一、SpringBoot简介

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化老的Spring 应用开发。该框架使用了自动方式来对开发工程进行配置，减少开发人员定义配置复杂度。

1.1 设计初衷

- 为Spring开发者提供一种，更快速、体验更好的Spring应用开发方式。
- 开箱即用，同时也可快速扩展，嵌入式的Tomcat。
- 绝对没有冗余代码，无需XML配置。

1.2 核心功能

- 核心能力：Spring容器、日志、自动配置AutoCongfiguration、Starters
- web应用的能力：MVC、嵌入式容器
- 数据访问(持久化)：关系型数据库、非关系型数据库
- 强大的整合其他技术的能力
- 测试：强悍的应用测试

怎么定位SpringBoot在开发中的地位？

1. **农业时代Java开发方式**：

- 基于Java底层原生API，纯手动去实现，典型技术Html、JavaScript、CSS，JDBC，DBUtils，Socket.....
- 框架是拯救者，解放了农业时代的程序猿们，框架为我们做的更多

2. **工业时代Java开发方式**：

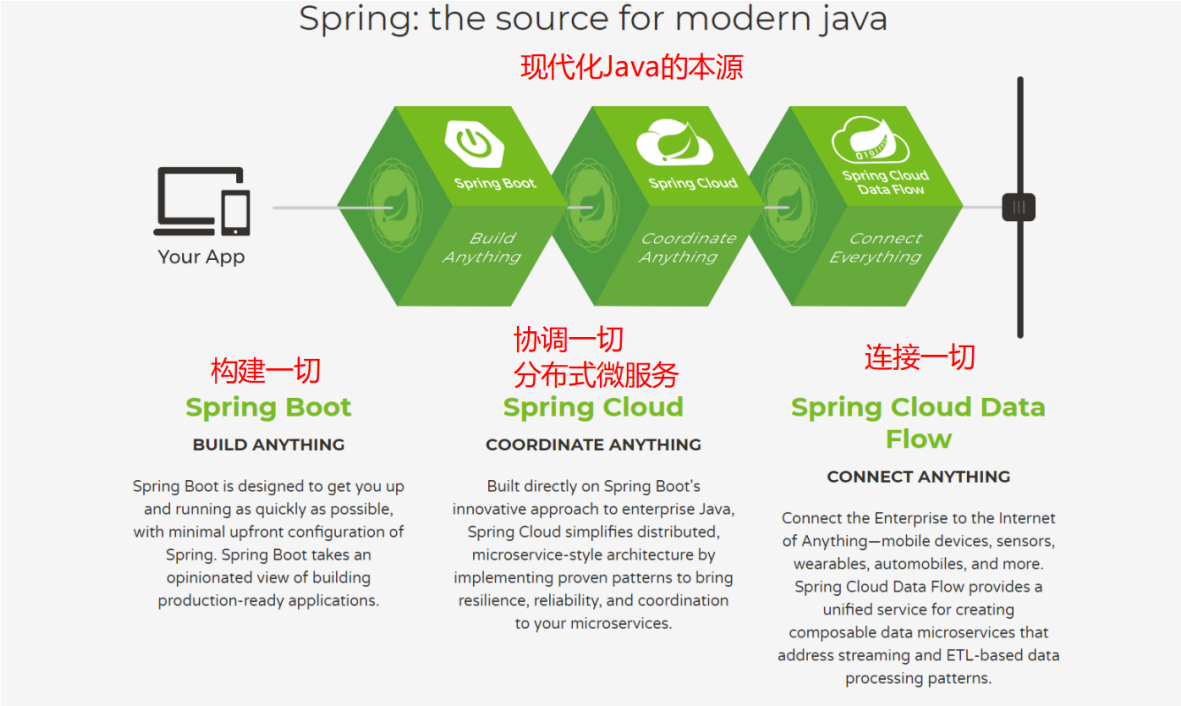
- 各种框架一顿搞：典型代表Spring，SpringMVC，Mybatis，Hibernate，Struts，Freemaker，JBPM...
    - 微服务是拯救者，解放了工业时代的程序猿们，微服务让我们过上了小康生活

3. **现代化Java开发方式**：

- SpringBoot整合并简化一切Spring应用开发中的技术
- 各种SpringCloud微服务：服务注册与发现，负载均衡与熔断，网关和集群
- 想要学习SpringCloud的整套微服务架构系统，必先学习SpringBoot，它是SpringCloud的基础。SpringCloud项目都是SpringBoot开发出来的。

4. **人工智能化的Java开发方式**：

- 在未来。智能Ai可以替我们写代码，到时候我们都就做机器人的指挥者，不用干活。闲余时间天天玩游戏...



1.3 开发环境要求

Spring Boot 的2.1.7.RELEASES正式发行版，必须要使用Java8或 Java 11，Spring版本也必须是5.1.8及以上

构建工具版本：Maven ，版本要求是3.3及以上。

Servlet容器版本：

SpringBoot 支持如下的嵌入式Servlet容器，Spring Boot应用程序最低支持到Servlet 3.1的容器。

Name	Servlet Version
Tomcat 9.0	4.0
Jetty 9.4	3.1
Undertow 2.0	4.0

1.4 Spring怎么做Web开发？

我们怎么开发一个web项目：

1. web.xml配置：SpringMVC核心控制器(DispatchServlet)，Spring容器监听器，编码过滤器....
2. Spring 配置：包扫描(service、dao)，配置数据源，配置事务....
3. SpringMVC配置：包扫描(controller)，视图解析器，注解驱动，拦截器，静态资源....
4. 日志配置
5. 少量业务代码
6. ...
7. 部署 Tomcat 调试，每次测试都需要部署
8. ...

但是如果用 Spring Boot 呢？

超简单！无需配置！！无感Tomcat！超迅速搭建功能强大的整套 Web！到底多简单？入门案例揭晓。

二、SpringBoot快速入门

2.1 Maven搭建SpringBoot工程

Maven搭建SpringBoot工程，实现web的请求响应。浏览器访问在页面中输出`helloworld`。

实现步骤：

1. 创建Maven工程
2. pom.xml文件中配置起步依赖
3. 编写SpringBoot启动引导类
4. 编写Controller
5. 访问http://localhost:8080/hello测试

实现过程：

1. 创建Maven工程
2. pom.xml文件中配置父坐标和web的起步依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
    <!--继承SpringBoot父POM文件-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.7.RELEASE</version>
    </parent>

    <groupId>com.itheima</groupId>
    <artifactId>day01_springboot_helloword</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!--web 开发的相关依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>
```

3. 编写SpringBoot引导类

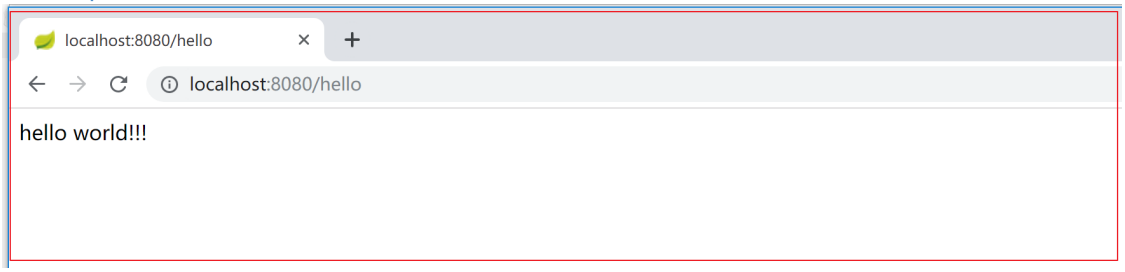
```
@Configuration//配置类
@EnableAutoConfiguration//开启自动配置
@ComponentScan//包扫描
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class,args);
    }
}
```

4. 编写Controller

    1. controller

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String hello(String name){
        return "hello world!!!";
    }
}
```

5. 访问http://localhost:8080/hello测试



2.2 使用IDEA快速创建SpringBoot项目

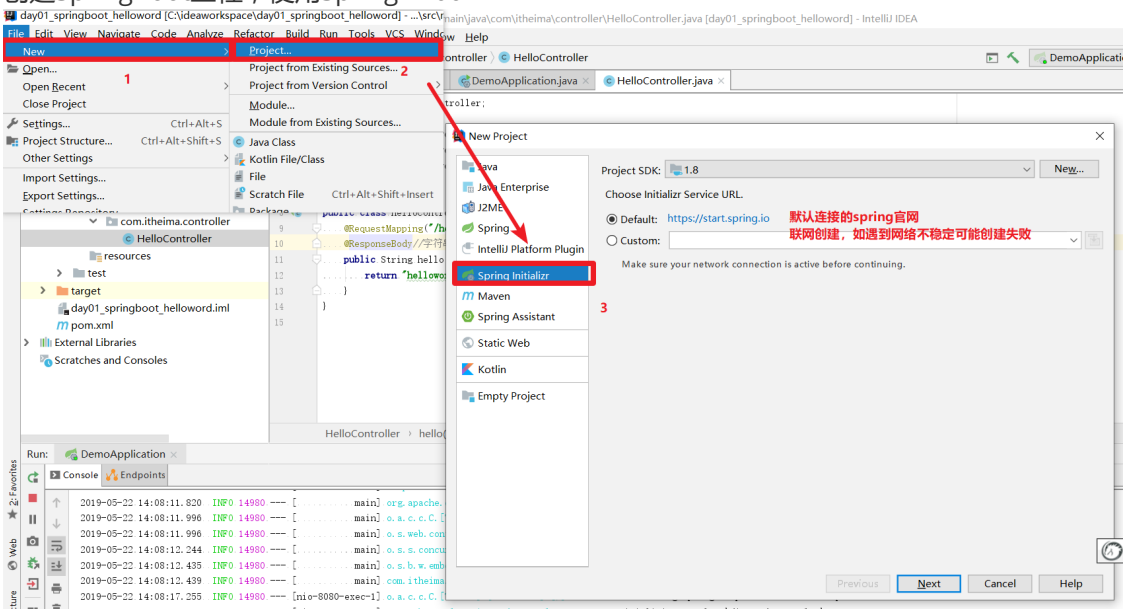
使用Spring Initializr 方式创建SpringBoot工程。然后实现入门案例的代码。

实现步骤：

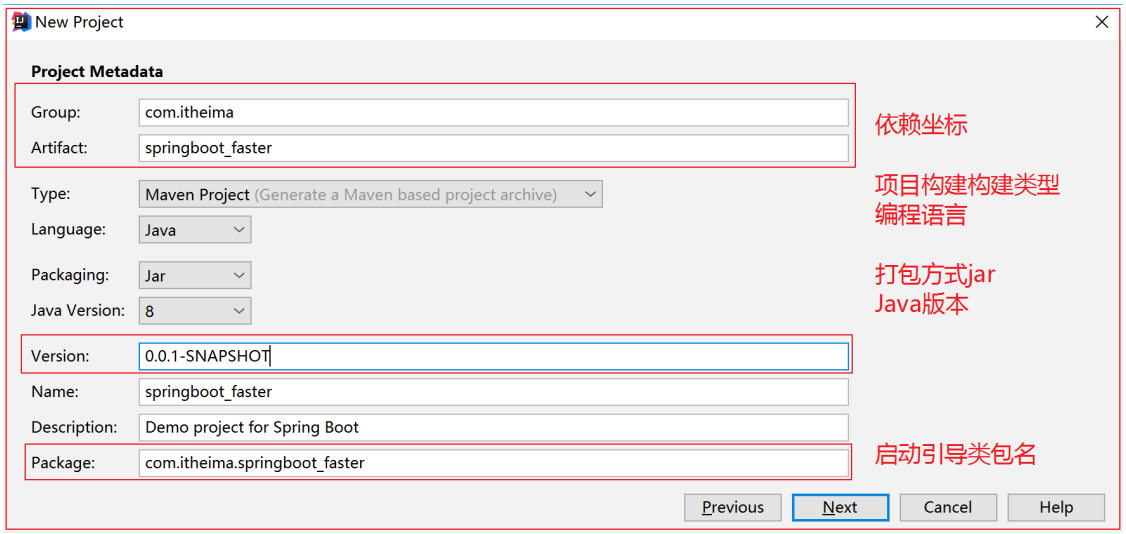
1. 创建SpringBoot项目：使用Spring Initializr
2. 配置项目元信息
3. 勾选起步依赖
4. 再次编写controller
5. 访问接口测试：http://localhost:8080/hello

实现过程：

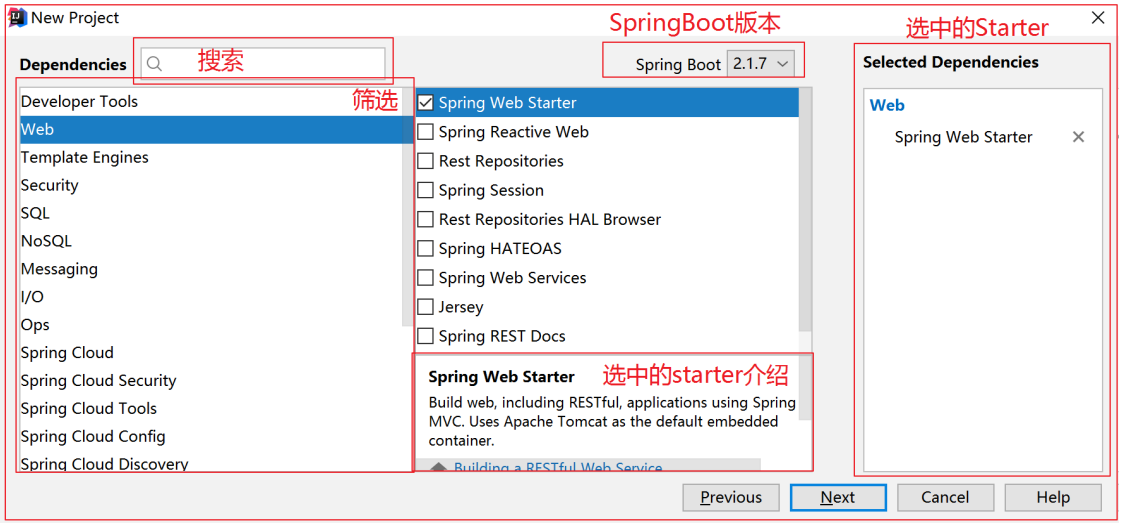
1. 创建SpringBoot工程，使用Spring Initializr



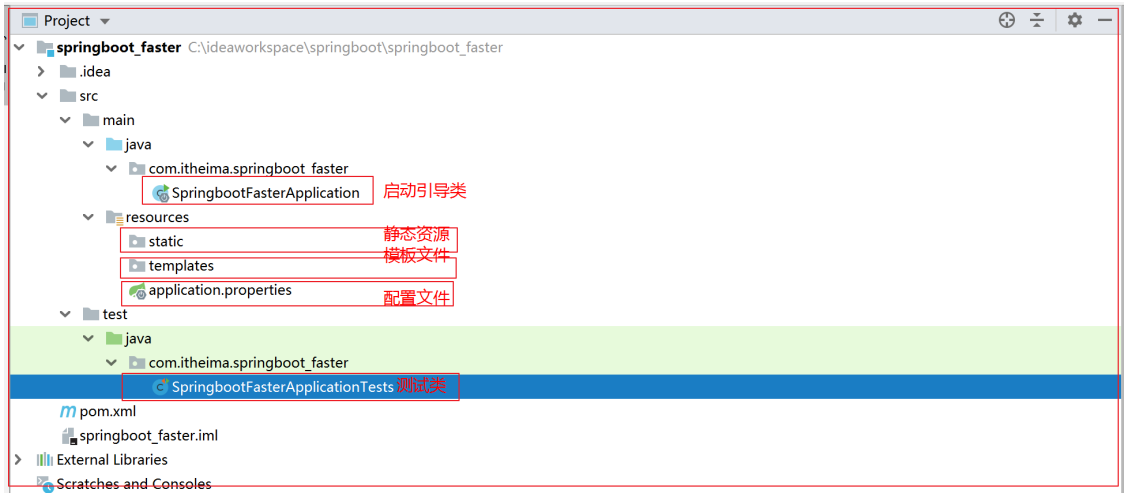
2. 配置项目信息



3. 勾选起步依赖



创建完成后工程目录结构



- pom文件介绍

4. 再次编写Controller，同上案例
5. 访问http://localhost:8080/hello接口测试

2.3 SpringBoot工程热部署(LiveReload)

热部署依赖坐标：

```
<!--spring-boot开发工具jar包，支持热部署-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

注意：加入坐标之后，IDEA进行热部署也会失败。

原因：默认情况IDEA不会自动编译，需要设置`Build Project Automatically`

`Shift + Ctrl + Alt + /` 打开Maintenance维护，选择Registry(注册表)

三、SpringBoot原理分析

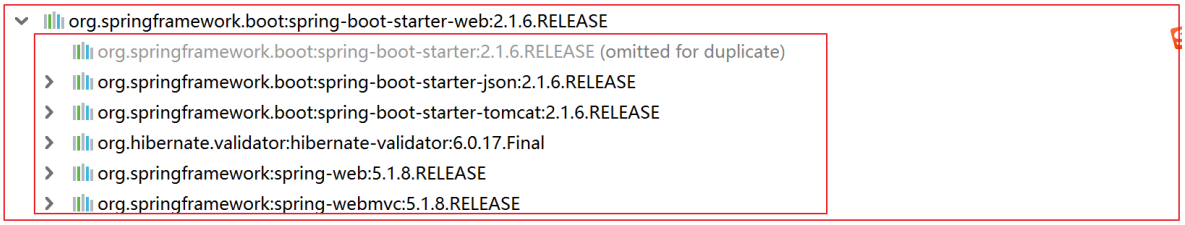
3.1 starters的原理

starters是依赖关系的整理和封装。是一套依赖坐标的整合，可以让导入应用开发的依赖坐标更方便。

有了这些Starters，你获得Spring和其整合的所有技术的一站式服务。无需配置(自动配置)、无需复制粘贴依赖坐标，一个坐标即可完成所有入门级别操作。举例：JPA or Web开发，只需要导入`spring-boot-starter-data-jpa`或`spring-boot-starter-web`。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

每个Starter包含了当前功能下的许多必备依赖坐标，这些依赖坐标是项目开发，上线和运行必须的。同时这些依赖也支持依赖传递。举例：`spring-boot-starter-web`包含了所有web开发必须的依赖坐标



starter的命名规范：官方的starter写法`spring-boot-starter-*`，非官方的starter写法`thirdpartyproject-spring-boot-starter`

常用的starters有哪些？

非常多，一下只列举部分：

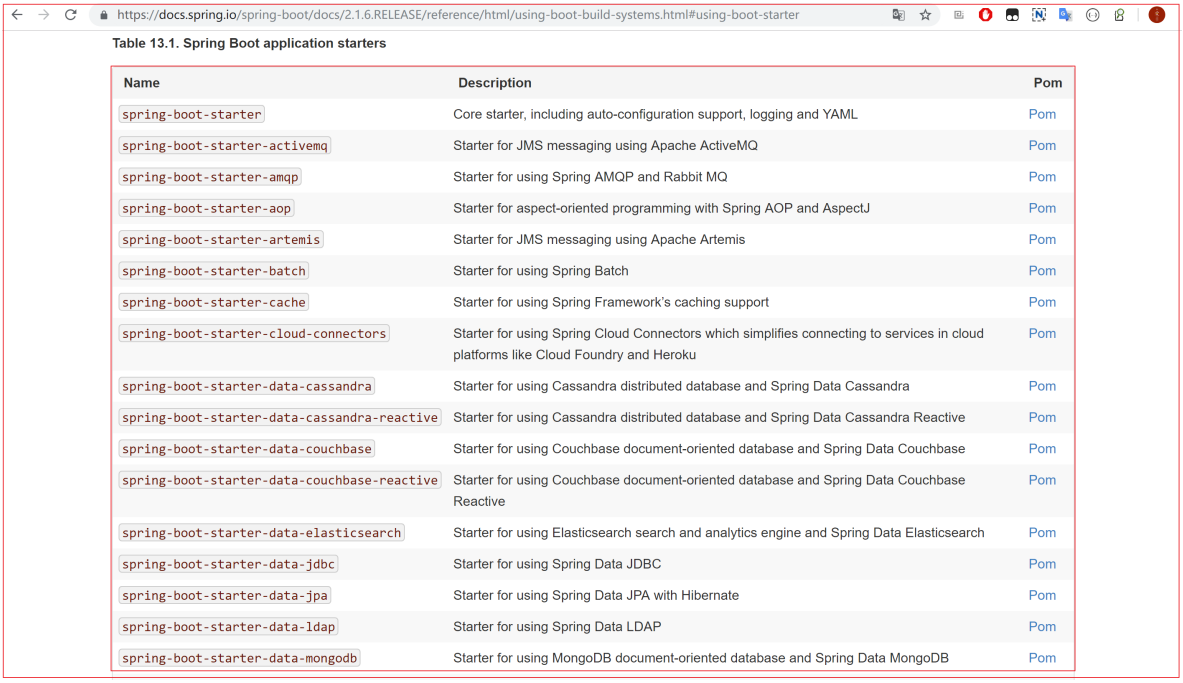


Table 13.1. Spring Boot application starters

Name	Description	Pom
spring-boot-starter	Core starter, including auto-configuration support, logging and YAML	Pom
spring-boot-starter-activemq	Starter for JMS messaging using Apache ActiveMQ	Pom
spring-boot-starter-amqp	Starter for using Spring AMQP and Rabbit MQ	Pom
spring-boot-starter-aop	Starter for aspect-oriented programming with Spring AOP and AspectJ	Pom
spring-boot-starter-artemis	Starter for JMS messaging using Apache Artemis	Pom
spring-boot-starter-batch	Starter for using Spring Batch	Pom
spring-boot-starter-cache	Starter for using Spring Framework's caching support	Pom
spring-boot-starter-cloud-connectors	Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku	Pom
spring-boot-starter-data-cassandra	Starter for using Cassandra distributed database and Spring Data Cassandra	Pom
spring-boot-starter-data-cassandra-reactive	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive	Pom
spring-boot-starter-data-couchbase	Starter for using Couchbase document-oriented database and Spring Data Couchbase	Pom
spring-boot-starter-data-couchbase-reactive	Starter for using Couchbase document-oriented database and Spring Data Couchbase Reactive	Pom
spring-boot-starter-data-elasticsearch	Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch	Pom
spring-boot-starter-data-jdbc	Starter for using Spring Data JDBC	Pom
spring-boot-starter-data-jpa	Starter for using Spring Data JPA with Hibernate	Pom
spring-boot-starter-data-ldap	Starter for using Spring Data LDAP	Pom
spring-boot-starter-data-mongodb	Starter for using MongoDB document-oriented database and Spring Data MongoDB	Pom

starter为什么不需要写版本？

3.2 依赖管理的原理分析

依赖管理(Dependency Management)

继承了SpringBoot的父pom文件继承了很多东西，其中最重要的要数。

1. 继承`spring-boot-starter-parent`的
    - spring-boot-denpendencies通过Maven的标签特性实现jar版本管理
    - 通过spring-boot-denpendencies的pom管理所有公共Starter依赖的版本
    - Starter是随用随去，避免一下子继承父类所有的starter依赖。
2. POM文件中的Maven插件

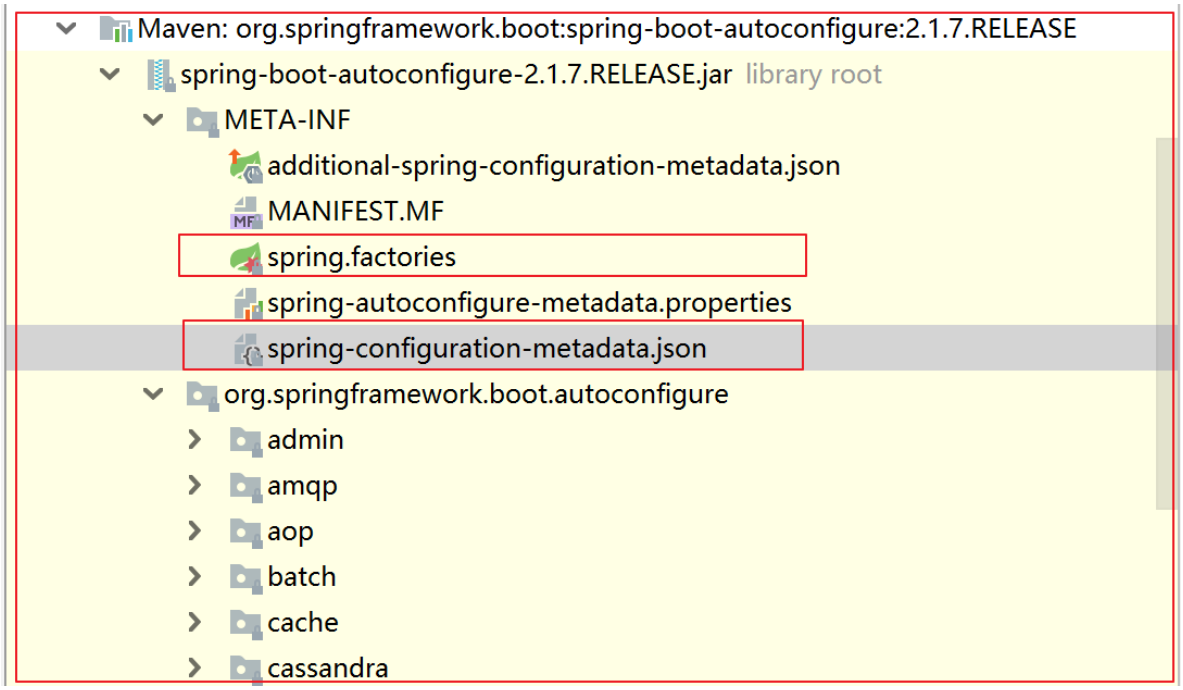
```
<! -- 作用：将一个SpringBoot的工程打包成为可执行的jar包 -->
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

如果想使用父pom文件中的任何插件，无需配置即可使用

3.3 自动配置(AutoConfiguration)原理

每个Starter基本都会有自动配置AutoConfiguration，AutoConfiguration的jar包定义了约定的默认配置信息。SpringBoot采用约定大于配置设计思想。

自动配置的值在哪里？

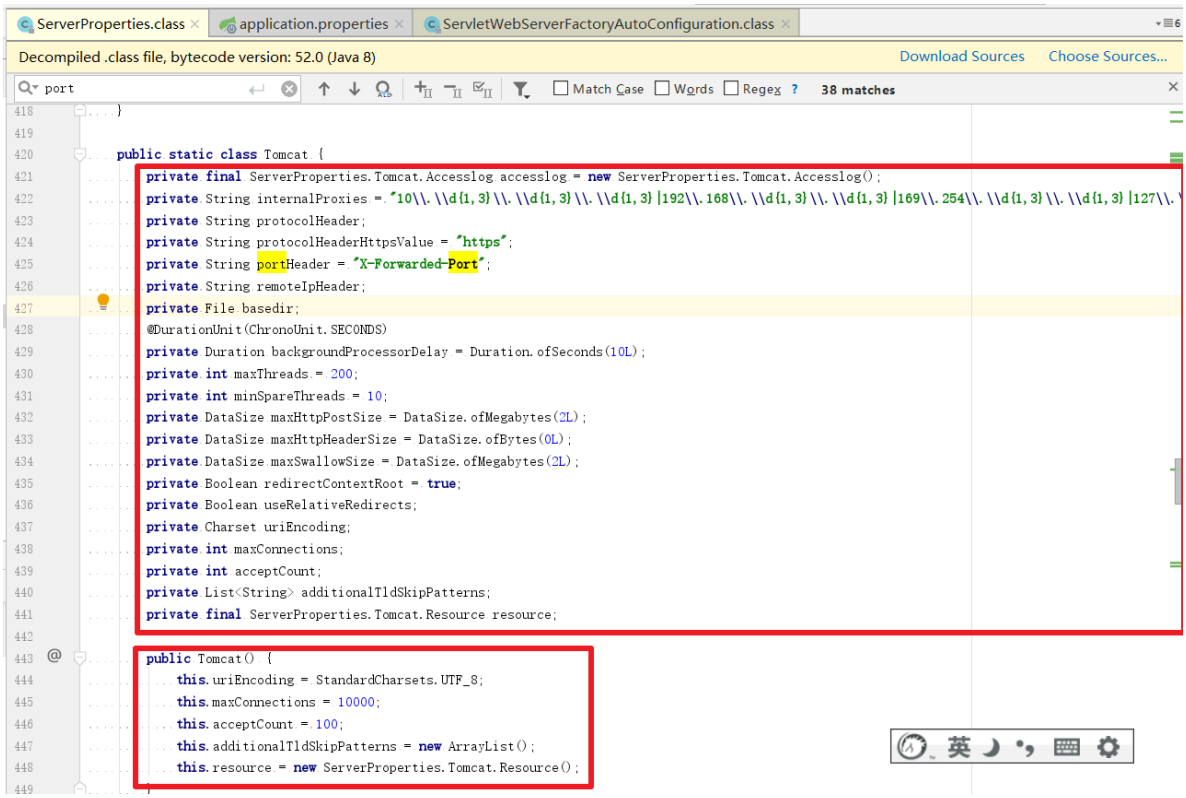


自动配置的值怎么才能生效？

查看启动类注解@SpringBootApplication

追踪步骤：

2. @EnableAutoConfiguration
3. @Import({AutoConfigurationImportSelector.class})
4. spring.factories
5. org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfiguration
6. @EnableConfigurationProperties({ServerProperties.class})
7. private final ServerProperties.Tomcat tomcat = new ServerProperties.Tomcat();



有了自动配置，那么基本全部采用默认配置。当然也可以更改默认配置，怎么改？

四、SpringBoot的配置文件

我们知道SpringBoot是**约定大于配置**的，所以很多配置都有默认值。如果想修改默认配置，可以使用application.properties或application.yml(application.yaml)自定义配置。SpringBoot默认从Resource目录加载自定义配置文件。application.properties是键值对类型。application.yml是SpringBoot中一种新的配置文件方式。

例如：

application.properties文件

```
server.port=8888
server.servlet.context-path=demo
```

application.yml文件

```
server:
  # 端口
  port: 8888
  # Path路径
  servlet:
    context-path: /demo
```

4.1 application.yml配置文件

YML文件格式是YAML(YAML Aint Markup Language)编写的文件格式。可以直观被电脑识别的格式。容易阅读，容易与脚本语言交互。可以支持各种编程语言(C/C++、Ruby、Python、Java、Perl、C#、PHP)。以数据为核心，**比XML更简洁。**扩展名为.yml或.yaml

配置普通数据语法：`key: value`

示例代码：

```
# yaml
username: haohao
```

注意：Value之前有一个空格

配置对象数据：

示例代码：

```
person:
    name: haohao
    age: 31
    addr: beijing
# 行内配置
person: {name: haohao,age: 31,addr: beijing}
```

注意：yml语法中，相同缩进代表同一个级别

配置集合、数组数据语法：

示例代码：

```
# 数组
citys:
    - beijing
    - tianjin
    - shanghai
    - chongqing
# 或者行内注入
citys: [beijing,tianjin,shanghai,chongqing]
```

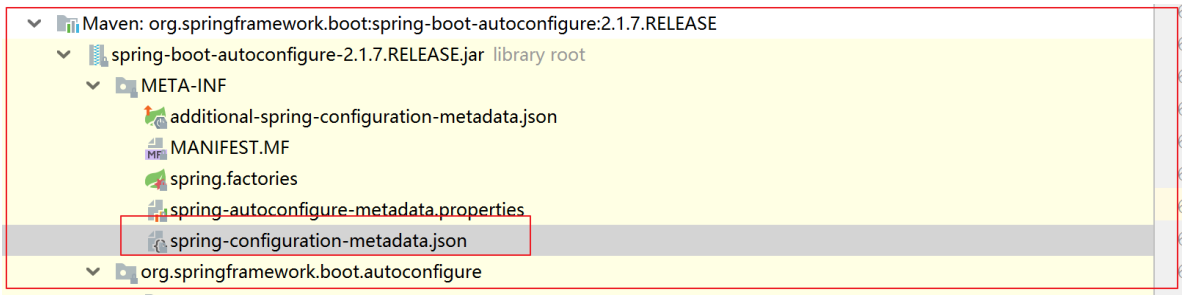
```
#集合中的元素是对象形式
students:
    - name: zhangsan
      age: 18
      score: 100
    - name: lisi
      age: 28
      score: 88
    - name: wangwu
      age: 38
      score: 90
# 或者使用行内注入
student: [{name: zhangsan,age: 18,score: 100},{name: lisi,age: 28,score: 88},
{name: wangwu,age: 38,score: 90}]
```

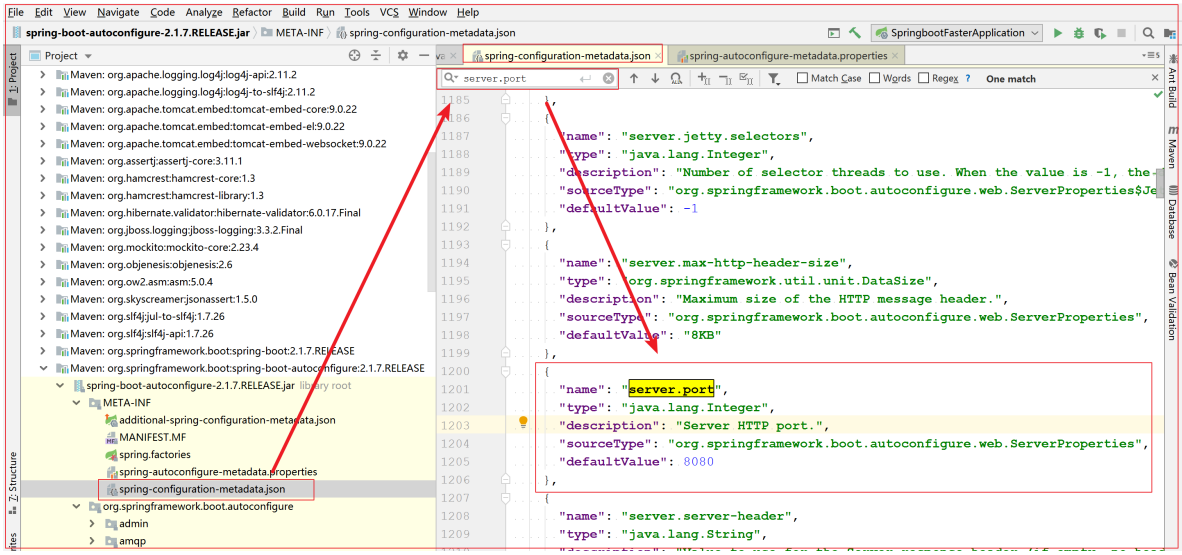
注意：value1与-之间存在一个空格

4.2 SpringBoot配置信息的查询

修改配置时，配置项目查询方式

第一种：自动配置jar包中的META-INF文件夹下，spring-configuration-metadata.json文件中





第二种：官方配置文件地址

官方查询地址：https://docs.spring.io/spring-boot/docs/2.0.1.RELEASE/reference/htmlsingle/#common-application-properties

常用配置：

```
# QUARTZ SCHEDULER (QuartzProperties)
spring.quartz.jdbc.initialize-schema=embedded # Database schema initialization
mode.
spring.quartz.jdbc.schema=classpath:org/quartz/impl/jdbcjobstore/tables_@@platfo
rm@@. sql # Path to the SQL file to use to initialize the database schema.
spring.quartz.job-store-type=memory # Quartz job store type.
spring.quartz.properties.*= # Additional Quartz Scheduler properties.
# ----------------------------------------
# WEB PROPERTIES
# ----------------------------------------
# EMBEDDED SERVER CONFIGURATION (ServerProperties)
server.port=8080 # Server HTTP port. server.servlet.context-path= # Context path
of the application. server.servlet.path=/ # Path of the main dispatcher servlet.
# HTTP encoding (HttpEncodingProperties)
spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and responses.
Added to the "Content-Type" header if not set explicitly.
# JACKSON (JacksonProperties)
spring.jackson.date-format= # Date format string or a fully-qualified date
format class name. For instance, `yyyy-MM-dd HH:mm:ss`.
```

扩展点

1. 配置简写

```yaml
server:
  port: 8888
  servlet:
    context-path: /demo
# 简写
server.port: 8888
server.servlet.context-path: /demo
```

2. 配置自动补全功能
3. properties文件转换为yml文件

4.3 配置文件属性注入Bean

1、使用注解@Value映射

@value注解将配置文件的值映射到Spring管理的Bean属性值

2、使用注解@ConfigurationProperties映射

通过注解@ConfigurationProperties(prefix="配置文件中的key的前缀")可以将配置文件中的配置自动与实体进行映射。

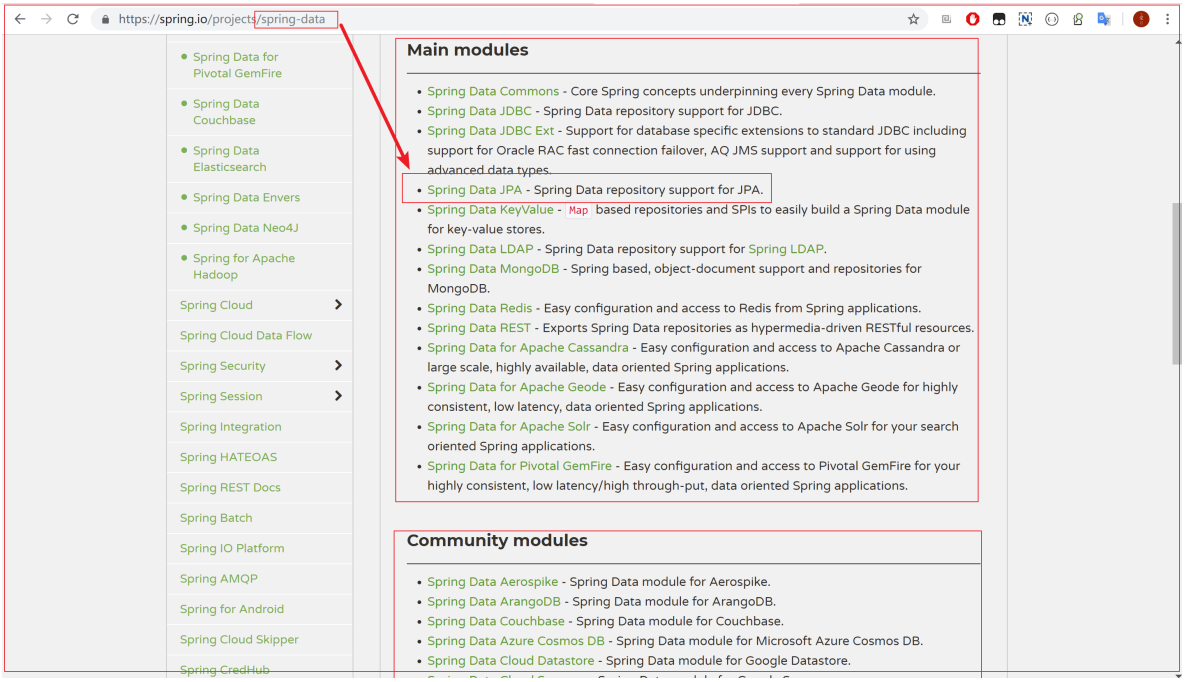
使用@ConfigurationProperties方式必须提供Setter方法，使用@Value注解不需要Setter方法。

五、SpringBoot集成一切

5.1 集成 Spring Data JPA

什么是SpringData ?

Spring Data是一个用于**简化数据访问**，并支持云服务的开源框架。**其主要目标是使得对数据的访问变得方便快捷**。Spring Data JPA 是其中之一。



Spring Data JPA 是Spring 基于 ORM 框架、JPA 规范的基础上封装的一套JPA应用框架，可使开发者用极简的代码即可实现对数据库的访问和操作。它提供了包括增删改查等在内的常用功能，且易于扩展！学习并使用 Spring Data JPA 可以极大提高开发效率！

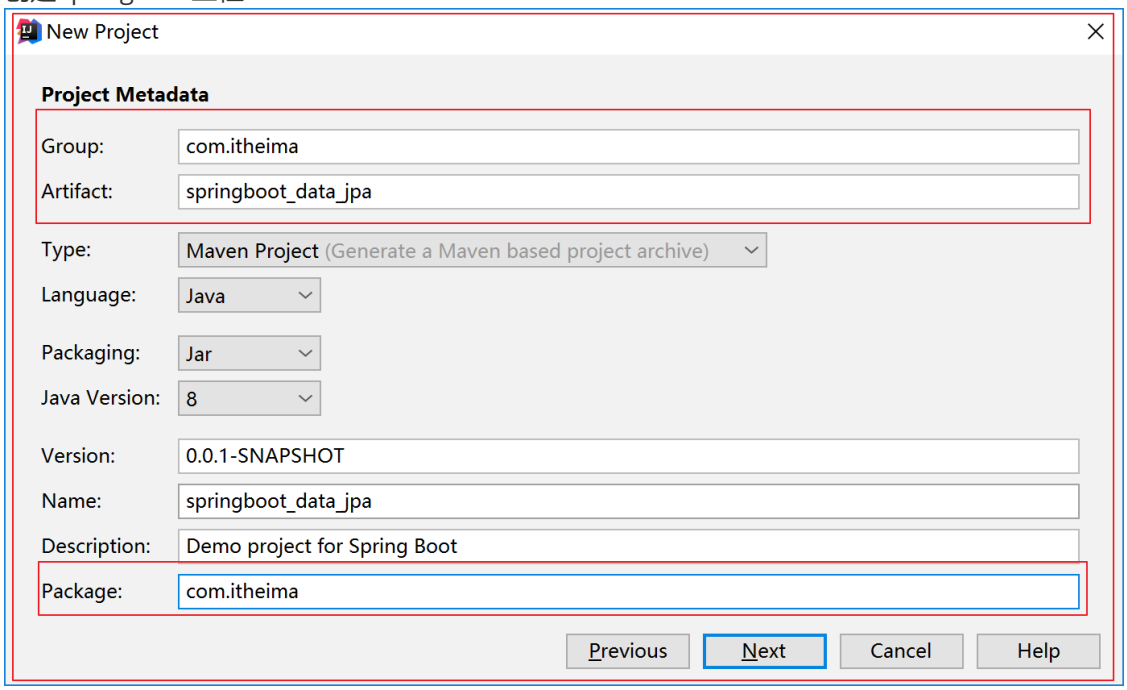
目标：使用SpringBoot整合SpringDataJPA，完成数据的增删改查基本功能。

实现步骤：

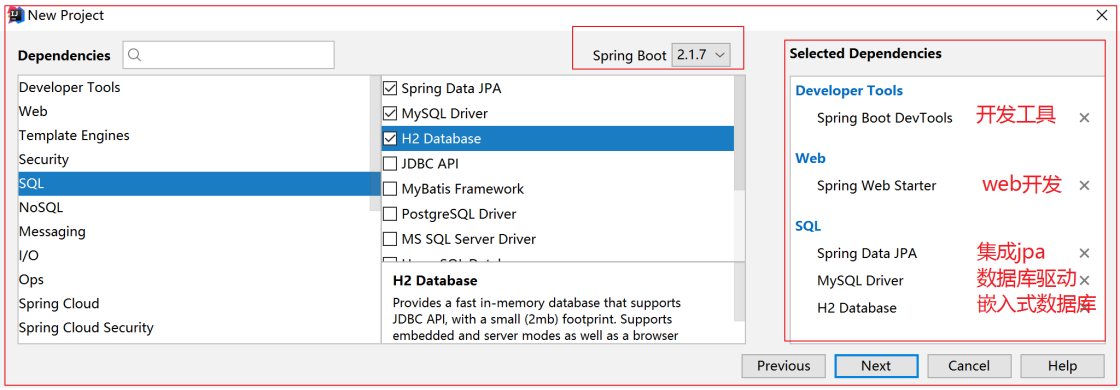
1. 创建SpringBoot工程
2. 勾选依赖坐标
3. 配置：数据库连接、jpa相关
4. 创建User表、创建实体User配置实体
5. 编写UserRepository
6. 编写Controller、Service
7. 访问测试

实现过程：

1. 创建SpringBoot工程



2. 勾选依赖坐标



3. 配置：数据库连接、jpa相关

```
# DB 配置(可以不写，有嵌入式数据库，如果不写必须显示导入嵌入式数据库starter，
h2database)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.password=root
spring.datasource.username=root
spring.datasource.url=jdbc:mysql://127.0.0.1/test?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
#jpa 相关配置(可以不写，有默认值)
# 数据库类型
spring.jpa.database=mysql
# 是否显示sql
spring.jpa.show-sql=true
# hibernate初始化数据库表策略
spring.jpa.hibernate.ddl-auto=update
# 是否生成数据库定义表语句
spring.jpa.generate-ddl=true
```

4. 创建表，创建实体配置实体

```
-- ----------------------------
-- Table structure for `user`
-- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(50) DEFAULT NULL,
`password` varchar(50) DEFAULT NULL,
`name` varchar(50) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'zhangsan', '123', '张三');
INSERT INTO `user` VALUES ('2', 'lisi', '123', '李四');
```

```
@Entity//实体类注解
@Table(name = "user")//关联数据库表
public class User {
    //注解设置当前id为注解
```

```
    @Id
    //注解值生成策略
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;//用户名
    private String password;//密码
    private String name;//姓名
    //getter  setter
    //toString
}
```

5. 编写UserRepository
    - 泛型需要实体类，和实体类的ID

```
public interface UserDao extends JpaRepository<User,Integer> {}
```

6. 编写Controller、Service

```
@RestController
@RequestMapping("/user")
public class UserController {
    @Resource
    private UserService userService;

    @GetMapping("/findAll")
    public List<User> findAll(){
        return userService.findAll();
    }

    @GetMapping("/findById/{id}")
    public User findById(@PathVariable("id") Integer id){
        return userService.findById(id);
    }

    @GetMapping("/delete/{id}")
    public void delete(@PathVariable("id") Integer id){
        userService.delete(id);
    }

    @PostMapping("/update")
    public void update(@RequestBody User User){
        userService.update(User);
    }
}
```

```
@Service
public class UserServiceImpl implements UserService {
    @Resource
    private UserMapper userMapper;
    @Resource
    private UserDao userDao;

    @Override
    public List<User> findAll() {
        return userDao.findAll();
```

```
    }

    @Override
    public User findById(Integer id) {
        return userDao.findById(id).get();
    }

    @Override
    public void delete(Integer id) {
        User User = new User();
        User.setId(id);
        userDao.delete(User);
    }

    @Override
    public void update(User User) {
        userDao.save(User);
    }

}
```

7. 访问测试地址
    - http://localhost:8080/user/findById/1
    - http://localhost:8080/user/delete/1
    - http://localhost:8080/user/update
    - http://localhost:8080/user/findAll

5.2 集成MyBatis

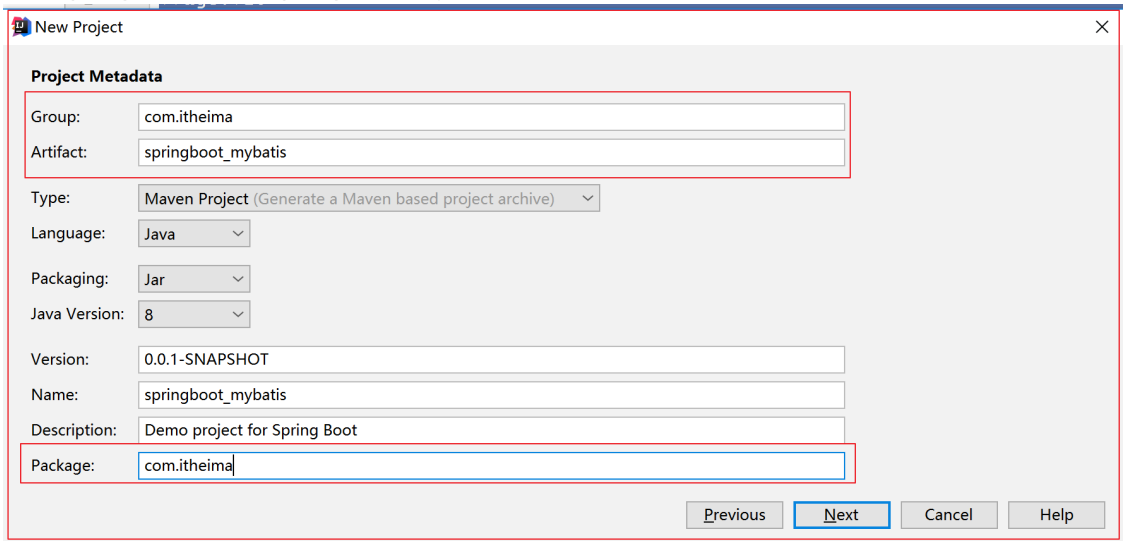
使用SpringBoot整合MyBatis，完成**查询所有功能。**

实现步骤：

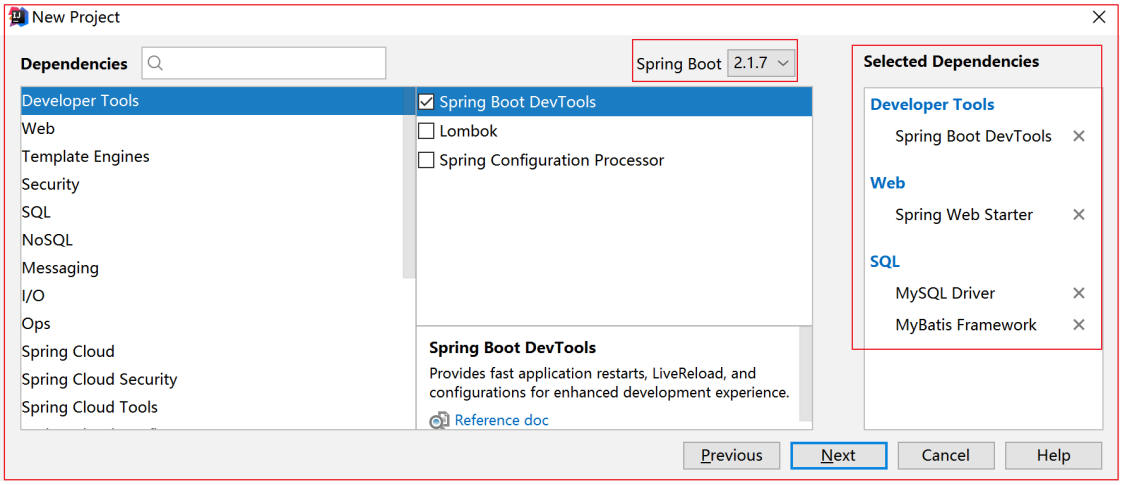
1. 创建SpringBoot工程
2. 勾选依赖坐标
3. 数据库连接信息
4. 创建User表、创建实体User
5. 编写三层架构：Mapper、Service、controller，编写查询所有的方法
6. 配置Mapper映射文件
7. 在application.properties中添加MyBatis配置，扫描mapper.xml和mapper
8. 访问测试地址http://localhost:8080/queryUsers

实现过程：

1. 创建SpringBoot工程，springboot_mybatis



2. 勾选依赖坐标



3. 在application.properties中添加数据库连接信息

```
# DB 配置
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.password=root
spring.datasource.username=root
spring.datasource.url=jdbc:mysql://127.0.0.1/test?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
```

   - 数据库连接地址后加`?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC`，不然会报错

4. 创建User表—>创建实体User
   - 创建表

```
-- ----------------------------
-- Table structure for `user`
-- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(50) DEFAULT NULL,
`password` varchar(50) DEFAULT NULL,
`name` varchar(50) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'zhangsan', '123', '张三');
INSERT INTO `user` VALUES ('2', 'lisi', '123', '李四');
```

- 创建实体

```
public class User {
    private Integer id;
    private String username;//用户名
    private String password;//密码
    private String name;//姓名
    //getter setter...
    //toString
}
```

5. 编写Mapper：使用@Mapper标记该类是一个Mapper接口，可以被SpringBoot自动扫描

```
@Repository
@Mapper//表明当前接口是一个Mapper，被Mybatis框架扫描
public interface UserMapper {

    List<User> findAll();

    User findById(Integer id);

    void save(User user);

    void update(User user);

    void delete(Integer id);

}
```

6. 配置Mapper映射文件：在src/main/resources/mapper路径下加入UserMapper.xml配置文件

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mapper.UserMapper">
    <select id="findAll" resultType="user">
        select * from user
    </select>
    <select id="findById" parameterType="Integer" resultType="user">
        select * from user where id = #{id}
    </select>
    <insert id="save" parameterType="user">
        INSERT into user (username,password,name) VALUES (#{username},#
{password},#{name})
    </insert>
    <update id="update" parameterType="user">
        update user set username=#{username},password=#{password},name=#
{name} where id=#{id}
    </update>
    <delete id="delete" parameterType="Integer">
```

```
        delete from user where id=#{id}
    </delete>
</mapper>
```

7. 在application.properties中添加MyBatis信息

```
# 扫描实体
mybatis.type-aliases-package=com.itheima.domain
# mapper.xml配置文件路径
mybatis.mapper-locations=classpath:mapper/*Mapper.xml
```

8. 编写Controller

```
@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    UserService userService;
    //查询所有
    @RequestMapping("/findAll")
    public List<User> findAll() throws JsonProcessingException {
         return userService.findAll();
    }

    //根据id查询
    @RequestMapping("/findById")
    public User findById(Integer id) {
        return userService.findById(id);
    }

    //新增
    @RequestMapping("/save")
    public void save(User user) {
        userService.save(user);
    }

    //修改
    @RequestMapping("/update")
    public void update(User user) {
        userService.update(user);
    }

    //删除
    @RequestMapping("/delete")
    public void delete(Integer id) {
        userService.delete(id);
    }
}
```

9. 访问测试地址

    1. http://localhost:8080/user/findById/1
    2. http://localhost:8080/user/delete/1
    3. http://localhost:8080/user/update
    4. http://localhost:8080/user/findAll

5.3 集成Spring Data Redis

SpringBoot整合了Redis之后，做用户数据查询缓存。

实现步骤：

1. 添加Redis的Starter
2. 在application.properties中配置redis端口、地址
3. 注入RedisTemplate操作Redis缓存查询所有用户数据
4. 测试缓存

实现过程：

1. 添加Redis起步依赖

```
<!--spring data redis 依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

2. 配置Redis连接信息

```
# Redis 配置(不填也是可以的)
spring.redis.host=localhost
spring.redis.port=6379
```

3. 注入RedisTemplate测试Redis操作

```
@Test
public void testRedis() throws JsonProcessingException {
    String users = (String)
redisTemplate.boundValueOps("user.findAll").get();
    if (users == null) {
        List<User> userList = userMapper.queryUserList();
        ObjectMapper jsonFormat = new ObjectMapper();
        users = jsonFormat.writeValueAsString(userList);
        redisTemplate.boundValueOps("user.findAll").set(users);
        System.out.println("==============从数据库中获取用户数据
==================");
    }else {
        System.out.println("==============从Redis缓存中获取用户数据
==================");
    }
    System.out.println(users);
}
```

5.4 集成定时器

使用SpringBoot开发定时器，每隔5秒输出一个当前时间。

实现步骤：

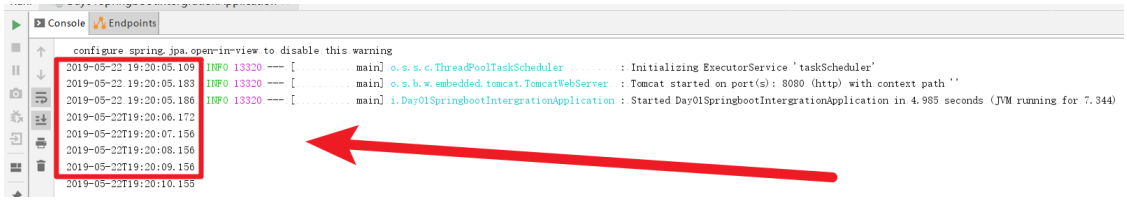
1. 开启定时器注解

```
@SpringBootApplication
@EnableScheduling//开启定时器
public class Day01SpringbootIntergrationApplication {
    public static void main(String[] args) {
        SpringApplication.run(Day01SpringbootIntergrationApplication.class,
args);
    }

}
```

2. 配置定时器方法

```
@Component
public class TimerUtil {

    @Scheduled(initialDelay = 1000,fixedRate = 1000)
    public void mytask(){
        System.out.println(LocalDateTime.now());
    }
}
```

3. 测试定时器。



5.5 扩展了解：除此之外还可以集成什么?

1. 集成 MongoDB
2. 集成 ElasticSearch
3. 集成 Memcached
4. 集成邮件服务：普通邮件、模板邮件、验证码、带Html的邮件
5. 集成RabbitMQ消息中间件
6. 集成Freemarker或者Thymeleaf
7. ……

六、SpringBoot如何代码测试

SpringBoot集成JUnit测试功能，进行查询用户接口测试。

实现步骤：

1. 添加Junit起步依赖(默认就有)

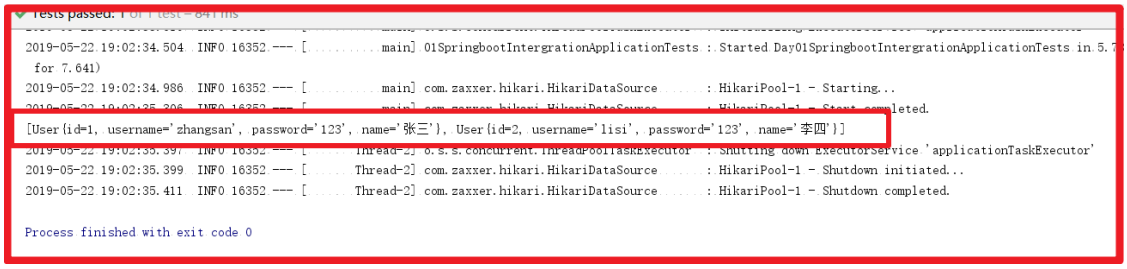
```
<!--spring boot测试依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2. 编写测试类：

- SpringRunner继承SpringJUnit4ClassRunner，使用哪一个Spring提供的测试引擎都可以。指定运行测试的引擎
- @SpringBootTest的属性值指的是引导类的字节码对象

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class ApplicationTests {

    @Autowired
    private UserDao userDao;

    @Test
    public void test() {
        List<User> users = userDao.findAll();
        System.out.println(users);
    }
}
```

3. 控制台打印信息



七、Spring Boot 如何打包部署

启动方式有两种，一种是打成jar直接执行，另一种是打包成war包放到Tomcat服务下，启动Tomcat。

6.1 打成Jar包部署

执行maven打包命令或者使用IDEA的Maven工具打包

```
## 移动至项目根目录，与pom.xml同级
mvn clean package
## 或者执行下面的命令  排除测试代码后进行打包
mvn clean package  -Dmaven.test.skip=true
```

需要注意项目pom.xml文件中的打包类型

```
<packaging>jar</packaging>
```

启动命令：启动之前先检查自己的pom.xml文件中是否有springboot的maven插件

```
java -jar  target/springboot_demo.jar
```

启动命令的时候配置jvm参数也是可以的。然后查看一下Java的参数配置结果

```
java -Xmx80m -Xms20m  -jar target/springboot_demo.jar
```

6.2 打成war包部署

1. 执行maven打包命令或者使用IDEA的Maven工具打包，需要修改pom.xml文件中的打包类型。

```
<packaging>war</packaging>
```

2. 注册启动类

- 创建 ServletInitializer.java，继承 SpringBootServletInitializer ，覆盖 configure()，把启动类 Application 注册进去。外部 Web 应用服务器构建 Web Application Context 的时候，会把启动类添加进去。

```
//web.xml
public class ServletInitializer extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
builder) {
        return builder.sources(DemoApplication.class);
    }
}
```

3. 然后执行打包操作。同6.1 小节打包是一样的
    - 拷贝到Tomcat的webapp下，启动Tomcat访问即可
    - 因为访问地址不再是根目录了，所有路径中需要加入项目名称：http://localhost:8080/springboot_demo/hello