Overpass-Workshop

Nakaner

Salzburg
3. Juli 2016

Gliederung

Tags abfragen (Grundlagen)


```
1 [out:json][timeout:25];
2 node[amenity=post_box]({{bbox}});
3 out;
```

- Befehle enden mit einem Semikolon
- Tag-Abfragen in eckigen Klammern
- andere Abfragen (Benutzer, Zeitstempel, räumlich) in runden Klammern
- `(50.6,7.0,50.8,7.3)` Abfrage für dieses Rechteck
- `{{bbox}}` ist ein Overpass-Turbo-Makro
- als Nodes gemappte Briefkästen in abgefragtem Gebiet

Tags abfragen (Grundlagen)


```
1 [out:json][timeout:25];
2 node[amenity]({{bbox}});
3 out;
```

- alles mit `amenity=*`

Set

- Alles wird in einem Set gespeichert
- Default: `._`
- Sets beginnen mit einem Punkt

```
1 [out:json][timeout:25];
2 node[amenity=post_box]
3   ({{bbox}})->.briefkaesten;
4 .briefkaesten out;
```

Set


```
1 [out:json][timeout:25];
2 node[amenity=post_box]
3   ({{bbox}})->.briefkaesten;
4 node.briefkaesten[operator];
5 out;
```

- Ergebnis von amenity=post_box in *briefkaesten* schreiben
- Briefkästen mit Operator in Set _ schreiben
- Set _ ausgeben

Vereinigung (Union)


```
1 [out:json][timeout:25];
2 (
3   node[amenity]({{bbox}});
4   node[shop]({{bbox}});
5 ); //Semikolon nicht vergessen
6 node[opening_hours];
7 out;
```

- Vereinigung aus Nodes mit `amenity=*` und Nodes mit `shop=*` und in Set `_` schreiben
- Set `_` nach `opening_hours=*` filtern und in Set `_` schreiben

Recurse
Von oben nach unten


```
1 [out:json][timeout:25];
2 way[highway]({{bbox}});
3 out;
4 >;
5 out;
```

- `>` heißt *Recurse down*
  - Alle Nodes aus der Eingabe, plus
  - alle Nodes und Ways, die Mitglied einer Relation der Eingabe sind, plus
  - alle Nodes von Ways, die zu einem Way aus dem Ergebnis gehören
- `>>` heißt *Recurse down relations* – Relation, die Relationsmitglieder sind
- `<` und `<<` existieren auch

Tags, Tags und Tags
Existenz, Gleichheit und Ungleichheit

sehr ähnlich zu MapCSS

Keine gültige Abfrage!

```
1 node[shop] // shop=*
2 node[shop=bakery] // shop=bakery
3 node[shop][wheelchair] // shop=* +
    ↪ wheelchair=*
4 way[highway][surface!=cobblestone] //
    ↪ highway=* + surface!=cobblestone
```

Tags, Tags und Tags
Reguläre Ausdrücke

Reguläre Ausdrücke sind notwendig für ...

Keine gültige Abfrage!

```
1 node[shop~"^$"] // shop="" (leerer Wert)
2 node[shop!~"^Baeckerei"] // shop=*, Wert
    ↪ beginnt mit "Baeckerei"
3 node[shop~"."] // gleichbedeutend mit node
    ↪ [shop]
4 node[shop!~"."] // Nodes ohne shop-Tag
5 node[~"^name:.*$"~"stadt$"]; // Schluessel
    ↪  beginnt mit "name" (name, name:de,
    ↪ name:en), Wert endet mit "stadt"
```