参考：<http://tengj.top/tags/Spring-Boot/>

**一、获取配置文件（properties）中的参数**

1、@Value(value=”${config.name}”)

在将要使用配置文件中属性的地方使用如上注解即可完成属性值的注入，config.name为属性名。例：

@Value("${com.dudu.name}")  
 private String name;

2、@ConfigurationProperties(prefix = “com.sunsir”)

创建一个与配置文件相对应的配置类，使用如上注解即可将配置属性与类成员变量一一对应注入。com.sunsir为配置属性的前缀。

@ConfigurationProperties(prefix = "com.dudu")  
public class ConfigBean {  
 private String name;  
 private String want;  
  
 // 省略getter和setter  
}

注：此时需要在spring boot入口类上添注解@EnableConfigurationProperties({ConfigBean.class})。

如此配置之后，configbeen即可像常规been一样通过注解@Autowired完成依赖注入。

注：以上方式配置的configbeen为application.properties默认配置文件对应的配置类。当需要自定义配置文件时（例：tes.properties），则配置对应的configbeen需要在配置类上添加注解@Configuration和@PropertySource(“classpath:test.properties”)，classpath表示需要配置的配置文件路径。

**二、多环境配置**

当应用程序需要部署到不同运行环境时，一些配置细节通常会有所不同，最简单的比如日志，生产日志会将日志级别设置为WARN或更高级别，并将日志写入日志文件，而开发的时候需要日志级别为DEBUG，日志输出到控制台即可。

在Spring Boot中多环境配置文件名需要满足application-{profile}.properties的格式，其中{profile}对应你的环境标识，比如：

* application-dev.properties：开发环境
* application-prod.properties：生产环境

想要使用对应的环境，只需要在application.properties中使用spring.profiles.active属性来设置，值对应上面提到的{profile}，这里就是指dev、prod这2个。

**三、静态资源**

使用Spring Boot的默认配置方式，提供的静态资源映射如下:

* classpath:/META-INF/resources
* classpath:/resources
* classpath:/static
* classpath:/public
* 对应的配置文件配置如下（自定义静态资源访问路径）：

|  |  |
| --- | --- |
| * 1 2 3 4 | * # 默认值为 /\*\*（配置访问路径） spring.mvc.static-path-pattern= * # 默认值为 classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/ spring.resources.static-locations=这里设置要指向的路径，多个使用英文逗号隔开 |

**四、拦截器**

要实现拦截器功能需要完成以下2个步骤：

* 创建我们自己的拦截器类并实现 HandlerInterceptor 接口
* 其实重写WebMvcConfigurerAdapter中的addInterceptors方法把自定义的拦截器类添加进来即可。

以判断是否登录为例实现拦截器：

首先，自定义拦截器代码：

public class MyInterceptor implements HandlerInterceptor {  
 @Override  
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 boolean flag =true;  
 User user=(User)request.getSession().getAttribute("user");  
 if(null==user){  
 response.sendRedirect("toLogin");  
 flag = false;  
 }else{  
 flag = true;  
 }  
 return flag;  
 }  
  
 @Override  
 public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {  
 }  
  
 @Override  
 public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {  
 }  
}

简单实现了根据session中是否有User对象来判断是否登录，为空就跳转到登录页，不为空就通过。

接着，重写WebMvcConfigurerAdapter（已过时，用WebMvcConfigurationSupport代替）中的addInterceptors方法如下：

*/\*\*  
 \* 拦截器  
 \** ***@param*** *registry  
 \*/*@Override  
public void addInterceptors(InterceptorRegistry registry) {  
 // addPathPatterns 用于添加拦截规则  
 // excludePathPatterns 用户排除拦截  
 registry.addInterceptor(new MyInterceptor()).addPathPatterns("/\*\*").excludePathPatterns("/toLogin","/login");  
 super.addInterceptors(registry);  
}

addPathPatterns("/\*\*"))对所有请求都拦截，但是排除了/toLogin和/login请求的拦截。

**十、application.properties文件常见配置：**

1、配置返回文件的路径以及类型

spring.mvc.view.prefix: /WEB-INF/jsp/  
spring.mvc.view.suffix: .jsp

2、配置程序端口，默认为8080  
server.port= 8080  
3、用户绘画session过期时间，以秒为单位  
server.session.timeout=  
4、配置默认访问路径，默认为/  
server.context-path=

**十一、常用接口/类**

1、WebMvcConfigurationSupport