如何加载和显示分层结构化评论

韩博孙

2020 年 7 月 19 日 麻省理工学院

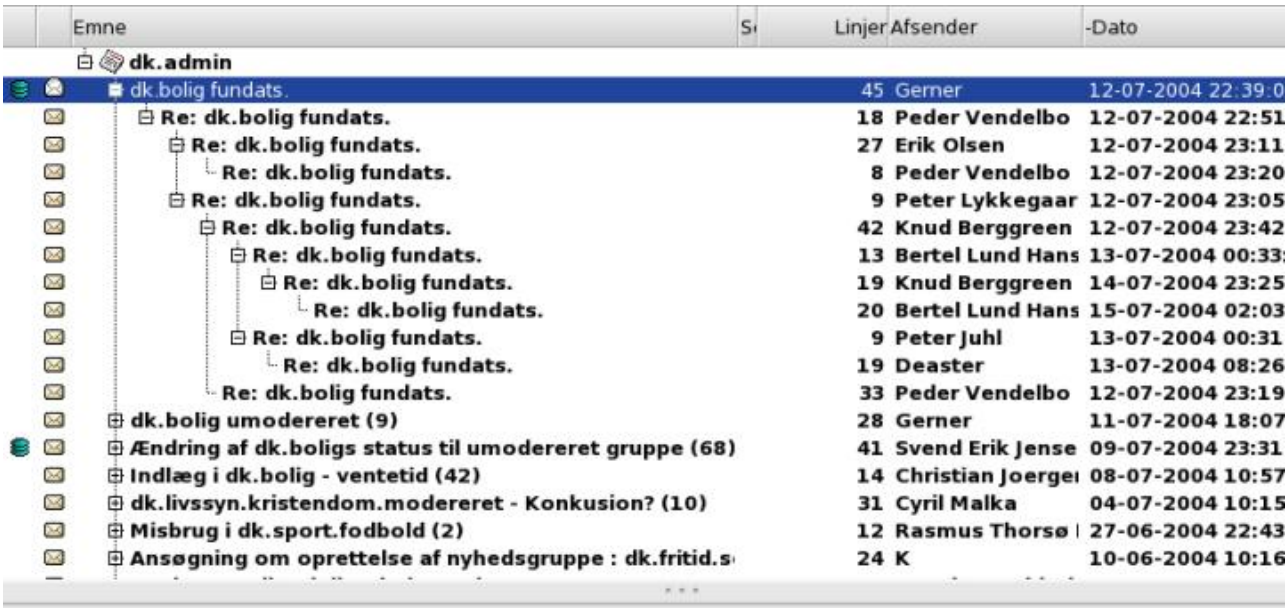
本教程将向您展示如何使用 RESTFul 服务和 JavaScript 加载和显示分层结构化注释。

下载源代码 - 461.6 KB

介绍

什么是分层评论？分层注释是一种以倒置树的形式显示注释的方式。也就是说，在顶层，有一个或多个节点。这些顶级节点中的每一个都有零个或多个子节点。然后每个子评论将有零个或多个孙子。孙辈评论也是如此，每个人都会有零个或多个曾孙辈，依此类推。

将其可视化的最简单方法是考虑`TreeView`在 Windows 操作系统中调用的 UI 控件。如果您从未使用过 Win32 编程或 C# WinForms，这里是另一种可视化分层注释显示的方法 - 对话线程。这是来自 wikipedia.org 的屏幕截图：



资料来源：维基百科

在 1996 年到 2000 年之间，这种显示论坛消息的方式最为流行。到 2002 年，他们中的大多数都切换到了不同的显示方式（评论按时间顺序排列，最新或最旧的评论在顶部）。如今，论坛消息的分层评论显示几乎绝迹。但是，这种对博客文章评论或对新闻报道的评论的显示仍然很突出。您甚至可以在 Microsoft Outlook 中为电子邮件对话启用此类显示格式。我相信上面的截图是 Outlook 的早期版本。

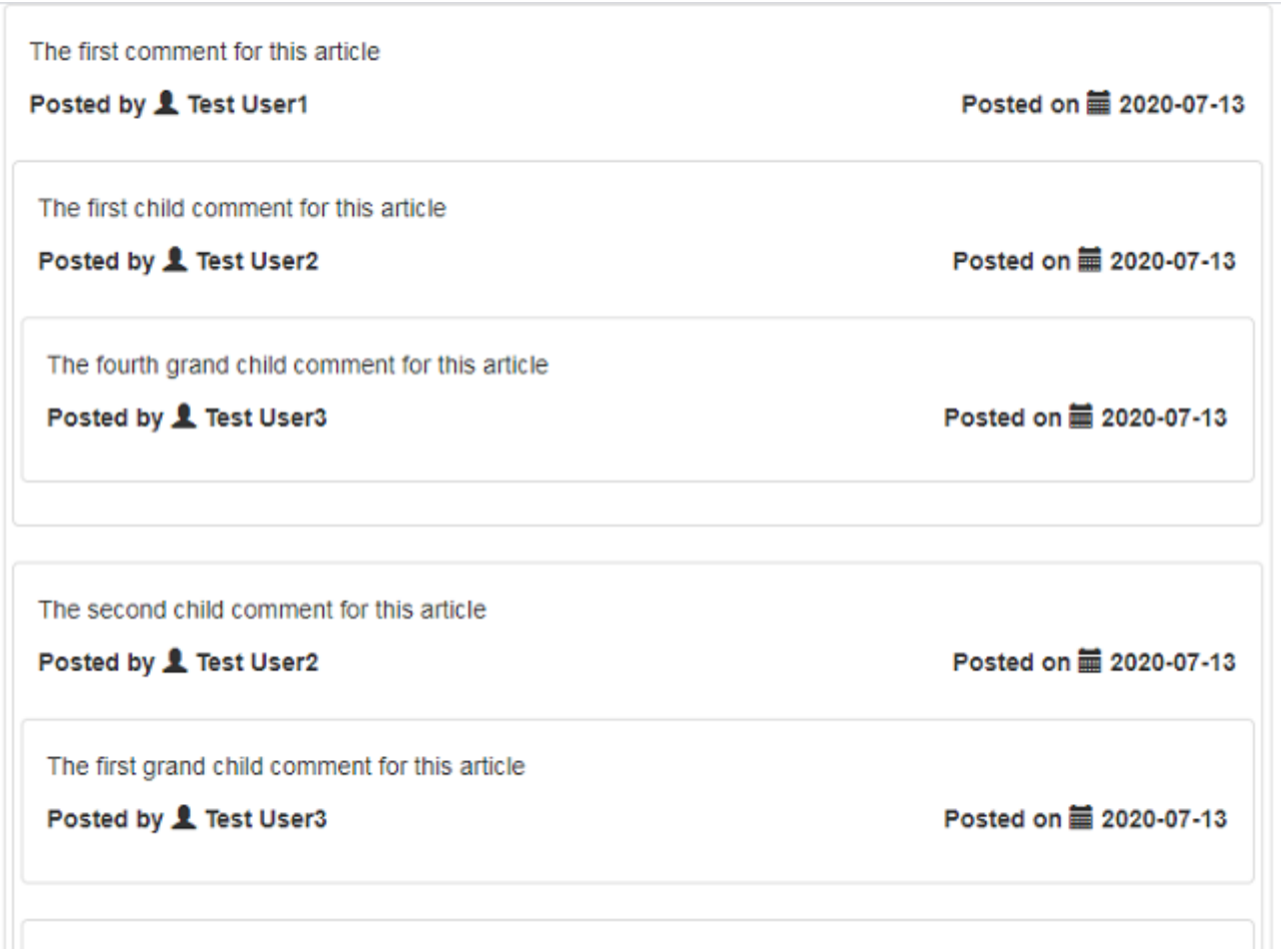
为什么这对我很重要？从我发现第一个论坛的那天起，我就对复制这种技术很感兴趣。起初，这似乎很难。然后我忘记了它（因为失去了兴趣）。最近，我手头有一点时间，所以我尝试复制这个技术。经过一番思考（无需上网寻找答案），结果证明这很容易。所以我做了实现，并将在本教程中讨论该技术。

算法概述

分层评论加载/显示没有秘诀。您只需要深度优先搜索算法。如果你不知道这是什么，请查一下。该算法的难点在于递归的使用。如果您不想使用递归，则可以使用堆栈来完成。我不想用 JavaScript 实现堆栈数据结构。所以我用递归做到了。此时，您只需要知道算法是什么，使用什么编程技术，深度优先搜索和递归。关于实现的更多信息将在后面的部分中解释。

架构概述

本文包含一个示例项目。这个示例项目是一个基于 Spring Boot 的 Web 应用程序。当用户的浏览器运行此应用程序时，它将使用 RestFUL Web 服务从后端获取两组评论，然后页面使用 JavaScript 以正确的层次结构呈现它们。这是一个屏幕截图：



有两组评论，每组有一个根评论，然后有多个孩子，孙子和可能的曾孙子在他们之下。

该应用程序有两个部分，一个是后端 Web 服务。Web 服务检索所有评论，然后将所有评论组织在右父子结构中，然后将它们发送回前端进行渲染以进行显示。这里分开的问题是：

- 后端服务负责根据父子层次结构组织评论。
- 前端应用程序将假定评论以正确的层次结构构建，并使用 DFS（深度优先搜索）在页面上呈现评论。

后端 Web 服务使用 Spring Boot 和 Spring REST 向前端提供一些模拟数据。将有一个服务对象将评论组织到正确的层次结构中。然后评论列表将作为 JSON 列表发回。

前端是使用 RequireJS、StapesJS 和 JQuery 编写的 JavaScript 应用程序。为了呈现页面，必须进行大量的 DOM 操作。JQuery 是其中最好的。StapesJS 和 RequireJS 只是为了很好地构建应用程序。我还添加了 axios 以与后端服务交互、获取数据或尝试出错。

评论的数据模型

我所做的假设是评论与博客文章、页面或某些评论相关联。这使我可以通过一个简单的查询获取所有评论。在这个示例应用程序中，没有数据库查询。仅供参考，以防您想实现应用程序的其余部分。

以下是 Comment 数据模型对象的完整代码列表：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.rest.models;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonFormat;

public class CommentModel
{
   private String articleId;

   private String commentId;

   private String parentCommentId;

   private List<CommentModel> childComments;

   private String content;

   private String commenterName;

   private String commenterEmail;

   @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")
   private Date createDate;

   public String getArticleId()
   {
      return articleId;
   }

   public void setArticleId(String articleId)
   {
      this.articleId = articleId;
   }

   public String getCommentId()
   {
      return commentId;
   }

   public void setCommentId(String commentId)
   {
      this.commentId = commentId;
   }

   public String getContent()
   {
      return content;
   }

   public void setContent(String content)
   {
      this.content = content;
   }

   public String getCommenterName()
   {
      return commenterName;
   }
```

```
    public void setCommenterName(String commenterName)
    {
        this.commenterName = commenterName;
    }

    public String getParentCommentId()
    {
        return parentCommentId;
    }

    public void setParentCommentId(String parentCommentId)
    {
        this.parentCommentId = parentCommentId;
    }

    public List<CommentModel> getChildComments()
    {
        return childComments;
    }

    public void setChildComments(List<CommentModel> childComments)
    {
        this.childComments = childComments;
    }

    public String getCommenterEmail()
    {
        return commenterEmail;
    }

    public void setCommenterEmail(String commenterEmail)
    {
        this.commenterEmail = commenterEmail;
    }

    public Date getCreateDate()
    {
        return createDate;
    }

    public void setCreateDate(Date createDate)
    {
        this.createDate = createDate;
    }


    public CommentModel()
    {
        this.childComments = new ArrayList<CommentModel>();
    }
}
```

如代码所示，这个类类型有一个`articleId`，这允许我一次获取所有评论。接下来，每个评论都有`commentId`，这个属性唯一地标识了评论本身。每个评论对象都有一个`parentCommentId`. 这是父评论的参考。如果评论是根评论，则其`parentCommentId`将设置为`null`. 这个属性也是我可以按层次顺序重新组织评论的方式。最后，有`childComments`. 此列表属性在重新组织期间使用，收集当前评论的所有直接子级。此类的其他属性用于显示目的。

获取评论并重新组织

获取和重新组织评论的方式是在服务对象中完成的。我声明了这个服务对象的一个接口。像这样：

爪哇 复制代码

```
package org.hanbo.boot.rest.services;

import java.util.List;
```

```
import org.hanbo.boot.rest.models.CommentModel;

public interface CommentsService
{
   List<CommentModel> getArticleComments();
}
```

这是接口的完整实现：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.rest.services;

import java.util.Date;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.hanbo.boot.rest.models.CommentModel;
import org.springframework.stereotype.Service;

@Service
public class CommentsServiceImpl
   implements CommentsService
{
   @Override
   public List<CommentModel> getArticleComments()
   {
      List<CommentModel> allCmnts = dummyComments();
      List<CommentModel> reorganizedCmnts = reorganizeComments(allCmnts);

      return reorganizedCmnts;
   }

   private List<CommentModel> reorganizeComments(List<CommentModel> allCmnts)
   {
      Map<String, CommentModel> mappedComments
         = sortCommentsIntoMap(allCmnts);

      addChildCommentsToParent(allCmnts, mappedComments);

      List<CommentModel> retVal = topMostComments(allCmnts);

      return retVal;
   }

   private List<CommentModel> topMostComments(List<CommentModel> allCmnts)
   {
      List<CommentModel> retVal = new ArrayList<CommentModel>();

      if (allCmnts != null)
      {
         for (CommentModel cmnt : allCmnts)
         {
            if (cmnt != null)
            {
               if (cmnt.getParentCommentId() == null || cmnt.getParentCommentId().equals(""))
               {
                  retVal.add(cmnt);
               }
            }
         }
      }

      return retVal;
   }
```

```
private void addChildCommentsToParent(List<CommentModel> allCmnts,
             Map<String, CommentModel> mappedComments)
{
   if (allCmnts != null && mappedComments != null)
   {
      for (CommentModel cmnt : allCmnts)
      {
         if (cmnt != null)
         {
            String parentCmntId = cmnt.getParentCommentId();
            if (parentCmntId != null && !parentCmntId.equals("") &&
                                    parentCmntId.length() > 0)
            {
               if (mappedComments.containsKey(parentCmntId))
               {
                  CommentModel parentCmnt = mappedComments.get(parentCmntId);
                  if (parentCmnt != null)
                  {
                     parentCmnt.getChildComments().add(cmnt);
                  }
               }
            }
         }
      }
   }
}

private Map<String, CommentModel> sortCommentsIntoMap(List<CommentModel> allCmnts)
{
   Map<String, CommentModel> mappedComments  = new HashMap<String, CommentModel>();

   if (allCmnts != null)
   {
      for (CommentModel cmnt : allCmnts)
      {
         if (cmnt != null)
         {
            if (!mappedComments.containsKey(cmnt.getCommentId()))
            {
               mappedComments.put(cmnt.getCommentId(), cmnt);
            }
         }
      }
   }

   return mappedComments;
}

private List<CommentModel> dummyComments()
{
   List<CommentModel> retVal = new ArrayList<CommentModel>();

   CommentModel modelAdd = new CommentModel();
   modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
   modelAdd.setCommenterEmail("testuser1@goomail.com");
   modelAdd.setCommenterName("Test User1");
   modelAdd.setCommentId("05e6fe83ab3f4f3bbf2e5ab75eda277b");
   modelAdd.setContent("The first comment for this article");
   modelAdd.setCreateDate(new Date());
   modelAdd.setParentCommentId(null);

   retVal.add(modelAdd);

   modelAdd = new CommentModel();
   modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
   modelAdd.setCommenterEmail("testuser2@goomail.com");
   modelAdd.setCommenterName("Test User2");
   modelAdd.setCommentId("c2769bfd2d0a49a0920737d854e43c53");
   modelAdd.setContent("The first child comment for this article");
```

```
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("05e6fe83ab3f4f3bbf2e5ab75eda277b");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser2@goomail.com");
        modelAdd.setCommenterName("Test User2");
        modelAdd.setCommentId("b22b8a8cce0b4aa196d5e54e902be761");
        modelAdd.setContent("The second child comment for this article");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("05e6fe83ab3f4f3bbf2e5ab75eda277b");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser3@goomail.com");
        modelAdd.setCommenterName("Test User3");
        modelAdd.setCommentId("4d457f242dd34cef89835067c45a7d3f");
        modelAdd.setContent("The first grand child comment for this article");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("b22b8a8cce0b4aa196d5e54e902be761");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser3@goomail.com");
        modelAdd.setCommenterName("Test User3");
        modelAdd.setCommentId("f009e0879b0e4538b4f45788ca5e0adc");
        modelAdd.setContent("The second grand child comment for this article");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("b22b8a8cce0b4aa196d5e54e902be761");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser3@goomail.com");
        modelAdd.setCommenterName("Test User3");
        modelAdd.setCommentId("3b51af94ad7944359967f7df6436a9b0");
        modelAdd.setContent("The third grand child comment for this article");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("b22b8a8cce0b4aa196d5e54e902be761");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser3@goomail.com");
        modelAdd.setCommenterName("Test User3");
        modelAdd.setCommentId("2c6d0abd27a2404caeef29f3dc1049cd");
        modelAdd.setContent("The fourth grand child comment for this article");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("c2769bfd2d0a49a0920737d854e43c53");

        retVal.add(modelAdd);

        //----------------

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser3@goomail.com");
        modelAdd.setCommenterName("Test User3");
        modelAdd.setCommentId("ef0d7c94fb6948f383835def70c09a79");
        modelAdd.setContent("This is a second comment on the same article.");
        modelAdd.setCreateDate(new Date());
```

```
        modelAdd.setParentCommentId(null);

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser5@goomail.com");
        modelAdd.setCommenterName("Test User5");
        modelAdd.setCommentId("fcf5c1f63ec84f65bcac7fcd44fd0509");
        modelAdd.setContent("Child comment #1 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("ef0d7c94fb6948f383835def70c09a79");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser5@goomail.com");
        modelAdd.setCommenterName("Test User5");
        modelAdd.setCommentId("f0e383e91bb9456abf13d0dc1f7d1ba7");
        modelAdd.setContent("Child comment #2 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("ef0d7c94fb6948f383835def70c09a79");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser5@goomail.com");
        modelAdd.setCommenterName("Test User5");
        modelAdd.setCommentId("8d26b047dedf4d948cc87b72fb55bba4");
        modelAdd.setContent("Child comment #3 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("ef0d7c94fb6948f383835def70c09a79");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser6@goomail.com");
        modelAdd.setCommenterName("Test User6");
        modelAdd.setCommentId("f7e1c2cfbe00474ab5caafc8497641ea");
        modelAdd.setContent("Grand child comment #1 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("f0e383e91bb9456abf13d0dc1f7d1ba7");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser7@goomail.com");
        modelAdd.setCommenterName("Test User7");
        modelAdd.setCommentId("c5de23d8609f4d819d235dc6867f7917");
        modelAdd.setContent("Grand child comment #2 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("8d26b047dedf4d948cc87b72fb55bba4");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser7@goomail.com");
        modelAdd.setCommenterName("Test User7");
        modelAdd.setCommentId("1d8a871aebbe486595e3d9d3aecb8713");
        modelAdd.setContent
             ("Grand child comment #3 of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("8d26b047dedf4d948cc87b72fb55bba4");
```

```
        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser8@goomail.com");
        modelAdd.setCommenterName("Test User8");
        modelAdd.setCommentId("f700db39af674f939165a4f6799668ec");
        modelAdd.setContent("Great Grand child comment #1
                             of the second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("c5de23d8609f4d819d235dc6867f7917");

        retVal.add(modelAdd);

        modelAdd = new CommentModel();
        modelAdd.setArticleId("525d346cbd784180bd09bc8a52be3e1a");
        modelAdd.setCommenterEmail("testuser8@goomail.com");
        modelAdd.setCommenterName("Test User8");
        modelAdd.setCommentId("05eddfc462834adf84a2e4a4b7c81b06");
        modelAdd.setContent("Great Grand child comment #2 of the
                             second comment of the same article.");
        modelAdd.setCreateDate(new Date());
        modelAdd.setParentCommentId("1d8a871aebbe486595e3d9d3aecb8713");

        retVal.add(modelAdd);

        return retVal;
    }
}
```

这个服务对象做了几个操作：

- `dummyComments()`实现类中的最后一个方法准备所有模拟数据。这些都是与同一博客文章或页面相关的无组织评论。
- 调用最后一个方法的第二个方法`sortCommentsIntoMap()`。该方法的目的是将所有评论放入一个`HashMap`对象中，关键是评论Id。价值是评论本身。
- 调用底部的第三个方法`addChildCommentsToParent()`。此方法将对评论进行排序并将子评论添加到父评论。它是通过在`HashMap`集合的帮助下使用原始列表来完成的。两个集合具有相同的对象引用，因此可以通过反向查找将子注释添加到父级。
- 调用倒数第四个方法`topMostComments()`。它挑选出所有的根注释。这些是有子级的注释，但没有父级注释引用。调用此方法时，注释的重新组织已经完成。
- 下一个方法（来自前一个）被称为`reorganizeComments()`。该方法调用所有其他方法，根据父子关系对所有评论进行重新组织，然后挑选出根评论，放入列表并返回。
- top 方法是`public`rest 控制器将调用的方法。它首先调用该方法`dummyComments()`来返回无组织的评论列表。然后它调用`reorganizeComments`方法将评论组织成正确的层次结构。最后，返回根注释列表。

此实现中最重要的方法是`sortCommentsIntoMap()`and `addChildCommentsToParent()`，`sortCommentsIntoMap()`将遍历无组织的注释并将它们放入`HashMap`. 由此`HashMap`，将子评论链接到父评论会更容易。这就是方法的`addChildCommentsToParent()`作用。一旦重组完成。我所需要的只是找到根注释并将它们作为列表返回，这就是方法`topMostComments()`所做的。

用于返回评论的 RestFUL API 控制器在名为 的类中定义，该类`CommentsController`可以在 *src/main/java/org/hanbo/boot/rest/controllers*子文件夹中找到。你可以亲眼看看它的作用。

这些就是我们后端服务所需要的。接下来是前端渲染。

分层评论渲染

后端网络服务可以将评论组织成层次结构。接下来，前端将检索根评论并以正确的层次顺序显示所有评论。本节将解释如何做到这一点。

在正确的位置呈现评论是其中最难的部分。在此解决方案之前，我从来没有时间考虑解决方案。现在想想，最明显的解决方案是使用深度优先搜索（DFS）算法。

DFS 的目的是树遍历，访问树的每个节点。这个想法很简单，在任何给定的节点上，首先访问它的所有子节点。在每个子节点上，该节点成为当前节点，重复同样的过程，先访问子节点。通过这个递归，它将遍历所有的孩子、孙子和曾孙子，等等。一旦没有更多子节点要访问，则将当前节点标记为已访问。使用此算法的棘手部分是何时呈现消息以及何时呈现子评论。原来这相当于访问子评论与将当前节点标记为已访问。我选择在访问完所有子项后呈现当前评论的消息。我想它不会

在进入 DFS 的实现之前，我将首先用 JavaScript 展示从后端服务中获取评论的服务对象：

JavaScript 复制代码

```
define (["axios"], function (axios) {
   var svc = {
      loadAllComments: function(articleId) {
         reqUrl = "/allComments";
         return axios.get(reqUrl);
      }
   };

   return svc;
});
```

正如我之前提到的，我在整个应用程序中使用了 RequireJS。该`define()`函数创建了一个可重用的组件，该组件可以注入到其他组件中。该组件执行一项操作，向`GET`硬编码的 URL发送请求。然后将承诺返回给调用者。调用者将处理响应。这在名为*commentService.js*的文件中定义。

接下来，我们将进入前端应用程序的核心，重新组织评论集合的实际渲染。这是整个源代码，都在名为*forum.js*的文件中：

JavaScript 缩小▲ 复制代码

```
define(["jquery", "bootstrap", "stapes", "underscore", "commentsService"],
         function($, boot, Stapes, _, commentsService) {
   var articleComments = Stapes.subclass({
      $el: null,
      cmntsArea: null,

      constructor : function() {
         var self = this;
         self.$el = $("#articleComments");
         self.cmntsArea = self.$el.find("#allCmnts");
      },

      allComments : function() {
          commentsService.loadAllComments().then(function(results) {
            if (results && results.status === 200) {
               if (results.data && results.data.length > 0) {
                  console.log(results.data);
                  var allCmnts = renderComments(results.data);
                  if (allCmnts != null) {
                     for (var i = 0; i < allCmnts.length; i++)
                     self.cmntsArea.append(allCmnts[i][0]);
                  }
               }
            } else {
               console.log("error occurred.");
            }
         }).catch(function(error) {
            if (error) {
               console.log(error);
            }
            console.log("HTTP error occurred.");
         }).finally(function() {

         });
      }
   });

   function renderComments(comments) {
      var allCmnts = [];
      if (comments != null && comments.length > 0) {
```

```
        for (var i = 0; i < comments.length; i++) {
            var cmnt = renderComment(comments[i]);
            if (cmnt != null) {
                allCmnts.push(cmnt);
            }
        }
    }

    return allCmnts;
}

function renderComment(comment) {
    var retVal = $("<div></div>");

    if (comment != null) {
        var childComments = null;
        if (comment.childComments != null &&
            comment.childComments.length > 0) {
            childComments = renderChildComments(comment.childComments);
        }

        if (childComments == null) {
            childComments = [];
        }

        var content = renderCommentContent(comment);

        var cmntFrame= $("<div class='row'></div>").append(
            $("<div class='col-xs-12'></div>").append(
                $("<div class='thumbnail'></div>").append(content).append(childComments)
            )
        );

        retVal = cmntFrame;
    }

    return retVal;
}

function renderChildComments(childComments) {
    var childCmnts = [];
    if (childComments != null && childComments.length > 0) {
        for (var i = 0; i < childComments.length; i++) {
            var cmnt = renderComment(childComments[i]);
            if (cmnt != null) {
                childCmnts.push(cmnt[0]);
            }
        }
    }

    return childCmnts;
}

function renderCommentContent(comment) {
    var retVal = $("<div style='margin: 12px 10px;'></div>");

    if (comment != null) {
        var cmntContentTmpl = "<div class='row'>" +
            "<div class='col-xs-12'>" +
            "<p><%= postContent %></p>" +
            "</div>" +
            "<div class='col-xs-6'>" +
            "<p><strong>Posted by <i class='glyphicon glyphicon-user'></i>
             <%= commenterName %></strong></p>" +
            "</div>" +
            "<div class='col-xs-6 text-right'>" +
            "<p><strong>Posted on <i class='glyphicon glyphicon-calendar'></i>
             <%= postDate %></strong>" +
            "</div>" +
```

```
            "</div>";
         var cmntContentTmpl = _.template(cmntContentTmpl);
         var contentToAdd = cmntContentTmpl({
            commenterName: comment.commenterName,
            postDate: comment.createDate,
            postContent: comment.content
         });

         retVal.append($(contentToAdd));
      }

      return retVal;
   }

   return {
      run: function () {
         console.log("forum run.");

         var cmnts = new articleComments();
         cmnts.allComments();
      }
   };
});
```

这是整个项目最大的代码文件。在这个文件中，我使用 StapesJS、下划线 JS 和 JQuery 定义了一个组件。它将在页面上找到一个区域，并添加显示分层结构化注释的 DOM。该组件的主要方法称为`allComments()`. 它从后端加载所有注释，然后运行渲染。这里是：

JavaScript 复制代码

```
allComments : function() {
    commentsService.loadAllComments().then(function(results) {
      if (results && results.status === 200) {
         if (results.data && results.data.length > 0) {
            console.log(results.data);
            var allCmnts = renderComments(results.data);
            if (allCmnts != null) {
               for (var i = 0; i < allCmnts.length; i++)
               self.cmntsArea.append(allCmnts[i][0]);
            }
         }
      } else {
         console.log("error occurred.");
      }
   }).catch(function(error) {
      if (error) {
         console.log(error);
      }
      console.log("HTTP error occurred.");
   }).finally(function() {

   });
}
```

在这个方法中`allComments()`，在从后端服务返回评论列表后，它会调用一个内部方法`renderComments()`。此方法获取评论列表并进行渲染。在其中，渲染被委托给另一个实现了 DFS 的方法：

JavaScript 复制代码

```
function renderComments(comments) {
   var allCmnts = [];
   if (comments != null && comments.length > 0) {
      for (var i = 0; i < comments.length; i++) {
         var cmnt = renderComment(comments[i]);
         if (cmnt != null) {
            allCmnts.push(cmnt);
         }
      }
```

```
    }

    return allCmnts;
}
```

基于 DFS 的渲染通过两种方法实现，第一种称为`renderComment()`. 它需要一个评论，并试图呈现自己和它的所有后代。这个方法是这样的：

JavaScript 缩小▲ 复制代码

```
function renderComment(comment) {
    var retVal = $("<div></div>");

    if (comment != null) {
        var childComments = null;
        if (comment.childComments != null &&
            comment.childComments.length > 0) {
            childComments = renderChildComments(comment.childComments);
        }

        if (childComments == null) {
            childComments = [];
        }

        var content = renderCommentContent(comment);

        var cmntFrame= $("<div class='row'></div>").append(
            $("<div class='col-xs-12'></div>").append(
                $("<div class='thumbnail'></div>").append(content).append(childComments)
            )
        );

        retVal = cmntFrame;
    }

    return retVal;
}
```

这是实现 DFS 搜索和递归的地方。如果评论没有任何子级，我会创建一个空数组并将该空数组作为子级 DOM 附加到当前评论的 DOM 元素中。如果有任何孩子，则该方法调用另一个方法`renderChildComments()`。该方法`renderChildComments()`将子级注释呈现为 DOM 元素列表。渲染子评论后，调用`renderCommentContent()`来渲染评论内容。最后，当子评论列表和当前评论内容可用时，将它们作为父子连接在一起。然后将返回最终结果。

这是该方法的代码`renderChildComments()`：

JavaScript 复制代码

```
function renderChildComments(childComments) {
    var childCmnts = [];
    if (childComments != null && childComments.length > 0) {
        for (var i = 0; i < childComments.length; i++) {
            var cmnt = renderComment(childComments[i]);
            if (cmnt != null) {
                childCmnts.push(cmnt[0]);
            }
        }
    }

    return childCmnts;
}
```

上面列出的代码非常简单，对于每个子注释，只需递归调用方法即可`renderComment()`。返回的结果将是呈现的 html DOM 节点。我将所有 DOM 节点收集到一个数组中并返回该数组。请注意：

JavaScript 复制代码

```
if (cmnt != null) {
   childCmnts.push(cmnt[0]);
}
```

我必须这样做的原因是，我使用 JQuery 来构造 DOM 对象。输出将是一个数组。如果我不取数组的元素，将JQuery对象加回到返回的数组中，我将返回一个数组的数组。这将无法正确显示。这就是为什么我必须使用数组索引并将 HTML 节点放入要返回的数组中。

在这里做一个简单的总结，方法`renderComment()`和方法`renderChildComments()`构成了一个递归，完成了DFS算法。如果您仍然不明白，请运行应用程序，并通过此递归调试步骤。时间一长，就清楚了。

关于这个组件，我想介绍的最后一件事是评论内容的呈现。`UnderscroreJS`是一个很酷的实用程序库。它提供了广泛的帮助方法，可以使生活变得非常轻松。其中一种方法是`_.template()`。它可以将常规字符串转换为字符串模板。模板本身是一个方法，它接受一个对象并使用该对象的属性来替换字符串模板中的占位符以创建最终的字符串。您可以在名为 的方法中看到这一点`renderCommentContent()`。

在这个方法中`renderCommentContent()`，字符串模板的定义方式是这样的：

JavaScript 复制代码

```
var cmntContentTmpl = "<div class='row'>" +
   "<div class='col-xs-12'>" +
   "<p><%= postContent %></p>" +
   "</div>" +
   "<div class='col-xs-6'>" +
   "<p><strong>Posted by <i class='glyphicon glyphicon-user'></i>
       <%= commenterName %></strong></p>" +
   "</div>" +
   "<div class='col-xs-6 text-right'>" +
   "<p><strong>Posted on <i class='glyphicon glyphicon-calendar'></i>
       <%= postDate %></strong>" +
   "</div>" +
   "</div>";
```

这是创建实际模板的方法，然后用对象的属性值替换占位符：

JavaScript 复制代码

```
var cmntContentTmpl = _.template(cmntContentTmpl);
var contentToAdd = cmntContentTmpl({
   commenterName: comment.commenterName,
   postDate: comment.createDate,
   postContent: comment.content
});
```

这就是评论渲染的全部内容。在下一节中，我想介绍设置 RequireJS 配置的技术，以便 BootStrap 和 JQuery 可以在 HTML 页面中加载一次。

使用 RequireJS 配置应用程序

刚开始使用RequireJS的时候，不知道怎么用JQuery和BootStrap JS文件来配置。在我制作的第一个教程中，这两个文件显示在同一个 HTML 文件的两个不同位置。有一个更好的配置方法，这样两个 JS 文件只会在 HTML 文件中显示一次。

主要的应用程序源代码可以在文件*app.js 中*找到，它看起来像这样：

JavaScript 复制代码

```
requirejs.config({
   paths: {
      jquery: "/assets/jquery/js/jquery.min",
      bootstrap: "/assets/bootstrap/js/bootstrap.min",
      stapes: "/assets/stapes/stapes-min-1.0.0",
      axios: "/assets/axios/axios.min",
      underscore: "/assets/underscore/underscore-min-1.9.2",

      commentsService: "/assets/app/commentsService",
      forum: "/assets/app/forum"
```

```
    },
    shim: {
        bootstrap: {
            deps: ["jquery"]
        }
    }
});
require([ "forum" ], function (forum) {
    forum.run();
});
```

使用 RequireJS，我需要使用一种叫做 shim 配置的东西。我需要它的原因是 bootstrap 的 JavaScript 文件不是在 RequireJS 的支持下编写的。所以为了解决这个问题，需要垫片配置。配置基本上意味着它就像这样简单：

JavaScript 复制代码

```
shim: {
    bootstrap: {
        deps: ["jquery"]
    }
}
```

它没有完成。在 HTML 页面上，我仍然需要设置要加载的 JavaScript 文件。请看一下文件*index.html*。JavaScript 文件包含为：

HTML 复制代码

```
<html>
    ...
    <body>
        <script type="text/javascript" src="/assets/requirejs/require.js"></script>
        <script type="text/javascript" src="/assets/app/app.js"></script>
    </body>
</html>
```

如图所示，只有两个 JavaScript 文件被添加到*index.html*文件中。当它在浏览器（如 Chrome）中加载时，页面将注入所有其他 JavaScript 文件。这是一个屏幕截图，您只能在页面呈现后通过“检查”页面来查看这些注入的文件：

```
<html>
▼<head>
    <title>Stacked Comments Display - by Han</title>
    <link href="/assets/bootstrap/css/bootstrap.min.css" rel="stylesheet">
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="forum" src="/assets/app/forum.js"></script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="jquery" src="/assets/jquery/js/jquery.min.js"></script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="stapes" src="/assets/stapes/stapes-min-1.0.0.js"></script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="underscore" src="/assets/underscore/underscore-min-1.9.2.js">
    </script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="commentsService" src="/assets/app/commentsService.js"></script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="axios" src="/assets/axios/axios.min.js"></script>
    <script type="text/javascript" charset="utf-8" async data-requirecontext="_" data-requiremodule="bootstrap" src="/assets/bootstrap/js/bootstrap.min.js"></script>
  </head>
▼<body>
```

这是页面中两个 JavaScript 文件的屏幕截图：

```
    ▶<footer class="footer">…</footer>
      ::after
    </div>
    <script type="text/javascript" src="/assets/requirejs/require.js"></script>
    <script type="text/javascript" src="/assets/app/app.js"></script>
  </body>
</html>
```

如何测试此应用程序

下载源代码后，请转到文件夹*src/main/resources/static/assets*，并将*.sj文件重命名为*.js*。

这是一个基于 Spring Boot 的应用程序，要编译整个项目，请在基本文件夹中运行以下命令，您可以在其中找到*POM.xml*：

复制代码

```
mvn clean install
```

等待构建完成，它会成功。然后使用以下命令运行应用程序：

复制代码

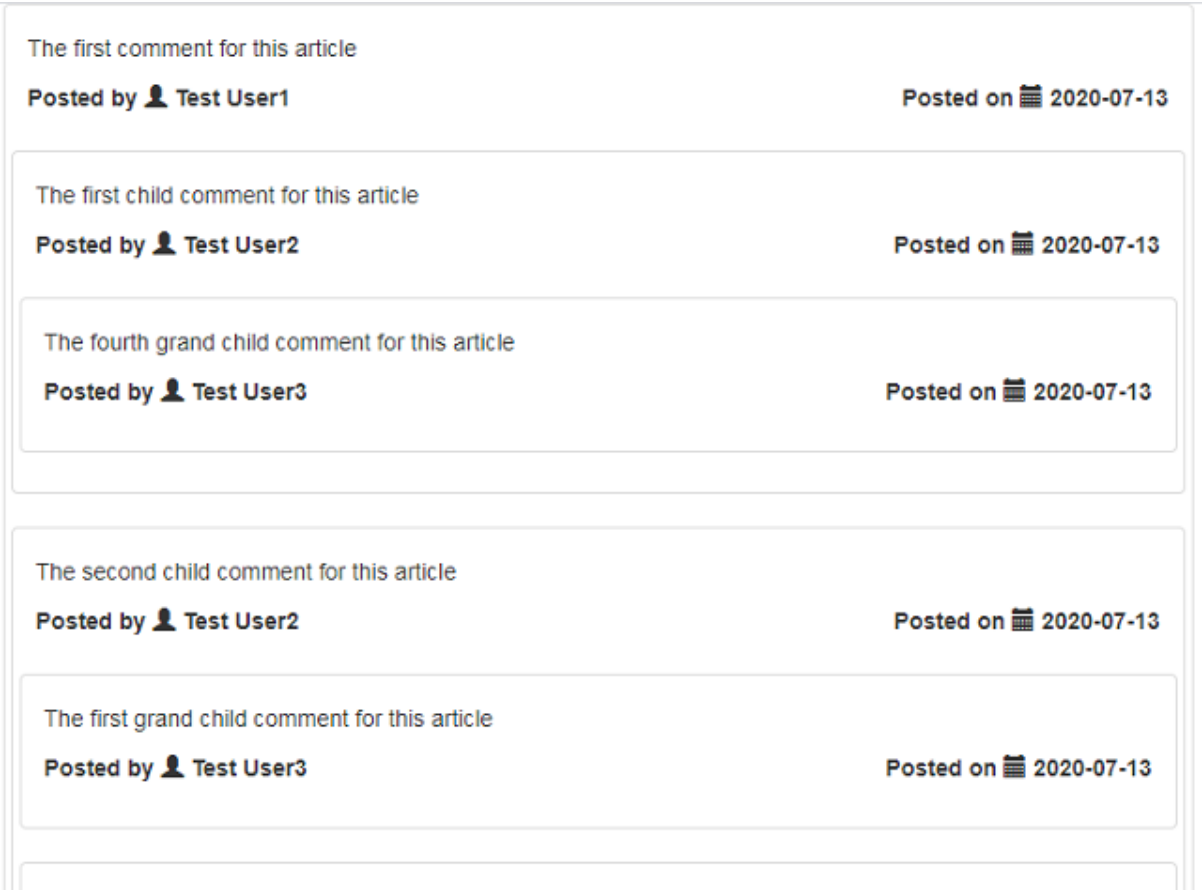
```
java -jar target/hanbo-forumtest-1.0.1.jar
```

等待应用程序启动。然后打开 Chrome 或 Firefox，并导航到以下 URL：

复制代码

```
http://localhost:8080
```

页面加载完成后，将显示所有按层次结构排列的评论，如本教程中显示的第二个屏幕截图，如下所示：



如果您能看到这一点，则说明示例应用程序构建成功。

概括

本教程解释了如何加载分层注释并在网页上正确呈现它们。有两个技术问题需要解决。一种是如何加载评论，然后在层次结构中组织它们。这是在后端 Web 服务上完成的。如教程中所示，我使用HashMap, 和多次循环迭代将子注释添加到父注释。

更大的问题是如何在前端应用程序获取评论后呈现评论。在本教程中，我描述了使用深度优先搜索作为遍历评论树并在正确的层次结构中呈现所有评论的一种方式。该实现使用递归来完成评论遍历。此外，本教程还展示了如何在运行时构建 html DOM 节点。最后，我展示了如何使用 RequireJS shim 配置加载 JavaScript 文件。

我知道本教程使用了一些现成的 JavaScript 库。它们用于实施解决方案。只要了解它的工作原理，就可以使用其他库轻松实现相同的解决方案。

历史

- 2020 年 7 月16日 - 初稿

执照

分享

关于作者

韩博孙

组长 The Judge Group

美国

手表该会员

没有提供传记

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

-- 本论坛暂无消息 --