1.1 举例：防御式编程

防御式编程是一种防御式设计的具体体现，它保证在对程序的不可预见性的使用时，也不会造成程序功能上的破坏。下面就项目的实际代码，对防御式编程做进一步的解释说明。

代码片段 1：后端添加博文

```
90      @Override
91      public long addPost(long uid, @NonNull PostInfo post) throws EntityNotFoundException {
92          val blogger = bloggerMapper.getBloggerById(uid); // 先获取博主，后面通知粉丝时会用到博主的名字
93
94          if (blogger == null) {
95              throw new EntityNotFoundException("未找到当前用户");
96          }
97
98          post.setBlogger(blogger);
99          postMapper.addPost(post);
100         val pid = post.getId();
101         val tags = post.getTags();
102
103         if (tags ≠ null) {
104             // tags.forEach { it → /* 为每个标签设置拥有者 */ }
105             for (val tag : tags) {
106                 // 标签不存在 ⟺ tag.id == null.
107                 // 在更新博文的标签前，要先确保这些标签存在。
108                 // 因而要设置这些标签的id
109                 var tid = postMapper.findTagIdByNameAndAuthor(tag.getName(), uid);
110                 if (tid ≠ null) {
111                     tag.setId(tid);
112                     continue;
113                 }
114                 // 标签不存在。将其添加到数据库。
115                 tag.setOwner(new BloggerInfo(uid));
116                 tag.setDescription("");
117                 tagMapper.addTag(tag);
118             }
119             postMapper.updatePostTags(pid, tags);
120         }
121         if (post.getPermission().isPublic()) {
122             for (val fan : subscribeMapper.getOnesFans(uid)) {
123                 notifyFan(fan, post);
124             }
125         }
126         return pid;
127     }
```

个人博客系统的后端使用 Java 作为实际开发代码。在 Java 中，默认情况下允许对对象赋值为空值，即 null；当调用一个空对象的方法时，代码将抛出空指针异常（Null Pointer Exception, NPE），该异常是 Java 中出现最多且最难调试并解决的问题之一，甚至 null 的发明者在发明它的 50 年后在自己的文章中提到，“null 是个十亿美元的错误”。在 Java 中，避免 NPE 是防御式编程的主要工作之一。

上述代码片段中方法实现的功能为：输入一个博文信息 post 和它的作者博主 ID，将该博文插入到数据库中。具体插入数据库的功能由 postMapper 对象实现。为了避免 NPE 的产生，此方法的第 93、101、108 行分别判断相应字段是否为空，以避免产生 NPE。而根据其意义不同，防御式编程对未预见情况的处理也有所不同。如第 93~95 行，blogger 为空表示博主不存在，方法无法在这种状态下产生有意义的工作，因此直接抛出异常结束了此方法；而对于 101~123 行，标签列表为空(null)是一个未预见的情况，因为正常情况下应该会得到一个空列表([])。但这个问题不影响方法的继续执行，只要在这种情况下视为博文不需要添加标签即可，因此以“tags != null”为条件开辟了新分支，保证在列表为 null 时不执行添加标签的功能，而其余部分仍然可以正常执行。

代码片段 2：保存附件

```
48      @Override
49      public long saveAttachment(long uid, String filename, MultipartFile file) throws IOException {
50          val path = new File(ATTACHMENT_REAL_PATH);
51          if (!path.exists()) {
52              //noinspection ResultOfMethodCallIgnored
53              path.mkdirs();
54          }
55          String md5 = getFileMD5(file);
56          val f = new File(path, md5);
57          if (!f.exists()) {
58              file.transferTo(f);
59          }
60
61          val owner = new BloggerInfo();
62          owner.setId(uid);
63          int pos = filename.lastIndexOf( ch: '.');
64          val suffix = pos == -1 ? "" : filename.substring( beginIndex: pos + 1);
65          val attachInfo = new AttachmentInfo( id: 0L, filename, suffix, url: "", owner, file.getSize());
66          attachmentMapper.addAttachmentInfo(attachInfo, md5);
67          return attachInfo.getId();
68      }
```

这是另一段展示防御式编程的代码，此方法的功能是为 ID 为 uid 的博主添加一名为 filename 的附件，附件内容在 file 中。由于涉及到文件保存，所以防御式编程主要针对文件系统，如第 51 行对“文件保存到的目录不存在”的情况做出防御，这种情况会先创建目录；再如第 57 行检查是否已经有过相同的文件了，如果存在就不再重新保存。

1.2 举例：编码规范

如今软件开发界，单打独斗的软件开发人员难以获得成功。当开发某代码库的所有人遵循一致的代码规范时，会更方便开发人员相互理解相互的代码，这也是为何编码是否规范也是代码质量指标之一。

代码片段 3：SubscribeServiceImpl.java

```
1   package com.weblog.business.service.impl;
2
3   import com.weblog.business.exception.EntityNotFoundException;
4   import com.weblog.business.service.SubscribeService;
5   import com.weblog.persistence.mapper.SubscribeMapper;
6   import lombok.val;
7   import org.springframework.beans.factory.annotation.Autowired;
8   import org.springframework.stereotype.Service;
9
10  @Service
11  public class SubscribeServiceImpl implements SubscribeService {
12
13      @Autowired
14      private SubscribeMapper subscribeMapper;
15
16
17      @Override
18      public void subscribe(long publisher, long fan) throws EntityNotFoundException {
19          if (!subscribeMapper.bothBloggerExists(publisher, fan)) {
20              throw new EntityNotFoundException(
21                      String.format("Either publisher or fan not found. ids = [%d, %d]", publisher, fan));
22          }
23          subscribeMapper.subscribe(publisher, fan);
24      }
25
26      @Override
27      public void unsubscribe(long publisher, long fan) throws EntityNotFoundException {
28          val ok = subscribeMapper.unsubscribe(publisher, fan);
29          if (ok != 1) {
30              throw new EntityNotFoundException(
31                      String.format("Either publisher or fan not found. ids = [%d, %d]", publisher, fan));
32          }
33      }
34
35      @Override
36      public boolean subscribed(long publisher, long fan) {
37          return subscribeMapper.subscribed(publisher, fan);
38      }
39  }
```

上述代码片段展示了项目后端一名为 SubscribeServiceImpl.java 的源文件，这段代码展示了一些 Java 编程的编码规范，包括：

1. 源文件按顺序包含 package 语句、import 语句、一个顶级类，且三部分之间用一个空行隔开；

2. import 语句不使用通配符，且按照导入内容成组（先导入一般类，再导入注解类）；

3. 每行代码的长度均不超过 100 列；

4. 对于源文件定义的类，按照“先类成员、后类方法”的顺序排序这些成员；

5. 无论是否可以省略，都使用大括号，且左大括号前不换行且留一空格，左大括号后换行；右大括号前换行且与和它匹配的左大括号所在行缩进对齐；右大括号后在除了 else 关键字或逗号的情况都换行；

6. 使用 4 个空格作为缩进，缩进不用 TAB；

7. 继承的方法都用“@Override”注解标记；

8. 类、对象、字段、方法的命名均使用驼峰命名法，即除了第一个单词外，所有的单词都首字母大写；对于第一个单词，类名均首字母大写（即 UpperCamelCase），而对于字段或方法，首字母不大写（即 lowerCamelCase）；命名只使用 ASCII 字母，且不使用包含下划线“_”的命名；

9. 二元运算符前后留有空格，如“!=”、“=”等；调用函数的实参之间的逗号后留有空格。

代码片段 4：前端的某个函数

```
import 'mavon-editor/dist/css/index.css'
import { postBlog, editDetail } from '@/api/blog'

async function addPost(authorId, postInfo) {
  let ret = await postBlog(authorId, postInfo)
  if (ret.code !== 0) {
    console.error(ret.reason)
    return
  }
  let postId = ret.content
  ret = await editDetail(authorId, postId, that.content)
  if (ret.code !== 0) {
    console.error(ret.reason)
    return
  }
}

function submit() {
  let that = this
  const postInfo = that.contents

  postInfo.permission.isPublic = this.value
  if(that.$route.params.userid !== '') {
    const authorId = postInfo.author.id
    addPost(authorId, postInfo).catch(e => {
      console.log('Network ERROR!')
      console.error(e)
    })
  }
}
```

这段 javascript 代码来自项目前端，它展示了 Javascript 的一些编码规范，如：

1. 使用两个空格缩进；

2. 字符串统一使用单引号；

3. 判断是否相等时，始终使用三个等号“===”而非两个等号“==”，以避免期望之外的结果；

4. 避免使用 debugger 语句。仅在调试中偶尔使用它，在提交代码时不要包含它；

5. 不使用 eval 语句；

6. 避免在函数中嵌套定义函数；

7. 从同一个模块导入时，将被导入的模块一次性导入完（代码第二行）；
8. 命名方法上和 Java 相似，使用驼峰命名法；
9. 采用 JS Standard Style 的建议，没有使用分号。

1.3 举例：代码重用

代码重用是一种使用现有的代码（包括变量、函数、类、模板等）、软件以及经验来灵活构建新软件的编程技术。Weblog 在编码时也考虑了代码重用，包括：
1. 框架重用：框架已经成为特定领域开发“事实上的标准”。例如，SpringBoot 框架是 Java 后端开发事实上的标准，而 Vue/React/Angular 是前端开发事实上的标准。框架统一地定义了标准的接口，提供了可复用的抽象算法、架构、组织形式，使用户可以更关注于具体的业务逻辑实现，而不需要将精力过多放在事务处理、安全性、数据流控制等通用问题上。Weblog 个人博客管理系统在此层面上使用了 SpringBoot 和 Vue 两个框架；
2. 程序库重用：程序库为用户提供了开箱即用的变量、函数或类。Weblog 中，后端采用 Maven 管理程序库（依赖库），除了和 Springboot 框架相关的程序库，还包括了提供如 MySQL 数据库连接、内存缓存、邮件客户端、微服务发现、单元测试等功能的程序库；前端采用 node 管理程序库，除了 Vue 还包含了提供前后端交互、界面主题、Cookie 管理等功能的程序库，如 axios、element-ui、js-cookie 等；
3. 面向对象的代码重用：在 Weblog 个人博客管理系统中，后端 SpringBoot 框架提供了很多面向对象的代码重用方式。如在后端项目中，为了配置对前端网络的响应，只需要继承 springboot 框架的 WebMvcConfigurer 类，重写其中的 configureMessageConverters() 和 corsConfigurer()函数，就可以让项目按照新的配置运行；

代码片段 5：WebConfig.java

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {
        converters.add( i: 0, new MappingJackson2HttpMessageConverter());
    }

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(@NotNull CorsRegistry registry) {
                registry.addMapping( pathPattern: "/**")
                        .allowedOriginPatterns("*")
                        .allowedMethods("*")
                        .allowedHeaders("*")
                        .allowCredentials(true)
                        .exposedHeaders(HttpHeaders.SET_COOKIE).maxAge(3600L);
            }
        };
    }
}
```