知识图谱构建子系统

1 编写目的

本测试报告旨在全面梳理并记录数据爬取与构建项目在测试阶段的各项工作与成果，帮助团队成员及相关利益相关者清晰了解当前项目的测试情况与整体质量。报告内容涵盖测试的方法、环境、目标、执行过程、结果分析及优化建议，旨在为项目质量评估与后续改进提供有力支撑。同时，本报告也有助于提升项目的完整性与实用性，增强用户对系统的信任感和满意度。

2 项目背景

在全球化持续推进与信息技术高速发展的背景下，海外博物馆中收藏的中国文物日益受到公众关注。这些文物凝聚着丰富的历史、文化与艺术价值，对于弘扬中华文明、促进中外文化交流具有重要意义。然而，受限于地理距离、语言差异和信息不对称等因素，公众获取和理解这些文物信息仍面临诸多困难。为此，本项目致力于运用先进的信息技术手段，构建海外中国文物的知识图谱，旨在更直观、系统地展示中国文化的深厚底蕴，凸显中国文物在世界文化史中的独特地位。

编写程序从海外博物馆网站爬取中国文物信息，进行加工处理，构建海外文物知识图谱。主要包括以下功能：

1. **数据爬取**：本项目要求采集海外博物馆中的文物信息，涵盖指定的3家博物馆（具体网址见附件），确保尽可能完整地获取其馆藏的所有中国文物数据。爬取内容包括文物名称、图片、年代、简介等核心信息，所有采集到的数据需按照指定格式进行整理与保存，具体格式要求详见附件。
2. **数据清洗**：数据缺失处理，异常值检测与处理，数据格式统一，重复数据处理，数据转换与映射，自动化清洗流程构建
3. **数据翻译**：利用百度翻译的API批量翻译csv文件中的某列内容。
4. **数据建模**：将爬取的数据转化为三元组形式。
5. **数据存储**：将翻译后的数据导入到mysql中，根据步骤2中构建的三元组模型，将数据导入Neo4j图数据库进行存储，并发布为开放链接数据，便于实现关系图谱、时间轴等知识图谱的可视化及问答功能开发。

3 测试范围

1. **数据爬取：**海外数据爬取完整性
2. **数据清洗：**数据补充以及更改
3. **数据翻译：**翻译准确性
4. **数据建模：**三元组完整性与可持续性
5. **数据存储：**服务器与MySQL数据库连接的稳定性，定时备份功能

4 测试环境

环境	详细信息
测试服务端	操作系统：Windows11 处理器：AMD R7-6800H CPU 内存：16G
测试客户端	操作系统：Windows11 处理器：AMD R7-6800H CPU 内存：16G
MySQL服务器	

环境	详细信息
Neo4J图数据库服务器	操作系统：Alibaba Cloud Linux 3.2104 LTS 64位 CPU&内存： 2核(vCPU)2 GiB 公网带宽：1 Mbps

5 测试时间

开始时间：第八周

结束时间：第十二周

按照原定项目推进计划进行测试，每周测试一个版块并进行测试总结。

6 测试人员

人员	任务
宋玉佳	数据爬取 数据补充 服务器配置 数据上传以及知识图谱构建
陈旭东	数据清洗岗位，主要负责原始数据的预处理与质量提升工作
林瀚之	对清洗翻译后的数据进行三元组设计，通过代码导入三元组csv文件形成知识图谱，编写测试报告
杨博超	负责克利夫兰博物馆的数据爬取工作，知识图谱子系统ppt制作
宋朝威	对清洗后的数据进行翻译操作
徐承昊	对清洗后的数据进行翻译操作

7 测试内容

7.1数据爬取

1. 海外数据爬取的稳定性：测试海外数据爬取过程中的稳定性，包括网络连接稳定性、爬虫程序稳定性以及数据 源的稳定性

2. 数据补充的完整性：验证爬取的数据是否完整，包括数据是否有缺失、数据是否有重复以及数据是否准确

```
\venv_cs2101\model_code\null_
空行数量: 3618
```

```
列 'museum' 的语言种类: {'sq', 'et', 'fi'}
列 'title' 的语言种类: {'ko', 'zh-cn'}
列 'era' 的语言种类: {'tl'}
列 'material' 的语言种类: {'tl'}
列 'size' 的语言种类: {'zh-cn'}
列 'description' 的语言种类: {'ko'}
指定列的语言种类:  {'sq', 'et', 'fi', 'tl', 'ko', 'zh-cn'}
```

7.2数据清洗

1. 筛选数据列

```
# 读取原始 CSV 文件
df = pd.read_csv('D:/pycharm/pythonProject/sp/16penn_museum_chinese_artifacts.csv')

#要保留的字段
fields = ['object_id', 'title','link','image','object_type','provenience', 'creator', 'date_made', 'materia
```

2. 重复值、空值处理

```
# 4. 删除完全重复的行
df.drop_duplicates(inplace=True)
###############
df.reset_index(drop=True, inplace=True)
df['id'] = df.index + 1
```

3. 数据格式规范

```
# 2. 字符串标准化：统一小写 & 去除前后空格
for col in ['tombstone', 'title', 'creation_date', 'collection',
            'provenance', 'url', 'image_web']:
    df[col] = df[col].astype(str).str.strip().str.lower()
```

4. 缺省值处理

```
# 1. 替换空值：通用字段 → "unknown"，image_web → "no_image"
df.fillna(value={
    'tombstone': 'unknown',
    'title': 'unknown',
    'creation_date': 'unknown',
    'collection': 'unknown',
    'provenance': 'unknown',
    'url': 'unknown',
    'image_web': 'no_image'
}, inplace=True)
```

7.3 数据翻译

1. 对爬取的csv文件进行翻译，通过调用百度翻译API，进行csv文件按列翻译

```python
# 百度翻译 API 信息
appid = "20250514002357144"
secret_key = "gn6EDPtfVy01_0TkMkwc"

# 最大字符限制（百度限制4800）
MAX_CHARS = 4800


# 封装的单批翻译函数（请求一次）
def baidu_batch_translate_batch(sentences, from_lang="auto", to_lang="zh"):
    q = "\n".join(sentences)
    salt = str(random.randint(32768, 65536))
    sign_str = appid + q + salt + secret_key
    sign = hashlib.md5(sign_str.encode()).hexdigest()

    url = "http://api.fanyi.baidu.com/api/trans/vip/translate"
    params = {
        "q": q,
        "from": from_lang,
        "to": to_lang,
        "appid": appid,
        "salt": salt,
        "sign": sign,
    }

    try:
        response = requests.get(url, params=params, timeout=8)
        result = response.json()
        if "trans_result" in result:
            return [item["dst"] for item in result["trans_result"]]
        else:
            print("❌ 翻译失败:", result)
            return sentences  # 返回原句，避免崩溃
    except Exception as e:
        print("⚠️ 请求异常:", e)
        return sentences
```

2. 由于进行批量翻译，百度翻译对单次请求字符有限制，对于较长的翻译内容进行拆分，使其符合字符要求在请求翻译

```python
# 批量翻译并按字符数拆分
def baidu_batch_translate(sentences, from_lang="auto", to_lang="zh"):
    results = []
    batch = []
    total_length = 0

    for sentence in sentences:
        sentence = sentence.strip()
        if sentence == "":
            results.append("")
            continue

        if len(sentence) > MAX_CHARS:
            print("⚠️ 单句过长，跳过或截断：", sentence[:30])
            results.append(sentence)
            continue

        if total_length + len(sentence) + 1 > MAX_CHARS:
            results.extend(retry_translate(batch))
            batch = [sentence]
            total_length = len(sentence) + 1
        else:
            batch.append(sentence)
            total_length += len(sentence) + 1

    if batch:
        results.extend(retry_translate(batch))

    return results
```

3. 参数配置，可以指定起始行进行翻译，也可以规定多少进行一次保存，以免数据丢失

```python
# === 参数配置 ===
input_file = "1.csv"
output_file = "translated1.csv"
columns_to_translate = ["Description"]  # 你可以填多个列名
start_row = 3676  # 从第几行开始翻译（从0起）
batch_size = 1000 #多少行保存一次
sleep_time = 0.6 #请求时间间隔
```

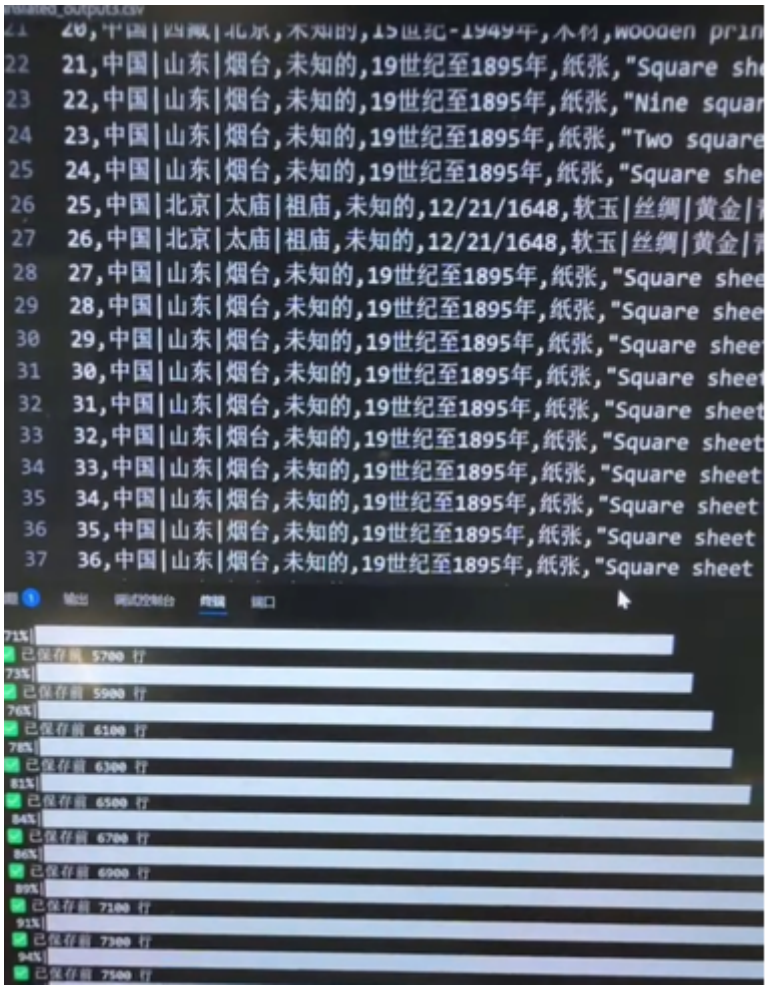
4. 支持断点续传，可以接着翻译已经翻译一部分的的文件

```python
# 断点续传支持
if os.path.exists(output_file):
    df_translated = pd.read_csv(output_file)
    df.update(df_translated)
    print("📄 已加载已有翻译进度，支持断点续传")
```

5. 主题翻译函数

```python
# === 开始翻译 ===
for col in columns_to_translate:
    print(f"\n🔄 正在翻译列：{col}（从第 {start_row} 行开始）")

    for start in tqdm(range(start_row, len(df), batch_size)):
        end = min(start + batch_size, len(df))
        row_indices = df.iloc[start:end].index  # 确保正确切片

        to_translate = []
        valid_indices = []

        for idx in row_indices:
            val = str(df.at[idx, col]) if pd.notna(df.at[idx, col]) else ""
            if val.strip() != "":
                to_translate.append(val)
                valid_indices.append(idx)

        if not to_translate:
            continue  # 跳过空行

        translated = baidu_batch_translate(to_translate)

        for i, idx in enumerate(valid_indices):
            if i < len(translated):
                df.at[idx, col] = translated[i]

        # 每1000行保存一次
        if (end - start_row) % 1000 == 0:
            df.to_csv(output_file, index=False)
            print(f"✅ 已保存至第 {end} 行")

        time.sleep(sleep_time)

# 最终保存
df.to_csv(output_file, index=False)
print("\n🎉 所有翻译完成，结果保存到：", output_file)
```

6. 进行翻译后的效果



7. 复检未翻译成功的内容，通过代码输出txt文件反馈未完全翻译成功的内容

```python
import pandas as pd
import re

# 读取 Excel 文件
df = pd.read_csv("C:/Users/pc/Desktop/时间轴.csv", encoding="gbk")

# 定义用于识别“含英文”的正则表达式
def contains_english(text):
    if isinstance(text, str):
        return bool(re.search(r'[A-Za-z]', text))
    return False

# 对整行判断是否“几乎全是英文”
def is_mostly_english(row):
    english_count = sum(contains_english(str(cell)) for cell in row)
    return english_count >= len(row) / 2

# 筛选含英文的行
mostly_english_rows = df[df.apply(is_mostly_english, axis=1)]

# 提取所有文本字段中仍存在的“英文字符串”
english_phrases = set()
for row in mostly_english_rows.itertuples(index=False):
    for cell in row:
        if isinstance(cell, str):
            matches = re.findall(r'[A-Za-z0-9][A-Za-z0-9\s\-,.()\'"/:;!?&%]+', cell)
            for match in matches:
                english_phrases.add(match.strip())

# 导出结果
with open("C:/Users/pc/Desktop/untranslated_english.txt", "w", encoding="utf-8") as f:
    for phrase in sorted(english_phrases):
        f.write(phrase + "\n")

print(f"总记录数：{len(df)}")
print(f"含英文的记录：{len(mostly_english_rows)}")
print(f"唯一仍含英文的字符串：{len(english_phrases)}")
```

8. 统计物品名称，检查翻译后的中文名称的准确性。

```python
import pandas as pd
from collections import Counter

# 读取 CSV 文件（注意根据实际编码调整 encoding）
df = pd.read_csv("C:/Users/pc/Desktop/时间轴.csv", encoding="gbk")

# 提取需要统计的列，并去除缺失值
names = df['Title'].dropna().tolist()

# 统计出现次数
name_counts = Counter(names)

# 转为 DataFrame 并按次数降序排序
count_df = pd.DataFrame(name_counts.items(), columns=['名称', '出现次数'])
count_df = count_df.sort_values(by='出现次数', ascending=False)

# 打印结果
print(count_df)

# 保存为 CSV 文件
output_path = "C:/Users/pc/Desktop/名称统计结果.csv"
count_df.to_csv(output_path, index=False, encoding="utf-8")
```

7.4 数据建模

三元组设计

实体设计

文物（ID，文物名，文物详细信息，图片链接）

关系设计

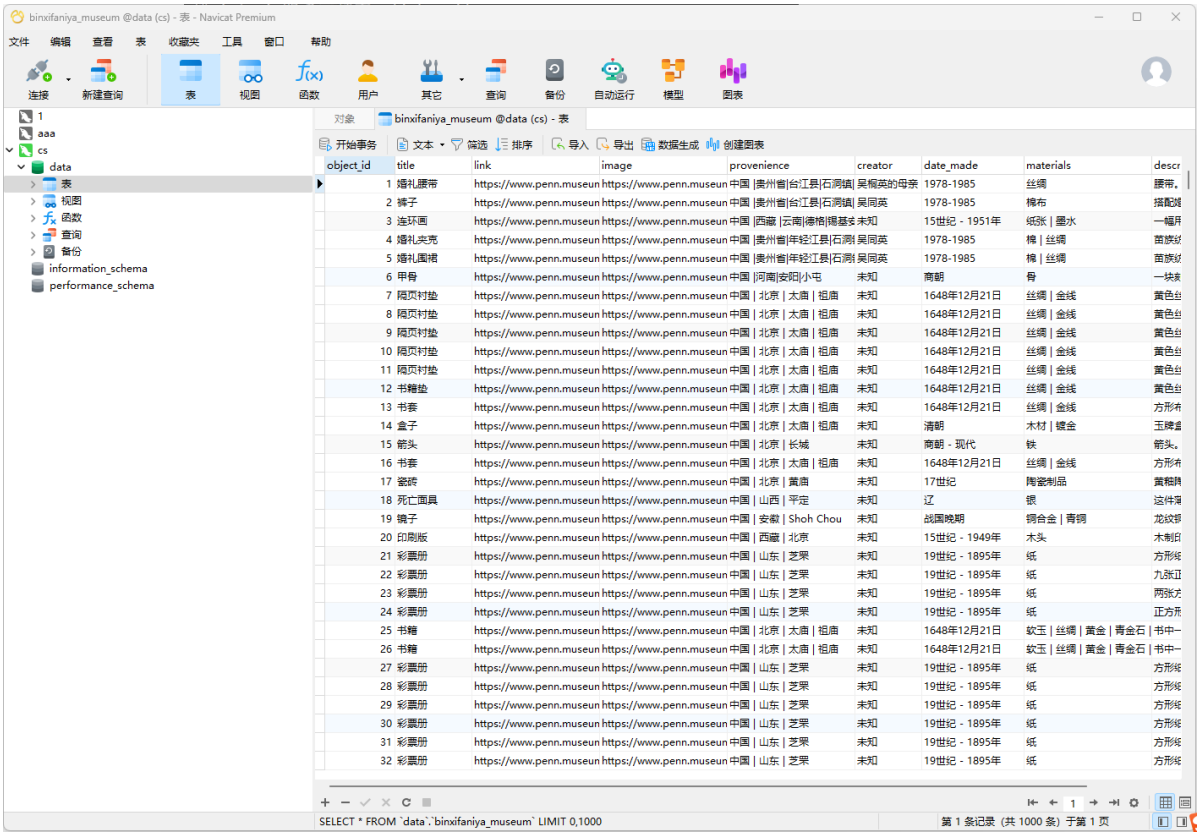
博物馆-包含-文物

文物-年代-朝代或时间区间

文物-作者-作者

7.5 数据存储

将最后翻译的csv文件导入到mysql中



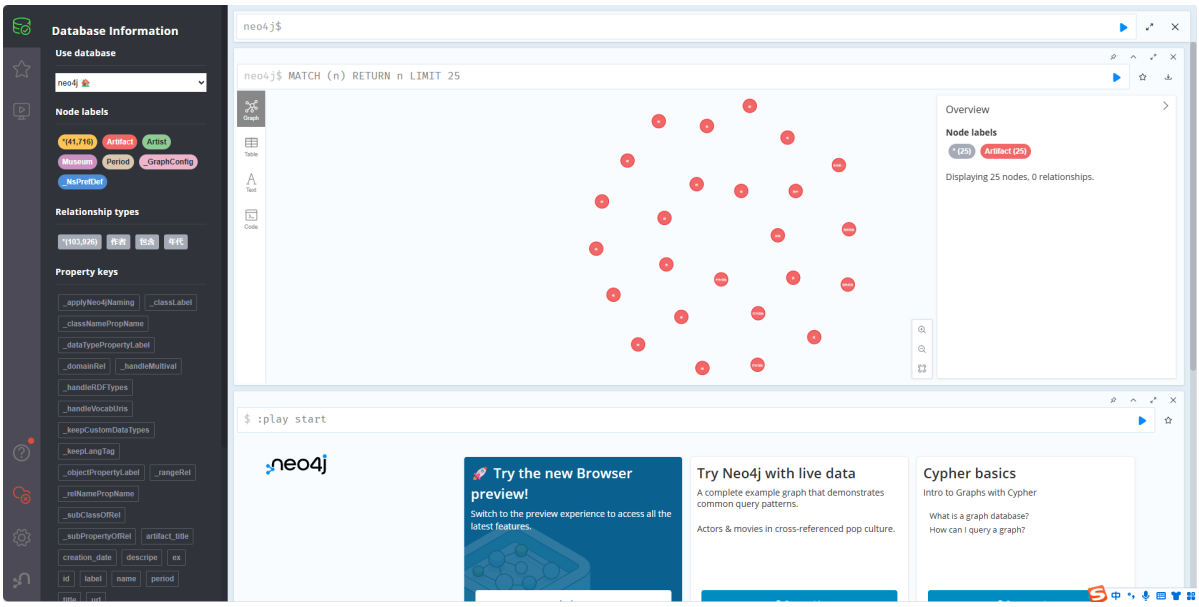
导入三元组

代码将三元组csv文件导入neo4j

通过load语句先在neo4j中创建包含多项信息的文物点

而后load导入博物馆，作者，年代与各个文物之间的关系

最终形成完整知识图谱



8 测试结果

8.1 数据获取

8.1.1 海外数据与国内数据爬取的稳定性：

- 测试结果显示，数据爬取过程总体稳定，但在某些情况下，数据爬取的速度较慢，可能是由于网络延迟或数据源服务器的响应 速度问题。

8.1.2 数据补充的完整性：

- 数据获取过程中的完整性总体较好，但存在少量数据缺失或不完整的情况。
- 对比多个数据源的数据结果，可以发现少量数据的重复，需要进行去重处理。

8.2 数据清洗

- 清洗效果较好

8.3 数据翻译

- 翻译效果整体较好，仍有部分错误需要手动纠正

8.4 数据建模

8.2.1 构建三元组的可持续性：

- 数据建模过程稳定，结果较为稳定，资源消耗保持在可接受范围内。

8.2.2 正确性：

- 实体识别和关系抽取的准确性较高，但属性提取有时存在错误。

8.5数据存储

8.5.1 服务器与MySQL数据库连接的稳定性：

- 连接稳定，但在数据传输高峰期可能出现短暂延迟。
- 数据传输整体可靠，但需要监控并调整数据库连接池的配置以提高效率。

8.5.2 图数据库与服务器连接的稳定性：

- 连接总体稳定，但在大规模数据可视化操作时可能出现短暂延迟。
- 数据传输稳定性可接受，但建议优化数据传输通道。

8.5.3 可扩展性：

- 系统在处理大规模数据和多用户访问时表现良好，但仍需进一步优化。 用户请求响应速度较快，但可以进一步改进