1.基本配置

1.1 入口类和@SpringBootApplication

Spring Boot项目通过使用main方法,在main方法中使用SpringApplication.run(Config.class,args),启动Spring Boot项目

@SpringBootApplication注解是个组合注解,

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(
   excludeFilters = {@Filter(
   type = FilterType.CUSTOM,
   classes = {TypeExcludeFilter.class}
), @Filter(
   type = FilterType.CUSTOM,
   classes = {AutoConfigurationExcludeFilter.class}
)}
)
```

其中@EnableAutoConfiguration让Spring Boot根据类路径中的jar包依赖为当前项目进行自动配置.

Spring Boot会自动扫描@SpringBootApplication所在类的同级包以及下级包的Bean.

1.2 关闭特定的自动配置

```
@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
```

1.3 Spring Boot的配置文件

Spring Boot使用一个全局的配置文件:application.properties或application.yml放置在src/main/resources目录或者类路径的/config下

@PropertySource不支持yml文件

1.4 使用xml配置

Spring Boot提倡零配置,即无xml配置,但是在实际开发中有特殊要求,必须使用xml配置,可以通过Spring 提供的@ImportResource来加载xml配置,注解使用在启动类上

```
@ImportResource({"classpath:some-context.xml","classpath:another-context.xml"})
```

2.外部配置

Spring Boot 允许使用properties文件,yaml文件或者命令行参数作为外部配置

2.1 命令行参数配置

```
java -jar xx.jar
#修改tomcat端口号
java -jar xx.jar --server.port=9090
```

2.2 类型安全的配置(基于properties)

Spring Boot提供基于类型安全的配置方式,通过@ConfigurationProperties将properties属性和一个Bean及其属性关联,从而实现类型安全的配置.

properties文件,可以是默认的application.properties,也可以是another.properties文件

```
author.name=txx
```

类型安全的Bean:

```
@Component
@ConfigurationProperties(prefix="anthor",location={"classpath:config/another.properties"})
public class AuthorSettings{
   private String name;
   #set get 方法
}
```

启动类:

```
@SpringBootApplication
public class App{
   @Autowired
   private AuthorSettings set;
   private static void main(){
      SpringApplication.run()
   }
}
```

3.日志配置

默认情况下,Spring Boot使用logback作为日志框架

日志配置文件:

```
#配置日志文件
logging.file=d:/mylog/log.log
#配置日志级别,格式为logging.level.包名=级别
logging.level.org.springframework.web=DEBUG
```

4.Profile配置

Profile是Spring用来针对不同环境对不同的配置提供支持的,全局Profile配置使用application-{profile}.properties(如application-prod.properties)

通过application.properties中设置spring.profiles.active=prod来指定活动的Profile

5.Spring Boot运行原理

Spring Boot关于自动配置的源码在spring-boot-autoconfigure.jar内

可以通过下面三种方式查看当前项目中已启用和未启用的自动配置的报告

(1) 运行jar时增加--debug参数:

```
java -jar xx.jar --debug
```

(2)在application.properties中设置属性:

```
debug=true
```

5.1 运作原理

关键是@Import(Spring 提供)注解导入的配置功能,EnableAutoConfigurationImportSelector使用SpringFactoriesLoader.loadFactoryNames方法来扫描具有META-INF/spring.factories文件的jar包

最新版Spring Boot @Import的类是:AutoConfigurationImportSelector.class

5.2 核心注解

@ConditionOnBean:当容器里有指定的Bean的条件下

@ConditionOnClass:当类路径下有指定的类的条件下

@ConditionalOnExpression:基于SpEL表达式作为判断条件

@ConditionalOnJava:基于JVM版本作为判断依据

@ConditonalJndi:在JNDI存在的条件下查找指定的位置

@ConditionOnMissingBean:当容器里没有指定Bean的情况下

@ConditionOnNotWebApplication:当前项目不是web项目的条件下

@ConditionalO你Property:指定的属性是否有指定值

@ConditionalOnResource:l类路径是否有指定的值

@ConditionalOnSingleCandidate:当指定Bean在容器中只有一个,或者虽然有多个,但是指定首选

@ConditonalOnWebApplication:当前项目是web项目的条件下

定义一个http编码自动配置类

1.配置参数类:

```
@ConfigurationProperties(profile="spring.http.encoding")
public class HttpEncodingProperties{
   public static final Charset DEFAULT_CHARSET=Charset.forName("UTF-8")
   private Charset charset = DEFAULT_CHARSET;
   private boolean force = true
   ==get set 方法==
}
```

配置Bean:

```
@Configuration
@EnableConfigurationProperties(HttpEncodingProperties.class)//1
@ConditionalOnClass(CharacterEncodingFilter.class)//2
@ConditionalOnProperty(prefix="spring.http.encoding",value="enabled",matchIfMissing=true)//3

public class HttpEncodingAutoConfiguration{
   @AutoWired
   private HttpEncodingProperties httpEncodingProperties;
   @Bean
   @ConditionOnMissingBean(CharacterEncodingFilter.claaa)
   public CharacterEncodingFilter characterEncodingFilter(){
      CharacterEncodingFilter filter = new OrderedCharacterEncodingFilter();
      filter.setEncoding(thia.httpEncodingProperties.getCharset().name());
      filter.setForceEncoding(this.httpEncodingProperties.isForce());
      return filter;
   }
}
```

#1 开启属性注入,通过@EnableConfigurationProperties声明,使用@Autowired注入
#2 当CharacterEncodingFilter在类路径的条件下
#3 当设置spring.http.encoding=enabled的情况下,如果没有设置则默认为true,即条件符合;

注意添加 META-INF/spring.factories文件:

```
org.springframework.autoconfigure.EnableAutoConfiguration=\x.x.x.x.***AutoCOnfiguration
```