SpringBoot配置文件 1课时

最开始jsp， 原始开发方式， 我们使用JSP开发网站没问题

不适合开发大型应用，jsp专门为开发网站设计的， jsp的中心在渲染网页

Spring SpringMVC就出现了

有了SpringMVC 和 Spring 框架， 我们能够写很复杂的应用了， 应用分层， MVC分层

纯面向对象开发了， 他的这个缺点就是 配置复杂

全世界对SPring框架贡献前三的公司， 有一个是中国互联网公司， 阿里巴巴

eleme Python 转 Java， 七牛 go Java， 腾讯 C++ Java， 阿里巴巴 Java， Java辅助工具库

世界的尽头就是 Java

开箱即用

把 jsp Servlet Spring SpringMVC Tomcat 进行了一个整合

又增加了一些新的 特性， 最终出炉的就是 SpringBoot

1. 解决了 配置复杂的问题
    a. Spring SpringMVC xml 多个配置文件， 配置文件又是必须的， Springboot 可以没有复杂的配置文件
2. 解决了 版本依赖问题
    a. 我们整合第三方应用， 需要大量的操作
    b. Maven 依赖操作， Springboot 解决了 这个问题
        i. 很多配置不需要输入版本号了
        ii. 从而解决了 组件兼容问题

- properties 文件配置
- yaml文件配置
- yml文件配置
- 多Profile文件
    - 激活指定的profile
    - 1、在配置文件中指定spring.profiles-active=dev
    - 2、命令行参数绑定方式 --spring.profiles.active=dev 命令行 jar包运行时 指定环境参数
    - 3、虚拟机参数绑定方式 -Dspring.profiles.active=dev
    - 我们在主配置文件编写的时候，文件名可以是 application-{profile}.properties/yml 默认使用的 application.properties/yml的配置;
- 如何配置一个springboot

```
<parent>
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.8</version>
</parent>
```

- 数据库操作演示

```
spring.datasource.url=jdbc:mysql://127.0.0.1:13306/test?characterEncoding=utf-8&useSSL=false&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.password=0JZBdtlYoiOepddh
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

SpringBoot特性：

1. SpringBoot Parent 特性

   统一由父组件提供版本控制

2. SpingBoot Starter 特性

   自动处理依赖关系， 自动装配Bean对象

3. SpringBoot 统一配置档

   所有的组件 配置都在Springboot 配置档内

SpringBoot starter 2课时

- https://docs.spring.io/spring-boot/docs/1.5.9.RELEASE/reference/htmlsingle/#using-boot-starter
- 官方的`stater` 命名规范： `spring-boot-starter-模块名`
- 非官方`stater` 命名规范： `模块名-spring-boot-starter`
- 只有 `Stater` 版本才能够实现 自动装配功能！！
- `stater` 在框架启动的过程中， 就会被创建成 `bean`对象， 并且注入依赖参数！！
- @EnableAutoConfiguration 这个注解的功能时 启动自动装配功能
  - 它能够执行一段代码， 完成组件的 初始化功能
- spring配置方式的进化过程：
  - xml的方式配置
  - 使用@configuration注解
  - 自动装配
- springboot starter作用：
  - 引入模块所需的相关jar包
  - 自动配置各自模块所需的属性

自定义一个Stater步骤

1. 创建一个stater的模块， 这个模块可以没有任何代码， 但是需要包含当前stater所有的依赖项
2. 矿建一个自动装配的模块， 需要实现自动装配功能!
    a. 第一我们需要实现一个 参数装配类
    b. 我们需要实现一个业务组装类
    c. 我们还需要开发一个业务类

开发自定义自动装配

主要对核心注解进行说明

- Spring1.0：刚刚出现注解。
    - @Transaction：简化了事务的操作
- Spring2.0：一些配置开始被xml代替，但是还不能完全摆脱xml，主要是component-scan标签。
    - @Required：用在set方法上，如果加上该注解，表示在xml中必须设置属性的值，不然就会报错。
    - @Aspect ： AOP相关的一个注解，用来标识配置类。
    - @Autowired，@Qualifier：依赖注入
    - @Component，@Service，@Controller，@Repository：主要是声明一些bean对象放入IOC中。
    - @RequestMapping： 声明请求对应的处理方法
- Spring3.0：已经完全可以用注解代替xml文件了
    - @Configuration：配置类，代理xml配置文件
    - @ComponentScan：扫描其他注解，代理xml中的component-scan标签。
    - @Import：只能用在类上，主要是用来加载第三方的类。
        - @import(value = {XXX.class})：加载一个普通的类
        - @Import(MyImportSelector.class)：这种主要是根据业务选择性加载一些类。

1. 开发启动项目依赖包

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

2. 开发配置文件类

```
@Component
@ConfigurationProperties(prefix="demo.text")
```

```
public class HelloWorldProperties {
    private String greeting;
    public String getSay() {
        return greeting;
    }
    public void setSay(String greeting) {
        this.greeting = greeting;
    }
}
```

3. 开发业务接口

```
public interface HelloWorld {
    void say();
}
```

4. 开发实现业务接口

```
public class HelloWorldImpl implements HelloWorld {
    public HelloWorldProperties getHelloWorldProperties() {
        return helloWorldProperties;
    }
    public void setHelloWorldProperties(HelloWorldProperties helloWorldProperties) {
        this.helloWorldProperties = helloWorldProperties;
    }
    HelloWorldProperties helloWorldProperties;
    @Override
    public void say() {
        System.out.println( helloWorldProperties.getSay() );
    }
}
```

5. 开发自动装配类

```
@Configuration
@EnableConfigurationProperties(HelloWorldProperties.class)
public class HelloWorldConfgration {
    @Autowired
    HelloWorldProperties helloWorldProperties;
```

```
    @Bean
    public HelloWorld getHelloWorld(){
        HelloWorldImpl helloWorld = new HelloWorldImpl();
        helloWorld.setHelloWorldProperties(helloWorldProperties);
        return helloWorld;
    }
}
```

6. 开发配置文件

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
  com.lovecoding.demo.HelloWorldConfgration
```

SpringBoot parent

统一由 spring-boot-dependencies 关闭版本依赖

SpringBoot Mock 3课时

Springboot 官方文档地址

https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.testing.spring-boot-applications

SpringMVC 官方文档地址

https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html#spring-mvc-test-server

```

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

@AutoConfigureMockMvc
@SpringBootTest
OR
@WebMvcTest(DemoController.class)
when(testMapper.counter()).thenReturn(300L);
MvcResult mvcResult = mockMvc.perform(
```

```
                MockMvcRequestBuilders.request(HttpMethod.GET, "/")
                        .contentType(MediaType.APPLICATION_JSON_VALUE)
        )
        .andExpect(MockMvcResultMatchers.status().is(200))
        .andExpect(MockMvcResultMatchers.content().string("2"))
        .andExpect(MockMvcResultMatchers.jsonPath("$.data.code").value(200))
        .andDo(MockMvcResultHandlers.print())
        .andReturn();
String contentAsString = mvcResult.getResponse().getContentAsString();
log.info( contentAsString );
```

官方文档
https://docs.spring.io/spring-framework/docs/5.1.8.RELEASE/spring-framework-reference/testing.html#webtestclient

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <scope>test</scope>
</dependency>
@WebFluxTest
webClient.get().uri("/test").exchange().expectStatus().isEqualTo(300);
```

YAML语法简介 1、大小写敏感 2、使用缩进表示层级关系 3、禁止使用tab缩进，只能使用空格键 4、缩进长度没有限制，只要元素对齐就表示这些元素属于一个层级 5、使用#表示注释 6、字符串可以不用引号标注

SpringBookSwagger 4课时

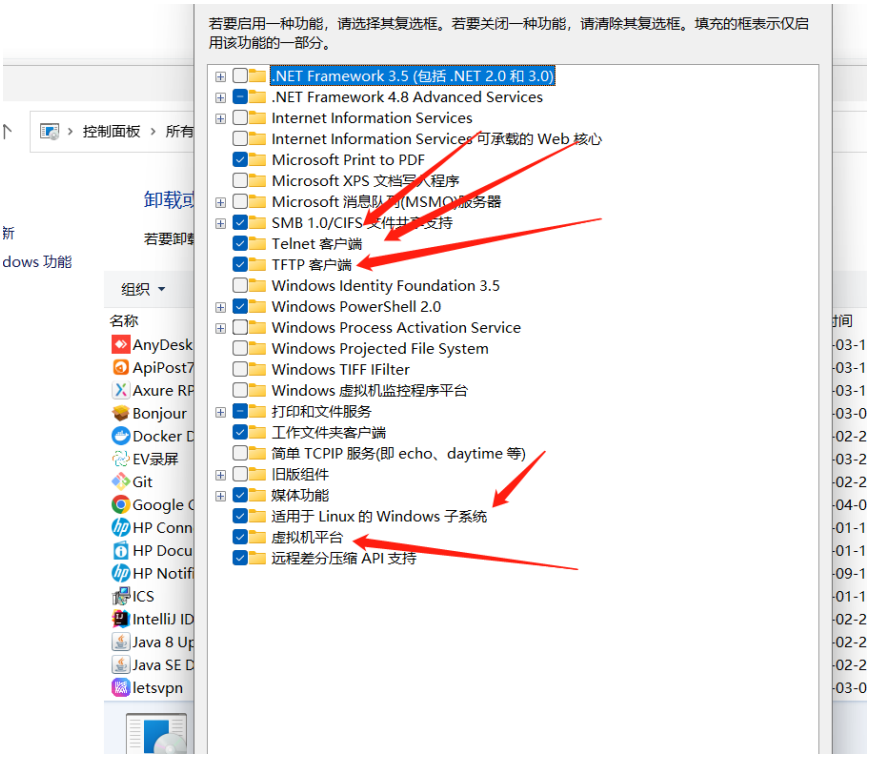
```
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
```

```
    <version>2.6.1</version>
</dependency>
```

Docker 安装与简单使用

docker 安装失败的几种问题

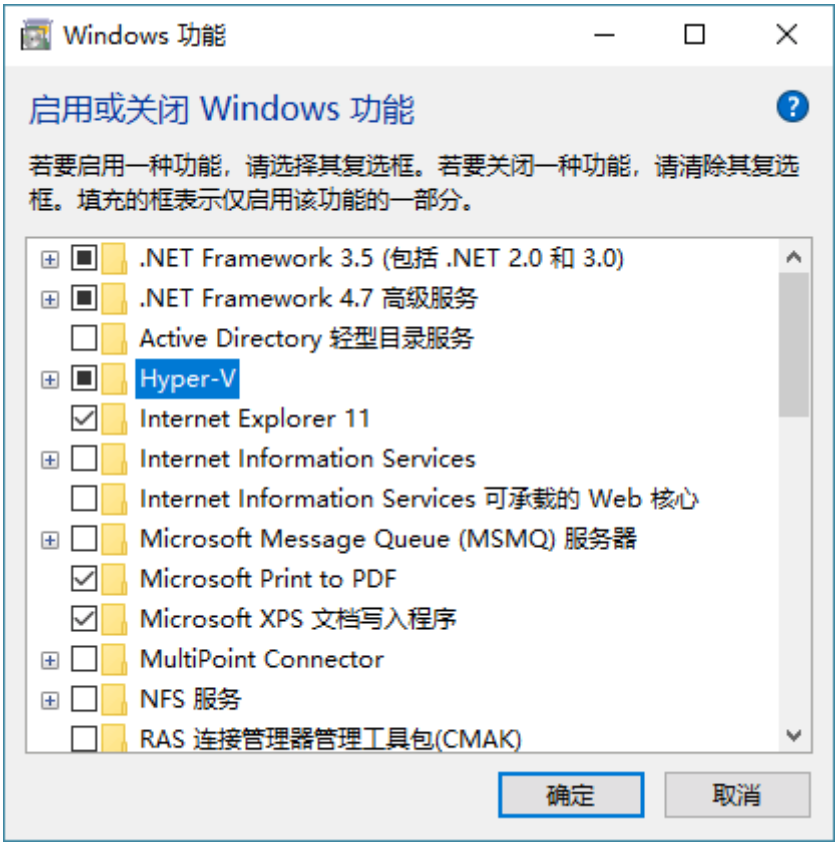
1. 你的硬件不支持 （1%）， CPU不支持虚拟化
2. 第二种情况， 你的硬件支持， 但是主板没有启用 （10%）
3. 第三种情况， 你的window系统没有开启虚拟化支持



安装前的检查操作

首先，读者需要确认在 Windows 10 操作系统中，Hyper-V 和容器特性已安装并且开启。

1. 右键单击 Windows 开始按钮并选择“应用和功能”页面。
2. 单击“程序和功能”链接。
3. 单击“启用或关闭Windows功能”。
4. 确认Hyper-V和容器复选框已经被勾选，并单击确定按钮。



下载地址 https://www.docker.com/products/docker-desktop

如果安装不成功检查问题方案

& "C:\Program Files\Docker\Docker\resources\com.docker.diagnose.exe" check