

- **【会议安排】：**
 - **主题：**【Java 代码格式规范-内部-小范围讨论】
 - **时间：**16:30 - 17:30（预期10~30 mins，快速同步、讨论）
 - **地点：**3F-新加坡会议室，7人
 - **人员：**不超过 7 人（群中成员，选择性参加，一般不建议参加）
- **【内容要点】：**
 - 代码格式规范，Java Code Style： <https://wiki.mobike.com/pages/viewpage.action?pageId=32567024>

小结

- 摩拜的 **Code Style**：代码格式化
 - 统一的、良好的、通用的；
 - 定制版的 *Google Java Code Style*
- 摩拜的 **Check Style**：代码校验
 - 跟 Code Style 完全对应
 - 定制版的 *Google Check Style*
- Checkstyle 检查什么？
 - Import
 - 空白、修饰符
 - 命名约定、代码、类设计
 - 混合检查（包括一些有用的比如非必须的System.out和printStackTrace)

Java 代码规范: Code Style 和 Check Style

guoning@mobike.com
2018/08/02

目录

- 为什么?
- 怎么做?
 - 明确规范
 - 使用规范
 - 代码格式化
 - 代码格式校验
- 参考资料

为什么

- 背景：
 - 不同的小组\同学，采用**不同的**代码**格式**规范
 - 导致每次 **format 代码**，都有大量的变化
 - **review 代码**时，引入很多**干扰**项
- 目标：
 - **统一**代码**格式**规范
 - 保证 format 代码时，**不会**引入格式上的**干扰**
 - 提升小组**协作效率**、代码 review 效率

为什么

```
1 package com.mobike.abacus.order;
2
3 import javax.annotation.Resource;
4
5 import org.springframework.boot.SpringApplication;
6 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8 @@ -18,19 +20,14 @@ import
9 org.springframework.security.web.util.matcher.AntPathRequestMatcher;
10 import org.springframework.session.data.redis.config.ConfigureRedisAction;
11 import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;
12
13 import com.mobike.abacus.order.common.CustomLogoutHandler;
14 import com.mobike.abacus.order.filter.WhiteListUtils;
15
16 import lombok.extern.slf4j.Slf4j;
17
18 @SpringBootApplication
19 @EnableDiscoveryClient
20 @EnableOAuth2Sso
21 @Slf4j
22 @EnableRedisHttpSession(redisNamespace = "abacus_order")
23 @EnableAutoConfiguration
24 @EnableFeignClients(basePackages = { "com.mobike.client.payment", "com.mobike.client.user",
25     "com.mobike.client.member", "com.mobike.client.sms",
26     "com.mobike.client.creditsystem", "com.mobike.push.client" })
27
28 public class AbacusOrderApplication extends WebSecurityConfigurerAdapter {
29
30     public static void main(String[] args) {
31
32 @@ -46,11 +43,17 @@ public class AbacusOrderApplication extends
33 WebSecurityConfigurerAdapter {
34
35     // 1. 基本配置
36     http
37
38         // a. 登出控制
39         .logout().logoutSuccessUrl("/").logoutRequestMatcher(new AntPathRequestMat
40 cher("/logout")).addLogoutHandler(customLogoutHandler).and()
```

```
1 package com.mobike.abacus.order;
2
3 + import com.mobike.abacus.order.common.CustomLogoutHandler;
4 + import com.mobike.abacus.order.filter.WhiteListUtils;
5 import javax.annotation.Resource;
6 + import lombok.extern.slf4j.Slf4j;
7 import org.springframework.boot.SpringApplication;
8 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
9 import org.springframework.boot.autoconfigure.SpringBootApplication;
10 @@ -18,19 +20,14 @@ import
11 org.springframework.security.web.util.matcher.AntPathRequestMatcher;
12 import org.springframework.session.data.redis.config.ConfigureRedisAction;
13 import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;
14
15 import com.mobike.abacus.order.common.CustomLogoutHandler;
16 import com.mobike.abacus.order.filter.WhiteListUtils;
17
18 import lombok.extern.slf4j.Slf4j;
19
20 @SpringBootApplication
21 @EnableDiscoveryClient
22 @EnableOAuth2Sso
23 @Slf4j
24 @EnableRedisHttpSession(redisNamespace = "abacus_order")
25 @EnableAutoConfiguration
26 @EnableFeignClients(basePackages = {"com.mobike.client.payment", "com.mobike.client.user",
27     "com.mobike.client.member",
28     "com.mobike.client.sms", "com.mobike.client.creditsystem", "com.mobike.push.clien
29 t"})
30
31 public class AbacusOrderApplication extends WebSecurityConfigurerAdapter {
32
33     public static void main(String[] args) {
34
35 @@ -46,11 +43,17 @@ public class AbacusOrderApplication extends
36 WebSecurityConfigurerAdapter {
37
38     // 1. 基本配置
39     http
40
41         // a. 登出控制
42         .logout().logoutSuccessUrl("/")
43         .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
44         .addLogoutHandler(customLogoutHandler).and()
```

位置变化

格式变化

目录

- 为什么?
- 怎么做?
 - 明确规范
 - 使用规范
 - 代码格式化
 - 代码格式校验
- 参考资料

怎么做：明确规范

- 统一的、良好的、通用的，Java 代码规范：
 - 定制版的 *Google Java Code Style*
- 选用 Google Java Code Style，具体原因：
 - 业界使用广泛，基本是通用标准
 - 自动化校验工具完善，有完善的 checkstyle 配置文档

-

怎么做：明确规范

- 为满足「代码美感」，借鉴其他公司的定制 & 公司内部建议，进行一些**摩拜的定制**：

- GoogleStyle-Mobike 的 Java 代码规范：

- **IntelliJ IDEA**：intellij-java-google-style.xml ([在线查看](#))

- 摩拜定制的工程：<https://mobike.io/guoning/google-styleguide>

-

修改 `intellij-java-google-style.xml` 中的内容：

项	原始取值	定制之后	定制时间	备注
<code>INDENT_SIZE</code>	2	4	20180731	行缩进
<code>TAB_SIZE</code>	2	4	20180731	TAB 缩进
<code>CONTINUATION_INDENT_SIZE</code>	2 or 4	8	20180731	换行缩进
<code>RIGHT_MARGIN</code>	100	120	20180731	单行的长度
<code>JD_PRESERVE_LINE_FEEDS</code>	无	true	20180731	注释中, 保留手动的换行
<code>KEEP_LINE_BREAKS</code>	无	true	20180731	Java 代码中, 保留手动的换行

目录

- 为什么?
- 怎么做?
 - 明确规范
 - 使用规范
 - 代码格式化
 - 代码格式校验
- 参考资料

怎么做：使用规范-代码格式化

- 配置了 code style 后，在 Mac 下，IntelliJ IDEA 进行代码格式化：
 - 快捷键：Shift + Command + L
 - 界面操作：参考截图

怎么做：使用规范-代码格式化

The screenshot shows an IDE interface with a project structure on the left, a code editor in the center, and a terminal at the bottom. The project structure includes folders like 'src', 'main', 'java', 'com', 'mobike', 'abacus', 'order', 'util', and files like 'CertFile', 'DateUtils', 'HttpUtil', 'MakerToObject', 'Md5Encrypt', 'PingUtil', 'SslContextUtils', 'Utils', 'UUIDUtil', 'WechatUtil', 'XmlUtil', 'AbacusOrderApplication', 'resources', 'test', 'target', '.gitignore', '.gitlab-ci.yml', 'abacus-order.iml', 'docker-compose.yml', 'init.sql', 'Makefile', 'pom.xml', 'README.md', 'wait-for-mysql.sh', and 'External Libraries'.

The code editor displays the file 'SslContextUtils.java' with the following code:

```
63 ... allHostsValid = new HostnameVerifier() {
64     @Override
65     public boolean verify(String hostname, SSLSession session) { return true; }
66 };
67
68 ...
69 ...
70 ...
71 public void initHttpsConnect(HttpsURLConnection httpsConn) {
72     httpsConn.setSSLSocketFactory(sslcontext.getSocketFactory());
73     httpsConn.setHostnameVerifier(allHostsValid);
74 }
75
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 SSLContext sslcontext = SSLContexts.custom().loadKeyMaterial(keyStore, partner.toCharArray()).build();
87 SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext, new String[]{"TLSv1"}, null,
88     SSLConnectionSocketFactory.BROWSER_COMPATIBLE_HOSTNAME_VERIFIER);
89
90 String re = "";
91 try (CloseableHttpClient httpclient = HttpClients.custom().setSSLSocketFactory(sslsf).build()) {
92     HttpPost httpPost = new HttpPost(urlStr);
93     StringEntity myEntity = new StringEntity(xml, "UTF-8");
```

A red arrow points to the 'Reformat Code' option in the 'Enter action or option name:' dialog box. The dialog box also shows 'Show Reformat File Dialog', 'Show notification after reformat code action', and 'Formatting: Show notification after reformat ...' (ON).

The terminal at the bottom shows the following output:

```
+ merge Join two or more development histories together
- rebase Reapply commits on top of another base tip
- tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch Download objects and refs from another repository
  pull Fetch from and integrate with another repository or a local branch
  push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
guoning:abacus-order guoning$
```

怎么做：使用规范-代码格式校验

- 代码**自动化检查**，意义：
 - 节省人力：机器能做的事情，交给机器，特别是枯燥的事情，让机器去做
 - 避免遗漏：机器自动执行，全范围扫描
- 根据初步调研，决定采用通用的自动化**代码检查**工具：*Checkstyle*。
- 备注：蚂蚁金服、快手，团队开发过程中，都在使用 Checkstyle，进行自动化代码检查。

怎么做：使用规范-代码格式校验

- Checkstyle 会在代码开发过程中，检查代码规范，一般检查的内容包括：
 - Javadoc注释
 - 命名约定
 - 标题
 - Import
 - 大小写
 - 空白
 - 修饰符
 - 代码
 - 类设计
 - 混合检查（包活一些有用的比如非必须的System.out和printstackTrace)

怎么做：使用规范-代码格式校验

- 基于「摩拜定制的 Google Code Style」，需要对 原始 Google Code Style 的 Checkstyle 摩拜的定制：
 - 摩拜定制的 Checkstyle 配置：[在线查看](#)
 - 摩拜定制的 Checkstyle：<https://mobike.io/guoning/checkstyle>

摩拜定制 Java Code Style

为了增加代码的「美感」，摩拜内部, 进行了 java code style 的部分定制;

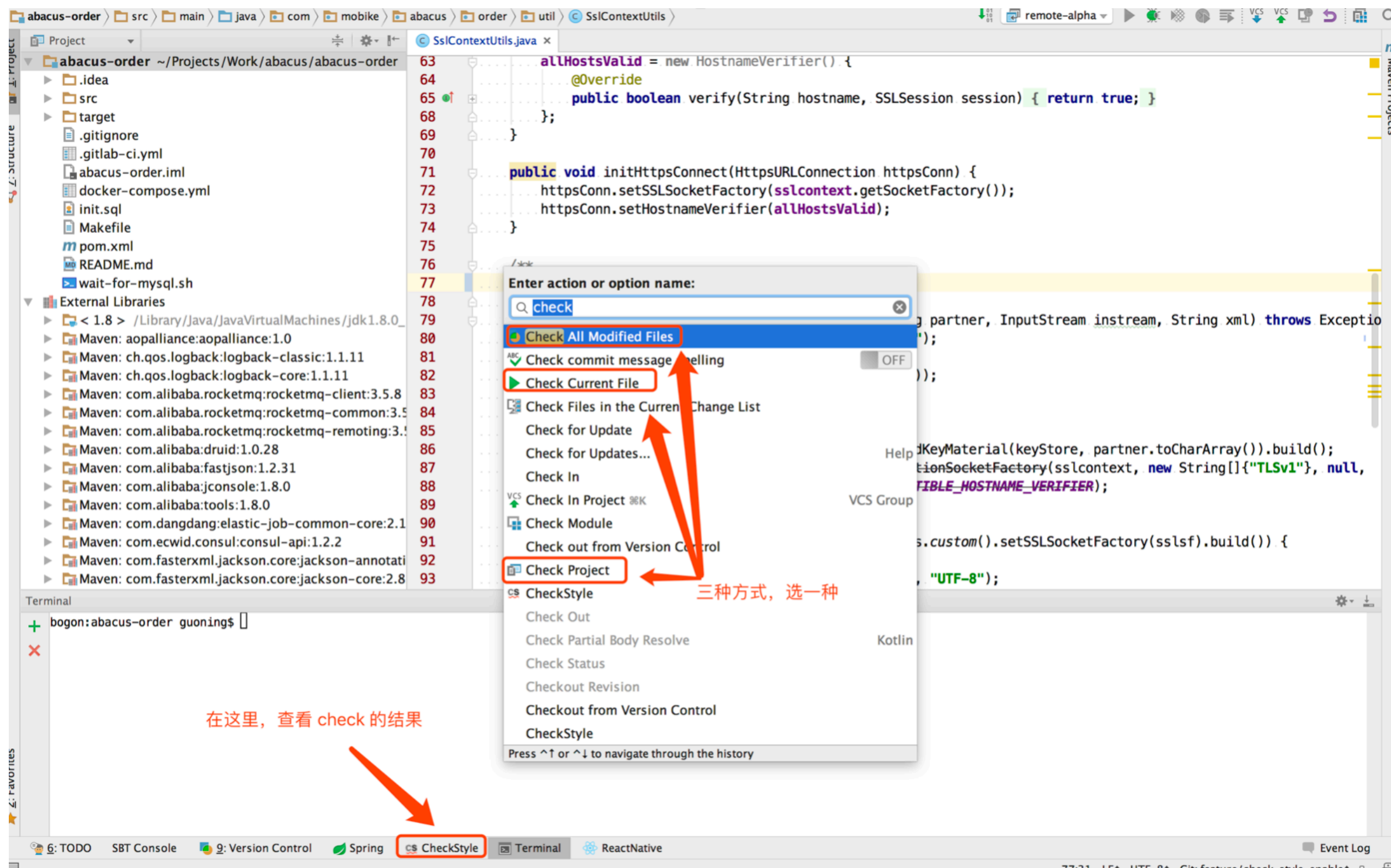
因此, 需要同步定制 checkstyle。

具体定制的内容:

项	原始取值	定制之后	定制时间	备注
<code>Indentation.basicOffset</code>	2	4	20180731	行缩进
<code>Indentation.caseIndent</code>	2	4	20180731	行缩进
<code>Indentation.throwIndent</code>	4	8	20180731	行缩进
<code>Indentation.lineWrappingIndentation</code>	4	8	20180731	行缩进
<code>Indentation.arrayInitIndent</code>	2	4	20180731	行缩进
<code>LineLength</code>	100	120	20180731	行长度
<code>JavadocParagraph</code>	启用	删除	20180731	JavaDoc 的校验
<code>SummaryJavadoc</code>	启用	删除	20180731	JavaDoc 的校验
<code>VariableDeclarationUsageDistance.allowedDistance</code>	3	10	20180801	变量的声明和使用
<code>LineLength.max</code>	120	180	20180801	单行长度校验

更多细节, 参考: [Google's Java Style Checkstyle Coverage](#)

怎么做：使用规范-代码格式校验



目录

- 为什么?
- 怎么做?
 - 明确规范
 - 使用规范
 - 代码格式化
 - 代码格式校验
- 参考资料

参考资料

- 代码格式规范：Java Code Style
- <https://github.com/google/styleguide>
- <https://github.com/checkstyle/checkstyle>

小结

- 摩拜的 **Code Style**：代码格式化
 - 统一的、良好的、通用的；
 - 定制版的 *Google Java Code Style*
- 摩拜的 **Check Style**：代码校验
 - 跟 Code Style 完全对应
 - 定制版的 *Google Check Style*
- Checkstyle 检查什么？
 - Import
 - 空白、修饰符
 - 命名约定、代码、类设计
 - 混合检查（包括一些有用的比如非必须的System.out和printStackTrace)