クラスファイルの構造解析(1)

2002年6月16日

2003年6月8日 改訂

海谷 治彦

目次

- アセンブラとマシン語のギャップ
- クラスファイルの構造の概要
- バイトコード解析補助ツール
- コンスタントプールの構造とその解析

Javaとアセンブラの違い

```
public class Add{
    int add(int a, int b){
        return a+b;
    }
}
```

```
.class  public Add
.super  java/lang/Object

.method public <init>()V
    .limit stack 1
    .limit locals 1
    aload_0
    invokespecial java/lang/Object/<init>()V
    return
.end method

.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

- 結構長くなる.
- メソッド等の対応はそのまま.
- 暗黙定義のコンストラクタなどが補完されている.

アセンブラとクラスファイル

```
.class  public Add
.super  java/lang/Object

.method public <init>()V
    .limit stack 1
    .limit locals 1
    aload_0
    invokespecial java/lang/Object/<init>()V
    return
.end method

.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

な，何が何やら？
(236 Byteあります．)

```
CA FE BA BE 00 03 00 2D ; .......-
00 0F 0A 00 03 00 0C 07 ; ........
00 0D 07 00 0E 01 00 06 ; ........
3C 69 6E 69 74 3E 01 00 ; <init>..
03 28 29 56 01 00 04 43 ; .()V...C
6F 64 65 01 00 0F 4C 69 ; ode...Li
6E 65 4E 75 6D 62 65 72 ; neNumber
54 61 62 6C 65 01 00 03 ; Table...
61 64 64 01 00 05 28 49 ; add...(I
49 29 49 01 00 0A 53 6F ; I)I...So
75 72 63 65 46 69 6C 65 ; urceFile
01 00 08 41 64 64 2E 6A ; ...Add.j
61 76 61 0C 00 04 00 05 ; ava.....
01 00 03 41 64 64 01 00 ; ...Add..
10 6A 61 76 61 2F 6C 61 ; .java/la
6E 67 2F 4F 62 6A 65 63 ; ng/Objec
74 00 21 00 02 00 03 00 ; t.!.....
00 00 00 00 02 00 01 00 ; ........
04 00 05 00 01 00 06 00 ; ........
00 00 1D 00 01 00 01 00 ; ........
00 00 05 2A B7 00 01 B1 ; ...*....
00 00 00 01 00 07 00 00 ; ........
00 06 00 01 00 00 00 03 ; ........
00 00 00 08 00 09 00 01 ; ........
00 06 00 00 00 1C 00 02 ; ........
00 03 00 00 00 04 1B 1C ; ........
60 AC 00 00 00 01 00 07 ; `.......
00 00 00 06 00 01 00 00 ; ........
00 05 00 01 00 0A 00 00 ; ........
00 02 00 0B             ; ....
```

クラスファイル内部の分類

定数データの表 (コンスタントプール) (50.4%)

```
CA FE BA BE 00 03 00 2D 00 0F 0A 00 03 00 0C 07 ; .......-........
00 0D 07 00 0E 01 00 06 3C 69 6E 69 74 3E 01 00 ; ........<init>..
03 28 29 56 01 00 04 43 6F 64 65 01 00 0F 4C 69 ; .()V...Code...Li
6E 65 4E 75 6D 62 65 72 54 61 62 6C 65 01 00 03 ; neNumberTable...
61 64 64 01 00 05 28 49 49 29 49 01 00 0A 53 6F ; add...(II)I...So
75 72 63 65 46 69 6C 65 01 00 08 41 64 64 2E 6A ; urceFile...Add.j
61 76 61 0C 00 04 00 05 01 00 03 41 64 64 01 00 ; ava........Add..
10 6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 ; .java/lang/Objec
74 00 21 00 02 00 03 00 00 00 00 00 02 00 01 00 ; t.!.............
04 00 05 00 01 00 06 00 00 00 1D 00 01 00 01 00 ; ................
00 00 05 2A B7 00 01 B1 00 00 00 01 00 07 00 00 ; ...*............
00 06 00 01 00 00 00 03 00 00 00 08 00 09 00 01 ; ................
00 06 00 00 00 1C 00 02 00 03 00 00 00 04 1B 1C ; ................
60 AC 00 00 00 01 00 07 00 00 00 06 00 01 00 00 ; `...............
00 05 00 01 00 0A 00 00 00 02 00 0B             ; ............
```

プログラム(アセンブラ命令)の部分は，たったこれだけ．(3.8%)

参考までですが表記法について

- 0, 1, 2, ... D, E, F で4bitを表現.
- 上記を2つ組み合わせて，1Byte を表現.
- よって，CA = 1100 1010 という1Byteデータ.

- いきなり，マイコンらしい泥臭い話に突入 (涙)

命令文はどうなってる？

```
.class  public Add
.super  java/lang/Object

.method public <init>()V
    .limit stack 1
    .limit locals 1
    aload_0
    invokespecial java/lang/Object/<init>()V
    return
.end method

.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

コイツの話が今日のメイン

```
2A       ; p.227 aload_0
B7 00 01 ; p.368 invoke ...
B1       ; p.452 return


1B ; p.355 iload_1
1C ; p.355 iload_2
60 ; p.318 iadd
AC ; p.379 ireturn
```

今日は命令全体の話は
あまりしません.

実用規模のクラスの場合

- Djava.class 8113バイト
- 内，63.6% (5159B)が定数表.
- 命令文は 20.6% (1672B)程度.
- 詳細は，HP上の ./djava/ 以下を参照．(都合，一部，学科内限定)

まとめ
アセンブラとバイトコードの対応

- 単純に対応付いて無い.
- 定数データ表(コンスタントプール)の割合が大きい. (実用プログラムでも.)
- 命令文の割合は大きくない.
  - トイ問題では誤差程度.
- フィールド, メソッドの境界はちゃんとある. (後述)

クラスファイルの構造概要

- 教科書 p.36 図1-13 である.
- 大雑把にいって,
  1. クラスファイル自体の情報
  2. コンスタントプールのリスト
  3. クラス自体の情報
  4. 実装しているインタフェースのリスト
  5. フィールドのリスト
  6. メソッドのリスト
  7. クラスの付加的情報のリスト(Innerクラスもこれに含まれる)

の7つのパートからなる.

簡単なクラスの構造例

- 詳細は ./runnable/ を参照.
  - インタフェース1, フィールド2, メソッド2 のクラス

```
public class MyRun implements Runnable{
private int a=0;
int b=2;
    public void run(){
        a++;
        b=incb();
    }
    private int incb(){  return b+1;}
}
```

クラスファイルの情報

- ほとんど定数のようなもん.
- コンパイラのバージョンによって異なる場合があり.

CA FE BA BE ; magic number
00 03 ; minor version
00 2D ; major version

コンスタントプール

- 今日のメイン・・・・詳細は別途

```
00 1B ; constant pool number= 26(d) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

0A 00 06 00 14 ;(00 01) Methodref => java/lang/Object.<init>:()V
09 00 05 00 15 ;(00 02) Fieldref => MyRun.a:I
09 00 05 00 16 ;(00 03) Fieldref => MyRun.b:I
0A 00 05 00 17 ;(00 04) Methodref => MyRun.incb:()I
07 00 18 ;(00 05) Class => MyRun
07 00 19 ;(00 06) Class => java/lang/Object
07 00 1A ;(00 07) Class => java/lang/Runnable
01 00 01 61 ;(00 08) Utf8 = "a"
01 00 01 49 ;(00 09) Utf8 = "I"
01 00 01 62 ;(00 0A) Utf8 = "b"
01 00 06 3C 69 6E 69 74 3E ;(00 0B) Utf8 = "<init>"
01 00 03 28 29 56 ;(00 0C) Utf8 = "()V"
01 00 04 43 6F 64 65 ;(00 0D) Utf8 = "Code"
01 00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65 ;(00 0E) Utf8 = "LineNumberTable"
01 00 03 72 75 6E ;(00 0F) Utf8 = "run"
01 00 04 69 6E 63 62 ;(00 10) Utf8 = "incb"
01 00 03 28 29 49 ;(00 11) Utf8 = "()I"
01 00 0A 53 6F 75 72 63 65 46 69 6C 65 ;(00 12) Utf8 = "SourceFile"
01 00 0A 4D 79 52 75 6E 2E 6A 61 76 61 ;(00 13) Utf8 = "MyRun.java"
0C 00 0B 00 0C ;(00 14) NameAndType => <init>:()V
0C 00 08 00 09 ;(00 15) NameAndType => a:I
0C 00 0A 00 09 ;(00 16) NameAndType => b:I
0C 00 10 00 11 ;(00 17) NameAndType => incb:()I
01 00 05 4D 79 52 75 6E ;(00 18) Utf8 = "MyRun"
01 00 10 6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 ;(00 19) Utf8 = "java/lang/Object"
01 00 12 6A 61 76 61 2F 6C 61 6E 67 2F 52 75 6E 6E 61 62 6C 65 ;(00 1A) Utf8 = "java/lang/Runnable"
```

クラスの情報

- クラス自体固有の情報

p.58 or p.195

```
00 21 ; access flag= PUBLIC + SUPER
00 05 ; this class= "MyRun"
00 06 ; super class= "java/lang/Object"
```

インタフェースのリスト

- 実装してるインタフェースの数を示し，それぞれをリストアップ

```
00 01 ; num. of implemented interface(s)= 1(d)
00 07 ; "java/lang/Runnable"
```

フィールドのリスト

- クラス内のフィールド数とそれぞれを列挙.

p.62 or p.209

```
00 02 ; num. of field(s)= 2(d) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;BEGIN field1(d) def.
00 02 ; access flag of attributes = PRIVATE
00 08 ; method name ="a"
00 09 ; method type ="I"
00 00 ; num. of attribute(s), 0(d)

;BEGIN field2(d) def.
00 00 ; access flag of attributes =
00 0A ; method name ="b"
00 09 ; method type ="I"
00 00 ; num. of attribute(s), 0(d)
```

メソッドのリスト

- 実装してるメソッド数と，それぞれのメソッドをリストアップ

```
00 03 ; num. of methods(s)= 3(d)

;; メソッド1, 2は複雑だから省略

                    p.73 or p.210

00 02 ; access flag of method = PRIVATE
00 10 ; method name ="incb"
00 11 ; method type ="()I"
00 01 ; num. of attribute(s), 1(d)
;;begin attribute 1(d)
00 0D ; attribute type= "Code"
00 00 00 1F ; attribute length= 31(d)
```

```
00 02 ; stacks limits
00 01 ; locals limits
00 00 00 07 ; length of codes = 7(d)
2A       ; aload_0
B4 00 03 ; getfield MyRun/b I
04       ; iconst_1
60       ; idd
AC       ; ireturn
00 00 ; num. of exception handler(s) = 0(d)
;;
00 01 ; num. of attr. of this code attr. = 1(d)
00 0E ; "LineNumberTable"
00 00 00 06 ; length of attr.=6(d)
00 01 ; 1(d) line number(s)
00 00 00 0D ; .line 13
```

クラス属性のリスト

- もとになったファイル名，Innerクラスなどの情報.

```
00 01 ; num. of class attribute(s)= 1(d) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;BEGIN class attribute1(d)
00 12 ; class's attr.1(d) name="SourceFile"
00 00 00 02 ; length of attr.=2(d)
00 13 ; filename="MyRun.java"
;END class attribute1(d)
```

クラスファイル解析の意義

- 演習では，皆さんに，かなりローレベルなツールを使って，解析を行ってもらいます．
- 世間には，便利な解析ツールもでまわってますが，実際にバイト列をいじくり回すのが目的なので，そこんところを理解してください．

bin2hex & hex2bin

- bin2hex: バイナリデータを16進テキストに変換する汎用ツール.
- hex2bin: その逆.
- 所謂, バイナリエディタの類. bit単位の修正を可能とする.
- 現状ではperlでできている. 残念.
- 教科書 p.38- みたいな結果を出すため作った.

djavaによる情報取得

- djavaは，逆アセンブラだが，バイトコード解析データも得られる.
- 典型例: djava –c –o DJava クラスファイルで，コンスタントプールをリストアップできる.
- 詳細は，djava –h でヘルプを見よ.

その他ツール

- javaclass API: クラスファイルを解析するAPI. 配布はしてるので自由に使って.
- jcf-dump: gcc version 2.95.2 以降に付属されたコンスタントプール表示ツール.
  - 現3年の標準OSには入っていないようだ.

コンスタントプール(CP)とは何か？

- 教科書 p.50～52 の説明がすばらしい.
- 要は定数(コンスタント)の置き場，リスト.
  - 12種類のCPエントリがある.
    - Utf8
    - Integer, Float, Long, Double
    - Class
    - String
    - Fieldref, Methdref, InterfaceMethodref
    - NameAndType

  詳細は，教科書 p.51表，p.199～207参照.

何故CPが必要か？

- メソッド名，フィールド名は度々出てくるので，番号付けして管理したほうがお得.
  - Javaは動的リンクなので，これら名前をコンパイル時に剥ぎ取れない.
- オンデマンドロード，ダイナミックリンクの実現．詳細は教科書 p.115～ 参照.
- 名前解決の効率化．(後述)

簡単な例 ./add/ より

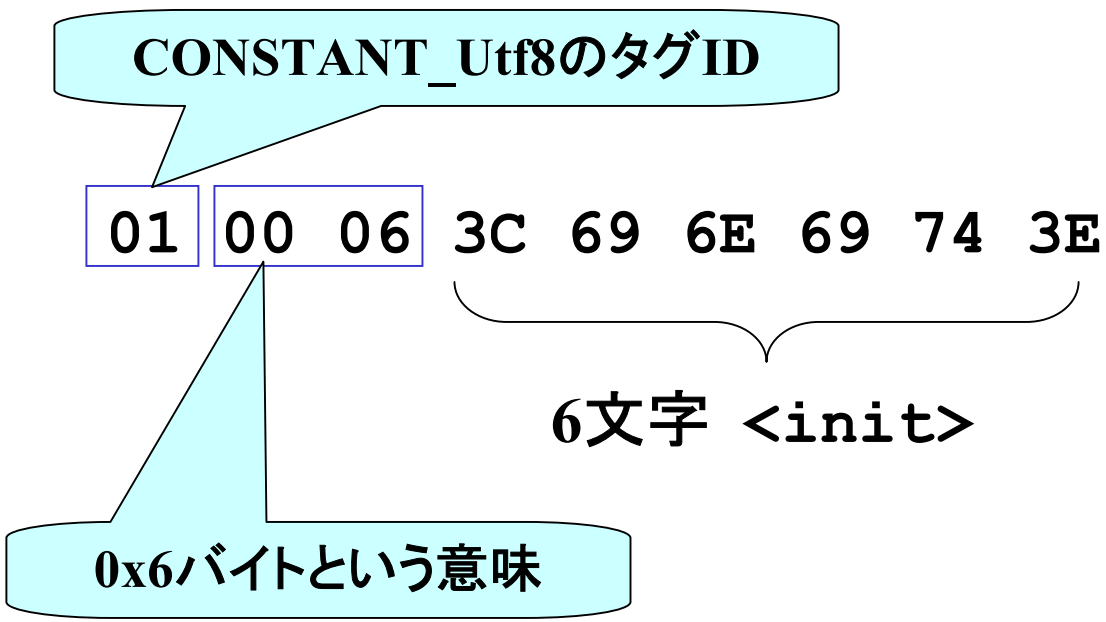
CPの個数-1？

14個
(E個)

```
00 0F ; constant pool number= 14(d)
0A 00 03 00 0C ;(00 01) Methodref => java/lang/Object.<init>:()V
07 00 0D ;(00 02) Class => Add
07 00 0E ;(00 03) Class => java/lang/Object
01 00 06 3C 69 6E 69 74 3E ;(00 04) Utf8 = "<init>"
01 00 03 28 29 56 ;(00 05) Utf8 = "()V"
01 00 04 43 6F 64 65 ;(00 06) Utf8 = "Code"
01 00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65
            ;(00 07) Utf8 = "LineNumberTable"
01 00 03 61 64 64 ;(00 08) Utf8 = "add"
01 00 05 28 49 49 29 49 ;(00 09) Utf8 = "(II)I"
01 00 0A 53 6F 75 72 63 65 46 69 6C 65 ;(00 0A) Utf8 = "SourceFile"
01 00 08 41 64 64 2E 6A 61 76 61 ;(00 0B) Utf8 = "Add.java"
0C 00 04 00 05 ;(00 0C) NameAndType => <init>:()V
01 00 03 41 64 64 ;(00 0D) Utf8 = "Add"
01 00 10 6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74
            ;(00 0E) Utf8 = "java/lang/Object"
```

CONSTANT_Utf8

- Utf8形式で文字列を保持，ASCII文字列は1バイト，日本語は3バイトになる.



Utf8

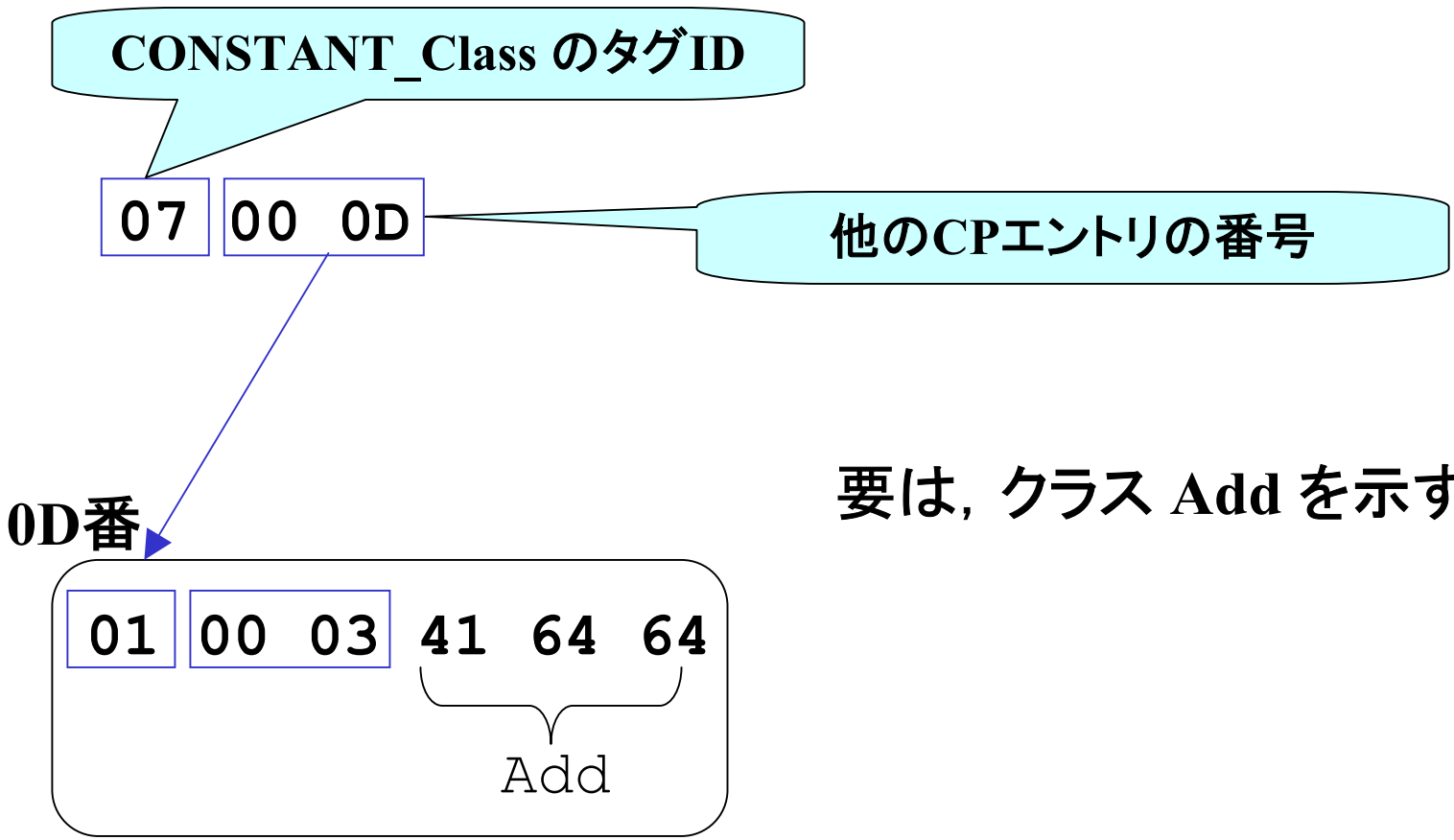
- ASCIIやEBCDIC等と同様，文字を計算機の中で表現する規約.
- There are seven character encoding schemes in Unicode: UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE and UTF-32LE.

  In http://www.unicode.org/

  glossary/index.html#character_encoding_form
- 要は文字をコード化する方法の一つと思ってください.
- 教科書p.51にJVMに関する解説がちょっとある.

CONSTANT_Class

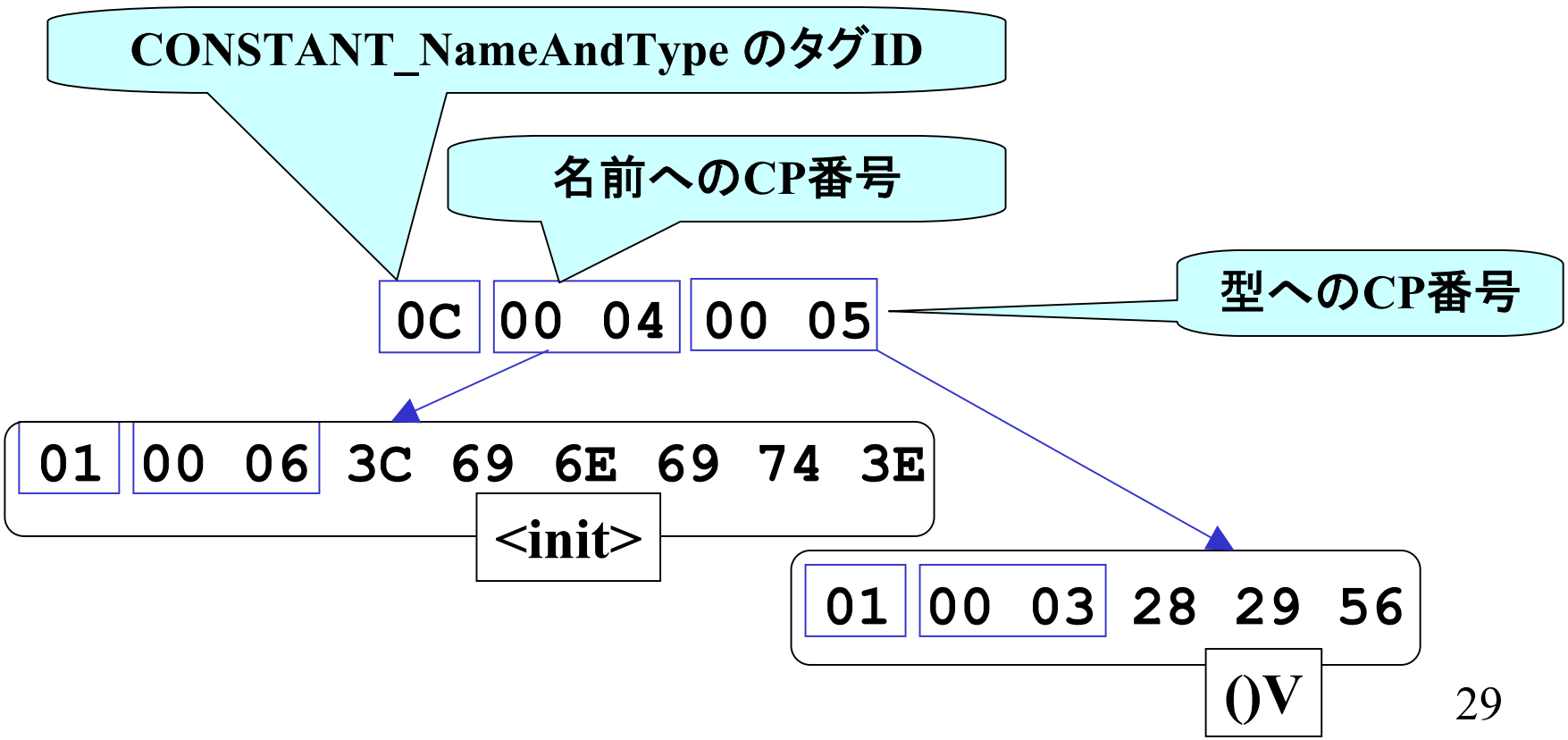
- クラス・インタフェースへの参照



要は，クラス Add を示す．

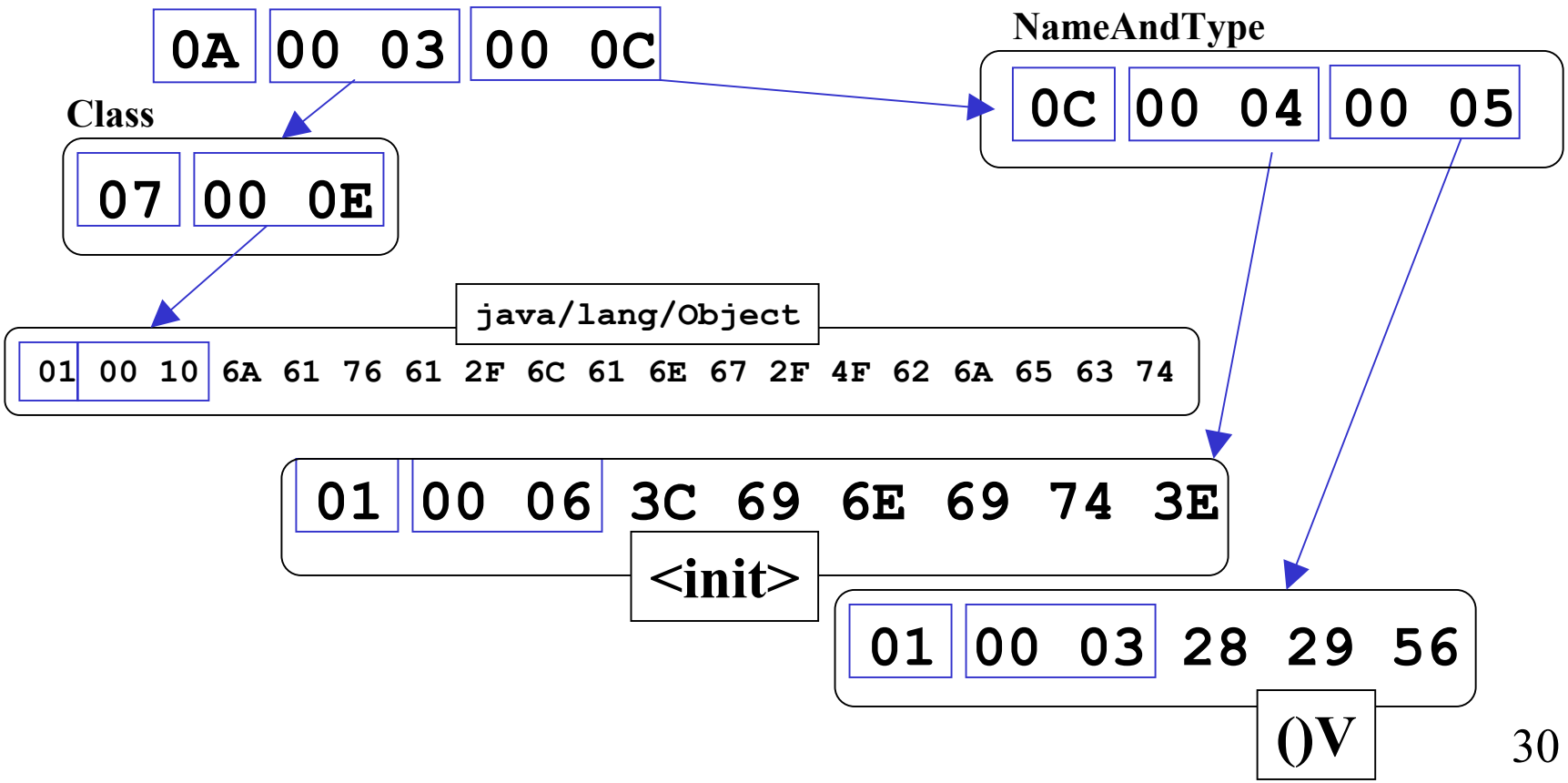
CONSTANT_NameAndType

- 名前と型(引数と返り値)の対情報(シグニチャ)を持つ.



CONSTANT_Methodref

- メソッドの属するクラスと，メソッドの名前・型の対の対を持つ．



コンスタントプール リゾリューション

- Constant Pool Resolution
- CP中の，クラス，メソッド，フィールドの参照を，JVM中の実際のメモリ空間のアドレスに置きかえること.
- JVMでのダイナミックリンクの実現手段
  - リンクについては講義第2回資料参照.
- 教科書 p.115~

今日はこの辺でお終い