Springboot 启动Tomcat 热身

不得不说的后置处理器 org.springframework.beans.factory.config.BeanPostProcessor

调用时机: **在每个Bean调用构造方法之后，初始化前后进行工作**

```
public interface BeanPostProcessor {
    default Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

    }
   default Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
     return bean;
   }
}
```

关键触发时机:

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#doCreateBean

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#initializeBean

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#applyBeanPostProcessorsBeforeIni

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#invokeInitMethods

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#applyBeanPostProcessorsAfterIniti

那我们就来大致看下applyBeanPostProcessorsBeforeInitialization**方法**

```
    调用所有后置处理器的postProcessAfterInitialization方法 对当前bean进行拦截
    public Object applyBeanPostProcessorsAfterInitialization(Object existingBean, String beanName)
            throws BeansException {

        Object result = existingBean;
        for (BeanPostProcessor processor : getBeanPostProcessors()) {
            Object current = processor.postProcessAfterInitialization(result, beanName);
            if (current == null) {
                return result;
            }
            result = current;
        }
        return result;
    }
```

invokeInitMethods**方法**

```
        boolean isInitializingBean = (bean instanceof InitializingBean);
        if (isInitializingBean && (mbd == null || !mbd.isExternallyManagedInitMethod("afterPropertiesSet"))) {
            if (logger.isDebugEnabled()) {
                logger.debug("Invoking afterPropertiesSet() on bean with name '" + beanName + "'");
            }
            if (System.getSecurityManager() != null) {
```

```
                    try {
                        AccessController.doPrivileged((PrivilegedExceptionAction<Object>) () -> {
                            ((InitializingBean) bean).afterPropertiesSet();
                            return null;
                        }, getAccessControlContext());
                    }
                    catch (PrivilegedActionException pae) {
                        throw pae.getException();
                    }
                }
                //调用InitializingBean的 afterPropertiesSet方法
                else {
                    ((InitializingBean) bean).afterPropertiesSet();
                }
            }

        //调用自定义的init方法
            if (mbd != null && bean.getClass() != NullBean.class) {
                String initMethodName = mbd.getInitMethodName();
                if (StringUtils.hasLength(initMethodName) &&
                        !(isInitializingBean && "afterPropertiesSet".equals(initMethodName)) &&
                        !mbd.isExternallyManagedInitMethod(initMethodName)) {
                    invokeCustomInitMethod(beanName, bean, mbd);
                }
            }
```

然后再看下:applyBeanPostProcessorsAfterInitialization

```
        //调用所有后置处理器的postProcessAfterInitialization方法
        public Object applyBeanPostProcessorsAfterInitialization(Object existingBean, String beanName)
                throws BeansException {

            Object result = existingBean;
            for (BeanPostProcessor processor : getBeanPostProcessors()) {
                Object current = processor.postProcessAfterInitialization(result, beanName);
                if (current == null) {
                    return result;
                }
                result = current;
            }
            return result;
        }
```

BeanPostProcessor**是什么时候注册到容器中去的**?

>org.springframework.context.support.AbstractApplicationContext#registerBeanPostProcessors

>org.springframework.context.support.PostProcessorRegistrationDelegate#registerBeanPostProcessors

```
public static void registerBeanPostProcessors(
                ConfigurableListableBeanFactory beanFactory, AbstractApplicationContext applicationContext) {
```

```
//找到所有的后置处理器的名称
        String[] postProcessorNames = beanFactory.getBeanNamesForType(BeanPostProcessor.class, true, false);

        // Register BeanPostProcessorChecker that logs an info message when
        // a bean is created during BeanPostProcessor instantiation, i.e. when
        // a bean is not eligible for getting processed by all BeanPostProcessors.
        int beanProcessorTargetCount = beanFactory.getBeanPostProcessorCount() + 1 + postProcessorNames.length
        beanFactory.addBeanPostProcessor(new BeanPostProcessorChecker(beanFactory, beanProcessorTargetCount))

        // Separate between BeanPostProcessors that implement PriorityOrdered,
        // Ordered, and the rest.
        List<BeanPostProcessor> priorityOrderedPostProcessors = new ArrayList<>();
        List<BeanPostProcessor> internalPostProcessors = new ArrayList<>();
        List<String> orderedPostProcessorNames = new ArrayList<>();
        List<String> nonOrderedPostProcessorNames = new ArrayList<>();

        //把后置处理器区分开来(出分 包括 创建bean对象)
        for (String ppName : postProcessorNames) {
            if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
                BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
                priorityOrderedPostProcessors.add(pp);
                if (pp instanceof MergedBeanDefinitionPostProcessor) {
                    internalPostProcessors.add(pp);
                }
            }
            else if (beanFactory.isTypeMatch(ppName, Ordered.class)) {
                orderedPostProcessorNames.add(ppName);
            }
            else {
                nonOrderedPostProcessorNames.add(ppName);
            }
        }

    //按照上面区分的后置处理器 来进行注册(加入到容器中)

        //注册实现PriorityOrderd接口的
        sortPostProcessors(priorityOrderedPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, priorityOrderedPostProcessors);

            //注册实现Ordered接口的
        List<BeanPostProcessor> orderedPostProcessors = new ArrayList<>();
        for (String ppName : orderedPostProcessorNames) {
            BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
            orderedPostProcessors.add(pp);
            if (pp instanceof MergedBeanDefinitionPostProcessor) {
                internalPostProcessors.add(pp);
            }
        }
        sortPostProcessors(orderedPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, orderedPostProcessors);

        // Now, register all regular BeanPostProcessors.
        List<BeanPostProcessor> nonOrderedPostProcessors = new ArrayList<>();
        for (String ppName : nonOrderedPostProcessorNames) {
            BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
            nonOrderedPostProcessors.add(pp);
            if (pp instanceof MergedBeanDefinitionPostProcessor) {
                internalPostProcessors.add(pp);
            }
        }
        registerBeanPostProcessors(beanFactory, nonOrderedPostProcessors);
```

```
        // Finally, re-register all internal BeanPostProcessors.
        sortPostProcessors(internalPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, internalPostProcessors);

        // Re-register post-processor for detecting inner beans as ApplicationListeners,
        // moving it to the end of the processor chain (for picking up proxies etc).
        beanFactory.addBeanPostProcessor(new ApplicationListenerDetector(applicationContext));
    }
```

SpringBoot**依靠 自动装配 来如何装配**Tomcat**的**