spring面试题

<https://blog.csdn.net/a745233700/article/details/80959716>

<https://www.jianshu.com/p/1c07f909cdc1>

BeanPostProcessor

<https://www.jianshu.com/p/369a54201943>

Spring Bean的生命周期

<https://www.cnblogs.com/weixuqin/p/11027841.html>

<https://www.jianshu.com/p/3944792a5fff>

ApplicationContext

IOC源码

<https://www.cnblogs.com/toby-xu/p/11324776.html>

<https://blog.csdn.net/nuomizhende45/article/details/81158383>

public void refresh() throws BeansException, IllegalStateException {

synchronized (this.startupShutdownMonitor) {

//1:准备刷新上下文环境

prepareRefresh();

//2:获取初始化Bean工厂

ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

//3:对bean工厂进行填充属性

prepareBeanFactory(beanFactory);

try {

//4:Spring开放接口 留给子类去实现该接口

postProcessBeanFactory(beanFactory);

//5:调用我们的bean工厂的后置处理器

invokeBeanFactoryPostProcessors(beanFactory);

//6:注册我们bean后置处理器

registerBeanPostProcessors(beanFactory);

//7:初始化国际化资源处理器

initMessageSource();

//8:初始化事件多播器

initApplicationEventMulticaster();

//9:这个方法同样也是留个子类实现,其中springboot也是从这个方法进行tomcat的启动

onRefresh();

//10:把我们的事件监听器注册到多播器上

registerListeners();

//11:实例化所有的非懒加载的单实例bean

finishBeanFactoryInitialization(beanFactory);

//12:最后刷新容器 发布刷新事件(Spring cloud eureka也是从这里启动的)

finishRefresh();

}

catch (BeansException ex) {

if (logger.isWarnEnabled()) {

logger.warn("Exception encountered during context initialization - " +

"cancelling refresh attempt: " + ex);

}

// Destroy already created singletons to avoid dangling resources.

destroyBeans();

// Reset 'active' flag.

cancelRefresh(ex);

// Propagate exception to caller.

throw ex;

}

finally {

// Reset common introspection caches in Spring's core, since we

// might not ever need metadata for singleton beans anymore...

resetCommonCaches();

}

}

}

BeanFactory ignoreDependencyInterface方法

<https://blog.csdn.net/qq_36951116/article/details/99587519>

1.spring如何解决循环依赖问题

构造器注入会抛出异常BeanCurrentLyInCreationException异常

2.属性单利注入利用三级缓存解决(三个map)

singletonFactories

earlySingletonObjects

singletonObjects

3.prototype属性注入也会报BeanCurrentLyInCreationException异常

spring mvc

常见面试题

<https://blog.csdn.net/a745233700/article/details/80963758>

@ResponseBody 加上之后不能解析ModelAndView

@responseBody注解的作用是将controller的方法返回的对象通过适当的转换器转换为指定的格式之后，写入到response对象的body区，通常用来返回JSON数据或者是XML数据，需要注意的呢，在使用此注解之后不会再走视图处理器，而是直接将数据写入到输入流中，他的效果等同于通过response对象输出指定格式的数据

@RequestMapping("/login")

　　@ResponseBody

　　public User login(User user){

　　　　return user;

　　}

　　User字段：userName pwd

　　那么在前台接收到的数据为：'{"userName":"xxx","pwd":"xxx"}'

　　效果等同于如下代码：

　　@RequestMapping("/login")

　　public void login(User user, HttpServletResponse response){

　　　　response.getWriter.write(JSONObject.fromObject(user).toString());

　　}

SpringMVC 中ModelAndView用法

<https://blog.csdn.net/qq_41648616/article/details/81776145>

注意：如果方法声明了注解@ResponseBody ，则会直接将返回值输出到页面（可以return一个JSONPObject，直接到前台就是json）。

例如：

Java代码 收藏代码

@RequestMapping(value = "/something", method = RequestMethod.GET)

@ResponseBody

public String helloWorld() {

return "Hello World";

}

上面的结果会将文本"Hello World "直接写到http响应流。

Java代码 收藏代码

@RequestMapping("/welcome")

public String welcomeHandler() {

return "center";

}

对应的逻辑视图名为“center”，URL= prefix前缀+视图名称 +suffix后缀组成。

拦截器和过滤器

两者都是AOP编程思想的实现，都能够实现权限控制和日志记录等问题的处理，但是两者粒度不同拦截对象不一样

适用范围不同：Filter是servlet的规范，只能用于web程序，但是拦截器可以用于application等程序。

规范不同：Filter是servlet的规范。但是Interceptor是spring容器支撑，有spring框架支持。

使用资源不一样：spring的拦截器由于依赖spring，也是spring的一个组件，因此能够在拦截器中使用spring的任何资源和对象。例如service对象，数据源，事务管理等，通过ioc注入拦截器即可，而filter不能

粒度不同：Filter只能在servlet的前后起作用，而拦截器能在方法前后异常前后执行，更加灵活，粒度更小，spring框架程序优先使用拦截器

执行顺序

<https://blog.csdn.net/u012995888/article/details/103398278>

HttpServletRequest 详解

https://blog.csdn.net/wwq0813/article/details/90256058

Servlet

@WebServlet(name = "firstServlet", urlPatterns = "/firstServlet") //标记为servlet，以便启动器扫描。

public class FirstServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

resp.getWriter().append("firstServlet");

}

}

SpringBoot自动配置原理

SpringBoot自定义starter

一、创建maven工程

Spring官方建议非官方Starter命名应遵循{name}-spring-boot-starter的格式, 如mybatis-spring-boot-starter

引入依赖

<dependencies>

<dependency>

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

<artifactId>spring-boot-autoconfigure</artifactId>

<version>2.0.0.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<version>2.0.0.RELEASE</version>

<optional>true</optional>

</dependency>

</dependencies>

二、src/main/resources/META-INF/spring.factories

META-INF是自己手动创建的目录，spring.factories也是手动创建的文件,在该文件中配置自己的自动配置类

org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.mengday.helloworld.PersonServiceAutoConfiguration

三、定义Configuration类

@Configuration

@EnableConfigurationProperties(PersonProperties.class)

@ConditionalOnClass(PersonService.class)

@ConditionalOnProperty(prefix = "spring.person", value = "enabled", matchIfMissing = true)

public class PersonServiceAutoConfiguration {

@Autowired

private PersonProperties properties;

@Bean

@ConditionalOnMissingBean(PersonService.class) // 当容器中没有指定Bean的情况下，自动配置PersonService类

public PersonService personService(){

PersonService personService = new PersonService(properties);

return personService;

}

}

四、定义Properties.类

@ConfigurationProperties(prefix = "spring.person")

public class PersonProperties {

// 姓名

private String name;

// 年龄

private int age;

// 性别

private String sex = "M";

// Getter & Setter

}

五、定义Service类

public class PersonService {

private PersonProperties properties;

public PersonService() {

}

public PersonService(PersonProperties properties) {

this.properties = properties;

}

public void sayHello(){

System.out.println("大家好，我叫: " + properties.getName() + ", 今年" + properties.getAge() + "岁"

+ ", 性别: " + properties.getSex());

}

}

六、打包

mvn clean install

七、其他工程引入

<dependency>

<groupId>com.mengday</groupId>

<artifactId>helloworld-spring-boot-starter</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

八、测试

配置application.properties

spring.person.name=mengday

spring.person.age=28

public class MystarterApplicationTests {

@Autowired

private PersonService personService;//直接注入

@Test

public void testHelloWorld() {

personService.sayHello();

}

}

常用注解

@Configuration

@EnableConfigurationProperties(PersonProperties.class)

@ConfigurationProperties(prefix = "spring.person")

@Conditional注解

@ConditionalOnClass：当类路径classpath下有指定的类的情况下进行自动配置

@ConditionalOnMissingClass:当类路径下没有指定的类的条件下

@ConditionalOnBean:当容器(Spring Context)中有指定的Bean的条件下

@ConditionalOnMissingBean:当容器(Spring Context)中没有指定Bean的情况下进行自动配置

@ConditionalOnProperty(prefix = “example.service”, value = “enabled”, matchIfMissing = true)，当配置文件中example.service.enabled=true时进行自动配置，如果没有设置此值就默认使用matchIfMissing对应的值

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

@ConditionalOnJava:基于JVM版本作为判断条件

@ConditionalOnJndi:在JNDI存在的条件下查找指定的位置

@ConditionalOnNotWebApplication:当前项目不是Web项目的条件下

@ConditionalOnWebApplication:当前项目是Web项目的条件下

@ConditionalOnResource:类路径下是否有指定的资源

@ConditionalOnSingleCandidate:当指定的Bean在容器中只有一个，或者在有多个Bean的情况下，用

@EnableEurekaClient

@EnableDiscoveryClient

@EnableFeginClients

Eureka

自我保护模式

开启自我保护模式将不删除服务注册表的信息

eureka.server.enable-self-preservation=false禁用自我保护模式

多网卡ip选择

1、忽略指定网卡

spring:

cloud:

inetutils:

ignored-interfaces:

- docker0

- veth.\*

eureka:

instance:

prefer-ip-addres true

2.正则表达式指定网络地址

spring:

cloud:

inetutils:

preferredNetworks:

- 192.168

- 10.0

eureka:

instance:

prefer-ip-addres true

健康检查

将spring boot actuator信息传播到eureka

eureka:

client:

healthcheck:

enabled: true

脱离eureka使用ribbon

service\_name:

ribbon:

listOfServers: localhost:8000,localhost:8001

@Bean

@LoadBalanced

public RestTemplate restTemplate(){

return new RestTempte();

}

@AutoWired

RestTemplate restTemplate;

restTemplate.getForObject("http://service\_name/user/id");

Fegin

一种接口式http客户端，整合了spring mvc,可以搭配springmvc注解使用

自定义fegin

使用属性自定义fegin配置

断路器状态

打开

全开

半开

Ribbon + @HystrixCommand

使用信号量隔离

@HystrixCommand(fallbackMethod="",commandProperties={@HystrixProperty(name="execution.isolation.strategy",value="SEMAPHORE")})

fegin自带hystrix

设置fegin.hystrix.enabled=true

@HystrixCommand

获取熔断器断开原因，只需要在fallback方法添加throwable参数即可

@Feign

实现FallbackFactory