Spring Boot

Spring 时为了解决企业级应用开发的复杂性而创建的，简化开发

1、第一个Spring Boot程序

2、Spring Boot自动装配原理

精髓：

1、SpringBoot启动会加载大量的自动配置类（spring.factories）

2、我们根据我们需要的功能有没有在SpringBoot默认写好的自动配置类当中；

3、我们再来看这个自动配置类中到底配置了哪些组件；（只要我们要用的组件存在其中，我们就不需要手动配置了）

4、给容器中自动配置类添加组件的时候，会从xxxxProperties类中获取某些属性。我们只需要在配置文件中指定这些属性的值即可；

xxxxAutoConfigurartion:自动配置类；给容器中添加组件

xxxx Properties：封装配置文件中相关属性（通过Spring Boot配置 .yaml来配置属性）

重要

深入：

我们在application.yaml配置文件中能配置的东西，都存在这样一个固有的规律：

xxxAutoConfiguration:默认值 xxxProperties 和 application.yaml配置文件绑定，我们就可以使用自定义的配置了！！

步骤：

- 首先通过SpringBoot自动装配，spring.factories中的每一个xxxAutoConfiguration类都是容器中的一个组件，最后加入到容器中，用他们来自动配置；
- 每个自动配置类可以进行自动配置功能；**根据当前不同的条件判断，决定这个配置类是否生效；**
- 然后，通过@EnableConfigurationProperties(ServerProperties.class)以及ServerProperties.class中的@ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)与application.yaml配置文件进行绑定；
- 最后，就可以在application.yaml配置文件中自定义配置了

##

@Conditional注解

了解完自动装配的原理之后，我们关注一个细节，**自动配置类必须在一定条件下才能生效；@Conditional派生注解（Spring注解版原生的@Conditional作用）**

作用：必须是@Conditional指定条件成立，才给容器中添加组件，配置里面的所有内容才生效

@Conditional扩展注解	作用（判断是否满足指定条件）
@ConditionalOnBean	（仅仅在当前上下文中存在某个对象时，才会实例化一个Bean）
@ConditionalOnClass	（某个class位于类路径上，才会实例化一个Bean）
@ConditionalOnExpression	（当表达式为true的时候，才会实例化一个Bean）
@ConditionalOnMissingBean	（仅仅在当前上下文中不存在某个对象时，才会实例化一个Bean）
@ConditionalOnMissingClass	（某个class类路径上不存在的时候，才会实例化一个Bean）
@ConditionalOnNotWebApplication	（不是web应用）

3、Spring Boot配置

application.properties ===>application.yaml

- 更改端口号：server.port = 8081

```yaml
#yaml 对空格的要求十分高
#可以注入到我们的配置中
#普通的key-value
name: chamfers

#对象
student:
  name: chamfers
  age: 18
```

```yaml
#行内写法
student: {name: chamfers,age: 18}

#数组
pets:
  - cat
  - dog
  - pig
#行内写法
pets: [cat,dog,pig]
```

yaml：可以直接给我们的实体类赋值：

```
/*
@ConfigurationProperties注解作用：
将.yaml配置文件中的每一个属性的值，映射到这个组件中；
告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定
参数perfix = "person" :将配置文件中的person下面的属性一一对应

只有这个组件是容器中的组件，才能使用容器提供的@ConfigurationProperties注解功能
*/
@Component //注册bean
@ConfigurationProperties(perfix = "person")
public class Person{

}

```

JSR303校验

- @Validated注解//数据校验
    - @Email
    - @Length
    - @NotEmpty
    - @Range

多环境配置和配置文件位置

优先级：

- file:./config
- file:./
- classpath:/config
- classpath:/

多环境配置：

application.properties-->spring.profiles.active = test

Out:8081

application.properties-->spring.profiles.active = dev

Out:8082

```
#application-test.properties
server.port=8081

#application-dev.properties
server.port=8082
```

application.yaml

```
server:
  port: 8081
#激活版本
spring:
  profiles:
    active: dev #激活dev环境

---
server:
  port: 8082
spring:
  profiles: dev
---
server:
  port: 8083
spring:
  profiles: test
```

debug=true

```
#可以通过debug=true来查看，哪些配置生效，哪些没有生效
debug: true
```

Positive match:（自动配置类启用的：正匹配）

Negative match:（没有启动，没有匹配成功的自动配置类：负配置）

Uncoditional classes:（没有条件的类）

4、Spring Boot Web 开发

SpringBoot到底帮我们配置了什么？我们能不能进行修改？能修改哪些东西？能不能拓展？

- xxxxAutoConfiguration：向容器中自动配置组件
- xxxxProperties：自动配置类，装配配置文件中自定义的一些内容！

要解决的问题：

- 导入静态资源：html，css.....
- 首页
- jsp,模版引擎 Thymeleaf
- 装配扩展SpringMVC
- 增删改查
- 拦截器
- 国际化

4.1 静态资源

```
@Override
    protected void addResourceHandlers(ResourceHandlerRegistry registry) {
      super.addResourceHandlers(registry);
      //如果在applica.yaml中配置了之后，这个if就不会走了
      if (!this.resourceProperties.isAddMappings()) {
        logger.debug("Default resource handling disabled");
        return;
      }
      ServletContext servletContext = getServletContext();
      addResourceHandler(registry, "/webjars/**", "classpath:/META-
INF/resources/webjars/");
      addResourceHandler(registry,
this.mvcProperties.getStaticPathPattern(), (registration) -> {

registration.addResourceLocations(this.resourceProperties.getStaticLocatio
ns());
        if (servletContext != null) {
          registration.addResourceLocations(new
ServletContextResource(servletContext, SERVLET_LOCATION));
        }
      });
    }


private String staticPathPattern = "/**";
private static final String[] CLASSPATH_RESOURCE_LOCATIONS = {
"classpath:/META-INF/resources/",
"classpath:/resources/", "classpath:/static/", "classpath:/public/" };

/**
* Locations of static resources. Defaults to classpath:[/META-
INF/resources/,
```

```
* /resources/, /static/, /public/].
*/
private String[] staticLocations = CLASSPATH_RESOURCE_LOCATIONS;
```

总结：

1、在springboot中，我们可以使用以下防护处理静态资源

- web jars: localhost:8080/webjars/**
- Public,static,/**,resource 目录下:localhost:8080/**

2、优先级： >static(默认)>public

如何定制首页

4.2 模版引擎

结论：只要需要使用thymeleaf，只需要导入对应的依赖就可以了，我们将html放在我们的templates目录下即可！

```
public static final String DEFAULT_PREFIX = "classpath:/templates/";

public static final String DEFAULT_SUFFIX = ".html";
```

5、扩展Spring Boot配置

自定义Spring Boot配置时 **不要加'@EnableWebMvc'注解**

6、系统功能实现

6.1、首页配置

1、注意点：所有页面的静态资源都需要使用thymeleaf接管

2、url：@{}

6.2、页面国际化

1、需要配置i18n文件

2、如果需要在项目中进行按钮自动切换，需要自定义组件`LocaleResolver`

3、记得将自己的组件配置到Spring容器`@Bean`

4、message：#{}

6.3、登陆+拦截器

```
public class LoginHandlerInterceptor implements HandlerInterceptor {
```

6.4、增删改查

员工列表展示：

1、提取公共页面

```
th:fragment="sidebar"
```

```
<div th:replace="~{commons/common::topbar}"></div>
```

如果需要传递参数，可以直接使用（）获取参数

```
<div th:replace="~{commons/common::sidebar(active = 'main.html')}"></div>
```

2、列表循环展示

6.5、添加员工

1、提交按钮

2、跳转到添加页面

3、添加员工成功

4、返回首页

```
<div class="form-group">
    <label>Department</label>
  <!--name="department.id"-->
    <select class="form-control" name="department.id">
        <option th:each = "dept:${departmentList}"
th:text="${dept.getDepartmentName()}" th:value="${dept.getId()}"></option>

    </select>
</div>
```

前端：

- 模版
- 框架：bootstrap,layui,semantic-ui
    - 栅格

1、前端搞定：页面长什么样子：数据

2、设计数据库

3、前端让他能够自动运行，独立话工程

4、数据接口如何对接：json，对象 all in one！

5、前后端联调测试！

有一套自己熟悉的后台模版：工作必要！x-admin

2、前端界面：至少能够自己通过前端框架，组合出来一个网站页面

- index
- about
- blog
- post
- user

3、让这个网站能够独立运行

7、整合JDBC

application.yaml

```yaml
spring:
  datasource:
    username: root
    password: cf19971101
    url: jdbc:mysql://localhost:3306/mybatis?
useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
```

JDBCController.class

```
@RestController
public class JDBCController {
    @Autowired
    JdbcTemplate jdbcTemplate;
    @RequestMapping("/update/{id}")
```

```
    public String update(@PathVariable("id") Integer id){
        String sql = "update mybatis.user set name=?,pwd=?  where id =?;";
        Object[] objects = new Object[3];
        objects[0] = "CF";
        objects[1] = "111222";
        objects[2] = id;
        jdbcTemplate.update(sql,objects);
        return "update-ok";
    }
}
```

7.1、DruidDataSource数据源（德鲁伊）

```
spring:
  datasource:
    username: root
    password: cf19971101
    url: jdbc:mysql://localhost:3306/mybatis?useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource

    filters: wall,stat,log4j
```

```
@Configuration
public class DruidConfig {
    @ConfigurationProperties(prefix = "spring.datasource")
    @Bean
    public DataSource druidDatasource(){
        /**
         * 绑定application.yaml
         */
        return new DruidDataSource();
    }
    @Bean
    public ServletRegistrationBean statViewServletServletRegistrationBean(){
        /**
         * 后台监控功能
         */
        ServletRegistrationBean<StatViewServlet> bean = new ServletRegistrationBean<>(new StatViewServlet(),"/druid/*");

        HashMap<String,String> initParameter = new HashMap<>();
        //账号密码配置；必须用loginUsername、loginPassword
        initParameter.put("loginUsername","chamfers");
        initParameter.put("loginPassword","cf19971101");
```

```
        //其他配置
        initParameter.put("allow","");
        bean.setInitParameters(initParameter);//设置初始化参数
        return bean;
    }
    @Bean
    public FilterRegistrationBean bean(){
        //filter
        FilterRegistrationBean bean = new FilterRegistrationBean();
        bean.setFilter(new WebStatFilter());
        HashMap<String,String> initParameter = new HashMap<>();
        //这些不进行统计
        initParameter.put("exclusions","*.js,*.css,/druid/**");

        bean.setInitParameters(initParameter);
        return bean;
    }
}
```

8、SpringSecurity

在web开发中，安全第一位！过滤器、拦截器

非功能性需求

在设计之初就考虑安全

shiro、SpringSecurity：很像

- 认证
- 授权

- 功能权限
- 访问权限
- 菜单权限
- ……拦截器、过滤器：大量原生代码：冗余

认证授权：

1、读取内存中权限数据（auth.inMemoryAuthentication()）

```
@EnableWebSecurity
public class MySpringSecurity extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
      //以“/toEdit”开头的所有路由（URL地址）都会需要进行权限认证；其他路由不需要权限认证
```

```
     http.authorizeRequests().antMatchers("/").permitAll()
             .antMatchers("/toEdit/**").hasRole("vip1");
         http.formLogin();
     }

     @Override
     protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
         auth.inMemoryAuthentication().passwordEncoder(new
BCryptPasswordEncoder())
                 .withUser("chamfers").password(new
BCryptPasswordEncoder().encode("111111")).roles("vip1");
     }
}

```

2、读取数据库中权限数据

```
@Autowired
DataSource dataSource;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
//         auth.inMemoryAuthentication().passwordEncoder(new
BCryptPasswordEncoder())
//                 .withUser("chamfers").password(new
BCryptPasswordEncoder().encode("111111")).roles("vip1");
    auth.jdbcAuthentication()
            .dataSource(dataSource)
            .withDefaultSchema()

.withUser(user.username("user").password("111111").role("user"))

.withUser(user.username("admin").password("111111").role("user","admin"));
}
```