

第10回のキーワード

1

アルゴリズム関係

- 連結リストへの挿入
- 連結リストからの削除
- 双方向連結リスト
(doubly linked list)
- 循環リスト
(circular linked list)
- $O(1)$
- 連結リストの探索と整列

Java関係

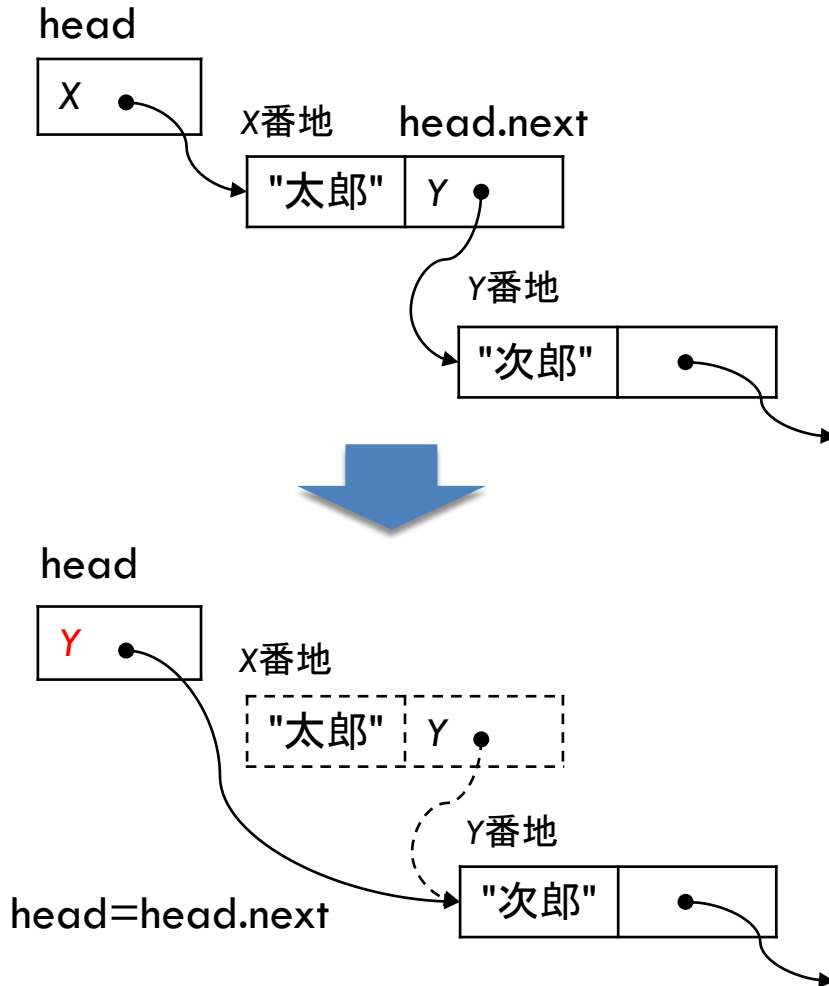
- ジェネリクス / 総称型
(generics)
- ジェネリッククラスの定義

```
class Node<E> { ... }  
class List<E> { ... }
```
- `java.util.LinkedList<E>`

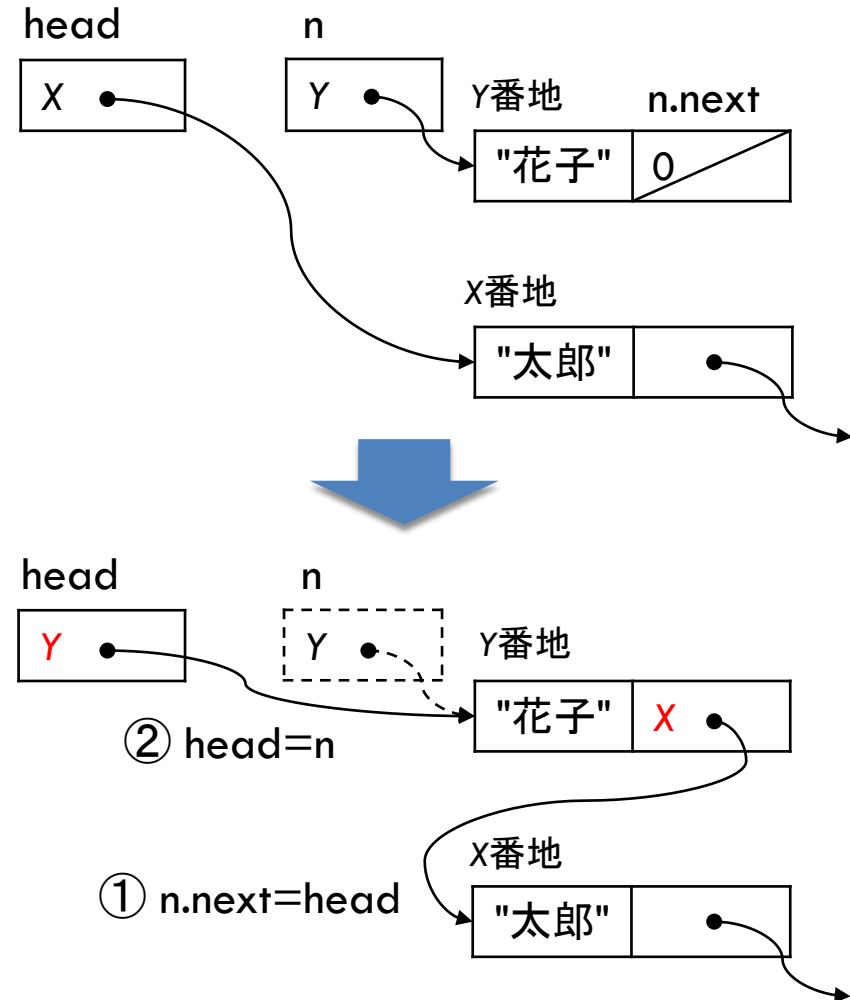
リスト先頭での削除と挿入

2

□ 先頭ノードの削除 (pop)



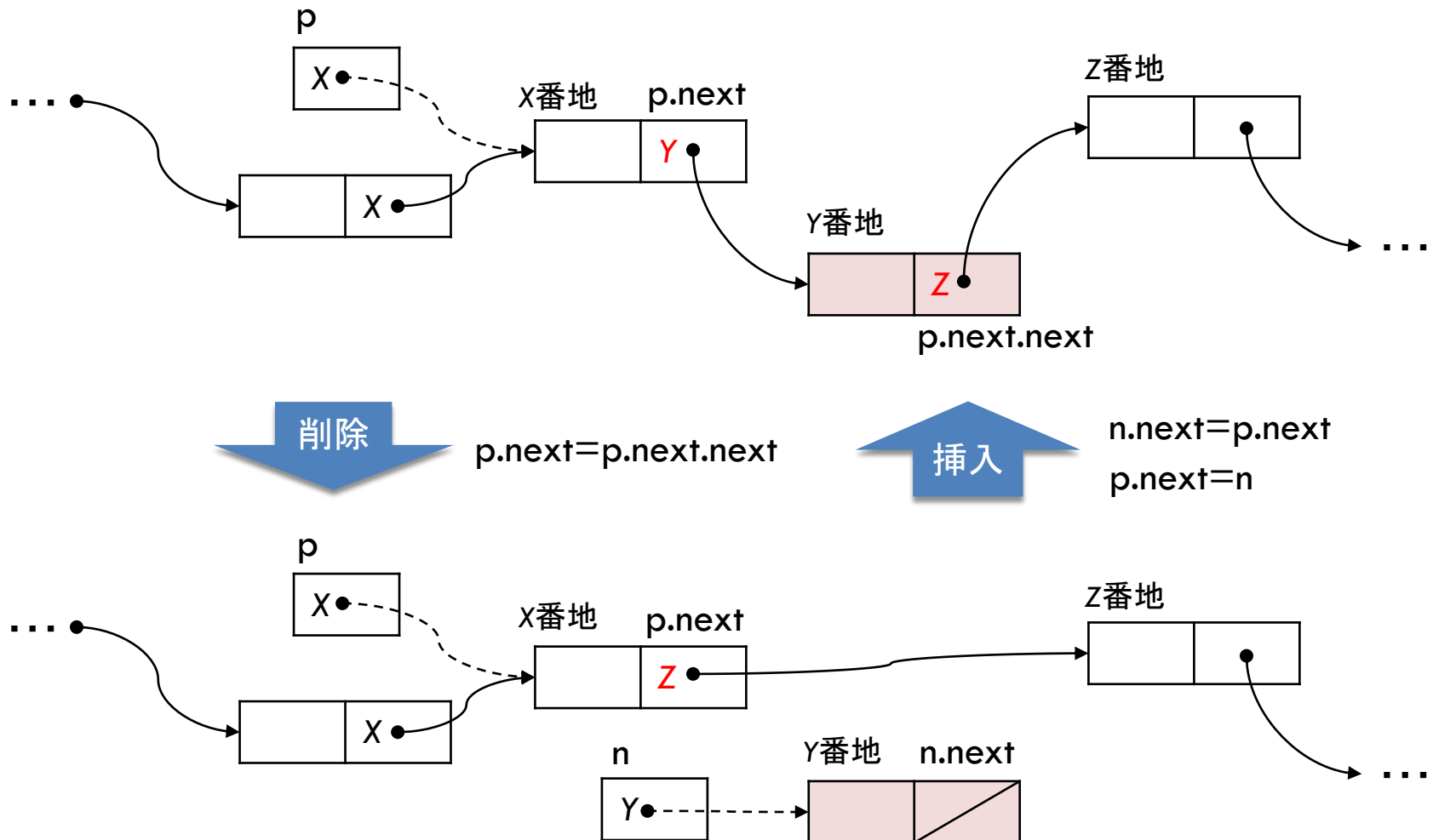
□ 先頭ノードの挿入 (push)



リスト途中での削除と挿入

3

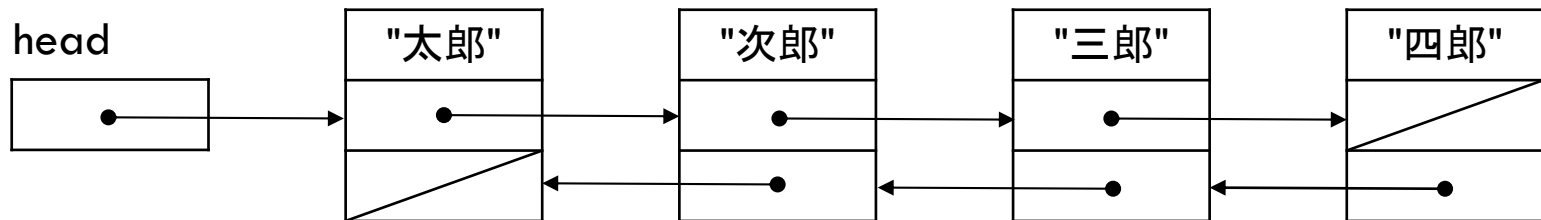
- リストの途中(や末尾)でノードを削除・挿入する



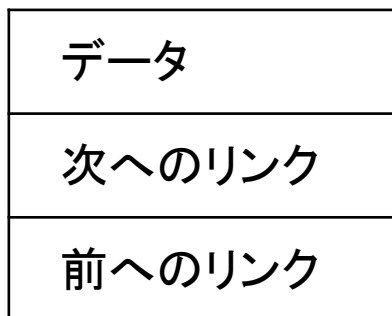
双方向連結リスト

4

- 前後のノードへのリンクを保持
 - ▣ 前から後ろだけでなく, 後ろから前にもたどれる
 - ▣ 現在指しているノードを削除することができる



- ノードの構造

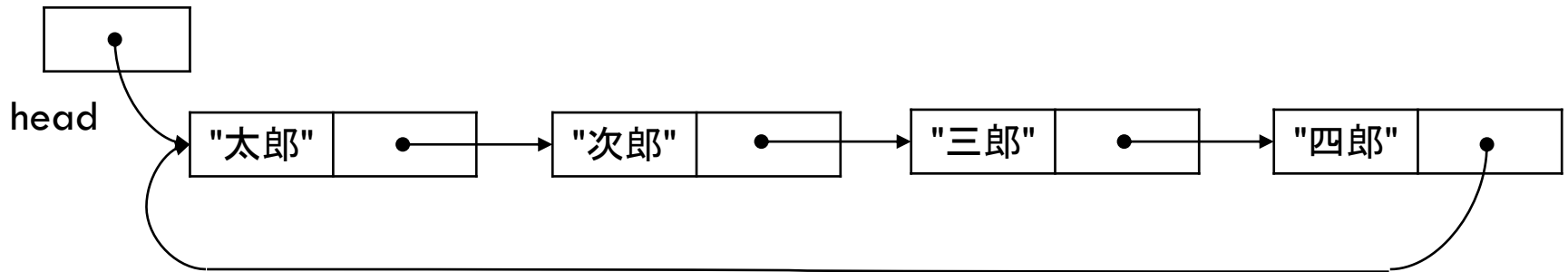


```
class Node {
    String data;
    Node next; // 次へのリンク
    Node prev; // 前へのリンク
}
```

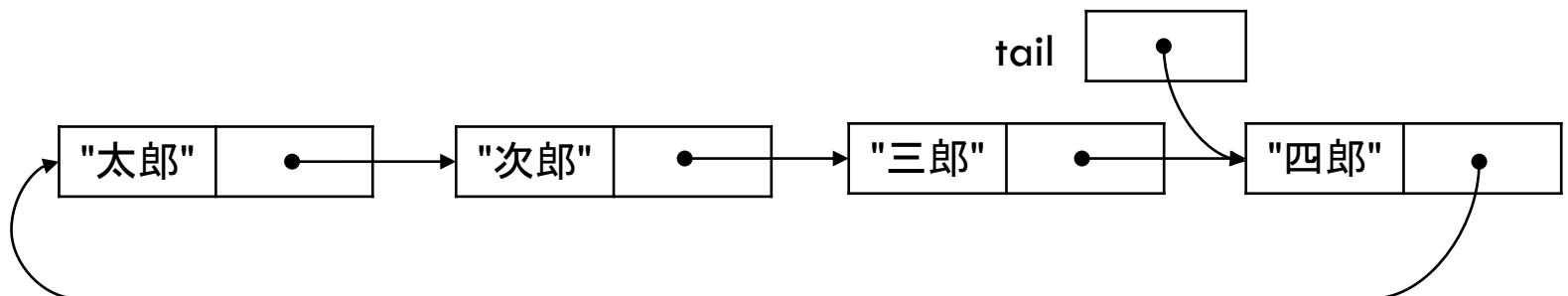
循環リスト

5

- 末尾から先頭につなげる
 - リストを「回転」し, 順に繰り返して処理することが容易



- 効率的なキューの実装に利用できる
 - 末尾 (tail) だけ分かれば, 先頭はその次で求められる



連結リストの探索と整列

6

- 配列 vs 連結リスト
 - ▣ 配列: ランダムアクセス(途中の要素を直接参照可能)
 - ▣ 連結リスト: シーケンシャルアクセス(順にたどるしかない)

- 連結リストの探索
 - ▣ 基本的には, 先頭から順にたどって線形探索する
 - ▣ 高速化したい場合は, リストではなく木構造などを用いる

- 連結リストの整列
 - ▣ 要素の交換ではなく, ノードの移動を使うように工夫する
 - ▣ 挿入ソートは, 要素をずらす処理をせずに実現できる
 - ▣ マージソートは, 一時的な領域を確保せずに実現できる

Javaジェネリクス

7

- ジェネリック(総称的)プログラミング
 - ▣ アルゴリズムを特定のデータ型(クラス)に依存しないようにプログラムし, 再利用可能なテンプレートとする
 - ▣ テンプレートに様々なデータ型を当てはめると, それぞれに対応したコードが生成される

- Javaジェネリクス
 - ▣ クラスまたはメソッド定義において, 「<仮クラス名>」という書式によって使用するクラスをパラメータ化できる
 - ▣ 定義の中では, 仮クラス名(「総称型」という)を使用する
 - ▣ インスタンスは, 実際のクラス名を当てはめて生成する
 - ▣ 例: `class Node<E> { ... } → Node<String> node;`