# Информатика. Лекция №4. Исследование протоколов, форматов обмена информацией и языков разметки документов

2024-11-02 21:04

Status: #proccess

Tags: Информатика

# Исследование протоколов, форматов обмена информацией и языков разметки документов

## Описание синтаксиса и семантики

*Синтаксис* — форма или структура выражений, предложений и программных единиц.
*Семантика* — значение выражений, предложений и программных единиц.
*Лексема* — это мельчайшая синтаксическая единица языка (например, ?, sum, begin)
*Элементарная лексема* — например, идентификатор

*Распознаватель* - устройство распознавания считывает входную строку и решает, принадлежит ли она языку
Пример: часть компилятора, осуществляющая синтаксический анализ

*Генератор* - устройство, порождающее предложения языка

- Определить корректность конкретного предложения можно сравнив его со структурой генератора

## Формальные методы описания синтаксиса

- Форма Бэкуса-Наура (BNF) и контекстно-свободные грамматики
- Расширенная BNF
- Грамматики и распознаватели

## Форма Бэкуса-Наура

**Форма Бэкуса-Наура** (Бэкус-Науровская форма, BNF, от английского *Backus-Naur Form*) — это нотация для представления грамматик формальных языков. Она позволяет компактно и наглядно описывать правила синтаксиса языков программирования, естественных языков и других систем, где требуется строгое

определение структуры. BNF была предложена Джоном Бэкусом и Питером Науром в 1960-х годах.

## Основные компоненты BNF

BNF использует следующие базовые элементы:

1. **Нетерминалы** — категории языка, которые можно развернуть в терминальные или другие нетерминальные символы. В BNF их обычно записывают в угловых скобках, например, `<expression>`, `<term>`.
2. **Терминалы** — конкретные символы или слова, составляющие итоговую строку. Терминальные символы обычно пишут без угловых скобок, например, `+`, `*`, `(`, `)`.
3. **Оператор выбора** `|` — обозначает альтернативы в правилах. Например, правило `<digit> ::= 0 | 1 | 2 | ... | 9` говорит о том, что `<digit>` может быть любой одной цифрой от `0` до `9`.
4. **Присваивание** `::=` — символ, который связывает нетерминал с его определением. Он означает «может быть определено как».

## Пример BNF-описания

Для примера рассмотрим грамматику арифметических выражений, которые состоят из чисел и операций сложения и умножения. Определим её в BNF.

1. **Описание грамматики**:

```
<expression> ::= <term> | <expression> "+" <term>
<term> ::= <factor> | <term> "*" <factor>
<factor> ::= "(" <expression> ")" | <number>
<number> ::= <digit> | <digit> <number>
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
"9"
```

2. **Пояснение правил**:
   - `<expression>` может быть либо `<term>`, либо `<expression> "+" <term>`, то есть выражение может состоять из одного терма или из суммы термов.
   - `<term>` может быть `<factor>` или `<term> "*" <factor>`, то есть терм может быть либо фактором, либо произведением терма и фактора.
   - `<factor>` может быть выражением в скобках или числом.

- `<number>` — это последовательность одной или нескольких цифр.
- `<digit>` определяет возможные значения для одной цифры, от `0` до `9`.

3. **Разбор примера**:

   Допустим, у нас есть строка `"3 + 5 * (2 + 1)"`. Используя правила, мы можем разобрать её следующим образом:

   - Выражение `3 + 5 * (2 + 1)` соответствует `<expression>`.
   - Это `<expression>` состоит из `<term> "+" <term>`, где первый `<term>` — это `3`, а второй `<term>` — это `5 * (2 + 1)`.
   - `5 * (2 + 1)` соответствует `<term> "*" <factor>`, где `<factor>` — выражение в скобках `(2 + 1)`.

Таким образом, с помощью BNF можно описать практически любой синтаксис, задав строгие правила для допустимых строк в языке. BNF часто используется для описания грамматик языков программирования в спецификациях (например, для языков C, JavaScript).

## Формальные грамматики

Формальные грамматики используются для описания синтаксиса и структуры формальных языков. Основные принципы их организации позволяют задать точные правила для формирования строк в языке, что помогает создавать интерпретаторы и компиляторы для языков программирования, а также обрабатывать данные в других структурированных форматах. Давайте рассмотрим основные принципы.

## Определение формальной грамматики

Формальная грамматика — это набор правил, которые определяют возможные строки языка. Обычно грамматика представляется как четверка G=(N,Σ,P,S), где:

- **N** — множество нетерминальных символов (абстрактных категорий),
- **Σ** — множество терминальных символов (конкретных символов, которые появляются в итоговых строках),
- **P** — множество правил (или продукций), где каждый элемент задает, как можно заменить один символ или последовательность символов на другие,
- **S** — стартовый символ, с которого начинается разбор строки.

Пример: для языка арифметических выражений G=(N,Σ,P,S), где

N = { expression, term, factor},
Σ = {+, ∗, (, ), цифры},

P — правила, например: expression ::= term | expression + term,
S = expression.

## Типы грамматик по иерархии Хомского

Ноам Хомский разработал иерархию для классификации грамматик по их мощности и сложности:

- **Тип 0 (неограниченные грамматики)**: могут описывать любой вычислимый язык, но правила сложные и не всегда практически применимы.
- **Тип 1 (контекстно-зависимые грамматики)**: правила зависят от контекста. Пример — натуральные языки, где грамматическое значение слова зависит от других слов.
- **Тип 2 (контекстно-свободные грамматики)**: правила не зависят от контекста. Применяются для языков программирования и синтаксиса математических выражений.
- **Тип 3 (регулярные грамматики)**: самые простые грамматики. Они задаются правилами, которые позволяют создавать конечные автоматы.

## Компоненты и структура правил

Каждое правило грамматики задает способ замены символа на строку символов:

- **Левая часть** — это один нетерминал, который нужно заменить.
- **Правая часть** — строка терминалов и/или нетерминалов, на которые заменяется левая часть.

Пример:
`<expression> ::= <term> "+" <expression> | <term>`

## JSON (обозначение объектов JavaScript)

JSON — это легкий, текстовый, независимый от языка формат обмена данными, полученный из JavaScript, но используемый во многих языках программирования. JSON идеально подходит для обмена данными между сервером и веб-приложением благодаря своей простоте и удобству использования и является самым популярным современным форматом обмена данными. JSON построен на двух структурах:

- Коллекция пар имя/значение (объект в терминах JavaScript)
- Упорядоченный список значений (массив в терминах JavaScript)

В веб-приложениях JSON используется для отправки данных с сервера на клиент и наоборот. Он может представлять простые структуры данных и ассоциативные массивы, называемые объектами. Данные JSON сериализуются с

помощью `JSON.stringify()` и десериализуются с помощью `JSON.parse()` в JavaScript.

Давайте рассмотрим простой пример объекта пользователя, который включает имя, возраст и адрес электронной почты. Мы представим этот объект пользователя в различных форматах сериализации.

```
{  "name": "John Doe",  "age": 30,  "email": "johndoe@example.com"}
```

## XML (расширяемый язык разметки)

XML — это язык разметки, который определяет набор правил для кодирования документов в формате, который может быть как читаем человеком, так и читаем машиной. Он в основном используется для хранения и передачи данных. Данные XML организованы в древовидную структуру. Он позволяет вам определять собственные теги и структуру документа.

XML широко используется в корпоративных приложениях, веб-сервисах (например, SOAP) и файлах конфигурации. XML-документы можно разобрать в дерево DOM с помощью `DOMParser` и сериализовать с помощью `XMLSerializer`.

Пример:

```
<user>  <name>John Doe</name>  <age>30</age>
<email>johndoe@example.com</email></user>
```

## YAML (YAML не является языком разметки)

YAML — это формат сериализации данных, который может читать человек. Он особенно подходит для файлов конфигурации и данных, которые напрямую редактируются людьми. YAML использует нестрогий синтаксис пробелов с парами ключ-значение. Он может представлять скаляры (строки, числа), списки и ассоциативные массивы.

YAML часто используется в файлах конфигурации и для данных, требующих высокой степени удобочитаемости для человека.

Пример:

```
name: John Doeage: 30email: johndoe@example.com
```

## CSV (значения, разделенные запятыми)

CSV — это простой формат, используемый для хранения табличных данных, таких как электронные таблицы или базы данных. Он хранит данные в виде обычного текста, при этом каждая строка файла представляет собой запись данных. Каждая запись состоит из полей, разделенных запятыми. CSV обычно используется для экспорта и импорта данных в электронные таблицы или базы данных и из них.

Пример:

```
name, age, emailJohn Doe, 30, johndoe@example.com
```

## Протокольные буферы (ProtoBuf)

Разработанный Google, Protocol Buffers — это метод сериализации структурированных данных. Он полезен при разработке программ, которые взаимодействуют друг с другом по проводам или для хранения данных.

ProtoBuf используется в ситуациях, когда необходима эффективная и расширяемая сериализация данных. Он компактнее и быстрее, чем XML и JSON.

Пример:

```
message User {  string name = 1;  int32 age = 2;  string email = 3;}//
Serialized data would be in binary format
```

# References