Why Nominatim Can’t Find Hiroshima Peace Memorial Museum

and How to Fix It

Abe Tomoaki

ClearCode Inc.

Read the full write-up

https://www.clear-code.com/blog/2026/8/19/nominatim-with-pgroonga.html

About me

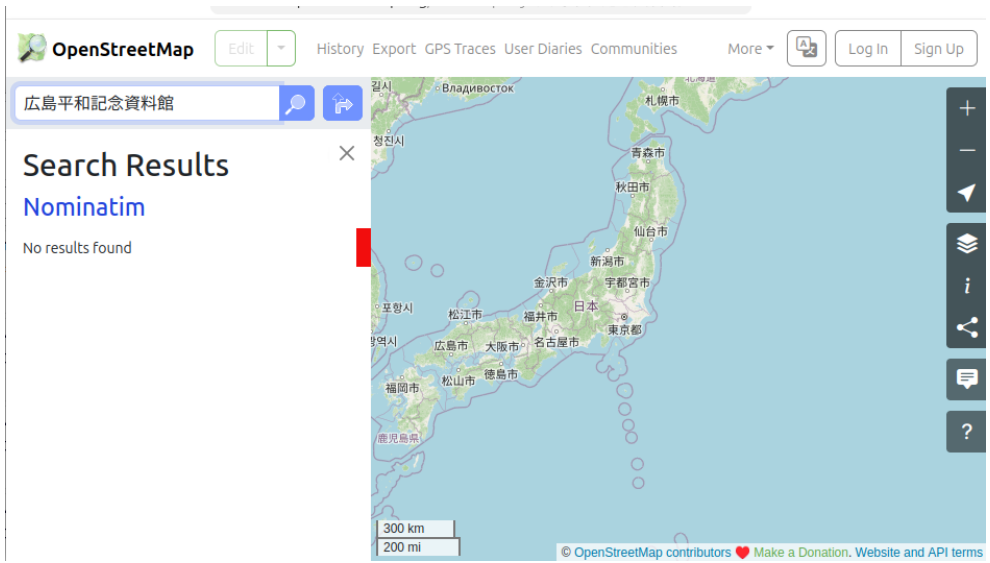
- ✓ Abe Tomoaki
  - ✓ GitHub: abetomo
- ✓ ClearCode Inc.
  - ✓ FOSS development and support
  - ✓ Developers of the full-text search engines Groonga and PGroonga

Let's try it

Search from `www.openstreetmap.org`

広島平和記念資料館

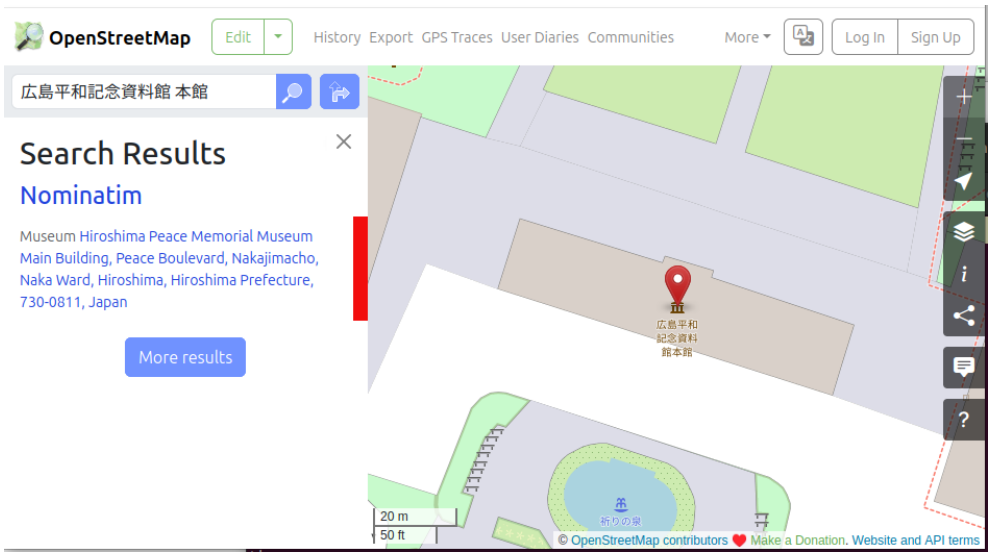
(Hiroshima Peace Memorial Museum)



Let’s try it

広島平和記念資料館 本館

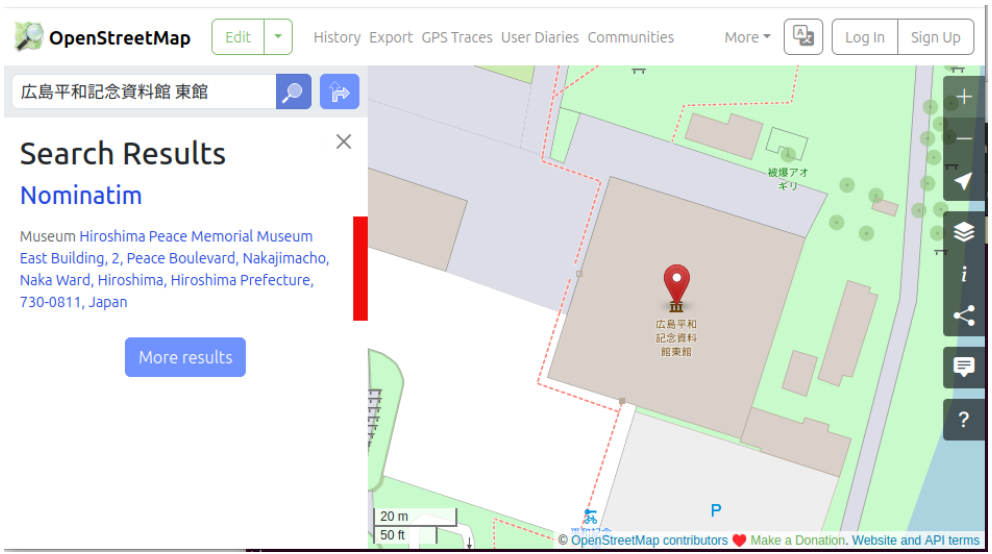
(... Main Building)



Let's try it

広島平和記念資料館 東館

(... East Building)



Results

Query	Result
広島平和記念資料館	**No hits**
広島平和記念資料館 本館	Hit
広島平和記念資料館 東館	Hit

People rarely type the building name (本館/東館) when searching. → **You never reach it.**

The problem

The data for “広島平和記念資料館 本館” exists, yet “広島平和記念資料館” finds nothing

This kind of miss comes from how Nominatim searches (explained next).

Nominatim

- ✓ The software behind openstreetmap.org search
  - ✓ Also shown in the search-result heading
- ✓ An OSM-based geocoder, multilingual
- ✓ But CJK (Chinese / Japanese / Korean) doesn’t always work as expected

How search works

- ✓ “Token-based search” = split a name into words and match them
- ✓ Both names and queries are split into words the same way

Search (partial match)

A place hits if it contains all of the query's words (= partial match)

- ✓ By its nature, a partial match can hit a lot
- ✓ For performance, various search optimizations are applied

So what is actually happening?

1. The data is “広島平和記念資料館 本館 / 東館” (official name + building name)
2. “広島平和記念資料館” should be a subset of it
3. But the internal search optimization backfires
   → 0 results (details in the appendix)

Why CJK doesn’t work well

1. The search optimization backfires
   - ✓ Tends to happen with long names / queries
2. ICU treats kanji with Chinese readings

1. The search optimization backfires

- ✓ The longer the name, the more tokens
  - ✓ The optimization backfires
  - ✓ It gets long, so details are in the appendix
- ✓ More likely in CJK, where word boundaries are unclear

2. ICU treats kanji with Chinese readings

- ✓ ICU romanizes kanji as Chinese
  - ✓ Not the Japanese reading
- ✓ Different places can end up with the same reading
  - ✓ Search precision drops

Example: li yuan

```
from icu import Transliterator
t = Transliterator.createInstance('Any-Latin; Latin-ASCII; Lower()')
for w in ['笠原','栗原','栃原','立原','梨原']:
    print(w, '->', repr(t.transliterate(w)))

# Output:
# 笠原 -> 'li yuan'
# 栗原 -> 'li yuan'
# 栃原 -> 'li yuan'
# 立原 -> 'li yuan'
# 梨原 -> 'li yuan'
```

Example: query “笠原”

```
docker compose exec -T -e NOMINATIM_PGROONGA=off nominatim nominatim search --query "笠原" | grep '"display_name"'
        "display_name": "栃原，大台町，多気郡，三重県，519-2423，日本",
        "display_name": "立原，福知山市，京都府，620-0917，日本",
        "display_name": "栃原，下市町，吉野郡，奈良県，638-0041，日本",
        "display_name": "栗原，上郡町，赤穂郡，兵庫県，678-1256，日本",
        "display_name": "栗原，度会町，度会郡，三重県，516-1238，日本",
        "display_name": "栃原，伯耆町，西伯郡，鳥取県，689-4222，日本",
        "display_name": "栃原，美咲町，久米郡，岡山県，日本",
        "display_name": "栗原，大津市，滋賀県，520-0516，日本",
        "display_name": "栗原，真庭市，岡山県，719-3153，日本",
        "display_name": "梨原，佐治町高山，佐治，鳥取市，鳥取県，689-1312，日本",
```

(Environment details come later. These are results from the demo environment for this talk.)

Search demo

Demo environment: Docker built by importing the chugoku + kansai data

```
docker compose exec \
  -e NOMINATIM_PGROONGA=off \
  nominatim \
  nominatim search --query "広島平和記念資料館"
```

0 results

Solution: full-text search

- ✓ Introduce PGroonga to do full-text search
- ✓ It’s full-text search in SQL, so it works through the ORM too
- ✓ Only 3 Python files, a few lines changed

What is PGroonga?

- ✓ A full-text search extension for PostgreSQL
  - ✓ Based on Groonga
- ✓ You can choose a CJK-capable tokenizer
  - ✓ This PoC uses the default Bigram
  - ✓ Morphological analysis (MeCab) is also available

PGroonga: easy to use

- ✓ Usable from standard SQL
- ✓ Full-text search even with LIKE
  - ✓ This PoC uses the more efficient **&@** operator

Nominatim + PGroonga

- ✓ Full-text search over `all_names`
  - ✓ Hits even on a partial match
- ✓ More places are found without adding new `alt_name` / `short_name`
  - ✓ But if the alias is completely different from the official name, data still has to be added

The change (code excerpt)

```
# src/nominatim_api/search/db_searches/place_search.py
- for lookup in self.lookups:
-   sql = sql.where(lookup.sql_condition(t))
+ lookup_conditions = [lookup.sql_condition(t) for lookup in self.lookups]
+ if self.query_text:
+     name_match_condition = t.c.all_names.op('&@')(self.query_text)
+     if lookup_conditions:
+         sql = sql.where(sa.or_(sa.and_(*lookup_conditions), name_match_condition))
+     else:
+         sql = sql.where(name_match_condition)
+ elif lookup_conditions:
+     sql = sql.where(sa.and_(*lookup_conditions))
```

Just **.op('&@')** for full-text search.

Demo with PGroonga

```
docker compose exec \
  nominatim \
  nominatim search --query "広島平和記念資料館"
```

2 results (本館 / 東館 = Main / East buildings) are returned.

Performance (speed)

Measured on chugoku + kansai (search_name: 785,292 rows):

- ✓ Extra import: ~3 min (+16% on top of the base import)
- ✓ Search for “広島平和記念資料館”: ~0.2 ms
  - ✓ Measured inside PostgreSQL

Performance (size)

- ✓ PGroonga index size: 106 MB
  - ✓ Same as the existing `name_vector`
  - ✓ Smaller than `nameaddress_vector` (389 MB)

Relatively light on both build cost and search speed, even when added on top of an existing Nominatim.

Why a PostgreSQL extension?

- ✓ Nominatim’s strength = it runs on PostgreSQL alone
- ✓ No separate external search-engine service
- ✓ Simple to operate, light to deploy, architecture unchanged

Proposal: a plugin mechanism for Nominatim

PGroonga is only one example.
There are many other useful extensions.
I’d like to propose a mechanism to plug in the extension that fits the language / use case.

pg_trgm, pg_bigm, pgvector, fuzzystrmatch, ...

The other axis: improving OSM data

- ✓ PGroonga should raise the hit rate
- ✓ But with no data, nothing will ever hit
- ✓ Technology × Data is what completes the search experience

Example: gaps in OSM data

Facility	Common name	short_name	Result
広島平和記念資料館	原爆資料館	empty	not found

Even widely-used common names are often unregistered.

“原爆資料館” differs in characters from the official name, so PGroonga can’t find it either (data must be added).

A challenge common to CJK

This time I only verified with Japanese.

Chinese and Korean should have the same problem, but the PostgreSQL + PGroonga (plugin mechanism) framework **can in principle apply to all three languages.**

Summary

- ✓ CJK search has real problems
- ✓ **With PGroonga, much of it can be filled in using existing data**
  - ✓ Runs on PostgreSQL alone, no external service

Outlook

- ✓ PGroonga is one example. A “pluggable extension” mechanism in Nominatim would make it even better
- ✓ Improve CJK search with both wheels: technology × data

Thank you

- ✓ Email: abe@clear-code.com
- ✓ This PoC (Nominatim + PGroonga)
  - ✓ https://github.com/abetomo/Nominatim (pgroonga branch)
- ✓ Demo environment (Docker Compose)
  - ✓ https://github.com/abetomo/Nominatim-docker-compose

Appendix: Background 1

Nominatim tokens: there are name tokens and address tokens.

Example for “広島平和記念資料館 本館”:

- ✓ Name: **広島 / 平和 / 記念 / 資料館 / 本館**
- ✓ Address: **平和大通り / 中島町 / 中区 / 広島市 / 広島県**

Appendix: Background 2

The query is tokenized the same way.

Example when the query is “広島平和記念資料館”:

広島 / 平和 / 記念 / 資料館

Appendix: Looks like it should hit...

- ✓ Data: **広島 / 平和 / 記念 / 資料館 / 本館**
- ✓ Query: **広島 / 平和 / 記念 / 資料館**

The data has all of the query's tokens.

Appendix: The actual search

Nominatim splits the query into a “name” part and an “address” part.

Example: query “**広島 / 平和 / 記念 / 資料館**”

✓ Name part: **資料館**

✓ Address part: **広島 / 平和 / 記念**

Appendix: The actual search (address)

As a result, the address doesn’t contain “記念”, so it doesn’t hit.

- ✓ The data’s address:
  - ✓ **平和大通り / 中島町 / 中区 / 広島市 / 広島県**
- ✓ Query (address part):
  - ✓ **広島 / 平和 / 記念**

Appendix: Search optimization

Above I showed only one split example.

In reality **many candidates are generated, and the one easiest to search is chosen**.

A hitting candidate is generated too, but it's judged too costly and dropped.
So only non-hitting candidates run → **0 results**.

Appendix: Note

The appendix explanations prioritize clarity.

So there are many imprecise descriptions. Please bear with me.