

问题：

- 解决办法:**

- 作为一个项目管理工具，maven有个很大的特色就是类库管理。通过在pom.xml中定义jar包版本和依赖，能够方便的管理jar文件。Maven依赖传递机制会自动加载我们引入的依赖包的依赖。

如：

总结：

搞清楚项目到底依赖哪些包，根据项目所需选择jar包，不必要的、重复的、冲突都拿掉。

maven生命周期阶段与插件

Maven定义了三套生命周期：clean、default、site，每个生命周期都包含了一些阶段（phase）。

1.2 default生命周期部分阶段：

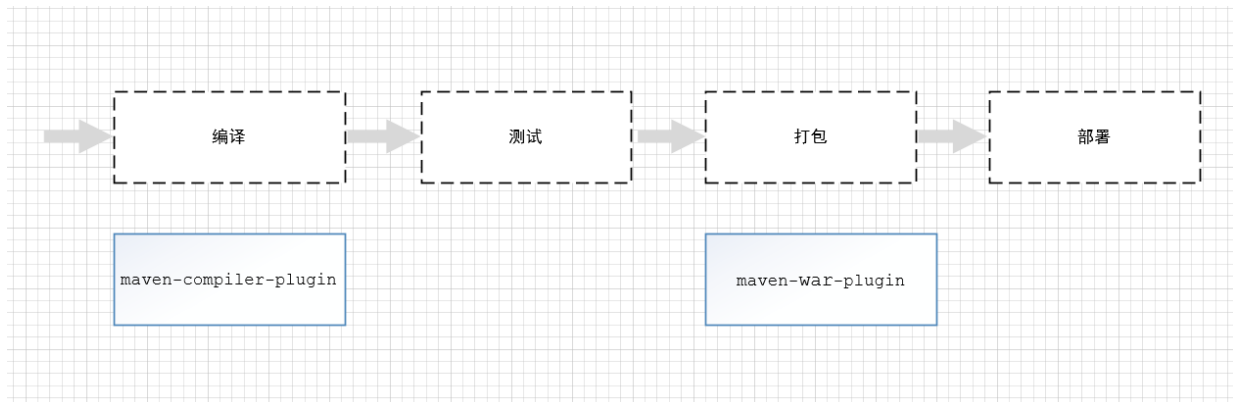


常用的生命周期default如下：

生命周期与插件

Maven 将所有项目的构建过程统一抽象成一套生命周期：项目的清理、初始化、编译、测试、打包、集成测试、验证、部署和站点生成 ... 几乎所有项目的构建，都能映射到这一组生命周期上。但生命周期是抽象的(Maven的生命周期本身是不做任何实际工作)，任务执行(如编译源代码)均交由插件完成。其中每个构建步骤都可以绑定一个或多个插件的目标，而且Maven为大多数构建步骤都编写并绑定了默认插件。

如图：



由于websocket包在编译时需要但是打包时不需要否则数据网关无法正常运行，因为页面和后台进行websocket通信用的是tomcat下的websocket-api.jar，所以需要在打包插件排除jar包，否则会有冲突。导致websocket不可用

```
<dependency>
  <groupId>javax.websocket</groupId>
  <artifactId>javax.websocket-api</artifactId>
  <version>${websocket.version}</version>
  <scope>provided</scope>
</dependency>

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>${java.version}</source>
      <target>${java.version}</target>
    </configuration>
```

```

</plugin>

<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.1.0</version>
  <configuration>
    <packagingExcludes>WEB-INF/lib/javax.websocket-api-1.1.jar</packagingExcludes>
  </configuration>
</plugin>

</plugins>

```

总结：

Maven的核心文件很小，主要的任务都是由插件来完成。通过生命周期、阶段、插件配合才能更好的实现项目的构建

环境问题

日志jar包问题

war包在测试环境上怎么也部署不成功，部署环境用的是tomcat8，但是开发用的是tomcat7。通过查看日志发现，log....stackoverflow了，原来是log4j那边死循环了，所以根本原因项目本身配置的日志jar包不规范，通过slf4j来串联导致存在死循环的风险。错误信息如同官方所说：

Detected both log4j-over-slf4j.jar AND slf4j-log4j12.jar on the class path, preempting StackOverflowError.

The purpose of slf4j-log4j12 module is to delegate or redirect calls made to an SLF4J logger to log4j. The purpose of the log4j-over-slf4j module is to redirect calls made to a log4j logger to SLF4J. If SLF4J is bound withslf4j-log4j12.jar and log4j-over-slf4j.jar is also present on the class path, a StackOverflowError will inevitably occur immediately after the first invocation of an SLF4J or a log4j logger.

Here is how the exception might look like:

```

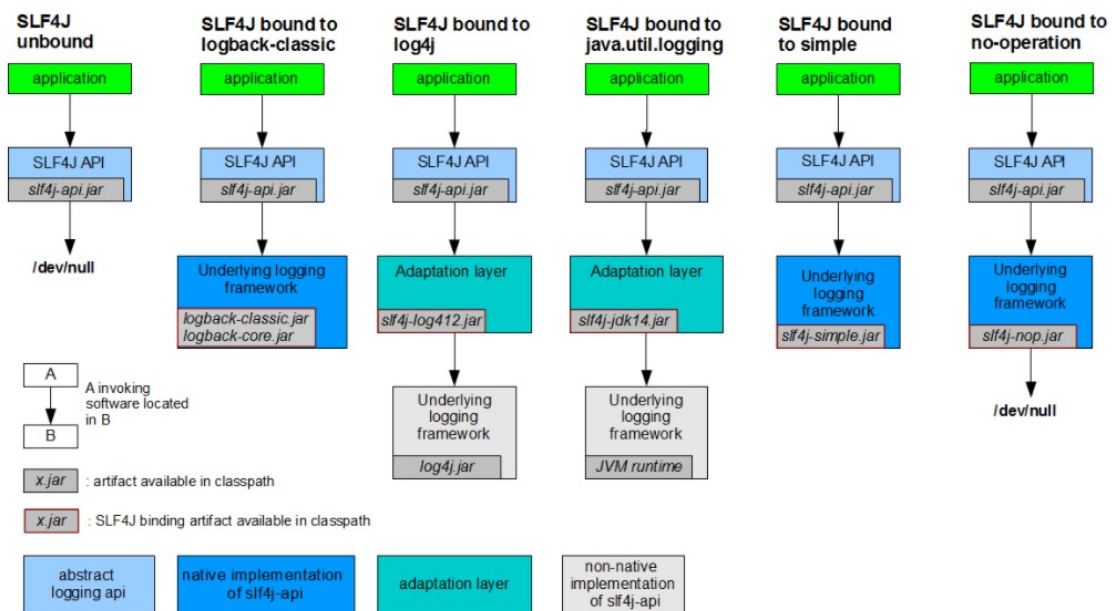
Exception in thread "main" java.lang.StackOverflowError
  at java.util.Hashtable.containsKey(Hashtable.java:306)
  at org.apache.log4j.Log4jLoggerFactory.getLogger(Log4jLoggerFactory.java:36)
  at org.apache.log4j.LogManager.getLogger(LogManager.java:39)
  at org.slf4j.impl.Log4jLoggerFactory.getLogger(Log4jLoggerFactory.java:73)
  at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:249)
  at org.apache.log4j.Category.<init>(Category.java:53)
  at org.apache.log4j.Logger.<init>(Logger.java:35)
  at org.apache.log4j.Log4jLoggerFactory.getLogger(Log4jLoggerFactory.java:39)
  at org.apache.log4j.LogManager.getLogger(LogManager.java:39)
  at org.slf4j.impl.Log4jLoggerFactory.getLogger(Log4jLoggerFactory.java:73)
  at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:249)
  at org.apache.log4j.Category.<init>(Category.java:53)
  at org.apache.log4j.Logger.<init>(Logger.java:35)
  at org.apache.log4j.Log4jLoggerFactory.getLogger(Log4jLoggerFactory.java:39)
  at org.apache.log4j.LogManager.getLogger(LogManager.java:39)
  subsequent lines omitted...

```

SINCE 1.5.13 SLF4J software preempts the inevitable stack overflow error by throwing an exception with details about the actual cause of the problem. This is deemed to be better than leaving the user wondering about the reasons of the StackOverflowError.

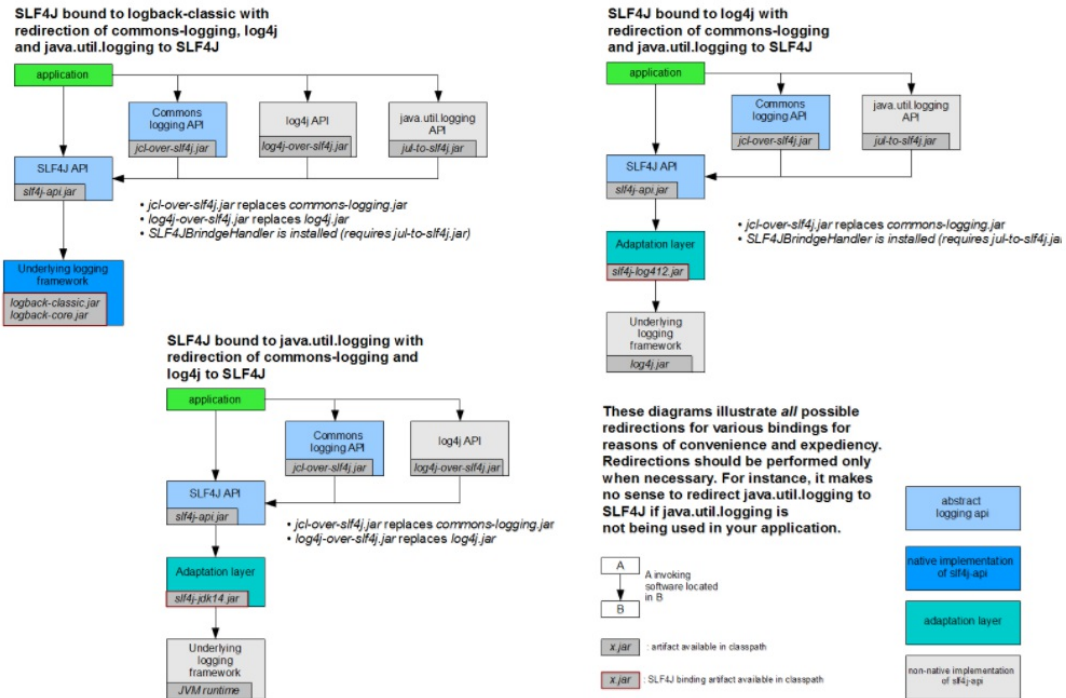
问题原因分析：

首先看官方给出的slf4j与具体日志框架结合的方案：



看第三层中间的两个湖蓝色块，这是适配层，也可以叫桥接器，log4j通过它来实现与slf4j的无缝对接这点很关键。

接着我们再看下官方给出的另一类桥接器：



这类桥接器它们将其它日志框架的API转调回slf4j的API上，上图展示了目前为止能安全地从别的日志框架API转调回slf4j的所有三种情形。看完三种情形以后，会发现几乎所有其他日志框架的API，都能够随意的转调回slf4j。但是有一个唯一的限制就是转调回slf4j的日志框架不能跟slf4j当前桥接到的日志框架相同。这个限制就是为了防止A-to-B.jar跟B-to-A.jar同时出现在类路径中，从而导致A和B一直不停地互相递归调用，最后堆栈溢出。

这就是项目中的错误实践：

```
<!-- 日志 -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>${slf4j.version}</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>${logback.version}</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>log4j-over-slf4j</artifactId>
  <version>${slf4j.version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>jcl-over-slf4j</artifactId>
  <version>${slf4j.version}</version>
</dependency>
```

以上的依赖导致log4j-over-slf4j和slf4j-log4j12两者互相递归调用。但是如果log4j-over-slf4j在slf4j-log4j12之前的时候是可以规避该风险。所以tomcat7下面没有报错，但是因为tomcat8加载资源的方式与tomcat7发生了改变所以导致log4j-over-slf4j在之后加载从而发生了堆栈溢出。不是当时同事说的是因为tomcat8与log4j相关jar冲突。

改为了：

```
<!-- logback -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.1.3</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
```

```
<artifactId>logback-classic</artifactId>
<version>1.1.3</version>
</dependency>
<!--日志 end-->
```

总结：

1. 用maven管理jar包时，需要注意各个jar直接的依赖关系。
2. 当使用了更高版本的的工具的时候，需要知道高版本和低版本的差异

Spring 问题

问题原因分析：

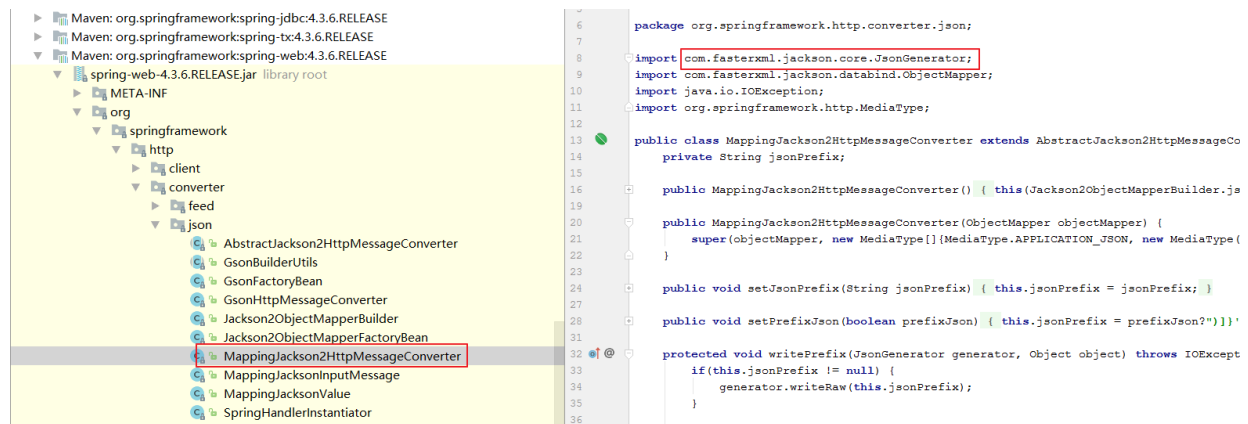
由于同事部署时把spring依赖从spring3修改为spring4导致，项目不能正常运行。页面报错405。405通常

405错误通常意味着，web服务器与处理请求之间发生了问题。

如：

- 1) 请求的方法名称写错
- 2) 请求方法参数类型与标准不一致
- 3) 请求方法异常、返回值类型与标准不一致

通过下图查看源码得知，spring 4.x已经删掉了Spring3.x时MappingJackson2HttpMessageConverter等json转换的类，所以需要更高版本的jackson



上网查询得知Spring4.x，json转换需要添加jackson-databind、jackson-core两个jar包，并且在servlet分发器中需要添加

```
<!--json转换器-->
<mvc:annotation-driven>
    <mvc:message-converters>
        <bean class="org.springframework.http.converter.StringHttpMessageConverter"/>
        <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"/>
    </mvc:message-converters>
</mvc:annotation-driven>
```

总结：

1. 升级jar包时，需要注意高版本和低版本的区别。要了解高版本高在哪儿，不盲目升级。

总结

有问题不怕丢人，不要怕有问题，不管这个问题是谁导致的都去积极解决，不逃避问题，要学会发现问题，并解决问题，同时还需要了解问题发生的根本原因。学会追根究底。这样才会成长。