

# 目标

---

- 能够理解SpringBoot的设计初衷
- 能够理解SpringBoot的开发环境要求
- 能够搭建SpringBoot的入门开发工程
- 能够理解SpringBoot的配置文件常见配置
- 能够使用SpringBoot整合MyBatis
- 能够使用SpringBoot整合Redis进行简单的数据缓存
- 能够使用SpringBoot发送Http请求
- 能够使用SpringBoot进行简单代码测试
- 能够打包部署SpringBoot项目

## 一、SpringBoot简介

---

当前互联网后端开发中，JavaEE占据了主导地位。对JavaEE开发，首选框架是Spring框架。在传统的Spring开发中，需要使用大量的与业务无关的XML配置才能使Spring框架运行起来，这点备受许多开发者诟病。随着Spring4.x发布，Spring已经完全脱离XML，只使用注解就可以运行项目。为了进一步简化Spring应用的开发，SpringBoot诞生了。它是由Pivotal团队提供的全新框架，其设计目的是简化Spring应用的搭建及开发过程，并迎合时下流行的分布式微服务设计思想，越来越多企业在使用SpringBoot。本课程跟随时代的潮流，带大家掌握这门技术。

### 1.1 设计初衷

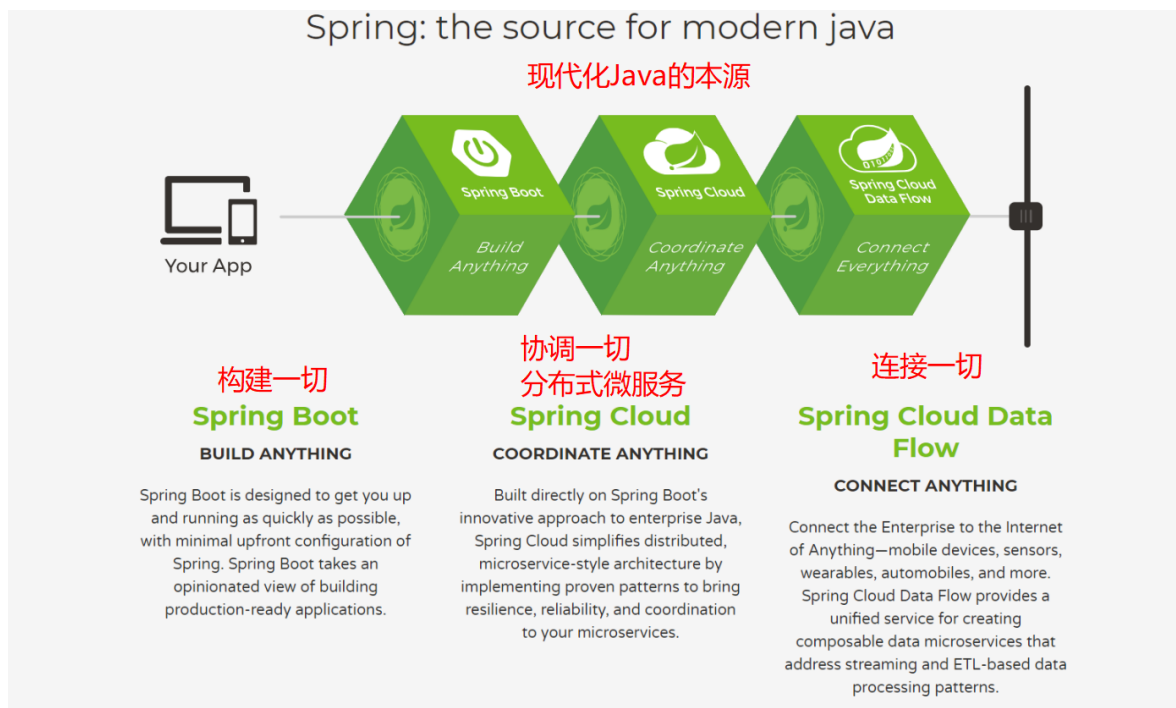
---

- 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AutoConfiguration、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



## 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day01\_springboot\_helloworld
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_helloworld</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);
    }
}
```

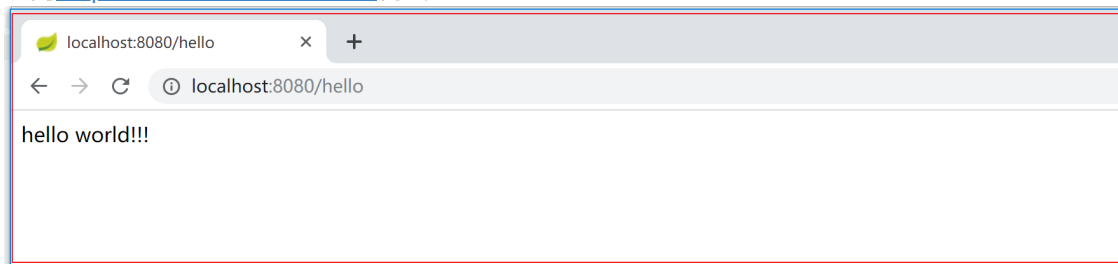
#### 4. 编写三层架构代码：Controller

##### 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

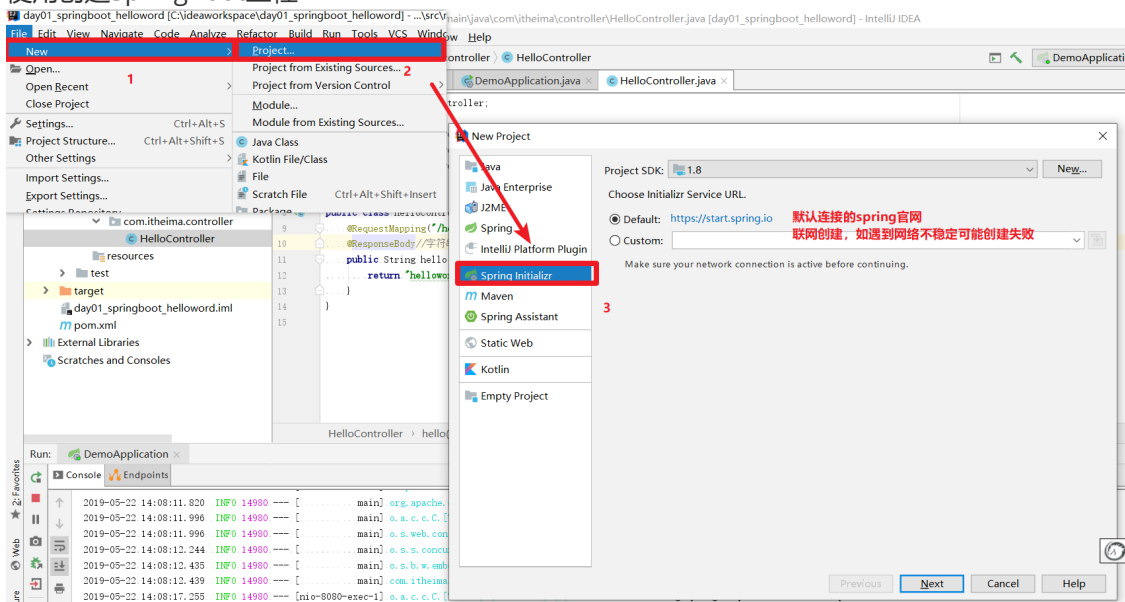
使用Spring Initializr 方式创建SpringBoot工程。然后实现入门案例的代码。

实现步骤：

1. 使用Spring Initializr创建SpringBoot
2. 配置项目信息
3. 勾选起步依赖
4. 配置文件存储路径地址
5. 再次编写入门案例三层架构代码
6. 访问<http://localhost:8080/hello>接口测试

实现过程：

##### 1. 使用创建SpringBoot工程



## 2. 配置项目信息

The 'New Project' dialog box is shown with the following fields and annotations:

- Project Metadata**
  - Group:** com.itheima (Annotation: 依赖坐标)
  - Artifact:** springboot\_faster (Annotation: 依赖坐标)
  - Type:** Maven Project (Generate a Maven based project archive) (Annotation: 项目构建构建类型)
  - Language:** Java (Annotation: 编程语言)
  - Packaging:** Jar (Annotation: 打包方式jar)
  - Java Version:** 8 (Annotation: Java版本)
  - Version:** 0.0.1-SNAPSHOT
  - Name:** springboot\_faster
  - Description:** Demo project for Spring Boot
  - Package:** com.itheima.springboot\_faster (Annotation: 启动引导类包名)
- Buttons:** Previous, Next, Cancel, Help

## 3. 勾选起步依赖

The 'New Project' dialog box is shown with the following fields and annotations:

- Dependencies**
  - Search:** 搜索
  - Filter:** 筛选
  - Spring Boot Version:** Spring Boot 2.1.7 (Annotation: SpringBoot版本)
  - Selected Starter:** Spring Web Starter (Annotation: 选中的starter介绍)
  - Selected Dependencies:** Spring Web Starter (Annotation: 选中的Starter)
- Buttons:** Previous, Next, Cancel, Help

## 4. 配置文件存储路径地址

## 5. 创建完成后工程目录结构

The Project Explorer shows the directory structure of the 'springboot\_faster' project. The annotations are as follows:

- Project:** springboot\_faster
- src/main/java:**
  - com.itheima.springboot\_faster:**
    - SpringbootFasterApplication:** 启动引导类
- src/main/resources:**
  - static:** 静态资源
  - templates:** 模板文件
  - application.properties:** 配置文件
- src/test/java:**
  - com.itheima.springboot\_faster:**
    - SpringbootFasterApplicationTests:** 测试类
- Files:** pom.xml, springboot\_faster.iml

## o pom文件介绍

```
<?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>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.7.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.itheima</groupId>
    <artifactId>springboot_faster</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot_faster</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

继承SpringBoot父POM文件

Web开发的Starter

SpringBoot测试Starter

SpringBoot的Maven插件

## 6. 编写入门案例代码

## 7. 访问<http://localhost:8080/hello>接口测试

# 2.3 SpringBoot工程热部署

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

```
▼ org.springframework.boot:spring-boot-starter-web:2.1.6.RELEASE
  org.springframework.boot:spring-boot-starter:2.1.6.RELEASE (omitted for duplicate)
  > org.springframework.boot:spring-boot-starter-json:2.1.6.RELEASE
  > org.springframework.boot:spring-boot-starter-tomcat:2.1.6.RELEASE
  > org.hibernate.validator:hibernate-validator:6.0.17.Final
  > org.springframework:spring-web:5.1.8.RELEASE
  > org.springframework:spring-webmvc:5.1.8.RELEASE
```

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：

← → ↺ https://docs.spring.io/spring-boot/docs/2.1.6.RELEASE/reference/html/using-boot-build-systems.html#using-boot-starter

Table 13.1. Spring Boot application starters

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

starter为什么不需要写版本？

## 3.2 依赖管理的原理

依赖管理(Dependency Management)

### 1. 继承了 `spring-boot-starter-parent` 的好处和特点

- 默认编译Java 1.8
- 默认编码UTF-8
- 通过spring-boot-dependencies的pom管理所有公共Starter依赖的版本
- spring-boot-dependencies通过Maven的一个特性来实现版本管理
- 随用随取，不用继承父类所有的starter依赖。

### 2. POM文件中的Maven插件

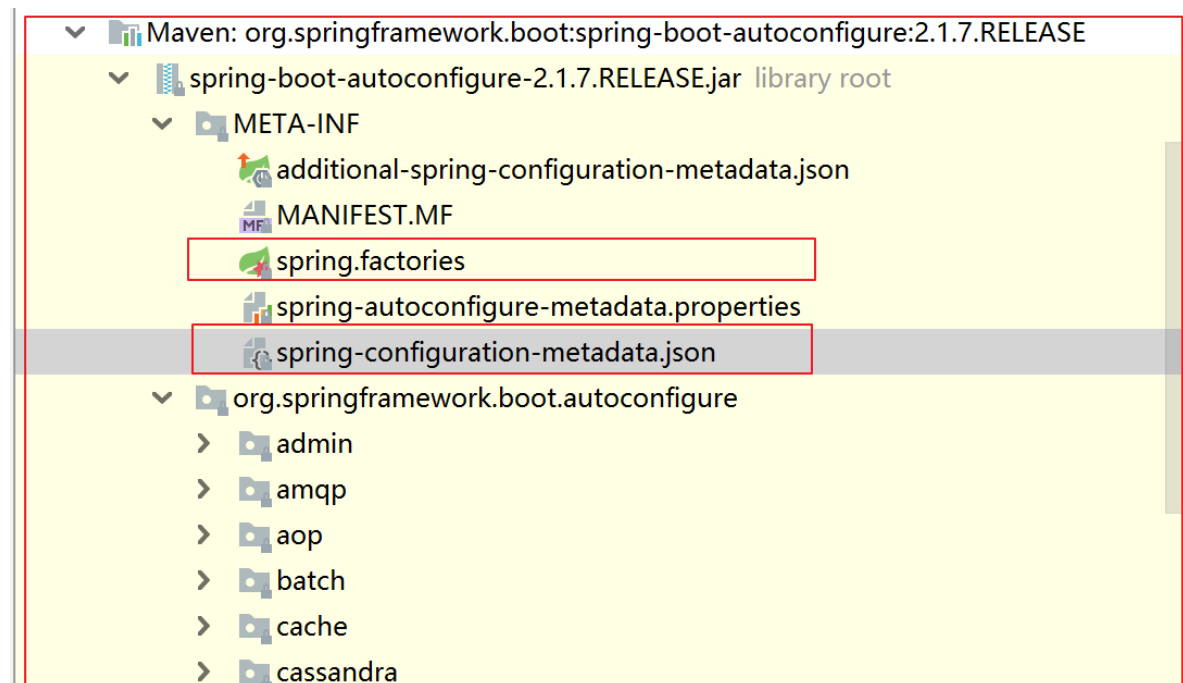
```
<!-- 作用：将一个SpringBoot的工程打包成为可执行的jar包 -->
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

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

SpringBoot采用约定大于配置设计思想，将所有可能遇到的配置信息提前配置好，写在自动配置的jar包中。每个Starter基本都会有对应的自动配置。

#### 自动配置的值在哪里？



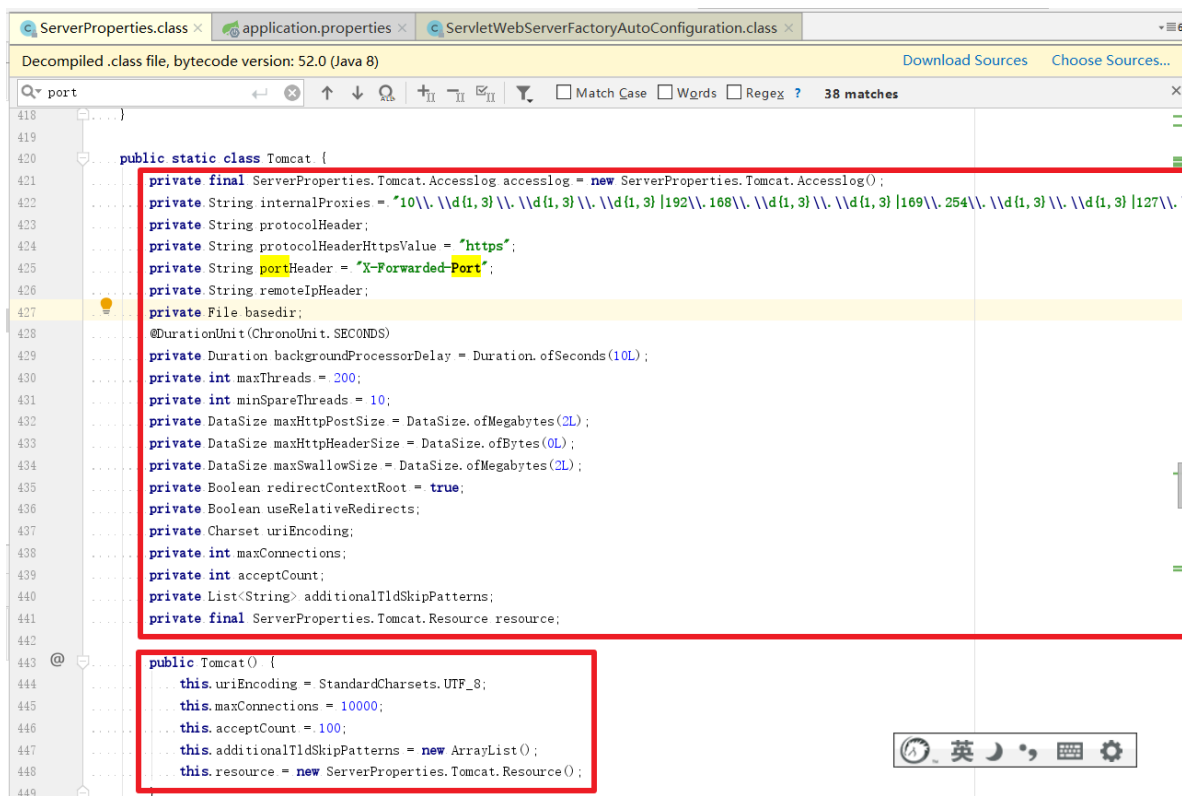
#### 自动配置的值怎么才能生效？

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(一直在用)。application.yml是SpringBoot中一种新的配置文件方式。

### 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
```

**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
city:
  - beijing
  - tianjin
  - shanghai
  - chongqing

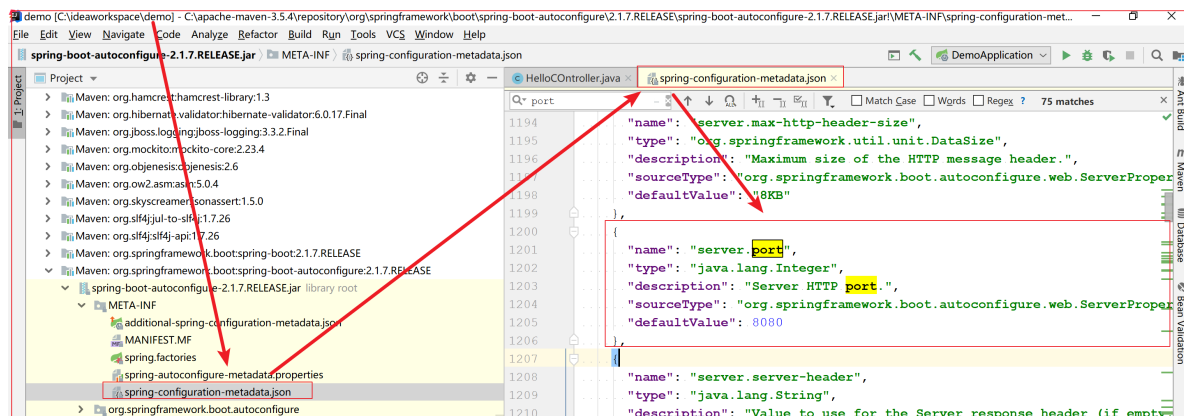
#集合中的元素是对象形式
students:
  - name: zhangsan
    age: 18
    score: 100
  - name: lisi
    age: 28
    score: 88
  - name: wangwu
    age: 38
    score: 90
```

**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：



第二种：

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_@platform@@.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`.

```

可以通过修改application.properties或者application.yml来修改SpringBoot的默认配置

例如：

application.properties文件

```

# 常见配置项目
# 端口
server.port=8080
# 项目的contentpath路径
server.servlet.context-path=/demo
# 开启debug模式
debug=true
# 配置日志级别,为debug
logging.level.com.example=debug

```

application.yml文件

```

server:
  port: 8888
  servlet:
    # 应用的
    context-path: /demo

```

扩展点

2. 配置自动补全功能配置
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与其他技术集成

### 5.1 集成MyBatis

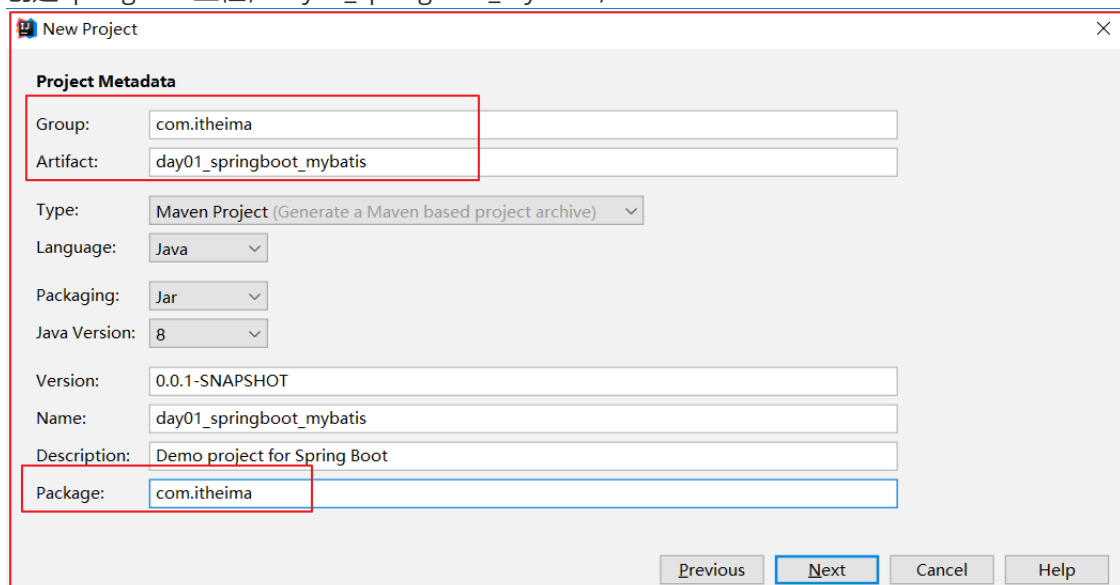
使用SpringBoot整合MyBatis，完成查询所有功能。

实现步骤：

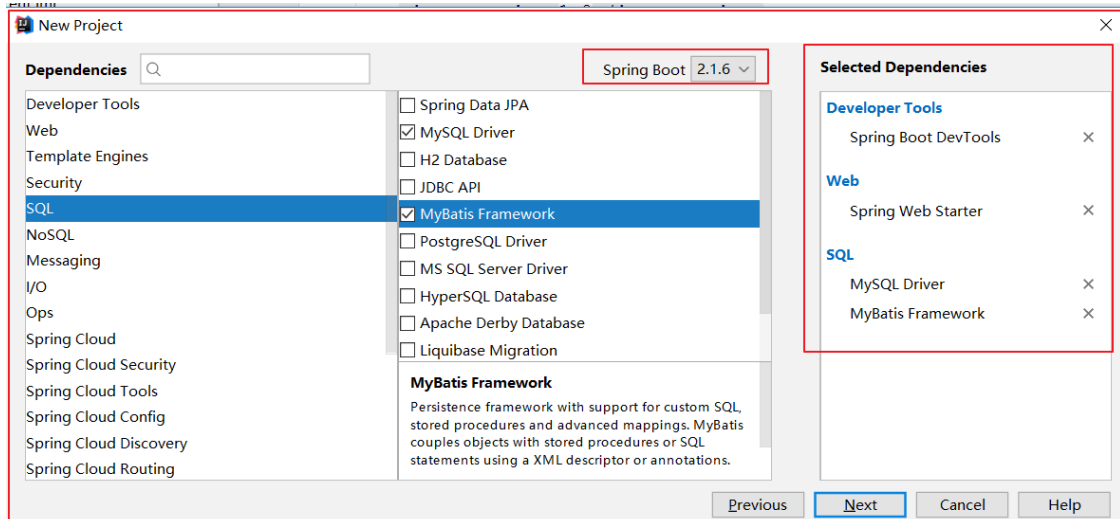
1. 创建SpringBoot工程
2. 勾选依赖坐标
3. 数据库连接信息
4. 创建User表、创建实体UserBean
5. 编写三层架构：Mapper、Service、controller，编写查询所有的方法
6. 配置Mapper映射文件
7. 在application.properties中添加MyBatis配置，扫描mapper.xml和mapper
8. 访问测试地址<http://localhost:8080/queryUsers>

实现过程：

1. 创建SpringBoot工程，day01\_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`，否则会报错

## 4. 创建User表—>创建实体UserBean

- 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
public interface UserMapper {
    List<User> findAll();
}
```

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.dao.UserMapper">
    <select id="findAll" resultType="com.itheima.domain.User">
        select * from user
    </select>
</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
public class HelloController {

    @Autowired
    private UserMapper userMapper;

    @GetMapping("/queryUsers")
    public List<User> queryUsers(){
        return userMapper.queryUserList();
    }
}
```

9. 访问测试地址<http://localhost:8080/queryUsers>

## 5.2 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起步依赖
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);
}
```

## 5.3 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

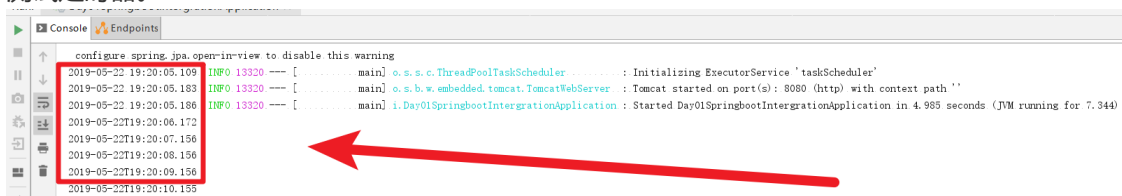
```

@Component
public class TimerUtil {

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

```

## 3. 测试定时器。



# 5.4 发送HTTP请求

## 1、服务调用方式：RPC和HTTP

常见远程调用方式：

- RPC：(Remote Produce Call)远程过程调用
  - 基于Socket、自定义数据格式、速度快，效率高。
  - 典型应用代表：Dubbo, WebService, ElasticSearch集群间互相调用。
- HTTP：网络传输协议
  - 基于TCP/IP、规定数据传输格式、缺点是消息封装比较臃肿、传输速度比较慢。
  - 优点是对服务提供和调用方式没有任何技术限定，自由灵活，更符合微服务理念。

RPC和HTTP的区别：RPC是根据语言API来定义，而不是根据基于网络的应用来定义。

## 2、Spring的RestTemplate

- RestTemplate是Rest的HTTP客户端模板工具类
- 对基于Http的客户端进行封装
- 实现对象与JSON的序列化与反序列化
- 不限定客户端类型，目前常用的3种客户端都支持：HttpClient、OKHttp、JDK原生URLConnection(默认方式)

## 3、RestTemplate案例

目标：发送Http请求

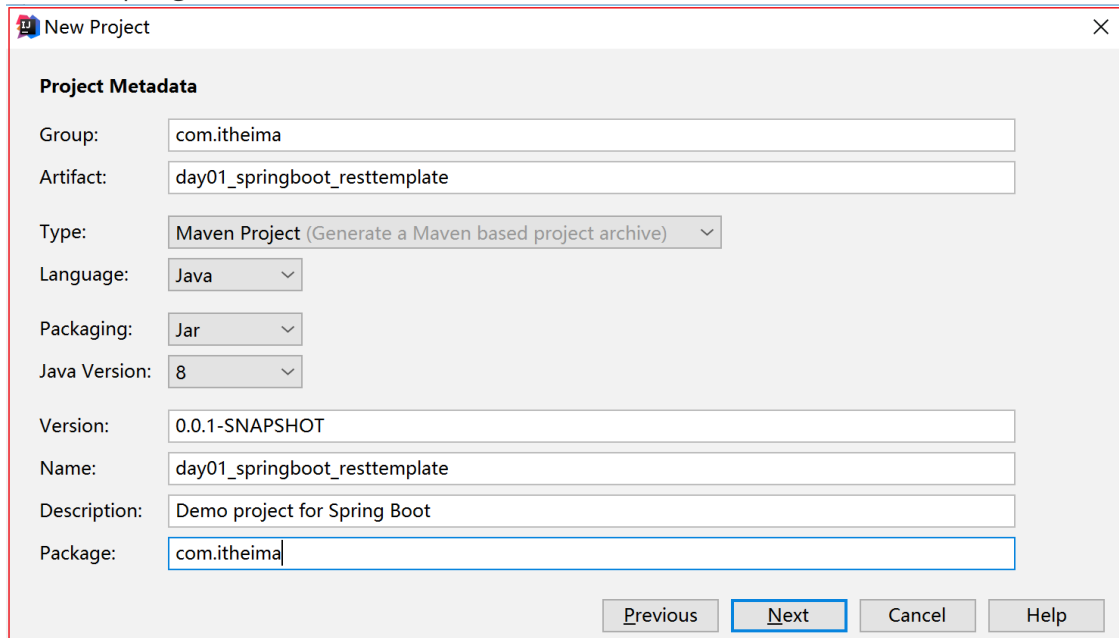


实现步骤：

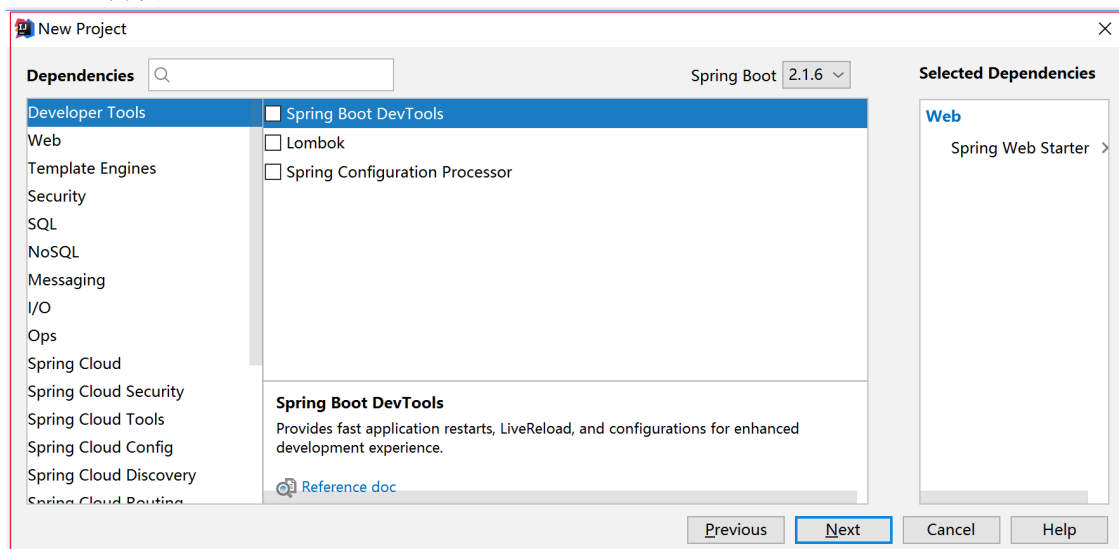
1. 创建一个springboot的工程
2. 配置RestTemplate的对象Bean到Spring容器中
3. 在测试类中用@Autowired注入Spring容器中的RestTemplate对象
4. 通过RestTemplate对象的getForObject发送请求
5. 运行测试类的测试方法

实现过程：

1. 创建一个springboot的工程，勾选Web的Starter



勾选web开发的Starter



2. 在项目启动类位置中注册一个RestTemplate对象

```
@Configuration
public class MyConfiguration {

    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

3. 在测试类ApplicationTests中 @Autowired 注入RestTemplate
4. 通过RestTemplate的getForObject()方法，传递url地址及实体类的字节码

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

    @Autowired
    private RestTemplate restTemplate;

    @Test
    public void testREST() {
        String url = "http://baidu.com";
        String json = restTemplate.getForObject(url, String.class);
        System.out.println(json);
    }
}
```

- RestTemplate会自动发起请求，接收响应
- 并且帮我们对响应结果进行反序列化

5. 运行测试类中的testREST方法;



```
Tests passed: 1 of 1 test - 312 ms
2019-08-01 10:44:13.762 INFO 8016 --- [main] 01SpringbootResttemplateApplicationTests :
2019-08-01 10:44:15.399 INFO 8016 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor :
2019-08-01 10:44:15.821 INFO 8016 --- [main] 01SpringbootResttemplateApplicationTests :
<html>
<meta http-equiv="refresh" content="0;url=http://www.baidu.com/">
</html>
2019-08-01 10:44:16.163 INFO 8016 --- [Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor :
```

## 5.5 扩展了解：除此之外还可以整合什么？

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

# 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 UserMapper userMapper;

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

}
```

### 3. 控制台打印信息

```
Tests passed: 1 of 1 test = 841 ms
2019-05-22 19:02:34.504 INFO 16352 --- [main] o1SpringbootIntergrationApplicationTests : Started Day01SpringbootIntergrationApplicationTests in 5.7
for 7.64s)
2019-05-22 19:02:34.986 INFO 16352 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2019-05-22 19:02:35.206 INFO 16352 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
[User(id=1, username='zhangsan', password='123', name='张三'), User(id=2, username='lisi', password='123', name='李四')]
2019-05-22 19:02:35.397 INFO 16352 --- [Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'
2019-05-22 19:02:35.399 INFO 16352 --- [Thread-2] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2019-05-22 19:02:35.411 INFO 16352 --- [Thread-2] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

Process finished with exit code 0
```

## 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/day01_springboot_demo01-1.0-SNAPSHOT.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/day01_springboot_demo01-1.0-SNAPSHOT.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/day01\\_springboot\\_demo01-1.0-SNAPSHOT/hello](http://localhost:8080/day01_springboot_demo01-1.0-SNAPSHOT/hello)