Part I. 基础回顾

1. 约定优于配置

- 什么是SpringBoot?

  它是 所有基于Spring开发的项目的 起点。SpringBoot的设计是为了让你尽可能快的跑起来 Spring 应用程序并且尽可能减少自定义配置文件。

- 为什么能减少自定义配置文件编写?

  因为SpringBoot遵循 约定优于配置， 又称按约定编程（是一种软件设计范式）。例如，在使用SpringData JPA时，程序中有User类，数据库中对应的表会默认命名为User，如果表名和实体类名称不一致时，才有必要写有关这个名字的配置。

2. SpringBoot 概念

2.1 Spring 的优缺点

- 优点：Spring组件代码是轻量级的，通过依赖注入和AOP，用简单的Java对象（POJO）来实现业务代码功能。
- 缺点：
  - 配置是重量级的。例如各种显式XML配置
  - 项目的依赖管理耗时耗力 。在环境搭建时，考虑要导入哪些dependency，还要分析导入与之有依赖关系的其他dependency，由于版本众多，有不兼容的风险。

2.2 SpringBoot解决了上述Spring的缺点

- 起步依赖

  本质上是一个Maven项目对象模型（Project Object Model，POM），定义了对其他库的传递依赖，即将具备某种功能的dependency们打包到一起，并提供一些默认的功能。

- 自动配置

  SpringBoot 会自动将一些配置类的bean注册进IoC容器，程序中若要用，直接用 @Autowired 或者 @Resource 注解引用即可。（这样就不用在 applicationContext.xml 中写一堆 标签）

3. SpringBoot 案例实现

需求：请求 Controller 中的方法，并将返回值 response 到页面。

Step 1. 使用 Spring Initializr 来创建 SpringBoot 项目

Step 2. 创建一个用于Web访问的 Controller

【注意】要把 `Springboot01DemoApplication` 主程序启动Class 移动到 com.lagou pkg下面，因为SpringBoot只会扫描 主程序启动Class 所在到pkg以及这个pkg下面的子pkg，这样才能扫描到 Controller pkg。

Step 3. 运行 `Springboot01DemoApplication` 主程序启动Class中的 main function

4. 单元测试与热部署

4.1 Unit Test

```
@RunWith(SpringRunner.class)  // 测试启动器，并且 加载Spring boot测试注解
@SpringBootTest  // 标记该class是springboot单元测试类 并且 加载项目的applicationContext上下文环境
class Springboot01DemoApplicationTests {

    @Autowired
    private HelloController helloController;

    @Test
    void contextLoads() {
        String result = helloController.demo();
        System.out.println(result);
    }

}
```

4.2 热部署

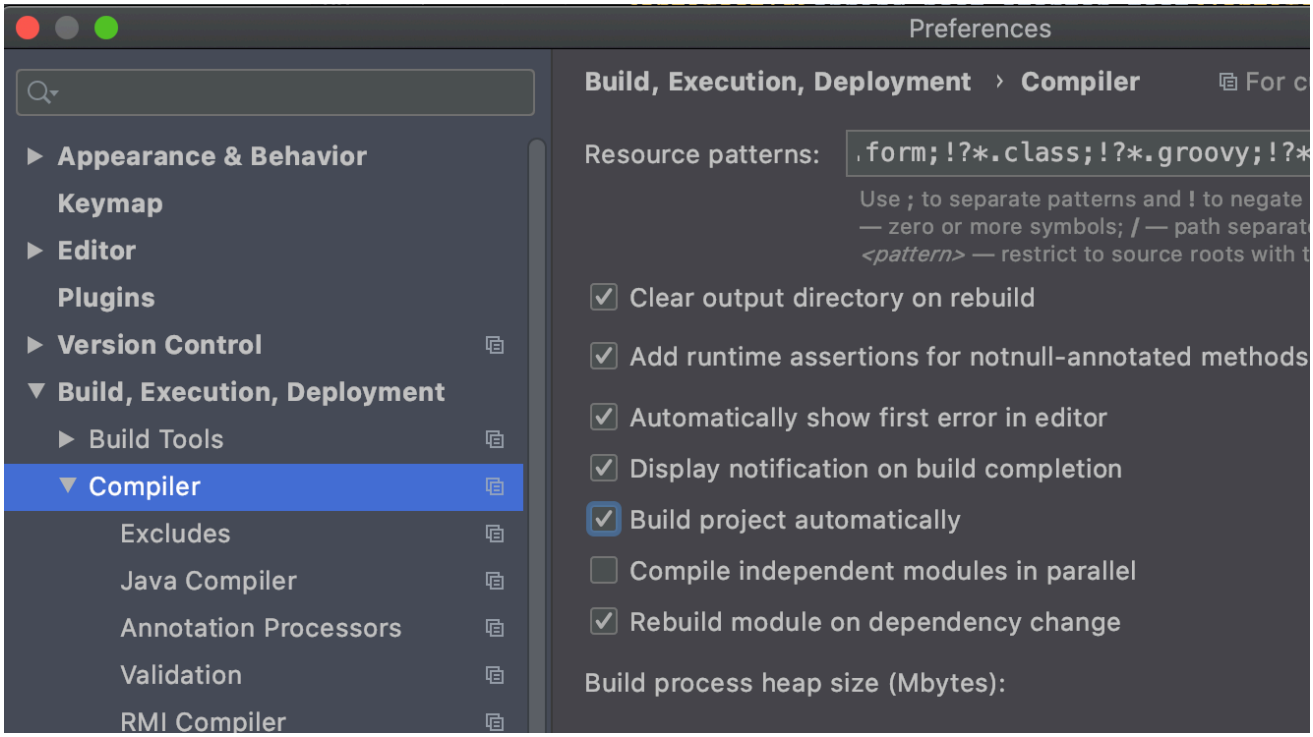
SpringBoot 提供了进行热部署的依赖启动器，这样，每次修改代码后，不需要再手动重启项目，就能直接在browser中看到更改结果。

Step 1. 添加热部署依赖在pom 配置文件中

```
<!-- 引入热部署依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Step 2. IDEA 设置

【File】/ 【Settings】，打开 Compiler：



Step 3. “Ctrl+Shift+Alt+/“ 打开 Maintenance 选项框，选中并打开 Registry 页面，勾选 ”compiler.automake.allow.when.app.running“，然后colse。这个可以用于指定IDEA工具在程序运行过程中自动编译。

5. 全局配置文件

全局配置文件（application.properties 或者 application.yaml ）可以对一些默认配置值进行修改，这个文件存放在resource folder中。

5.1 application.properties 文件

可以定义 系统属性、环境变量、命令参数等，也可以是自定义配置文件名称和位置

下面演示 如何在application.properties 中将自定义属性注入到Person class中。

Step 1. Create Person class, Pet class

```
@Component
@ConfigurationProperties(prefix = "person")  // 将配置文件中，以person开头的属性值注入到这个Person class中
public class Person {

    private int id;
    private String name;
    private List hobby;
    private String[] family;
    private Map map;
    private Pet pet;
  ...
```

```
public class Pet {

    private String type;
    private String name;
```

Step 2. 在 application.properties 中给person到属性赋值，并用注解把值注入到实体class中

```
server.port=8081

person.id=1
person.name=Jenny
person.hobby=gym,music
person.family=mother,father
person.map.myKey1=myValue1
person.map.key2=value2
person.pet.type=dog
person.pet.name=Wangcai
```

Step 3. Test:

```
Person{id=1, name='Jenny', hobby=[gym, music], family=[mother, father], map=
{key2=value2, myKey1=myValue1}, pet=Pet{type='dog', name='Wangcai'}}
```

5.2 application.xml 文件

Yaml 文件格式与JSON类似，比传统的properties配置文件更 reader-friendly。扩展名可用 .yml or .yaml，文件中使用 “key:(space)value” 格式来配置属性，使用缩进来控制层级关系。

- value值为普通数据类型（例如数字、字符串、boolean等）-- 只用 “Enter” 和 “Space” 键即可

```yaml
server:
 port: 8081
 path: /hello
```

- value值为数组

```yaml
person:
 hobby:
  - play
  - read
  - sleep
```

或者用：

```yaml
person:
 hobby:
  play,
  read,
  sleep
```

也可以用 in-line 写法：

```yaml
person:
 hobby: [play,read,sleep]
```

- Value 值为map或者对象

```yaml
person:
 map:
  myKey1: myValue1
  key2: value2
```

也可以用 in-line 写法：

```
person:
 map: {myKey1: myValue1,key2: value2}
 pet: {type: dog,name: Wangcai}
```

案例改造：使用 application.yml 来配置person

```
person:
  id: 2
  name: Lisa
  hobby: [gym,music]
  family: [mother,father]
  map: {myKey1: myValue1,key2: value2}
  pet: {type: dog,name: Wangwang}
```

Test：

```
Person{id=2, name='Lisa', hobby=[gym, music], family=[mother, father], map=
{key2=value2, myKey1=myValue1}, pet=Pet{type='dog', name='Wangwang'}}
```

【注意】 在 spring-boot-starter-parent 中, 配置文件的加载顺序是： `application*.yml` `application*.yaml` `application*.properties` 所以 .properties 会覆盖掉之前对person的设置，把其注释掉即可。

6. 配置文件属性值的注入

在配置文件中，若配置的是SpirngBoot已有属性，如 server.port，那么SpringBoot内部会自动扫描并读取新的值来覆盖默认值。

若配置的是用户自定义的属性，如Person实体类，那么需要手动注入这些属性值。注入方式可使用注解 @ConfigurationProperties 或者 @Value

6.1 @ConfigurationProperties

- 一种批量注入方式，其背后是通过属性的setter方法来实现。

6.2 @Value （from Spring framework）

- 单个注入，且无需使用属性的setter方法来实现。
- 不支持 在map、对象以及yaml文件中用 in-line 写法的属性赋值，会报错

```
@Component
public class Student {

    @Value("3")  // 这里可以直接给id赋值，但这种用法很没必要
    private int id;
    @Value("${person.name}")  // 这里的 person 是yaml文件中的person
```

7. 自定义配置

7.1 @PropertySource

如何让 SpringBoot 能认识并加载 `mytest.properties` ？ 使用 `@PropertySource` 注解。

```
# mytest.properties
test.id=8
test.name=Peter
```

```
@Component
@PropertySource("classpath:mytest.properties")  // 引入 自定义配置文件的名称及位置
@ConfigurationProperties(prefix = "test")  // 这个 test 是mytest.properties中的test
public class MyProperties {

    private int id;
    private String name;
```

Test:

MyProperties{id=8, name='Peter'}

7.2 @Configuration

用于表明配置类，SpringBoot能扫描到。

```
@Configuration  // 标明该类为配置类
public class MyConfig {

    @Bean(name = "myFavService") // 把当前方法return的对象 作为一个Bean添加到Spring IOC容器中，id默认为方法名称 如果Bean中无name的话
    public MyService myService() {
        return new MyService();
    }
}
```

8. 随机数设置及参数间的引用

8.1 随机值设置属性值

- 使用场景：给一些隐秘属性值 或者 测试用例属性值 注入
- 实现：SpringBoot 内嵌的RandomValuePropertySource class
- 语法格式：${random.xx}, xx 表示指定生成的随机数类型及范围，可以生成整数、uuid或字符串：

```
my.secret=${random.value}
my.number=${random.int}
my.bignumber=${random.long}  // 随机long型整数
my.uuid=${random.uuid}
my.number.less.than.ten=${random.int(10)}   // 小于10的随机整数
my.number.in.range=${random.int[1024,65536]}   // 随机整数在区间[1024,65536]
```

8.2 参数间引用

即 后一个配置的属性 直接引用 先前已经定义过的属性。

- 使用场景：多处会使用同一个属性值，只需要对一处进行配置，以后修改也只改这一处，就把所有用到的地方都修改了
- 语法格式：

```
# 在 application.properties中，随机值设置 + 参数间引用
tom.age=${random.int[10,30]}
tom.description=Tom's age may be ${tom.age}
```

Test:

```
// 将 application.properties 中的值赋值过来
@Value("${tom.description}")
private String description;

@Test
void argumentsReferenceTest() {
    System.out.println(description); // Tom's age may be 28
}
```

随堂

关于 @ConfigurationProperties，其实他就是个configuration里面的property：

https://juejin.im/post/5d3e40ec51882551c37fc309