電脳螺旋網公司　2002年下期決算報告書

RELAX NG + Relaxerで

Java / XMLを2.56倍使うための本

どなどな（電脳螺旋網公司）・著

商標

本書に掲載されている会社名、製品名は、一般に各メーカーの登録商標または商標です。
なお本文中には™、®、©などを明記していません。

目次

目次 **3**

XML 周辺事態。 **6**

そもそも XML とはなんぞや。 6

CSV と XML の違い *7*

XML とスキーマとの関係 9

DTD の問題点 *9*

DTD に代わるもの *11*

スキーマの比較 *12*

XML 開発に向いたプログラミング言語 14

XML パーサとパーサ API *14*

Java 言語と XML な関係 *15*

RELAXER を触る用意をする 16

Java 開発環境 *16*

J2SE 1.4.1 のインストールと注意点 *17*

Relaxer のインストール *18*

XML 入門 **21**

XML 文書の種類 21

整形式なＸＭＬの構成 22

ＸＭＬ宣言 *23*

ＸＭＬインスタンス *23*

要素 *23*

ルート要素 *24*

空要素 *25*

属性(Attribute) *26*

命名規則 *26*

使えない文字 *27*

CDATA *27*

コメント *28*

スキーマ 29

DTD スキーマ *30*

文書型宣言 *30*

XML にまつわるエトセトラ 32

名前空間(Namespace) *32*

文字コード *33*

RELAX NG 入門 **35**

基本構成 35
要素 element *35*
属性 attribute *36*
出現条件 37
択一出現 choice *37*
グルーピング group *38*
省略可能 optional *39*
繰り返しの定義 zeroOrMore, oneOrMore *40*
パターンの定義 42
パターン定義と参照 define, ref *42*
文法構成 grammar, start *43*
出現条件の特殊なケース 44
混合内容モデル mixed *44*
出現順を無視するパターン interleave *45*
要素の中身・属性の値 47
空 empty *47*
普通のテキスト text *47*
データ型指定 data *48*
データ制約 param *50*
データ候補 value *50*
名前空間 NS 52

RELAXER で開発してみる **53**

RELAXER を触ってみる。 53
スキーマを生成する 54
スキーマをコンパイルする *55*
Hello.java を操作してみる *55*
Relaxer オブジェクトを操作する *56*
Relaxer のコマンドラインオプション *59*
演習(1) JUNKNEWS アプリケーション 61
アプリケーションのネタ *61*
junknews スキーマ *62*
演習(2) テキスト入力系 68
テキストファイルの書式を検討する *68*
テキストを読み込み、Relaxer オブジェクトを操作する。 *69*
演習(3) XML ドキュメントから HTML を起こす 78
XSLT とはなんぞや。 *78*

Relaxer XSLT 連携 *79*
HTMLTranspoter を実装する *83*
APPENDIX.A) RELAXER コマンドラインオプション一覧 **86**
APPENDIX.B) RELAXER に関するライセンス **90**
APPENDIX.C) 参考文献 **91**
APPENDIX.D) J2SE 1.3 以前の場合の JAXP の利用 **93**
APPENDIX.E) 付録 FD について **94**
利用に際しての諸注意 94
編集後記 - 後の末莉 **95**

動作環境

以下の環境で動作検証をしています。

OS	Windows 2000 Professional	Windows XP Professional
CPU	Cereron 633MHz	PentiumIII 850MHz
メモリ	384MB	256MB
JDK	J2SE 1.3.0	J2SE 1.4.1
JAXP	-	JAXP 1.1
エディタ	xyzzy 0. 2.2.227 + xml-mode	
ブラウザ	Opera 6.05, InternetExplorer 6.01	

XML 周辺事態。

XML、XML と言われ始めて幾星霜……ってほど時間も経っていないのだが、さすがに

"ブーム"

というレベルの熱さはなくなって来ているだろう。

というのも『XML っていうものは何でも解決できる魔法の杖なのだ』という強烈な誤解がギョーカイ内を席捲したために、クライアントもサプライヤも過度の期待を寄せたりして、それに乗じた（もしくは勘違いした）IT コンサルティングな連中だとか SIer と呼ばれている輩だとかが成熟していないソリューションを展開した結果、クライアント側は

見たくも無い「夢」と「現実」の乖離

を直視せざるをえない状況に立たされたりして、結局は多大な費用を掛けて構築した、例えば『ナレッジマネジメントしましょ！』とかいうフレコミの社内文書管理システムが使われないまま塩漬けになってしまった……なんて記事が専門誌に踊ったのが 1 年ほど前の動きだった。

しかし、だからといって、XML 自体が使われなくなったわけではない。いわゆる e ビジネスの分野や分散処理の分野でむしろ中心技術として XML が利用されている。そういう意味では XML というのは依然として熱いというか、単に一般的で標準的な技術になっただけだろう。

そもそも XML とはなんぞや。

XML は魔法の杖ではないにしても、キチンと使われている。つまり使い方さえ正しければ有用な技術となるようなのだが、XML とは一体なんだろうか。
まぁ、次章で詳しく文法面については触れるが、

とりあえず大雑把な解説

ということで、その手の用語辞典などで調べてみると大抵こんなことが書いてある。

eXtensible Markup Language [1]

拡張マークアップ言語規約。テキストベースのタグ付きフォーマットを定義するためのマークアップ言語。
SGML のサブセット。HTML と比較して、独自定義タグ、リンク構造強化、1 対多・拡張リンクなどがある。
W3C（The World Wide Web Consortium）提唱 （英辞郎より引用）

[1] 正確には Extensible Markup Language である。この手の間違った記述は多いみたいだ。

すなわち「テキストベース」で「タグ付フォーマットを定義する」ための**規約記述言語**に過ぎないのだ。それで表現されるものというのは単なるデータということになる。
そういう意味では従来から存在し、未だに大きなちぃ…もとい地位を占めている

ＣＳＶ

とそんなに変わらない気がするし、狭義的にはそれで間違っていないだろう。

CSV と XML の違い

じゃあ CSV と何が違うのだろうかというか。

まず、ツリー構造が表現できないことが挙げられる。
CSV はテーブル構造での表現は得意であるが、データ表現がテーブルでしか行えないのは扱えるデータの範囲が狭くなってしまう。最近 LDAP などのツリー構造データも多く、そのエクスポートを CSV でやるのは大変というか、あとで処理するのが大変だから歓迎したくないところである。

また、データ構造定義方法がルール化されていない。
XML も CSV もデータ構造定義なしで処理を行うことのできる柔軟性をもっているが、XML はそれに併せてデータ構造定義を行うことができ、XML を処理するライブラリにデータ構造やデータ型を検証するための機能があるので、データ構造に関するチェックロジックをプログラム中に書く必要がなくなる。
なお、データ記述方法としてカンマ区切りにするという CSV のスタイルであるが、たとえば TAB 区切りという別表現もあるし、1 行目がタイトル行かデータ行か判断できないし、項目をダブルクォーテーションで挟むか否かというものもあるし、項目内の改行表現に関する扱いにいたってはマチマチである。
つまり、ローカルルールだらけであるわけだが、XML では **W3C**(The World Wide Web Consortium)により仕様が制定されており逸脱記述についての心配がない。

日本などに特有の問題として文字コードの問題がある。CSV はいわゆる ASCII コードの範囲では問題ないのだが、日本にはメジャーな文字コードだけで Shift-JIS、JIS(ISO-2022-JP)、EUC-JP の三つがある[2]。さらに個人レベルではあまり関係ないが、多国語環境にも対応できない。
一方、XML は XML 1.0 仕様で XML パーサと呼ばれるライブラリは UTF-8, UTF-16 [3]の解析が出来なければならないとされている。
すなわち Unicode で記述すれば、XML が処理できる環境を事実上無視できるということだ。構造定義をいう取り決めだけを意識すればよくなる。無論、その構造定義を策定することが最大の難関ではあるのだが（^^;

[2] さらにいえば Shift-JIS の「方言」も多い。
[3] いわゆる Unicode。実際には完全一致ではないが。

もちろん XML にも、次のようなデメリットがある。

第一に挙げられるのは XML を用いると、他のデータ形式に比較してデータサイズが大きくなることである。バイナリベースで定義すれば圧倒的に、テキストベースでも UTF などを使わなければデータ量を削減できる。しかし Unicode 記述によりいわゆる半角文字でも少なくとも 2 バイト以上の長さを必要とする。
また、構造解析が複雑であるため処理コストが掛かってしまうことも挙げられる。
固定長やカンマ区切りのように単純な列構成であれば解析処理も簡単に行えるが、タグによる要素表現、単純な列ではないツリーベースの要素列挙が行われるので処理は当然遅くなるわけである。

そういう意味では一昔前に比べると無限に等しいハードウェアリソースを勝ち得た今だからこそ XML の普及が可能となったとも言え、ソニーコンピュータサイエンス研究所の増井俊之氏が提唱されるところの「富豪的プログラミング」[4] においてはＣＳＶに代わるカウンタパートとなる技術、いわば

富豪的データ

ということになるのかもしれない。

[4] 富豪的プログラミング（http://www.csl.sony.co.jp/person/masui/fugo.html）はステキな考え方です。

XML とスキーマとの関係

その昔 **SGML**（Standard Generalized Markup Language）という自動テキスト処理を行うために開発された文書構造記述言語が存在した。HTML の解説本なんかで仰々しく歴史を紐解いている系統の本を読まれた向きは頭の片隅にその文字の並びが残っているかもしれない。というのも HTML は SGML を大幅に簡略化して産まれたという経緯があるからだ。その簡易性がなければ今日の WWW の普及はなかったとまで言われている。
ともあれそういう SGML はあらゆる文書の構造表現をするために仕様の塊のようになり、気軽に文法記述をするのは大変だった…らしい。そういった複雑さを解決すべく、またインターネット時代に合うカタチにするべく、SGML の仕様を簡略化して生まれたのが XML である。
つまり HTML と XML というのは SGML を親とした兄弟なのである。と、まぁありがちな歴史をまた紐解いてしまったが。

で、この SGML においてその定義文書の妥当性を検査するのに用いられていたのが DTD(Document Type Definition)である。HTML も W3C から勧告が出されるときには DTD が公開されている。HTML の上部にある DOCTYPE で参照されているのがそれである[5]。
そして XML 文書も XML 1.0 では DTD で妥当性検査を行うとされているのだが、この DTD が

とても問題があるもの

ものなのである。

DTD の問題点

問題点としては次の３つが挙げられる。

- XML との記述性の乖離
- データ型が存在しない
- 名前空間の欠如

まず、ツリー構造を表現できる XML の定義をするものであるにも関わらず、かけ離れた記述スタイルをとっているため、全体を見渡せず可読性が悪くなっていることを挙げたい。

[5] Web ブラウザでは HTML に対して厳格な検査を行わないので、DTD があってもあんまり意味はなかったりする。

Greeting.xml

```
<?xml version="1.0" encoding="Shift_JIS"?>
<hello to="hogehoge@hogehoge.com">
  こんにちわ
</hello>
```

という XML の DTD は次のようになる。

Greeting.dtd

```
<!ELEMENT hello (#PCDATA)>
<!ATTLIST hello to CDATA #REQUIRED>
```

このようにあまりに違う。
そもそも XML があらゆるテキストデータの構造表現できるように作られているのにもかかわらず、その検査用の XML で表現できていないというのはかなり致命的と思われないだろうか。

第二に、データ型の概念が存在せず、文字列としてのデータしか存在し得ないことが挙げられる。
先ほども書いたとおり DTD はもともと SGML の記述に用いていたので、当然 XML というものを意識していなくて当然だったともいえる。
また元来、文書構造記述定義のために生まれたものであったため、例えば「この要素は内容として正の整数を取る」「さらに 0 から 99 の整数しか取らない」「これは日付のデータである」といったデータ型定義に不向きであるため、多目的データの検証に向いていないということになる。

三点目に名前空間の概念の欠如がある。
複数のスキーマが交錯する XML 文書において、同じ要素名を共存させる要請もある。

サンプルデータ

```
<注文書>
  <商品>
    <名前>カフェ・モカ</商品>
    <サイズ>Tall</サイズ>
  </商品>
  <バリスタ>
    <名前>
      <姓>田中</姓>
      <名>太郎</名>
    </名前>
  </バリスタ>
</注文書>
```

バリスタと商品の子要素である「名前」要素であるが、その持つ意味合いは大きく異なり、検証のためには何らかの区別が必要になる。こういった複数のスキーマを共存させるために「**名前空間**」（Namespace）という概念を使用する。Java でいうところの Package によって同一クラス名を共存できると同じだと思っていただいて結構である。
DTD ではこの名前空間を記述することができないのである。

DTD に代わるもの

ここまで挙げた問題点は XML に望まれた将来性を損なうものになるため、業界内では

このままではいけない！

という声が当然のように上がり、次世代の XML の主導権を握ることもできるため、定義用言語の開発ラッシュという局面を迎えた。XDR や SOX などという様々なスキーマ言語が各社・団体によって作られたが、最終的に W3C が取りまとめる **XML Schema** に収束するかと思われたが、策定に参加する企業の思惑は錯綜して、度重なる協議の結果できあがったのは

複雑怪奇な仕様の塊

以外のなにものでもなかった。
仕様こそ完成したものの「仕様のすべてを実装するのが困難」といわれるところまでいってしまったのである[6]。データ形式の共通化を図れるはずの XML の世界で仕様に「準拠」しかしていないアプリケーションが存在すると、相互通信を行うこと自体に差し障りが出てくる。その状況は、

「看板に偽りあり」

と謗られても、~~ぅ~~ぐぅの音も出ない状況であるし、そもそも DTD が XML の将来を損なうのを回避するという目的に適わないことになってしまうのである。

そんな問題点を解決すべく、IBM の村田真氏を中心にしたグループが数学的理論をベースにした日本発のオリジナルスキーマ言語 **RELAX Core** を発表。斬新でシンプルな構成は支持され、JIS のテクニカルレポートを経て、ISO により承認され、国際規格となったのである[7]。
一方、W3C の重鎮 James Clark 氏が発表した **TREX** というスキーマ言語は ISO ではなく OASIS という団体での規格化に乗り出した。
ともに重厚・難解な XML Schema に対するアンチテーゼとして発生したスキーマ言語であるため、軽快・平易なものとなっている。しかし XML Schema と相反するスキーマ言語が乱立することで「緊急避難」で XML Schema を選択する者が出てくる可能性がでてきたため、二者が肩を組んで作られたのが、本書で取り上げる

RELAX NG

（リラクシングと読む。RELAX Next Generation を意味している）なのである。

---

[6] PowerPoint のチュートリアルが 300 ページを超えるということである。
[7] 現在、名前空間をサポートした RELAX Namespace の策定中である。

RELAX NG は XML Schema と異なり、全仕様を実装するのが比較的容易だということであり、実装の程度の違いから発生する不利益もないと予想される。

何しろ文法を理解するのが簡単なので、本書では RELAX NG を推していきたい。

スキーマの比較

それぞれをちょっと比較してみるため、サンプルとなる XML 文書のスキーマを記述してみよう。

Greeting.xml

```
<?xml version="1.0" encoding="Shift_JIS"?>
<hello to="hogehoge@hogehoge.com">
  こんにちわ
</hello>
```

RELAX NG (Greeting.rng)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="hello"/>
  </start>
  <define name="hello">
    <element name="hello">
      <attribute name="to">
        <data type="token"/>
      </attribute>
      <text/>
    </element>
  </define>
</grammar>
```

RELAX (Greeting.rxm)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<module xmlns="http://www.xml.gr.jp/xmlns/relaxCore">
  <interface>
    <export label="hello"/>
  </interface>
  <elementRule label="hello" type="string">
```

```
    <tag name="hello">
      <attribute name="to" type="token"/>
    </tag>
  </elementRule>
</module>
```

XML Schema (Greeting.xsd)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="hello" type="hello"/>
  <xsd:complexType name="hello">
    <xsd:simpleContent>
      <xsd:restriction base="xsd:string">
        <xsd:attribute name="to" type="xsd:token"/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

DTD (Greeting.dtd)

```
<!ELEMENT hello (#PCDATA)>
<!ATTLIST hello to CDATA #REQUIRED>
```

ぱっと見た感じ「何を意味しているのか」が考えやすいのは RELAX NG なのではないだろうか。

XML 開発に向いたプログラミング言語

実際に XML を絡めたアプリケーション開発に取り組むにあたって、問題になるのがどのような開発環境・実行環境を設定すればいいのかということにあるが、幸か不幸かスペック面の富豪性を要求したことにより、マルチプラットホーム性を勝ち得ている。
また先述したように業務の世界での需要があるため、ビジネスの場で主要なプラットホームとなるハード・OS や開発言語の大半に何らかの形で、程度の違いはあれども XML とのインターフェイスが用意されているのである。
つまり、ただ単純に「XML を扱う」という側面だけで見れば、どんな言語でも可能なので、

「COBOL だから XML なんて扱えませんョ（´･ω･`）」

なんてことは言えなかったりするのだ。いや、ほんとに。

XML パーサとパーサ API

XML 文書とプログラムを繋いでいるライブラリ、すなわち XML 文書の I/O を扱うためのライブラリとも言える。それを XML プロセッサもしくは **XML パーサ**と呼ぶ。
XML パーサは大抵 **DOM** (Document Object Model)・**SAX** (Simple API for XML)という API のいずれか、もしくはどちらもをサポートしている。

DOM は W3C が規定している文書のツリー構造を表現するためのオブジェクトモデルであり、要素や属性といった文書の構成物が**オブジェクト**として扱われる。DOM は文書全体をオブジェクトとして扱うため、メモリ上でツリーの追加・修正・削除を行ったうえで文書自身の更新をしたり、もちろん新規作成も行うことができる。
ぱっと見使いやすいように感じるが、文書構造を知らないとプログラムが組めない（オブジェクトの構成が分からないからだ）というデメリットのほかに、文書をメモリ上にすべて展開するため、かなりのリソースをかなり使ってしまうだけではなく、速度面でも不利だという一面がある。

一方、SAX はその名の通りシンプルなもので DOM のリソースコストや速度の問題を解決したものになっている。アプローチはオブジェクトにマッピングするのではなく、XML 文書を頭から走査していき、発見した要素などを順次アプリケーションに**イベント通知**するモデルを取っている。
もちろんこういう動きであるため、XML 文書の更新や作成といったことはできない。

どっちを使えばいいのかというと非常に難しい話になってくるのだが、それぞれの特性にあったものを使うべきだという

玉虫色の決着

にしか持っていきようがない。

Java 言語と XML な関係

どの言語でも大抵 XML パーサがあるのだが、その性能・機能・実勢の差は存在するし、そもそもパーサ API として DOM や SAX すらサポートされていない可能性すらあるわけで。
逆に XML と一番親和性が高い言語というと XML 発表以後に開発された言語である C#が挙げられるだろう。なんせコメントさえも XML で記述できるわけだから。

しかしながら本稿では **Java 言語**を採用した XML アプリケーション開発手法を展開していく。
最大の理由としてはサポートするライブラリやツールがそろっていることが挙げられる。
Java は C#登場以前から XML とシステム実装を繋ぐ開発言語として、XML を牽引してきた立役者であり、XML 関連規格で Java のライブラリを持っていないものは無いと言っていいだろう[8]。
特に XML パーサに関して言えば Java には DOM/SAX パーサを扱うための API として **JAXP**（Java APIs for XML Processing）が用意されており、これに対応した XML パーサを使えばパーサ製品に依存しないコーディングが出来、ソースの可搬性があがるというメリットがある。
また、最新版の Java の標準 API として JAXP も含まれているので、Java 開発環境をそろえるだけで即座に XML アプリケーション開発ができる、という状態になっているのだ。

無論、それだけではない。選択したキーとなってくるのが

RELAX NG と Relaxer

である。

RELAX NG はすでに説明している通り、スキーマを定義するための言語（スキーマ言語）である。
そして Relaxer というのはスキーマに適合する XML 文書データを操るための Java のソースを自動的に生成してくれる「**スキーマコンパイラ**」というカテゴリに属するツールなのである。
Relaxer のような便利なツールが.NET サイドにあればいいのであるが、残念ながら今のところ現在の Relaxer のレベルまで達しているものを寡聞にして筆者は知らないため、必然的に

Relaxer との親和性がある Java で行きましょう（＾＾）

という結論になるわけだ。

幸い、GUI でも CUI でもサーバサイドにしても、今なら Java のサンプルコードは入手しやすいのである。分かりやすい入門書や Web サイトも存在する。アプレット全盛期がなんだったのか[9] と思う勢いであるが、無闇に多い Java 信仰もこういうときは吉とでるわけだ。

---

[8] ただし、「Java 界が XML に向ける視線の熱さほど、XML 界は Java に目を向けていない」という指摘もある。特定の言語に固執することは XML の将来にとって不利益になるから当然であろう。
[9] 筆者が Java を本格的に触り始めた 3 年前も言うほど充実していなかった。Servlet/JSP に端を発する J2EE のおかげであろう。ただ世間はそこまで追いかけて来られていないような気がするが（^^;

Relaxer を触る用意をする

とりあえず Relaxer を使ってみようかということになるわけであるが、Relaxer を動かすのにあたって少なくとも Java の実行環境(Java Runtime Environment = **JRE**)が必要となるし、できあがった Java のソースをコンパイルするのにやはり Java の開発環境(Java Deveploment Kit = **JDK**)が必要となるため、必然的に JDK のインストールをすることになる。
というわけで、インストールしていない向きへの前提情報としてインストール手順を追ってみる。すでにインストールしている向きは次の項目をあっさりと読み飛ばして頂いて結構である。

Java 開発環境

Java の開発環境として JDK を Sun がライセンスに基づき配布している。
ライセンス供与を受けた企業のみが再配布を行える仕組みになっているため、おいそれと製品に JDK を付属できない。当然、本書の付録にも付けられない。

尚、JDK には今のところ、３種類ある。

- Java 2 SDK Enterprise Edition　(J2EE)
- Java 2 SDK Standard Edition　(J2SE)
- Java 2 SDK Micro Edition　(J2ME)

このうち、今回セットアップするのは２番目の J2SE となる。

また Relaxer で生成されたクラスソースの中身を見たりするだけではなく、開発を行っていくのにあたって、JBuilder や Eclipse などの IDE（統合開発環境）があれば非常に楽であるわけだが、世の倣いに従ってフツーの SDK 環境で行ってみたいと思う。
というか Relaxer がコマンドラインでの利用が前提になっているみたい[10]なので。

んなこと言ってもＩＤＥじゃなきゃやっていけない

という向きは先に挙げたような有償無償で提供されている IDE をぶちこんで頂いても何ら問題ない。Relaxer の実行時だけコマンドプロンプトを利用すればいいだけだから[11]だ。

そうそう大事なことを忘れていた。本稿では最新版の **J2SE 1.4 ベース**で話をすすめていく。
というのも XML プログラミングに必須となる XML パーサライブラリである JAXP が標準搭載されているからである。しかし、J2SE1.3 でも J2SE 1.2 でも JAXP さえあれば問題ない。JAXP を別で入れる場合の例を示しておくので、システム開発業に従事しており職場環境が古いんだと

[10] Relaxer の食わず嫌いを出さないために Eclipse の Relaxer 操作 View とか作ってみたいなぁとか自爆発言。
[11] ちゅーか、本書のサンプルは Eclipse 2.02＋コマンドプロンプトを使って開発しています（^^;

いう向きも安心して読み進めて頂きたい。

J2SE 1.4.1 のインストールと注意点

とりあえず Sun の Java2 SDK Standard Edition のページ(http://java.sun.com/j2se/)からダウンロードすることになるのだが、これがかなりデカい。J2SE の JDK だけでもでかいのにドキュメントもかなりのボリュームがある。合計 60MB ぐらいになるんじゃないだろうか。
ブロードバンドならいいが、ナローバンドな向きは**月刊誌 Java World**（IDG 刊行）の付録 CD-ROM を使うと良い。この雑誌には Relaxer の作者である浅海智晴氏が連載[12]を持っているだけではなく、Java + XML な開発をするにあたって有用な情報が多いので読んで損はない。
上記のような雑誌付録の CD-ROM ではなく、インターネットからダウンロードする場合は以下のサイトにアクセスし、Windows (各国語版)の j2sdk-1_4_1_01-windows-i586.exe[13] をダウンロードしていただきたい。同ページにある日本語版 J2SE v 1.4.0 ドキュメントも併せてダウンロードしておいた方がいいかもしれない。

Java2 Platform, Standard Edition ダウンロード

```
http://java.sun.com/j2se/1.4.1/ja/download.html
```

ダウンロードもしくは CD-ROM 上のセットアップファイルを実行する。
インストール先は別段どこでも構わないが、次の点に気をつけておいたほうが何かといい。

- スペースが間に入ったフォルダをまたがない (C:¥Program Files¥jdk1.4.1 など)
- マルチバイト文字の混じったフォルダをまたがない (C:¥開発言語¥jdk1.4.1 など)

以上を踏まえて、ここでは **C:¥jdk1.4.1** ということにするが適宜読み替えていただきたい。

そしてインストール後にやっておいてほしいのが環境変数の設定だ。
特にしなくても動くので、設定していない向きも多いようだが、Relaxer や Jakarta Project の Ant などのツールなどを使うにあたって必要となるので、ここで設定しておくべきである。
設定するべきなのは以下の内容だ。

```
SET JAVA_HOME=c:\jdk1.4.1
SET PATH=%JAVA_HOME%\bin;既存の PATH 設定
```

ここで肝心なのはここでコマンドラインでのコンパイルと実行のテストすることだ。
あとになって

コンパイルも実行もできないのですが（´Д｀;;）

なんてことになったら目もあてられないし、原因の切り分けをするのも大変だ。

---

[12] 『Java デザイン・ノート』。巻頭のラーメン話も楽しみです(笑)
[13] 当然だがバージョンが変わればファイル名が変わるので、適宜読み替えていただきたい。

手抜きのための努力だと思って面倒くさがらずにやりましょう。
それに javac コマンドやら java コマンドのパラメータを与える面倒さを知ってこそ、後々Ant がありがたくなるってものであるし、

若いうちは苦労を買ってでもしろ

ということでがんばっていただきたい。

尚、本書のコードは **C:¥src¥relaxer** というフォルダ以下に随時作成していく。

Hello.java （C:¥src¥relaxer¥javatest）

```
public class Hello {
  public static void main(String[] argv){
        System.out.println("Hello, World. Zap!Zap!Zap!");
  }
}
```

コンパイルと実行

```
C:\src\relaxer\javatest>javac Hello.java
C:\src\relaxer\javatest>java Hello
Hello, World. Zap!Zap!Zap!

C:\src\relaxer\javatest>
```

これできちんとコンソールに文字列が表示されれば確認完了だ。

Relaxer のインストール

ようやく真打 Relaxer のインストールとあいなった。長い道のりだった。
早速ダウンロードだ。公式サイトは **http://www.relaxer.org/** である。
原稿着手段階ではまだ最新版が 0.16.1 であったため、RELAX NG スキーマを扱えなかったのだが、先日(12 月中旬)待望の Ver.0.17 がリリースされたので、何も心配することはない。

Relaxer/Download

```
http://www.relaxer.org/download/index_ja.html
```

以上のダウンロードページから setup.zip をダウンロードしていただきたい。

ダウンロードしたファイルを任意のフォルダ（ここでは C:¥src¥relaxer）に置き、コマンドプロンプトで以下のようにしてインストールを開始する。

拡張子が ZIP になっているが実体はややこしいことに

実行可能な Java Archive (JAR) なだけ

なので、その手のツールで解凍しようとしてもダメなので注意すること。
上記ページにも手順は触れられているが念のため、こちらにも掲載しておく。
ちなみに Relaxer のインストール時のクラスローディングに多少時間がかかるが、

そういうもの

なので気にしないことである。

Relaxer のインストール

```
C:\src\relaxer>java -jar setup.zip
Install directory [default: C:\usr\local\lib\relaxer]: C:\usr\local\lib\relaxer
Command directory [default: C:\usr\local\bin]: C:\usr\local\bin

[Configuration]
Install directory = C:\usr\local\lib\relaxer
Command directory = C:\usr\local\bin

Type "yes" to install, "no" to re-enter, "exit" to exit
> yes
Extract archives...
Generate script...
  script = C:\usr\local\bin\relaxer.bat
  script = C:\usr\local\bin\relaxer
Done.
```

インストール先のフォルダ(起動バッチ格納先と実体部)を聞いてくるので、ここではデフォルトの c:¥usr¥bin と c:¥usr¥lib¥relaxer のままにしておく。これも任意のフォルダにしても構わないが、インストール時に環境変数の設定をしておきたい。
先程の JAVA_HOME の設定と同様の箇所になるのだが、PATH に c:¥usr¥bin を追加しておこう。

さて動作テストをするのだが、環境変数の設定をした場合、プロファイルの再読み込みをおこなうため Windows NT/2000/XP 系では一旦今開いているコマンドプロンプトを終了させる必要がある。Exit 等でざっくり抜けていただき、再度アクセサリなどからコマンドプロンプトを立ち上げ、次のコマンドを打つ。

```
C:\src\relaxer>relaxer -version
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021213) by asami@relaxer.org
```

これで Relaxer の動作環境に path が通っていることが確認できる。
ちなみにこのバージョン情報の出力は Relaxer に関する議論が行われている Relax-user-ml 上で不具合報告したりする際に重要な情報となるので、確認方法にこういうものがあることを頭の片隅に置いておいていただきたいところだ。

これだけでは Relaxer がちゃんと動いてるかどうか分からないじゃんという向きには、先ほど例として出した Greeting.rng の処理を実際にやってみよう。

とりあえず Greeting.rng を Relaxer で処理する

```
C:\Src\relaxer\hello>relaxer -dir:C:\Src\relaxer\hello\java Greeting.xml


C:\Src\relaxer\hello>dir java
2002/12/10  21:57    <DIR>          .
2002/12/10  21:57    <DIR>          ..
2002/12/10  22:17            12,644 Hello.java
2002/12/10  22:17             4,700 RStack.java
2002/12/10  22:17            26,439 UJAXP.java
2002/12/10  22:17           152,529 URelaxer.java
               4 個のファイル             196,312 バイト
```

ひとまずの動作確認はできたので、これから Relaxer での開発といきたいが、XML と RELAX NG のサワリを挟んでからとなる。
しばらくは実践と関係ないところが続くが、ちょっとだけ辛抱していただきたい。

XML 入門

これから Relaxer を用いたＸＭＬアプリケーションを開発していくことにあたり、いくら DOM などの表面的な API 操作がラッピングされているとはいえＸＭＬ文書について何も知らないのは

あまりにあまりなので

一通りのことをさらっと眺めていこう。

XML 文書の種類

ＸＭＬ文書には大きく分けて以下の 2 種類ある。

- 整形式の文書　(well-formed document)
- 妥当な文書　(valid document)

整形式の文書とはＸＭＬ文書としての形式を備えている（文法が正しい）XML 文書のことを指す。
妥当な文書とは整形式である文書であるとともに別途設けられた構造定義（スキーマ）による検証を通過した XML 文書のことをいう。
日本語でたとえるならば「象の子どもはキリンである。」という文章が整形式の文書であり「蛙の幼生はオタマジャクシである」というのが妥当な文書にあたる。
すなわち

カタチ（文法）は整っていても中身が妥当か検証されていない

のが整形式の文書となる。

ただし、どちらも正しいＸＭＬ文書としては成立し機能することに注意していただきたい。

整形式なＸＭＬの構成

整形式なＸＭＬ文書 PizzaOrder.xml をサンプルに使い、ＸＭＬ文書の構成について説明していく。

PizzaOrder.xml

```
<?xml version="1.0" encoding="Shift_JIS"?>
<pizza0rder>
  <customer id="HSJ00001">
    <name>PIZZA 太郎</name>
    <address>O阪市東Y川区○○１－２－３　104 号室</address>
    <sex type="male"/>
  </customer>
  <orderList>
    <order>
      <pizza name="シーフードピザ" size="S" quantity="1">
        <topping name="ダブルチーズ"/>
        <topping name="ブラックオリーブ"/>
      </pizza>
      <pizza name="デラックスピザ" size="M">
      </pizza>
      <sidemenu name="コーラ" quantity="3"/>
      <sidemenu name="シーサーサラダ" quantity="1"/>
    </order>
  </orderList>
</pizza0rder>
```

ＸＭＬ文書の構造を二つに大きく分けると、以下の二つに分かれる。

- ＸＭＬ宣言
- ＸＭＬインスタンス

ＸＭＬ宣言

先ほどの PizzaOrder.xml では次の行が **XML 宣言**にあたる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

いわばＸＭＬ文書においての「おまじない」になる部分である。
ここでは version と encoding という値を指定している。
前者の version に関しては XML 1.0 に従った文書であるということを指している。
後者の encoding はこの文書がどの文字コードセットで書かれているかを指すものであり、ここでは Shift-JIS で書かれているとしている。
おそらくユーザがいじるとすると後者の encoding になると思われる。
ここで指定できる文字コードの識別子にはルールがあり、例えば「encoding="Shift-JIS"」とすると、正しく処理されない。

日本で使われるだろう代表的な文字コード 5 つの識別子を挙げる。

文字コードセット	識別子
Shift-JIS	Shift_JIS
JIS	ISO-2022-JP
EUC-JP	EUC-JP
UTF-8	UTF-8
UTF-16	UTF-16

なお、XML1.0 の仕様では UTF-8、UTF-16 で記述されている XML 文書においてはこの XML 宣言を省略できるようになっている。

ＸＭＬインスタンス

XML インスタンスは XML 宣言以降の<pizzaOrder>...</pizzaOrder>までの部分すべてとなり、これが XML 文書の本体部分となる。
XML インスタンスを構成するものについて、順に説明していく。

要素

要素（Element）は「<」と「>」で挟まれた**開始タグ**と「</」と「>」で挟まれた**終了タグ**の間で構成される。

```
<要素名>要素の内容</要素名>
```

PizzaOrder.xml では pizzaOrder, customer, name, address, sex, orderList, order, pizza, topping,

sidemenu のそれぞれが要素として成立している。
要素は「要素の内容」として、要素とテキストデータを持つことができる。
PizzaOrder.xml では「PIZZA 太郎」「O阪市東Y川区○○１－２－３　104 号室」がそれに該当する。

尚、</要素名>を省略することはできない。HTML では終了タグを打たない記述が可能であるが、XML では必ず終了タグを記述しなければならない。

HTML では可能であるが XML では NG のケース

```
<ul>
  <li>・・・・
  <li>・・・・
</ul>
```

XML のルールにあわせると、こうなる。

```
<ul>
  <li>・・・・</li>
  <li>・・・・</li>
</ul>
```

また、要素の入れ子になった場合、後で開いたものを先に閉じる必要がある（つまり、スタックである）ため、以下のような記述は許されず、整形式として評価されない。

```
<customer id="HSJ00001">
  <name>
    PIZZA 太郎
    <address>
      O阪市東Y川区○○１－２－３　104 号室
    </name>
  </address>
</customer>
```

ルート要素

最上位の要素のことを**ルート要素**というのだが、XML 文書においてルート要素は一つしか存在できないことになっている。

つまり、以下の XML 文書は整形式ではない。

整形式ではない XML 文書(1)　customer と orderList がルート要素になっている

```
<?xml version="1.0" encoding="Shift_JIS"?>
<customer id="HSJ00001">
  <name>PIZZA 太郎</name>
  <address>O阪市東Y川区○○１－２－３　104 号室</address>
  <sex type="male"/>
</customer>
<orderList>
  <order>
    <pizza name="シーフードピザ" size="S" quantity="1">
      <topping name="ダブルチーズ"/>
      <topping name="ブラックオリーブ"/>
    </pizza>
    <pizza name="デラックスピザ" size="M">
    </pizza>
    <sidemenu name="コーラ" quantity="3"/>
    <sidemenu name="シーサーサラダ" quantity="1"/>
  </order>
</orderList>
```

整形式ではない XML 文書(2)　pizza 要素が二つルート要素として存在している

```
<?xml version="1.0" encoding="Shift_JIS"?>
<pizza name="シーフードピザ" size="S" quantity="1">
  <topping name="ダブルチーズ"/>
  <topping name="ブラックオリーブ"/>
</pizza>
<pizza name="デラックスピザ" size="M">
</pizza>
```

空要素

要素の終了タグを省略できないと説明したが、たとえば HTML におけるような BR や IMG のように要素の中身を持たないようなデータも存在する。
これを**空要素**と呼び、以下のいずれかの表現をする。

```
<要素名></要素名>
<要素名/>
```

属性(Attribute)

要素は**属性**というものを持つことができる。
属性は以下の形で記述する。

```
<要素名 属性名 1="属性値 1" 属性名 2="属性値 2" 属性名 n="属性値 n">要素の内容</要素名>
```

先ほどのサンプルでは次のものなどが該当する。

- <customer id="HSJ00001"> の id
- <pizza name="シーフードピザ" size="S" quantity="1"> の name, size, quantity

また、次のように空要素も属性を持つことができる。

```
    <sex type="male"/>
```

ちなみに属性値は必ず'(引用符＝アポストロフィ)もしくは"(二重引用符＝ダブルクォーテーション)で囲む必要があり、省略できない。よって以下の XML 文書は整形式ではない。

整形式ではない XML 文書（属性記述ミス）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<pizza name="シーフードピザ" size='S' quantity=1/>
```

また同じ属性名を併記することはできない。

整形式ではない XML 文書（属性併記）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!-- 個数は合計三つ…にはならない -->
<pizza name="シーフードピザ" size='S' quantity='1' quantity='2' />
```

命名規則

XML の要素名、属性名（名前）は以下の規則に準じなければならない。

- 先頭には英字・ひらがな・カタカナ、漢字、「_」(アンダーバー)のいずれかを使用する
- 二文字目以降は先頭の文字種類に加えて、数字や記号が追加される。
- 大文字小文字は区別される(ABC と abc は別）
- 文字の大小を問わず「XML」で始まる名前は使用できない（「xMLArchive」は NG）

文字の大小を区別するため、開始タグ<pizzaOrder>を終了タグ</PizzaOrder>で閉じることはできないが、逆に次の例は一応以下の形式では整形式と認められる。

整形式の XML…であるが、悪い例（属性名が同一ではない扱いになる）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<pizza name="シーフードピザ" size='S' quantity='1' Quantity='2' />
```

使えない文字

要素の中身で「<」などは XML 文書の区切り文字と見なしてしまい使用できない。
他にも「>」「&」「'」「"」も使用できない。
これらを用いるために**実体参照**という仕組みが存在する。

```
&実体名;
```

以下に定義済み実体名の一覧を示す。

文字	実体名	由来
<	lt	Lesser Than
>	gt	Greater Than
&	amp	AMPersand
'	apos	APOStrophe
"	quot	QUOTation

<と書けば「<」として扱われ、「&」としたい場合は「&」と書くことになる。
尚、&#文字番号;という記述をすることで任意の文字番号の文字を出力できる。

CDATA

前項で「<」や「&」といった文字を直接記述できないと書いたが、次に示す **CDATA セクション**という部分ではテキストデータを「そのまま」取り扱うことができる。

```
<![CDATA[
  CDATA セクション
]]>
```

例えば次のようなスクリプトを埋め込んだり、

```
<?xml version="1.0" encoding="Shift_JIS"?>
<script><![CDATA[
function validateLesserThan(a, b){
  if(a < b){
        alert('ok!');
```

```
  }else{
        alert('ng');
  }
}
]]>
</script>
```

本稿のようなタグなどを表現する必要のある文書を記述しているケースが考えられる。

```
<document><![CDATA[
  <customer id="HSJ00001">
    <name>PIZZA 太郎</name>
    <address>O阪市東Y川区○○１－２－３　104 号室</address>
  </customer>
]]>
</document>
```

ただし、上記の例では「document」は要素として認識されるが、「customer」や「name」「address」は要素として認識されず、document 要素の中身であるテキストとして扱われる点を注意したい。
また、この CDATA セクションにおいて「]]>」は記述できないので、これも注意したい。

コメント

XML 文書内に例えば修正履歴など認識される必要のない情報を**コメント**として記述が出来る。

```
<!-- コメント -->
```

たとえば次のような複数行であったり、中にタグを記述するようなことも可能である。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!--  コメントの例 -->
<!--
  <caution>quotity と Quantity は区別されるが、このような書き方はするべきではない。</caution>
 -->
<pizza name="シーフードピザ" size='S' quantity='1' Quantity='2' />
```

ちなみにコメント中の要素記述は要素として認識されない。
なお、コメント中に「--」という文字列は使用できないので、次のコメントは認められない。

```
<!-- あー、ほんとに原稿間に合うのかなぁ(--;; -->
```

スキーマ

たとえば先ほどの PizzaOrder.xml の文書を別の人が次のように書いたとする。

OtherPizzaOrder1.xml

```
<注文書>
  <顧客データ 顧客番号="HSJ00001">
    <名前>PIZZA 太郎</名前>
    <住所>O阪市東Y川区○○１－２－３　104 号室</住所>
    <性別 type="male"/>
  </顧客データ>
  <注文明細>
    <注文>
      <ピザ 名前="シーフードピザ" サイズ="S" 数量="1">
        <トッピング 名前="ダブルチーズ"/>
        <トッピング 名前="ブラックオリーブ"/>
      </ピザ>
      <ピザ 名前="デラックスピザ" サイズ="M">
      </ピザ>
      <サイドメニュー 名前="コーラ" 数量="3"/>
      <サイドメニュー 名前="シーサーサラダ" 数量="1"/>
    </注文>
  </注文明細>
</注文書>
```

確かに内容自体は同じであるが、処理する側のプログラムは PizzaOrder.xml と注文書の両要素を同一視することはできない。
また、勝手に次のようなことをされても困る。

OtherPizzaOrder2.xml

```
<order>
  <pizza name="シーフードピザ" size="１人前" quantity="TWO" price="無料">
    <topping name="ダブルチーズ"/>
    <topping name="ブラックオリーブ"/>
  </pizza>
</order>
```

size として想定されているのが、たとえば S, M, L の三種類だとすると「1 人前」と書かれてもさ

ばきようがない。また quantity(数量)として数値が来る想定なのだが英単語で書かれてしまっている。そして、price という属性を無理やり追加され、そこには「無料」の文字がある。
これでは形式こそ整っているが、妥当な文書としては認められない。

そこで必要になるのが、文書の妥当性を評価する「規則」であり、それが**スキーマ**(Schema)と呼ばれるものなのである。
先ほどの size の候補として S/M/L しか存在しないとか、price を勝手に指定できないとか、そういうのをスキーマに基づいて検証をかけることで規則を実現できる。
無論、たとえば「ピザの注文書記述規則」とかいう文書を配布して、注文書を記述する人の手にルール遵守を委ねるというプランもあるのだが、最終的に処理するだろうプログラムがその XML 文書を妥当かどうか判断する根拠が存在しないことになってしまう。

まとめるとスキーマが存在すれば、次のメリットが発生する。

- 文書の妥当性をプログラム的に自動判定することが可能
- 規則を厳密に定めることができる

DTD スキーマ

スキーマ言語にはいろいろある。
本稿で扱う RELAX NG の他にも W3C が制定している XML Schema などがあるが、XML 1.0 では文書の構造定義の手段として **DTD**(Document Type Definition)が用意されている。
逆に言えば XML 1.0 のレベルでは DTD の範囲でしか、妥当性の検証を行えないのだが DTD を書くのは大変である。本稿で紹介する Relaxer を用いれば、DTD よりもよっぽど記述しやすく見通しもいい RELAX NG によるスキーマから DTD を生成できるので、それを使うようにしたい。

文書型宣言

DTD による文書構造定義を XML 文書内に埋め込むことができる。
先ほどの pizzaOrder.xml を例にすると、XML 宣言と XML インスタンスの間に記述する。

文書型宣言をインライン宣言とする

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE pizzaOrder[
  <!ELEMENT pizzaOrder(customer, orderList)>
  ...略...
]>
<pizzaOrder>
  ...略...
```

```
</pizza0rder>
```

また、外部ファイルや公開されている DTD を用いる場合は次のような記述をする。

http://hsj.jp/xml/dtd/pizzaOrder.dtd を参照する

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE pizza0rder SYSTEM "http://hsj.jp/xml/dtd/pizza0rder.dtd">
<pizza0rder>
  ...略...
</pizza0rder>
```

同じディレクトリ階層に存在する pizzaOrder.dtd を参照する。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE pizza0rder SYSTEM "pizza0rder.dtd">
<pizza0rder>
  ...略...
</pizza0rder>
```

尚、詳しい DTD の記述方法については、

面倒くさいのでサクっと割愛

させていただく[14]。

[14] すいません。本気で DTD を書くことが嫌いで説明もままならんという（^^;

XML にまつわるエトセトラ

名前空間(Namespace)

XML には**名前空間**という概念が存在する。
XML 文書の種類を XML 文書自身に埋め込んでおくための概念であるのだが、そんなことをして一体何の意味があるのだろうか。

まず、XML 文書には拡張子やファイル名などで区別する基準がない
pizzaOrder だろうが、住所録だろうが、abc.xml などというファイル名にすることが出来るし、中身を見ない限りどういったスキーマで検証すればいいかは分からないようになっている。

次に、複数のスキーマが混在する XML 文書において、同じ名前を区別できない、例えば商品スキーマと担当者スキーマが混在した注文書文書があったとする。

NamespaceOrder.xml

```
<注文書>
  <商品>
    <名前>カフェ・モカ</商品>
    <サイズ>Tall</サイズ>
  </商品>
  <バリスタ>
    <名前>
      <姓>田中</姓>
      <名>太郎</名>
    </名前>
  </バリスタ>
</注文書>
```

二箇所に存在する「名前」という要素の意味が異なり、検証を正しく行われない可能性がある。
こういったことを回避するためには「こちらの名前は人をさす」「こちらの名前は商品名をさす」といった区別をする必要があり、そこで導入されるのが名前空間という概念となるのである。

名前空間を定義するには要素に xmlns 属性と接頭辞を用いる。

```
<接頭辞:ローカル要素名 接頭辞:ローカル属性名="属性値" xmlns:接頭辞="名前空間 URI">
</接頭辞:ローカル要素名>
```

使用時にローカル名に接頭辞をつける。

```
<itm:商品 xmlns:itm="http://hsj.jp/xml/ns/item/dummy">
  <itm:名前 xmlns:itm="http://hsj.jp/xml/ns/item/dummy">カフェ・モカ</itm:名前>
  <itm:サイズ xmlns:itm="http://hsj.jp/xml/ns/item/dummy">Tall</itm:サイズ>
</itm:商品>
```

親要素での名前空間の設定は子要素に継承される。

```
<itm:商品 xmlns:itm="http://hsj.jp/xml/ns/item/dummy">
  <itm:名前>カフェ・モカ</itm:名前>
  <itm:サイズ>Tall</itm:サイズ>
</itm:商品>
```

次のように親要素で名前空間を複数設定すれば、名前の衝突を防ぐことができるのである。

```
<注文書 xmlns:itm="http://hsj.jp/xml/ns/item/dummy"xmlns:nam="http://hsj.jp/xml/ns/name/dummy">
  <itm:商品>
    <itm:名前>カフェ・モカ</itm:名前>
    <itm:サイズ>Tall</itm:サイズ>
  </itm:商品>
  <nam:バリスタ>
    <nam:名前>
      <nam:姓>田中</nam:>
      <nam:名>太郎</nam:名>
    </nam:名前>
  </nam:バリスタ>
</注文書>
```

ちなみに名前空間 URI は重複しない名前をつけるためのものであり、

そこにスキーマが存在するわけではない [15]

ことに注意していただきたい。
Java におけるパッケージ名をドメインの逆順で付与するけれども、そのホストやサブドメインが存在するわけではないのと同様なのである。

文字コード

先ほど、XML 宣言の項で「UTF-8、UTF-16 の場合は XML 宣言を省略できる」と書いた。
これは XML1.0 の仕様において「XML 処理を行うプログラムは少なくとも Unicode に対応できること」と定められているためである。

[15] ややこしいが検証に必要な DTD なりスキーマは別のところに存在するので、それを使うことになる。

逆をいえば「encoding="Shift_JIS"」と書いたからといって、受け取る側のプログラムが

Shift-JIS の文字を解釈してくれる保証なんてまったくない

のである。
そのため、出来る限り Unicode で XML 文書を記述するべきである、ということになる。
特にグローバルに使われるだろうスキーマや多言語混合文書などは Unicode で記述されることが必須用件であるといえる。

しかし Windows のメモ帳でも UTF で保存することは可能であるが、やはり一般的には Shift_JIS を使うことになると思われる。そのため、本書でも「encoding=”Shift_JIS”」としている。
概ね日本国内で流通している XML パーサで Shift_JIS を認識できないものはほぼないだろうから、先ほどの心配はないだろう。

余談であるが XML 文書に使用できる文字は正確には Unicode ではなく ISO/IEC 10646 の文字集合ということになる（Unicode は ISO ではなく業界団体である「Unicode コンソーシアム」により制定されている）。

RELAX NG 入門

個人的な感想であるが、RELAX というスキーマ言語は「シンプルだ」という振れこみのわりに直感的に掴めなかったため苦手であったりする。おそらくツリー構造表現できるスキーマにもかかわらず DTD 的な表現で紹介されていたからかもしれない。
それに対して RELAX NG はそういった「ぱっと見」の複雑さが極力排除されており、やろうと思えばひたすらツリー構造のスキーマを記述することも可能である。結果的に非常に直感的に掴める間口が広く敷居の低いスキーマ言語にしあがっていると思う。
もちろん RELAX がベースに持っている数学的な理論なども根底にあるため、かなり細かいことをしようとしても対応できる懐の深さももっているようだ（筆者はそこまで使いこなせていない）。
リファレンスには役不足ではあるが、簡単に RELAX NG 文法説明を通した入門をしていきたい。

なお、リスト List、モジュール Modularity、名前クラス Name classes、注釈 Annotations については本書の範囲では扱わないため記載していない。RELAX NG Tutorial に掲載されているので、そちらをご参照願いたい[16]。

基本構成

XML 文書の中身を構成する要素と属性について、まず見てみよう。

要素 element

要素の定義[17]は次のように行う。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <text/>
</element>
```

[16] 当サークルでも引き続き RELAX NG を扱っていくので、何らかの形でフォローしたいと思う。
[17] RELAX NG の名前空間は http://relaxing.org/ns/structure/1.0 である。

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1>textvalue</e1>
```

また当然であるが、入れ子の定義もできる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <element name="c1">
    <text/>
  </element>
  <element name="c2">
    <text/>
  </element>
  <text/>
</element>
```

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1><c1>childtext1</c1><c2>childtext2</c2>textvalue</e1>
```

なお、スキーマ内での配置順（この場合では c1 と c2）はこの通りにしなければならないため、次の XML 文書は不適合となる。

不適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1><c2>childtext2</c2><c1>childtext1</c1>wrong XML</e1>
```

属性 attribute

属性の定義も基本的に要素定義と同様となる。
無論、要素の属性であるため、記述位置は element 要素の子要素となる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <attribute name="a1">
    <text/>
  </attribute>
```

```
  <attribute name="a2">
    <text/>
  </attribute>
  <text/>
</element>
```

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1 a1="a1text" a2="a2text">textvalue</e1>
```

要素と異なり、属性は記述順により左右されない。

出現条件

RELAX NG においてスキーマに特に指定していない場合必須項目となり、また記述回数が出現回数の縛りとなるようになっているが、これを任意項目にしたり、影響範囲をグルーピングしたりするためには以下の要素を用いて宣言する。

択一出現 choice

属性や要素が候補の中から一つ出現するパターンを表現することができる。
たとえば HTML における TR 要素の子要素として TD もしくは TH が出てくるようなイメージを考えていただければよいだろう。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="tr" xmlns="http://relaxng.org/ns/structure/1.0">
  <choice>
    <element name="td">...</element>
    <element name="th">...</element>
  </choice>
</element>
```

適合する XML 文書(1)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<tr><td>...</td></tr>
```

適合する XML 文書(2)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<tr><th>...</th></tr>
```

ここでは二者択一となっているが、候補はいくつあってもかまわない。

グルーピング group

choice 要素では通常、一つの element 要素や attribute 要素しか候補にできないが、一定の組み合わせを候補にしたい場合、グルーピングすることができる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <choice>
    <element name="c1">
      <text/>
    </element>
    <group>
      <element name="g1">
        <attribute name="a1">
          <text/>
        </attribute>
        <element name="g2">
          <text/>
        </element>
      </element>
    </group>
    <text/>
  </choice>
</element>
```

適合する XML 文書(1)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1><c1>...</c1></e1>
```

適合する XML 文書(2)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1><g1 a1="atr1"><g2>g2text</g2></g1></e1>
```

こうすることで複雑なスキーマ表現を記述できるようになる。

省略可能 optional

普段使わないような属性などを省略できるようにしたい場合などは optional 要素を親要素として囲んでやると、省略可能となる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <attribute name="a1">
    <text/>
  </attribute>
  <optional>
    <attribute name="a2">
      <text/>
    </attribute>
  </optional>
  <text/>
</element>
```

適合する XML 文書(1)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1 a1="a1text" >textvalue</e1>
```

適合する XML 文書(2)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1 a1="a1text" a2="a2text">textvalue</e1>
```

気をつけたいのが次のような記述をした場合である。

```
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <optional>
    <attribute name="a1">
      <text/>
    </attribute>
    <attribute name="a2">
      <text/>
    </attribute>
  </optional>
```

```
  <text/>
</element>
```

こうなると一瞬 a1 と a2 のいずれかを記述すればいいような気がするが、それは認められない。つまり a1 も a2 も記述しないか、a1 と a2 ともに記述するかのいずれかになる。
それぞれを省略可能にするなら、

```
  <optional>
    <attribute name="a1">
      <text/>
    </attribute>
  </optional>
  <optional>
    <attribute name="a2">
      <text/>
    </attribute>
  </optional>
```

あるいはどちらかを記述するか、省略するなら次のように記述することになる。

```
  <optional>
    <choice>
      <attribute name="a1">
        <text/>
      </attribute>
      <attribute name="a2">
        <text/>
      </attribute>
    </choice>
  </optional>
```

繰り返しの定義 zeroOrMore, oneOrMore

出現パターン、特に個数が限定できない繰り返しを表現するためには次に紹介する zeroOrMore 要素、oneOrMore 要素を用い、対象となるパターンを子要素として記述する。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="e2">
      <text/>
```

```
    </element>
  </zeroOrMore>
  <oneOrMore>
    <element name="e3">
      <text/>
    </element>
  </oneOrMore>
</element>
```

適合する XML 文書(1)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1>
  <e2>e2-1</e2>
  <e2>e2-2</e2>
  <e3>e3-1</e3>
  <e3>e3-2</e3>
</e1>
```

適合する XML 文書(2)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1>
  <e3>e3-1</e3>
</e1>
```

たとえば、次のように二つ以上の要素を囲んだ場合、出現順を維持したままの繰り返しとなる。

```
<?xml version="1.0"?>
<element name="e1" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="e2">
      <text/>
    </element>
    <element name="e3">
      <text/>
    </element>
  </zeroOrMore>
</element>
```

適合する XML 文書(3)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<e1>
  <e2>e2-1</e2>
  <e3>e3-1</e3>
  <e2>e2-2</e2>
  <e3>e3-2</e3>
</e1>
```

パターンの定義

パターン定義と参照 define, ref

今までパターンをルート要素からのひと塊として定義してきたが、小さなパターンを定義して参照することが可能である。サブルーチンの定義とそれを呼び出すイメージに例えられるだろう。

パターン定義

```
<define name="def1">
  <element name="d1">
    <empty/>
  </element>
</define>
```

パターン参照

```
<element name="e1">
  <ref name="def1"/>
</element>
```

これは下のスキーマ表現と同一となる。

```
<element name="e1">
  <element name="d1">
    <empty/>
  </element>
</element>
```

文法構成 grammar, start

ここまではルート要素として element 要素を使ってきたが、この define 要素は element の子要素として存在できないため、実際には次のような記述となる。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="e1">
      <ref name="def1"/>
    </element>
  </start>
  <define name="def1">
    <element name="d1">
      <empty/>
    </element>
  </define>
</grammar>
```

ルート要素である grammar 要素の子要素として start 要素と 0 個以上の任意の define 要素を配置できる。
start 要素は**検証の起点**となるパターンである。本書の範囲ではルート要素の記述パターンとなる。無論、部分的な検証を行うようなケースではルート要素とは限らない。

また、定義したパターン内で自分に対する参照をする、**循環参照**が可能である。
次の例はディレクトリとフォルダの関係を記述したもので、いわゆる Composite Pattern[18]である。

```
<define name="directory">
  <element name="directory">
    <zeroOrMore>
      <choice>
        <ref name="directory"/>
        <element name="file">
          <empty/>
        </element>
      </choice>
    </zeroOrMore>
  </element>
</define>
```

[18] GoF の Design Patterns の一つ。

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<directory>
  <directory>
    <directory>
      <file/>
    </directory>
    <file/>
    <file/>
    <file/>
  </directory>
  <file/>
  <file/>
</directory >
```

出現条件の特殊なケース

混合内容モデル mixed

HTML が絶好の例になるが、要素内にテキストデータと子要素が含まれるようなケースがある。これを**混合内容**という。

これをここまで紹介してきてきた文法で表現としようとすることは困難であるが、mixed 要素の子要素とすることだけで表現することができる。

```
<define name="paragraph">
  <element name="paragraph">
    <mixed>
      <zeroOrMore>
        <choice>
          <ref name="breakLine"/>
          <ref name="strong"/>
        </choice>
      </zeroOrMore>
    </mixed>
  </element>
</define>
```

```
<define name="breakLine">
  <element name="breakLine"><empty/></element>
</define>
<define name="strong">
  <element name="strong"><text/></element>
</define>
```

適合する XML 文書の一部

```
<paragraph>
  この文書は定義したスキーマに<strong>適合</strong>している。<breakLine/>
  この文書は<strong>妥当な文書</strong>である。<breakLine/>
</paragraph>
```

先に choice 要素と zeroOrMore 要素を使ったパターンを紹介したが、次のようなケースでは出現順・回数などは維持される。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="mix">
  <mixed>
    <element name="m1"><text/></element>
    <element name="m2"><text/></element>
  </mixed>
</element>
```

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<mix>
  text1<m1>m1text</m1>
  text2<m2>m2text</m2>
  text3
</mix>
```

出現順を無視するパターン interleave

HTML の head 要素の子要素として一つの title 要素と任意の meta 要素を持てるが、その出現順について特に指定されていない。

このように出現順を無視できるようにするのが interleave 要素である。

```
<element name="head">
  <interleave>
    <ref name="title"/>
    <zeroOrMore>
      <ref name="meta"/>
    </zeroOrMore>
  </interleave>
</element>
```

適合する XML 文書(1)

```
<head>
  <meta>...</meta>
  <title>...</title>
  <meta>...</meta>
</head>
```

適合する XML 文書(2)

```
<head>
  <title>...</title>
  <meta>...</meta>
  <meta>...</meta>
</head>
```

適合する XML 文書(3)

```
<head>
  <title>...</title>
</head>
```

要素の中身・属性の値

要素と属性のならびについて紹介してきたが、次はそれぞれの中身の定義について進めていく。

空 empty

HTML の BR 要素のような空要素を定義するには empty 要素を用いる。

```
<element name="p">
  <element name="br">
    <empty/>
  </element>
</element>
```

適合する XML 文書

```
<p>
  <br/>
</p>
```

interleave, group, oneOrMore, zeroOrMore, optional 内に empty 要素がある場合、無視される。なお、choice では何も記述しないというケースを作ることができる。

普通のテキスト text

要素の中身や属性の値が普通のテキストの場合、text 要素を用いる。通常、空要素で定義をする。

```
<element name="p">
  <element name="strong">
    <text/>
  </element>
</element>
```

適合する XML 文書

```
<p>
  <strong>強調表示（多分）</strong>
</p>
```

データ型指定 data

値が普通のテキストではなく、例えば数値であるとか URI などといったデータ型として定義するのに用いるのが data 要素である。
data 要素を用いるためには自分もしくは先祖要素において datatypeLibrary 属性でデータ型ライブラリを明示する必要がある。
XML Schema Part 2 のデータ型ライブラリを用いるのがポピュラーなのである。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<element name="profile" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <element name="name">
    <data type="string"/>
  </element>
  <element name="birthday">
    <data type="date"/>
  </element>
  <element name="homepage">
    <data type="anyURI"/>
  </element>
</element>
```

適合する XML 文書

```
<?xml version="1.0" encoding="Shift_JIS"?>
<profile>
  <name>どなどな</name>
  <birthday>1977-08-30</birthday>
  <homepage>http://hsj.jp/</homepage>
</profile>
```

使えるデータ型については以下の通りである。

ジャンル	XML データタイプ	説明	Relaxer による Java マッピング
文字列	String	文字列	java.lang.String
	normalizedString	正規化された文字列 (改行や TAB を含まない)	java.lang.String
	Token	トークン文字列 (改行・空白・TAB を含まない)	java.lang.String
	Name	XML 文法上の Name	java.lang.String
	QName	XML 文法上の QName	java.lang.String
	NCName	XML 文法上の NCName	java.lang.String
数値	Decimal	無限精度数値	java.math.BigDecimal
	Boolean	ブール値	boolean/Boolean
	Float	16bit 浮動小数点数	float/Float
	Double	32bit 浮動小数点数	double/Double
	Integer	無限精度整数	java.math.BigInteger
	nonPoitiveInteger	0 以下の無限精度整数	java.math.BigInteger
	negativeInteger	0 未満の無限精度整数	java.math.BigInteger
	Long	64bit 整数	long/Long
	int	32bit 整数	int/Integer
	short	16bit 整数	short/Short
	byte	8bit 整数	byte/Byte
	nonNegativeInteger	0 以上無限精度整数	java.math.BigInteger
	positiveInteger	1 以上の無限精度整数	java.math.BigInteger
	unsignedLong	0 以上の 64bit 整数	java.math.BigInteger
	unsignedInt	0 以上の 32bit 整数	long/Long
	unsignedShort	0 以上の 16bit 整数	int/Integer
	unsignedByte	0 以上の 8bit 整数	short/Short
時間	duration	経過時間	java.lang.String
	dateTime	日付 + 時間	java.sql.Timestamp
	Time	時間	java.sql.Time
	Date	日付	java.sql.Date
	gYearMonth	年月	java.lang.String
	gYear	年	java.lang.String
	gMonthDay	月日	java.lang.String
	gDay	日	java.lang.String
	gMonth	月	java.lang.String
XML1.0 互換	CDATE	XML1.0 仕様の CDATA	java.lang.String

	ID	XML1.0 仕様の ID	java.lang.String
	IDREF	XML1.0 仕様の IDREF	java.lang.String
	ENTITY	XML1.0 仕様の ENTITY	java.lang.String
	NOTATION	XML1.0 仕様の NOTATION	java.lang.String
	IDREFS	XML1.0 仕様の IDREFS	java.lang.String[]
	ENTITIES	XML1.0 仕様の ENTITIES	java.lang.String[]
	NMTOKEN	XML1.0 仕様の NMTOKEN	java.lang.String
	NMTOKENS	XML1.0 仕様の NMTOKENS	java.lang.String[]
その他	hexBinary	HEX 形式バイナリ	byte[]-
	base64Binary	BASE64 形式バイナリ	byte[]
	anyURI	URI	java.net.URL
	language	言語	java.util.Locale

データ制約 param

文字数の制限をかけるなど、データの型だけではなく中身に対しての制約をかけることができる。これを RELAX NG では**パラメータ**という[19]。

```
<element name="name">
  <data type="string">
    <param name="maxLength">32</param>
  </data>
</element>
```

このようにすると文字長を 32 文字以下に制限できる。
しかし、残念ながら Relaxer の現在のバージョンではこの制約機構が組み込まれない。
そのため、説明はこの程度とさせていただく。

データ候補 value

特定の値だけを受け入れたいケースなど、たとえば yes, no のいずれかを候補とする属性値を定義する場合に用いるのが value 要素である。

```
<element name="frame">
  <attribute name="scroll">
    <choice>
      <value>yes</value>
      <value>no</value>
```

[19] XML Schema でファセットと呼ばれているものと同じものと考えていい。

```
      <value>auto</value>
    </choice>
  </attribute>
</element>
```

適合する XML 文書

```
<frame scroll="yes" />
```

適合しない XML 文書

```
<frame scroll="false" />
```

また、整数(integer)もしくは "null" のどちらかしか存在しないような場合は次のように定義する。

```
<element name="intdata">
  <choice>
    <data type="integer"/>
    <value>null</value>
  </choice>
</element>
```

適合する XML 文書

```
<intdata>-1</intdata>
<intdata>500</intdata>
<intdata>null</intdata>
```

適合しない XML 文書

```
<intdata>3.14</intdata>
<intdata>PI</intdata>
<intdata>undefined</intdata>
```

名前空間 ns

RELAX NG は名前空間に対応しているので、element や attribute の検証時に名前空間も合わせて検証することが可能である。
ちなみに親要素での設定は上書きしない限り、子孫要素に継承される。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="nsTest" ns="http://www.spiralcable.net/ns/test">
      <attribute name="atr">
        <text></text>
      </attribute>
      <element name="nsTestChild">
        <empty/>
      </element>
   </element>
  </start>
</grammar>
```

適合する XML 文書(1)

```
<nsTest atr="abc" xmlns="http://www.spiralcable.net/ns/test">
  <nsTestChild/>
</nsTest>
```

適合する XML 文書(2)

```
<tst:nsTest atr="abc" xmlns:tst="http://www.spiralcable.net/ns/test">
  <tst:nsTestChild/>
</tst:nsTest>
```

適合しない XML 文書

```
<nsTest atr="abc" xmlns="http://www.spiralcable.net/ns/test2">
  <nsTestChild/>
</nsTest>
```

Relaxer で開発してみる

XML と RELAX NG についてみてきたが、ここでようやく Relaxer の再登場となる。
簡単な Relaxer の使い方や Relaxer が生成する Java クラス(Relaxer オブジェクト)についての説明を行い、実際にアプリケーションを開発するという流れで進めていきたい。

Relaxer を触ってみる。

Relaxer の機能をもう一度説明すると、 RELAX NG で記述されたスキーマからそのスキーマに適合する XML 文書を操るための Java クラスソース（DOM などの API をほとんど意識せずにオブジェクトを操作することが可能）を生成するスキーマコンパイラである。
しかし、Relaxer は設計から開発までいたる一連のプロセスを構成できるほどに

とてつもなく機能が豊富

なのである。

そんな機能の中に、XML 文書からスキーマを（類推して）生成するという機能がある。
わざわざイチから面倒くさいスキーマ記述をしなくても､扱うべき XML 文書を用意してやれば､少なくともその XML 文書を妥当だとみなせるスキーマが出来てしまうのである。

とりあえず一般常識に従い、次のような XML 文書を作成してみた。

Hello.xml(C:¥src¥relaxer¥hello に作成)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<hello from="me" to="you">
  <world>s*fm@p</world>
</hello>
```

まぁ、なんというか

一足先を走って

くれそうである。

スキーマを生成する

先ほど作成した Hello.xml を Relaxer で処理することにしよう。
インストール時に正しく Path 設定していれば、次のコマンドで実行できるはずである。

```
C:\src\relaxer\hello>relaxer -verbose -rng Hello.xml
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/hello/Hello.xml
        artifact = Hello.rng
```

ちなみにこのコマンドでは「-rng」で RELAX NG スキーマ[20]を作成、「-vebose」で動作中の詳細情報を出力という指定をしている。

Hello.rng が同フォルダに出来ているので、メモ帳などで開いてみよう。

Hello.rng

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="hello"/>
  </start>
  <define name="hello">
    <element name="hello">
      <attribute name="from">
        <data type="token"/>
      </attribute>
      <attribute name="to">
        <data type="token"/>
      </attribute>
      <element name="world">
        <data type="token"/>
      </element>
    </element>
  </define>
</grammar>
```

[20] RELAX NG スキーマのスタンダードな拡張子は rng である。

いやぁ、まったく一分の隙もない正しいスキーマであるが、通常こうはうまいこと行かない。
とりあえず修正を行うなら、この段階で行うこととなる。

スキーマをコンパイルする

スキーマを用意したら、早速 Relaxer で処理しよう。

```
C:\src\relaxer\hello>relaxer -verbose Hello.rng
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/hello/Hello.rng
        artifact = Hello.java
        artifact = URelaxer.java
        artifact = RStack.java
        artifact = UJAXP.java
```

Hello.rng を処理することで以下のファイルが生成された[21]。

Hello.java	14,928bytes
RStack.java	4,700bytes
UJAXP.java	26,439bytes
URelaxer.java	152,529bytes

この Hello.java が Hello.rng に適合する XML 文書を処理する機能が備えられている。
これを **Relaxer オブジェクト**という。

他の RStack.java, UJAXP.java, URelaxer,java などは Relaxer オブジェクトの内部挙動などに用いるライブラリクラスであるので、積極的に利用することは無いと思われる。
ちなみにこれらの Relaxer オブジェクト、ライブラリクラスを初めとした Relaxer から生成されたファイルのことを **Relaxer マテリアル**と呼ぶ。

Hello.java を操作してみる

Hello.java の中身を見る前に HelloCtrl.java というソースを書いてみる。

HelloCtrl.java

```
public class HelloCtrl{
  public static void main(String[] argv){
```

[21] このファイルサイズ、ファイル数は Version によって異なる可能性がある。

```
    Hello hello = new Hello();
    hello.setFrom("芥川龍之介");
    hello.setTo("菊池寛");
    hello.setWorld("わーるど");
    System.out.println(hello.makeTextDocument());
  }
}
```

まず Hello クラスのインスタンス hello を生成し、そのプロパティ From, To, World にそれぞれの値を突っ込んでいる。
最後に hello の makeTextDocument()というメソッドを呼び出している。
実際にどうなるか、実際にコンパイルし実行してみよう。

```
C:\src\relaxer\hello>javac HelloCtrl.java

C:\src\relaxer\hello>java HelloCtrl
<hello from="芥川龍之介" to="菊池寛"><world>わーるど</world></hello>
```

つまり、makeTextDocument()は現状の hello のインスタンスの内容をテキストの XML 文書として出力するためのものであったのだ。
ひとまず DOM の API などを意識しないで、

手軽に XML を扱える

という感覚はつかんでいただけたのではないかと思う。

Relaxer オブジェクトを操作する

先ほど、いきなり Hello クラスのインスタンスを操作するプログラムを書いてみたが、実際に Hello クラスはどういう構成になっているのだろうか。とはいえここに 15KB、570 行以上もある Hello.java を一挙掲載するとそれだけで紙面を埋め尽くすので、公開メソッドだけを抽出して以下の表にまとめてみた。

先ほど、いきなり Hello クラスのインスタンスを操作するプログラムを書いてみたが、実際に Hello クラスはどういう構成になっているのだろうか。とはいえここに 15KB、570 行以上もある Hello.java を一挙掲載するとそれだけで紙面を埋め尽くすので、公開メソッドだけを抽出して以下にまとめてみた。

まず、コンストラクタであるが 10 個もある。
およそ考えられるコンストラクタが自動生成されることで、ルーティンワークになりがちな手続きを簡略化できると共にそういうところからのバグの発生を防御できるようになっている。

コンストラクタ名	機能
Hello()	Relaxer オブジェクトを新規に生成する。
Hello (org.w3c.dom.Document doc)	DOM の Document ツリーのインスタンスを元に Relaxer オブジェクトを生成する。もちろん Hello.rng に適合する内容でなければ正しく読み込まれない。
Hello (org.w3c.dom.Element element)	DOM の部分木を元に Relaxer オブジェクトを生成する。
Hello (java.io.File file)	指定したファイル(ただしくは java.io.File のインスタンス)を元に Relaxer オブジェクトを生成する。 大抵は new Hello(new File("hello.xml")みたいな使われ方になると思われる。
Hello (org.xml.sax.InputSource is)	SAX で定義されている XML ドキュメント指定用オブジェクトから Relaxer オブジェクトを生成する。
Hello (java.io.InputStream in)	InputStream からの入力を元に Relaxer オブジェクトを生成する。他のオブジェクトから引き渡された InputStream を用いるなどが考えられる
Hello (java.io.Reader reader)	StreamReader からの入力を元に Relaxer オブジェクトを生成する。たとえば String から StringReader を作って、そこから Relaxer オブジェクトを生成するなど。
Hello (RStack stack)	Relaxer が内部で用いるコンストラクタであり、あまり利用者側が意識することはない。
Hello (java.lang.String uri)	指定された URI からリソースを読み出し、それを元に Relaxer オブジェクトを生成する。
Hello (java.net.URL url)	指定された URL からリソースを読み出し、それを元に Relaxer オブジェクトを生成する。

また、既にインスタンス化された Relaxer オブジェクトを初期化する setup メソッドが用意されている。

- setup(org.w3c.dom.Document doc)
- setup(org.w3c.dom.Element element)
- setup(java.io.File file)
- setup(org.xml.sax.InputSource is)
- setup(java.io.InputStream in)
- setup(java.io.Reader reader)
- setup(RStack stack)
- setup(java.lang.String uri)
- setup(java.net.URL url)

一覧を見ていただければ分かるが、コンストラクタとほぼ同じである。

コンストラクタ・初期化メソッドの引数を見ても分かるとおり、さまざまなリソースから Relaxer オブジェクトができるが、これらのメソッドでは Relaxer オブジェクトからは各種 XML ドキュメントを生成できる。

戻り値	メソッド名	説明
org.w3c.dom.Document	makeDocument()	DOM の Document オブジェクトを生成する。大抵の XML 関連のライブラリは Document オブジェクトをインターフェイスとして準備しているので、各種部品に応用できる。
Void	makeElement (org.w3c.dom.Node parent)	-
Java.lang.String	makeTextDocument()	テキスト形式の XML ドキュメントを String として返す。 出力される XML ドキュメントは CanonicalXML（空白や改行などを除去した正規化された XML 文書）に準拠したものである。
Void	makeTextElement (java.io.PrintWriter buffer)	ファイル作成・ネットワーク送信に利用している PrintWriter を介して要素や属性をテキストとして出力する。
Void	makeTextAttribute (java.io.PrintWriter buffer)	
Void	makeTextElement (java.lang.StringBuffer buffer)	プログラム内で利用している StringBuffer を介して要素や属性をテキストとして出力する。オブジェクト生成のコストを削減するのに最適である。
void	makeTextAttribute (java.lang.StringBuffer buffer)	

属性・要素アクセスメソッド、いわゆるアクセサも自動生成される。DOM では属性と要素の中身の扱い方が異なるのだが、Relaxer オブジェクトではどちらも get/set でのアクセスで統一されているので、あれこれ悩む必要がなくなっている。

戻り値	メソッド名	説明
java.lang.String	getFrom()	属性・要素の値を取得する。 From, To, World ともに String にマッピングされるため、戻り値がすべて String であるが、マッピングされているのが別のクラスである場合は戻り値がそのクラスのオブジェクトとなる。
java.lang.String	getTo()	
java.lang.String	getWorld()	
java.lang.String	getFromAsString()	属性・要素の値を String として取得する。 マッピングされたクラスがどうであれ、String として返す。
java.lang.String	getToAsString ()	
java.lang.String	getWorldAsString ()	
void	setFrom (java.lang.String from)	属性・要素の値を設定する。 From, To, World ともに String にマッピングされるため、戻り値がすべて String であるが、マッピングされているのが別のクラスである場合は引数がそのクラスのオブジェクトとなる。
void	setTo (java.lang.String to)	
void	setWorld (java.lang.String world)	
void	setFromByString (java.lang.String from)	属性・要素の値を String として設定する。 マッピングされたクラスがどうであれ、String として設定する。
void	setToByString (java.lang.String to)	
void	setWorldByString (java.lang.String world)	

また、オブジェクトが受け付けられる XML ドキュメントかを判定する isMatch メソッドが準備されている。これらのメソッドは static なクラスメソッドである。

戻り値	メソッド名	説明
boolean	isMatch (org.w3c.dom.Element element)	オブジェクトが受け付けられる XML ドキュメントかどうかを判定する。
boolean	isMatch (RStack stack)	Relaxer 内部で使われているメソッドであり、無視してもよい。
boolean	isMatchHungry (RStack stack)	

Relaxer のコマンドラインオプション

豊富な機能を誇る Relaxer であるが、連動して数多くのコマンドラインオプションが存在する。それぞれの簡単な説明は Appendix.A にまとめたが、本書で使っているオプションについて、ここで軽くまとめておきたい。

-verbose

前述しているが実行時の詳細情報を出力させるオプションである。
付けておくと進捗が分かるし、どれぐらいのファイルが生成されたかが記録として残るので、積極的に付けるようにしていただきたい。

-dir:フォルダ名

Relaxer が生成する Java クラスソースやスキーマファイルなどの出力先フォルダを指定する。
デフォルトはカレントディレクトリになるのだが、出力数がハンパじゃなくなるスキーマも多々存在するので、指定するように心がけよう。

```
>relaxer -dir:C:/src/relaxer/hello/java Hello.rng
```

-package:パッケージ名

Relaxer が生成する Relaxer オブジェクトにパッケージ名を付与する。
たとえば、A.rng と B.rng という二つのスキーマそれぞれから生成した Relaxer オブジェクトは同一パッケージ内に共存できないという制約がある。
使い捨ての Relaxer オブジェクトでない限り、付けておかなくてはならない。

```
>relaxer -package:com.hogehoge.relaxer.a A.rng
>relaxer -package:com.hogehoge.relaxer.b B.rng
```

-rng

先ほども説明したが、XML 文書から RELAX NG スキーマを生成するオプションである。

-dtd

これは-rng と同様、XML 文書や別のスキーマ(RELAX NG や RELAX のスキーマ定義)から DTD による文書型定義を生成するオプションである。
基本的に DTD でなければ XML1.0 レベルのパーサでは外部的な検証ができないため、こういうものを持ち出す必要があるのだ。

ちなみに先ほどの Hello.xml から次の DTD が生成したみた。

```
C:\src\relaxer\hello>relaxer -verbose -dtd Hello.xml
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/hello/Hello.xml
        artifact = Hello.dtd
```

Hello.dtd

```
<!ELEMENT hello (world)>
<!ATTLIST hello from CDATA #REQUIRED>
<!ATTLIST hello to CDATA #REQUIRED>

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

とりあえずこの辺を用いて以下の話を進めていくので、適当に覚えておいていただきたい。

演習(1) Junknews アプリケーション

例題程度の操作はこの程度にして、実際の XML アプリケーションの開発に移っていきたい。
手順としては先ほどと変わらず、

扱うべき XML 文書イメージを作成
↓
スキーマの生成＋手直し
↓
Relaxer オブジェクトの生成とコンパイル
↓
操作オブジェクトの作成
↓
ウマー(ﾟдﾟ)

というものになる。

アプリケーションのネタ

あまり現実的でない「注文書アプリケーション」とかを作ってもあまり意味が無いので、個人レベルでも多少役に立つだろうものとして、

テキストサイト構築支援アプリケーション

を提案してみる。

モデルとして、筆者がほぼ毎日更新している「興味があったページへのリンクに瑣末なコメントを付けたもの」を列挙している junknews というコーナーに適用してみようと思う。
現在、適当に巡回したサイトで興味を持ったリンクをガシガシと開いていき、その URL とタイ

トルを控えて、手動で HTML 化している。あらためて文章にして気づいたのだが

非常に泥臭い。

前々から何とかしたいなぁという思いこそあれ、あれやこれやと忙しい毎日を送っていてなかなか手が付けられなかったのである。これはまたとない良いチャンス！ということで公私混同も甚だしいが、実例ということで junknews というサイトのシステム化を展開していく。

junknews スキーマ

junknews は概ね次のようなパターンの情報で構成されている。

日付	日付単位で区切りを付けている。
ジャンル	かなり適当であり、ほとんどの場合でついていない。 また情報の順序もジャンルで整理されていない。
リンク＋タイトル	指定された URL を別ウィンドウで開くようにしている。 アンカー文字列は対象ページのタイトルとなっている
タイトル	たまにリンクのないタイトルがあったりする。
コメント	そのタイトルに関するコメントがあったりなかったりする。 行数も複数行になり、任意のタイミングで改行したい。 またリンクを貼ったりする場合もある。

以上のような情報を持ったデータが複数存在する、ということになる。
とりあえずこの条件に従った XML 文書を作ってみた。

junknews1.xml

```
<?xml version="1.0" encoding="Shift_JIS"?>
<junknews>
  <news date="2002/12/30">
    <title>タイトルおんりー</title>
    <comment>
      コメント<a href="about:blank" />リンク</a>もあったり、<strong>太字</strong>とかも。
      改行もしたいねぇ。
    </comment>
  </news>
  <news date="2002/12/30">
    <title href="">コメントなし</title>
  </news>
</junknews>
```

無論、こんなものを作ったのは Relaxer によるスキーマの生成機能を用いるためである。

あれこれ試行錯誤するにしても、基礎データがあるだけで

曇泥の差

である。

ではここからスキーマを起こそう。

```
C:\src\relaxer\junknews>relaxer -verbose -rng junknews1.xml
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/junknews/junknews1.xml
        artifact = junknews1.rng
```

すこし長くなるが、生成されたファイルを引用する。

junknews1.rng

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="junknews"/>
  </start>

  <define name="junknews">
    <element name="junknews">
      <oneOrMore>
        <ref name="news"/>
      </oneOrMore>
    </element>
  </define>
  <define name="news">
    <element name="news">
      <attribute name="date">
        <data type="token"/>
      </attribute>
      <ref name="title"/>
      <optional>
        <ref name="comment"/>
      </optional>
    </element>
```

```
  </define>
  <define name="title">
    <element name="title">
      <optional>
        <attribute name="href">
          <empty/>
        </attribute>
      </optional>
      <data type="token"/>
    </element>
  </define>
  <define name="comment">
    <element name="comment">
      <mixed>
        <zeroOrMore>
          <choice>
            <ref name="a"/>
            <ref name="strong"/>
          </choice>
        </zeroOrMore>
      </mixed>
    </element>
  </define>
  <define name="a">
    <element name="a">
      <attribute name="href">
        <data type="token"/>
      </attribute>
    </element>
  </define>
  <define name="strong">
    <element name="strong">
      <data type="token"/>
    </element>
  </define>
</grammar>
```

結構大量なスキーマが勝手に作られて、

ウマー(ﾟдﾟ)

作られたスキーマを 3 章のクイックリファレンスを見ながら解読すると、修正したいと思われる点が何箇所か出てくると思われる。修正したものを以下に示す。

junknews2.rng

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <ref name="junknews"/>
  </start>
  <define name="junknews">
    <element name="junknews">
      <attribute name="date">
        <data type="token"/>
      </attribute>
      <oneOrMore>
        <ref name="news"/>
      </oneOrMore>
    </element>
  </define>
  <define name="genre">
    <element name="genre">
      <attribute name="name">
        <data type="token"/>
      </attribute>
      <oneOrMore>
        <ref name="news"/>
      </oneOrMore>
    </element>
  </define>
  <define name="news">
    <element name="news">
      <attribute name="date">
        <data type="token"/>
      </attribute>
      <optional>
        <attribute name="genre">
          <data type="token"/>
        </attribute>
```

```
    </optional>
    <ref name="title"/>
    <optional>
      <ref name="comment"/>
    </optional>
  </element>
</define>
<define name="title">
  <element name="title">
    <optional>
      <attribute name="href">
        <data type="token"/>
      </attribute>
    </optional>
    <data type="token"/>
  </element>
</define>
<define name="comment">
  <element name="comment">
    <mixed>
      <zeroOrMore>
        <choice>
          <ref name="a"/>
          <ref name="br"/>
          <ref name="strong"/>
        </choice>
      </zeroOrMore>
    </mixed>
  </element>
</define>
<define name="a">
  <element name="a">
    <attribute name="href">
      <data type="token"/>
    </attribute>
    <data type="token"/>
  </element>
</define>
<define name="br">
```

```
    <element name="br">
    </element>
  </define>
  <define name="strong">
    <element name="strong">
      <data type="token"/>
    </element>
  </define>
</grammar>
```

とりあえず、このシステムで用いるスキーマに適合する XML 文書のことを junknews 文書、junknews 文書のスキーマを junknews スキーマという用語の定義をしておきたい。
なんだか GNU[22] やら PHP[23] みたいに

再帰的な定義

になっているが、ご容赦願いたい。

さて、junknews スキーマから Relaxer オブジェクトを生成しておこう。

```
C:\src\relaxer\junknews>relaxer -verbose -dir:./jp/hsj/junknews -package:jp.hsj.junknews junknews2.rng
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/junknews/junknews2.rng
        artifact = ICommentMixed.java
        artifact = ICommentMixedChoice.java
        artifact = Junknews.java
        artifact = News.java
        artifact = Title.java
        artifact = Comment.java
        artifact = A.java
        artifact = Br.java
        artifact = Strong.java
        artifact = RString.java
        artifact = URelaxer.java
        artifact = RStack.java
        artifact = UJAXP.java
```

13 個のファイル 287,240 バイトが生成された。

[22] GNU is Not Unix
[23] PHP:Hypertext Preprocessor

演習(2) テキスト入力系

開発に着手した junknews 文書を扱うアプリであるが、junknews 文書をいちいち手で起こしているのが

実は面倒くさい

のである(^^;

となれば、テキストファイルにフィルタをかけて XML に起こしなおすという手続きを踏んで手抜きをしてみたいと思う。
ここには「出先からメールで所定の箇所に送れば、junknews を自動的に更新してくれるかなぁ」とかいう妄想めいた願望も含まれている(笑)
例によって次のようなプロセスが必要となる。

テキストファイルを一括で読み込む
↓
フォーマットに従って読み込み、News オブジェクトを順次作成していく
↓
全データが処理されたら、Junknews オブジェクトに追加。
↓
Junknews オブジェクトから XML ドキュメントを生成する。
↓
ウマー(ﾟдﾟ)

じゃ、早速作っていくとしよう。

テキストファイルの書式を検討する

とりあえずこんな書式から次の junknews 文書を起こしてみよう。

```
[ジャンル A]
ページのタイトル 1
http://hogehoge.com/hogehoge1.html

  コメント 1 行目。
  コメント 2 行目。

[ジャンル B]
ページのタイトル 2
```

```
http://hogehoge.com/hogehoge2.html

ページのタイトル 3

  コメント 1 行目。
```

書式は以下のようなものになる。

- スタートが[...]で始まる行があれば、それをジャンルとして扱う。[News]ならば News ジャンル
- http://で始まる行か空行までをタイトル文字列として扱う。
- http://で始まる行から空行まで一連の文字列をリンクの URL とする。ない場合は title 要素の href 属性を持たない。
- 空行後に半角スペースかタブ文字でインデントされた行から空行までをコメントとして扱う。ない場合は comment を空要素として扱う。
- 空行後は条件の先頭に戻り、EOF まで読み込み続ける。

実はこのようなテキスト処理において、Java というのは有利なプログラミング言語ではない。
J2SE 1.4 から正規表現ライブラリが標準で搭載されたりしてそれなりに進化はしてきているが、Perl や Ruby などを用いた方がよっぽどスマートなコーディングできる。
しかし、クロスランゲージの混乱を生むだけなので、今回は Java で処理することにする。
もっとも、最終的に Relaxer オブジェクトを用いることになるので、Java でなければならないわけであるが。

テキストを読み込み、Relaxer オブジェクトを操作する。

テキストから Relaxer オブジェクトを操作するクラスを TextImporter として、作ってみた。
紙面の都合上（というかテキスト読み込み処理があまりにあまりなので）、ソースは抜粋という形で掲載するが付録 FD には

恥ずかしげも無く全面掲載 [24]

しているので、ご笑覧頂きたい。

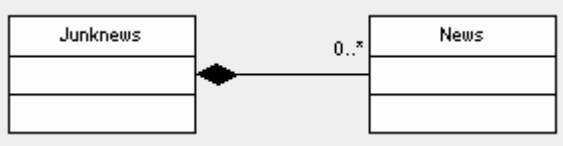
makeJunknews(ArrayList lineBuffer)メソッドがメインの処理部になる。
テキストデータを 1 行単位に String オブジェクトとして ArrayList に格納したものを引数にしている。List にしたのは ListIterator によるポインタ処理を行うためなので、別段オブジェクト配列を用いてもなんら問題はない。

[24] いやホントに急ぎでつくっているので（^^;;　申し訳ない限り。

では Relaxer オブジェクトと XML スキーマとのマッピングを中心にみていきたい。

① Junknews と News の関係

zeroOrMore という形で news 要素の出現条件を設けたところ、



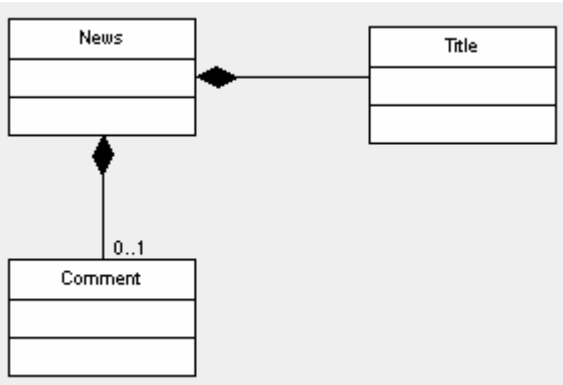
というクラス図の構成となり、Relaxer は News オブジェクトの配列格納を行うことになる。

```
Junknews jn = new Junknews();
News     n1 = new News();
News     n2 = new News();
/* n1, n2 になんらかの処理 */
jn.addNews(n1);
jn.addNews(n2);
```

以上のような流れである。

TextImporter では１つのニュース項目を処理するたびに addNews()している。

② News と Title や Comment の関係



Title については省略不可ということで初期化を行い、インスタンスを生成している。

対して Comment は Optional(省略可)な要素であるので、あらかじめ null を初期値としている。

③ Relaxer-Java マッピング戦略

本来は項目を設けて説明するべき重要なポイントであるのだが、たとえば title 要素のスキーマを取り上げると、

```
  <element name="title">
    <optional>
      <attribute name="href">
        <data type="token"/>
```

```
      </attribute>
    </optional>
    <data type="token"/>
  </element>
```

となり、href と要素の中身が token データ型とされている。
token データ型は java.lang.String にマッピングされる。
要素、属性関係なく、Title クラスの Href プロパティとしての Getter/Setter メソッド(Title#getHref(), Title#setHref(String href))が設けられる。他にも Java のプリミティブタイプ(int, float など)や Java のクラスライブラリにあるクラスで表現できるデータ形態である場合、それが用いられる(java.lang.String, java.sql.Time など)。

要素の中身は共通して content というプロパティとしてマッピングされる。
この場合、token なので java.lang.String を介した Getter/Setter が用意されている。
(Title#getContent(), Title#setContent(String content))

次に news 要素のスキーマを抜書きする。

```
  <element name="news">
    <attribute name="date">
      <data type="token"/>
    </attribute>
    <optional>
      <attribute name="genre">
        <data type="token"/>
      </attribute>
    </optional>
    <ref name="title"/>
    <optional>
      <ref name="comment"/>
    </optional>
  </element>
```

属性 date, genre については先の説明の通りである。ref[title]は先ほどの title 要素定義が参照されるが、title 自身のように自分に複数の値を持つような要素（コンプレックスレコード）の場合、Relaxer は別クラスの扱いにし、そのインスタンスをプロパティとするようになっている。そのため Getter/Setter については戻り値、引数ともに Title クラスとなっている。

そして、junknews 要素のスキーマは次のような形になっている。

```
  <define name="junknews">
    <element name="junknews">
    <attribute name="date">
        <data type="token"/>
    </attribute>
    <oneOrMore>
      <ref name="news"/>
    </oneOrMore>
  </element>
</define>
```

junknews 要素の子要素として一つ以上の news 要素を包含している。
date 属性については Junknews クラスのインスタンス変数としてマッピングされているが、oneOreMore という出現状態を news については java.util.List インターフェイスとしてマッピングされ、java.util.ArrayList のインスタンスが割り当てられている。
これがプリミティブなデータや JavaAPI ライブラリで表現出来る範囲であれば、配列としてマッピングされる。

④ Comment における混合内容の扱い
Java マッピングの続きであるが、ここまでは

まぁ、そんなもんだろなぁ

と想像できるレベルであったと思われるが、択一出現(choice)や混合内容(mixed)の実現にはデータバインディングツールによって大きく癖の異なる挙動をする。

```
    <element name="comment">
      <mixed>
        <zeroOrMore>
          <choice>
            <ref name="a"/>
            <ref name="br"/>
            <ref name="strong"/>
          </choice>
        </zeroOrMore>
      </mixed>
    </element>
```

comment 要素において、混合内容でかつ複数の a,br,strong 要素が任意に出現するという構造になっている。
この部分を構成している Relaxer オブジェクトは以下のファイルになる。

- ICommentMixed.java
- ICommentMixedChoice.java
- Comment.java
- A.java
- Br.java
- Strong.java
- RString.java

Comment クラスが母体となっているわけであるが、ここで setContent(String content)メソッドを用いて、要素の中身を設定すると毎回内容が上書きされてしまうため、混合内容として成立しない。

以下はコメントが続く限り、テキストデータと br 要素を追加していくソースの一部である。

```
          while (iterator.hasNext()) {
            String thisLine = (String) iterator.next();
            if (thisLine.equals("")) {
              Object objDummy = iterator.previous();
              break;
            }
            /* TEXT + br */
            Br br = new Br();
            comment.addContent(new RString(thisLine));
            comment.addContent(br);
          }
```

comment#addContent(ICommentMixed)というメソッドを用いて、混合内容を追加している。ICommentMixed は comment 要素内の混合内容を表現するための Java Interface になっており、混合内容に入りうる要素のクラスは ICommentMixed インターフェイスを実装（implements）する必要がある。

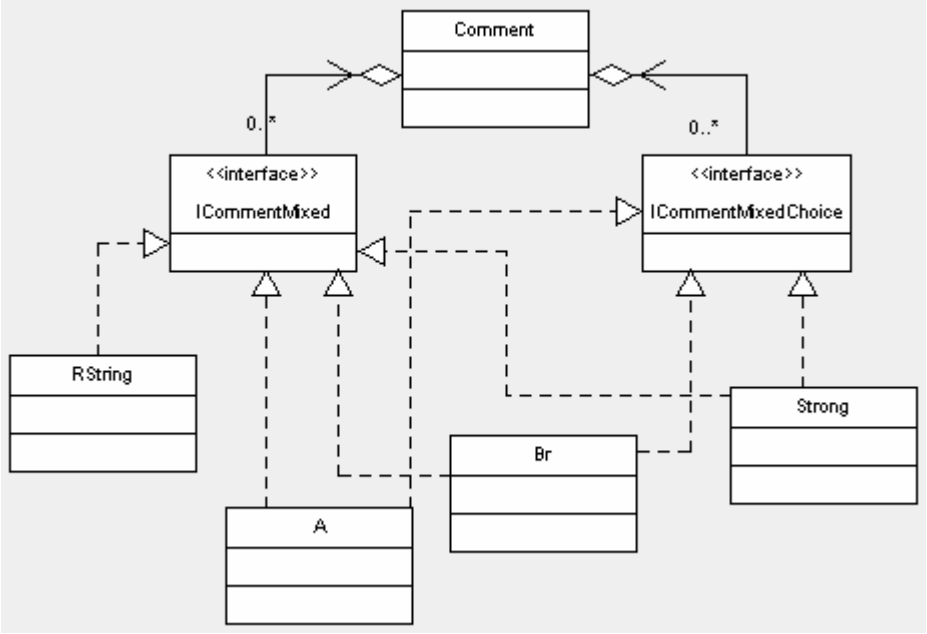
RString クラスという耳慣れないクラスを用いているが、これは混合内容モデルにおいてテキストを格納する補助クラスである。実際にソースを見てみると、

```
public class RString implements java.io.Serializable, ICommentMixed{
  ...略...
}
```

と、確かに ICommentMixed を implements している。一方、Br クラスは次のような状態だ。

```
public class Br implements java.io.Serializable, ICommentMixed, ICommentMixedChoice{
  ...略...
}
```

ICommentMixed だけでなく、ICommentMixedChoice というインターフェイスも実装している。これは Mixed とは直接関係無く、Choice による出現パターンを設定した場合に出てくるインターフェイスである。とは意味合いはさほど変わらず、Mixed の子要素としての Choice に該当する要素が implements しなければならないものである。



⑤ マッピング

現在の Relaxer オブジェクトには assertion の機能を付加していないので、特に要素の中身や属性値のマッピングに気をつけるべき点は多くないが、文書の DOM ツリーの末端からルートに向かって逆順に追いかけた方が無難であるといえるだろう（結局は設定・文法自体の問題）。

```
        title.setContent(titleString);
        title.setHref(urlString);
        news.setTitle(title);
        news.setGenre(genreString);
        news.setDate(this.getDateString());
        news.setComment(comment);
        junknews.addNews(news);
        junknews.setDate(this.getDateString());
```

ともあれこのクラスに Relaxer オブジェクトも含めて、ざくっとコンパイルする。

```
C:\src\relaxer\junknews>javac .\jp\hsj\junknews\*.java
```

junknews.20021230.txt

```
[DAIRY]
とりあえず仮想日記を展開しようかと思ったが、そんな文才も無いので適当に。
ともあれ恐らく死ぬほど寒いなか、わざわざ当サークルをご覧頂き感謝感激。

[XML]
OASIS Technical Committee: RELAX NG
http://www.oasis-open.org/committees/relax-ng/

ささやかなる実験場: HSJ.jp
http://hsj.jp/

[参考文献]
RELAX NG 私的解説メモ
http://www3.sppd.ne.jp/lena/relax-ng/

  結構参考にさせていただきました。

[RELAX NG]
RELAX NG 日本語ポータル
http://fortunecat.sourceforge.net/

  最近、更新されていないみたい。

[Linux]
ＵＮＩＸユーザーのための
コミックマーケット 63 情報ページ
http://www.lss.eternity.ne.jp/ibento/02_12_28/index.php

  うちも何でか知りませんが、載っています。
  UNIX ユーザ枠だったのね…、全部 Windows でやってるんだが(^^;
```

```
C:\src\relaxer\junknews>java jp.hsj.junknews.TextImporter "2002-12-30" junknews.20021230.txt
  junknews.20021230.xml
```

この結果が junknews.20021230.xml に出力される。

ただし、Relaxer オブジェクトの makeTextDocument メソッドでは空要素が開始・終了タグに分離されたり、改行がなくなったりした正規化された状態[25]で出されるため、以下にインデントや改行などで整形して結果を示す。

junknews.20021230.xml（整形済み）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<junknews date="2002-12-18">
  <news date="2002-12-18" genre="DAIRY">
    <title href="">
      とりあえず仮想日記を展開しようかと思ったが、そんな文才も無いので適当に。ともあれ恐らく死ぬほど
  寒いなか、わざわざ当サークルをご覧頂き感謝感激。
    </title>
    <comment>
    </comment>
  </news>
  <news date="2002-12-18" genre="XML">
    <title>
      OASIS Technical Committee: RELAX NG
    </title>
    <comment>
    </comment>
  </news>
  <news date="2002-12-18" genre="">
    <title>
      ささやかなる実験場: HSJ.jp
    </title>
    <comment>
    </comment>
  </news>
  <news date="2002-12-18" genre="参考文献">
    <title>
      RELAX NG 私的解説メモ
    </title>
    <comment>
      結構参考にさせていただきました。<br></br>
    </comment>
  </news>
  <news date="2002-12-18" genre="RELAX NG">
```

[25] CanonicalXML（http://www.w3.org/TR/xml-c14n）に準拠した形式として出力される。

```
    <title>
      RELAX NG 日本語ポータル</title>
    <comment>
      最近、更新されていないみたい。<br></br>
    </comment>
  </news>
  <news date="2002-12-18" genre="Linux">
    <title>
      ＵＮＩＸユーザーのためのコミックマーケット 63 情報ページ
    </title>
    <comment>
      うちも何でか知りませんが、載っています。<br></br>
      UNIX ユーザ枠だったのね…、全部 Windows でやってるんだが(^^;<br></br>
    </comment>
  </news>
</junknews>
```

演習(3) XML ドキュメントから HTML を起こす

前節でテキストデータから junknews 文書を起こすことに成功したわけであるが、どうせなら

HTML の出力結果が欲しい

と思うのは贅沢な願いではないだろう。
というわけでここでは XSLT を用いて、junknews 文書を HTML に変換するというアプローチで攻めてみたい。
ちなみにここでもキチンと Relaxer は活躍してくれるのである。

XSLT とはなんぞや。

何事も無く「XSLT を用いる」とかいったが一体なんだろうか。
XSLT(Extensible Stylesheet Language Transformation)は XML 文書の構造を変換して、他の XML 文書やテキストデータ（たとえば CSV）を生成する仕組みである。
XML 文書に XSLT スタイルシートと呼ばれる XML 文書を組み合わせ、XSLT プロセッサが処理を行う。

ためしに次のような XML 文書と XSLT スタイルシートを書いてみる。

test.xml

```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet href="test.xsl" type="text/xsl" ?>
<hello>Hello, world.</hello>
```

test.xsl　（test.xml と同じディレクトリに保存する）

```
<?xml version='1.0' encoding="Shift_JIS"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output indent="yes" method="html"/>
  <xsl:template match="/">
    <html><head><title>test.xml + test.xsl のサンプル</title></head>
  <body>
      <xsl:apply-templates/>
    </body></html>
  </xsl:template>
  <xsl:template match="hello">
  <h1><xsl:value-of select="."/></h1>
  <p><xsl:value-of select="."/></p>
```

```
  </xsl:template>
</xsl:stylesheet>
```

そして Internet Explorer など XSLT プロセッサを搭載したブラウザなどで test.xml を開くと、単なる XML 文書と異なる表示をするようになっている。
XSLT の動作結果をつかむために MSXSL というツールを使うと変換結果を出力できる。

MSXSL で変換結果を出力した実行例

```
<html><head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-16">
<title>test.xml + test.xsl のサンプル</title></head>
<body>
<h1>Hello, world.</h1>
<p>Hello, world.</p>
</body></html>
```

このような動作をするのが XSLT というものである。

余談ながら、XSLT は XML 文書の構造を変換して、他の XML 文書（以外も可能だが）に組み替えること機能を有するため、Relaxer オブジェクトと XML 文書は相互に変換可能であるという事実を踏まえると、Relaxer オブジェクト A に対して XSLT 変換を行い、Relaxer オブジェクト B に組み替えることが可能だということになる。積極的に使える場面というのはあまり想像できないが、柔軟度が高いことには間違いない。

Relaxer XSLT 連携

では、Relaxer に搭載されている XSLT 連携機能を使い、junknews スキーマから XSLT スタイルシートを生成してみる。

```
C:\src\relaxer\junknews>relaxer -verbose -xslt junknews2.rng
Copyright(c) 2000-2002 ASAMI,Tomoharu. All rights reserved.
Relaxer Version 0.17 (20021211) by asami@relaxer.org
Source file     : file:/C:/src/relaxer/junknews/junknews2.rng
        artifact = junknews2.xsl
```

こうして出来上がったのが、junknews2.xsl である。

junknews2.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:output indent="yes" method="xml"/>
<xsl:template match="strong">
  <strong>
    <xsl:apply-templates/>
  </strong>
</xsl:template>
<xsl:template match="comment">
  <comment>
    <xsl:apply-templates/>
  </comment>
</xsl:template>
<xsl:template match="news[title]">
  <news>
    <xsl:attribute name="date">
      <xsl:value-of select="@date"/>
    </xsl:attribute>
    <xsl:attribute name="genre">
      <xsl:value-of select="@genre"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </news>
</xsl:template>
<xsl:template match="junknews[news]">
  <junknews>
    <xsl:attribute name="date">
      <xsl:value-of select="@date"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </junknews>
</xsl:template>
<xsl:template match="br">
  <br>
    <xsl:apply-templates/>
  </br>
</xsl:template>
<xsl:template match="title">
  <title>
    <xsl:attribute name="href">
      <xsl:value-of select="@href"/>
```

```
      </xsl:attribute>
      <xsl:apply-templates/>
    </title>
  </xsl:template>
  <xsl:template match="a">
    <a>
      <xsl:attribute name="href">
        <xsl:value-of select="@href"/>
      </xsl:attribute>
      <xsl:apply-templates/>
    </a>
  </xsl:template>
</xsl:stylesheet>
```

XSLT(および XPath)の読み方については本稿では割愛させていただくが、junknews 文書をこの XSLT スタイルシートで処理しても、単純に写像するだけである。

junknews2.xsl の試し方

```
<?xml version="1.0"?>
<?xml-stylesheet href="junknews2.xsl" type="text/xsl" ?>
<junknews date="2002-12-18">
  ...略...
</junknews>
```

とりあえずこのままでは役に立たないので、HTML を吐き出せるように手を入れることにする。

junknews2.xsl

```
<?xml version="1.0" encoding="shift_jis" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="junknews">
  <xsl:element name="h3">
    <xsl:value-of select="substring(@date, 1, 4)"/>年<xsl:value-of select="substring(@date, 6, 2)"/>
  月<xsl:value-of select="substring(@date, 9, 2)"/>日
  </xsl:element>
  <xsl:apply-templates select="news">
```

```
  <xsl:sort select="@date"/>
  <xsl:sort select="@genre"/>
</xsl:apply-templates>
</xsl:template>
<xsl:template match="news">
<!-- ジャンルの最初でタイトル表示して UL を開きます -->
  <xsl:if test="../news[@genre=current()/@genre][position()=1]=.">
  <xsl:element name="h4">
       <xsl:value-of select="@genre"/>
  </xsl:element>
  <xsl:text disable-output-escaping="yes">&lt;div class="news_genre"&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;ul class="news"&gt;</xsl:text>
  </xsl:if>
<li>
  <xsl:apply-templates/>
</li>
<!-- ジャンルの最後で UL を閉じます -->
  <xsl:if test="../news[@genre=current()/@genre][position()=last()]=.">
  <xsl:text disable-output-escaping="yes">&lt;/ul&gt;</xsl:text>
  <xsl:text disable-output-escaping="yes">&lt;/div&gt;</xsl:text>
  </xsl:if>
</xsl:template>

<xsl:template match="title">
  <xsl:if test="@href!=''">
  <a target="_blank">
       <xsl:attribute name="href">
         <xsl:value-of select="@href"/>
       </xsl:attribute>
       <xsl:value-of select="."/>
  </a>
</xsl:if>
  <xsl:if test="@href=''">
  <xsl:value-of select="."/>
  </xsl:if>
<xsl:element name="br">
</xsl:element>
</xsl:template>
<xsl:template match="comment" >
```

```
  <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="br">
  <xsl:element name="br">
  </xsl:element>
  </xsl:template>
  <xsl:template match="strong">
    <strong>
      <xsl:apply-templates/>
    </strong>
  </xsl:template>
  <xsl:template match="a">
    <a>
      <xsl:attribute name="href">
        <xsl:value-of select="@href"/>
      </xsl:attribute>
      <xsl:apply-templates/>
    </a>
  </xsl:template>
</xsl:stylesheet>
```

これを先ほどの方法で処理命令を適当なサンプル junknews 文書に突っ込んでみると結構きれいな結果が出てくるはずである。

HTMLTranspoter を実装する

junknews 文書に対して XSLT スタイルシートを適用し、HTML に変換することに成功したが、処理命令でスタイルシートの参照を毎回追加するのは無意味であり、自動化したいと思うのが

人情

ってもんである。

というわけで、TextImporter での処理された junknews オブジェクトにある makeDocument メソッドを活用し、最小限の努力でスタイルシート処理をかける HTMLTranspoter を組んでみた。

```
package jp.hsj.junknews;

import java.io.*;
import java.util.*;
import java.text.*;
```

```
import javax.xml.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class HTMLTranspoter {
  public static void main(String[] args) throws Exception{
    Junknews     junknews = new Junknews();
    TextImporter ti       = new TextImporter();
    Document     doc      = null;
    Transformer  trans    = null;

    ti.setDateString(args[0]);
    ti.setSourceTextFile(args[1]);
    trans    = getTransformer(args[2]);
    junknews = ti.importFromTextFile();
    doc      = junknews.makeDocument();
    trans.transform(new DOMSource(doc), new StreamResult(System.out));
  }

  private static Transformer getTransformer(String xslFilePath) throws TransformerException{
    Source             xslSource = new StreamSource(new File(xslFilePath));
    TransformerFactory tf        = TransformerFactory.newInstance();
    Transformer        trans     = tf.newTransformer(xslSource);
    return trans;
  }
}
```

あまりの手抜き加減に悶絶しそうだが、以下のコマンドを実行する。

```
C:\src\relaxer\junknews>java jp.hsj.junknews.HTMLTranspoter "2002-12-30" junknews.20021230.txt
  junknews2.1.xsl > junknews.20021230.html
```

とすると、あら不思議、junknews.20021230.html というファイルが出来上がるのでした。

junknews.20021230.html

```
<?xml version="1.0" encoding="UTF-8"?>
<h3>2002 年 12 月 18 日
```

```
  </h3>
<h4/><div class="news_genre"><ul class="news">
<li>
<a target="_blank" href="http://hsj.jp/">ささやかなる実験場: HSJ.jp</a>
<br/>
</li></ul></div>
<h4>DAIRY</h4><div class="news_genre"><ul class="news">
<li>とりあえず仮想日記を展開しようかと思ったが、そんな文才も無いので適当に。ともあれ恐らく死ぬほど寒
  いなか、わざわざ当サークルをご覧頂き感謝感激。<br/>
</li></ul></div>
<h4>Linux</h4><div class="news_genre"><ul class="news">
<li>
<a target="_blank" href="http://www.lss.eternity.ne.jp/ibento/02_12_28/index.php">ＵＮＩＸユーザーの
  ためのコミックマーケット 63 情報ページ</a>
<br/>　　うちも何でか知りませんが、載っています。<br/> UNIX ユーザ枠だったのね…、全部 Windows でやっ
  てるんだが(^^;<br/>
</li></ul></div>
<h4>RELAX NG</h4><div class="news_genre"><ul class="news">
<li>
<a target="_blank" href="http://fortunecat.sourceforge.net/">RELAX NG 日本語ポータル</a>
<br/>　　最近、更新されていないみたい。<br/>
</li></ul></div>
<h4>XML</h4><div class="news_genre"><ul class="news">
<li>
<a target="_blank" href="http://www.oasis-open.org/committees/relax-ng/">OASIS Technical Committee:
  RELAX NG</a>
<br/>
</li></ul></div>
<h4>参考文献</h4><div class="news_genre"><ul class="news">
<li>
<a target="_blank" href="http://www3.sppd.ne.jp/lena/relax-ng/">RELAX NG 私的解説メモ</a>
<br/>　　結構参考にさせていただきました。<br/>
</li></ul></div>
```

いや、なんつーかこれで満足ですって感じ。

Appendix.A)
Relaxer コマンドラインオプション一覧

残念ながら現状の Relaxer の機能を網羅したリファレンスマニュアルは存在せず、Ver.1.00 の登場を待つより他にない。予定では Ver.0.17+リファレンスマニュアル＝Ver.1.00 となり、2003 年 1 月にリリースとのこと。

そのため、コマンドラインオプションの存在も"relaxer -help"を実行した際に出てくるものを追うしかない。以下に一覧としてまとめている[26] が、そのすべてを用いることは、まずないだろう。

[help] **ヘルプ関連。**	
-version	バージョンを表示する。
-help, -?	コマンドラインヘルプが表示される。 かなり大雑把なオプション解説のみ。
[generator] **生成対象物の指定。ソースとして XML 文書、RELAX・RELAX NG スキーマの三つである。**	
-java	RELAX/RELAX NG スキーマから Relaxer オブジェクト（Java クラス）を生成する。
-jdbc	Relaxer オブジェクトに JDBC 接続の機能を追加して Java クラスを生成する。
-cdl	Relaxer CDL に基づき、コンポーネントとなる Java クラスを生成する。
-rng	XML 文書のインスタンス・スキーマから RELAX NG スキーマを生成する。
-rxm	XML 文書のインスタンス・スキーマから RELAX スキーマを生成する。
-dtd	XML 文書のインスタンス・スキーマから DTD 定義を生成する。
-xsd	XML 文書のインスタンス・スキーマから W3C XML Schema スキーマを生成する。
-html	XPath な名前を付けられた HTML フォームを生成する。

[26] 一部調べ切れてないのがあります。申し訳ないです。

-xslt	XML 文書のインスタンス・スキーマから XSLT スタイルシートを生成する。
-editor	XML エディタ向けの設定データを生成する。
-meta	Relaxer 内部処理形式(Relaxer Meta Data)を XML 文書に変換する。
-setup	RELAX の検証に用いられる DTD,RELAX 文書を生成する。
[common] 生成処理などに用いる共通オプション。	
-verbose	処理中に詳細情報を表示する。
-properties:[propertyFileURI]	使用するプロパティファイルを指定する。 デフォルトは Relaxer.properties。
-dir:targetDirectory	生成ファイルの出力先となるディレクトリを指定する。 デフォルトはカレントディレクトリ。
-grammarVerify:[relax|dtd|none]	RELAX 文法の検証方法を指定する。 relax は RELAX スキーマ、dtd は DTD、none は検証をしない。デフォルトは relax。
-xmlBase:baseURI	スキーマ内から参照されるファイルの基点となるディレクトリを指定する。 デフォルトはカレントディレクトリ。
-dataConfig:fileName	データマッピングの設定
-dataProfile:fileName	データマッピングのプロファイル
[java] -java、-jdbc 時に対象となるオプション	
-useJAXP	XML パーサに JAXP を利用する。デフォルトは true。 J2SE 1.4 や J2SE 1.3 に同梱されている。 false に指定した場合、XML パーサの種類に依存しない Relaxer オブジェクトが生成される。
-useJAXPValidation	JAXP 上で検証を行うかどうか指定する。 デフォルトは true。
-useJAXPNamespaceAware	JAXP 上で名前空間の機能を有効にする。
-useJAXPDTD	JAXP 上で DTD を実体として使用する。
-useXMLPI	XML 文書内の処理命令の参照を可能とする。
-useXMLNSURI	名前空間 URI の取り扱いを可能とする。
-xmlElement	クラス生成に用いた DOM element を使用可能とする。
-verify:[relax|dtd|none]	Relaxer オブジェクトを XML 文書から生成する際に XML 文書の妥当性検査をする。 指定した場合、実行時に ISO RELAX API を実装したベリファイアが必要となる。デフォルトは none。

-objectVerify	Relaxer オブジェクトに実行時自らの内容を検証する機能を備える。
-assertion	Relaxer オブジェクトに実行時自らの内容が妥当でなくなった際に例外を発行するようにする。
-robust:[default|stable|rigid]	specifies robustness of a object
-sax	-saxInput と-saxOutput を同時に有効にする
-saxInput	SAX による Relaxer オブジェクト生成を有効にする
-saxOutput	Relaxer オブジェクトに SAX イベント機能を付加する
-text	テキスト出力を有効にする
-factory	Factory メカニズムの生成
-composite	composite メカニズムの生成
-visitor	visitor メカニズムの生成
-visitor.style:[light | plain | heavy]	visitor メカニズムの動作形態を指定する
-interpreter	interpreter メカニズムの生成
-context	context メカニズムの生成
-idmap	XML の属性型 ID,IDREF,IDREFS を用いた参照機能を Relaxer オブジェクトに付与する。
-pathlist	パスリスト形式の入力を可能にする
-package:packageName	生成される Relaxer オブジェクトにパッケージ名を割り当てる。デフォルトはなし。
-nameStyle:[java|c]	名前のスタイルを指定する。デフォルトは java。
-classPrefix:prefix	生成される Relaxer オブジェクトのクラス名にプレフィックスをつける。
-classPrefixes:prefixes	prefixes of generated classes
-java.dataConfig:fileName	Java マッピングのデータマッピングでのコンフィグレーション
-java.dataProfile:fileName	Java マッピングのデータマッピングでのプロファイル
[JDBC] **-jdbc の際に使用されるオプション**	
-jdbc.ee	データベースアクセスに J2EE 仕様を使用する
-jdbc.mapping:[direct|index]	データマッピングのマッピング戦略を指定する。
-jdbc.dataConfig:fileName	JDBC 上でのデータマッピングコンフィグレーション
-jdbc.dataProfile:fileName	JDBC 上でのデータマッピングのプロファイル
-jdbc.quote.id	SQL 文の識別子の区切り記号。デフォルトは"。
-jdbc.quote.where	SQL 文の WHERE 句の区切り記号
-jdbc.quote.idInWhere	SQL 文の WHERE 句内の識別子の区切り記号
-jdbc.documentColumnName	XML 文書全体をストアするカラム名を指定する。
-jdbc.mapping.strict	厳格なマッピングを行うかどうか指定する。

[CDL] -cdl の際に用いられるオプション	
-cdl.rmi	RMI コンポーネントを生成する。
-cdl.iiop	RMI over IIOP コンポーネントを生成する。
-cdl.ejb	EJB コンポーネントを生成する。
-cdl.ejb.local	EJB ローカルコンポーネントを生成する。
-cdl.ejb.gateway	EJB ゲートウェイを生成する。
-cdl.ejb.gateway.uri	EJB コンポーネントの URI を指定する。
-cdl.jaxm	JAXM コンポーネントを生成する。
-cdl.gateway	Generic ゲートウェイを生成する。
-cdl.filter	flter names to filter requests and responses
-cdl.dialog	Relaxer ダイアログマネージャ用設定ファイルを生成する。
-cdl.package	生成されるコンポーネントのパッケージ名を指定する。
-cdl.sp.package	生成されるサービスプロバイダのパッケージ名を指定する。
-cdl.context	コンポーネントでコンテクストを使用する
[DTD] -dtd の際に用いられるオプション	
-tagPrefix	タグ名に付けるプレフィックスを指定する。 名前空間対応の DTD を作成するためのオプションか。
[XSD] -xsd の際に用いられるオプション	
-xsd.policy	specifies a policy for XML schema model
[XSLT] -xslt の際に用いられるオプション	
-xslt.template	-xslt オプションで XSLT スタイルシートを作る際にテンプレートとする XSLT スタイルシートを指定する。

Appendix.B)
Relaxer に関するライセンス

個人利用であろうと商用利用であろうと、Relaxer で生成した各種ファイル、および Relaxer 付属のクラスライブラリの扱いを理解しておかないと、気軽に用いることはできないだろう。
Relaxer Ver.0.17 より以下のライセンスとなっている。

Relaxer コマンド	Relaxer.jar	GPL
利用、配布、改変の自由とソースの入手が保障される		
改変した場合、改変した箇所を示し、GPL で再配布する義務が発生		
組み込む場合、自分のソフトウェアも GPL にする義務が発生。		
Relaxer クラスライブラリ	**RelaxerOrg.jar**	**LGPL**
利用、配布、改変の自由とソースの入手が保障される		
改変した場合、改変した箇所を明確に示し LGPL か GPL で再配布する義務が発生。		
組み込む場合、 自分のソフトウェアは独自のライセンスで配布可能。		
Relaxer マテリアル	**Java ソースなど**	**BSD ライセンス**
再配布する際に著作権表示を行なうことのみを条件とする		
組み込み後のコードに公開義務はない		
可能であればリンクアイコンによる表示をお願いします、とのこと。		

単に Relaxer を利用して、Relaxer マテリアルを組み込む分には出来る限りリンクアイコンの表示をするということになる。
できてもできなくても

浅海さんに感謝しながら使いませう

ということになるだろう。

Appendix.C)
参考文献

行き当たりばったりの原稿だったので、すべてをフォローしきれていないと思いますが、少なくとも以下の書籍・サイト・ツールには大変お世話になりました。
また、Relaxer についての議論が行われている「RELAX 開発者日本語メーリングリスト」のログも随所で参考にさせていただきました。
この場を借りて感謝申し上げます。

[書籍] Relaxer Java/XML による Web 開発（浅海 智晴）
　http://www.amazon.co.jp/exec/obidos/ASIN/4894715279
[書籍] XML world（第 2 弾）IDG ムックシリーズ
　http://www.amazon.co.jp/exec/obidos/ASIN/4872801598
　■XML パーフェクト・ ガイド
　Part2）XML 活用のためのガイダンス XML で何ができるのか、そのためには何が必要なのか（川口耕介）
　Part3）詳説！XML 文法の基礎　　XML 文書と DTD の記述方法を理解する（村上隆一）
　■詳解！RELAX NG 入門（川口耕介）
　■アプリケーション開発における XML の活用（浅海智晴）
[雑誌] XML Press Vol.1～Vol.6（技術評論社）
　■Vol.5）特集 1. XML と J2EE による Web エンタープライズ開発 （浅海智晴）
　■Vol.6）特集 3. 実践！Relaxer 開発（浅海智晴）
[書籍] XML 技術者認定試験 XML マスターハンドブック ベーシック編（小沢 仁 / XML 技術者育成推進委員会）
　http://www.amazon.co.jp/exec/obidos/ASIN/4897974208
[RELAX NG] OASIS Technical Committee: RELAX NG
　http://www.oasis-open.org/committees/relax-ng/
[RELAX NG] RELAX NG Tutorial
　http://www.oasis-open.org/committees/relax-ng/tutorial-20011203.html
[RELAX NG] RELAX NG 日本語ポータル
　http://fortunecat.sourceforge.net/
[RELAX NG] RELAX NG 私的解説メモ
　http://www3.sppd.ne.jp/lena/relax-ng/
[RELAX NG] XML Cafe(News_RELAX NG)
　http://www.fxis.co.jp/xmlcafe/news/relax.html
[XML] @IT: XML eXpert eXchange
　http://www.atmarkit.co.jp/fxml/

[XML] XML 用語事典
http://www.atmarkit.co.jp/fxml/dictionary/indexpage/xmlindex.html
[XML] XML Schema Part 2: Datatypes
http://www.w3.org/TR/xmlschema-2/
[XSLT] XSLT スタイルシート書き方講座
http://www.atmarkit.co.jp/fxml/tanpatsu/10xslt/xslt01.html
[XSLT] サンプルで覚える XSLT プログラミング
http://www.atmarkit.co.jp/fxml/tanpatsu/xslt/xslt00.html
[TOOL] Relaxer
http://relaxer.org/
[TOOL] Jing - A RELAX NG validator in Java
http://www.thaiopensource.com/relaxng/jing.html
[TOOL] Sun Multi-Schema XML Validator
http://wwws.sun.com/software/xml/developers/multischema/
[TOOL] Command Line Transformations Using msxsl.exe
http://msdn.microsoft.com/library/en-us/dnxslgen/html/msxsl.asp
[TOOL] Eclipse
http://www.eclipse.org/
[TOOL] Poseidon for UML
http://www.gentleware.com/
[TOOL] xyzzy
http://www.jsdlab.co.jp/~kamei/
[TOOL] xyzzy 研究室〜ライブラリ - xml-mode
http://sugi.pobox.ne.jp/xyzzy/library.html
[License] License Tips
http://www.sip.eee.yamaguchi-u.ac.jp/kou/license.html

Appendix.D)
J2SE 1.3 以前の場合の JAXP の利用

JAXP が標準パッケージに入っていない J2SE 1.3 以前の JDK を使っている向きが Relaxer オブジェクトを使おうとすると、JAXP（もしくは JAXP 対応の XML パーサ）のインストールをする必要が出てくる。

現状で Sun から提供されている XML 関連の API パッケージのディストリビューションとしては、以下の二つがある。

- Java XML Pack Downloads & Specifications
  http://java.sun.com/xml/downloads/javaxmlpack.html
- Java Web Services Developer Pack(JWSDP)
  http://java.sun.com/webservices/webservicespack.html

しかし、こんなものを使わなくても実は手元に JAXP に対応したパーサがあったりするのである。それは Relaxer のアーカイブに同梱されている **xerces.jar** である。つまりこれに CLASSPATH を通してしまえばいいわけである。

まず環境変数の設定をこんな感じでしておいて頂きたい。

```
set JAXP_PATH=C:\usr\local\lib\relaxer\xerces.jar
```

そして、javac コマンドや java コマンドの実行の際に、次のような形で CLASSPATH を指定してやる形になる

```
javac -classpath:.;%CP_JAXP% *.java
java  -classpath:.;%CP_JAXP% *
```

これで最新版の JDK でなくても Relaxer の甘い汁が吸えるのだ。

びば、りらくさー。

Appendix.E)
付録 FD について

収録しているコンテンツは以下の通りです。

- 本書の PDF ファイル(SCN_C63.PDF)
  (<FD>:¥PDF フォルダ)
- 本書でサンプルとして利用した Java ソース、XML 文書など
  (<FD>:¥SOURCE フォルダ)
- （おまけ）エディタ xyzzy の xml-mode で RELAX NG キーワードを対応させるパッチ
  (<FD>:¥XYZZY フォルダ)

利用に際しての諸注意

- FD 内のコンテンツの内、各種 PDF ファイルは著作権法によって保護されています。著作者の承諾なしに第三者への譲渡／販売／複写の実施、改変することはできません。
- 収録されている各ファイルの動作は本書目次ページ末にて掲示している環境での動作しか確認しておりません。ご利用の環境によっては正常に動作しない場合があります。あらかじめご了承ください。
- 本 FD のコンテンツを使用して発生した如何なる障害などに関して、直接／間接的に関わらず、頒布会、電脳螺旋網公司、筆者は一切の責任を負いません。

編集後記 - 後の末莉[27]

12 月 20 日の金曜日の晩。最寄り駅のモスバーガーで VAIO SR を使って、原稿を書いている。
つまり――なんというかまだ編集中どころか、執筆中である。修正しないといけないところが大幅に放置されている。
C62 のあとから書き溜めていた原稿であったが、あまりのアウトラインのダメさ加減に構成を変更したのが今週の頭であり、そこから１週間も経っていない。毎日毎夜原稿を打ち続けていたわけであるが、やはりギリギリにならないとできない自分の体質が恨めしい。

そもそも Relaxer というものは今は亡き XML Press を創刊から購読していたので、存在は知っていたが、使い出したのは浅見さんの Relaxer 本が発売された後。それでもなお RELAX というスキーマ言語に馴染めず騙しだまし使っていたため、積極的に使おうという気もないまま、某ゲームに大いにハマったりしていたわけだ（コミケ後はどっぷり**絆箱**の世界に…）。
夏前に多少動きがあった。一つは RELAX NG の ISO/IEC 標準化の動き。もう一つはそれに連動した Relaxer の RELAX NG スキーマへの対応である。とはいえ、RELAX NG スキーマ対応の Relaxer 0.17 がリリースされる時期というのが見えないまま、RELAX NG＋Relaxer による原稿を書くのは結構冒険をしたものだと、今では思う。
ともあれ、それなりの形に（無理やりにではあるが）持っていけただろうと思っている[28]。

楽しいプログラミングこそホビープログラミングの聖域である。Relaxer を使えば鬱陶しいパーサ API を意識せずに快適なプログラミングに打ち込むことができる。
これだけは間違いなく保証できるので、皆さんも本書を片手に簡単な実践をし Relaxer の魅力に触れて頂けたらと願うばかりである。

さて、次回の電脳螺旋網公司の決算報告書であるが、いまのところ未定であるが Relaxer を使いながら、最近ハマっている Eclipse+SWT なプログラムについて扱ってみたい。
某グループに参加しているので**「声に出して読みたいＷＥＢ」**とかでもいいかとは思うが……
あと、本書のブラッシュアップ＋オフセット印刷を望んでいるのだが、Relaxer 1.0 のリリース時にリファレンスマニュアルが添付されるようなので、筆者のような門前の小僧がどうこういうよりもいいような気がするので、こちらも未定。
ともあれ、予想以上に長くなった本書でありますが、最後までお付き合いくださった方々に最大限の感謝を申し上げます。

2002 年 12 月 20 日(金)(執筆中)
筆者　拝

[27] 誤植ではありません(笑)　でも、どちらかというと準や青葉のほうが……げふげふ。
[28] それでも時間が足りなかったために開発編の１章分が丸ごと吹っ飛んでしまいましたが（^^;;

```
<?xml version=”1.0” encoding=”Shift_JIS”?>
<奥付 xmlns="http://hsj.jp/xmlns/copyright/1.0">
  <題名>
        電脳螺旋網公司 2002 年下期決算報告書
        RELAX NG + Relaxer で Java/XML を 2.56 倍使うための本
  </題名>
  <著作者>どなどな</著作者>
  <発行>電脳螺旋網公司[Spiral Cable Network]</発行>
  <印刷所>KINKO's 新大阪店</印刷所>
  <頒布所>
        <頒布会 名前="C63">西地区【る】39a</頒布会>
  </頒布所>
  <発行日 版数="1">2002 年 12 月 30 日(月)</発行日>
  <連絡先>
        <URL>http://www.spiralcable.net/</URL>
        <e-mail>donadona@spiralcable.net</e-mail>
  </連絡先>
</奥付>
```

RELAX NG + Relaxer で Java / XML を 2.56 倍使うための本