链滴

spring 的 IOC 流程（源码流程）

作者：18380422102

原文链接：https://ld246.com/article/1534751513447

来源网站：链滴

先从源码来看spring的IOC都经历了哪些方法：

1.开始加载xml配置文件

```
ApplicationContext context = new ClassPathXmlApplicationContext("test.xml");
```

2.ApplicationContext的构造方法

```
this(new String[] {configLocation}, true, null);
public ClassPathXmlApplicationContext(
      String[] configLocations, boolean refresh, @Nullable ApplicationContext parent)
      throws BeansException {

   super(parent);
   setConfigLocations(configLocations);
   if (refresh) {
      refresh();
   }
}
```

3.父类构造方法

一直跳转到AbstractApplicationContext文件

```
public AbstractApplicationContext(@Nullable ApplicationContext parent) {
   this();
   setParent(parent);
}
```

其中this（）方法是创建一个新的AbstractApplicationContext，并生成一个默认的资源模式解析器
因为parent为null,所以setParent没有实质作用。

4.设置配置文件路径

返回ClassPathXmlApplicationContext查看setConfigLocations方法。此方法是将传入的资源路径存下来。

```
public void setConfigLocations(@Nullable String... locations) {
   if (locations != null) {
      Assert.noNullElements(locations, "Config locations must not be null");
      this.configLocations = new String[locations.length];
      for (int i = 0; i < locations.length; i++) {
         this.configLocations[i] = resolvePath(locations[i]).trim();
      }
   }
   else {
      this.configLocations = null;
   }
}
```

5.refresh方法

主要的IOC方法。主要看spring的beanfactory怎么生成的，以及保存了什么。

```
@Override
public void refresh() throws BeansException, IllegalStateException {
   synchronized (this.startupShutdownMonitor) {
      // 准备此上下文以进行刷新。
  prepareRefresh();

      // 告诉子类刷新内部bean工厂。
  ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

      // 准备在这种情况下使用的Bean工厂。
  prepareBeanFactory(beanFactory);

      try {
         // 允许在上下文子类中对bean工厂进行后处理。
  postProcessBeanFactory(beanFactory);

         // 调用工厂处理器在上下文中注册为bean。
  invokeBeanFactoryPostProcessors(beanFactory);

         // 注册bean处理器，拦截bean创建。
  registerBeanPostProcessors(beanFactory);

         // Initialize message source for this context.
  initMessageSource();

         // 初始化此上下文的消息源。
  initApplicationEventMulticaster();

         // 在特定上下文子类中初始化其他特殊bean。
  onRefresh();

         // Check for listener beans and register them.
  registerListeners();

         // 检查侦听器bean并注册它们。
```

```
  finishBeanFactoryInitialization(beanFactory);

        // 最后一步：发布相应的事件。
  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();
      }
    }
}
```

6.obtainFreshBeanFactory方法

在refreshBeanFactory生成beanFactory，然后通过getBeanFactory获取

```
protected ConfigurableListableBeanFactory obtainFreshBeanFactory() {
  refreshBeanFactory();
  ConfigurableListableBeanFactory beanFactory = getBeanFactory();
  if (logger.isDebugEnabled()) {
    logger.debug("Bean factory for " + getDisplayName() + ": " + beanFactory);
  }
  return beanFactory;
}
```

7.refreshBeanFactory的实现方法

在idea中可以使用ctrl+alt+b的快捷键查看方法的实现方法。该方法在AbstractRefreshableApplicationContext中，

可以看到，方法中先销毁了之前的beanFactory（如果有的话）,然后生成了一个默认的DefaultListableBeanFactory。然后将beanFactory保存，使得上一步的getBeanFactory可以获取。然后看loadBeanDefinitions方法。

```
protected final void refreshBeanFactory() throws BeansException {
  if (hasBeanFactory()) {
    destroyBeans();
    closeBeanFactory();
  }
```

```
    try {
      DefaultListableBeanFactory beanFactory = createBeanFactory();
      beanFactory.setSerializationId(getId());
      customizeBeanFactory(beanFactory);
      loadBeanDefinitions(beanFactory);
      synchronized (this.beanFactoryMonitor) {
        this.beanFactory = beanFactory;
      }
    }
    catch (IOException ex) {
      throw new ApplicationContextException("I/O error parsing bean definition source for " +
etDisplayName(), ex);
    }
}
```

8.loadBeanDefinitions方法

这里有很多的loadBeanDefinitions在互相调用，这些loadBeanDefinitions的作用就是将最开始传入资源文件路径加载，封装成InputSource，方便后续解析文件。

```
//生成一个xml读取器
@Override
protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory) throws BeansEx
eption, IOException {
    // Create a new XmlBeanDefinitionReader for the given BeanFactory.
   XmlBeanDefinitionReader beanDefinitionReader = new XmlBeanDefinitionReader(beanFacto
y);

    // Configure the bean definition reader with this context's
  // resource loading environment.  beanDefinitionReader.setEnvironment(this.getEnvironment
));
//传入一个资源加载器，就是AbstractXmlApplicationContext
    beanDefinitionReader.setResourceLoader(this);
    beanDefinitionReader.setEntityResolver(new ResourceEntityResolver(this));

    // Allow a subclass to provide custom initialization of the reader,
  // then proceed with actually loading the bean definitions.  initBeanDefinitionReader(beanDef
nitionReader);
    loadBeanDefinitions(beanDefinitionReader);
}
protected void loadBeanDefinitions(XmlBeanDefinitionReader reader) throws BeansException,
IOException {
    Resource[] configResources = getConfigResources();
    if (configResources != null) {
      reader.loadBeanDefinitions(configResources);
    }
//用reader读取最开始传入的资源文件路径
    String[] configLocations = getConfigLocations();
    if (configLocations != null) {
      reader.loadBeanDefinitions(configLocations);
    }
}
@Override
public int loadBeanDefinitions(String... locations) throws BeanDefinitionStoreException {
    Assert.notNull(locations, "Location array must not be null");
```

```
   int counter = 0;
   for (String location : locations) {
      counter += loadBeanDefinitions(location);
   }
   return counter;
}
@Override
public int loadBeanDefinitions(String location) throws BeanDefinitionStoreException {
   return loadBeanDefinitions(location, null);
}
public int loadBeanDefinitions(String location, @Nullable Set actualResources) throws BeanDe
initionStoreException {
   ResourceLoader resourceLoader = getResourceLoader();
   if (resourceLoader == null) {
      throw new BeanDefinitionStoreException(
            "Cannot import bean definitions from location [" + location + "]: no ResourceLoader a
ailable");
   }
  //前面传入了一个资源加载器，可以通过继承树看出AbstractXmlApplicationContext是ResourceP
tternResolver的子类
   if (resourceLoader instanceof ResourcePatternResolver) {
      // Resource pattern matching available.
  try {
//将资源文件的路径封装成Resource
         Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(locati
n);
         int loadCount = loadBeanDefinitions(resources);
         if (actualResources != null) {
            for (Resource resource : resources) {
               actualResources.add(resource);
            }
         }
         if (logger.isDebugEnabled()) {
            logger.debug("Loaded " + loadCount + " bean definitions from location pattern [" + l
cation + "]");
         }
         return loadCount;
      }
      catch (IOException ex) {
         throw new BeanDefinitionStoreException(
               "Could not resolve bean definition resource pattern [" + location + "]", ex);
      }
   }
   else {
      // Can only load single resources by absolute URL.
  Resource resource = resourceLoader.getResource(location);
      int loadCount = loadBeanDefinitions(resource);
      if (actualResources != null) {
         actualResources.add(resource);
      }
      if (logger.isDebugEnabled()) {
         logger.debug("Loaded " + loadCount + " bean definitions from location [" + location + "
");
      }
```

```
    return loadCount;
  }
}
@Override
public int loadBeanDefinitions(Resource... resources) throws BeanDefinitionStoreException {
  Assert.notNull(resources, "Resource array must not be null");
  int counter = 0;
  for (Resource resource : resources) {
    counter += loadBeanDefinitions(resource);
  }
  return counter;
}
@Override
public int loadBeanDefinitions(Resource resource) throws BeanDefinitionStoreException {
  return loadBeanDefinitions(new EncodedResource(resource));
}
//封装资源文件和inputstream
public int loadBeanDefinitions(EncodedResource encodedResource) throws BeanDefinitionSto
eException {
  Assert.notNull(encodedResource, "EncodedResource must not be null");
  if (logger.isInfoEnabled()) {
    logger.info("Loading XML bean definitions from " + encodedResource.getResource());
  }

  Set currentResources = this.resourcesCurrentlyBeingLoaded.get();
  if (currentResources == null) {
    currentResources = new HashSet<>(4);
    this.resourcesCurrentlyBeingLoaded.set(currentResources);
  }
  if (!currentResources.add(encodedResource)) {
    throw new BeanDefinitionStoreException(
        "Detected cyclic loading of " + encodedResource + " - check your import definitions!")

  }
  try {
    InputStream inputStream = encodedResource.getResource().getInputStream();
    try {
      InputSource inputSource = new InputSource(inputStream);
      if (encodedResource.getEncoding() != null) {
        inputSource.setEncoding(encodedResource.getEncoding());
      }
      return doLoadBeanDefinitions(inputSource, encodedResource.getResource());
    }
    finally {
      inputStream.close();
    }
  }
  catch (IOException ex) {
    throw new BeanDefinitionStoreException(
        "IOException parsing XML document from " + encodedResource.getResource(), ex);
  }
  finally {
    currentResources.remove(encodedResource);
    if (currentResources.isEmpty()) {
```

```
        this.resourcesCurrentlyBeingLoaded.remove();
      }
    }
  }
```

8.doLoadBeanDefinitions方法

```
protected int doLoadBeanDefinitions(InputSource inputSource, Resource resource)
      throws BeanDefinitionStoreException {
    try {
//使用前面封装的资源文件和inputSource读取资源文件，并封装为Document对象
      Document doc = doLoadDocument(inputSource, resource);
      return registerBeanDefinitions(doc, resource);
    }
    catch (BeanDefinitionStoreException ex) {
      throw ex;
    }
    catch (SAXParseException ex) {
      throw new XmlBeanDefinitionStoreException(resource.getDescription(),
          "Line " + ex.getLineNumber() + " in XML document from " + resource + " is invalid", e
);
    }
    catch (SAXException ex) {
      throw new XmlBeanDefinitionStoreException(resource.getDescription(),
          "XML document from " + resource + " is invalid", ex);
    }
    catch (ParserConfigurationException ex) {
      throw new BeanDefinitionStoreException(resource.getDescription(),
          "Parser configuration exception parsing XML from " + resource, ex);
    }
    catch (IOException ex) {
      throw new BeanDefinitionStoreException(resource.getDescription(),
          "IOException parsing XML document from " + resource, ex);
    }
    catch (Throwable ex) {
      throw new BeanDefinitionStoreException(resource.getDescription(),
          "Unexpected exception parsing XML document from " + resource, ex);
    }
  }
```

9.registerBeanDefinitions方法

```
public int registerBeanDefinitions(Document doc, Resource resource) throws BeanDefinitionS
oreException {
//创建一个BeanDefinition读取器
    BeanDefinitionDocumentReader documentReader = createBeanDefinitionDocumentReader(
;
    int countBefore = getRegistry().getBeanDefinitionCount();
    documentReader.registerBeanDefinitions(doc, createReaderContext(resource));
    return getRegistry().getBeanDefinitionCount() - countBefore;
  }
  @Override
  public void registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {
    this.readerContext = readerContext;
```

```
    logger.debug("Loading bean definitions");
    Element root = doc.getDocumentElement();
    doRegisterBeanDefinitions(root);
}
```

10.doRegisterBeanDefinitions

```
protected void doRegisterBeanDefinitions(Element root) {
    // Any nested  elements will cause recursion in this method. In
 // order to propagate and preserve  default-* attributes correctly, // keep track of the current
(parent) delegate, which may be null. Create // the new (child) delegate with a reference to th
 parent for fallback purposes, // then ultimately reset this.delegate back to its original (parent)
reference. // this behavior emulates a stack of delegates without actually necessitating one.
eanDefinitionParserDelegate parent = this.delegate;
    this.delegate = createDelegate(getReaderContext(), root, parent);

    if (this.delegate.isDefaultNamespace(root)) {
        String profileSpec = root.getAttribute(PROFILE_ATTRIBUTE);
        if (StringUtils.hasText(profileSpec)) {
            String[] specifiedProfiles = StringUtils.tokenizeToStringArray(
                    profileSpec, BeanDefinitionParserDelegate.MULTI_VALUE_ATTRIBUTE_DELIMITERS);
            if (!getReaderContext().getEnvironment().acceptsProfiles(specifiedProfiles)) {
                if (logger.isInfoEnabled()) {
                    logger.info("Skipped XML bean definition file due to specified profiles [" + profileSp
c +
                            "] not matching: " + getReaderContext().getResource());
                }
                return;
            }
        }
    }

    preProcessXml(root);
//开始解析Document中的Element信息
    parseBeanDefinitions(root, this.delegate);
    postProcessXml(root);

    this.delegate = parent;
}
```

11.parseBeanDefinitions

```
protected void parseBeanDefinitions(Element root, BeanDefinitionParserDelegate delegate) {
    if (delegate.isDefaultNamespace(root)) {
        NodeList nl = root.getChildNodes();
        for (int i = 0; i < nl.getLength(); i++) {
            Node node = nl.item(i);
            if (node instanceof Element) {
                Element ele = (Element) node;
                if (delegate.isDefaultNamespace(ele)) {
//                  使用默认的命名空间解析element
                    parseDefaultElement(ele, delegate);
                }
                else {
```

```
                delegate.parseCustomElement(ele);
            }
        }
    }
  }
  else {
    delegate.parseCustomElement(root);
  }
}
```

12.parseDefaultElement

```
private void parseDefaultElement(Element ele, BeanDefinitionParserDelegate delegate) {
//注册import标签信息
  if (delegate.nodeNameEquals(ele, IMPORT_ELEMENT)) {
    importBeanDefinitionResource(ele);
  }
//注册alias标签信息
  else if (delegate.nodeNameEquals(ele, ALIAS_ELEMENT)) {
    processAliasRegistration(ele);
  }
//注册Bean标签信息
  else if (delegate.nodeNameEquals(ele, BEAN_ELEMENT)) {
    processBeanDefinition(ele, delegate);
  }
//解析beans标签信息
  else if (delegate.nodeNameEquals(ele, NESTED_BEANS_ELEMENT)) {
    // recurse
 doRegisterBeanDefinitions(ele);
  }
}
```

13.processBeanDefinition

```
protected void processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) {
//这里已经将配置文件的信息封装为BeanDefinitionHolder，该类保存了BeanDefinition，BeanNa
e和BeanAlices
  BeanDefinitionHolder bdHolder = delegate.parseBeanDefinitionElement(ele);
  if (bdHolder != null) {
    bdHolder = delegate.decorateBeanDefinitionIfRequired(ele, bdHolder);
    try {
      // Register the final decorated instance.
 BeanDefinitionReaderUtils.registerBeanDefinition(bdHolder, getReaderContext().getRegistry(
);
    }
    catch (BeanDefinitionStoreException ex) {
      getReaderContext().error("Failed to register bean definition with name '" +
          bdHolder.getBeanName() + "'", ele, ex);
    }
    // Send registration event.
 getReaderContext().fireComponentRegistered(new BeanComponentDefinition(bdHolder));
  }
}
```

14.registerBeanDefinition

```
public static void registerBeanDefinition(
      BeanDefinitionHolder definitionHolder, BeanDefinitionRegistry registry)
      throws BeanDefinitionStoreException {

   // Register bean definition under primary name.
  String beanName = definitionHolder.getBeanName();
   registry.registerBeanDefinition(beanName, definitionHolder.getBeanDefinition());

   // Register aliases for bean name, if any.
  String[] aliases = definitionHolder.getAliases();
   if (aliases != null) {
      for (String alias : aliases) {
         registry.registerAlias(beanName, alias);
      }
   }
}
```

15.registerBeanDefinition

```
@Override
public void registerBeanDefinition(String beanName, BeanDefinition beanDefinition)
      throws BeanDefinitionStoreException {

   Assert.hasText(beanName, "Bean name must not be empty");
   Assert.notNull(beanDefinition, "BeanDefinition must not be null");

   if (beanDefinition instanceof AbstractBeanDefinition) {
      try {
         ((AbstractBeanDefinition) beanDefinition).validate();
      }
      catch (BeanDefinitionValidationException ex) {
         throw new BeanDefinitionStoreException(beanDefinition.getResourceDescription(), bea
Name,
               "Validation of bean definition failed", ex);
      }
   }

   BeanDefinition oldBeanDefinition;

   oldBeanDefinition = this.beanDefinitionMap.get(beanName);
   if (oldBeanDefinition != null) {
      if (!isAllowBeanDefinitionOverriding()) {
         throw new BeanDefinitionStoreException(beanDefinition.getResourceDescription(), bea
Name,
               "Cannot register bean definition [" + beanDefinition + "] for bean '" + beanName +
               "': There is already [" + oldBeanDefinition + "] bound.");
      }
      else if (oldBeanDefinition.getRole() < beanDefinition.getRole()) {
         // e.g. was ROLE_APPLICATION, now overriding with ROLE_SUPPORT or ROLE_INFRAST
UCTURE
  if (this.logger.isWarnEnabled()) {
            this.logger.warn("Overriding user-defined bean definition for bean '" + beanName +
                  "' with a framework-generated bean definition: replacing [" +
                  oldBeanDefinition + "] with [" + beanDefinition + "]");
```

```
        }
      }
      else if (!beanDefinition.equals(oldBeanDefinition)) {
        if (this.logger.isInfoEnabled()) {
          this.logger.info("Overriding bean definition for bean '" + beanName +
              "' with a different definition: replacing [" + oldBeanDefinition +
              "] with [" + beanDefinition + "]");
        }
      }
      else {
        if (this.logger.isDebugEnabled()) {
          this.logger.debug("Overriding bean definition for bean '" + beanName +
              "' with an equivalent definition: replacing [" + oldBeanDefinition +
              "] with [" + beanDefinition + "]");
        }
      }
//将前面封装的BeanDefinitionHolder中的beanName和beanDefinition保存到一个ConcurrentHas
Map中
      this.beanDefinitionMap.put(beanName, beanDefinition);
    }
    else {
      if (hasBeanCreationStarted()) {
        // Cannot modify startup-time collection elements anymore (for stable iteration)
  synchronized (this.beanDefinitionMap) {
          this.beanDefinitionMap.put(beanName, beanDefinition);
          List updatedDefinitions = new ArrayList<>(this.beanDefinitionNames.size() + 1);
          updatedDefinitions.addAll(this.beanDefinitionNames);
          updatedDefinitions.add(beanName);
          this.beanDefinitionNames = updatedDefinitions;
          if (this.manualSingletonNames.contains(beanName)) {
            Set updatedSingletons = new LinkedHashSet<>(this.manualSingletonNames);
            updatedSingletons.remove(beanName);
            this.manualSingletonNames = updatedSingletons;
          }
        }
      }
      else {
        // Still in startup registration phase
  this.beanDefinitionMap.put(beanName, beanDefinition);
        this.beanDefinitionNames.add(beanName);
        this.manualSingletonNames.remove(beanName);
      }
      this.frozenBeanDefinitionNames = null;
    }

    if (oldBeanDefinition != null || containsSingleton(beanName)) {
      resetBeanDefinition(beanName);
    }
  }
```

终于结束了，可以看到，最后将beanDefinition存放在了beanDefinitionMap中，而beanDefinitio
Map就是一个ConcurrentHashMap集合。