive\2016\bin\win32\fc-cache.exe
```

4.2.2 listings 代码换行

添加 `breaklines=true` 后, 还是存在没有空格的 url 无法换行, 需设置 `breakatwhitespace=false`. 另外, 可以在代码换行的地方添加箭头标识换行[4].

```
\lstset{
    breaklines=true,
    % 箭头标识换行
    postbreak=\raisebox{0ex}[0ex][0ex]{\ensuremath{\color{red}\
        ↪ hookrightarrow\space}},
    breakatwhitespace=false
}
```

5 编辑器及插件

5.1 Visual Studio Code

插件

- **Markdown TOC**, markdown 文本的目录生成, 可应用于 wiki 或博客目录生成.
- **REST Client**, 在文本编辑器中测试 API
- **Project Manager**, 类似于 Sublime Text 快捷切换项目目录

[3] LaTeX 编译卡顿怎么办?

[4] Pandoc: Markdown to PDF, without cutting off code block lines that are too long , Code not wrapping in listings even though breaklines=true , lstlisting line wrapping

6 工具安装

6.1 nodejs

安装的方式包括: 从源码编译安装, 直接下载二进制, 依靠工具直接从库下载安装.

从源码安装

```
curl -o node-v6.10.0.tar.gz https://nodejs.org/dist/v6.10.0/node-v6
    ↪ .10.0.tar.gz
tar -xvzf node-v6.10.0.tar.gz
cd node-v6.10.0
./configure
make
make install
```

CentOS 下载二进制安装

```
# 64bit node install
curl -o node-v6.10.0-linux-x64.tar.xz https://nodejs.org/dist/v6.10.0/
    ↪ node-v6.10.0-linux-x64.tar.xz
mv /opt/
cd /opt/
tar -xvJf node-v6.10.0-linux-x64.tar.xz
mv node-v6.10.0-linux-x64 node

vi ~/.bash_profile
# 添加以下内容到 .bash_profile
# export NODE_HOME=/opt/node
# export PATH=$NODE_HOME/bin:$PATH
source ~/.bash_profile

node -v
```

Ubuntu 从库中下载安装

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

6.2 git

6.2.1 Ubuntu

安装最新版的 git[5].

```
sudo add-apt-repository ppa:git-core/ppa
sudo apt-get update
sudo apt-get install git
```

如果上面执行报错 `Errors were encountered while processing: runit git -daemon-run`, 尝试执行下面命令.

```
sudo apt-get purge runit
sudo apt-get purge git-all
sudo apt-get purge git
sudo apt-get autoremove
sudo apt update
sudo apt install git
```

6.2.2 CentOS 从源码安装 git

执行 `yum install git` 安装的版本比较老, 安装最新版可从源码安装[6].

```
# 依赖工具安装
yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel
yum install gcc perl-ExtUtils-MakeMaker

# git 源码下载
# https://www.kernel.org/pub/software/scm/git/
curl -o git-2.9.3.tar.xz https://www.kernel.org/pub/software/scm/git/git-2.9.3.tar.xz
tar -xvJf git-2.9.3.tar.xz

# 编译安装 git
cd git-2.9.3
make prefix=/usr/local all
sudo make prefix=/usr/local install
```

[5]Installing Latest version of git in ubuntu, How to fix processing with runit and git-daemon-run [duplicate]

[6]How to install latest version of git on CentOS 6.x/7.x , 起步 - 安装 Git