链滴

# spring 的 IOC 流程（资源定位）

作者：18380422102

原文链接：https://ld246.com/article/1534840673213

来源网站：链滴

spring的IOC流程大致可以分为三步：1.资源定位；2.资源解析；3.注册BeanDefinition

现在，来详细了解一下各个步骤

# 资源定位

资源定位的最后结果是封装了资源，在spring中为Resource类，其默认实现类为FileUrlResource，装了文件信息。

## 起始

资源定位的开始在AbstractBeanDefinitionReader文件，我们看一下加载资源的代码：

```
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");
   }

   if (resourceLoader instanceof ResourcePatternResolver) {
  try {
//在这里，开始将我们传入的文件地址转化为spring封装的Resource类
         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;
  }
}
```

## 解析

在上面，我们看到定位的主要函数是getResources，通过此函数可以将资源文件地址转化为Resourc类，而此函数的主要实现方法在PathMatchingResourcePatternResolver（如果对此类面生，可以看ClassPathXmlApplicationContext构造方法，在那个构造方法中，调用父类创建了一个默认的PatMatchingResourcePatternResolver）类中

```
public Resource[] getResources(String locationPattern) throws IOException {
  Assert.notNull(locationPattern, "Location pattern must not be null");
  if (locationPattern.startsWith(CLASSPATH_ALL_URL_PREFIX)) {

  if (getPathMatcher().isPattern(locationPattern.substring(CLASSPATH_ALL_URL_PREFIX.length(
))) {

        return findPathMatchingResources(locationPattern);
    }
    else {
     return findAllClassPathResources(locationPattern.substring(CLASSPATH_ALL_URL_PREFIX.
ength()));
    }
  }
  else {
        int prefixEnd = (locationPattern.startsWith("war:") ? locationPattern.indexOf("*/") + 1 :

        locationPattern.indexOf(':') + 1);
    if (getPathMatcher().isPattern(locationPattern.substring(prefixEnd))) {
```

```
            return findPathMatchingResources(locationPattern);
        }
        else {
//我们的测试文件是直接写的路径，既没有classpath前缀也没有war等前缀，所以只执行到了这一步
            return new Resource[] {getResourceLoader().getResource(locationPattern)};
        }
    }
}
```

## 封装

上一步，在默认的解析过后，进入了getResource方法（DefaultResourceLoader类中），这个方法开始进行封装操作，

```
public Resource getResource(String location) {
    Assert.notNull(location, "Location must not be null");

    for (ProtocolResolver protocolResolver : this.protocolResolvers) {
        Resource resource = protocolResolver.resolve(location, this);
        if (resource != null) {
            return resource;
        }
    }

    if (location.startsWith("/")) {
        return getResourceByPath(location);
    }
    else if (location.startsWith(CLASSPATH_URL_PREFIX)) {
        return new ClassPathResource(location.substring(CLASSPATH_URL_PREFIX.length()), getCl
ssLoader());
    }
    else {
        try {
//在这里，将资源文件路径封装为URL，并进一步封装为 FileUrlResource
    URL url = new URL(location);
            return (ResourceUtils.isFileURL(url) ? new FileUrlResource(url) : new UrlResource(url));
        }
        catch (MalformedURLException ex) {
            // No URL -> resolve as resource path.
    return getResourceByPath(location);
        }
    }
}
```

上面，最终将资源文件路径封装为了FileUrlResource，整个定位流程结束。