封装为通用的树形结构处理代码

1. /\*\*
2. \* 树型数据深度遍历
3. \* @param child 树节点
4. \* @return boolean
5. \*/
6. private boolean isIterator(TreeNode child) {
7. for (Iterator<TreeNode> itsub = new DepthIterator(root.iterator()); itsub.hasNext();) {
8. TreeNode childone = itsub.next();
9. if (childone == null)
10. return Boolean.FALSE;
11. if (child.getParentId().equals(childone.getId())) {
12. keyMap.put(child.getId(), null);
13. childone.addNode(child);
14. return Boolean.TRUE;
15. }
16. }
17. return Boolean.FALSE;
18. }
20. 深度遍历代码
22. import java.util.Iterator;
24. /\*\*
25. \* <p>
26. \* 普通树的深度遍历方法
27. \* </p>
28. \*
29. \* @author kinjoYang
30. \* @version 1.0 2011-03-06
31. \* @since 1.0
32. \* @see java.util.Iterator
33. \*/
34. public class DepthIterator implements Iterator<TreeNode> {
36. /\*\* 迭代器栈 \*/
37. private RzStack<Iterator<TreeNode>> stack = new RzStack<Iterator<TreeNode>>();
39. public DepthIterator(Iterator<TreeNode> it) {
40. stack.push(it);
41. }
43. public DepthIterator() {
44. }
46. /\*\* 是否存在下个元素,没有则返回下级元素的迭代器遍历 \*/
47. public boolean hasNext() {
48. if (stack.isEmpty()) {
49. return Boolean.FALSE;
50. } else {
51. Iterator<TreeNode> it = stack.peek();
52. if (it.hasNext()) {
53. return Boolean.TRUE;
54. } else {
55. stack.pop();
56. return hasNext();
57. }
58. }
59. }
61. /\*\* 返回下个元素,无则返回<tt>null</tt> \*/
62. public TreeNode next() {
63. if (hasNext()) {
64. Iterator<TreeNode> it = stack.peek();
65. TreeNode next = it.next();
66. if (next.getSize() > 0) {
67. stack.push(next.iterator());
68. }
69. return next;
70. } else {
71. return null;
72. }
73. }
75. /\*\* 不提供删除方法 \*/
76. public void remove() {
77. throw new UnsupportedOperationException("Can't remove node :Kinjo");
78. }
80. }
81. //大概应该是这样,当每个节点展开时,
82. List<TreeNode> getTreeNode(){
83. Collection<Authrition> currentAuths = getCurrentAuths();
84. String sql = "from TreeNode t where t.permission in (:perms)";
85. Query q = ..;
86. q.setParameterList("perms",currentAuths );
87. return q.list():
88. }