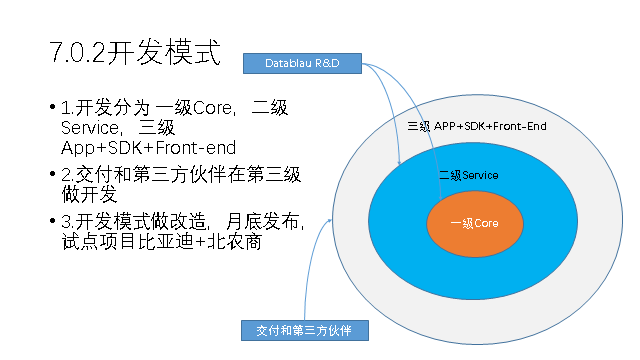
# 7.X 微服务SDK开发模式

按照公司最新的规划，在7.X之后代码会进一步隔离分层：



按照此指导思想，产品的代码会有所改变来适应新的需求，总体思路是：

1. 产品本身功能逻辑打包成一个完成的jar包，发布到nexus仓库中，包含完整的MVC几层，发布artificat id类似 metadata-backend-lib。
2. 产品默认发布会有一个独立的spring boot 的module，此module只包含spring boot的main函数，作为产品默认发版配置。
3. 交付和第三方伙伴在独立的仓库开发，通过nexus仓库依赖meatadata-backend-lib作为依赖，然后所有定开的代码，包括新的Controller、Service、DAO等等。 所有的metadata-backend-lib的service、component都可以继承，所有的Bean都可以被覆盖。

## 修改7.0.2Usermanagement样例步骤：

### 调整代码结构

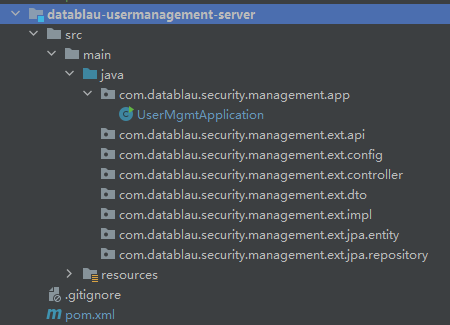
1. 打开Usermanagement仓库，把原来的datablau-usermanagement-server名称改为datablau-usermanagement-backend。
2. 创建新的datablau-usermanagement-server的模块，把datablau-usermangement-backend的pom.xml, .gitignore, src/main/resources目录都拷贝到此模块下。
3. 修改datablau-usermangement-backend的pom.xml,
   1. 把artifact id修改为datablau-usermangement-backend-lib,
   2. 以及name修改为Datablau User Management Backend Library。
   3. Version修改为7.0.2-SNAPSHOT
   4. 修改<build>块，把build块内容替换为以下内容

<build>  
 <resources>  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.csv</include>  
 <include>\*\*/\*.xml</include>  
 <include>\*\*/\*.properties</include>  
 </includes>  
 <filtering>true</filtering>  
 </resource>  
 <resource>  
 <directory>src/main/resources</directory>  
 <includes>  
 <include>\*\*/\*.csv</include>  
 </includes>  
 <filtering>true</filtering>  
 </resource>  
 </resources>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

1. 从datablau-usermangement-backend-lib模块剪切启动程序UserMgmtApplication.java 到datablau-usermangement-server项目，并且修改pom.xml删除所有dependencies，以及修改<build>内容如下所示：

<dependencies>  
 <dependency>  
 <groupId>com.datablau</groupId>  
 <artifactId>datablau-usermanagement-backend-lib</artifactId>  
 <version>7.0.2-SNAPSHOT</version>  
 </dependency>  
</dependencies>  
  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <includeSystemScope>true</includeSystemScope>  
 </configuration>  
 <executions>  
 <execution>  
 <goals>  
 <goal>repackage</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-war-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>  
  
<profiles>  
 <profile>  
 <dependencies>  
 <dependency>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 <groupId>org.springframework.boot</groupId>  
 </dependency>  
 </dependencies>  
 <activation>  
 <activeByDefault>true</activeByDefault>  
 </activation>  
 <id>tom</id>  
 </profile>  
 <!--宝兰德 Tomcat 适配 -->  
 <profile>  
 <dependencies>  
 <dependency>  
 <artifactId>bes-lite-spring-boot-2.x-starter</artifactId>  
 <groupId>com.bes.appserv-web</groupId>  
 <version>9.5.2.008</version>  
 </dependency>  
 </dependencies>  
 <id>bes</id>  
 </profile>  
 <!--适配tongweb-->  
 <profile>  
 <dependencies>  
 <dependency>  
 <groupId>com.tongweb.springboot</groupId>  
 <artifactId>tongweb-spring-boot-starter-2.x</artifactId>  
 <version>7.0.E.5\_P2</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.tongweb</groupId>  
 <artifactId>tongweb-embed</artifactId>  
 <version>7.0.E.5\_P2</version>  
 </dependency>  
 </dependencies>  
 <id>tongweb</id>  
 </profile>  
</profiles>

1. 在工程里面创建一些空的package，方便交付或者其它二开在里面继承和增加新的微服务，在SpringApplication和JpaConfig中添加扫描目录，拿Usermanagement为例，目录结构如下所示：



1. 至此，最基本的代码架构调整已经完成，我们可以尝试启动微服务，看看微服务是否能正常编译和启动。

### 修改代码内容：

完成上述修改之后，我们产品的代码已经是通过nexus发布jar包，交付可以通过复用我们的datablau-usermanagement-server程序来编译他们自己的微服务。

接下来我们需要让交付可以扩展甚至修改我们产品的逻辑。

总体来说我们的策略是：

1. 所有的已有的service都可以通过“继承”来扩展的
2. 所有的bean都需要优先使用二开的bean。 比如产品的UserServiceImpl实现了UserService接口，为了保证产品在二开的时候交付可以改变UserServiceImpl的某个函数的实现逻辑，交付可以定义一个新的类叫UserServiceImplExt，override目标方法，并且系统在创建Bean的时候，只会构造UserServiceImplExt的Bean而不会构造UserServiceImpl。
3. 所有持久层的代码不允许改变，二开需要重新定义自己的Entity和Repository，通过业务主键关联已有的Entity。

为了实现以上目标，我们需要对代码进行如下修改：

1. 所有的服务实现类和他们的方法都需要可继承和可重载，所有的private的函数或者private的class，内部类，都要变成protected或者public。
2. 所有的Bean上面都需要增加一个注解@ConditionalOnMissingBean(name=”xxx”)。
3. 所有的Bean都需要显式命名，比如@Service的都需要给出名称，比如@Service(“userService”)。
4. 所有扩展Bean的名称都必须叫 XXXExt, 比如上述UserService, 那么扩展的Bean的名称需要为UserServiceExt, 所以UserServiceImpl的定义为：

@Service(“userService”)

@ConditionalOnMissingBean(name=“userServiceExt”)

### 测试修改内容：

当上述完成之后需要确保改动的有效性：

1. 我们需要启动我们的微服务，看看是否所有的Bean都可以正常创建，不会存在缺失Bean的问题。
2. 在包含main函数的项目中，创建一些测试Bean，覆盖基类，检查Bean的创建是否如预期。

### 发布到项目工程：

现在git开辟了两个新的仓库：

DAM的仓库为：

<http://192.168.1.34/datablauex/datablau_dam_projects>

DDM的仓库为:

<http://192.168.1.34/datablauex/datablau_ddm_projects>

每个仓库包含了此产品线所需的所有微服务和前端代码

目录结构如下：

backend

|- pom.xml

|- datablau-usermangement-server

|- datablau-metadata-server

|- datablau-dataquality-server

|- ……

……

frontend

|- datablau-web-ddc

|- datablau-ui

|-……

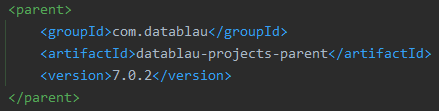
……

1. 每个项目将会是此仓库的一个分支
2. 每个分支包含这个项目的前后端代码
3. 产研会针对每个release版本提交一个分支

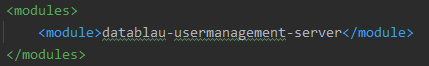
对于后端来说，开发只要把原来在产研分支上面的spring boot jar拷贝到此目录下即可。

依旧拿datablau-usermangement-server举例

1. 把这个module的代码完整拷贝到backend目录之下
2. 然后修改pom.xml的parent为datablau-projects-parent, 如下所示：



1. 在backend目录下的pom.xml内增加module datablau-usermanagement-server



启动datablau-management-server, 检查是否能正常工作，如果能正常工作，那么迁移工作就完成。

以上代码都已经提交：

1. Datablau-usermanagement在仓库 <http://192.168.1.34/datablauex/datablau-usermgmt>， 分支为rd-7.0.2.1

二开代码在仓库 <http://192.168.1.34/datablauex/datablau_ddm_projects> 和 <http://192.168.1.34/datablauex/datablau_dam_projects>