SpringBoot

主讲：崔译

一、简介

> Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
>
> stand-alone: 独立：创建一个SpringWebMVC应用，这个应用可以 独立运行：
>
> ```
> 1.  不借助任何第三方组件（tomcat），内置了tomcat
> ```
>
> just run：通过main方法启动服务器
>
> 1. 将传统的J2EE 项目（war包） 变成了 J2SE 项目（jar 包）
> 2. JSP 将不再内置支持

二、优势

- Create stand-alone Spring applications

  创建独立的Spring 应用程序

- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)

  内置Servlet容器（不需要发布war包）

- Provide opinionated 'starter' dependencies to simplify your build configuration

  提供可选的`starter/启动器` 坐标 来 简化Maven 配置

  SpringBoot 提供了很多`starter`. 这些`starter` 分别配置了需要的各种依赖，程序员在使用时，只需要提供`starter`的坐标即可

- Automatically configure Spring and 3rd party libraries whenever possible

  默认提供了大量的`Spring`和`第三方jar 包`的 基础配置

- Provide production-ready features such as metrics, health checks and externalized configuration

  提供了 准生产环境 下的特性： 运行时监控，健康检查和 外部配置

- Absolutely no code generation and no requirement for XML configuration

  没有冗余代码生成（轻量级/非侵入式），无需配置XML 文件

  - 不是说没有配置文件，只是说没有XML配置文件
  - 也可以配置XML文件（**老项目的移植**）

三、HelloWorld

1、检查开发环境

- JDK8+
  - 编译版本
    - 项目的jdk
    - 项目的编译版本
      - IDEA 设置
      - Maven的设置

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

  - 运行版本

```
java -version
```

- Spring 5.0.8+
- Maven 3.2+
- Tomcat 8.5+
- IDEA 2013+

2、创建maven quickstart 项目

3、配置POM文件

3-1 配置父项目

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.4.RELEASE</version>
</parent>
```

3-2 配置starter坐标

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

3-3 配置Maven 插件 (可选)

```
 <!-- Package as an executable jar  Maven插件，将项目打包成可执行的jar包-->
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

4、编写Controller

```
@RestController
@RequestMapping("/hello")
public class HelloController {
    @RequestMapping("/f1")
    public Map f1()
    {
        Map map = new HashMap();
        map.put("abc","1233");
        map.put("admin","456");
        return map;
    }
}
```

5、编写主程序

```
// 变成SpringBoot 主程序类，默认扫描当前类所在的包以及其子包
@SpringBootApplication
public class ApplicationMain {

    public static void main(String[] args) {
        // 参数1： 当前类的class对象
        // 参数2： main 方法的参数
        SpringApplication.run(ApplicationMain.class,args);
    }

}
```

四、快速构建SpringBoot

1、进入 SpringBoot Initializer

根据页面提示信息，创建项目。

2、使用IDEA创建SpringBoot

2-1 新建模块

- 选择Spring Initializer
- jdk 选择1.8+

2-2 填写基本信息

2-3 选择`starter`

`Web` `MySql` `MyBatis` `ThymeLeaf`

2-3 目录结构

```
项目
|---- .mvn  maven 相关配置  可删除
|---- src
    |--- main
        |--- java 写java类
        |--- resources 放配置文件
            |--- static 存放静态资源(css/js/images), 相当于webroot
            |--- templates 存放模板页 相当于web-inf
                           用于存放ftl 或者  html
                           SpringBoot没有内置支持、也不建议使用JSP
            |--- application.properties /  application.yml
                           SpringBoot的配置文件，相当于ApplicationContext.xml
    |--- test 测试类  可删除
|---- .gitignore  git 的忽略文件（git/github） 相关  可删除
|---- mvnw  mvn 配置脚本 可删除
|---- mvnw.cmd mvn 配置脚本 可删除
|---- pom.xml
|---- 项目名.iml  IDEA的配置文件
```

五、关于`application.properties`

- 是SpringBoot的配置文件
- 相当于`applicationContext.xml`
- 文件名必须叫做`application.properties`
- 放在classpath下（resources）中或者 classpath:/config(resources/config) 文件夹下
- 配置项

```
# 配置数据源
spring.datasource.url=jdbc:mysql://localhost:3306/films
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
# MyBatis整合
mybatis.type-aliases-package=com.itany
mybatis.mapper-locations=classpath:mapper/*Mapper.xml

server.port=8080
# DispatcherServlet 的 url-pattern
server.servlet.path=/
# request.contextPath
server.servlet.context-path=/

server.tomcat.uri-encoding=utf-8
```

完整配置项，参考官方文档

六、关于`application.yml`

是 `application.properties` 的 替代品， 两者可以相互替代，建议使用 yml

- 是SpringBoot的配置文件
- 相当于 `applicationContext.xml`
- 文件名必须叫做 `application.yml`
- 放在classpath下（resources）中或者 classpath:/config(resources/config) 文件夹下
- **是一种专门 用于 配置文件 的语法**
- 文件名后缀 可以是 `yml` 或者 `yaml`
- 语法要求
  - 大小写敏感
  - 使用 缩进 表示 层级关系

    ```
    # <bean class="User"> <property name="address"></property> </bean>
    User:
       address:"南京"
    password:111
    ```

  - 缩进 **不允许** 使用 `tab` 键 ， 只能用空格
  - 空格的数量 不重要， 只要 对齐
  - 注释：`#xxxx`

```yaml
# 数据源
spring:
  datasource:
    username: root
    password: root
    url: jdbc:mysql://localhost:3306/films
    driver-class-name: com.mysql.jdbc.Driver

# 服务器相关配置
server:
  port: 8080
  servlet:
    context-path: /
  tomcat:
    uri-encoding: utf-8
```

允许自定义属性（yml/properties）

```
user:
  laoxie: 嘟嘟
  address: bb
  // 数组， 集合 ， map 等注入方式
```

```
@Component
@ConfigurationProperties("user")
public class User {
    private String laoxie;
    private String address;
}
```

yml 自定义属性 VS **PropertyPlaceHolderConfigurer**

- PropertyPlaceHolderConfigurer 定义Properties文件，有properties文件的所有的局限性
  - 中文支持
  - 层级关系
- PropertyPlaceHolderConfigurer 支持单属性注入，yml 支持整个对象注入

七、导入Spring配置文件

SpringBoot 支持使用Spring 的xml配置

```
@ImportResource("classpath:applicationContext.xml")
public class Springboot2Application {
    public static void main(String[] args) {
        SpringApplication.run(Springboot2Application.class, args);
    }
}
```

八、支持Spring/SpringMVC 所有使用方式

```
@Configuration
public class MVCConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/showHello").setViewName("hello");
    }
}
```

九、thymeleaf模板引擎

1、导入`starter`

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

2、在SpringBoot中配置thymeleaf

```
spring:
  thymeleaf:
    cache: false
```

3、编写模板页

- 模板页放在`templates`中，相当于`WEB-INF` ，无法直接访问
  - 使用Controller 返回视图名
  - 使用ViewController
- thymeleaf 会自动的拼接前缀和后缀

```
public class ThymeleafProperties {
    private static final Charset DEFAULT_ENCODING;
    public static final String DEFAULT_PREFIX = "classpath:/templates/";
    public static final String DEFAULT_SUFFIX = ".html";
}
```

4、基础语法

```
<!DOCTYPE html>
<!--添加thymeleaf的命名空间：有提示-->
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        window.onload = function(){
            console.log(document.getElementById("laoxie"))
        }
    </script>
</head>
<body style="padding-bottom: 100px">

    <th:block th:include="footer.html :: header"></th:block>

    <h1>取作用域中的值</h1>
    <h4>[[${msgHTML}]]:不解析标签</h4>
    [(${msgHTML})]:解析标签
    <!--相当于 jquery text()-->

    <h4 th:text="${msgHTML}"></h4>
```

```
<!--相当于 jquery html()-->
<h4 th:utext="${msgHTML}"></h4>

<h4 th:text="${user.laoxie}"></h4>
<h4 th:text="${user.test()}">${user.test()}</h4>
<h4 th:text="${user.test()+'aaaaaa'}"></h4>


<!--
    可以使用th:object 指定对象
        th:object="${作用域中的key}"
     使用 th:text 取对象中的属性
        th:text="*{属性名}"
-->
<table width="100%" cellspacing="0" cellpadding="5" border="1">
    <tr>
        <td>姓名</td>
        <td>地址</td>
    </tr>
    <tr th:object="${user}">
        <td th:text="*{laoxie}"></td>
        <td th:text="*{address}"></td>
    </tr>
</table>

<!--
    每个HTML属性abc 都有一个对应的 th:abc
    只有th:xx 可以解析取值表达式
-->
<div id="${msgHTML}" >aa</div>
<div th:id="${msgHTML}" >bb</div>

<!--关于绝对路径
    @{} : request.contextPath
-->
<a th:href="@{/hello/f1}">f1......</a>


<h1>判断</h1>

<div th:if="${user.laoxie != 'laoxie'}">不等于</div>
<div th:if="${user.laoxie == 'laoxie'}">等于</div>


<h1>循环</h1>
<table width="100%" cellspacing="0" cellpadding="5" border="1">
    <tr>
        <td>编号</td>
        <td>姓名</td>
        <td>地址</td>
    </tr>
    <!--

        c:foreach items="${data}" var="item" varStatus="status"
```

```
        -->
        <tr th:each="item,status:${data}">
            <td th:text="${status.index + 1}"></td>
            <td th:text="${item.laoxie}"></td>
            <td th:text="${item.address}"></td>
        </tr>
    </table>


    <div th:text="${#numbers.sequence(1,10)}"></div>

    <ul>
        <li th:each="item:${#numbers.sequence(1,10,2)}" th:text="${item}"></li>
    </ul>

    <!--用于表单对象，绑定 id  name  value-->
    <form th:action="@{/th/add}" method="post" th:object="${user}">
        <input type="text" th:field="*{laoxie}">

        <input type="text" th:field="*{address}">

        <select name="" id="">
            <option th:each="a : ${#numbers.sequence(10,20)}"
                    th:value="${a}"
                    th:text="${a+'岁'}"></option>
        </select>

        <input type="submit" value="ssss">
    </form>


    <h1 th:text="${date}"></h1>
    <h1 th:text="${#dates.format(date)}"></h1>
    <h1 th:text="${#dates.format(date,'yyyyMMdd')}"></h1>


    <h1>th:block</h1>
    <span>是一个标签，无具体意义的标签</span>

    <th:block th:text="${msgHTML}"></th:block>

    <h1>引入其他html</h1>
    <th:block th:include="footer.html :: someFooter">

    </th:block>

</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>
```

```
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <th:block th:fragment="someFooter">
        <div style="text-align: center">
            &copy; copyright 2018 ~
        </div>
    </th:block>

    <th:block th:fragment="header">
        <div style="height: 50px;background-color: aquamarine;
                    position: fixed">
            top........
        </div>
    </th:block>
</body>
</html>
```

十、整合MyBatis

1、添加依赖

```
<dependency>
   <groupId>org.mybatis.spring.boot</groupId>
   <artifactId>mybatis-spring-boot-starter</artifactId>
   <version>1.3.2</version>
</dependency>
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper-spring-boot-starter</artifactId>
    <version>1.2.7</version>
</dependency>
```

2、配置文件

```
mybatis:
  type-aliases-package: com.itany.entity
  mapper-locations: classpath:mapper/*Mapper.xml
```

3、扫描dao

```
@Configuration
@MapperScan("com.itany.dao")
public class MyBatisConfig {
}
```