package cn.wlyf.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import cn.wlyf.common.pojo.EasyUITreeNode;

import cn.wlyf.common.utils.E3Result;

import cn.wlyf.content.service.ContentCategoryService;

@Controller

public class ContentCatController{

@Autowired

private ContentCategoryService contentCategoryService

@RequestMapping("/content/category/list")

@ResponseBody

public List<EasyUITreeNode> getContentCatList(

@RequestParam(name="id",defaultValue="0") Long parentId){

List<EasyUITreeNode> list = contentCategoryService.getContentCatList(parentId);

return list;

}

/\*\*

\* 添加分类节点

\*/

@RequestMapping(value="/content/category/create",method=RequestMethod.POST)

@ResponseBody

public E3Result createContentCategory(Long parentId,String name){

//调用服务添加节点

E3Result e3Result = contentCategoryService.addContentCategory(parentId, name);

return e3Result;

}

}

package cn.wlyf.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import cn.wlyf.common.utils.E3Result;

import cn.wlyf.content.service.ContentService;

import cn.wlyf.pojo.TbContent;

/\*\*

\*

\* 内容管理Controller

\*

\*/

@Controller

public class ContentController {

@Autowired

private ContentService contentService;

@RequestMapping(value="/content/save",method=RequestMethod.POST)

@ResponseBody

public E3Result addContent(TbContent content){

//调用服务把内容数据保存到数据库

E3Result e3Result = contentService.addContent(content);

return e3Result;

}

}

package cn.wlyf.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import cn.wlyf.common.pojo.EasyUITreeNode;

import cn.wlyf.service.ItemCatService;

@Controller

public class ItemCatController {

@Autowired

private ItemCatService itemCatService;

@RequestMapping("/item/cat/list")

@ResponseBody

public List<EasyUITreeNode> getItemCatList(@RequestParam(name="id",defaultValue="0")Long parentId)

//调用服务查询节点列表

List<EasyUITreeNode> list = itemCatService.getItemCatlist(parentId);

return list;

}

}

package cn.wlyf.common.pojo;

import java.io.Serializable;

import java.util.List;

public class EasyUIDataGridResult implements Serializable{

/\*\*

\* serial version UID

\*/

private static final long serialVersionUID = 1L;

private long total;

private List rows;

public long getTotal() {

return total;

}

public void setTotal(long total) {

this.total = total;

}

public List getRows() {

return rows;

}

public void setRows(List rows) {

this.rows = rows;

}

}

package cn.wlyf.common.pojo;

import java.io.Serializable;

public class EasyUITreeNode implements Serializable{

private long id;

private String text;

private String state;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

}

package cn.wlyf.common.jedis;

public interface JedisClient {

String set(String key, String value);

String get(String key);

Boolean exists(String key);

Long expire(String key, int seconds);

Long ttl(String key);

Long incr(String key);

Long hset(String key, String field, String value);

String hget(String key, String field);

Long hdel(String key, String... field);

}

package cn.wlyf.common.jedis;

import org.springframework.beans.factory.annotation.Autowired;

import redis.clients.jedis.JedisCluster;

public class JedisClientCluster implements JedisClient {

@Autowired

private JedisCluster jedisCluster;

public JedisCluster getJedisCluster() {

return jedisCluster;

}

public void setJedisCluster(JedisCluster jedisCluster) {

this.jedisCluster = jedisCluster;

}

@Override

public String set(String key, String value) {

return jedisCluster.set(key, value);

}

@Override

public String get(String key) {

return jedisCluster.get(key);

}

@Override

public Boolean exists(String key) {

return jedisCluster.exists(key);

}

@Override

public Long expire(String key, int seconds) {

return jedisCluster.expire(key, seconds);

}

@Override

public Long ttl(String key) {

return jedisCluster.ttl(key);

}

@Override

public Long incr(String key) {

return jedisCluster.incr(key);

}

@Override

public Long hset(String key, String field, String value) {

return jedisCluster.hset(key, field, value);

}

@Override

public String hget(String key, String field) {

return jedisCluster.hget(key, field);

}

@Override

public Long hdel(String key, String... field) {

return jedisCluster.hdel(key, field);

}

}

package cn.wlyf.common.jedis;

import org.springframework.beans.factory.annotation.Autowired;

import redis.clients.jedis.Jedis;

import redis.clients.jedis.JedisPool;

public class JedisClientPool implements JedisClient {

@Autowired

private JedisPool jedisPool;

public JedisPool getJedisPool() {

return jedisPool;

}

public void setJedisPool(JedisPool jedisPool) {

this.jedisPool = jedisPool;

}

@Override

public String set(String key, String value) {

Jedis jedis = jedisPool.getResource();

String result = jedis.set(key, value);

jedis.close();

return result;

}

@Override

public String get(String key) {

Jedis jedis = jedisPool.getResource();

String result = jedis.get(key);

jedis.close();

return result;

}

@Override

public Boolean exists(String key) {

Jedis jedis = jedisPool.getResource();

Boolean result = jedis.exists(key);

jedis.close();

return result;

}

@Override

public Long expire(String key, int seconds) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.expire(key, seconds);

jedis.close();

return result;

}

@Override

public Long ttl(String key) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.ttl(key);

jedis.close();

return result;

}

@Override

public Long incr(String key) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.incr(key);

jedis.close();

return result;

}

@Override

public Long hset(String key, String field, String value) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.hset(key, field, value);

jedis.close();

return result;

}

@Override

public String hget(String key, String field) {

Jedis jedis = jedisPool.getResource();

String result = jedis.hget(key, field);

jedis.close();

return result;

}

@Override

public Long hdel(String key, String... field) {

Jedis jedis = jedisPool.getResource();

Long result = jedis.hdel(key, field);

jedis.close();

return result;

}

}

package cn.wlyf.common.pojo;

import java.io.Serializable;

public class SearchItem implements Serializable{

private String id;

private String title;

private String sell\_point;

private long price;

private String image;

private String category\_name;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getSell\_point() {

return sell\_point;

}

public void setSell\_point(String sell\_point) {

this.sell\_point = sell\_point;

}

public long getPrice() {

return price;

}

public void setPrice(long price) {

this.price = price;

}

public String getImage() {

return image;

}

public void setImage(String image) {

this.image = image;

}

public String getCategory\_name() {

return category\_name;

}

public void setCategory\_name(String category\_name) {

this.category\_name = category\_name;

}

public String[] getImages() {

if(image != null && !"".equals(image)) {

String[] strings = image.split(",");

return strings;

}

return null;

}

@Override

public String toString() {

return "SearchItem [id=" + id + ", title=" + title + ", sell\_point=" + sell\_point + ", price=" + price

+ ", image=" + image + ", category\_name=" + category\_name + "]";

}

}

package cn.wlyf.common.pojo;

import java.io.Serializable;

import java.util.List;

public class SearchResult implements Serializable{

private long recordCount;

private int totalPages;

private List<SearchItem> itemList;

public long getRecordCount() {

return recordCount;

}

public void setRecordCount(long recordCount) {

this.recordCount = recordCount;

}

public int getTotalPages() {

return totalPages;

}

public void setTotalPages(int totalPages) {

this.totalPages = totalPages;

}

public List<SearchItem> getItemList() {

return itemList;

}

public void setItemList(List<SearchItem> itemList) {

this.itemList = itemList;

}

@Override

public String toString() {

return "SearchResult [recordCount=" + recordCount + ", totalPages=" + totalPages + "]";

}

}

package cn.wlyf.common.utils;

import java.io.Serializable;

import java.util.List;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

/\*\*

\* E3Result

\*

\*/

public class E3Result implements Serializable{

// 定义jackson对象

private static final ObjectMapper MAPPER = new ObjectMapper();

// 响应业务状态

private Integer status;

// 响应消息

private String msg;

// 响应中的数据

private Object data;

public static E3Result build(Integer status, String msg, Object data) {

return new E3Result(status, msg, data);

}

public static E3Result ok(Object data) {

return new E3Result(data);

}

public static E3Result ok() {

return new E3Result(null);

}

public E3Result() {

}

public static E3Result build(Integer status, String msg) {

return new E3Result(status, msg, null);

}

public E3Result(Integer status, String msg, Object data) {

this.status = status;

this.msg = msg;

this.data = data;

}

public E3Result(Object data) {

this.status = 200;

this.msg = "OK";

this.data = data;

}

// public Boolean isOK() {

// return this.status == 200;

// }

public Integer getStatus() {

return status;

}

public void setStatus(Integer status) {

this.status = status;

}

public String getMsg() {

return msg;

}

public void setMsg(String msg) {

this.msg = msg;

}

public Object getData() {

return data;

}

public void setData(Object data) {

this.data = data;

}

/\*\*

\* 将json结果集转化为TaotaoResult对象

\*

\* @param jsonData json数据

\* @param clazz TaotaoResult中的object类型

\* @return

\*/

public static E3Result formatToPojo(String jsonData, Class<?> clazz) {

try {

if (clazz == null) {

return MAPPER.readValue(jsonData, E3Result.class);

}

JsonNode jsonNode = MAPPER.readTree(jsonData);

JsonNode data = jsonNode.get("data");

Object obj = null;

if (clazz != null) {

if (data.isObject()) {

obj = MAPPER.readValue(data.traverse(), clazz);

} else if (data.isTextual()) {

obj = MAPPER.readValue(data.asText(), clazz);

}

}

return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);

} catch (Exception e) {

return null;

}

}

/\*\*

\* 没有object对象的转化

\*

\* @param json

\* @return

\*/

public static E3Result format(String json) {

try {

return MAPPER.readValue(json, E3Result.class);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* Object是集合转化

\*

\* @param jsonData json数据

\* @param clazz 集合中的类型

\* @return

\*/

public static E3Result formatToList(String jsonData, Class<?> clazz) {

try {

JsonNode jsonNode = MAPPER.readTree(jsonData);

JsonNode data = jsonNode.get("data");

Object obj = null;

if (data.isArray() && data.size() > 0) {

obj = MAPPER.readValue(data.traverse(),

MAPPER.getTypeFactory().constructCollectionType(List.class, clazz));

}

return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);

} catch (Exception e) {

return null;

}

}

}

package cn.wlyf.common.utils;

import org.csource.common.NameValuePair;

import org.csource.fastdfs.ClientGlobal;

import org.csource.fastdfs.StorageClient1;

import org.csource.fastdfs.StorageServer;

import org.csource.fastdfs.TrackerClient;

import org.csource.fastdfs.TrackerServer;

public class FastDFSClient {

private TrackerClient trackerClient = null;

private TrackerServer trackerServer = null;

private StorageServer storageServer = null;

private StorageClient1 storageClient = null;

public FastDFSClient(String conf) throws Exception {

if (conf.contains("classpath:")) {

conf = conf.replace("classpath:", this.getClass().getResource("/").getPath());

}

ClientGlobal.init(conf);

trackerClient = new TrackerClient();

trackerServer = trackerClient.getConnection();

storageServer = null;

storageClient = new StorageClient1(trackerServer, storageServer);

}

/\*\*

\* 上传文件方法

\* <p>Title: uploadFile</p>

\* <p>Description: </p>

\* @param fileName 文件全路径

\* @param extName 文件扩展名，不包含（.）

\* @param metas 文件扩展信息

\* @return

\* @throws Exception

\*/

public String uploadFile(String fileName, String extName, NameValuePair[] metas) throws Exception {

String result = storageClient.upload\_file1(fileName, extName, metas);

return result;

}

public String uploadFile(String fileName) throws Exception {

return uploadFile(fileName, null, null);

}

public String uploadFile(String fileName, String extName) throws Exception {

return uploadFile(fileName, extName, null);

}

/\*\*

\* 上传文件方法

\* <p>Title: uploadFile</p>

\* <p>Description: </p>

\* @param fileContent 文件的内容，字节数组

\* @param extName 文件扩展名

\* @param metas 文件扩展信息

\* @return

\* @throws Exception

\*/

public String uploadFile(byte[] fileContent, String extName, NameValuePair[] metas) throws Exception {

String result = storageClient.upload\_file1(fileContent, extName, metas);

return result;

}

public String uploadFile(byte[] fileContent) throws Exception {

return uploadFile(fileContent, null, null);

}

public String uploadFile(byte[] fileContent, String extName) throws Exception {

return uploadFile(fileContent, extName, null);

}

}

package cn.wlyf.common.utils;

import java.util.Random;

/\*\*

\* 各种id生成策略

\* <p>Title: IDUtils</p>

\* <p>Description: </p>

\* <p>Company: www.itcast.com</p>

\*

\*

\* @version 1.0

\*/

public class IDUtils {

/\*\*

\* 图片名生成

\*/

public static String genImageName() {

//取当前时间的长整形值包含毫秒

long millis = System.currentTimeMillis();

//long millis = System.nanoTime();

//加上三位随机数

Random random = new Random();

int end3 = random.nextInt(999);

//如果不足三位前面补0

String str = millis + String.format("%03d", end3);

return str;

}

/\*\*

\* id生成

\*/

public static long genItemId() {

//取当前时间的长整形值包含毫秒

long millis = System.currentTimeMillis();

//long millis = System.nanoTime();

//加上两位随机数

Random random = new Random();

int end2 = random.nextInt(99);

//如果不足两位前面补0

String str = millis + String.format("%02d", end2);

long id = new Long(str);

return id;

}

public static void main(String[] args) {

for(int i=0;i< 100;i++)

System.out.println(genItemId());

}

}

package cn.wlyf.common.utils;

import java.util.List;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.JavaType;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

/\*\*

\* 自定义响应结构

\*/

public class JsonUtils {

// 定义jackson对象

private static final ObjectMapper MAPPER = new ObjectMapper();

/\*\*

\* 将对象转换成json字符串。

\* <p>Title: pojoToJson</p>

\* <p>Description: </p>

\* @param data

\* @return

\*/

public static String objectToJson(Object data) {

try {

String string = MAPPER.writeValueAsString(data);

return string;

} catch (JsonProcessingException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 将json结果集转化为对象

\*

\* @param jsonData json数据

\* @param clazz 对象中的object类型

\* @return

\*/

public static <T> T jsonToPojo(String jsonData, Class<T> beanType) {

try {

T t = MAPPER.readValue(jsonData, beanType);

return t;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 将json数据转换成pojo对象list

\* <p>Title: jsonToList</p>

\* <p>Description: </p>

\* @param jsonData

\* @param beanType

\* @return

\*/

public static <T>List<T> jsonToList(String jsonData, Class<T> beanType) {

JavaType javaType = MAPPER.getTypeFactory().constructParametricType(List.class, beanType);

try {

List<T> list = MAPPER.readValue(jsonData, javaType);

return list;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

package cn.wlyf.content.service;

import java.util.List;

import cn.wlyf.common.pojo.EasyUITreeNode;

import cn.wlyf.common.utils.E3Result;

public interface ContentCategoryService {

List<EasyUITreeNode> getContentCatList(long parentId);

E3Result addContentCategory(long parentId,String name);

}

package cn.wlyf.content.service;

import java.util.List;

import cn.wlyf.common.utils.E3Result;

import cn.wlyf.pojo.TbContent;

public interface ContentService {

E3Result addContent(TbContent content);

List<TbContent> getContentListByCid(long cid);

}

package cn.wlyf.content.service.impl;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import cn.wlyf.common.pojo.EasyUITreeNode;

import cn.wlyf.common.utils.E3Result;

import cn.wlyf.content.service.ContentCategoryService;

import cn.wlyf.mapper.TbContentCategoryMapper;

import cn.wlyf.pojo.TbContentCategory;

import cn.wlyf.pojo.TbContentCategoryExample;

import cn.wlyf.pojo.TbContentCategoryExample.Criteria;

/\*\*

\* 内容分类管理Service

\*

\*

\*/

@Service

public class ContentCategoryServiceImpl implements ContentCategoryService {

@Autowired

private TbContentCategoryMapper contentCategoryMapper;

@Override

public List<EasyUITreeNode> getContentCatList(long parentId) {

//根据parentid查询子节点列表

TbContentCategoryExample example = new TbContentCategoryExample();

Criteria criteria = example.createCriteria();

//设置查询条件

criteria.andParentIdEqualTo(parentId);

//执行查询

List<TbContentCategory> catList = contentCategoryMapper.selectByExample(example);

//转换成EasyUITreeNode的列表

List<EasyUITreeNode> nodeList = new ArrayList<>();

for(TbContentCategory tbContentCategory : catList){

EasyUITreeNode node = new EasyUITreeNode();

node.setId(tbContentCategory.getId());

node.setText(tbContentCategory.getName());

node.setState(tbContentCategory.getIsParent()?"closed":"open");

//添加到列表

nodeList.add(node);

}

return nodeList;

}

@Override

public E3Result addContentCategory(long parentId, String name) {

// 创建一个tb\_content\_category表对应的pojo对象

TbContentCategory contentCategory = new TbContentCategory();

// 设置pojo属性

contentCategory.setParentId(parentId);

contentCategory.setName(name);

//1(正常),2(删除)

contentCategory.setStatus(1);

//默认排序就是1

contentCategory.setSortOrder(1);

//添加的新节点一定是叶子节点

contentCategory.setIsParent(false);

contentCategory.setCreated(new Date());

contentCategory.setUpdated(new Date());

// 插入到数据库中

contentCategoryMapper.insert(contentCategory);

// 判断父节点的isparent属性，如果不是true改为true

//根据parentId查询父节点

TbContentCategory parent = contentCategoryMapper.selectByPrimaryKey(parentId);

if(!parent.getIsParent()){

parent.setIsParent(true);

//更新到数据库中

contentCategoryMapper.updateByPrimaryKey(parent);

}

// 返回结果，返回E3Result，包含pojo

return E3Result.ok(contentCategory);

}

}

package cn.wlyf.content.service.impl;

import java.util.Date;

import java.util.List;

import org.apache.commons.lang3.StringUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import cn.wlyf.common.jedis.JedisClient;

import cn.wlyf.common.utils.E3Result;

import cn.wlyf.common.utils.JsonUtils;

import cn.wlyf.content.service.ContentService;

import cn.wlyf.mapper.TbContentMapper;

import cn.wlyf.pojo.TbContent;

import cn.wlyf.pojo.TbContentExample;

import cn.wlyf.pojo.TbContentExample.Criteria;

/\*\*

\*

\* 内容管理service

\*

\*/

@Service

public class ContentServiceImpl implements ContentService {

@Autowired

private TbContentMapper contentMapper;

@Autowired

private JedisClient jedisClient;

//在conf/resource.properties中可配置该参数

@Value("$(CONTENT\_LIST)")

private String CONTENT\_LIST;

@Override

public E3Result addContent(TbContent content) {

//将内容数据插入到内容表

content.setCreated(new Date());

content.setUpdated(new Date());

//插入到数据库

contentMapper.insert(content);

//缓存同步，删除缓存中对应的数据

jedisClient.hdel(CONTENT\_LIST, content.getCategoryId().toString());

return E3Result.ok();

}

@Override

public List<TbContent> getContentListByCid(long cid) {

//查询缓存

try {

//如果缓存中存在直接响应结果

String json = jedisClient.hget(CONTENT\_LIST, cid + "");

if (StringUtils.isNotBlank(json)) {

List<TbContent> list = JsonUtils.jsonToList(json, TbContent.class);

return list;

}

} catch (Exception e) {

e.printStackTrace();

}

//如果没有查询数据库

TbContentExample example = new TbContentExample();

Criteria criteria = example.createCriteria();

//设置查询条件

criteria.andCategoryIdEqualTo(cid);

//执行查询

List<TbContent> list = contentMapper.selectByExampleWithBLOBs(example);

//把结果添加到缓存

try {

jedisClient.hset(CONTENT\_LIST, cid + "",JsonUtils.objectToJson(list));

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

}

package cn.wlyf.item.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import cn.wlyf.item.pojo.Item;

import cn.wlyf.pojo.TbItem;

import cn.wlyf.pojo.TbItemDesc;

import cn.wlyf.service.ItemService;

/\*\*

\*详情页面展示Controller

\* <p>Title: ItemController</p>

\* <p>Description: </p>

\* <p>Company: www.itcast.cn</p>

\* @version 1.0

\*/

@Controller

public class ItemController {

@Autowired

private ItemService itemService;

@RequestMapping("/item/{itemId}")

public String showItemInfo(@PathVariable Long itemId, Model model) {

//调用服务取基本信息

TbItem tbItem = itemService.getItemById(itemId);

Item item = new Item(tbItem);

//取描述信息

TbItemDesc itemDesc = itemService.getItemDescById(itemId);

//把信息传递给页面

model.addAttribute("item", item);

model.addAttribute("itemDesc", itemDesc);

//返回逻辑视图

return "item";

}

}

package cn.wlyf.item.pojo;

import cn.wlyf.pojo.TbItem;

public class Item extends TbItem

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public Item(TbItem tbItem) {

this.setId(tbItem.getId());

this.setTitle(tbItem.getTitle());

this.setSellPoint(tbItem.getSellPoint());

this.setPrice(tbItem.getPrice());

this.setNum(tbItem.getNum());

this.setBarcode(tbItem.getBarcode());

this.setImage(tbItem.getImage());

this.setCid(tbItem.getCid());

this.setStatus(tbItem.getStatus());

this.setCreated(tbItem.getCreated());

this.setUpdated(tbItem.getUpdated());

}

public String[] getImages(){

String image2 = this.getImage();

if(image2 != null && !"".equals(image2)) {

String[] strings = image2.split(",");

return strings;

}

return null;

}

}

package cn.wlyf.handler.authentication;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpHandler;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import cn.wlyf.helpers.Base64;

/\*\*

\* Adds HTTP Basic authentication to a page. Users should provide an implementation of UsernamePasswordAuthenticator

\* to check the supplied credentials.

\* <p/>

\* See samples.authentication.SimplePasswordsExample in the src/tests directory for a really basic usage. To implement

\* a custom authenticator that performs background IO, see samples.authentication.AsyncPasswordsExample.

\*

\* @see PasswordAuthenticator

\* @see InMemoryPasswords

\*/

public class BasicAuthenticationHandler implements HttpHandler {

public static final String USERNAME = "user";

private static final String BASIC\_PREFIX = "Basic ";

private final String realm;

private final PasswordAuthenticator authenticator;

public BasicAuthenticationHandler(PasswordAuthenticator authenticator) {

this(authenticator, "Secure Area");

}

public BasicAuthenticationHandler(PasswordAuthenticator authenticator, String realm) {

this.realm = realm;

this.authenticator = authenticator;

}

@Override

public void handleHttpRequest(final HttpRequest request, final HttpResponse response, final HttpControl control) throws Exception {

String authHeader = request.header("Authorization");

if (authHeader == null) {

needAuthentication(response);

} else {

if (authHeader.startsWith(BASIC\_PREFIX)) {

String decoded = new String(Base64.decode(authHeader.substring(BASIC\_PREFIX.length())));

final String[] pair = decoded.split(":", 2);

if (pair.length == 2) {

final String username = pair[0];

final String password = pair[1];

PasswordAuthenticator.ResultCallback callback = new PasswordAuthenticator.ResultCallback() {

@Override

public void success() {

request.data(USERNAME, username);

control.nextHandler();

}

@Override

public void failure() {

needAuthentication(response);

}

};

authenticator.authenticate(request, username, password, callback, control);

} else {

needAuthentication(response);

}

}

}

}

private void needAuthentication(HttpResponse response) {

response.status(401)

.header("WWW-Authenticate", "Basic realm=\"" + realm + "\"")

.content("Need authentication")

.end();

}

}

package cn.wlyf.handler.authentication;

import cn.wlyf.HttpRequest;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.Executor;

/\*\*

\* Implementation of PasswordAuthenticator that verifies usernames and password from a prepopulated hashmap.

\*/

public class InMemoryPasswords implements PasswordAuthenticator {

private final Map<String, String> usernameToPasswords = new HashMap<String, String>();

public InMemoryPasswords add(String username, String password) {

usernameToPasswords.put(username, password);

return this;

}

@Override

public void authenticate(HttpRequest request, String username, String password, ResultCallback callback, Executor handlerExecutor) {

String expectedPassword = usernameToPasswords.get(username);

if (expectedPassword != null && password.equals(expectedPassword)) {

callback.success();

} else {

callback.failure();

}

}

}

package cn.wlyf.handler.authentication;

import cn.wlyf.HttpRequest;

import java.util.concurrent.Executor;

/\*\*

\* Provided to BasicAuthenticationHandler to verify the supplied username and password are valid.

\* <p/>

\* Implementations should check username/password are valid and call

\* ResultCallback.success() or ResultCallback.failure(). One of these methods must called - once and only once.

\* <p/>

\* If the result cannot be obtained automatically, the code should not block (as this will block the entire server).

\* Instead, the work should be offloaded to another thread/process, and the ResultCallback methods should be invoked

\* using the handlerExecutor when done.

\* <p/>

\* For simple cases, use InMemoryPasswords.

\* <p/>

\* See samples.authentication.SimplePasswordsExample in the src/tests directory for a really basic usage. To implement

\* a custom authenticator that performs background IO, see samples.authentication.AsyncPasswordsExample.

\*

\* @see BasicAuthenticationHandler

\* @see InMemoryPasswords

\*/

public interface PasswordAuthenticator {

void authenticate(HttpRequest request, String username, String password, ResultCallback callback, Executor handlerExecutor);

interface ResultCallback {

void success();

void failure();

}

}

package cn.wlyf.handler;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpHandler;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import java.io.IOException;

import java.io.InputStream;

import java.nio.ByteBuffer;

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.Executor;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public abstract class AbstractResourceHandler implements HttpHandler {

static {

Map<String, String> mimeTypes = new HashMap<String, String>();

mimeTypes.put("txt", "text/plain");

mimeTypes.put("css", "text/css");

mimeTypes.put("csv", "text/csv");

mimeTypes.put("htm", "text/html");

mimeTypes.put("html", "text/html");

mimeTypes.put("xml", "text/xml");

mimeTypes.put("js", "text/javascript"); // Technically it should be application/javascript (RFC 4329), but IE8 struggles with that

mimeTypes.put("xhtml", "application/xhtml+xml");

mimeTypes.put("json", "application/json");

mimeTypes.put("pdf", "application/pdf");

mimeTypes.put("zip", "application/zip");

mimeTypes.put("tar", "application/x-tar");

mimeTypes.put("gif", "image/gif");

mimeTypes.put("jpeg", "image/jpeg");

mimeTypes.put("jpg", "image/jpeg");

mimeTypes.put("tiff", "image/tiff");

mimeTypes.put("tif", "image/tiff");

mimeTypes.put("png", "image/png");

mimeTypes.put("swf", "application/x-shockwave-flash");

mimeTypes.put("svg", "image/svg+xml");

mimeTypes.put("ico", "image/vnd.microsoft.icon");

DEFAULT\_MIME\_TYPES = Collections.unmodifiableMap(mimeTypes);

}

private static final Pattern SINGLE\_BYTE\_RANGE = Pattern.compile("bytes=(\\d+)?-(\\d+)?");

public static final Map<String, String> DEFAULT\_MIME\_TYPES;

protected static final String DEFAULT\_WELCOME\_FILE\_NAME = "index.html";

protected final Executor ioThread;

protected final Map<String, String> mimeTypes;

protected String welcomeFileName;

protected DirectoryListingFormatter directoryListingFormatter;

protected final TemplateEngine templateEngine;

private boolean isDirectoryListingEnabled = false;

public AbstractResourceHandler(Executor ioThread, TemplateEngine templateEngine) {

this.ioThread = ioThread;

this.templateEngine = templateEngine;

this.mimeTypes = new HashMap<String, String>(DEFAULT\_MIME\_TYPES);

this.welcomeFileName = DEFAULT\_WELCOME\_FILE\_NAME;

}

public AbstractResourceHandler(Executor ioThread) {

this(ioThread, new StaticFile());

}

public AbstractResourceHandler addMimeType(String extension, String mimeType) {

mimeTypes.put(extension, mimeType);

return this;

}

public AbstractResourceHandler welcomeFile(String welcomeFile) {

this.welcomeFileName = welcomeFile;

return this;

}

public AbstractResourceHandler enableDirectoryListing(boolean isDirectoryListingEnabled) {

return enableDirectoryListing(isDirectoryListingEnabled, new DefaultDirectoryListingFormatter());

}

public AbstractResourceHandler enableDirectoryListing(boolean isDirectoryListingEnabled, DirectoryListingFormatter directoryListingFormatter) {

this.isDirectoryListingEnabled = isDirectoryListingEnabled;

this.directoryListingFormatter = directoryListingFormatter;

return this;

}

@Override

public void handleHttpRequest(final HttpRequest request, final HttpResponse response, final HttpControl control) throws Exception {

// Switch from web thead to IO thread, so we don't block web server when we access the filesystem.

ioThread.execute(createIOWorker(request, response, control));

}

protected void serve(final String mimeType,

final byte[] staticContents,

HttpControl control,

final HttpResponse response,

final HttpRequest request,

final String path) {

// Switch back from IO thread to web thread.

control.execute(new Runnable() {

@Override

public void run() {

// TODO: Check bytes read match expected encoding of mime-type

response.header("Content-Type", mimeType);

byte[] dynamicContents = templateEngine.process(staticContents, path, request.data(TemplateEngine.TEMPLATE\_CONTEXT));

ByteBuffer contents = ByteBuffer.wrap(dynamicContents);

if (maybeServeRange(request, contents, response)) {

return;

}

// TODO: Don't read all into memory, instead use zero-copy.

response.header("Content-Length", contents.remaining())

.content(contents)

.end();

}

});

}

private boolean maybeServeRange(HttpRequest request, ByteBuffer contents, HttpResponse response) {

String range = request.header("Range");

if (null == range) {

return false;

}

Matcher matcher = SINGLE\_BYTE\_RANGE.matcher(range);

if (!matcher.matches()) {

return false;

}

String startString = matcher.group(1);

String endString = matcher.group(2);

if (null != startString && null != endString) {

int start = Integer.parseInt(startString);

int end = Integer.parseInt(endString);

if (start <= end) {

serveRange(start,

Math.min(contents.remaining() - 1, end),

contents,

response);

return true;

}

} else if (null != startString) {

serveRange(Integer.parseInt(startString),

contents.remaining() - 1,

contents,

response);

return true;

} else if (null != endString) {

int end = Integer.parseInt(endString);

serveRange(contents.remaining() - end,

contents.remaining() - 1,

contents,

response);

return true;

}

return false;

}

protected void serveRange(int start, int end, ByteBuffer contents, HttpResponse response) {

if (start > contents.remaining()) {

response.status(416).header("Content-Range", "bytes \*/" + contents.remaining()).end();

return;

}

response.status(206)

.header("Content-Length", end - start + 1) // since its inclusive

.header("Content-Range",

"bytes " + start + "-" + end + "/" + contents.remaining());

contents.limit(contents.position() + end + 1)

.position(contents.position() + start);

response.content(contents).end();

}

protected abstract IOWorker createIOWorker(HttpRequest request,

HttpResponse response,

HttpControl control);

/\*\*

\* All IO is performed by this worker on a separate thread, so we never block the HttpHandler.

\*/

protected abstract class IOWorker implements Runnable {

protected String path;

private final HttpRequest request;

protected final HttpResponse response;

protected final HttpControl control;

protected IOWorker(String path, HttpRequest request, HttpResponse response, HttpControl control) {

this.path = path;

this.request = request;

this.response = response;

this.control = control;

}

protected void notFound() {

// Switch back from IO thread to web thread.

control.execute(new Runnable() {

@Override

public void run() {

control.nextHandler();

}

});

}

protected void error(final IOException exception) {

// Switch back from IO thread to web thread.

control.execute(new Runnable() {

@Override

public void run() {

response.error(exception);

}

});

}

@Override

public void run() {

String pathWithQuery = path;

path = withoutQuery(path);

// TODO: Cache

try {

byte[] content = null;

if (!exists()) {

notFound();

return;

}

if (isDirectory()) {

// Assumes if path has been changed since the original request,

// its current value with a trailing slash will still resolve properly

if (!path.endsWith("/")) {

response.status(301).header("Location", path + "/" + extractQuery(pathWithQuery)).end();

return;

} else if ((content = welcomeBytes()) != null) {

serve(guessMimeType(welcomeFileName), content, control, response, request, path);

return;

} else if (isDirectoryListingEnabled && (content = directoryListingBytes()) != null) {

serve(guessMimeType(".html"), content, control, response, request, path);

return;

}

// TODO: Do something other than 404 if directory listing is disabled

} else if ((content = fileBytes()) != null) {

serve(guessMimeType(path), content, control, response, request, path);

return;

} else if ((content = welcomeBytes()) != null) {

serve(guessMimeType(welcomeFileName), content, control, response, request, path);

return;

}

notFound();

} catch (IOException e) {

error(e);

}

}

protected abstract boolean exists() throws IOException;

protected abstract boolean isDirectory() throws IOException;

protected abstract byte[] fileBytes() throws IOException;

protected abstract byte[] welcomeBytes() throws IOException;

protected abstract byte[] directoryListingBytes() throws IOException;

protected byte[] read(int length, InputStream in) throws IOException {

byte[] data = new byte[length];

try {

int read = 0;

while (read < length) {

int more = in.read(data, read, data.length - read);

if (more == -1) {

break;

} else {

read += more;

}

}

} finally {

in.close();

}

return data;

}

// TODO: Don't respond with a mime type that violates the request's Accept header

private String guessMimeType(String path) {

int lastDot = path.lastIndexOf('.');

if (lastDot == -1) {

return null;

}

String extension = path.substring(lastDot + 1).toLowerCase();

String mimeType = mimeTypes.get(extension);

if (mimeType == null) {

return null;

}

if (mimeType.startsWith("text/") && response.charset() != null) {

mimeType += "; charset=" + response.charset().name();

}

return mimeType;

}

protected String withoutQuery(String path) {

int queryStart = path.indexOf('?');

if (queryStart > -1) {

path = path.substring(0, queryStart);

}

return path;

}

protected String extractQuery(String path) {

int queryStart = path.indexOf('?');

if (queryStart > -1) {

return path.substring(queryStart);

}

return "";

}

}

}

package cn.wlyf.handler;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpHandler;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import java.util.Timer;

import java.util.TimerTask;

import java.util.concurrent.Executor;

/\*\*

\* Wraps a standard HttpHandler, and will introduce an artificial delay. Useful for testing

\* how things will behave when they are slow.

\*/

public class DelayedHttpHandler implements HttpHandler {

private final Executor executor;

private Timer timer;

private final long delayInMillis;

private final HttpHandler handler;

public DelayedHttpHandler(Executor executor, long delayInMillis, HttpHandler handler) {

this.delayInMillis = delayInMillis;

this.handler = handler;

timer = new Timer();

this.executor = executor;

}

@Override

public void handleHttpRequest(final HttpRequest request, final HttpResponse response, final HttpControl control) throws Exception {

timer.schedule(new TimerTask() {

@Override

public void run() {

executor.execute(new Runnable() {

@Override

public void run() {

try {

handler.handleHttpRequest(request, response, control);

} catch (Exception e) {

// TODO

e.printStackTrace();

}

}

});

}

}, delayInMillis);

}

}

package cn.wlyf.handler;

/\*\*

\* Abstract interface for template engines. It can be passed as an argument to the constructors of

\* {@link StaticFileHandler} and {@link EmbeddedResourceHandler}.

\* <p/>

\* for implementations.

\*/

public interface TemplateEngine {

String TEMPLATE\_CONTEXT = "TEMPLATE\_CONTEXT";

/\*\*

\* Renders a template.

\* <p/>

\* Most template engines merge a {@code templateContext} with a template to produce output. What constitutes a valid

\* context is template-engine specific.

\* <p/>

\* Rendering happens on your main thread, so it is important that this method is not IO-bound. For example,

\* rendering a template using a context that is some sort of <a href="http://en.wikipedia.org/wiki/Active\_record\_pattern">active record</a>

\* that makes trips to the database will block Webbit preventing it from handling other requests. It's important to

\* make sure any remote data is pre-fetched.

\* <p/>

\* Webbit will pass the request data value keyed with {@link #TEMPLATE\_CONTEXT} as the {@code templateContext} argument.

\* It's the programmer's responsibility to make sure the data value is set before the template is rendered,

\* i.e. before the {@link StaticFileHandler} or {@link EmbeddedResourceHandler} handler instance handles a request.

\*

\* @param length number of bytes in the template

\* @param template the template source

\* @param templatePath the path the template is read from. Allows implementations to cache compiled templates.

\* @param templateContext object to merge into the template

\* @return a rendered template

\* @see cn.wlyf.HttpRequest#data(String)

\* @see cn.wlyf.HttpRequest#data(String, Object)

\*/

byte[] process(byte[] template, String templatePath, Object templateContext) throws RuntimeException;

}

package cn.wlyf.handler;

import org.jboss.netty.handler.codec.http.HttpHeaders;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpHandler;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import java.util.Date;

/\*\*

\* Handler that sets the HTTP 'Server' response header.

\*/

public class DateHeaderHandler implements HttpHandler {

@Override

public void handleHttpRequest(HttpRequest request, HttpResponse response, HttpControl control) throws Exception {

if (!response.containsHeader(HttpHeaders.Names.DATE)) {

response.header(HttpHeaders.Names.DATE, new Date());

}

control.nextHandler();

}

}

package cn.wlyf;

import java.util.Map;

import java.util.Set;

/\*\*

\* Objects implementing this interface can have arbitrary named values associated with

\* them, making it easy to pass data around an application.

\*

\*

\*/

public interface DataHolder {

/\*\*

\* Arbitrary data that can be stored for the lifetime of the connection.

\*/

Map<String, Object> data();

/\*\*

\* Retrieve data value by key.

\*

\* @see #data()

\*/

Object data(String key);

/\*\*

\* Store data value by key.

\*

\* @see #data()

\*/

DataHolder data(String key, Object value);

/\*\*

\* List data keys.

\*

\* @see #data()

\*/

Set<String> dataKeys();

}

package cn.wlyf;

import java.io.InputStream;

/\*\*

\* <p>Configures an event based webserver.</p>

\* <p/>

\* <p>To create an instance, use {@link WebServers#createWebServer(int)}.</p>

\* <p/>

\* <p>As with many of the interfaces in webbitserver, setter style methods return a

\* reference to this, to allow for simple initialization using method chaining.</p>

\* <p/>

\* <h2>Hello World Example</h2>

\* <pre>

\* class HelloWorldHandler implements HttpHandler {

\* void handleHttpRequest(HttpRequest request, HttpResponse response, HttpControl control) {

\* response.header("Content-Type", "text/html")

\* .content("Hello World")

\* .end();

\* }

\* }

\* WebServer webServer = WebServers.createWebServer(8080)

\* .add(new HelloWorldHandler());

\* webServer.start();

\* print("Point your browser to " + webServer.getUri());

\* </pre>

\* <p/>

\* <h2>Serving Static Files</h2>

\* <pre>

\* WebServer webServer = WebServers.createWebServer(8080)

\* .add(new StaticFileHandler("./wwwdata"));

\* webServer.start();

\* </pre>

\*

\*

\* @see WebServers

\* @see HttpHandler

\* @see WebSocketConnection

\* @see EventSourceConnection

\*/

public interface WebServer extends Endpoint<WebServer> {

/\*\*

\* Add an HttpHandler. When a request comes in the first HttpHandler will be invoked.

\* The HttpHandler should either handle the request, or pass the request onto the

\* next HttpHandler (using {@link HttpControl#nextHandler()}). This is repeated

\* until a HttpHandler returns a response. If there are no remaining handlers, the

\* webserver shall return 404 NOT FOUND to the browser.

\* <p/>

\* HttpHandlers are attempted in the order in which they are added to the WebServer.

\*

\* @see HttpHandler

\*/

WebServer add(HttpHandler handler);

/\*\*

\* Add an HttpHandler that will only respond to a certain path (e.g "/some/page").

\* <p/>

\* This is shortcut for {@code add(newPathMatchHandler(path, handler))}.

\*

\* @see HttpHandler

\* @see #add(HttpHandler)

\* @see cn.wlyf.handler.PathMatchHandler

\*/

WebServer add(String path, HttpHandler handler);

/\*\*

\* Add a WebSocketHandler for dealing with WebSockets.

\* <p/>

\* This is shortcut for {@code add(new PathMatchHandler(path, newHttpToWebSocketHandler(handler)))}.

\*

\* @see WebSocketHandler

\* @see HttpHandler

\* @see #add(HttpHandler)

\* @see cn.wlyf.handler.HttpToWebSocketHandler

\* @see cn.wlyf.handler.PathMatchHandler

\*/

WebServer add(String path, WebSocketHandler handler);

/\*\*

\* Add a WebSocketHandler for dealing with WebSockets.

\* <p/>

\* This is shortcut for {@code add(new PathMatchHandler(path, newHttpToEventSourceHandler(handler)))}.

\*

\* @see HttpHandler

\* @see #add(HttpHandler)

\* @see cn.wlyf.handler.HttpToEventSourceHandler

\* @see cn.wlyf.handler.PathMatchHandler

\*/

WebServer add(String path, EventSourceHandler handler);

/\*\*

\* Get base port that webserver is serving on.

\*/

int getPort();

/\*\*

\* Number of milliseconds before a stale HTTP keep-alive connection is closed by the server. A HTTP connection

\* is considered stale if it remains open without sending more data within the timeout window.

\*/

WebServer staleConnectionTimeout(long millis);

/\*\*

\* Setup SSL/TLS handler

\*

\* @param keyStore Keystore InputStream

\* @param storePass Store password

\* @param keyPass Key password

\* @return current WebServer instance

\* @throws cn.wlyf.WebbitException

\* A problem loading the keystore

\*/

WebServer setupSsl(InputStream keyStore, String storePass, String keyPass) throws WebbitException;

}

package cn.wlyf;

import java.net.HttpCookie;

import java.net.SocketAddress;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\* Retrieves information about inbound HTTP request.

\*

\*

\* @see HttpHandler

\* @see HttpResponse

\*/

public interface HttpRequest extends DataHolder {

public String COOKIE\_HEADER = "Cookie";

String uri();

/\*\*

\* Modify uri

\*

\* @param uri new uri

\*/

HttpRequest uri(String uri);

/\*\*

\* Retrieve the value single HTTP header.

\* <p/>

\* If the header is not found, null is returned.

\* <p/>

\* If there are multiple headers with the same name, it will return one of them, but it is not

\* defined which one. Instead, use {@link #headers(String)}.

\*/

String header(String name);

/\*\*

\* Retrieve all values for an HTTP header. If no values are found, an empty List is returned.

\*/

List<String> headers(String name);

/\*\*

\* Whether a specific HTTP header was present in the request.

\*/

boolean hasHeader(String name);

/\*\*

\* @return all inbound cookies

\*/

List<HttpCookie> cookies();

/\*\*

\* Get a cookie with a specific name

\*

\* @param name cookie name

\* @return cookie with that name

\*/

HttpCookie cookie(String name);

/\*\*

\* Get query parameter value.

\*

\* @param key parameter name

\* @return the value of the parameter

\* @see #queryParams(String)

\*/

String queryParam(String key);

/\*\*

\* Get all query parameter values.

\*

\* @param key parameter name

\* @return the values of the parameter

\* @see #queryParam(String)

\*/

List<String> queryParams(String key);

/\*\*

\* List all query parameter keys.

\*

\* @see #queryParam(String)

\*/

Set<String> queryParamKeys();

/\*\*

\* Get post parameter value.

\*

\* @param key parameter name

\* @return the value of the parameter

\* @see #postParams(String)

\*/

String postParam(String key);

/\*\*

\* Get all post parameter values.

\*

\* @param key parameter name

\* @return the values of the parameter

\* @see #postParam(String)

\*/

List<String> postParams(String key);

/\*\*

\* List all post parameter keys.

\*

\* @see #postParam(String)

\*/

Set<String> postParamKeys();

/\*\*

\* Get the value of named cookie

\* @param name cookie name

\* @return cookie value, or null if the cookie does not exist.

\*/

String cookieValue(String name);

/\*\*

\* Returns all headers sent from client.

\*/

List<Map.Entry<String, String>> allHeaders();

/\*\*

\* HTTP method (e.g. "GET" or "POST")

\*/

String method();

/\*\*

\* The body

\*/

String body();

/\*\*

\* The body's byte array

\*/

byte[] bodyAsBytes();

@Override

HttpRequest data(String key, Object value); // Override DataHolder to provide more specific return type.

/\*\*

\* Remote address of connection (i.e. the host of the client).

\*/

SocketAddress remoteAddress();

/\*\*

\* A unique identifier for this request. This should be treated as an opaque object,

\* that can be used to track the lifecycle of a request.

\*/

Object id();

/\*\*

\* Timestamp (millis since epoch) of when this request was first received by the server.

\*/

long timestamp();

}

package cn.wlyf.helpers;

public class XssCharacterEscaper {

/\*\*

\* Follows escaping rules from

\*

\* <a href="https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet#RULE\_.231\_-\_HTML\_Escape\_Before\_Inserting\_Untrusted\_Data\_into\_HTML\_Element\_Content">the OWASP</a>.

\*

\* @param input String to sanitize.

\* @return XSS-safe version of input.

\*

\*/

public static String escape(String input) {

StringBuilder builder = new StringBuilder(input.length());

for (int i = 0; i < input.length(); ++i) {

char original = input.charAt(i);

switch (original) {

case '&':

builder.append("&amp;");

break;

case '<':

builder.append("&lt;");

break;

case '>':

builder.append("&gt;");

break;

case '"':

builder.append("&quot;");

break;

case '\'':

builder.append("&#x27;");

break;

case '/':

builder.append("&#x2F;");

break;

default:

builder.append(original);

break;

}

}

return builder.toString();

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.ChannelFuture;

import org.jboss.netty.channel.ChannelFutureListener;

import org.jboss.netty.channel.ChannelHandlerContext;

import cn.wlyf.HttpConnection;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.Executor;

public abstract class AbstractHttpConnection implements HttpConnection {

private final Executor executor;

private final NettyHttpRequest nettyHttpRequest;

private final ChannelHandlerContext ctx;

public AbstractHttpConnection(ChannelHandlerContext ctx, NettyHttpRequest nettyHttpRequest, Executor executor) {

this.ctx = ctx;

this.nettyHttpRequest = nettyHttpRequest;

this.executor = executor;

}

protected ChannelFuture writeMessage(Object message) {

final ChannelFuture write = ctx.getChannel().write(message);

write.addListener(ChannelFutureListener.CLOSE\_ON\_FAILURE);

return write;

}

protected void closeChannel() {

ctx.getChannel().write(ChannelBuffers.EMPTY\_BUFFER).addListener(ChannelFutureListener.CLOSE);

}

protected void putData(String key, Object value) {

data().put(key, value);

}

@Override

public NettyHttpRequest httpRequest() {

return nettyHttpRequest;

}

@Override

public Map<String, Object> data() {

return nettyHttpRequest.data();

}

@Override

public Object data(String key) {

return data().get(key);

}

@Override

public Set<String> dataKeys() {

return data().keySet();

}

@Override

public Executor handlerExecutor() {

return executor;

}

@Override

public void execute(Runnable command) {

handlerExecutor().execute(command);

}

}

package cn.wlyf.netty;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelStateEvent;

import org.jboss.netty.channel.ExceptionEvent;

import cn.wlyf.WebbitException;

import java.nio.channels.ClosedChannelException;

import java.util.concurrent.Executor;

abstract class ConnectionHelper {

protected final Executor executor;

protected final Thread.UncaughtExceptionHandler exceptionHandler;

private final Thread.UncaughtExceptionHandler ioExceptionHandler;

public ConnectionHelper(Executor executor, Thread.UncaughtExceptionHandler exceptionHandler, Thread.UncaughtExceptionHandler ioExceptionHandler) {

this.ioExceptionHandler = ioExceptionHandler;

this.executor = executor;

this.exceptionHandler = exceptionHandler;

}

public void fireOnClose(final ChannelStateEvent e) {

final Thread thread = Thread.currentThread();

final Thread.UncaughtExceptionHandler uncaughtExceptionHandler = webbitExceptionWrappingExceptionHandler(e.getChannel());

executor.execute(new Runnable() {

@Override

public void run() {

try {

fireOnClose();

} catch (Throwable t) {

uncaughtExceptionHandler.uncaughtException(thread, t);

}

}

});

}

public void fireConnectionException(final ExceptionEvent e) {

if (e.getCause() instanceof ClosedChannelException) {

e.getChannel().close();

} else {

final Thread thread = Thread.currentThread();

executor.execute(new Runnable() {

@Override

public void run() {

ioExceptionHandler.uncaughtException(thread, WebbitException.fromExceptionEvent(e));

}

});

}

}

protected abstract void fireOnClose() throws Throwable;

// Uncaught exception handler including the connection for context.

protected Thread.UncaughtExceptionHandler webbitExceptionWrappingExceptionHandler(final Channel channel) {

return new Thread.UncaughtExceptionHandler() {

@Override

public void uncaughtException(Thread t, Throwable e) {

exceptionHandler.uncaughtException(t, WebbitException.fromException(e, channel));

}

};

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import cn.wlyf.WebSocketHandler;

import cn.wlyf.helpers.UTF8Exception;

import cn.wlyf.helpers.UTF8Output;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.Executor;

public class DecodingHybiFrame {

private final int opcode;

private final UTF8Output utf8Output;

private List<ChannelBuffer> fragments = new ArrayList<ChannelBuffer>();

private int length;

public DecodingHybiFrame(int opcode, UTF8Output utf8Output, ChannelBuffer fragment) throws UTF8Exception {

this.opcode = opcode;

this.utf8Output = utf8Output;

append(fragment);

}

public void append(ChannelBuffer fragment) throws UTF8Exception {

length += fragment.readableBytes();

if (opcode == Opcodes.OPCODE\_TEXT) {

utf8Output.write(fragment.array());

} else {

fragments.add(fragment);

}

}

private byte[] messageBytes() {

byte[] result = new byte[length];

int offset = 0;

for (ChannelBuffer fragment : fragments) {

byte[] array = fragment.array();

System.arraycopy(array, 0, result, offset, array.length);

offset += array.length;

}

return result;

}

public void dispatchMessage(final WebSocketHandler handler, final NettyWebSocketConnection connection, final Executor executor, final Thread.UncaughtExceptionHandler exceptionHandler) throws UTF8Exception {

switch (opcode) {

case Opcodes.OPCODE\_TEXT: {

final String messageValue = utf8Output.getStringAndRecycle();

executor.execute(new CatchingRunnable(exceptionHandler) {

@Override

protected void go() throws Throwable {

handler.onMessage(connection, messageValue);

}

});

return;

}

case Opcodes.OPCODE\_BINARY: {

final byte[] bytes = messageBytes();

executor.execute(new CatchingRunnable(exceptionHandler) {

@Override

public void go() throws Throwable {

handler.onMessage(connection, bytes);

}

});

return;

}

case Opcodes.OPCODE\_PING: {

final byte[] bytes = messageBytes();

executor.execute(new CatchingRunnable(exceptionHandler) {

@Override

protected void go() throws Throwable {

handler.onPing(connection, bytes);

}

});

return;

}

case Opcodes.OPCODE\_PONG: {

final byte[] bytes = messageBytes();

executor.execute(new CatchingRunnable(exceptionHandler) {

@Override

protected void go() throws Throwable {

handler.onPong(connection, bytes);

}

});

return;

}

default:

throw new IllegalStateException("Unexpected opcode:" + opcode);

}

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.handler.codec.frame.TooLongFrameException;

import static cn.wlyf.netty.HybiWebSocketFrameDecoder.applyMask;

public class EncodingHybiFrame {

private final int opcode;

private final boolean fin;

private final int rsv;

private byte[] maskingKey;

private final ChannelBuffer data;

public EncodingHybiFrame(int opcode, boolean fin, int rsv, byte[] maskingKey, ChannelBuffer fragment) {

this.opcode = opcode;

this.fin = fin;

this.rsv = rsv;

this.maskingKey = maskingKey;

this.data = fragment;

}

public ChannelBuffer encode() throws TooLongFrameException {

int b0 = 0;

if (fin) {

b0 |= (1 << 7);

}

b0 |= (rsv % 8) << 4;

b0 |= opcode % 128;

int b1 = maskingKey != null ? 0x80 : 0x00;

int headerLength = maskingKey != null ? 6 : 2;

ChannelBuffer header;

int length = data.readableBytes();

if (opcode == Opcodes.OPCODE\_PING && length > 125) {

throw new TooLongFrameException("invalid payload for PING (payload length must be <= 125, was " + length);

}

if (length <= 125) {

b1 |= length & 0x7F;

header = createBuffer(headerLength + length);

header.writeByte(b0);

header.writeByte(b1);

} else if (length <= 0xFFFF) {

b1 |= 126;

headerLength += 2;

header = createBuffer(headerLength + length);

header.writeByte(b0);

header.writeByte(b1);

header.writeByte((length >>> 8) & 0xFF);

header.writeByte((length) & 0xFF);

} else {

b1 |= 127;

headerLength += 8;

header = createBuffer(headerLength + length);

header.writeByte(b0);

header.writeByte(b1);

header.writeLong(length);

}

if (maskingKey != null) {

header.writeBytes(maskingKey);

applyMask(data, maskingKey);

}

return ChannelBuffers.wrappedBuffer(header, data);

}

private ChannelBuffer createBuffer(int length) {

return ChannelBuffers.buffer(length);

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.channel.ChannelPipeline;

import org.jboss.netty.handler.codec.frame.FrameDecoder;

import org.jboss.netty.util.CharsetUtil;

import java.util.concurrent.Executor;

/\*\*

\* Checks the received {@link org.jboss.netty.buffer.ChannelBuffer

\* ChannelBuffer}s for Flash policy file requests.

\* <p/>

\* <p>

\* If this decoder detects a Flash policy file request it adds a

\* {@link FlashPolicyFileHandler} to the

\* {@link org.jboss.netty.channel.ChannelPipeline ChannelPipeline} and removes

\* itself from the pipeline. If a Flash policy file request is not detected in

\* the first 23 bytes of the buffer, the decoder removes itself from the

\* pipeline.

\* <p>

\* <p/>

\* <p>

\* This implementation is based on the

\* "replacing a decoder with another decoder in a pipeline" section of the

\* {@link org.jboss.netty.handler.codec.frame.FrameDecoder FrameDecoder}

\* documentation.

\* </p>

\*/

public class FlashPolicyFileDecoder extends FrameDecoder {

private static final ChannelBuffer FLASH\_POLICY\_REQUEST = ChannelBuffers

.copiedBuffer("<policy-file-request/>\0", CharsetUtil.US\_ASCII);

private final Executor executor;

private final Thread.UncaughtExceptionHandler exceptionHandler;

private final Thread.UncaughtExceptionHandler ioExceptionHandler;

private final int publicPort;

public FlashPolicyFileDecoder(Executor executor, Thread.UncaughtExceptionHandler exceptionHandler, Thread.UncaughtExceptionHandler ioExceptionHandler, int publicPort) {

super(true);

this.publicPort = publicPort;

this.executor = executor;

this.exceptionHandler = exceptionHandler;

this.ioExceptionHandler = ioExceptionHandler;

}

@Override

protected Object decode(ChannelHandlerContext ctx, Channel channel, ChannelBuffer buffer) throws Exception {

// Will use the first 23 bytes to detect the policy file request.

if (buffer.readableBytes() >= 23) {

ChannelPipeline p = ctx.getPipeline();

ChannelBuffer firstMessage = buffer.readBytes(23);

if (FLASH\_POLICY\_REQUEST.equals(firstMessage)) {

p.addAfter("flashpolicydecoder", "flashpolicyhandler",

new FlashPolicyFileHandler(executor, exceptionHandler, ioExceptionHandler, this.publicPort));

}

p.remove(this);

if (buffer.readable()) {

return new Object[]{firstMessage, buffer.readBytes(buffer.readableBytes())};

} else {

return firstMessage;

}

}

// Forward the current buffer as is to handlers.

return buffer.readBytes(buffer.readableBytes());

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelFuture;

import org.jboss.netty.channel.ChannelFutureListener;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.channel.ExceptionEvent;

import org.jboss.netty.channel.MessageEvent;

import org.jboss.netty.channel.SimpleChannelUpstreamHandler;

import org.jboss.netty.util.CharsetUtil;

import java.util.concurrent.Executor;

/\*\*

\* Responds with a Flash socket policy file.

\* <p/>

\*/

public class FlashPolicyFileHandler extends SimpleChannelUpstreamHandler {

private final int publicPort;

private final ConnectionHelper connectionHelper;

public FlashPolicyFileHandler(Executor executor, Thread.UncaughtExceptionHandler exceptionHandler, Thread.UncaughtExceptionHandler ioExceptionHandler, int publicPort) {

this.publicPort = publicPort;

this.connectionHelper = new ConnectionHelper(executor, exceptionHandler, ioExceptionHandler) {

@Override

protected void fireOnClose() throws Exception {

throw new UnsupportedOperationException();

}

};

}

@Override

public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws Exception {

Channel ch = e.getChannel();

ChannelBuffer response = getPolicyFileContents();

ChannelFuture future = ch.write(response);

future.addListener(ChannelFutureListener.CLOSE);

ctx.getPipeline().remove(this);

}

private ChannelBuffer getPolicyFileContents() throws Exception {

return ChannelBuffers.copiedBuffer(

"<?xml version=\"1.0\"?>\r\n"

+ "<!DOCTYPE cross-domain-policy SYSTEM \"/xml/dtds/cross-domain-policy.dtd\">\r\n"

+ "<cross-domain-policy>\r\n"

+ " <site-control permitted-cross-domain-policies=\"master-only\"/>\r\n"

+ " <allow-access-from domain=\"\*\" to-ports=\"" + this.publicPort + "\" />\r\n"

+ "</cross-domain-policy>\r\n",

CharsetUtil.US\_ASCII);

}

@Override

public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e) throws Exception {

connectionHelper.fireConnectionException(e);

}

}

package cn.wlyf.netty;

import org.jboss.netty.channel.ChannelHandler;

import org.jboss.netty.handler.codec.http.HttpHeaders;

import org.jboss.netty.handler.codec.http.HttpRequest;

import org.jboss.netty.handler.codec.http.HttpResponse;

import org.jboss.netty.handler.codec.http.HttpResponseStatus;

import org.jboss.netty.handler.codec.http.websocket.WebSocketFrameDecoder;

import org.jboss.netty.handler.codec.http.websocket.WebSocketFrameEncoder;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.CONNECTION;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.ORIGIN;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.UPGRADE;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.WEBSOCKET\_LOCATION;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.WEBSOCKET\_ORIGIN;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.WEBSOCKET\_PROTOCOL;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Values.WEBSOCKET;

public class Hixie75 implements WebSocketVersion {

private final HttpRequest req;

private final HttpResponse res;

public Hixie75(HttpRequest req, HttpResponse res) {

this.req = req;

this.res = res;

}

@Override

public boolean matches() {

return false;

}

@Override

public void prepareHandshakeResponse(NettyWebSocketConnection webSocketConnection) {

webSocketConnection.setVersion("HIXIE-75");

res.setStatus(new HttpResponseStatus(101, "Web Socket Protocol Handshake"));

res.addHeader(UPGRADE, WEBSOCKET);

res.addHeader(CONNECTION, HttpHeaders.Values.UPGRADE);

String origin = req.getHeader(ORIGIN);

if (origin != null) {

res.addHeader(WEBSOCKET\_ORIGIN, origin);

}

res.addHeader(WEBSOCKET\_LOCATION, getWebSocketLocation(req));

String protocol = req.getHeader(WEBSOCKET\_PROTOCOL);

if (protocol != null) {

res.addHeader(WEBSOCKET\_PROTOCOL, protocol);

}

}

@Override

public ChannelHandler createDecoder() {

return new WebSocketFrameDecoder();

}

@Override

public ChannelHandler createEncoder() {

return new WebSocketFrameEncoder();

}

private String getWebSocketLocation(HttpRequest req) {

return getWebSocketProtocol(req) + req.getHeader(HttpHeaders.Names.HOST) + req.getUri();

}

private String getWebSocketProtocol(HttpRequest req) {

if(req.getHeader(HttpHeaders.Names.ORIGIN).matches("(?s)https://.\*")) { return "wss://"; } else { return "ws://"; }

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.ChannelHandler;

import org.jboss.netty.handler.codec.http.HttpHeaders;

import org.jboss.netty.handler.codec.http.HttpRequest;

import org.jboss.netty.handler.codec.http.HttpResponse;

import org.jboss.netty.handler.codec.http.HttpResponseStatus;

import org.jboss.netty.handler.codec.http.websocket.WebSocketFrameDecoder;

import org.jboss.netty.handler.codec.http.websocket.WebSocketFrameEncoder;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.CONNECTION;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.ORIGIN;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.SEC\_WEBSOCKET\_KEY1;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.SEC\_WEBSOCKET\_KEY2;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.SEC\_WEBSOCKET\_LOCATION;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.SEC\_WEBSOCKET\_ORIGIN;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.SEC\_WEBSOCKET\_PROTOCOL;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Names.UPGRADE;

import static org.jboss.netty.handler.codec.http.HttpHeaders.Values.WEBSOCKET;

public class Hixie76 implements WebSocketVersion {

private static final MessageDigest MD5;

static {

try {

MD5 = MessageDigest.getInstance("MD5");

} catch (NoSuchAlgorithmException e) {

throw new InternalError("MD5 not supported on this platform");

}

}

private final HttpRequest req;

private final HttpResponse res;

public Hixie76(HttpRequest req, HttpResponse res) {

this.req = req;

this.res = res;

}

@Override

public boolean matches() {

return req.containsHeader(SEC\_WEBSOCKET\_KEY1) && req.containsHeader(SEC\_WEBSOCKET\_KEY2);

}

@Override

public void prepareHandshakeResponse(NettyWebSocketConnection webSocketConnection) {

webSocketConnection.setVersion("HIXIE-76");

res.setStatus(new HttpResponseStatus(101, "Web Socket Protocol Handshake"));

res.addHeader(UPGRADE, WEBSOCKET);

res.addHeader(CONNECTION, UPGRADE);

res.addHeader(SEC\_WEBSOCKET\_ORIGIN, req.getHeader(ORIGIN));

res.addHeader(SEC\_WEBSOCKET\_LOCATION, getWebSocketLocation(req));

String protocol = req.getHeader(SEC\_WEBSOCKET\_PROTOCOL);

if (protocol != null) {

res.addHeader(SEC\_WEBSOCKET\_PROTOCOL, protocol);

}

// Calculate the answer of the challenge.

String key1 = req.getHeader(SEC\_WEBSOCKET\_KEY1);

String key2 = req.getHeader(SEC\_WEBSOCKET\_KEY2);

int a = (int) (Long.parseLong(key1.replaceAll("[^0-9]", "")) / key1.replaceAll("[^ ]", "").length());

int b = (int) (Long.parseLong(key2.replaceAll("[^0-9]", "")) / key2.replaceAll("[^ ]", "").length());

long c = req.getContent().readLong();

ChannelBuffer input = ChannelBuffers.buffer(16);

input.writeInt(a);

input.writeInt(b);

input.writeLong(c);

ChannelBuffer output = ChannelBuffers.wrappedBuffer(MD5.digest(input.array()));

res.setContent(output);

}

@Override

public ChannelHandler createDecoder() {

return new WebSocketFrameDecoder();

}

@Override

public ChannelHandler createEncoder() {

return new WebSocketFrameEncoder();

}

private String getWebSocketLocation(HttpRequest req) {

return getWebSocketProtocol(req) + req.getHeader(HttpHeaders.Names.HOST) + req.getUri();

}

private String getWebSocketProtocol(HttpRequest req) {

if(req.getHeader(HttpHeaders.Names.ORIGIN).matches("(?s)https://.\*")) { return "wss://"; } else { return "ws://"; }

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelFuture;

import org.jboss.netty.channel.ChannelFutureListener;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.handler.codec.frame.CorruptedFrameException;

import org.jboss.netty.handler.codec.frame.TooLongFrameException;

import org.jboss.netty.handler.codec.replay.ReplayingDecoder;

import cn.wlyf.helpers.UTF8Exception;

import cn.wlyf.helpers.UTF8Output;

import static cn.wlyf.netty.HybiWebSocketFrameDecoder.State.CORRUPT;

import static cn.wlyf.netty.HybiWebSocketFrameDecoder.State.FRAME\_START;

import static cn.wlyf.netty.HybiWebSocketFrameDecoder.State.MASKING\_KEY;

import static cn.wlyf.netty.HybiWebSocketFrameDecoder.State.PAYLOAD;

import static cn.wlyf.netty.Opcodes.OPCODE\_BINARY;

import static cn.wlyf.netty.Opcodes.OPCODE\_CLOSE;

import static cn.wlyf.netty.Opcodes.OPCODE\_CONT;

import static cn.wlyf.netty.Opcodes.OPCODE\_PING;

import static cn.wlyf.netty.Opcodes.OPCODE\_PONG;

import static cn.wlyf.netty.Opcodes.OPCODE\_TEXT;

public class HybiWebSocketFrameDecoder extends ReplayingDecoder<HybiWebSocketFrameDecoder.State> {

private final UTF8Output utf8Output = new UTF8Output();

private final boolean isServer;

private final boolean requireMaskedInboundFrames;

private final byte[] outboundMaskingKey;

private boolean frameFin;

private int frameOpcode;

private long framePayloadLen;

private byte[] inboundMaskingKey;

private DecodingHybiFrame currentFrame;

public static enum State {

FRAME\_START,

MASKING\_KEY,

PAYLOAD,

CORRUPT

}

public static HybiWebSocketFrameDecoder serverSide() {

return new HybiWebSocketFrameDecoder(true, null);

}

public static HybiWebSocketFrameDecoder clientSide(byte[] outboundMaskingKey) {

return new HybiWebSocketFrameDecoder(false, outboundMaskingKey);

}

private HybiWebSocketFrameDecoder(boolean isServer, byte[] outboundMaskingKey) {

super(FRAME\_START);

this.isServer = isServer;

this.requireMaskedInboundFrames = isServer;

this.outboundMaskingKey = outboundMaskingKey;

}

@Override

protected Object decode(ChannelHandlerContext ctx, final Channel channel, ChannelBuffer buffer, State state) throws Exception {

switch (state) {

case FRAME\_START: {

inboundMaskingKey = null;

// FIN, RSV, OPCODE

int b = buffer.readByte();

frameFin = (b & 0x80) != 0;

int frameRsv = (b & 0x70) >> 4;

frameOpcode = (b & 0x0F);

// MASK, PAYLOAD LEN 1

b = buffer.readByte();

boolean frameMasked = (b & 0x80) != 0;

int framePayloadLen1 = (b & 0x7F);

if (frameRsv != 0) {

protocolViolation(channel, "RSV != 0 and no extension negotiated, RSV:" + frameRsv);

return null;

}

if (isServer && requireMaskedInboundFrames && !frameMasked) {

protocolViolation(channel, "Received unmasked frame");

return null;

}

if (frameOpcode > 7) { // control frame (have MSB in opcode set)

// control frames MUST NOT be fragmented

if (!frameFin) {

protocolViolation(channel, "fragmented control frame");

return null;

}

// control frames MUST have payload 125 octets or less

if (framePayloadLen1 > 125) {

protocolViolation(channel, "control frame with payload length > 125 octets");

return null;

}

// check for reserved control frame opcodes

if (!(frameOpcode == OPCODE\_CLOSE || frameOpcode == OPCODE\_PING || frameOpcode == OPCODE\_PONG)) {

protocolViolation(channel, "control frame using reserved opcode " + frameOpcode);

return null;

}

// close frame : if there is a body, the first two bytes of the body MUST be a 2-byte

// unsigned integer (in network byte order) representing a status code

if (frameOpcode == 8 && framePayloadLen1 == 1) {

protocolViolation(channel, "received close control frame with payload len 1");

return null;

}

} else { // data frame

// check for reserved data frame opcodes

if (!(frameOpcode == OPCODE\_CONT || frameOpcode == OPCODE\_TEXT || frameOpcode == OPCODE\_BINARY)) {

protocolViolation(channel, "data frame using reserved opcode " + frameOpcode);

return null;

}

// check opcode vs message fragmentation state 1/2

if (currentFrame == null && frameOpcode == OPCODE\_CONT) {

protocolViolation(channel, "received continuation data frame outside fragmented message");

return null;

}

// check opcode vs message fragmentation state 2/2

if (currentFrame != null && frameOpcode != OPCODE\_CONT) {

protocolViolation(channel, "received non-continuation data frame while inside fragmented message");

return null;

}

}

if (framePayloadLen1 == 126) {

framePayloadLen = buffer.readUnsignedShort();

if (framePayloadLen < 126) {

protocolViolation(channel, "invalid data frame length (not using minimal length encoding)");

return null;

}

} else if (framePayloadLen1 == 127) {

framePayloadLen = buffer.readLong();

// TODO: check if it's bigger than 0x7FFFFFFFFFFFFFFF, Maybe just check if it's negative?

if (framePayloadLen < 65536) {

protocolViolation(channel, "invalid data frame length (not using minimal length encoding)");

return null;

}

} else {

framePayloadLen = framePayloadLen1;

}

if (frameMasked) {

checkpoint(MASKING\_KEY);

} else {

checkpoint(PAYLOAD);

return null;

}

}

case MASKING\_KEY: {

inboundMaskingKey = buffer.readBytes(4).array();

checkpoint(PAYLOAD);

}

case PAYLOAD: {

ChannelBuffer frame = buffer.readBytes(toFrameLength(framePayloadLen));

if (inboundMaskingKey != null) {

applyMask(frame, inboundMaskingKey);

}

checkpoint(FRAME\_START);

if (frameOpcode == OPCODE\_CLOSE) {

EncodingHybiFrame close = new EncodingHybiFrame(OPCODE\_CLOSE, true, 0, outboundMaskingKey, ChannelBuffers.buffer(0));

channel.write(close).addListener(new ChannelFutureListener() {

@Override

public void operationComplete(ChannelFuture channelFuture) throws Exception {

channel.close();

}

});

return null;

} else if (frameOpcode == OPCODE\_CONT) {

try {

currentFrame.append(frame);

} catch (UTF8Exception e) {

protocolViolation(channel, "invalid UTF-8 bytes");

}

} else if (frameOpcode == OPCODE\_PING || frameOpcode == OPCODE\_PONG) {

return new DecodingHybiFrame(frameOpcode, utf8Output, frame);

} else {

try {

currentFrame = new DecodingHybiFrame(frameOpcode, utf8Output, frame);

} catch (UTF8Exception e) {

protocolViolation(channel, "invalid UTF-8 bytes");

}

}

if (frameFin) {

DecodingHybiFrame result = currentFrame;

currentFrame = null;

return result;

} else {

return null;

}

}

case CORRUPT: {

// If we don't keep reading Netty will throw an exception saying

// we can't return null if no bytes read and state not changed.

buffer.readByte();

return null;

}

default:

throw new Error("Shouldn't reach here.");

}

}

static void applyMask(ChannelBuffer data, byte[] maskingKey) {

int length = data.writerIndex();

for (int i = 0; i < length; i++) {

data.setByte(i, data.getByte(i) ^ maskingKey[i % 4]);

}

}

private void protocolViolation(Channel channel, String reason) throws CorruptedFrameException {

checkpoint(CORRUPT);

if (channel.isConnected()) {

channel.write(ChannelBuffers.EMPTY\_BUFFER).addListener(ChannelFutureListener.CLOSE);

channel.close().awaitUninterruptibly();

}

throw new CorruptedFrameException(reason);

}

private int toFrameLength(long l) throws TooLongFrameException {

if (l > Integer.MAX\_VALUE) {

throw new TooLongFrameException("Length:" + l);

} else {

return (int) l;

}

}

}

package cn.wlyf.netty;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.channel.MessageEvent;

import org.jboss.netty.handler.codec.http.HttpRequest;

import org.jboss.netty.util.CharsetUtil;

import cn.wlyf.helpers.InboundCookieParser;

import cn.wlyf.helpers.QueryParameters;

import java.net.HttpCookie;

import java.net.SocketAddress;

import java.net.URI;

import java.util.\*;

public class NettyHttpRequest implements cn.wlyf.HttpRequest {

private final HttpRequest httpRequest;

private final MessageEvent messageEvent;

private final Map<String, Object> data = new HashMap<String, Object>();

private final Object id;

private final long timestamp;

private QueryParameters queryParameters;

private QueryParameters postParameters;

public NettyHttpRequest(MessageEvent messageEvent, HttpRequest httpRequest, Object id, long timestamp) {

this.messageEvent = messageEvent;

this.httpRequest = httpRequest;

this.id = id;

this.timestamp = timestamp;

}

@Override

public String uri() {

return httpRequest.getUri();

}

@Override

public NettyHttpRequest uri(String uri) {

httpRequest.setUri(uri);

return this;

}

@Override

public String header(String name) {

return httpRequest.getHeader(name);

}

@Override

public List<String> headers(String name) {

return httpRequest.getHeaders(name);

}

@Override

public boolean hasHeader(String name) {

return httpRequest.containsHeader(name);

}

@Override

public List<HttpCookie> cookies() {

return InboundCookieParser.parse(headers(COOKIE\_HEADER));

}

@Override

public HttpCookie cookie(String name) {

for (HttpCookie cookie : cookies()) {

if (cookie.getName().equals(name)) {

return cookie;

}

}

return null;

}

@Override

public String queryParam(String key) {

return parsedQueryParams().first(key);

}

@Override

public List<String> queryParams(String key) {

return parsedQueryParams().all(key);

}

@Override

public Set<String> queryParamKeys() {

return parsedQueryParams().keys();

}

@Override

public String postParam(String key) {

return parsedPostParams().first(key);

}

@Override

public List<String> postParams(String key) {

return parsedPostParams().all(key);

}

@Override

public Set<String> postParamKeys() {

return parsedPostParams().keys();

}

private QueryParameters parsedQueryParams() {

if (queryParameters == null) {

queryParameters = new QueryParameters(URI.create(uri()).getRawQuery());

}

return queryParameters;

}

private QueryParameters parsedPostParams() {

if (postParameters == null) {

postParameters = new QueryParameters(body());

}

return postParameters;

}

@Override

public String cookieValue(String name) {

HttpCookie cookie = cookie(name);

return cookie == null ? null : cookie.getValue();

}

@Override

public List<Map.Entry<String, String>> allHeaders() {

return httpRequest.getHeaders();

}

@Override

public String method() {

return httpRequest.getMethod().getName();

}

@Override

public String body() {

return httpRequest.getContent().toString(CharsetUtil.UTF\_8); // TODO get charset from request

}

@Override

public byte[] bodyAsBytes() {

ChannelBuffer buffer = httpRequest.getContent();

byte[] body = new byte[buffer.readableBytes()];

buffer.getBytes(buffer.readerIndex(), body);

return body;

}

@Override

public Map<String, Object> data() {

return data;

}

@Override

public Object data(String key) {

return data.get(key);

}

@Override

public NettyHttpRequest data(String key, Object value) {

data.put(key, value);

return this;

}

@Override

public Set<String> dataKeys() {

return data.keySet();

}

@Override

public SocketAddress remoteAddress() {

return messageEvent.getRemoteAddress();

}

@Override

public Object id() {

return id;

}

@Override

public long timestamp() {

return timestamp;

}

@Override

public String toString() {

return messageEvent.getRemoteAddress() + " " + httpRequest.getMethod() + " " + httpRequest.getUri();

}

}

package cn.wlyf.netty;

import org.jboss.netty.bootstrap.ServerBootstrap;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelPipeline;

import org.jboss.netty.channel.ChannelPipelineFactory;

import org.jboss.netty.channel.socket.nio.NioServerSocketChannelFactory;

import org.jboss.netty.handler.codec.http.HttpChunkAggregator;

import org.jboss.netty.handler.codec.http.HttpContentCompressor;

import org.jboss.netty.handler.codec.http.HttpContentDecompressor;

import org.jboss.netty.handler.codec.http.HttpRequestDecoder;

import org.jboss.netty.handler.codec.http.HttpResponseEncoder;

import org.jboss.netty.handler.ssl.SslHandler;

import cn.wlyf.EventSourceHandler;

import cn.wlyf.HttpHandler;

import cn.wlyf.WebServer;

import cn.wlyf.WebSocketHandler;

import cn.wlyf.WebbitException;

import cn.wlyf.handler.DateHeaderHandler;

import cn.wlyf.handler.HttpToEventSourceHandler;

import cn.wlyf.handler.HttpToWebSocketHandler;

import cn.wlyf.handler.PathMatchHandler;

import cn.wlyf.handler.ServerHeaderHandler;

import cn.wlyf.handler.exceptions.PrintStackTraceExceptionHandler;

import cn.wlyf.handler.exceptions.SilentExceptionHandler;

import cn.wlyf.helpers.NamingThreadFactory;

import cn.wlyf.helpers.SslFactory;

import javax.net.ssl.SSLContext;

import javax.net.ssl.SSLEngine;

import java.io.InputStream;

import java.net.InetAddress;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.net.URI;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.Callable;

import java.util.concurrent.Executor;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

import java.util.concurrent.FutureTask;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.TimeUnit;

import static org.jboss.netty.channel.Channels.pipeline;

public class NettyWebServer implements WebServer {

private static final long DEFAULT\_STALE\_CONNECTION\_TIMEOUT = 5000;

private final SocketAddress socketAddress;

private final URI publicUri;

private final List<HttpHandler> handlers = new ArrayList<HttpHandler>();

private final List<ExecutorService> executorServices = new ArrayList<ExecutorService>();

private final Executor executor;

private ServerBootstrap bootstrap;

private Channel channel;

private SSLContext sslContext;

protected long nextId = 1;

private Thread.UncaughtExceptionHandler exceptionHandler;

private Thread.UncaughtExceptionHandler ioExceptionHandler;

private ConnectionTrackingHandler connectionTrackingHandler;

private StaleConnectionTrackingHandler staleConnectionTrackingHandler;

private long staleConnectionTimeout = DEFAULT\_STALE\_CONNECTION\_TIMEOUT;

private int maxInitialLineLength = 4096;

private int maxHeaderSize = 8192;

private int maxChunkSize = 8192;

private int maxContentLength = 65536;

public NettyWebServer(int port) {

this(Executors.newSingleThreadScheduledExecutor(new NamingThreadFactory("WEBBIT-HANDLER-THREAD")), port);

}

private NettyWebServer(ExecutorService executorService, int port) {

this((Executor) executorService, port);

// If we created the executor, we have to be responsible for tearing it down.

executorServices.add(executorService);

}

public NettyWebServer(final Executor executor, int port) {

this(executor, new InetSocketAddress(port), localUri(port));

}

public NettyWebServer(final Executor executor, SocketAddress socketAddress, URI publicUri) {

this.executor = executor;

this.socketAddress = socketAddress;

this.publicUri = publicUri;

// Uncaught exceptions from handlers get dumped to console by default.

// To change, call uncaughtExceptionHandler()

uncaughtExceptionHandler(new PrintStackTraceExceptionHandler());

// Default behavior is to silently discard any exceptions caused

// when reading/writing to the client. The Internet is flaky - it happens.

connectionExceptionHandler(new SilentExceptionHandler());

setupDefaultHandlers();

}

protected void setupDefaultHandlers() {

add(new ServerHeaderHandler("Webbit"));

add(new DateHeaderHandler());

}

@Override

public NettyWebServer setupSsl(InputStream keyStore, String pass) throws WebbitException {

return this.setupSsl(keyStore, pass, pass);

}

@Override

public NettyWebServer setupSsl(InputStream keyStore, String storePass, String keyPass) throws WebbitException {

this.sslContext = new SslFactory(keyStore, storePass).getServerContext(keyPass);

return this;

}

@Override

public URI getUri() {

return publicUri;

}

@Override

public int getPort() {

if (publicUri.getPort() == -1) {

return publicUri.getScheme().equalsIgnoreCase("https") ? 443 : 80;

}

return publicUri.getPort();

}

@Override

public Executor getExecutor() {

return executor;

}

@Override

public NettyWebServer staleConnectionTimeout(long millis) {

staleConnectionTimeout = millis;

return this;

}

@Override

public NettyWebServer add(HttpHandler handler) {

handlers.add(handler);

return this;

}

@Override

public NettyWebServer add(String path, HttpHandler handler) {

return add(new PathMatchHandler(path, handler));

}

@Override

public NettyWebServer add(String path, WebSocketHandler handler) {

return add(path, new HttpToWebSocketHandler(handler));

}

@Override

public NettyWebServer add(String path, EventSourceHandler handler) {

return add(path, new HttpToEventSourceHandler(handler));

}

@Override

public Future<NettyWebServer> start() {

FutureTask<NettyWebServer> future = new FutureTask<NettyWebServer>(new Callable<NettyWebServer>() {

@Override

public NettyWebServer call() throws Exception {

if (isRunning()) {

throw new IllegalStateException("Server already started.");

}

// Configure the server.

bootstrap = new ServerBootstrap();

// Set up the event pipeline factory.

bootstrap.setPipelineFactory(new ChannelPipelineFactory() {

@Override

public ChannelPipeline getPipeline() throws Exception {

long timestamp = timestamp();

Object id = nextId();

ChannelPipeline pipeline = pipeline();

if (sslContext != null) {

SSLEngine sslEngine = sslContext.createSSLEngine();

sslEngine.setUseClientMode(false);

SslHandler ssl = new SslHandler(sslEngine);

ssl.setCloseOnSSLException(true);

pipeline.addLast("ssl", ssl);

}

pipeline.addLast("staleconnectiontracker", staleConnectionTrackingHandler);

pipeline.addLast("connectiontracker", connectionTrackingHandler);

pipeline.addLast("flashpolicydecoder", new FlashPolicyFileDecoder(executor, exceptionHandler, ioExceptionHandler, getPort()));

pipeline.addLast("decoder", new HttpRequestDecoder(maxInitialLineLength, maxHeaderSize, maxChunkSize));

pipeline.addLast("aggregator", new HttpChunkAggregator(maxContentLength));

pipeline.addLast("decompressor", new HttpContentDecompressor());

pipeline.addLast("encoder", new HttpResponseEncoder());

pipeline.addLast("compressor", new HttpContentCompressor());

pipeline.addLast("handler", new NettyHttpChannelHandler(executor, handlers, id, timestamp, exceptionHandler, ioExceptionHandler));

return pipeline;

}

});

staleConnectionTrackingHandler = new StaleConnectionTrackingHandler(staleConnectionTimeout, executor);

ScheduledExecutorService staleCheckExecutor = Executors.newSingleThreadScheduledExecutor(new NamingThreadFactory("WEBBIT-STALE-CONNECTION-CHECK-THREAD"));

staleCheckExecutor.scheduleWithFixedDelay(new Runnable() {

@Override

public void run() {

staleConnectionTrackingHandler.closeStaleConnections();

}

}, staleConnectionTimeout / 2, staleConnectionTimeout / 2, TimeUnit.MILLISECONDS);

executorServices.add(staleCheckExecutor);

connectionTrackingHandler = new ConnectionTrackingHandler();

ExecutorService bossExecutor = Executors.newSingleThreadExecutor(new NamingThreadFactory("WEBBIT-BOSS-THREAD"));

executorServices.add(bossExecutor);

ExecutorService workerExecutor = Executors.newSingleThreadExecutor(new NamingThreadFactory("WEBBIT-WORKER-THREAD"));

executorServices.add(workerExecutor);

bootstrap.setFactory(new NioServerSocketChannelFactory(bossExecutor, workerExecutor, 1));

channel = bootstrap.bind(socketAddress);

return NettyWebServer.this;

}

});

// don't use Executor here - it's just another resource we need to manage -

// thread creation on startup should be fine

final Thread thread = new Thread(future, "WEBBIT-STARTUP-THREAD");

thread.start();

return future;

}

public boolean isRunning() {

return channel != null && channel.isBound();

}

@Override

public Future<WebServer> stop() {

FutureTask<WebServer> future = new FutureTask<WebServer>(new Callable<WebServer>() {

@Override

public WebServer call() throws Exception {

if (channel != null) {

channel.close();

}

if (connectionTrackingHandler != null) {

connectionTrackingHandler.closeAllConnections();

connectionTrackingHandler = null;

}

if (bootstrap != null) {

bootstrap.releaseExternalResources();

}

// shut down all services & give them a chance to terminate

for (ExecutorService executorService : executorServices) {

shutdownAndAwaitTermination(executorService);

}

bootstrap = null;

if (channel != null) {

channel.getCloseFuture().await();

}

return NettyWebServer.this;

}

});

// don't use Executor here - it's just another resource we need to manage -

// thread creation on shutdown should be fine

final Thread thread = new Thread(future, "WEBBIT-SHUTDOW-THREAD");

thread.start();

return future;

}

// See JavaDoc for ExecutorService

private void shutdownAndAwaitTermination(ExecutorService executorService) {

executorService.shutdown(); // Disable new tasks from being submitted

try {

// Wait a while for existing tasks to terminate

if (!executorService.awaitTermination(5, TimeUnit.SECONDS)) {

executorService.shutdownNow(); // Cancel currently executing tasks

// Wait a while for tasks to respond to being cancelled

if (!executorService.awaitTermination(5, TimeUnit.SECONDS)) {

System.err.println("ExecutorService did not terminate");

}

}

} catch (InterruptedException ie) {

// (Re-)Cancel if current thread also interrupted

executorService.shutdownNow();

// Preserve interrupt status

Thread.currentThread().interrupt();

}

}

@Override

public NettyWebServer uncaughtExceptionHandler(Thread.UncaughtExceptionHandler exceptionHandler) {

this.exceptionHandler = exceptionHandler;

return this;

}

@Override

public NettyWebServer connectionExceptionHandler(Thread.UncaughtExceptionHandler ioExceptionHandler) {

this.ioExceptionHandler = ioExceptionHandler;

return this;

}

/\*\*

\* @see HttpRequestDecoder

\*/

public NettyWebServer maxChunkSize(int maxChunkSize) {

this.maxChunkSize = maxChunkSize;

return this;

}

/\*\*

\* @see HttpChunkAggregator

\*/

public NettyWebServer maxContentLength(int maxContentLength) {

this.maxContentLength = maxContentLength;

return this;

}

/\*\*

\* @see HttpRequestDecoder

\*/

public NettyWebServer maxHeaderSize(int maxHeaderSize) {

this.maxHeaderSize = maxHeaderSize;

return this;

}

/\*\*

\* @see HttpRequestDecoder

\*/

public NettyWebServer maxInitialLineLength(int maxInitialLineLength) {

this.maxInitialLineLength = maxInitialLineLength;

return this;

}

private static URI localUri(int port) {

try {

return URI.create("http://" + InetAddress.getLocalHost()

.getHostName() + (port == 80 ? "" : (":" + port)) + "/");

} catch (UnknownHostException e) {

throw new RuntimeException("can not create URI from localhost hostname - use constructor to pass an explicit URI", e);

}

}

protected long timestamp() {

return System.currentTimeMillis();

}

protected Object nextId() {

return nextId++;

}

}

package cn.wlyf.netty;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.channel.ChannelStateEvent;

import org.jboss.netty.channel.MessageEvent;

import org.jboss.netty.channel.SimpleChannelHandler;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.concurrent.Executor;

/\*\*

\* Keeps track of all connections and automatically closes the ones that are stale.

\*/

public class StaleConnectionTrackingHandler extends SimpleChannelHandler {

private final Map<Channel, Long> stamps = new HashMap<Channel, Long>();

private final long timeout;

private final Executor executor;

public StaleConnectionTrackingHandler(long timeout, Executor executor) {

this.timeout = timeout;

this.executor = executor;

}

@Override

public void channelOpen(ChannelHandlerContext ctx, ChannelStateEvent e) throws Exception {

stamp(e.getChannel());

super.channelOpen(ctx, e);

}

@Override

public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws Exception {

stamp(e.getChannel());

super.messageReceived(ctx, e);

}

private void stamp(final Channel channel) {

executor.execute(new Runnable() {

@Override

public void run() {

stamps.put(channel, System.currentTimeMillis());

}

});

}

public void closeStaleConnections() {

executor.execute(new Runnable() {

@Override

public void run() {

Iterator<Map.Entry<Channel, Long>> entries = stamps.entrySet().iterator();

while (entries.hasNext()) {

Map.Entry<Channel, Long> entry = entries.next();

if (isStale(entry.getValue())) {

entry.getKey().close();

entries.remove();

}

}

}

});

}

/\*\*

\* Stops tracking this channel for staleness. This happens for WebSockets and EventSource connections.

\*

\* @param channel

\*

\*/

public void stopTracking(final Channel channel) {

executor.execute(new Runnable() {

@Override

public void run() {

stamps.remove(channel);

}

});

}

private boolean isStale(Long timeStamp) {

return System.currentTimeMillis() - timeStamp > timeout;

}

}

package cn.wlyf.netty;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.channel.ChannelStateEvent;

import org.jboss.netty.channel.ExceptionEvent;

import org.jboss.netty.channel.MessageEvent;

import org.jboss.netty.channel.SimpleChannelUpstreamHandler;

import org.jboss.netty.handler.codec.http.websocket.WebSocketFrame;

import cn.wlyf.WebSocketHandler;

import java.util.concurrent.Executor;

public class WebSocketConnectionHandler extends SimpleChannelUpstreamHandler {

private final Executor executor;

private final NettyWebSocketConnection webSocketConnection;

private final WebSocketHandler webSocketHandler;

private final ConnectionHelper connectionHelper;

public WebSocketConnectionHandler(

Executor executor,

Thread.UncaughtExceptionHandler exceptionHandler,

Thread.UncaughtExceptionHandler ioExceptionHandler,

final NettyWebSocketConnection webSocketConnection,

final WebSocketHandler webSocketHandler

) {

this.executor = executor;

this.webSocketConnection = webSocketConnection;

this.webSocketHandler = webSocketHandler;

this.connectionHelper = new ConnectionHelper(executor, exceptionHandler, ioExceptionHandler) {

@Override

protected void fireOnClose() throws Throwable {

webSocketHandler.onClose(webSocketConnection);

}

};

}

@Override

public void channelUnbound(ChannelHandlerContext ctx, ChannelStateEvent e) throws Exception {

connectionHelper.fireOnClose(e);

}

@Override

public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e) throws Exception {

connectionHelper.fireConnectionException(e);

}

@Override

public void messageReceived(ChannelHandlerContext ctx, final MessageEvent e) throws Exception {

final Thread.UncaughtExceptionHandler exceptionHandlerWithContext = connectionHelper.webbitExceptionWrappingExceptionHandler(e.getChannel());

Object message = e.getMessage();

if (message instanceof DecodingHybiFrame) {

DecodingHybiFrame frame = (DecodingHybiFrame) message;

frame.dispatchMessage(webSocketHandler, webSocketConnection, executor, exceptionHandlerWithContext);

} else {

// Hixie 75/76

final WebSocketFrame frame = (WebSocketFrame) message;

executor.execute(new Runnable() {

@Override

public void run() {

try {

webSocketHandler.onMessage(webSocketConnection, frame.getTextData());

} catch (Throwable t) {

exceptionHandlerWithContext.uncaughtException(Thread.currentThread(), t);

}

}

});

}

}

}

package cn.wlyf.stub;

import cn.wlyf.EventSourceConnection;

import cn.wlyf.HttpRequest;

import cn.wlyf.WebSocketConnection;

import java.util.LinkedList;

import java.util.List;

import java.util.concurrent.Executor;

/\*\*

\* Implementation of {@link EventSourceConnection} and {@link WebSocketConnection} that is easy to construct and

\* makes it easy to inspect results. Useful for testing.

\*/

public class StubConnection extends StubDataHolder implements EventSourceConnection, WebSocketConnection {

private final List<String> sentMessages = new LinkedList<String>();

private final List<byte[]> sentBinaryMessages = new LinkedList<byte[]>();

private final List<byte[]> sentPings = new LinkedList<byte[]>();

private final List<byte[]> sentPongs = new LinkedList<byte[]>();

private boolean closed = false;

private HttpRequest httpRequest;

private String version = null;

public StubConnection(HttpRequest httpRequest) {

this.httpRequest = httpRequest;

}

public StubConnection() {

this(new StubHttpRequest());

}

@Override

public HttpRequest httpRequest() {

return httpRequest;

}

@Override

public StubConnection send(cn.wlyf.EventSourceMessage message) {

return send(message.build());

}

public StubConnection httpRequest(HttpRequest httpRequest) {

this.httpRequest = httpRequest;

return this;

}

@Override

public StubConnection send(String message) {

sentMessages.add(message);

return this;

}

@Override

public StubConnection send(byte[] message) {

return send(message, 0, message.length);

}

@Override

public StubConnection send(byte[] message, int offset, int length) {

byte[] subMessage = new byte[length];

System.arraycopy(message, offset, subMessage, 0, length);

sentBinaryMessages.add(subMessage);

return this;

}

@Override

public StubConnection ping(byte[] message) {

sentPings.add(message);

return this;

}

@Override

public StubConnection pong(byte[] message) {

sentPongs.add(message);

return this;

}

@Override

public StubConnection close() {

closed = true;

return this;

}

public boolean closed() {

return closed;

}

public List<String> sentMessages() {

return sentMessages;

}

public List<byte[]> sentBinaryMessages() {

return sentBinaryMessages;

}

public List<byte[]> sentPings() {

return sentPings;

}

public List<byte[]> sentPongs() {

return sentPongs;

}

@Override

public StubConnection data(String key, Object value) {

super.data(key, value);

return this;

}

@Override

public Executor handlerExecutor() {

return this;

}

@Override

public String version() {

return version;

}

public StubConnection version(String version) {

this.version = version;

return this;

}

@Override

public void execute(Runnable command) {

command.run();

}

}

package cn.wlyf.stub;

import cn.wlyf.EventSourceConnection;

import cn.wlyf.EventSourceHandler;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import cn.wlyf.WebSocketConnection;

import cn.wlyf.WebSocketHandler;

import java.util.concurrent.Executor;

public class StubHttpControl implements HttpControl {

private HttpRequest request;

private HttpResponse response;

private WebSocketHandler webSocketHandler;

private WebSocketConnection webSocketConnection;

public StubHttpControl() {

}

public StubHttpControl(HttpRequest request, HttpResponse response) {

this.request = request;

this.response = response;

}

public StubHttpControl(WebSocketConnection connection) {

this.webSocketConnection = connection;

}

public StubHttpControl(HttpRequest request, HttpResponse response, WebSocketConnection connection) {

this.request = request;

this.response = response;

this.webSocketConnection = connection;

}

public HttpRequest request() {

return request;

}

public HttpResponse response() {

return response;

}

public StubHttpControl request(HttpRequest request) {

this.request = request;

return this;

}

public StubHttpControl response(HttpResponse response) {

this.response = response;

return this;

}

@Override

public void nextHandler() {

nextHandler(request, response, this);

}

@Override

public void nextHandler(HttpRequest request, HttpResponse response) {

nextHandler(request, response, this);

}

@Override

public void nextHandler(HttpRequest request, HttpResponse response, HttpControl control) {

response.status(404).end();

}

@Override

public WebSocketConnection upgradeToWebSocketConnection(WebSocketHandler handler) {

this.webSocketHandler = handler;

return this.webSocketConnection;

}

@Override

public WebSocketConnection webSocketConnection() {

return this.webSocketConnection;

}

@Override

public EventSourceConnection upgradeToEventSourceConnection(EventSourceHandler handler) {

throw new UnsupportedOperationException();

// this.webSocketHandler = handler;

// return webSocketConnection;

}

@Override

public EventSourceConnection eventSourceConnection() {

throw new UnsupportedOperationException();

// return this.webSocketConnection;

}

public StubHttpControl webSocketConnection(WebSocketConnection connection) {

this.webSocketConnection = connection;

return this;

}

public WebSocketHandler webSocketHandler() {

return webSocketHandler;

}

@Override

public Executor handlerExecutor() {

return this;

}

@Override

public void execute(Runnable command) {

command.run();

}

}

package cn.wlyf.stub;

import cn.wlyf.EventSourceConnection;

import cn.wlyf.EventSourceHandler;

import cn.wlyf.HttpControl;

import cn.wlyf.HttpRequest;

import cn.wlyf.HttpResponse;

import cn.wlyf.WebSocketConnection;

import cn.wlyf.WebSocketHandler;

import java.util.concurrent.Executor;

public class StubHttpControl implements HttpControl {

private HttpRequest request;

private HttpResponse response;

private WebSocketHandler webSocketHandler;

private WebSocketConnection webSocketConnection;

public StubHttpControl() {

}

public StubHttpControl(HttpRequest request, HttpResponse response) {

this.request = request;

this.response = response;

}

public StubHttpControl(WebSocketConnection connection) {

this.webSocketConnection = connection;

}

public StubHttpControl(HttpRequest request, HttpResponse response, WebSocketConnection connection) {

this.request = request;

this.response = response;

this.webSocketConnection = connection;

}

public HttpRequest request() {

return request;

}

public HttpResponse response() {

return response;

}

public StubHttpControl request(HttpRequest request) {

this.request = request;

return this;

}

public StubHttpControl response(HttpResponse response) {

this.response = response;

return this;

}

@Override

public void nextHandler() {

nextHandler(request, response, this);

}

@Override

public void nextHandler(HttpRequest request, HttpResponse response) {

nextHandler(request, response, this);

}

@Override

public void nextHandler(HttpRequest request, HttpResponse response, HttpControl control) {

response.status(404).end();

}

@Override

public WebSocketConnection upgradeToWebSocketConnection(WebSocketHandler handler) {

this.webSocketHandler = handler;

return this.webSocketConnection;

}

@Override

public WebSocketConnection webSocketConnection() {

return this.webSocketConnection;

}

@Override

public EventSourceConnection upgradeToEventSourceConnection(EventSourceHandler handler) {

throw new UnsupportedOperationException();

// this.webSocketHandler = handler;

// return webSocketConnection;

}

@Override

public EventSourceConnection eventSourceConnection() {

throw new UnsupportedOperationException();

// return this.webSocketConnection;

}

public StubHttpControl webSocketConnection(WebSocketConnection connection) {

this.webSocketConnection = connection;

return this;

}

public WebSocketHandler webSocketHandler() {

return webSocketHandler;

}

@Override

public Executor handlerExecutor() {

return this;

}

@Override

public void execute(Runnable command) {

command.run();

}

}

package cn.wlyf.wrapper;

import cn.wlyf.EventSourceConnection;

import cn.wlyf.HttpRequest;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.Executor;

public class EventSourceConnectionWrapper implements EventSourceConnection {

/\*\*

\*connection property

\*

\*/

private EventSourceConnection connection;

public EventSourceConnectionWrapper(EventSourceConnection connection) {

this.connection = connection;

}

public EventSourceConnection underlyingControl() {

return connection;

}

public EventSourceConnectionWrapper underlyingControl(EventSourceConnection control) {

this.connection = control;

return this;

}

public EventSourceConnection originalControl() {

if (connection instanceof EventSourceConnectionWrapper) {

EventSourceConnectionWrapper wrapper = (EventSourceConnectionWrapper) connection;

return wrapper.originalControl();

} else {

return connection;

}

}

@Override

public HttpRequest httpRequest() {

return connection.httpRequest();

}

@Override

public EventSourceConnectionWrapper send(cn.wlyf.EventSourceMessage message) {

connection.send(message);

return this;

}

@Override

public EventSourceConnectionWrapper close() {

connection.close();

return this;

}

@Override

public Map<String, Object> data() {

return connection.data();

}

@Override

public Object data(String key) {

return connection.data(key);

}

@Override

public EventSourceConnectionWrapper data(String key, Object value) {

connection.data(key, value);

return this;

}

@Override

public Set<String> dataKeys() {

return connection.dataKeys();

}

@Override

public Executor handlerExecutor() {

return connection.handlerExecutor();

}

@Override

public void execute(Runnable command) {

connection.execute(command);

}

}