(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4699580号
(P4699580)
(45)発行日　平成23年6月15日(2011.6.15)　(24)登録日　平成23年3月11日(2011.3.11)

(51)Int.Cl.
G06F 9/45 (2006.01)

F I
G06F 9/44 320C
G06F 9/44 322A

請求項の数 9　外国語出願　(全 19 頁)

(21)出願番号	特願平11-98306
(22)出願日	平成11年4月6日(1999.4.6)
(65)公開番号	特開2000-35893(P2000-35893A)
(43)公開日	平成12年2月2日(2000.2.2)
審査請求日	平成18年4月3日(2006.4.3)
(31)優先権主張番号	09/055947
(32)優先日	平成10年4月7日(1998.4.7)
(33)優先権主張国	米国(US)

前置審査

(73)特許権者　510303992
オラクル・アメリカ・インコーポレーテッド
アメリカ合衆国、１９８０８　デラウェア、シティー・オブ・ウィルミントン　スイート４００、センタービレ・ロード　２７１１
(74)代理人　100095669
弁理士　上野　登
(72)発明者　イェリン　フランク
アメリカ合衆国　９４０６１　カリフォルニア州、レッドウッド　シティ、ベレスフォード　アベニュー　５１０

最終頁に続く

(54)【発明の名称】データ処理システムの配列の静的初期化方法、データ処理方法、並びにデータ処理システム及びその制御手順をコンピュータに実行させるプログラムを記憶したコンピュータ読み取り可能な記

(57)【特許請求の範囲】
【請求項１】
コンパイラとプリローダと仮想マシンが格納されたメモリと該メモリに格納されたコンパイラとプリローダと仮想マシンを起動するプロセッサを備えたデータ処理装置によって配列の静的初期化を行う配列の静的初期化方法であって、
配列の静的初期化により当該配列を静的値とするバイトコードを保存する特別メソッド付きのクラスファイルを生成するために、前記プロセッサが前記コンパイラを起動して当該静的配列を保存するソースコードをコンパイルするコンパイル工程と、
前記プロセッサが前記コンパイラを起動することにより送信された前記バイトコードを保存する特別メソッド付きのクラスファイルを、前記プロセッサが前記プリローダを起動することにより送信された場所から受信する受信工程と、　10
前記プロセッサが前記仮想マシンを起動することにより前記特別メソッドが実行されたとした場合になされる前記配列の静的初期化を識別するために、前記プロセッサが前記仮想マシンを起動して前記バイトコードを実行することなく、前記プロセッサが前記メモリ上の前記プリローダを起動することにより、
前記クラスファイルから前記特別メソッドを検索して取得するとともに、前記特別メソッドに関連するデータ構造に対して疑似実行変数を割り当て、割り当てた疑似実行変数を前記特別メソッドのバイトコードに基づいて操作して前記特別メソッドのバイトコードを疑似実行する疑似実行工程と、
前記疑似実行の結果に基づいて前記プロセッサが前記仮想マシンを起動して前記配列の　20

静的初期化を実現するための命令であって前記バイトコードより短い命令を生成するとともに前記バイトコードと置き換えて前記クラスファイルに保存する保存工程と、
前記配列の静的初期化を行うために前記プロセッサが前記仮想マシンを起動することにより前記出力ファイルに保存された前記命令の表現を解釈する解釈工程とを備えたことを特徴とする配列の静的初期化方法。
【請求項２】
前記疑似実行工程は、
スタック割当工程と、
前記特別メソッドから、前記スタックを処理するために前記プロセッサが前記仮想マシンを起動して実行するバイトコードを読み出す読出工程と、
バイトコードの読み出しに基づいて前記割当スタック上でスタックを処理するスタック処理工程とを含むことを特徴とする請求項１に記載される配列の静的初期化方法。
【請求項３】
前記疑似実行工程は、
変数割当工程と、
前記特別メソッドから、当該特別メソッドのローカル変数を処理するために前記プロセッサが前記仮想マシンを起動して実行するバイトコードを読み出す読出工程と、
バイトコードの読み出しに基づいて前記割当変数上でローカル変数を処理するローカル変数処理工程とを含むことを特徴とする請求項１に記載される配列の静的初期化方法。
【請求項４】
前記疑似実行工程は、
前記特別メソッドのコンスタント・プールに対するポインタを取得する取得工程と、
前記特別メソッドから、前記コンスタント・プールを処理するために前記プロセッサが前記仮想マシンを起動して実行するバイトコードを読み出す読出工程と、
バイトコードの読み出しに基づいて前記コンスタント・プールを処理するコンスタント・プール処理工程とを含むことを特徴とする請求項１に記載される配列の静的初期化方法。
【請求項５】
記憶デバイスと、メモリとプロセッサを備えたデータ処理システムであって、
前記記憶デバイスは、配列の静的初期化を行う際に使用されるソースコードを含むプログラムと、バイトコードを含み前記配列の静的初期化を行う際に実行される特別メソッドを保持するクラスファイルを少なくとも１つ含む１以上のクラスファイルとを保持し、
前記メモリは、コンパイラとプリローダと仮想マシンとを保持し、
前記プロセッサは、前記メモリ上の前記コンパイラを起動して、前記プログラムをコンパイルして前記クラスファイルを生成し、
前記プロセッサは、前記メモリ上のプリローダを起動して、前記１以上のクラスファイルを統合し、
前記プロセッサが前記メモリ上の仮想マシンを起動して前記特別メソッドを実行することにより行う静的初期化の効果を決定するために、前記プロセッサが前記メモリ上の仮想マシンを起動して当該特別メソッドを実行することなく、前記プロセッサは、前記メモリ上のプリローダを起動して、前記クラスファイルから前記特別メソッドを検索して取得するとともに、前記特別メソッドに関連するデータ構造に対して疑似実行変数を割り当て、前記特別メソッドから前記バイトコードを読み出し、読み出した前記バイトコードに基づいて割り当てた疑似実行変数を操作して疑似実行するとともに、疑似実行の結果に基づいて当該静的初期化を行う命令であって前記バイトコードより短い命令を作成し、
さらに、前記プロセッサは、前記メモリ上の前記仮想マシンを起動して、その作成済命令を解釈して配列の静的初期化を行うことを特徴とするデータ処理システム。
【請求項６】
前記作成済み命令は、コンスタント・プールへのエントリを含むことを特徴とする請求項５に記載されるデータ処理システム。

【請求項７】
コンパイラとプリローダと仮想マシンが格納されたメモリ及び該メモリに格納されたコンパイラとプリローダと仮想マシンを起動するプロセッサを有するコンピュータが読み取り可能な記憶媒体であって、
配列の静的初期化に関するオペレーションを行うために前記プロセッサが仮想マシンを起動して実行するバイトコードを、前記プロセッサがコンパイラを起動して送信し、前記プロセッサがプリローダを起動して送信された場所から受信する手順と、
前記プロセッサが仮想マシンを起動して前記バイトコードを実行したとした場合の前記オペレーションを識別するために、前記プロセッサが前記仮想マシンを起動して当該バイトコードを実行することなく、前記プロセッサが前記メモリ上の前記プリローダを起動して、前記バイトコードに関連するデータ構造に対して疑似実行変数を割り当て、割り当てた疑似実行変数を前記バイトコードに基づいて操作して当該バイトコードを疑似実行するシミュレーション手順と、
前記プロセッサが前記メモリ上の前記プリローダを起動して、当該疑似実行の結果に基づいて、前記オペレーションを行うために仮想マシンを起動する前記プロセッサに対する命令であって、前記バイトコードより短い命令を作成する作成手順とからなる制御手順をコンピュータに実行させるプログラムが記憶されていることを特徴とするコンピュータが読み取り可能な記憶媒体。
【請求項８】
更に、前記制御手順は、前記オペレーションを行うために前記プロセッサ上で、その作成済命令を実行させる実行手順を含むことを特徴とする請求項７に記載されるコンピュータが読取り可能な記憶媒体。
【請求項９】
更に、前記制御手順は、前記オペレーションを行うために、前記プロセッサが前記仮想マシンを起動することによりその作成済命令を解釈する解釈手順を含むことを特徴とする請求項７に記載されるコンピュータが読み取り可能な記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、データ処理システムに関し、更に詳しくは、静的初期化方法及びそのシステムに関するものである。
【０００２】
【従来の技術】
Java™は、プラットフォーム独立のコードの生成・実行を行うためにプログラミング言語及びプログラミング環境の両者を記述するものである。このプラットフォーム独立のコードは、Java™仮想マシン上で実行される。このJava™仮想マシンは、プラットフォーム独立のコードを解釈する抽象コンピュータマシンである。Java™仮想マシンの詳細については、ティム・リンドホルム及びフランク・イェリン著:アジソンウェスレイ出版(1997年版)"The Java™ 仮想マシン仕様"に記載されているのでそちらを参照されたい。Java™仮想マシンは、Java™プログラミング言語その他のプログラミング言語を特定して認識するものではないが、その代わりにJava仮想マシンは、特定のファイルフォーマット、即ちクラスファイルフォーマットを認識するものである。クラスファイルは、プラットフォーム独立のコードを構成するJava仮想マシンの命令(又はバイトコード)を保持したものである。
【０００３】
Javaプログラムの実行に際し、開発者は、図１に示した工程で作業を進める。まず、開発者は、コンピュータプログラムのコンパイルを行う(ステップ102)。Javaプログラミング言語等の高級言語においては、開発者は、ソースコードを保持したコンピュータプログラムを開発し、Java™コンパイラを起動させ、そのコードのコンパイルを行っている。Javaコンパイラは、カリフォルニア州マウンテンビューのサンマイクロシステムズから入手可能なJava™ソフトウエア開発者キットの一部である。Javaコンパイラは、Java仮想マシン

上で実行するのに適したバイトコードを含む1又は複数のクラスファイルを出力する。各クラスファイルは、クラスやインタフェースのいずれか1つの型を保持している。クラスファイルフォーマットの詳細については、"The Java™ 仮想マシン仕様"の83-137頁に説明されている。これによれば、クラスファイルフォーマットは、強固なファイルフォーマットである。

【０００４】

【発明が解決しようとする課題】

しかしながら、このクラスファイルフォーマットは、仮想マシンに対して効率的に配列の静的初期化を行うように命令することができない。以下この欠点に関し詳細に説明する。プログラムをコンパイルした後、開発者は、ステップ102におけるクラスファイルの出力をプリローダを用いて.mクラスファイルと称される単一ファイルに統合する(ステップ104)。サンマイクロシステムズから入手可能なプリローダは、クラスファイルを結合させ、そのクラスファイルの実行を容易にする予備処理を行う。クラスファイルを統合した後、開発者は、.mクラスファイルの仮想マシンへのロードを行う(ステップ106)。このステップにおいては、Java仮想マシンは、メモリ中に.mクラスファイルを保存する。そして、Java仮想マシンは、そのバイトコードを読み出して、.mクラスファイルが保持するバイトコードを解釈した後、それらを処理し実行する。バイトコードの解釈が完了するまで、.mクラスファイルは、メモリ中に保存される。Java仮想マシンにより認識されるバイトコードの詳細については、"The Java™仮想マシン仕様"の151-338頁に説明されている。

【０００５】

［仕様］

上述したように、クラスファイルフォーマットは、仮想マシンに配列の静的初期化を行うように命令することができない。この問題を解決するために、Java™コンパイラは、静的配列の初期化メソッドを含む特別メソッドである<clinit>を生成してクラスの初期化を行う。静的配列の初期化の一例を表１のコードテーブル#1に示す。

【０００６】

【表１】

ｺｰﾄﾞﾃｰﾌﾞﾙ#1

```
static int setup[ ]={1,2,3,4};
```

【０００７】

この例においては、配列"setup"は、静的初期化がなされた4つの整数－1，2，3，4を保持する。この所定の静的初期化がなされると、Java™コンパイラは、静的初期化を行う<特別>メソッドを生成する。この静的初期化は、仮コードで関数的に記述され、その一例を表２のコードテーブル#2に示す。

【０００８】

【表２】

ｺｰﾄﾞﾃｰﾌﾞﾙ#2

```
temp=new int[4];
temp[0]=1;
temp[1]=2;
temp[2]=3;
temp[3]=4;
this.setup=temp;
```

【０００９】
上述のコードテーブルが示すように、<特別>メソッドを関数的に記述する場合、いくつかのステートメントを必要とする。しかしながら、バイトコードによって行われる<特別>メソッドの実際の処理は、より多くのステートメントを必要とする。これらのバイトコードは、スタックを処理することにより、要求された静的初期化を行うものである。スタックは、Javaプログラミング環境の多くのメソッドにより使用されるメモリの一部である。上述したスタック初期化の一例に使用される<特別>メソッドにより行われる各ステップを表３のコードテーブル#3に示す。

【００１０】

【表３】

コードテーブル#3

```
Method void<clinit>()
0  iconst_4       //スタックに整数値４をプッシュする
1  newarray int   //新たな整数配列を作成し、スタックに
                    プッシュする
3  dup            //スタックの先頭を複製する
4  iconst_0       //スタックに整数値０をプッシュする
5  iconst_1       //スタックに整数値１をプッシュする
6  iastore        //配列のインデックス０に１を記憶させる
7  dup            //スタックの先頭を複製する
8  iconst_1       //スタックに整数値１をプッシュする
9  iconst_2       //スタックに整数値２をプッシュする
10 iastore        //配列のインデックス１に２を記憶させる
11 dup            //スタックの先頭を複製する
12 iconst_2       //スタックに整数値２をプッシュする
13 iconst_3       //スタックに整数値３をプッシュする
14 iastore        //配列のインデックス２に３を記憶させる
15 dup            //スタックの先頭を複製する
16 iconst_3       //スタックに整数値３をプッシュする
17 iconst_4       //スタックに整数値４をプッシュする
18 iastore        //配列のインデックス３に４を記憶させる
19 putstatic#3<Field foobar.setup[I>
                  //スタック上で新たな配列に従って
                    配列setupを修正する
22 return
```

【００１１】
<特別>メソッドの使用は、Java仮想マシンに静的配列の初期化を行うように命令する方法を提供するが、その配列の初期化に必要なコード量は、そのサイズの配列を何度も行うと相当量のメモリを必要とすることになる。従って、静的初期化の手法を改善する必要がある。

【００１２】
本発明の解決しようとする課題は、配列の静的初期化に必要なメモリ使用量を節約することができるデータ処理システムの配列の静的初期化方法、データ処理方法並びにデータ処理システム及びその制御手順をコンピュータに実行させるプログラムを記憶したコンピュータ読み取り可能な記憶媒体を提供することにある。

【００１３】

【課題を解決するための手段】

上記課題を解決するために、本発明に係る請求項１に記載のデータ処理装置が行う配列

の静的初期化方法は、配列の静的初期化により当該配列を静的値とするバイトコードを保存する特別メソッド付きのクラスファイルを生成するために、当該静的配列を保存するソースコードをコンパイルするコンパイル工程と、コンパイラから送信された前記クラスファイルをプリローダにより受信する受信工程と、前記プロセッサによって特別メソッドが実行されたとした場合になされる前記配列の静的初期化を識別するために、前記プロセッサ上で当該コードを実行することなく、メモリ上で前記プリローダにより前記特別メソッドのバイトコードを疑似実行する疑似実行工程と、前記疑似実行の結果に基づいて前記配列の静的初期化を実現するための命令を出力ファイルに保存する保存工程と、前記配列の静的初期化を行うために仮想マシンにより前記出力ファイルに保存された表現を解釈する解釈工程とを備えたことを要旨とするものである。

【００１４】
この場合に、前記保存工程は、請求項２に記載されるように、コンスタント・プール・エントリをコンスタント・プールに保存する保存工程を含むようにするとよい。

【００１５】
また、前記擬似実行工程は、請求項３に記載されるように、スタック割当工程と、前記特別メソッドから前記スタックを処理するバイトコードを読み出す読出工程と、バイトコードの読み出しに基づいて前記割当スタック上でスタックを処理するスタック処理工程とを含むようにするとよい。或いは、前記擬似実行工程は、請求項４に記載されるように、変数割当工程と、前記特別メソッドから当該特別メソッドのローカル変数を処理するバイトコードを読み出す読出工程と、バイトコードの読み出しに基づいて前記割当変数上でローカル変数を処理するローカル変数処理工程とを含むようにしてもよい。更に、これに代えて、前記擬似実行工程は、請求項５に記載されるように、前記特別メソッドのコンスタント・プールへの参照を取得する取得工程と、前記特別メソッドから前記コンスタント・プールを処理するバイトコードを読み出す読出工程と、バイトコードの読み出しに基づいて前記コンスタント・プールを処理するコンスタント・プール処理工程とを含めてもよい。

【００１６】
次に、本発明に係る請求項６に記載のデータ処理装置は、オペレーションを行うために前記プロセッサ上で実行されるコードをコンパイラから受信する受信工程と、前記コードが前記プロセッサにより実行されたとした場合の当該コードのオペレーションを識別するために、前記プロセッサ上で当該コードを実行することなくメモリ上で当該コードを前記プリローダにより疑似実行する疑似実行工程と、前記コードのオペレーションを行うために前記プロセッサに対する命令を作成する作成工程とを備えたことを要旨とするものである。

【００１７】
上記構成を有する請求項６に記載のデータ処理装置によれば、まず、オペレーションを行うプロセッサ上で実行されるべきコードが受信される。次に、このコードは、メモリ上で擬似実行されるが、この擬似実行は、当該プロセッサ上で当該コードを実行することなくなされ、これにより、当該コードが前記プロセッサにより実行されたとした場合の当該オペレーションが識別されることになる。その後、当該プロセッサ用の指令が作成され、当該オペレーションが行われることになる。

【００１８】
当該オペレーションがデータ構造を初期化するものである場合には、請求項７に記載されるように、前記擬似実行工程は、当該データ構造の初期化を識別するために前記コードを擬似実行するものであるとよい。或いは、当該オペレーションが配列の静的初期化を行うものである場合には、請求項８に記載されるように、前記擬似実行工程は、当該配列の静的初期化を識別するために前記コードを実行するものであるとよい。

【００１９】
更に、請求項９に記載されるように、前記オペレーション(データ構造の初期化や配列の静的初期化等)を行うために前記プロセッサ上で、その作成済命令を実行させる実行工程を含めるとよい。

【００２０】
更に、請求項１０に記載されるように、前記オペレーション(データ構造の初期化や配列の静的初期化等)を行うために仮想マシンにその作成済命令を解釈させる解釈工程を含めるとよい。
【００２１】
また、前記オペレーション(データ構造の初期化や配列の静的初期化)がメモリ効果を有する場合には、請求項１１に記載されるように、前記擬似実行工程は、当該メモリ効果を識別するために、前記コードを擬似実行するものであるとよい。
【００２２】
　次に、本発明に係る請求項１２に記載のデータ処理装置は、記憶デバイスとメモリを備え、当該記憶デバイスは、データ構造の静的初期化を行うソースコードを含むプログラムと、前記データ構造の静的初期化を行う特別メソッドを保持するクラスファイルを少なくとも１つ含むクラスファイルとを保持し、当該メモリは、前記プログラムをコンパイルして前記クラスファイルを生成するコンパイラと、前記クラスファイルを統合し、前記特別メソッドがメモリ上で行う静的初期化の効果を決定するために当該特別メソッドをプロセッサ上で実行することなくメモリ上で疑似実行し、疑似実行の結果に基づいて当該静的初期化を行う命令を作成するプリローダと、前記コンパイラと前記プリローダとを作動させるプロセッサとを備えたことを要旨とするものである。
【００２３】
上記構成を有するデータ処理システムによれば、クラスファイルが統合される際に、プリローダにより全ての<特別>メソッドが識別され、これらのメソッドが擬似実行される。これにより、それらのメソッドにより行われる静的初期化が決定される。次に、<特別>メソッドにより行われる静的初期化を示す表現式が当該プリローダにより作成され、この表現式が.mクラスファイルへ保存され、その<特別>メソッドと置き換えられる。このようにして、多くの命令を保持する<特別>メソッドのコードは、仮想マシンに対して静的初期化を行うように命令する単一の表現式に置き換えられる。これにより、メモリの相当量が節約され、仮想マシンにより実行されるコード量を減少させることができる。この場合、仮想マシンは、この表現式を認識するように修正され、適切な配列の静的初期化がこの仮想マシンにより行われることになる。
【００２４】
この場合に、当該プリローダは、請求項１３に記載されるように、その作成済命令を含む出力ファイルを生成するメカニズムを含むとよく、更に、当該メモリは、請求項１４に記載されるように、その静的初期化を行うために、当該作成済命令を解釈する仮想マシンを含むとよい。
【００２５】
また、前記データ構造は、請求項１５に記載されるように、配列であってもよい。そして、前記特別メソッドは、請求項１６に記載されるように、前記データ構造の静的初期化を行うバイトコードを有するとよい。また、前記作成済命令は、請求項１７に記載されるように、コンスタント・プールへのエントリを含むとよい。
【００２６】
【発明の実施の形態】
本発明に係るシステム及び方法は、<特別>メソッドを、当該<特別>メソッドと同一の静的初期化を仮想マシンに行わせる1又は複数の指令(その仮想マシンにより読み出された際に)に置き換えることにより、Javaプログラミング環境における配列の静的初期化を行う改良システム及び方法を提供するものである。この改良システム及び方法によれば、必要なメモリ容量が少なくて済み且つ必要な時間も短くて済むものである。その結果、配列の静的初期化を行う際のメモリの除去が可能となり、メモリの有効利用が図られることになる。
【００２７】
［概略］

本発明に係るシステム及び方法は、プリローダにおいて所定の予備処理を行うことにより<特別>メソッドに対する必要性を除去するものである。特に、プリローダは、統合用クラスファイルを受信し、<特別>メソッドを検索しながらそれらをスキャンするものである。プリローダが<特別>メソッドを見つける際、メモリに対して<特別>メソッド実行のシミュレーション(即ち、"擬似実行")を行う。これは、Java仮想マシンによって解釈されたとした場合のその<特別>メソッドが有するメモリ効果を決定するためである。即ち、そのプリローダは、Java仮想マシンにより実行される<特別>メソッドの結果となるであろう静的初期化を識別するためにその<特別>メソッド実行のシミュレーションを行うものである。この静的初期化を識別した後、プリローダは、その<特別>メソッドと同一の静的初期化を行わせるために1又は複数の指令(又は命令)を生成し、これらの指令をJava仮想マシンに出力し、当該<特別>メソッドと置き換えるものである。次に、これらの指令は、ランタイム時にJava仮想マシンによって読み出され、そのJava仮想マシンにその<特別>メソッドにより行われたのと同一の静的初期化を行わせる。その指令は、その<特別>メソッドよりも、必要とするメモリスペースが相当量少なくて済む。例えば、上述のコードテーブル#3に記述したバイトコードは、.mクラスファイルに保持される次の指令へ縮小されることになる。その.mクラスは、4つの整数からなる配列で、初期値1，2，3，4を保持している。

CONSTANT_Array T_INT 4 1 2 3 4

本実施形態である仮想マシンは、この表現式を認識し、その配列の静的初期化を行い、その配列を所定値とする。その結果、その形態に係る仮想マシンは、従来の仮想マシンに較べて、静的配列の初期化時に必要なメモリ消費を除去することができる。

【００２８】

［実装の詳細］

図２は、本発明に係るデータ処理システム200の構成を示したものである。データ処理システム200は、コンピュータシステム202を備え、このコンピュータシステム202は、インターネット204に接続されている。コンピュータシステム202は、メモリ206、補助記憶デバイス208、中央処理ユニット(CPU)210、入力デバイス212、及びビデオディスプレイ214を含むものである。メモリ206は、更に、Java™コンパイラ218、Java™プリローダ220、及びJava™ランタイムシステム221を含むものである。Java™ランタイムシステム221は、Java™仮想マシン222を含むものである。補助記憶デバイス208は、ソースコード形態のプログラム224、種々のクラスファイル226、及び.mクラスファイル228を含むものである。Java™コンパイラ218は、プログラム224を1又は複数のクラスファイル226にコンパイルするものである。次に、プリローダ220は、クラスファイル226を受信して.mクラスファイル228を生成する。その.mクラスファイルは、全てのクラスファイルを統合したものである。この統合により、その.mクラスファイル228は、仮想マシン222上で実行可能となる。

【００２９】

本発明に係る処理は、プリローダ220により行われるものであり、プリローダ220は、<特別>メソッドを検索する。そして、その<特別>メソッドが見つかった際に、プリローダ200は、(1)その<特別>メソッドが仮想マシン222により解釈されたとした場合の当該<特別>メソッドによるメモリ効果を決定するために当該<特別>メソッドの実行をシミュレートし、(2)これらのメモリ効果を複製する静的初期化指令を生成し、(3)その<特別>メソッドと置き換えるために、.mクラスファイルにこれらの指令を出力するものである。これにより、相当量のメモリが節約されることになる。

【００３０】

　加えて、本発明に係る処理は、仮想マシン222によって行われるものであるが、これは、プリローダ200の静的初期化指令を認識するように仮想マシン222を修正するためである。本発明の好適な実施形態は、メモリ206に記憶されているものとして説明されているが、いわゆる当業者に自明であるように、その他のコンピュータ読み取り可能な媒体、例えば、補助記憶デバイス(ハードディスク、フロッピーディスク、又はCD-ROM等)、又はRAMやROM等のその他の形態の記憶デバイスに記憶させるようにしてもよい。更に、いわゆる当業者に自明であるように、コンピュータ202が追加の又は異なるコンポーネントを含む

10

20

30

40

50

ものであってもよい。
【００３１】
［プリローダ］
図３は、静的配列の初期化を行うために本発明に係るプリローダ220によって行われる工程のフローチャートを示したものである。プリローダ220により行われる第1の工程では、プリローダ220は、<特別>メソッドを取得するために、クラスファイルを読み出す(ステップ302)。<特別>メソッドを取得した後、プリローダ220は、擬似実行中に使用する種々の変数を割り当てる(ステップ304)。擬似実行に際しては、以下に述べるように、プリローダ220は、<特別>メソッドに保持されるバイトコードの仮想マシンによる実行をシミュレートする。これらのバイトコードは、その<特別>メソッドに関連付けられる種々のデータ構造を処理するものであある。種々のデータ構造には、コンスタント・プール、スタック又はローカル変数(又はレジスタ)が含まれる。
【００３２】
コンスタント・プールは、可変長構造のテーブルであり、種々のストリング定数、クラス名、フィールド名、及びクラスファイル内で参照されるその他の定数を表現するものである。スタックは、そのメソッドの実行中にオペランドを記憶するのに使用するメモリの一部である。従って、スタックサイズは、このメソッドの実行中においては常にそのオペランドにより占有される最大スペースとなる。ローカル変数は、このメソッドにより使用される変数である。
【００３３】
変数割当の際、プリローダ220は、<特別>メソッドのコンスタント・プールに対するポインタを取得し、スタックに適当なサイズを割り当て、配列を割り当てる。その配列割当は、各配列の各エントリが各ローカル変数に対応するようになされる。以下に述べるように擬似実行は、これらの変数を操作するものである。
【００３４】
変数を割当た後、プリローダ220は、<特別>メソッドからバイトコードを読み出す(ステップ306)。次に、プリローダ220は、そのバイトコードを認識するか否かを決定する(ステップ308)。このステップにおいては、プリローダ220は、全てのバイトコードのサブセットを認識する。ここで、このサブセットは、配列の静的初期化を行うのに通常用いられるバイトコードのみを保持するものである。
以下に示すものは、好適な実施形態に係るプリローダ220により認識されるバイトコードのリストを示したものである。
【００３５】
【表４】

ｺｰﾄﾞﾃｰﾌﾞﾙ#4

```
aconst_null     iastore
iconst_m1       lastore
iconst_0        fastore
iconst_1        dastore
iconst_2        aastore
iconst_3        bastore
iconst_4        lastore
iconst_5        sastore
lconst_0        dup
lconst_1        newarray
fconst_0        anewarray
fconst_1        return
fconst_2        ldc
dconst_0        ldc_w
dconst_1        ldc2_w
bipush          putstatic
sipush
```

【００３６】
上記にリストしたバイトコード以外のバイトコードは、認識されない。上述した以外のその他のバイトコードは、<特別>メソッドが配列の静的初期化に対して追加機能を行うものであることを示している。この場合、<特別>メソッドは、最適化され得ない。バイトコードが認識されないと、プリローダ220は、最適化(擬似実行)に不適であると判断し、処理は、ステップ316へ続く。

【００３７】
プリローダ220がバイトコードを認識する場合には、プリローダ220は、そのバイトコードにより反映されるオペレーションを擬似実行する(ステップ310)。このステップにおいては、プリローダ220は、ステップ304において割り当てた変数に基づいてそのバイトコードを擬似実行する。その結果、ある値がスタックからポップされ、ローカル変数は更新され、あるいは、コンスタント・プールからある値が取り出されることになる。加えて、プリローダ220は、特定の静的変数が特定の手法で初期化されるべきことを示す"put static"バイトコードに出くわすことになる。プリローダ220がこのようなバイトコードを受信する場合、プリローダ220は、後で使用するハッシュテーブルへ要求された初期化指示を保存する。ハッシュテーブルは、次の通りである。

Setup: =Array(1,2,3,4)

【００３８】
そのバイトコードにより反映されたオペレーションを行った後、プリローダ220は、<特別>メソッド内に更にバイトコードがあるかどうか決定する(ステップ314)。まだバイトコードがある場合、処理は、ステップ306へ戻る。しかしながら、更なるバイトコードがない場合、プリローダ220は、配列の静的初期化を行うために指令を.mクラスファイルへ保存する(ステップ318)。このステップにおいては、プリローダ220は、コンスタント・プール・エントリを次のようにして.mクラスファイルへ保存する。

タグ	型	サイズ	値
CONSTANT_Array	T_INT	4	１２３４

【００３９】
そのコンスタント・プールのこのエントリは、特定の配列が4つの整数－その初期値がそれぞれ1,2,3,4である－を保持することを示す。実行時においては、仮想マシンは、その

クラスファイルを.mクラスファイルに初期化する場合に、このコンスタント・プールへの参照に出くわし、適当な配列を生成する。その結果、その<特別>メソッドに保持される多くの命令が除去され、1つの表現式として表されることになる。これにより、相当量のメモリが節約され、必要な時間が短縮されることになる。

【００４０】

［プリローダの実装例］

以下の表５から表７(コードテーブル#5)に示した仮コードは、好適な実施形態に係るプリローダの処理の一例を説明したものである。プリローダ220は、"JavaTM仮想マシン仕様"104-106頁に記載されているように、<特別>メソッドを定義するメソッド情報のデータ構造をパラメータとして受信する。そして、プリローダ220は、この<特別>メソッドのバイトコードを擬似実行する。尚、ここで説明するのは、実施形態の一例についてであることに留意されたい。即ち、2，3個のバイトコードがプリローダ220により処理されるものとして以下説明を行うということである。しかしながら、コードテーブル#4の全てのバイトコードが本実施形態により処理可能なものであることは、いわゆる当業者にとって自明のことである。

10

【００４１】

【表５】

コードテーブル#5

```
void emulateByteCodes(Method_info mb)
     int numberRegisters=mb.max_locals();          //ローカル変数の数
     int stackSize=mb.max_stack();                 //スタックサイズ
     byte byteCode[]=mb.code();                    //バイトコード取得
     ConstantPool constantPool=mb.constantPool(); //コンスタント・プール
                                                        //取得

     Object stack[]=new Object[stackSize];       //擬似実行用スタック作成
     Object registers[]=new Object[numberRegisters];// 擬似実行用ローカル
                                                          //変数作成

     /*空スタックの状態でスタート*/
     int stackTop=-1;                              以下、有効要素

     /*静止オブジェクトのマップ化*/
     Hashtable changes=new Hashtable();

try{
          boolean success;
     execution_loop:
          for(int codeOffset=0,nextCodeOffset;
                 ;codeOffset=nextCodeOffset) {
               int opcode=byteCode[codeOffset]&0xFF;   //0..255
               nextCodeOffset=codeOffset+1;  //一般値
               switch(opcode){
                    case opc_iconst_m1:    //スタックに－１をプッシュ
                         stack[++stackTop]=new Integer(-1);
                         break;

                    case opc_bipush:
                         nextCodeOffset=codeOffset+2;
                         stack[++stackTop]=new Integer(byteCode[
                              codeOffset+1);
                         break;

                    case opc_lload_3:          //レジスタ３の内容をロード
                         stack[++stackTop]=(Long)register[3];
                         stack[++stackTop]=null;  //スタック上で２語長
                                                        //使用
                         break;
```

【００４２】
【表６】

（表５の続き）

```
case opc_fsub:{ //アイテムからスタックの先頭を引く
     float b=stack[stackTop--].floatValue();
     float a=stack[stackTop].floatValue();
     stack[stackTop]=new Float(a-b);
     break;
}

case opc_1dc:
     nextCodeOffset=codeOffset+2;
     stack[++stackTop]=
               constantPool.getItem(byteCode(
                    codeOffset+1));
     break;

case sastore:{ //”短い”配列へその内容を保存する
     short value=(short)(stack[StackTop--].intValue(
          ));
     int index=stack[stackTop--].intValue();
     short[] array=(short[])stack[StackTop--];
     array[index]=value;
     break;
}
case opc_putstatic: {
     nextCodeOffset=codeOffset+3;
     int index=((byteCode[codeOffset+1])&0xFF)<<8) +
          (byteCode[codeOffset+2])&0xFF);
     Field f=constantPool.getItem(byteCode[
          codeOffset+1]);
     if (f.getClass() !=mb.getClass() ) {
          //自己のクラスの静的要素修正のみ可能
          throw new RuntimeException();
     }
     Type t=f.getType();
     if (t.isLong() | | t.isDouble())
          ++stackTop;
     Object value=stack[++stckTop]
     changes.put(f,value); //ハッシュテーブルへ
                              //エントリをプット
     break;
```

【００４３】
【表７】

（表６の続き）

```
                case opc_return:
                        success=true;
                        break execution_loop;

                default:        //理解できないバイトコード
                        success=false;
                        break execution_loop;
                }
            }
        } catch(RuntimeException) {
            //ランタイム時の例外処理は、失敗表示
            success=false;
        }
        if(success) {
            <modify .class file as indicated by "changes"hashtable>
            <Remove this<clinit>method from the class>
        }else{
            <ran into something we cannot understand>
            <do not replace this method>
    }
}
```

【００４４】

［実施形態に係る仮想マシン］

上述したように、Java仮想マシン222は、以下に説明する点について修正が加えられる以外は、"Java™仮想マシン仕様"で定義される標準のJava仮想マシンであればよい。従来の仮想マシンは、種々のコンスタント・プール・エントリ、例えば、CONSTANT_Integer，CONSTANT_String，及びCONSTANT_Long等、を認識するものである。これらの型のコンスタント・プール・エントリは、種々の変数情報を保持するものであり、その初期値も含むものである。しかしながら、実施形態に係る仮想マシンは、更に、コンスタント・プールのCONSTANT_Arrayエントリをも認識するものである。

【００４５】

クラスファイルフォーマットにおけるコンスタント・プール・エントリのCONSTANT_Arrayのフォーマットは、以下の通りである。

【００４６】

【表８】

コードテーブル#6

```
CONSTANT_Array_info {
    u1 tag;                /*CONSTANT_Array のリテラル値*/
    u1 type;               /*以下参照*/
    u4 length;             /*配列の要素数*/
    ux objects[length];  /*実際値*/
                         /*型がT-CLASSの場合に限り、次の
                           フィールドが含まれる*/
    u2 type2;            /*コンスタント・プール内の
                           CONSTANT_CLASSのインデックス*/

}
```

10

【００４７】
u1型フィールドは、表９に示したテーブルにリストされた値のいずれかである。
【００４８】
【表９】

20

配列型	値
T_CLASS	2
T_BOOLEAN	4
T_CHAR	5
T_FLOAT	6
T_DOUBLE	7
T_BYTE	8
T_SHORT	9
T_INT	10
T_LONG	11

30

【００４９】
フィールドuxオブジェクト[長さ]は、値の配列であり、その配列要素を提供するものである。その配列中の要素数は、コンスタント・プール・エントリの長さフィールドにより与えられるものである。これらの値の実際のサイズは、次の表１０に示す通りである。
【００５０】
【表１０】

40

型	ux	意味
T_BOOLEAN, T_BYTE	u1	1 byte
T_CHAR, T_SHORT, T_CLASS	u2	2 bytes
T_INT, T_FLOAT	u4	4 bytes
T_LONG, T_DOUBLE	u8	8 bytes

【００５１】

50

表１０に示したバイトは、T_CLASS以外の型については、配列の要素に保存される実際の値である。一方、T_CLASSについては、各u2は、それ自体がコンスタント・プールのエントリに対するインデックスである。参照されるコンスタント・プール・エントリは、CONSTANT_Array、CONSTANT_Object、又はNULL値を示す特別コンスタント・プールエントリ0のいずれかでなければならない。

【００５２】

例えば、配列

int[] = {10，20，30，40};

を示すには、コンスタント・プール・エントリは、次のようになる。

タグ	型	サイズ	初期値
CONSTANT_Array	T_INT	4	10 20 30 40

10

【００５３】

他の例としては、配列

new Foo[3] /*全てNULLに初期化済み*/

を示すには、コンスタント・プール・エントリは、次のようになる。

タグ	型	サイズ	初期値	クラス
CONSTANT_Array	T_CLASS	3	0 0 0	xx

ここで、"xx"は、コンスタント・プール内のクラスFooを示すコンスタント・プールのインデックスである。

20

【００５４】

二次元配列は次の通りである。

new byte[][]={{1，2，3，4}，{5，6，7，8}};

この二次元配列は、副次的配列を符号化する2つのコンスタント・プール・エントリを保持することにより、また、その副次的配列間の関連を示す2つの追加エントリを保持することにより、符号化されるものである。この符号化は、オブジェクトの型としてのJava™の配列概念、及び配列の配列としての多次元配列に対応している。上記の二次元配列のコンスタント・プール・エントリは、次の通りである。

Entry1:　CONSTANT_Array T_BYTE 4 1 2 3 4

30

Entry2:　CONSTANT_Array T_BYTE 4 5 6 7 8

Entry3:　CONSTANT_Class with name "[[B"

	タグ	型	サイズ	初期値	クラス
Entry 4:	CONSTANT_Array	T_CLASS	2	Entry1 Entry 2	Entry 3

ここで、Entry1，Entry2及びEntry3は、対応するコンスタント・プール・エントリのインデックスの2バイトで符号化したものである。

【００５５】

40

以上、本発明に係るシステム及び方法について好適な実施の形態に基づいて説明したが、本発明がこの実施形態に限定されるものでないことは、いわゆる当業者にとっては、自明である。種々の変形例が本発明の趣旨に従って可能である。尚、本発明の範囲は、特許請求の範囲によって定められる。

【００５６】

【発明の効果】

以上説明したように、本発明に係る請求項１に記載のデータ処理装置が行う配列の静的初期化方法によれば、配列の静的初期化により当該配列を静的値とするバイトコードを保存する特別メソッド付きのクラスファイルを生成するために、当該静的配列を保存するソースコードをコンパイルするコンパイル工程と、コンパイラから送信された前記クラスファ

50

イルをプリローダにより受信する受信工程と、前記プロセッサによって特別メソッドが実行されたとした場合になされる前記配列の静的初期化を識別するために、前記プロセッサ上で当該コードを実行することなく、メモリ上で前記プリローダにより前記特別メソッドのバイトコードを疑似実行する疑似実行工程と、前記疑似実行の結果に基づいて前記配列の静的初期化を実現するための命令を出力ファイルに保存する保存工程と、前記配列の静的初期化を行うために仮想マシンにより前記出力ファイルに保存された表現を解釈する解釈工程とを備えたので、配列の静的初期化に必要なメモリ使用量を節約することができる。

【００５７】

本発明に係る請求項６に記載のデータ処理装置によれば、オペレーションを行うために前記プロセッサ上で実行されるコードをコンパイラから受信する受信工程と、前記コードが前記プロセッサにより実行されたとした場合の当該コードのオペレーションを識別するために、前記プロセッサ上で当該コードを実行することなくメモリ上で当該コードを前記プリローダにより疑似実行する疑似実行工程と、前記コードのオペレーションを行うために前記プロセッサに対する命令を作成する作成工程とを備えたので、配列の静的初期化に必要なメモリ使用量を節約することができる。

【００５８】

本発明に係る請求項１２に記載のデータ処理装置によれば、クラスファイルが統合される際に、プリローダにより全ての<特別>メソッドが識別され、これらのメソッドが擬似実行され、それらのメソッドにより行われる静的初期化が決定される。次に、<特別>メソッドにより行われる静的初期化を示す単一の表現式が当該プリローダにより作成され、この単一の表現式が.mクラスファイルへ保存され、その<特別>メソッドと置き換えられる。これにより、多くの命令を保持する<特別>メソッドのコードが、仮想マシンに対して静的初期化を行うように命令する単一の表現式に置き換えられるため、仮想マシンにより実行されるコード量を減少させることができ、配列の静的初期化に必要なメモリ使用量を節約することができる。

【００５９】

本発明に係る方法及びシステムによれば、処理に必要な時間も短縮されることから、本発明は、情報通信産業の更なる進展に極めて有益なものである。

【図面の簡単な説明】

【図１】　Javaプログラム環境におけるプログラム開発工程のフローチャートである。

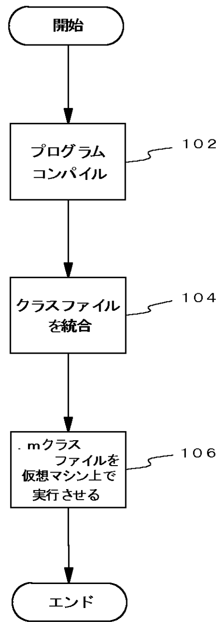
【図２】本発明に係るデータ処理システムの構成図である。

【図３】図２に示したプリローダによって行われる工程のフローチャートである。

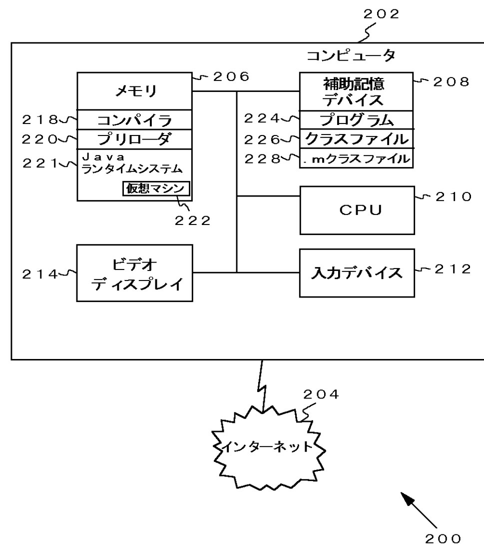
【符号の説明】

200　データ処理システム
202　コンピュータシステム
204　インターネット
206　メモリ
208　補助記憶デバイス
210　CPU
212　入力デバイス
214　ビデオディスプレイ
218　コンパイラ
220　プリローダ
221　Javaランタイムシステム
224　プログラム
226　クラスファイル
228　.mクラスファイル

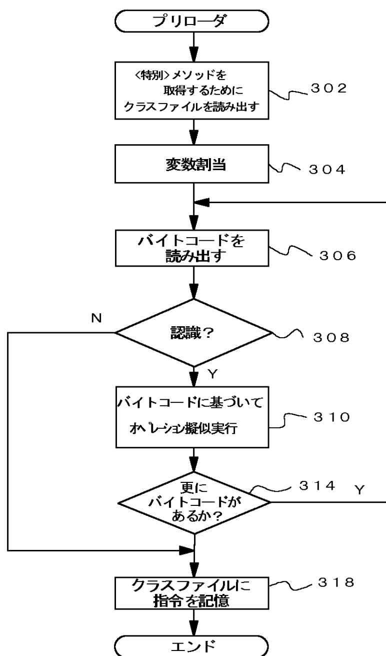
【図１】



【図２】



【図３】



フロントページの続き

(72)発明者　タック　リチャード　ディー
アメリカ合衆国　９４１１４　カリフォルニア州、サン　フランシスコ、ヒル　ストリート　３４
３

審査官　坂庭　剛史

(56)参考文献　特開平１０－０４０１０７（ＪＰ，Ａ）
特開平０７－０７３０４４（ＪＰ，Ａ）
とどそふと，Ｃ／Ｃ＋＋プログラマのための研究Ｊａｖａ，Ｃ MAGAZＩNE，日本，ソフトバンク株式会社，１９９６年　5月　1日，第8巻，第5号，ｐ．２５
Jon Meyer，Troy Downing，ＪＡＶＡバーチャルマシン，日本，株式会社オライリー・ジャパン，１９９７年　7月２8日，初版，ｐ．９９～１０１，１１５～１１６

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　9/45

(54)【発明の名称】データ処理システムの配列の静的初期化方法、データ処理方法、並びにデータ処理システム及びその制御手順をコンピュータに実行させるプログラムを記憶したコンピュータ読み取り可能な記憶媒体