芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2019-02-10
Spring

【死磕 Spring】—— IoC 之获取验证模型

本文主要基于 Spring 5.0.6.RELEASE

摘要：原创出处 http://cmsblogs.com/?p=2688 「小明哥」，谢谢！

作为「小明哥」的忠实读者，「老芳芳」略作修改，记录在理解过程中，参考的资料。

在上篇博客【死磕 Spring】—— IoC 之加载 Definitions 中提到，在核心逻辑方法 `#doLoadBeanDefinitions(InputSource inputSource, Resource resource)` 方法中，中主要是做三件事情：

1. 调用 `#getValidationModeForResource(Resource resource)` 方法，获取指定资源（xml）的验证模式。
2. 调用 `DocumentLoader#loadDocument(InputSource inputSource, EntityResolver entityResolver,ErrorHandler errorHandler, int validationMode, boolean namespaceAware)` 方法，获取 XML Document 实例。
3. 调用 `#registerBeanDefinitions(Document doc, Resource resource)` 方法，根据获取的 Document 实例，注册 Bean 信息。

这篇博客主要第 1 步，分析获取 xml 文件的验证模式。为什么需要获取验证模式呢？原因如下：

> XML 文件的验证模式保证了 XML 文件的正确性。

1. DTD 与 XSD 的区别

1.1 DTD

> 老芳芳：因为一些胖友，可能没了解过 DTD 和 XSD ，所以本小节先做一个科普。

DTD(Document Type Definition)，即文档类型定义，为 XML 文件的验证机制，属于 XML 文件中组成的一部分。DTD 是一种保证 XML 文档格式正确的有效验证方式，它定义了相关 XML 文档的元素、属性、排列方式、元素的内容类型以及元素的层次结构。其实 DTD 就相当于 XML 中的 “词汇”和“语法”，我们可以通过比较 XML 文件和 DTD 文件 来看文档是否符合规范，元素和标签使用是否正确。

要在 Spring 中使用 DTD，需要在 Spring XML 文件头部声明：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
```

DTD 在一定的阶段推动了 XML 的发展，但是它本身存在着一些缺陷：

1. 它没有使用 XML 格式，而是自己定义了一套格式，相对解析器的重用性较差；而且 DTD 的构建和访问没有标准的编程接口，因而解析器很难简单的解析 DTD 文档。
2. DTD 对元素的类型限制较少；同时其他的约束力也叫弱。
3. DTD 扩展能力较差。
4. 基于正则表达式的 DTD 文档的描述能力有限。

1.2 XSD

针对 DTD 的缺陷，W3C 在 2001 年推出 XSD。XSD（XML Schemas Definition）即 XML Schema 语言。XML Schema 本身就是一个 XML文档，使用的是 XML 语法，因此可以很方便的解析 XSD 文档。相对于 DTD，XSD 具有如下优势：

1. XML Schema 基于 XML ，没有专门的语法。
2. XML Schema 可以象其他 XML 文件一样解析和处理。
3. XML Schema 比 DTD 提供了更丰富的数据类型。
4. XML Schema 提供可扩充的数据模型。
5. XML Schema 支持综合命名空间。
6. XML Schema 支持属性组。

2. getValidationModeForResource

```
// XmlBeanDefinitionReader.java

// 禁用验证模式
public static final int VALIDATION_NONE = XmlValidationModeDetector.VALIDATION_NONE;
// 自动获取验证模式
public static final int VALIDATION_AUTO = XmlValidationModeDetector.VALIDATION_AUTO;
// DTD 验证模式
public static final int VALIDATION_DTD = XmlValidationModeDetector.VALIDATION_DTD;
// XSD 验证模式
public static final int VALIDATION_XSD = XmlValidationModeDetector.VALIDATION_XSD;

/**
 * 验证模式。默认为自动模式。
 */
private int validationMode = VALIDATION_AUTO;

protected int getValidationModeForResource(Resource resource) {
 // <1> 获取指定的验证模式
 int validationModeToUse = getValidationMode();
 // 首先，如果手动指定，则直接返回
 if (validationModeToUse != VALIDATION_AUTO) {
     return validationModeToUse;
    }
 // 其次，自动获取验证模式
 int detectedMode = detectValidationMode(resource);
 if (detectedMode != VALIDATION_AUTO) {
     return detectedMode;
```

```
    }
 // 最后，使用 VALIDATION_XSD 做为默认
 // Hmm, we didn't get a clear indication... Let's assume XSD,
 // since apparently no DTD declaration has been found up until
 // detection stopped (before finding the document's root tag).
 return VALIDATION_XSD;
}
```

<1> 处，调用 #getValidationMode() 方法，获取指定的验证模式(validationMode)。如果有手动指定，则直接返回。另外，对对于 validationMode 属性的设置和获得的代码，代码如下：

```
public void setValidationMode(int validationMode) {
 this.validationMode = validationMode;
}

public int getValidationMode() {
 return this.validationMode;
}
```

<2> 处，调用 #detectValidationMode(Resource resource) 方法，自动获取验证模式。代码如下：

```
  /**
   * XML 验证模式探测器
   */
  private final XmlValidationModeDetector validationModeDetector = new XmlValidationModeDetector();

  protected int detectValidationMode(Resource resource) {
 // 不可读，抛出 BeanDefinitionStoreException 异常
    if (resource.isOpen()) {
        throw new BeanDefinitionStoreException(
                "Passed-in Resource [" + resource + "] contains an open stream: " +
                "cannot determine validation mode automatically. Either pass in a Resource " +
                "that is able to create fresh streams, or explicitly specify the validationMode " +
                "on your XmlBeanDefinitionReader instance.");
    }

    // 打开 InputStream 流
    InputStream inputStream;
    try {
        inputStream = resource.getInputStream();
    } catch (IOException ex) {
        throw new BeanDefinitionStoreException(
                "Unable to determine validation mode for [" + resource + "]: cannot open InputStream. " +
                "Did you attempt to load directly from a SAX InputSource without specifying the " +
                "validationMode on your XmlBeanDefinitionReader instance?", ex);
    }

    // <x> 获取相应的验证模式
    try {
        return this.validationModeDetector.detectValidationMode(inputStream);
    } catch (IOException ex) {
        throw new BeanDefinitionStoreException("Unable to determine validation mode for [" +
                resource + "]: an error occurred whilst reading from the InputStream.", ex);
    }
  }
```

- 核心在于 <x> 处，调用 XmlValidationModeDetector#detectValidationMode(InputStream inputStream) 方法，获取相应的验证模式。详细解析，见 「3. XmlValidationModeDetector」 中。

<3> 处，使用 VALIDATION_XSD 做为默认。

3. XmlValidationModeDetector

org.springframework.util.xml.XmlValidationModeDetector ，XML 验证模式探测器。

```
public int detectValidationMode(InputStream inputStream) throws IOException {
    // Peek into the file to look for DOCTYPE.
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
    try {
        // 是否为 DTD 校验模式。默认为，非 DTD 模式，即 XSD 模式
        boolean isDtdValidated = false;
        String content;
        // <0> 循环，逐行读取 XML 文件的内容
        while ((content = reader.readLine()) != null) {
            content = consumeCommentTokens(content);
            // 跳过，如果是注释，或者
            if (this.inComment || !StringUtils.hasText(content)) {
                continue;
            }
            // <1> 包含 DOCTYPE 为 DTD 模式
            if (hasDoctype(content)) {
                isDtdValidated = true;
                break;
            }
            // <2>  hasOpeningTag 方法会校验，如果这一行有 < ，并且 < 后面跟着的是字母，则返回 true 。
            if (hasOpeningTag(content)) {
                // End of meaningful data...
                break;
            }
        }
        // 返回 VALIDATION_DTD or VALIDATION_XSD 模式
        return (isDtdValidated ? VALIDATION_DTD : VALIDATION_XSD);
    } catch (CharConversionException ex) {

        // <3> 返回 VALIDATION_AUTO 模式
        // Choked on some character encoding...
        // Leave the decision up to the caller.
        return VALIDATION_AUTO;
    } finally {
        reader.close();
    }
}
```

<0> 处，从代码中看，主要是通过读取 XML 文件的内容，来进行自动判断。

<1> 处，调用 #hasDoctype(String content) 方法，判断内容中如果包含有 "DOCTYPE“ ，则为 DTD 验证模式。代码如下：

```
/**
 * The token in a XML document that declares the DTD to use for validation
 * and thus that DTD validation is being used.
 */
```

```
private static final String DOCTYPE = "DOCTYPE";

private boolean hasDoctype(String content) {
 return content.contains(DOCTYPE);
}
```

<2> 处，调用 #hasOpeningTag(String content) 方法，判断如果这一行包含 < ，并且 < 紧跟着的是字幕，则为 XSD 验证模式。代码如下：

```
/**
 * Does the supplied content contain an XML opening tag. If the parse state is currently
 * in an XML comment then this method always returns false. It is expected that all comment
 * tokens will have consumed for the supplied content before passing the remainder to this method.
 */
private boolean hasOpeningTag(String content) {
 if (this.inComment) {
     return false;
    }
 int openTagIndex = content.indexOf('<');
 return (openTagIndex > -1 // < 存在
            && (content.length() > openTagIndex + 1) // < 后面还有内容
            && Character.isLetter(content.charAt(openTagIndex + 1))); // < 后面的内容是字幕
}
```

<3> 处，如果发生 CharConversionException 异常，则为 VALIDATION_AUTO 模式。

关于 #consumeCommentTokens(String content) 方法，代码比较复杂。感兴趣的胖友可以看看。代码如下：

```
/**
 * The token that indicates the start of an XML comment.
 */
private static final String START_COMMENT = "<!--";

/**
 * The token that indicates the end of an XML comment.
 */
private static final String END_COMMENT = "-->";

/**
 * Consumes all the leading comment data in the given String and returns the remaining content, which
 * may be empty since the supplied content might be all comment data. For our purposes it is only important
 * to strip leading comment content on a line since the first piece of non comment content will be either
 * the DOCTYPE declaration or the root element of the document.
 */
@Nullable
private String consumeCommentTokens(String line) {
 // 非注释
 if (!line.contains(START_COMMENT) && !line.contains(END_COMMENT)) {
     return line;
    }
    String currLine = line;
 while ((currLine = consume(currLine)) != null) {
     if (!this.inComment && !currLine.trim().startsWith(START_COMMENT)) {
         return currLine;
```

```
            }
        }
     return null;
    }

    /**
     * Consume the next comment token, update the "inComment" flag
     * and return the remaining content.
     */
    @Nullable
    private String consume(String line) {
     int index = (this.inComment ? endComment(line) : startComment(line));
     return (index == -1 ? null : line.substring(index));
    }

    /**
     * Try to consume the {@link #START_COMMENT} token.
     * @see #commentToken(String, String, boolean)
     */
    private int startComment(String line) {
     return commentToken(line, START_COMMENT, true);
    }

    private int endComment(String line) {
     return commentToken(line, END_COMMENT, false);
    }

    /**
     * Try to consume the supplied token against the supplied content and update the
     * in comment parse state to the supplied value. Returns the index into the content
     * which is after the token or -1 if the token is not found.
     */
    private int commentToken(String line, String token, boolean inCommentIfPresent) {
     int index = line.indexOf(token);
     if (index > - 1) {
         this.inComment = inCommentIfPresent;
        }
     return (index == -1 ? index : index + token.length());
    }
```

- 反正老艿艿没细看。哈哈哈哈。如果真看，如下两篇文章，有一定的辅助：
  - 《spring源码（六） - XmlValidationModeDetector（获取xml文档校验模式）》
  - 《XmlValidationModeDetector》

666. 彩蛋

好了，XML 文件的验证模式分析完毕。下篇，我们来分析 `#doLoadBeanDefinitions(InputSource inputSource, Resource resource)` 方法的第 2 个步骤：获取 Document 实例。

文章目录

4. 4. 666. 彩蛋

回到首页