# プログラムトレース情報のXML木への変換
# Conversion of Program Trace Data to XML Tree

安藤誠† 櫻井孝平† 古宮誠一‡
Makoto Ando† Kouhei Sakurai† Seiichi Komiya‡
† 芝浦工業大学　工学部
‡ 芝浦工業大学大学院　工学研究科
† College of Engineering, Shibaura Institute of Technology.
‡ Graduate School of Engineering, Graduate School of Shibaura Institute of Technology.

**要旨**

デバッグとはプログラムコードに含まれる欠陥を探し、修正する工程である。一般的にはプリント文の挿入や、デバッガと呼ばれるツールを使い、欠陥を発見する。本研究はトレース情報に基づくデバッガに着目する。トレース情報に基づくデバッガはプログラムの実行を遡ることができ、欠陥から生成される故障をたどることで欠陥を発見する。一般的なトレース情報に基づくデバッガはプログラムの実行をトレース情報として記録する。よって元のプログラムの規模が大きいとそれに比例してトレース情報も巨大なものとなり扱いにくくなる。またデバッグに必要な情報として表現方法が一様に決まっているためユーザーが対話的に変更できない。これらの問題を、XML 技術を用いて解決する。

## 1. はじめに

プログラムを実行すると、しばしば期待していない出力や挙動が故障として見られる。故障は間違った記述（欠陥）によって引き起こされる。欠陥を探す作業はデバッグと呼ばれ、開発者は一般にプリント文の挿入やデバッガと呼ばれるツールを使い行う。以下はその説明である。

- プリント文の挿入

プログラムの動作確認のためコードの中にプリント文を挿入して、ユーザーの欲しい情報が欲しい表現で得られる手法である。特別なツールを必要とせずどのような場面でも使えるがプログラムコードを書き換える必要があるため新たなバグを生み出す危険性などがある。

- ブレークポイントデバッガ

ブレークポイントとして指定した箇所からプログラムコードを一行ずつ実行して欠陥を探していくデバッガ。停止した行の時点の詳細な情報を取得することができるが、プログラムの実行を遡ってデバッグすることができない。

- トレース情報に基づくデバッガ[1]

プログラムのすべての実行履歴をトレース情報として記録するデバッガ。任意の時点から遡りながら欠陥を探すことができる。プログラムコードを手動で書き換えることなく、ユーザーが欲しい情報を取得できる。

本研究はトレース情報に基づくデバッガに注目する。一般的なトレース情報に基づくデバッガはプログラムの実行をトレース情報として記録する。よって元のプログラムの規模が大きいとそれに比例してトレース情報も巨大なものとなり扱いにくくなる。またデバッグに必要な情報として表現方法が一様に決まっているためユーザーが対話的に変更できない。トレース情報を XML[3]へ変換することで、これらの問題をそれぞれ XML データベースと XSL を用いて解決する。

## 2. トレース情報を使ったデバッグの現状

本節では本研究で使用するトレースに基づくデバッガを簡単なテキスト整形プログラム Tidy を利用して説明する。

Tidy は文字列間の２つ以上の連続した空白を１つにまとめるという機能がある。ただし、行頭にある連続した空白はインデントと見なし、まとめない。ここでは図１のように、期待とは逆にインデントがまとめられ、その他の連続した空白はまとめられないという欠陥を考える。本稿ではこの Tidy を例にと

り説明をしていく。

| 入力 | 期待する値 | 出力 |
|---|---|---|
| ___hello___world | ___hello_ world | _hello___world |

**図 1**　Tidy-空白をまとめる機能（_は空白を意味する）

### 2.1. トレースデバッガ

トレースを基にしたデバッガは、ブレークポイントデバッガとは異なり指定した箇所から逆戻りしながら欠陥を探すことができる。また、プリント文の挿入と異なりプログラムコードを書き換えることなく、デバッグに必要な情報を記録する。

本研究ではすでに開発したトレース情報に基づいたデバッガ（以降トレースデバッガと呼ぶ）の改良を行う。本トレースデバッガは Java プログラムの実行のイベントを最初から最後まで可能な限りトレース情報として取得し、専用の形式でファイルに保存してメソッド呼び出しの文脈による木構造で表示する。ノードは図２のような、代入文、分岐、メソッド呼び出し、return のいずれかで構成される。ノードの持つ値は基本型と文字列、固有の番号が付与されたオブジェクト、ローカル変数の名前を保持する。表示は専用の GUI で行われる。

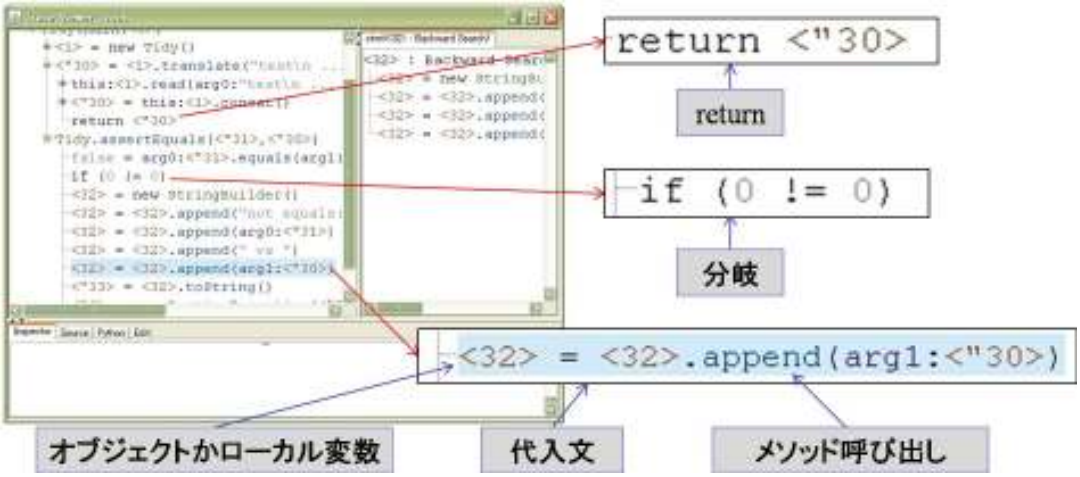



**図 2**　Tidy をトレースデバッガの GUI で表示した例

### 2.2. トレース情報のサイズと表現の問題点

トレースデバッガは、実行の最初から最後まで取りうる限りのトレース情報を取得するためトレース情報の量が膨大になる。デバッガはファイルをメモリに展開するため、特に大きなプログラムや繰り返しが用いられているプログラムの情報は巨大になり、一般的に性能が低下し扱いにくくなる[2]。

またトレース情報の表現方法が一様に決まってしまっているためユーザーの望むように変更ができない。多くのデータが一画面に表示されるため、ユーザーが必要な情報を見つけにくくなってしまいデバッグが困難になることがある。トレース情報として必要な情報を持っていてもユーザーが理解しやすい表現とは限らない。よって、ユーザーがトレース情報から必要な情報だけを理解可能な表現で対話的に得られるようにすべきである。

## 3. 提案手法

前述の２つの問題点を解決するために、本研究では XML を用いる。トレース情報は木構造になっているので、同じく木構造として表される XML に変換することができる。変換した XML を XML データベースに適用することでトレース情報量が巨大になり扱いにくくなるという問題点を、XSL を適用することで表現が一様に決められていてユーザーの望むように変更ができないという問題点の解決を図る。

### 3.1. トレース情報の XML 木への変換

本研究ではトレース情報を XML へ変換する規則を定義し、実装を行った。

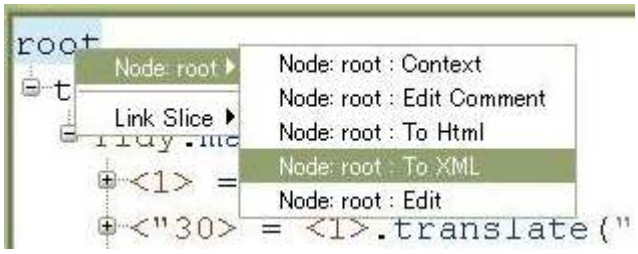
変換規則：あるトレース情報の要素 t に対する変換規則 ToXML(t)を図３のように定義する。

| | | |
|---|---|---|
| *ToXML*(*s*{*children*}) | =<node *srcLoc*(*s*)> *ToXML(s) ToXML*(*children*)</node> | (トレース木) |
| *ToXML*(*l*=*r*) | =<set> *ToXML*(*l*) *ToXML*(*r*) </set> | (代入文) |
| *ToXML*(*expr*) | =<call> *ToXML*(*expr*) </call> | (メソッド呼び出し文) |
| *ToXML*(*if expr*), | =<if><bin op＝“expr”> *ToXML(expr)* </bin></if> | (if 文) |
| *ToXML*(*obj.mthd*(*args*)) | =<m name=”*mthd*”> *ToXML(obj) ToXML(args)*</m> | (メソッド呼び出し式) |
| *ToXML*(*v*) | =<value value＝”*v*”/> | (値) |
| *ToXML*(*local* : *v*) | =<value name＝“*local*” value＝“*v*”/> | (ローカル変数付き値) |

※*srcLoc(s)*は *s* のソース行番号とソースファイル名の属性を意味する

**図 3** 変換規則の一部

この規則を基に XML へ変換するコードをトレースデバッガのコードへ書き加えた。
そして図４左のようにトレースデバッガの GUI のノードを右クリックして To XML を実行することでトレース情報を XML ファイルへ加工することができる。図４の右側は<3>=<3>.append(“t”)という StringBuilder オブジェクトへのメソッド呼び出しを変換した XML である。Tidy のトレース情報 607 行を変換するのにかかった時間は 156msec で、作成された XML ファイルは 104,976 バイトであった。




XML ファイルへ書き出す

```
・・・
<node line="13" src="Tidy">
  <set>
    <value value="$3"/>
    <m name="append"
     static="false" Type="(char)">
      <value value="$3"/>
      <value value="t"/>
    </m>
  </set>
</node>
・・・
```

**図 4**　To XML

### 3.2. トレース情報の XML データベースへの格納

XML データベースとは、XML の木構造をそのまま格納し、利用できるデータベースである。XML データベースを用いることで、扱うデータが大きくなっても高いパフォーマンスを保ったままの検索が可能となる。

トレース情報をファイルとして保存するのではなくデータベースに直接格納することで効率良く検索ができる。本研究では Xindice[6]というオープンソースの XML データベースを使用する。Tidy のトレース情報 607 行の Xindice への格納時間は 563msec であった。

Xindice は索引をタグや属性に対して作成することでそれらのタグや属性に関する検索を高速化できる。今後は実際に検索が速くなるのか実験を行う予定である。また、現在のトレースデバッガと、ある決められた順序で繰り返しデバッグしたときの時間の平均を比較する予定である。

### 3.3. XSL を使った XML の変換

XSL は XML 文書をもとにレイアウトないし変換を行うためのスタイルシート技術である。本研究では XSL をフロントエンドに利用し、XML で表されたトレース情報からユーザーが望む情報を探し出してくるために XSL を用いる。トレースデバッガでは表示に独自の GUI を用いているが、XSL を用いてトレース情報を HTML に変換することでウェブブラウザでの表示を可能にする。

XSL は XSLT[4]、XPath[5]、XSL-FO から構成されているが、本研究ではそのうち XSL と XPath のみを使用するため、それらの説明をする。

- XSL Transformations(XSLT)

XML 文書を構造の異なる XML 文書に変換するための XML に準拠した言語である。XML の一部を取得したり、HTML へ変換することができる。

- XML Path Language(XPath)

XML 文書の特定の部分（要素、属性、テキストなど）を指定する言語である。XSLT で処理対象の XML 文書の特定部分を指定するために使われている。

XPath を使用し XML データベースから欲しい情報だけを取得して、XSLT により新たな XML 文書を作成、または HTML へ変換することができる。

図４の XML ファイルに図５左の XSL を適用することにより図５右のような画面をウェブブラウザで表示することができる。この XSL は、Tidy の出力に関わる append メソッドの呼び出しを XML ファイルから XPath で検索し、XSLT でどのように表示するかが書かれている。

現状では Xindice へ格納せずに XML ファイルをウェブブラウザ内臓の XSL 系処理を使用してウェブブラウザで表示しているが、Xindice からは XPath で XML データを取得できるので、将来的に Xindice と XSL を組み合わせてウェブブラウザで直接表示できるようにし、デバッグを支援する予定である


```
<xsl:template match="/">
  <p><xsl:apply-templates select="//set"/></p>
</xsl:template>
<xsl:template match="m[@name='append']/value[2]">
  <p>append ("<xsl:value-of select="./@value"/>")
    //<xsl:value-of select="../../../@src"/>
    line:<xsl:value-of select="../../../@line"/></p>
</xsl:template>
```

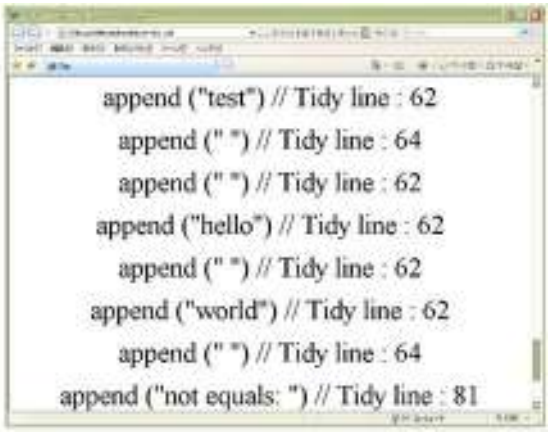



**図 5**　XSL、ウェブブラウザでの表示例

## 4. おわりに

本研究では古宮研究室で研究されているトレースデバッガの問題点を、XML 技術を用いて解決する手法を示した。トレース結果を XML に変換し、１つ目の問題である巨大なデータを扱いにくいという点は XML データベースで解決する。2 つ目の問題である、トレースデバッガでは専用の GUI を使用する。そのためトレースの表現方法が一様に決まってしまっていて柔軟ではなく、ユーザーの望むように変更ができない、という点は XSL によって解決する。

現在、変換規則の決定、変換コードの記述、XML ファイルを Xindice へ格納する作業まで終了した。今後は Xindice を使用して索引の検証実験とトレースデバッガとの速度比較実験をする。また、Xindice から取得したデータにどのように XSL を適用させていくかを検討する予定である。

## 参考文献

[1] B. Lewis, "Debugging Backwards in Time”, AADEBUG, 2003
[2] G. Pother, E. Tanter, and J. Piguer, “Scalable omniscient debugging”, OOSPLA’07, pages 535-552, New York, NY , USA , 2007. ACM
[3] “XML”, http://www.w3.org/TR/2006/REC-xml11-20060816/
[4] “XSLT”, http://www.w3.org/TR/xslt20/
[5] “XPath”, http://www.w3.org/TR/xpath20/
[6] “Xindice”, http://xml.apache.org/xindice/1.1/index.html