## 多次元配列の内部

多次元配列の扱いに慣れてきたところで、その内部を詳しく学習することにしましょう。最初に学習した２行４列の配列の宣言は、次のようになっていました。

```
int[][] x = new int[2][4];
```

この宣言では、２次元配列 x の配列変数を宣言するとともに、本体の生成を同時に行っています。配列変数の宣言と本体の生成を別々に行うと、次のようになります。

```
1 int[][] x;
2 x = new int[2][];
3 x[0] = new int[4];
4 x[1] = new int[4];
```

実に４段階もの宣言・処理に分解されます。このことは、２次元配列の内部構造が複雑であることを示唆しています。**Fig.6-17** を見ながら、理解していきましょう。

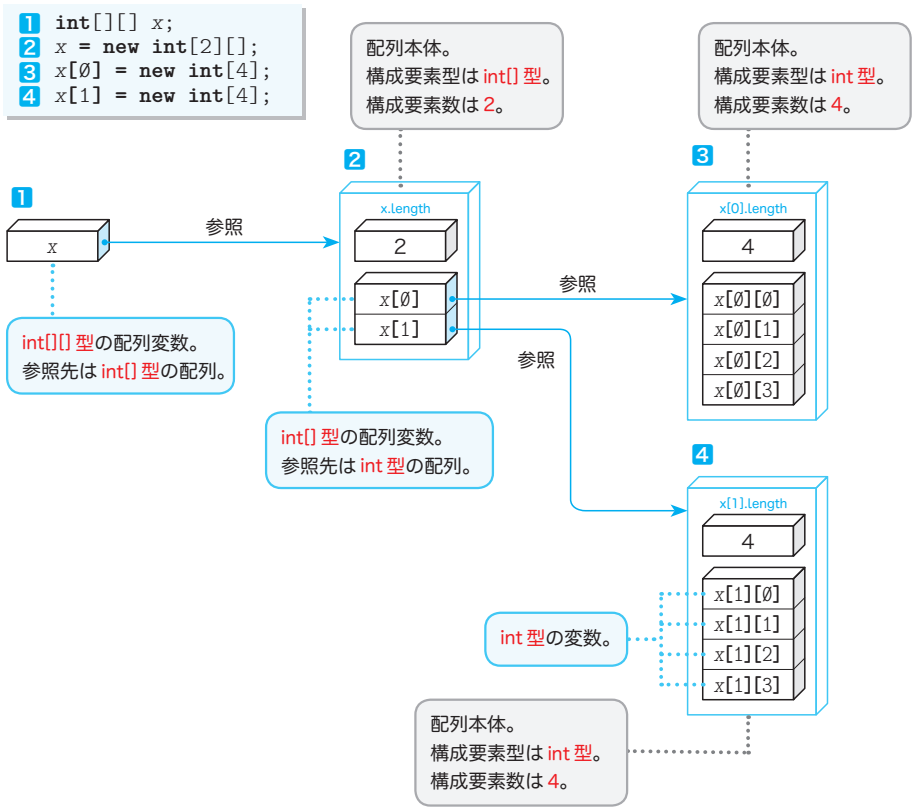


**Fig.6-17**　２次元配列の物理的なイメージ

1 ２次元配列 x の宣言です。int[][] 型の x は、配列本体ではなく、配列変数です。

2 配列本体を生成するとともに、x がそれを参照するように代入を行います。ここで生成するのは、以下の配列です。

> 構成要素型が int[] 型で構成要素数が 2 の配列

生成した配列は x によって参照されるわけですから、その各要素をアクセスする式は x[0]，x[1] です。

また、この配列の構成要素数 2 は、x.length として取得できます。

3 配列本体を生成するとともに、x[0] がそれを参照するように代入を行います。ここで生成するのは、以下の配列です。

> 構成要素型が int 型で構成要素数が 4 の配列

生成した配列は x[0] によって参照されるわけですから、その各要素をアクセスする式は x[0][0]，x[0][1]，x[0][2]，x[0][3] です。

また、この配列の構成要素数 4 は、x[0].length として取得できます。

4 配列本体を生成するとともに、x[1] がそれを参照するように代入を行います。ここで生成するのは、以下の配列です。

> 構成要素型が int 型で構成要素数が 4 の配列

生成した配列は x[1] によって参照されるわけですから、その各要素をアクセスする式は x[1][0]，x[1][1],x[1][2]，x[1][3] です。

また、この配列の構成要素数 4 は、x[1].length として取得できます。

*

x にとって、構成要素は int[] 型の x[0] と x[1] の２個です。そして、要素は int 型の x[0][0]，x[0][1]，…，x[1][3] の８個です。

**注意すべき点は、行が異なる要素の配置が連続しないことです。**たとえば、記憶域上の x[0][3] の直後に x[1][0] が格納されるわけではありません。

### Column 6-6 多次元配列の宣言の形

次の宣言を考えましょう。x と y の型は分かりますか？

```
int[] x, y[];
```

正解は、x は１次元配列 int[] 型で、y は２次元配列 int[][] 型です。このような紛らわしい宣言は避け、以下のように素直に宣言すべきです。

```
int[] x;          // xはint[]型の配列   （１次元配列）
int[][] y;        // yはint[][]型の配列 （２次元配列）
```