XMLによるJavaプログラムのAPIドキュメント生成ツール

中村 章人
産業技術総合研究所
nakamura-akihito@aist.go.jp

Javadocは、Java™のソースファイルからHTMLフォーマットのAPIドキュメントを生成するツールである。Javadocを利用することで、プログラマはコーディングと同時にAPIドキュメントを容易に作成できる。また、Docletというプラグインを組み込むことで、出力をカスタマイズできる。

XMLは、プラットフォーム、応用、言語等の制約を受けず、プログラム処理に適した、単一ソースで多利用が可能なフォーマットである。我々は、JavaのAPIドキュメントをXMLフォーマットで出力するDoclet (XMLDoclet) を開発した。本稿では、XMLDocletの設計と実装について述べる。

A Tool for Java API Documentation in XML

Akihito Nakamura
National Institute of Advanced Industrial Science and Technology (AIST)

Javadoc is a tool to generate HTML-formatted Java API documentation from Java™ source files. By using Javadoc, programmers can easily produce API documents while writing the source code. Users can customize the output of Javadoc by using a *Doclet* plug-in.

XML is a platform-, application-, and language-neutral, computer-comprehensive, and "single-source multi-use" document format. We developed a Doclet, named *XMLDoclet*, that generates Java API documents in XML. This paper describes the design and implementation of the XMLDoclet.

1　はじめに

Javadoc[1, 2] は、Java™*のソースファイルからAPIドキュメントを生成するツールである。プログラムとその説明文章を記述したソースファイルからJavadocがAPIドキュメントを自動生成するので、ユーザはコーディングとドキュメンテーションを同時に一貫して進められる。また、JavadocにはAPIドキュメントの出力を行う**Doclet**というプラグインを組み込むしくみが用意されているので、Docletを使って出力をカスタマイズできる。HTMLフォーマットで出力を行う組み込みのDocletがあらかじめ用意されており、相互にハイパーリンクの張られた利便性の高いドキュメントを出力する。多くのJavaプログラムはこのHTMLのAPIドキュメントと共に配布されている。

XML[4] は、プラットフォーム、応用、言語等の制約を受けず、プログラム処理に適したフォーマットである。データやドキュメントをXMLという標準フォーマットで表現することで、単一のソースを複数の目的に利用でき、応用の変更にも柔軟に対応できる。また、WWWやデータベースとの親和性が高いことから、種々の応用で利用されつつある。

我々は、Javaプログラムの API 情報を様々な応用で利用できるようにするために、XMLフォーマットでAPIドキュメントを出力するDoclet (**XMLDoclet**) を開発した。ソースコードを公開できない場合であっても、XMLのAPIドキュメントを公開しておけば、Docletと同様の応用プログラムを開発することもできる。本稿では、XMLドキュメントの構造とドキュメント間のハイパーリンクの形成方法を中心に、XMLDocletの設計と実装について述べる。

第2章ではJavadocのしくみとソースファイルの記述方法を説明し、XMLDocletの設計方針とAPIドキュメントの構造を第3章で示す。第4章ではXMLDocletの実装について述べ、第5章でAPIドキュメントの表示について述べる。関連研究との比較を第6章で行い、第7章でまとめと課題について述べる。

*Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

2 Javadocのしくみ

2.1 Javadocのアーキテクチャ

Javadocは、ユーザが指定したJavaのソースファイルを入力とし、その中の宣言とコメントを構文解析してAPI情報の内部表現を構築する [図1]。その後、Docletと呼ばれるプラグインプログラムに制御を渡す。Docletは、Doclet APIを通じてAPI情報を取り出し、APIドキュメントの出力を行う。ユーザは、Docletのクラスをコマンドオプションで指定することで、使用するDocletを切り替えられる。

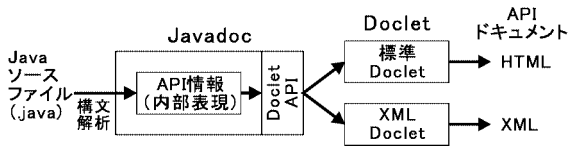

図 1: Javadocのアーキテクチャ

ユーザがDocletを指定せずにJavadocを実行すると、組み込みの**標準 Doclet**が使用され、HTMLフォーマットのAPIドキュメントを出力する。これに対し、我々が開発したXMLDocletはXMLフォーマットで出力を行う。この他に、RTFやTEXを出力するDocletが開発されている [1]。

Doclet APIを用いて以下のような情報を取り出せる。

- 各クラスについて、メンバ(フィールド、コンストラクタ、メソッド)、入れ子クラス、直接のスーパークラスとスーパーインタフェース。
- 各フィールドの型。各メソッドの戻り値の型とオーバライドしたクラス。各コンストラクタとメソッドのパラメータ及びスローする例外。
- 各メンバとクラスについて、それを含むクラス(クラスについては入れ子クラスの場合)、それを含むパッケージ、修飾子、名前、完全修飾名。

これらの情報は、プログラムコードの宣言部分を構文解析して得られる情報である。次節では、ユーザレベルの説明文章の記述方法を説明する。

2.2 ドキュメンテーションコメント

Javadocでは、ソースファイル中の各言語要素(クラスまたはインタフェース、フィールド、コンストラクタ、メソッド)の宣言の直前に置かれたコメント(/**から*/まで)の中に、ユーザレベルの説明文章を記述する。これを**ドキュメンテーションコメント**[2]と呼ぶ(以下ではdocコメントと略す)。

docコメントは**ドキュメンテーションタグ**というJavadoc用のタグを用いてタグ付けを行える。(以下ではdocタグと略す)。Javadocはdocタグを解釈し、Doclet APIで説明文章を取り出せるようにする。また、docコメントにはHTMLタグを使用できるので、表や箇条書き等の構造やリンクを記述できる。

図2 (a) はメソッドのdocコメントの例である。"`Sets the button <i>label</i> text.`"はメソッド`setTextLabel`の説明文章であり、`@`ではじまる`@param`はdocタグである。docタグに続くテキストは引数である。この`@param`タグは、後に続く引数"`The label text.`"がメソッドのパラメータ`text`の説明文章であることを意味する。標準Docletが出力したHTMLをブラウザで表示すると図2 (b) のようになる。下線を引いた`String`はHTMLのリンク(`<a>`タグ)を表している。

```
/**
 * Sets the button <i>label</i> text.
 * @param  text  The label text.
 */
public void setLabelText(String text) {...}
```

(a) ソースコード

setLabelText

`public void setLabelText(String text)`

Sets the button *label* text.

Parameters:

text - The label text.

(b) 標準Docletの出力イメージ (HTMLの表示)

図 2: ドキュメンテーションコメントの例

以下に主なdocタグを示す。docタグは、空白、`*`、`/**`を除いて行の先頭に置く。ただし、`{}`で囲まれたタグは説明文章中で使用する。

主なドキュメンテーションタグ

`@param` *name text*
: メソッドまたはコンストラクタのパラメータ *name* の説明。

`@return` *text*
: メソッドの戻り値の説明。

`@throws` *class-name text*
: メソッドまたはコンストラクタがスローする例外(クラス *class-name*)の説明。

`@version` *text*
: バージョン。

`@see` *reference*
リソースの参照。参照先 *reference* は、*"string"*、`<a href="`*URL*`">`*label*`<a>`、*package.class#member label* のいずれかの形式。

{`@link` *package.class#member label*}
API の参照。

2.3 名前参照とリンク

標準 Doclet の便利な機能の一つにリンクの自動生成 [2] がある。プログラムコードの宣言に現れる名前、または doc コメントの`@see` タグ及び`{@link}`タグで指定した言語要素の名前に対して、API ドキュメントにおけるその参照先の URL を求め、HTML のリンクタグ`<a href="`*URL*`">`を生成する。この処理を**名前参照の解決**と呼ぶ。プログラマは、HTML ブラウザを使って API ドキュメントを表示し、調べたいクラスやメソッドの情報をリンクを辿って参照できる。

例として、以下の doc コメントを考える。

```
/**
 * @see  Object#equals(Object)
 */
```

この doc タグに対し、以下の HTML が出力される。

```
<a href=".../Object.html
   #equals(java.lang.Object)">
     Object.equals(Object)</a>
```

また、図 2 の例では、パラメータの型 String に対して、String クラスの記述を参照先とするリンクが生成される。

このようなリンクは、Javadoc の 1 回の実行で生成するドキュメントだけでなく、既に作成済みのドキュメントに対しても形成できる。例えば、Java Platform API のドキュメントに対してリンクを張ることができる。Javadoc の実行で生成されるドキュメントへの参照を**内部参照**、生成されない既存のドキュメントへの参照を**外部参照**と呼ぶ。外部参照は、ローカルディスク上のドキュメントだけでなく、HTTP を利用してネットワーク上のドキュメントに対しても行える。名前参照の解決方法は 4.2 節で述べる。

3 XMLDoclet の設計

3.1 設計方針

標準 Doclet が生成する HTML フォーマットの API ドキュメントは、表示には便利だが、情報の再利用や加工は困難である。これは、HTML(のタグ) がデータモデルの表現に不向きであることと、標準 Doclet の目的が API 情報の表示のみであることによる。しかし、標準 Doclet は広く用いられており、ほとんどの Java プログラムには標準 Doclet で作成さた API ドキュメントが付属している。このため、XMLDoclet の開発に当たり、ユーザに新たな doc コメントの記法を強要したり、標準 Doclet が生成したドキュメントと相互にリンクを張れなくなってしまうのは避けたい。以下に XMLDoclet の設計方針を示す。

- **ユーザ透過性**: 標準 Doclet で処理できるソースファイルは、XMLDoclet でも処理できる。
- **プログラム透過性**: XMLDoclet は、標準 Doclet が生成した API ドキュメントを外部参照できる。逆に、標準 Doclet は、XMLDoclet が生成した API ドキュメントを外部参照できる。
- **データモデルとビューの分離**: XMLDoclet が生成する XML ドキュメントは API 情報のデータモデルであり、そのビューはスタイルシートと応用プログラムによって提供する。

すなわち、ユーザはソースファイルの作成時に標準 Doclet と XMLDoclet のどちらを使うかを意識する必要がなく (ユーザ透過性)、Javadoc の実行時に外部参照する API ドキュメントが標準 Doclet と XMLDoclet のどちらで生成されたかを意識する必要が無い (プログラム透過性)。データモデルとビューを分離することで、情報の再利用や加工と表示の制御を独立かつ柔軟に行える。

3.2 API ドキュメントのスキーマ

我々は、現在、API ドキュメントの公式なスキーマを公開しておらず、DTD による要素と属性の定義を参考情報として公開している [12]。DTD は、従来から広く普及しており、多くのユーザが理解できるという利点がある。しかし、名前空間を扱えない、XML プロセッサで処理できないという欠点がある。つまり、上記の参考 DTD には名前空間の定義が含まれていない。今後、公式なスキーマを記述するに当たり、名前空間を扱えて XML で記述を行う RELAX[7] をスキーマ言語に使う予定である。

ここでは、XMLDoclet が生成するドキュメントの主要部分の構造を DTD を用いて説明する。図 3〜図 6 に要素と属性の定義を示す。

```
<!ENTITY % generic
 "since?, sees?, description?">
<!ENTITY % class.generic
 "authors?, version?, %generic;">
<!ENTITY % link  "href  %URL;  #IMPLIED">
<!ENTITY % visibility
 "(public|protected|private)  #IMPLIED">
<!ENTITY % prefix
 "%visibility;
 modifier  NMTOKENS  #IMPLIED
 abstract  (true|false)  'false'">
<!ENTITY % html.block  "#PCDATA|div">
<!ENTITY % html.line   "#PCDATA|span">

<!ELEMENT javadoc
 (class|package|overview|index|hierarchy)>

<!ELEMENT class  (containingPackage?,
 containingClass?, superClass?,
 superInterfaces?, nestedClasses?,
 fields?, constructors?, methods?,
 %generic;)>
<!ATTLIST class
 id    ID     #REQUIRED
 type  (class|interface|exception|error)
              "class"
 %prefix;
 name  CDATA  #REQUIRED>

<!ELEMENT  description  (%html.block;)>

<!ELEMENT  name  (#PCDATA)>
```

図 3: DTD による要素と属性の定義

ルート要素と名前空間識別子: 全ドキュメントのルート要素は javadoc、名前空間識別子は http://www.aist.go.jp/2001/javadoc である。各 API の情報は、ルート要素の子要素以下に記述する。例えば、クラスの情報を記述したドキュメントは以下のようになる。

```
<javadoc xmlns=
 "http://www.aist.go.jp/2001/javadoc">
  <class id="..." name="..."> ... </class>
</javadoc>
```

クラス: クラスの情報は class 要素に記述する。子要素は、パッケージ (containingPackage)、直接のスーパークラス (superClass) とスーパーインタフェースのリスト (superInterfaces)、メンバのリスト (fileds、constructors、methods)、入れ子クラスのリスト (nestedClasses) 等である。

メソッド: メソッドの情報は method 要素に記述する [図 4]。子要素は、戻り値 (return)、パラメータのリスト (parameters)、スローされる例外のリスト (exceptions)、実装したメソッド (implement) とオーバライドしたメソッド (override) 等である。

```
<!ELEMENT methods (method+)>
<!ELEMENT method  (parameters?, exceptions?,
 return, implement?, override?, %generic;)>
<!ATTLIST method
 %prefix;
 name       CDATA  #REQUIRED
 signature  CDATA  #REQUIRED>

<!ELEMENT parameters (parameter+)>
<!ELEMENT parameter  (type, (description?))>
<!ATTLIST parameter  name  CDATA  #REQUIRED>

<!ELEMENT exceptions (exception+)>
<!ELEMENT exception  (type, (description?))>

<!ELEMENT return     (type, (description?))>

<!ELEMENT type  EMPTY>
<!ATTLIST type
 name       CDATA  #REQUIRED
 dimension  CDATA  #IMPLIED
 %link;>
```

図 4: DTD による要素と属性の定義 (メソッド)

リソースの参照: リソースの参照を示す@see タグは see 要素に記述する [図 5]。属性%link; は参照先の URL を表す。

```
<!ELEMENT sees  (see+)>
<!ELEMENT see   (%html.line;)>
<!ATTLIST see
 package    CDATA  #IMPLIED
 class      CDATA  #IMPLIED
 member     CDATA  #IMPLIED
 %link;>
```

図 5: DTD による要素と属性の定義 (リソースの参照)

パッケージ: パッケージの情報は package 要素に記述する [図 6]。子要素は、概要 (description)、クラスへの参照のリスト (classes) 等である。

説明文章: API の説明文章は description 要素で表現する [図 3]。2.2 節で述べたように、説明文章には HTML タグを記述できる。しかし、HTML では終了タグを省略できる場合があり、そのままでは XML の生成規則に適合しない。つまり、XML の要素として取り込めない。一方、HTML を XML で再定義した XHTML[5] は、XML の生成規則に従っているので、

```
<!ELEMENT package  (classes+, %generic;)>
<!ATTLIST package  id  ID  #REQUIRED>

<!ELEMENT classes (reference+)

<!ELEMENT reference  EMPTY>
<!ATTLIST reference
  name  CDATA  #REQUIRED
  %link;>
```

図 6: DTD による要素と属性の定義 (パッケージ)

名前空間を用いて XML ドキュメントに取り込める。

現在、多くの doc コメントの記述は従来の HTML で行われており、これをドキュメントに取り込めないのではユーザ透過性を保証できない。しかし、省略された終了タグを補って XHTML に変換するのは難しい。そこで、HTML の場合と XHTML の場合とで要素の内容モデルを別にすることで解決する。XHTML の場合は名前空間を用いて要素として取り込み、そうでない場合は#PCDATA の CDATA セクションでエスケープする (図 3 の `html.block` の定義を参照)。XMLDoclet の実装では、doc コメントを XHTML で記述している場合にだけユーザがそれを明示的にコマンドオプションで通知し、内容モデルの生成方法を切り替える。

例として、以下の doc コメントを考える。

```
/**
 * <ul>
 *   <li>A</li>
 *   <li>B</li>
 * </ul>
 */
```

ここで、doc コメントが XHTML で記述されているとすると、以下の XML を出力する。名前空間識別子のスコープをわかり易くするために HTML の `div` タグで囲む。

```
<div xmlns="http://www.w3.org/1999/xhtml">
  <ul>
   <li>A</li>
   <li>B</li>
  </ul>
</div>
```

これに対し、HTML で終了タグ`</li>`が省略されている場合は以下のように CDATA セクションでエスケープする。

```
<![CDATA[
  <ul>
   <li>A
   <li>B
  </ul>]
```

3.3 入出力ファイル

入力ファイル: 3.1 節で述べたユーザ透過性を保証するために、入力ファイルは標準 Doclet の場合 [2] と同じである。

- **クラスソースコードファイル**: 各クラスまたはインタフェースのプログラムコード及び doc コメントを記述した Java ソースファイル。
- **パッケージコメントファイル**: 各パッケージの doc コメントを記述した HTML ファイル。`body` タグの内容に doc コメントを記述する。
- **概要コメントファイル**: Javadoc でドキュメント化しようとする全パッケージ及びクラスに対しての doc コメントを記述した HTML ファイル。`body` タグの内容に doc コメントを記述する。

これらのファイルは図 7 に示すディレクトリ階層とファイル名で保存しておく。末尾に'/' が付いたものはディレクトリである。ここでは、パッケージ java.applet の例を示した。パッケージドコメントファイルは、そのパッケージのソースファイルを格納するパッケージサブディレクトリに配置する。

```
srcRoot/                        ソースルートディレクトリ
  overview.html                 概要コメントファイル
  java/                         サブパッケージディレクトリ
    applet/
      Applet.java               クラスソースコードファイル
      AppletContext.java
      AppletStub.java
      AudioClip.java
      package.html              パッケージコメントファイル
```

図 7: 入力ファイル (パッケージ java.applet の例)

出力ファイル: XMLDoclet は、API ドキュメントを以下の一連のファイルとして出力する。このファイル構成と名前は標準 Doclet の場合 [2] とほぼ同じであるが、フォーマットが XML なので拡張子が.xml である。

- **基本内容ファイル**: API の説明を内容とする XML ドキュメント。各クラスとそのメンバを記述した**クラスドキュメントファイル**、各パッケージの概要を記述した**パッケージドキュメントファイル**、全パッケージ及びクラスに対する概要を記述した**概要ドキュメントファイル**がある。それぞれ、ルート要素の子要素は `class`、

package、overview である (overview の定義は [12] を参照)。

- **相互参照ファイル**: 基本内容ファイルの参照を便利にするための XML ドキュメント。各パッケージ内のクラスの継承関係を記述した**パッケージクラス階層ドキュメントファイル**と全クラスの継承関係を記述した**クラス階層ドキュメントファイル**、全 API の名前を記述した**索引ドキュメントファイル**がある。それぞれ、ルート要素の子要素は hierarchy と index である (それぞれの定義は [12] を参照)。

- **サポートファイル**: API 情報の参照には直接必要のないファイル。ドキュメント化したパッケージを列挙したテキストフォーマットの**パッケージリストファイル**、ビューを作成するための**スタイルシート**がある。

これらのファイルは、ユーザがコマンドオプションで指定した出力先ディレクトリに図 8 のように格納する。

```
docRoot/                        出力先ルートディレクトリ
  overview-summary.xml          概要ドキュメントファイル
  hierarchy-all.xml             クラス階層ドキュメントファイル
  index-all.xml                 索引ドキュメントファイル
  package-list                  パッケージリストファイル
  package-list-xml              パッケージリストファイル
  stylesheet.xsl                スタイルシートファイル(XSL)
  stylesheet.css                スタイルシートファイル(CSS)
  java/                         サブパッケージディレクトリ
    applet/
      Applet.xml                クラスドキュメントファイル
      AppletContext.xml
      AppletStub.xml
      AudioClip.xml
      package-summary.xml       パッケージドキュメントファイル
      package-hierarchy.xml     パッケージクラス階層ドキュメント
                                ファイル
```

図 8: 出力ファイル (パッケージ java.applet の例)

4 XMLDoclet の実装

4.1 DOM ツリーの生成とシリアライズ

XMLDoclet は、Doclet API を用いて API 情報を取り出し、出力するドキュメント毎に DOM [8] の Document オブジェクトを生成する。XML プロセッサには Apache の Xerces と Xalan[9] を用いたが、ユーザが任意の XML プロセッサを指定することもできる。

Document オブジェクトは 3.3 節で示したファイルに保存する。ファイルへの出力には Xalan を用いた。Xalan は本来は XSLT プロセッサ [6] であるが、スタイルシートを使用せずに、変換後の出力先にファイルを指定することで、シリアライザとして利用できる。

4.2 名前参照の解決

2.3 節で述べた名前参照の解決、すなわち参照先ドキュメントの URL を求める方法を示す。基本的には標準 Doclet と同じ方法 [2] であるが、プログラム透過性を実現するための拡張を行った。

4.2.1 内部参照の解決

ドキュメントの出力先ルートディレクトリを docRoot とする。以下の手順で参照先の URL を求める。

1. 参照元から docRoot への相対パスを求める。参照元の言語要素が含まれるパッケージを p とすると、p 内のサブパッケージ名の数だけ上位ディレクトリを辿る。例えば図 8 のパッケージ java.applet の場合、docRoot への相対パスは ../../である。

2. 次に、参照先の言語要素を含むパッケージについて、docRoot からそのパッケージサブディレクトリへの相対パスを求める。例えばパッケージ java.lang の場合、java/lang となる。

3. 1 と 2 の結果を連結し、参照先がパッケージの場合はパッケージドキュメントファイル名 package-summary.xml を、クラスの場合はクラスドキュメントファイル名"クラス名.xml" を、メンバの場合は"クラス名.xml#メンバ名"を連結したものが参照先 URL である。ただし、メンバがコンストラクタまたはメソッドの場合、その名前の後に完全修飾名のシグネチャを連結する。

例えば、クラス java.applet.Applet からクラス java.lang.Object のメソッド equals(Object) への内部参照を解決すると、../../java/lang/Object.xml #equals(java.lang.Object) という URL を得る。

4.2.2 外部参照の解決

ドキュメントの生成時に、パッケージリストファイル package-list と package-list-xml (内容は同じ) を生

成しておく [3.3 節]。例えば、Java 2 Platoform API では、その内容は以下のようになる。

```
java.applet
java.awt
java.awt.color
...
```

外部参照を行う場合、参照先の一連のファイルが存在するディレクトリの URL(extRoot とする) をユーザがコマンドオプションで指定する。以下の手順で参照先の URL を求める。

1. extRoot からパッケージリストファイルを読み込む。
2. 参照先の言語要素を含むパッケージがパッケージリストに含まれているならば、内部参照の解決方法と同じ方法で extRoot から参照先の言語要素への相対 URL を求める。extRoot にこの相対 URL を連結したものが参照先 URL である。

例えば、http://java.sun.com/products/jdk/1.3/docs/api に保存されたクラス java.lang.Object のドキュメントへの外部参照を解決すると、http://java.sun.com/products/jdk/1.3/docs/api/java/lang/Object.xml という URL を得る。

4.2.3 プログラム透過性の保証

プログラム透過性を保証するには、標準 Doclet と XMLDoclet がそれぞれ作成したドキュメント間で相互に外部参照を解決できる必要がある。標準 Doclet は HTML ファイルを生成し、パッケージドキュメントとクラスドキュメントのファイル名にそれぞれ package-summary.html と”クラス名.html”を用いる。これに対し、XMLDoclet は XML ファイルを生成し、それぞれの拡張子が.xml である。

まず、標準 Doclet が生成したドキュメントへの外部参照を XMLDoclet が解決する方法を示す。外部参照の解決手順の開始前に、パッケージリストファイル package-list-xml が存在するかどうかを調べる。package-list-xml が存在しないならば、参照先のドキュメントは標準 Doclet が生成したものと判断し、拡張子が.html のファイル名を用いる。そうでなければ、拡張子が.xml のファイル名を用いる。

次に、XMLDoclet が生成したドキュメントを標準 Doclet が外部参照する場合を考える。標準 Doclet はドキュメントが HTML か XML かを判断する方法を持たず、package-list ファイルを用いて拡張子が.html のファイルの URL を求めてしまう。そこで、XMLDoclet では、標準 Doclet が生成するのと同じファイル名の HTML ファイルを用意してリンクを保証する。この HTML ファイルは、XML を変換して生成するものと XML ファイルにリダイレクトするものとの 2 種類がある。前者の変換については次節で述べる。後者の HTML ファイルでは、以下の `meta` タグを利用して XML ファイルにリダイレクトする (この例はクラスドキュメントの場合)。

```
<meta content="1;url=クラス名.xml"
  http-equiv="refresh"/>
```

5 API ドキュメントのビュー

3.1 節で述べたように、我々はデータモデルとしての XML とそのビューとの分離を図っている。ビューの提供は、API 情報の応用の一つと位置付けている。しかし、標準 Doclet が提供する HTML のビューは、多くのプログラマにとって有用な利便性の高いものである。そこで、ビューの作成を支援するためのスタイルシートのリンク機能と、標準 Doclet と同等のビューを提供する XML 変換機能を用意した。

スタイルシート: XMLDoclet は、ユーザが指定したスタイルシート (XSL と CSS の 2 種類) を XML にリンクする。また、それぞれ組み込みのものを用意した。組み込み XSL は、XML を変換して標準 Doclet の出力とほぼ同じ HTML を生成する (つまり XSLT[6])。

組み込みの XSLT と CSS をリンクしておけば、XSLT プロセッサを組み込んだブラウザ (例えば Microsoft Internet Explorer) ならば XML ファイルを開くことで変換結果を表示できる。また、Xalan 等の XSLT プロセッサで変換して HTML ファイルに保存すれば、HTML ブラウザで表示できる。後者の変換と保存は、Javadoc の実行時に XMLDoclet で行うこともできる。

リンクの生成: 組み込みの XSLT で XML を HTML に変換するときに、名前参照に対する参照元と参照先のアンカーを生成する。参照先がクラスまたはパッケージの場合はファイルがアンカーなので何もせず、メンバの場合は HTML のタグ`<a name="...">`を生成し `name` 属性の値を完全修飾名とする。参照元では、要素 (例えば `see` や `type`) の `href` 属性から HTML のタグ`<a href="...">`を生成する。この `href` 属性

の値は、名前参照の解決で求めた URL である (4.2 節を参照)。

6 関連研究

Literate Programming: Knuth が提唱する Literate Programming[10] は、プログラムの作成とそのドキュメント化を同時に行う手法である (以下 LP と略す)。ソースコードとその説明文章とを一つのファイルにまとめておき、常に整合性の取れたプログラムとドキュメントの開発を目指す。

Javadoc も、ソースコードと説明文章を一つのファイルに記述する点は LP と同じだが、ドキュメントにソースコードを含められない点が異なる。Javadoc はソースファイルを読み込むときに実装コード (メソッドボディ) を無視するので、Doclet API では取り出せない。LP ではソースコードの理解を目的とするユーザ向けのドキュメントを生成しようとするのに対し、Javadoc ではプログラマがクラス (ライブラリ) を利用するために必要な API 情報 (宣言とその説明) だけをドキュメント化する。

XMLDoclet は、Javadoc の Doclet プラグインのしくみを利用して API 情報のデータモデルを XML で生成する。LP のような高度な文書整形システム (TEX 等) を利用したい場合は、XML を別のフォーマットに変換する応用プログラムや XSLT プロセッサで対応する。

SmartDoc: SmartDoc[11] は、技術ドキュメントの作成を目的とする XML ドキュメント構造の定義及び処理系である。SmartDoc のソースファイルは、処理系を使って HTML、LATEX、JavaHelp に変換できる。LP や Javadoc と異なりプログラムの説明や開発が目的ではないので、開発対象のソースコードを説明文章と同じ一つのファイルに記述することはない。ただし、別ファイルからソースコードを取り込むことができる。

SmartDoc のねらいは、XML の長所を活かし、単一のソースを多目的に利用することである。XMLDoclet が生成する API ドキュメントも同じことをねらっている。XMLDoclet が生成する API ドキュメントと SmartDoc のソースコード取り込み機能を組み合わせれば、LP の目的に利用できる可能性がある。

7 おわりに

本稿では、Java のソースファイルから XML フォーマットの API ドキュメントを生成する XMLDoclet について述べた。XMLDoclet が生成する API ドキュメントは、プラットフォームやプログラミング言語等の制約を受けず、API 情報のデータモデルとして様々な応用に利用できる。

XMLDoclet は、Javadoc に組み込みの標準 Doclet との相互運用性を保証するように設計されている。ユーザは、ソースファイルの作成時にどちらの Doclet を使うか意識する必要がなく、またどちらで作成したドキュメントであっても相互にハイパーリンクを形成できる。

今後、RELAX[7] 版スキーマの記述と、Javadoc の次期バージョン [3] の新機能である Java プログラムからの Javadoc 呼び出し機能への対応を行う予定である。XMLDoclet の最新情報は [12] から入手できる。

参考文献

[1] Sun Microsystems: *Javadoc Tool Home Page.* http://java.sun.com/j2se/javadoc/

[2] Sun Microsystems: *javadoc - The Java API Documentation Generator.* http://java.sun.com/j2se/1.3/docs/tooldocs/{solaris,win32}/javadoc.html

[3] Sun Microsystems: *Javadoc 1.4 Tool.* http://java.sun.com/j2se/1.4/docs/tooldocs/javadoc/

[4] W3C Recommendation: *Extensible Markup Language (XML) 1.0 (2nd Edition)*, 6 Oct 2000. http://www.w3.org/TR/REC-xml

[5] W3C Recommendation: *XHTML™1.0: The Extensible HyperText Markup Language*, 26 Jan 2000. http://www.w3.org/TR/xhtml1/

[6] W3C Recommendation: *XSL Transformations (XSLT) Version 1.0*, 16 Nov 1999. http://www.w3.org/TR/xslt

[7] Murata, M.: *The official site of RELAX.* http://www.xml.gr.jp/relax/

[8] W3C Recommendation: *Document Object Model (DOM) Level 2 Core Specification Version 1.0*, 13 Nov 2000. http://www.w3.org/TR/DOM-Level-2-Core/

[9] *The Apache XML Project.* http://xml.apache.org/

[10] Knuth, D.E.: Literate Programming, *The Computer Journal*, Vol.27, No.2, pp.97-111, 1984.

[11] Asami, T.: *SmartDoc.* http://www.asahi-net.or.jp/ dp8t-asm/java/tools/SmartDoc/

[12] Nakamura, A.: *XMLDoclet Home Page.* http://staff.aist.go.jp/nakamura-akihito/xmldoclet/