Java后台项目开发流程

一般来说，Java后台项目开发流程包括工程搭建、代码结构组织、编码、单元测试、问题修复、集成测试、预发布、发布等几个典型过程。

看下面这个示意图：

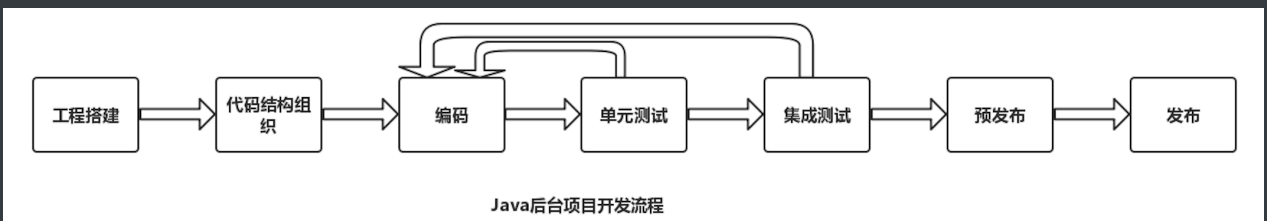


图1 Java后台项目开发流程图

下面说明每个流程的主要任务：

工程搭建：该步主要是创建一个满足应用类型的工程主体骨架，这一步需要确定项目的类型、依赖包等。例如，可以搭建使用Maven管理工具、基本Spring Boot框架，这些都需要在工程创建之初首先确定。此外，是否是一个Web项目，是否需要对外发布成组件，这些都是需要考虑的因素。

代码结构组织：该步主要是对工程代码的结构进行组织，包括模块、逻辑单元等层次划分。代码的组织结构能大体反映出项目的基本架构设计。对于Java后台项目而言，一般会有某种通用的划分原则，这就是按职责分层设计，大致可分为：数据访问层（DAL），服务层（Service），业务逻辑层（Business），应用层(Application)，实际应用中服务层与业务逻辑层可以合二为一，简化为三层，每一层对应的对象分别以xxxDAO，xxxService，xxxController命名。

编码：这步就是应用程序代码的设计与编写过程，关于基本原则可参考第一讲中关于代码设计原则及最佳实践部分。在编码之初就要考虑代码的内聚性及与其它模块的交互性，模块之间交互要尽可能少，并且交互接口要提前约定好。代码设计既要满足现有功能，又要有一定预见性，方便功能扩展。编码过程应该分层编写代码，每层单独进行测试。

单元测试：单元测试是指对代码的每个最小可测试单元进行测试，例如一个类/实例方法，通过使用一些单元测试工具如JUnit，可以方便进行这项工作。由于代码之间的往往存在依赖关系，因此在开发中，也应尽可能从底层代码开始编码，以便上层代码的编写与测试不受影响。对于因各种原因无法事先得到所依赖的代码的情况，也有相应的解决方案，那就是Mock测试，即可以伪造一个跟真实代码功能一样的对象，使上层代码的编写和测试不受限于底层代码的开发进度。单元测试的目的是使代码各部分都是可被测试的、可被反复测试、可被随时测试，以便错误及早被发现，由此保证代码各部分质量。

集成测试：集成测试顾名思义是同时对各代码单元，有时会包含多个系统进行测试，集成测试一般面向的是功能测试、性能评测，可以由人工操作来完成，也可以由自动化工具模拟用户行为来完成。集成测试是开发人员和测试人员都必须参与的工作，开发人员将集成测试通过后的代码提交给测试人员进行测试，这种测试性交付是开发人员和测试人员协同工作的纽带。

预发布：将开发环境测试通过后的代码发布到预发布环境，采用与线上环境一致的数据对代码各项功能进行验证来检查是否符合预期，这个过程预发布环境测试。预发布是代码上线前最后一项质量保证措施，需要开发人员和测试都参与确认。

发布：将代码发布到生产环境，实现最终交付。一般采用灰度发布的方案，先局部后全量，保证平稳上线。上线完成后需要开发人员和测试共同参与验证。

下面通过一个实际项目讲解Java后台项目的开发过程。

软件环境

- IDE：Eclipse
- 工程管理工具：Maven
- 开发框架：Spring Boot
- 数据库：MySQL
- 缓存：Redis

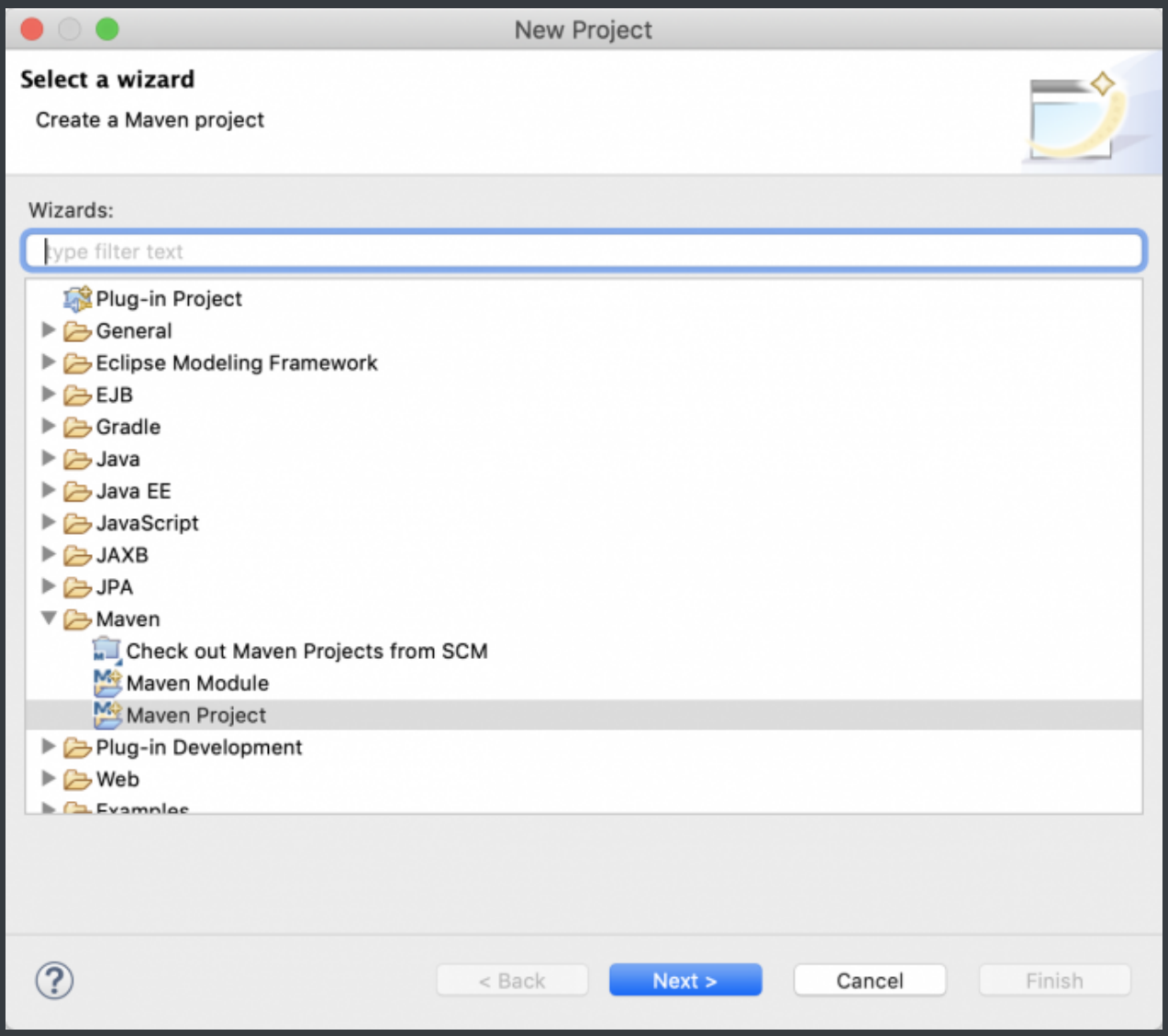
项目功能

实现用户信息管理HTTP API，包括注册、登录两个功能接口。

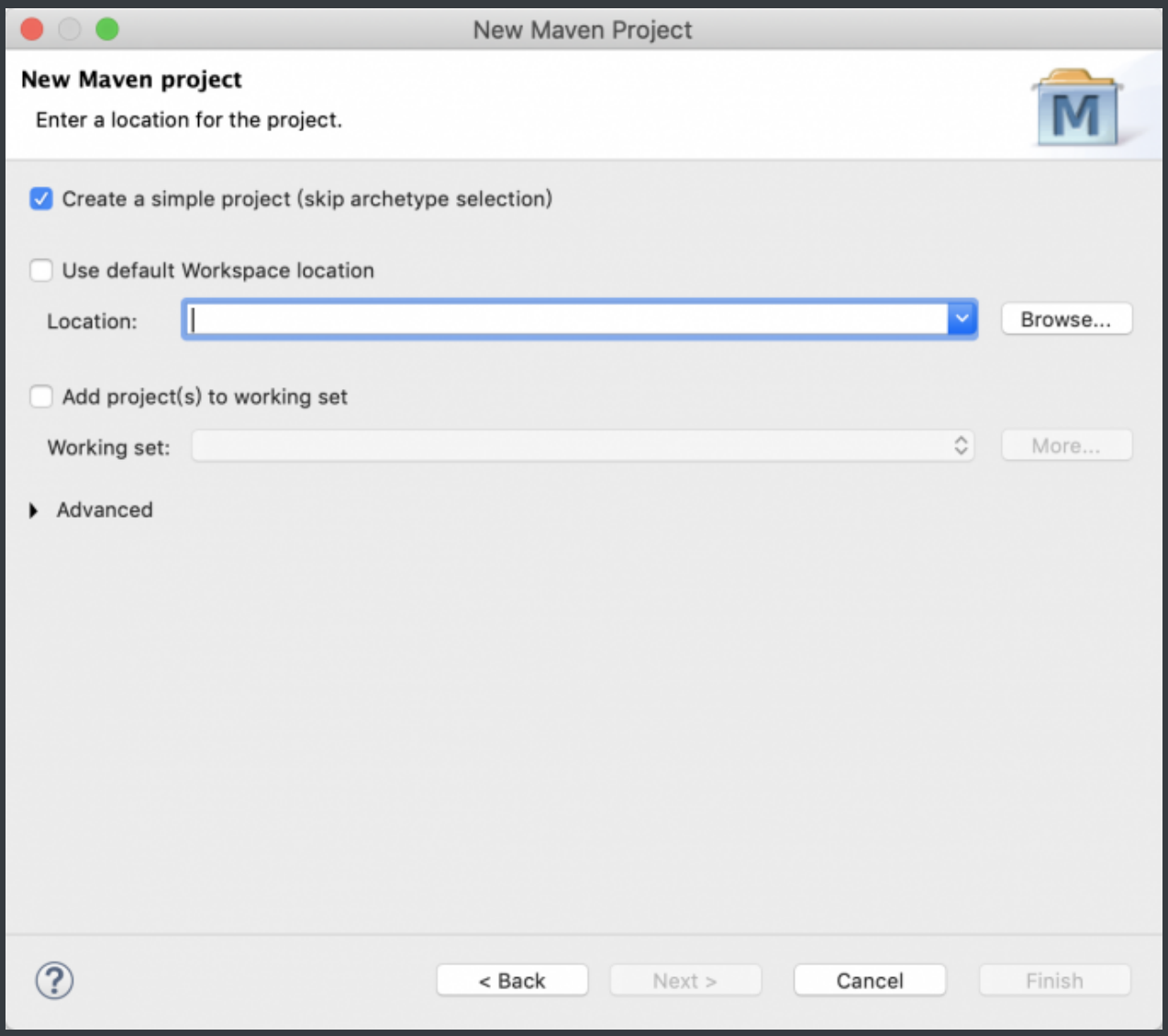
1. 项目搭建

在Eclipse中创建一个基于Maven的项目，取名为course3-demo，具体方法是

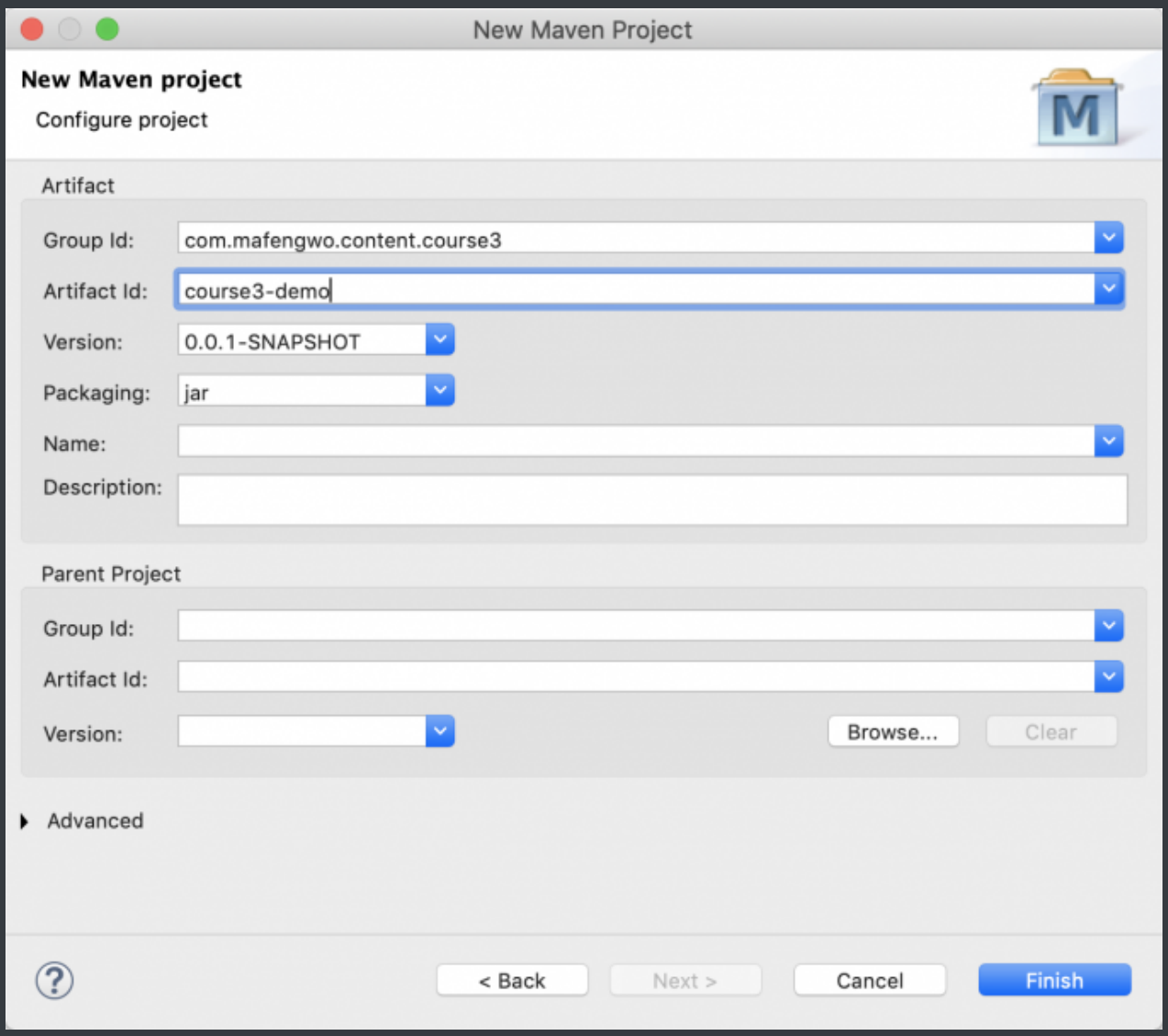
File -> New -> Project…，在对话框中选择Maven -> Maven Project，单击Next



在弹出的页面，勾选Create a simple project复选框，并选择Location，然后单击下一步



在下图中的界面，输入GroupId和ArtifactId名称，单击Finish，完成工程结构创建。



2. 引入项目依赖

完成工程创建后，需要根据实际项目类型，引入依赖包，这里我们是创建一个基于Spring Boot的工程，并引入一些需要的依赖包，如MyBatis，Jedis，而这只需要在pom.xml加入以下配置即可：

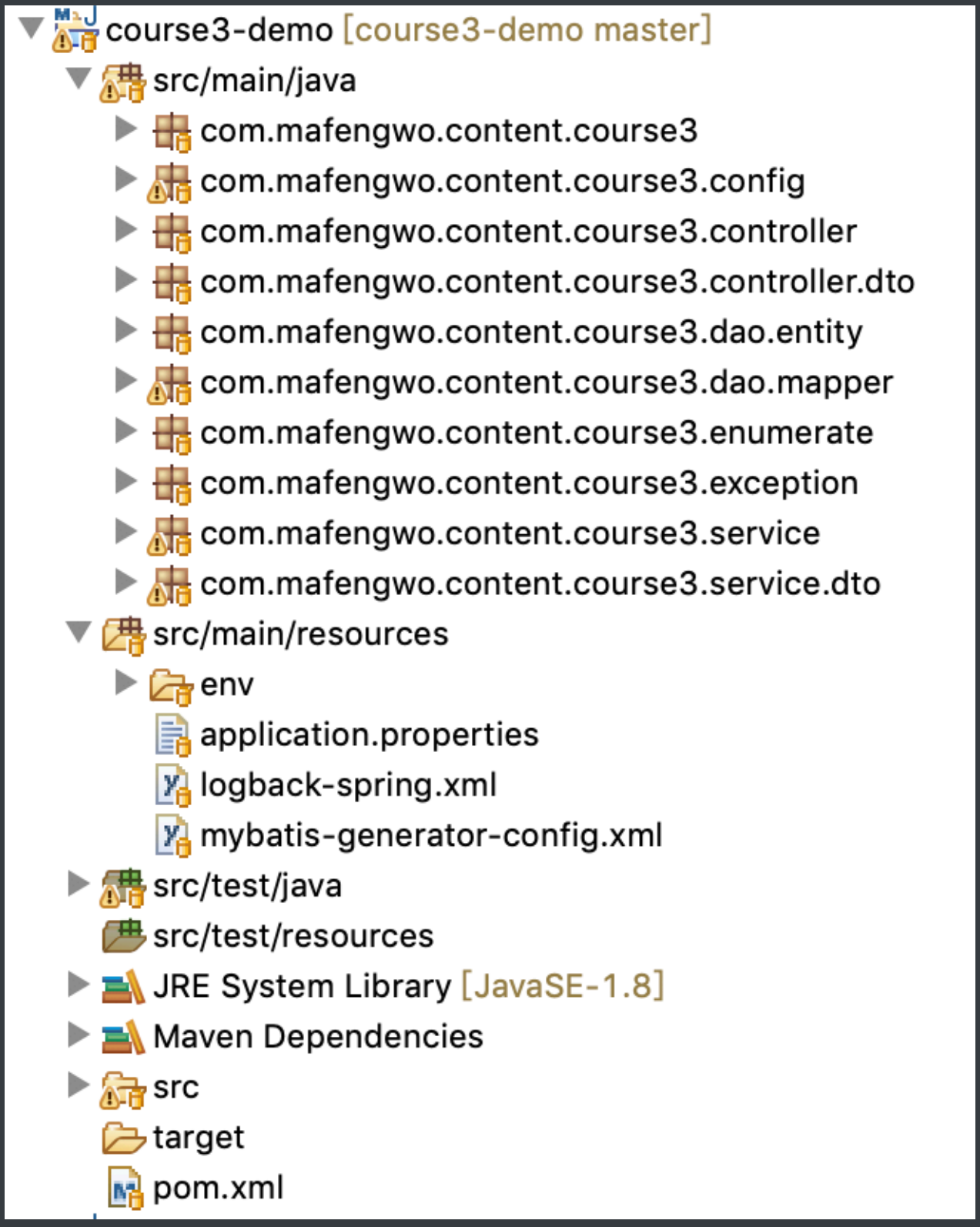
```
    <project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.mafengwo.content.course3</groupId>
        <artifactId>course3-demo</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <!--继承一个父POM，自动使用其引入的依赖包-->
        <parent>
            <groupId>com.mafengwo.content.framework</groupId>
```

```
        <artifactId>springboot-skeleton</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <dependencies>
        <!--依赖一个工具包-->
        <dependency>
            <groupId>com.mafengwo.content.component</groupId>
            <artifactId>scooter</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

    </dependencies>

    <build>
        <finalName>${project.artifactId}</finalName>
        <filters>
            <filter>${env.dir.name}.properties</filter>
        </filters>
    </build>

    <profiles>
        <profile>
            <id>dev</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>

<env.dir.name>${basedir}/src/main/resources/env/dev</env.dir.name>
            </properties>
            <build>
                <resources>
                    <resource>

<directory>${basedir}/src/main/resources</directory>
                        <filtering>false</filtering>
                        <excludes>
                            <exclude>env/**</exclude>
                        </excludes>
                    </resource>
                    <resource>

<directory>${basedir}/src/main/resources</directory>
                        <filtering>true</filtering>
```

```
                        <includes>
                            <include>logback-spring.xml</include>
                        </includes>
                    </resource>
                </resources>
            </build>
        </profile>
        <profile>
            <id>test</id>
            <properties>

<env.dir.name>${basedir}/src/main/resources/env/test</env.dir.name>
            </properties>
            <build>
                <resources>
                    <resource>

<directory>${basedir}/src/main/resources</directory>
                        <filtering>false</filtering>
                        <excludes>
                            <exclude>env/**</exclude>
                        </excludes>
                    </resource>
                    <resource>

<directory>${basedir}/src/main/resources</directory>
                        <filtering>true</filtering>
                        <includes>
                            <include>logback-spring.xml</include>
                        </includes>
                    </resource>
                </resources>
            </build>
        </profile>
    </profiles>


</project>
```

3. 工程代码结构

按照分层设计的原则，将工程按以下包结构进行组织，



3. 编码

代码结构组织划分完成，就可以分层编写代码了，可以采取由下层到上层的编写顺序，依次实现DAO、Service和Controller。

首先创建一个应用启动入口类，最上层包目录添加一个SpringBoot启动类：

```
package com.mafengwo.content.course3;
```

```
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.mafengwo.content.component.scooter.DateUtils;


@SpringBootApplication
@MapperScan(basePackages = "com.mafengwo.content.course3.dao.mapper")
public class Course3ApplicationBootStrap {

    public static void main(String[] args) {
        SpringApplication.run(Course3ApplicationBootStrap.class, args);
        System.out.println("course3 demo started at
["+DateUtils.formatToStandardTimeStringNow()+"]");
    }

}
```

3.1 创建一个数据库表

```
CREATE TABLE `t_arch_user_base_info` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT '自增主键',
  `user_id` bigint(20) NOT NULL COMMENT '用户唯一标识',
  `user_name_type` int(11) NOT NULL COMMENT '用户名类型',
  `user_name` varchar(64) NOT NULL COMMENT '用户唯一名称',
  `nick_name` varchar(20) NOT NULL COMMENT '昵称',
  `gender` int(11) NOT NULL COMMENT '性别',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `last_modify_time` datetime NOT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_user_id` (`user_id`) USING BTREE,
  UNIQUE KEY `idx_user_name` (`user_name`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='user_base_info';
```

3.2 DAO层代码编写与单元测试

DAO层代码生成方法可参考第二讲关于代码自动生成工具的篇节，这里不再说明。完成后在com.mafengwo.content.course3.dao包下将会看到DO和Mapper对象。

代码生成工具能生成最基本的数据库操作，如果不满足要求，可以通过在DAO对象中（Mapper）自定义一些方法来实现相应功能，以下就是一个自定义的查询方法

```
    @Select({
        "select",
        "id, user_id, user_name_type, user_name, nick_name, gender,
create_time, last_modify_time",
        "from t_arch_user_base_info",
        "where user_name = #{userName,jdbcType=VARCHAR}"
    })
    @ResultMap("userBaseInfoResultMap")
    UserBaseInfoDO selectByUserName(String userName);
```

编写完DAO方法，下面就立即可以进行单元测试了，具体方法参考第二讲。

3.3 Service层代码编写与单元测试

首先简单介绍一下Service层，Service层的定位是一组逻辑功能单元接口的集合，一般按业务场景来组织，比如一个用户服务，可以包含登录、注册等功能接口。Service层的代码分为Service接口和Service实现。下面编写一个UserService和UserServiceImpl类。

```
package com.mafengwo.content.course3.service;


import com.mafengwo.content.course3.exception.Course3DemoBizException;
import com.mafengwo.content.course3.service.dto.UserBaseInfoDTO;

public interface UserService {

    boolean add(UserBaseInfoDTO userDTO) throws Course3DemoBizException;
    UserBaseInfoDTO queryByUserName(String userName) throws
Course3DemoBizException;

}
```

UserService接口定义了三个业务功能接口，分别对应添加用户、按用户名查询用户信息。其具体实现类为UserServiceImpl，代码如下：

```java
package com.mafengwo.content.course3.service;

import java.util.Date;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mafengwo.content.component.scooter.DateUtils;
import com.mafengwo.content.component.scooter.IDGenerator;
import com.mafengwo.content.course3.dao.entity.UserBaseInfoDO;
import com.mafengwo.content.course3.dao.mapper.UserBaseInfoDOMapper;
import com.mafengwo.content.course3.exception.Course3DemoBizException;
import com.mafengwo.content.course3.service.dto.UserBaseInfoDTO;

@Service
public class UserServiceImpl implements UserService {

    private Logger logger = LoggerFactory.getLogger(UserServiceImpl.class);

    @Autowired
    private UserBaseInfoDOMapper userBaseInfoDOMapper;


    @Override
    public boolean add(UserBaseInfoDTO userDTO) throws
Course3DemoBizException {
        UserBaseInfoDO userBaseInfoDO = new UserBaseInfoDO();
        userBaseInfoDO.setCreateTime(new Date());
        userBaseInfoDO.setGender(userDTO.getGender());
        userBaseInfoDO.setLastModifyTime(new Date());
        userBaseInfoDO.setNickName(userDTO.getNickName());
        userBaseInfoDO.setUserId(IDGenerator.next());
        userBaseInfoDO.setUserName(userDTO.getUserName());
        userBaseInfoDO.setUserNameType(userDTO.getUserNameType());
        return userBaseInfoDOMapper.insert(userBaseInfoDO) == 1;
    }

    private UserBaseInfoDTO toUserBaseInfoDTO(UserBaseInfoDO
userBaseInfoDO) {
        UserBaseInfoDTO userBaseInfoDTO = new UserBaseInfoDTO();
```

```
 userBaseInfoDTO.setCreateTime(DateUtils.formatToStandardTimeString(userBas
eInfoDO.getCreateTime()));
        userBaseInfoDTO.setGender(userBaseInfoDO.getGender());

 userBaseInfoDTO.setLastModifyTime(DateUtils.formatToStandardTimeString(use
rBaseInfoDO.getLastModifyTime()));
        userBaseInfoDTO.setNickName(userBaseInfoDO.getNickName());
        userBaseInfoDTO.setUserId(userBaseInfoDO.getUserId());
        userBaseInfoDTO.setUserName(userBaseInfoDO.getUserName());
        userBaseInfoDTO.setUserNameType(userBaseInfoDO.getUserNameType());
        return userBaseInfoDTO;
    }

    @Override
    public UserBaseInfoDTO queryByUserName(String userName) throws
Course3DemoBizException {
        UserBaseInfoDO userBaseInfoDO =
userBaseInfoDOMapper.selectByUserName(userName);
        if(userBaseInfoDO == null){
            return null;
        }
        return toUserBaseInfoDTO(userBaseInfoDO);
    }

}
```

完成Service层的代码后，也要单元测试，以保证代码正确性。测试过程不再表述。

3.4 Controller层代码编写与单元测试

上面已经实现DAO层和Service层的代码并进行了单元测试，如果一切正常，就可以编写Controller层代码了，Controller层负责HTTP API的请求/响应交互处理。

利用Spring MVC框架的相关组件很容易实现HTTP API接口，用户注册、登录的Controller代码如下：

```
package com.mafengwo.content.course3.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.mafengwo.content.course3.controller.dto.LoginResponse;
import com.mafengwo.content.course3.controller.dto.LoginResponseResult;
import com.mafengwo.content.course3.controller.dto.SimpleOkayResponse;
import com.mafengwo.content.course3.controller.dto.UserRegisterRequest;
import com.mafengwo.content.course3.enumerate.GenderEnum;
import com.mafengwo.content.course3.enumerate.ResultCodeEnum;
import com.mafengwo.content.course3.exception.Course3DemoBizException;
import com.mafengwo.content.course3.service.UserService;
import com.mafengwo.content.course3.service.dto.UserBaseInfoDTO;


@RestController
@RequestMapping("/user")
public class UserController extends BaseController {

    private Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserService userService;

    /**
     * 注册
     * @throws Exception
     */
    @PostMapping(value = "/register")
    public SimpleOkayResponse register(HttpServletRequest request,
HttpServletResponse response,
            @RequestBody UserRegisterRequest userRegisterRequest,
            @RequestParam(value = "registerToken", required = true) String
registerToken,
            @RequestHeader(value = "appType", required = true) Integer
appType,
            @RequestHeader(value = "appVersion", required = true) Integer
appVersion) throws Exception {

        logger.debug("userRegisterRequest:{},appType:{},appVersion:
{}",userRegisterRequest,appType,appVersion);
```

```
        checkGender(userRegisterRequest.getGender());

        UserBaseInfoDTO userDTO = new UserBaseInfoDTO();
        userDTO.setGender(userRegisterRequest.getGender());
        userDTO.setNickName(userRegisterRequest.getNickName());
        userDTO.setUserName(userRegisterRequest.getUserName());
        userDTO.setUserNameType(userRegisterRequest.getUserNameType());

        boolean result = userService.add(userDTO);
        logger.debug("register status:{}",result);
        return new SimpleOkayResponse();

    }

    private void checkGender(Integer gender) throws Course3DemoBizException
{
        if(gender == null || GenderEnum.valueOf(gender) == null) {
            throw new
Course3DemoBizException(ResultCodeEnum.IllegalRequestArgument);
        }
    }

    /**
     * 登录
     * @throws Exception
     */
    @PostMapping(value = "/login")
    public LoginResponse login(HttpServletRequest request,
HttpServletResponse response,
            @RequestParam(value = "userName", required = true) String
userName,
            @RequestHeader(value = "appType", required = true) Integer
appType,
            @RequestHeader(value = "appVersion", required = true) Integer
appVersion) throws Exception {

        logger.debug("userName:{},appType:{},appVersion:
{}",userName,appType,appVersion);
        UserBaseInfoDTO userBaseInfoDTO =
userService.queryByUserName(userName);
        LoginResponse loginResponse = new LoginResponse();
        LoginResponseResult loginResponseResult = new
LoginResponseResult();
        if( userBaseInfoDTO != null ) {
```

```
            loginResponseResult.setNickName(userBaseInfoDTO.getNickName());
            loginResponseResult.setUserId(userBaseInfoDTO.getUserId());
            loginResponseResult.setUserName(userBaseInfoDTO.getUserName());

     loginResponseResult.setRegisterTime(userBaseInfoDTO.getCreateTime());
        }
        else {
            throw new Course3DemoBizException(ResultCodeEnum.UserNotExist);
        }
        loginResponse.setResData(loginResponseResult);
        logger.debug("resData:{}",loginResponse);
        return loginResponse;

    }


}
```

单元测试代码如下：

```
package com.mafengwo.content.course3.controller;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMv
c;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

@RunWith(SpringRunner.class)
@SpringBootTest({"spring.profiles.active=dev"})
@AutoConfigureMockMvc
public class UserControllerTest {

    @Autowired
    protected MockMvc mockMvc;
```

```
    @Test
    public void testLogin() throws Exception {

        MvcResult mvcResult =
mockMvc.perform(MockMvcRequestBuilders.post("/user/login.do")
                .param("userName", "arch003@gmail.com")
                .header("appType", "1")
                .header("appVersion", "101")
                .contentType(MediaType.APPLICATION_JSON)
                .content("")).andReturn();// 返回执行请求的结果

        int status = mvcResult.getResponse().getStatus();
        String content = mvcResult.getResponse().getContentAsString();


        System.out.println("HTTP Response Status Code:"+status);
        System.out.println("HTTP Response Body Data:"+content);

    }

    @Test
    public void testRegister() throws Exception {

        MvcResult mvcResult =
mockMvc.perform(MockMvcRequestBuilders.post("/user/register.do")
                .param("registerToken", "7747597357434")
                .header("appType", "1")
                .header("appVersion", "101")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"userName\":\"pm-002\",\"nickName\":\"pm-nn-
002\",\"gender\":1,\"userNameType\":32}")).andReturn();// 返回执行请求的结果

        int status = mvcResult.getResponse().getStatus();
        String content = mvcResult.getResponse().getContentAsString();


        System.out.println("HTTP Response Status Code:"+status);
        System.out.println("HTTP Response Body Data:"+content);

    }


}
```

完成Controller层代码之后，一个完整的Web 应用就可以运行起来对外提供服务了。

4 发布

项目所有代码开发并测试完成以后，就可以发布到测试环境进行测试了，利用 Maven的打包插件及Spring Boot集成打包插件，只需执行一个Maven命令即可实现项目打包，再结合部署系统，可实现一键发布。

关于Maven打包配置片段如下：

```
        ......
        <profile>
              <id>dev</id>
              <activation>
                  <activeByDefault>true</activeByDefault>
              </activation>
              <properties>

    <env.dir.name>${basedir}/src/main/resources/env/dev</env.dir.name>
              </properties>
              <build>
                  <resources>
                      <resource>

    <directory>${basedir}/src/main/resources</directory>
                          <filtering>false</filtering>
                          <excludes>
                              <exclude>env/**</exclude>
                          </excludes>
                      </resource>
                      <resource>

    <directory>${basedir}/src/main/resources</directory>
                          <filtering>true</filtering>
                          <includes>
                              <include>logback-spring.xml</include>
                          </includes>
                      </resource>
                  </resources>
              </build>
          </profile>
          <profile>
```

```
            <id>test</id>
            <properties>

    <env.dir.name>${basedir}/src/main/resources/env/test</env.dir.name>
            </properties>
            <build>
                <resources>
                    <resource>

    <directory>${basedir}/src/main/resources</directory>
                        <filtering>false</filtering>
                        <excludes>
                            <exclude>env/**</exclude>
                        </excludes>
                    </resource>
                    <resource>

    <directory>${basedir}/src/main/resources</directory>
                        <filtering>true</filtering>
                        <includes>
                            <include>logback-spring.xml</include>
                        </includes>
                    </resource>
                </resources>
            </build>
        </profile>
      ......
```

环境资源配置文件：

src/main/resources/env/dev.properties

```
demo.logbacklog.dir=/data/logs/course3-demo
demo.logbacklog.level=DEBUG
```

src/main/resources/env/test.properties

```
demo.logbacklog.dir=/data/logs/course3-demo
demo.logbacklog.level=DEBUG
```

logback-spring.xml配置片段：

```
        <property name="log_dir" value="@demo.logbacklog.dir@/" />
        <property name="logLevel" value="@demo.logbacklog.level@/" />
```

上面配置表示在不同的部署环境，应用不同的环境变量，配置完成后即可通过Maven命令行使用-P参数指定相应环境资源配置文件，如mvn -Ptest ...来对测试环境进行打包。

进入发布系统新增应用发布：

一键发布：

关于发布系统具体详细，可参见https://wiki.mafengwo.cn/pages/viewpage.action?pageId=33425455。