Основы XML-технологий

Слайды основаны
на презентации

Rajshekhar Sunderraman
Институт компьютерных наук
Государственный университет штата Джорджии
Атланта, GA 30302
raj@cs.gsu.edu

Введение

- XML: W3C-стандарт
- Две грани XML: ориентация на документ и ориентация на данные
- Причины появления
  - HTML описывает представление
  - XML описывает содержимое
- Пользователь описывает теги для разметки “содержимого”
- Основан на текстовом формате
- Идеален для формата “Обмен данными”
- Ключевая технология для “распределенных” программ
- XML близок к объектно-ориентированным и так называемым *полу-структурированным* данным.

Структурированность данных в реляционной модели

Name:	Id :	Address(Number):	Address (Street):
John Doe	s11111	123	Main
Joe Public	s22222	34	Mosat

Полуструктурированные данные в HTML

HTML описывает представление.

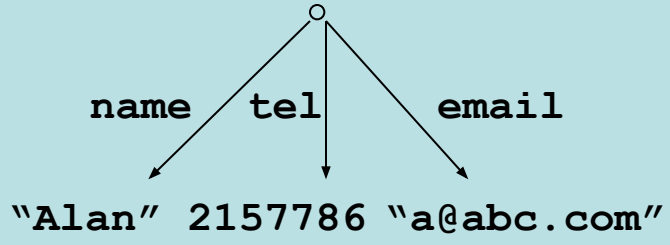
Пример HTML-документа «список студентов» для вывода на экран в Web-навигаторе

```
<dt>Name: John Doe
 <dd>Id: s111111111
 <dd>Address:
  <ul>
    <li>Number: 123</li>
    <li>Street: Main</li>
   </ul>
</dt>
<dt>Name: Joe Public
 <dd>Id: s222222222
  … … … …
</dt>
```

Модель полуструктурированных данных

- Множество пар.

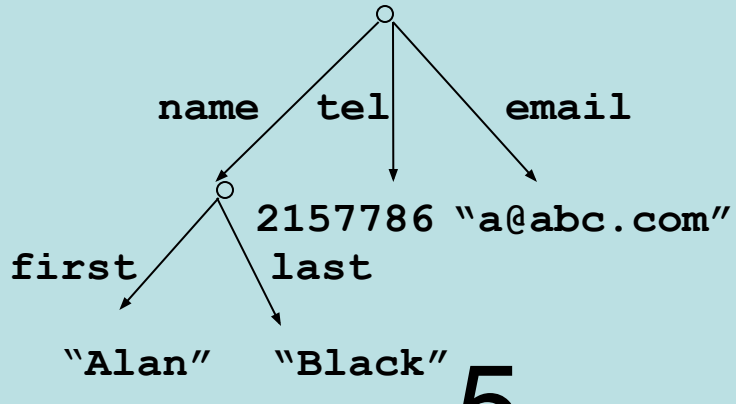
```
{name: "Alan",
 tel: 2157786,
 email: “a@abc.com”
}
```



Графовая модель: узлы представляют объекты, которые соединяются со значениями через дуги

- Значения сами могут быть структурами

```
{name: {first: “Alan”, last: “Black”},
 tel: 2157786,
 email: “a@abc.com”
}
```



Модель полуструктурированных данных

- Дубликаты допускаются

```
{name: "Alan", tel: 2157786, tel: 2498762"}
```

- Синтаксис просто генерирует описания множества объектов

```
{person: {name: “Alan”,tel: 2157786,email: “a@abc.com”}
 person: {name: “Sara”,tel: 2136877,email: “sara@abc.com”}
 person: {name: “Fred”,tel: 7786312,email: “fred@abc.com”}
}
```

- Все объекты внутри множества могут быть разной структуры

```
{person:{name: “Alan”,tel: 2157786,email: “a@abc.com”},
 person:{name: {first: “Sara”,last: “Black”},email: “s@abc.com”},
 person:{name: “Fred”, tel: 7786312, height: 168}
}
```

Модель полуструктурированных данных

- Простое представление реляционных данных

```
{r1: {row: {a: a1, b: b1, c: c1},
     {row: {a: a2, b: b2, c: c2}},
 r2: {row: {c: c2, d: d2},
      row: {c: c3, d: d3},
      row: {c: c4, d: d4}}
 }
```

- Объектно-ориентированные данные представляются естественно (каждый узел имеет уникальный идентификатор объекта)

```
{person: &o1{name: "Mary", age: 45,
             child: &o2, child: &o3},
person: &o2{name: "John", age: 17,
             relatives: {mother: &o1, sister: &o3}},
person: &o3{name: "Jane", country: "Canada", mother: &o1}
}
```

XML – Стандарт для полуструктурированных данных

- XML: eXtensible Markup Language
  - Удобен для полуструктурированных данных
  - Используется для описания содержимого, а не представления
  - Отличается от HTML т.к.
    - Автором документа могут быть определены новые теги
    - Нет семантики тегов. Например, HTML <table>…</table> означает таблицу,; in XML: не означает ничего определенного.
    - Структуры могут быть вложенными

Синтаксис XML. XML Элемент

Элемент – часть текста, ограниченная согласованными тегами, определенными пользователем:

```
<person>
  <name>Alan</name>
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

Комментарии:

- **Элемент включает открывающий и закрывающий теги**
- **Отсутствие кавычек в строках, т.к. все данные рассматриваются в виде текста. Определяются как PCDATA (Parsed Character Data – символьные данные, обрабатываемые синтаксическим анализатором).**
- **Возможность пустого элемента:**

Синтаксис XML

Коллекции определяются через повторяющиеся структуры.

Например, коллекция всех личностей, работающих на четвертом этаже:

```
<table>
 <description>People on the 4th floor</description>
 <people>
  <person>
   <name>Alan</name><age>42</age>
   <email>agb@abc.com</email>
  </person>
  <person>
   <name>Ryan</name><age>58</age>
   <email>rgz@abc.com</email>
  </person>
 </people>
</table>
```

Синтаксис XML. XML Атрибуты

- Атрибут определяет некоторые свойства элемента
- Представляется как пара “название-значение”

```
<product>
 <name language="French">trompette six trous</name>
 <price currency="Euro">420.12</price>
 <address format="XLB56" language="French">
  <street>31 rue Croix-Bosset</street>
  <zip>92310</zip>
  <city>Sevres</city>
  <country>France</country>
 </address>
</product>
```

- Внутри тега можно определить любое кол-во атрибутов
- Значения атрибутов должны быть расположены внутри двойных кавычек.

Синтаксис XML. Атрибуты или Элементы ?

- Атрибут внутри тега может появляться только один раз, его значение – всегда строка.
- Теги элемент/подэлемент могут повторяться любое кол-во раз, а их значениями могут быть строки или подэлементы
- Некоторые данные могут быть представлены с использованием атрибутов или элементов, или их комбинацией

```
<person name="Alan" age="42">
 <email>agb@abc.com</email>
</person>
```

или

```
<person name="Alan">
 <age>42</age>
 <email>agb@abc.com</email>
</person>
```

Синтаксис XML. XML Ссылки

- Использование id атрибута для определения ссылки
- Использование idref атрибута (в пустом элементе) для установления ссылки на ранее описанную ссылку (id).

```
<state id="s2">        -- определяет id или ссылку
   <scode>NE</scode>
   <sname>Nevada</sname>
</state>

<city id="c2">
  <ccode>CCN</ccode>
  <cname>Carson City</cname>
  <state-of idref="s2"/>    -- ссылается на объект s2;
</city>
```

Синтаксис XML. Другие XML конструкции

- **Комментарий:**

  <!-- this is a comment -->

- **Процессные инструкции (Processing Instruction - PI):**

  <?xml version="1.0"?>

  <?xml-stylesheet type="text/xsl" href="classes.xsl" encoding="UTF-8"?>

Такие инструкции могут обрабатываться программой, обрабатываемой XML-файл.

- **CDATA (Character Data): используется для записи особых блоков, содержащих текст с разметками, не являющимися тегами:**

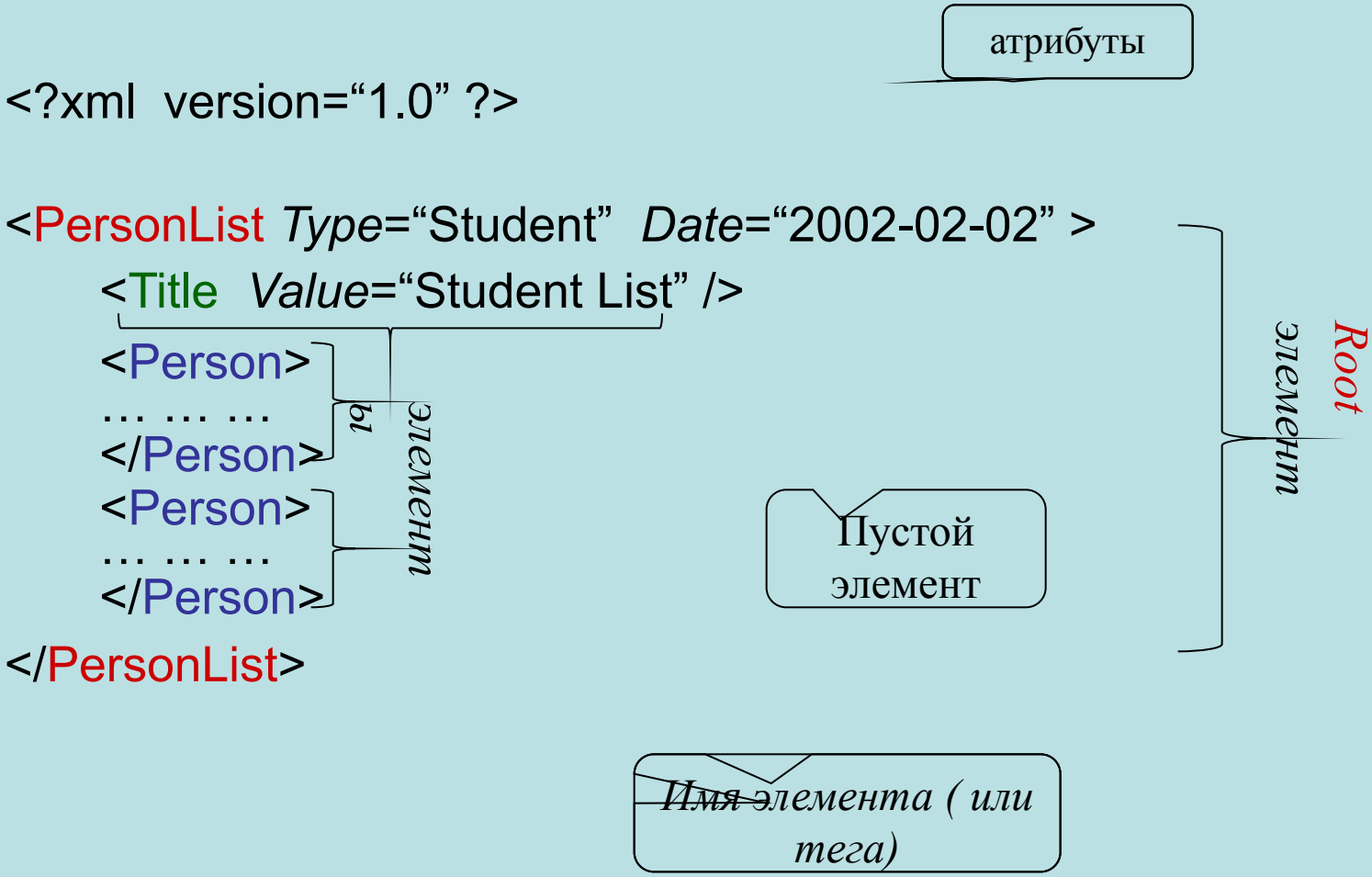
  <![CDATA[<start>this is not an element</start>]]>

- **Записи: &lt эквивалентно символу <**

Правильно созданный (Well-Formed) XML-документ

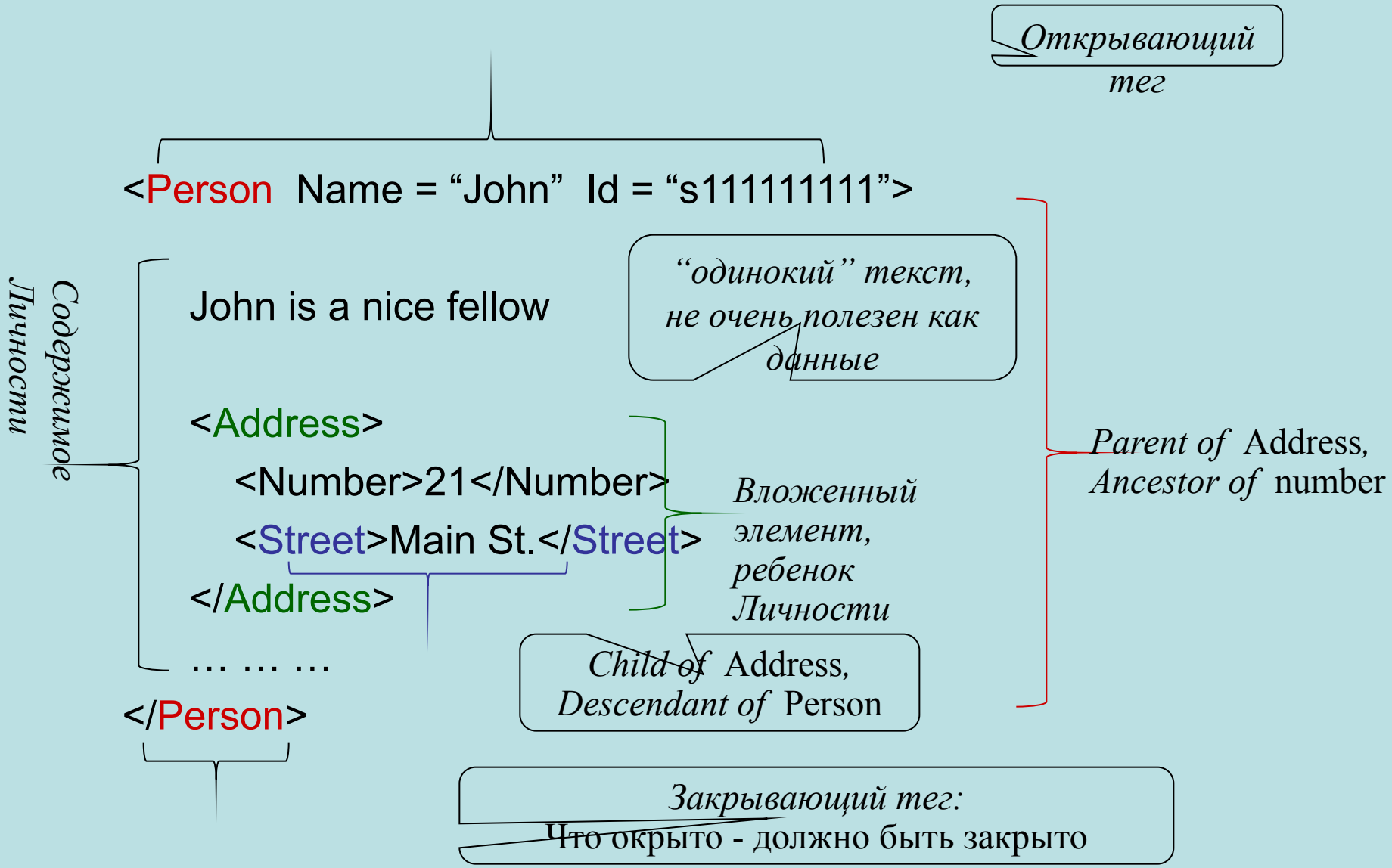
- XML-документ*well-formed* если
  - Теги синтаксически корректны
  - Каждый тег имеет закрывающий тег
  - Теги правильно вложены
  - Существует корневой (root) тег
  - В теге не может быть двух повторяющихся атрибутов
- XML-документ должен быть well-formed перед обработкой.
- well-formed XML-документ в процессе синтаксического анализа превращается в дерево узлов

Терминология

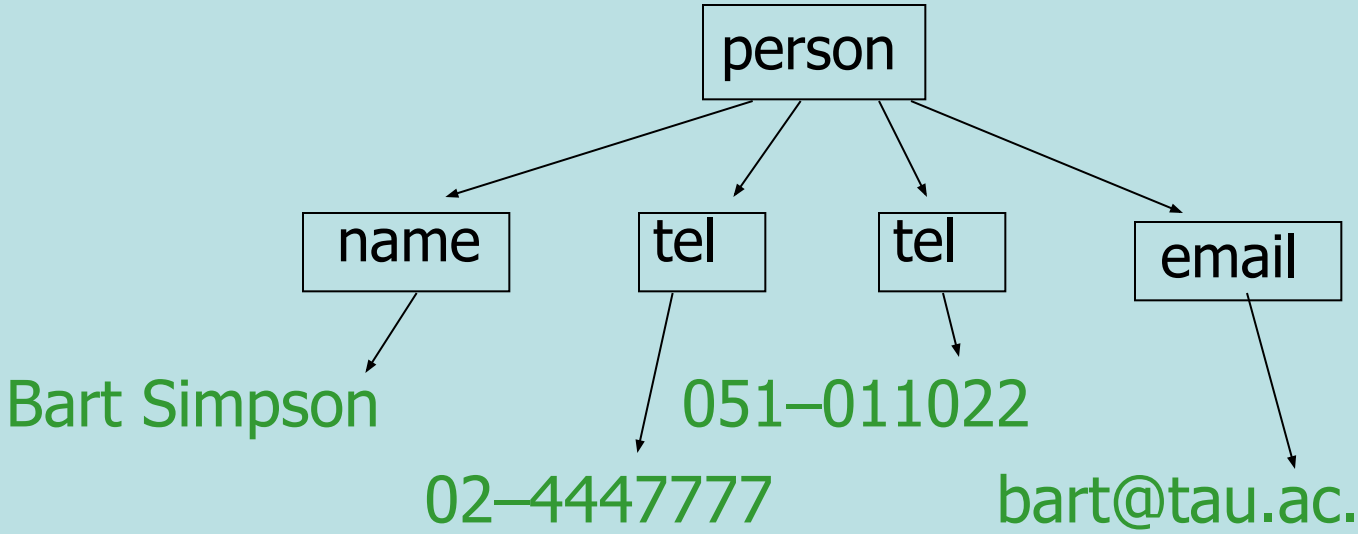


- Элементы могут быть вложенными
- Корневой элемент включает все остальные теги

Терминология



XML модель данных



- Document Object Model (DOM) – DOM Дерево
- Листья могут быть пустыми или содержать PCDATA

Пример реляционной базы данных сотрудников

Имя сотруд	Должность	Подразделение	Дата зач	Зарплата	Надбавка	Рук-тель
JONES	MANAGER	RESEARCH	02.04.81	2975		KING
CLARK	MANAGER	ACCOUNTING	09.06.81	1500		KING
BLAKE	MANAGER	SALES	01.05.81	2850		KING
WARD	SALESMAN	SALES	22.02.81	1250	500	BLAKE
JAMES	CLERK	SALES	03.12.81	950		BLAKE
TURNER	SALESMAN	SALES	08.09.81	1500	0	BLAKE
ALLEN	SALESMAN	SALES	20.02.81	1600	300	BLAKE
MARTIN	SALESMAN	SALES	28.09.81	1250	1400	BLAKE
MILLER	CLERK	ACCOUNTING	23.01.82	1300		CLARK
SCOTT	ANALYST	RESEARCH	09.12.82	3000		JONES
FORD	ANALYST	RESEARCH	03.12.81	3000		JONES
SMITH	CLERK	RESEARCH	17.12.80	800		FORD
ADAMS	CLERK	RESEARCH	12.01.83	1100		SCOTT

Пример XML базы данных сотрудников. Фрагмент

```
<!-- Example xml for emp -->
<employee>
<dept>
    <deptno id="10"/>
    <dname>dept1</dname>    <loc>UA</loc>
</dept>
<dept>
    <deptno id="10"/>   <dname>dept2</dname>    <loc>UA</loc>
</dept>
<emp empno="1010" >
    <ename>Black</ename>    <job>job1</job> <mgr></mgr>
    <hiredate>01.01.2010</hiredate> <sal>100</sal>  <comm>10</comm>
    <deptno idref="10"/>
</emp>
<emp empno="1020" >
    <ename>Joe</ename>  <job>job2</job>
    <mgr></mgr> <hiredate>02.10.2010</hiredate>
    <sal>200</sal>  <comm>15</comm>
    <deptno idref="10"/>
</emp>
</employee>
```

Document Type Definitions (DTD) – определение типа документа

- DTD: Document Type Definition – один из способов спецификации структуры XML документа.
- DTD добавляет синтаксические требования в дополнение к требованиям well-formed документа.
- DTDs помогает
  - Обнаруживать ошибки при создании или редактирования XML документов.
  - Упрощает процесс обработки XML документов.
- Использует “регулярные выражения” как синтаксис для спецификации граматики XML документа.
- Имеет ограничения: нет типов данных, нет возможности описания ограничений, нет поддержки схем.

Пример: Адресная книга

```
<person>
    <name> Homer Simpson </name>
    <greet> Dr. H. Simpson </greet>
    <addr>1234 Springwater Road </addr>
    <addr> Springfield USA, 98765 </addr>
    <tel> (321) 786 2543 </tel>
    <fax> (321) 786 2544 </fax>
    <tel> (321) 786 2544 </tel>
    <email> homer@math.springfield.edu </email>
</person>
```

Строго одно имя

Не более одного приветствия

Множество адресов(в порядке следования)

Смесь телефонов и факсов

Множество

Спецификация структуры

- name имя элемента
- greet? опционально (0 или 1) приветственных элементов
- name, greet? имя перед опциональным приветствием
- addr* для определения 0 или более адресов
- tel | fax элемент с телефоном или факс
- (tel | fax)* 0 или более повторений телефона или факса
- email* 0 или более элементов почтовых адресов

Определение типа элемента

Для каждого элемента типа E, описание формы:

<!ELEMENT E content-model>
где content-model выражение:

Content-model ::=
EMPTY | ANY | #PCDATA | E’ |
P1, P2 | P1 | P2 | P1? | P1+ | P1* | (P)

- E’ тип элемента
- P1 , P2 конкатенация
- P1 | P2 альтернатива
- P? 0 или 1 раз появлений
- P+ 1 или более появлений
- P* любое количество появлений (может отсутствовать)
- (P) группировка

XML-документ адресной книги с описанием DTD внутри самого файла

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE addressbook [
   <!ELEMENT addressbook (person*)>
   <!ELEMENT person (name, greet?, address*,
                     (fax | tel)*, email*)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT greet   (#PCDATA)>
   <!ELEMENT address (#PCDATA)>
   <!ELEMENT tel    (#PCDATA)>
   <!ELEMENT fax  (#PCDATA)>
   <!ELEMENT email   (#PCDATA)>
]>
```

Спецификация атрибута DTD

```
<!ELEMENT height (#PCDATA)>
<!ATTLIST height
      dimension CDATA #REQUIRED
      accuracy CDATA  #IMPLIED >
```

- Атрибут `dimension` обязан присутствовать
- Атрибут `accuracy` может отсутствовать
- CDATA – это тип атрибута – символьные данные

Формат определения атрибутов

```
<!ATTLIST имя_атрибута тип_атрибута
          определение_атрибута>
```

- Значение представляется внутри кавычек
- Типы атрибутов:
  - `CDATA`
  - `ID, IDREF, IDREFS`
- `ID, IDREF, IDREFS` используются для ссылок
- Определение атрибута
  - #REQUIRED: атрибут должен присутствовать
  - #IMPLIED: атрибут может отсутствовать

Включение DTD в документ

A DTD может быть

- *внутренним*
  - DTD – часть файла с документом
- *внешним*
  - DTD и документ располагаются в разных файлах
  - внешний DTD может располагаться
    - в локальной файловой системе
    - в удаленной файловой системе

Связь документа с DTD

- Внутренний DTD

```
<?xml version="1.0"?>
<!DOCTYPE db [<!ELEMENT ...> … ]>
<db> ... </db>
```

- DTD из локальной файловой системы:

```
<!DOCTYPE db SYSTEM "schema.dtd">
```

- DTD из удаленной файловой системы:

```
<!DOCTYPE db SYSTEM
"http://www.schemaauthority.com/schema.dtd">
```

Правильный (Valid) XML-документ

- well-formed XML-документ является правильным (*valid*) если он удовлетворяет своему DTD, т.е,
  - Документ удовлетворяет граматике регулярных выражений
  - Типы атрибутов корректны

XML Схема

XML Схема

XML схема определяет:

элементы из документа

- Атрибуты, появляющиеся в элементах
- Какие элементы являются вложенными
- Порядок следования вложенных элементов
- Кол-во вложенных элементов
- Пустой элемент или его содержимое в виде текста
- Значения по-умолчанию для атрибутов

Цели Схемы – определить легальные строительные блоки XML-документа как в DTD.

XML Схема – лучше DTD

XML Схема

- Проще для изучения, чем DTD
- Расширяемая для будущих расширений
- Богаче и полезнее, чем DTD
- Написана в XML
- Поддержка типов данных

Пример: Заказ товаров

```
<?xml version="1.0"?>
<shipOrder>

<shipTo>
<name>Svendson</name>
<street>Oslo St</street>
<address>400 Main</address>
<country>Norway</country>
</shipTo>
<items>
<item>
<title>Wheel</title>
<quantity>1</quantity>
<price>10.90</price>
</item>
<item>
<title>Cam</title>
<quantity>1</quantity>
<price>9.90</price>
</item>
</items>
</shipOrder>
```

XML Схема для заказа товаров

```
<xsd:schema xmlns:xsd=http://www.w3.org/1999/XMLSchema>
<xsd:element name="shipOrder" type="order"/>
<xsd:complexType name="order">
  <xsd:element name="shipTo" type="shipAddress"/>
  <xsd:element name="items" type="cdItems"/>
</xsd:complexType>
<xsd:complexType name="shipAddress">
  <xsd:element name="name“ type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="address" type="xsd:string"/>
  <xsd:element name="country" type="xsd:string"/>
</xsd:complexType>
```

XML Схема – Заказ товаров (продолжение)

```
<xsd:complexType name="cdItems">
  <xsd:element name="item" minOccurs="0"
            maxOccurs="unbounded" type="cdItem"/>
</xsd:complexType>
<xsd:complexType name="cdItem">
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="quantity“
            type="xsd:positiveInteger"/>
  <xsd:element name="price" type="xsd:decimal"/>
</xsd:complexType>
</xsd:schema>
```

Новые простые типы данных

Доступно множество встроенных типов

xsd:string, xsd:integer, xsd:positiveInteger,
xsd:decimal, xsd:boolean, xsd:date, xsd:NMTOKENS, etc.

Определение новых простых типов. Пример определяет myInteger (значение между 10000 и 99999):

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Новые простые типы данных

Перечислимый тип:

```
<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

Новые простые типы данных

XML Схема имеет 3 встроенных списочных типа:
NMTOKENS, IDREFS, ENTITIES

Создание нового списочного типа из простого:

```
<xsd:simpleType name="listOfMyIntType">
  <xsd:list itemType="myInteger"/>
</xsd:simpleType>
```

Следующий XML-фрагмент удовлетворяет вышеописанному типу SimpleType:

```
<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

Новые простые типы данных

Использование функций: length,minLength, maxLength, enumeration

Например, для определения ровно 6-ти штатов (SixUSStates)

- Вначале определяется новый списочный тип данных USStateList из типа USState
- Затем SixUSStates ограничивается через USStateList с использованием только 6 значений

```
<xsd:simpleType name="USStateList">
  <xsd:list itemType="USState"/>
</xsd:simpleType>
<xsd:simpleType name="SixUSStates">
  <xsd:restriction base="USStateList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
<sixStates>PA NY CA NY LA AK</sixStates>
```

Лексические анализаторы – Parsers

Parsers (анализаторы)

Что такое parser?

- **Программа, которая анализирует грамматические структуры в соответствии с заданной формальной грамматикой**

- Parser определяет как предложение может быть сконструировано из грамматики языка через описание атмарных элементов и отношений между ними

XML-стандарты анализаторов

В основном рассматриваются два метода, реализованный организацией W3C для доступа к XML

SAX (Simple API for XML) – простое API для XML

- Событийно управляемый анализатор
- Протокол “последовательного доступа”
- API “только для чтения”

DOM (Document Object Model) – модель объекта документа

- Преобразовывает XML в дерево объектов
- Протокол “случайного доступа”
- Может обновлять XML-документ (insert/delete узлы)

SAX-анализатор

- SAX = Simple API for XML
- XML читается последовательно
- Когда поисходит событие анализа, анализатор вызывает соответствующий метод
- Похож на I/O-потоки, работает в одном направлении

Простой пример данных

Данные заказа XML: несколько заказов, в каждом несколько пунктов, каждый пункт включает номер и кол-во

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC 94</recDate>
 <items>
  <item>
   <pnum>10506</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10507</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10508</pnum>
   <quantity>2</quantity>
  </item>
  <item>
   <pnum>10509</pnum>
   <quantity>3</quantity>
  </item>
 </items>
</order>
... </orders>
```

Простой пример данных

Данные заказа XML:
несколько заказов с несколькими пунктами
каждый пункт включает номер и кол-во

События анализатора

startDocument

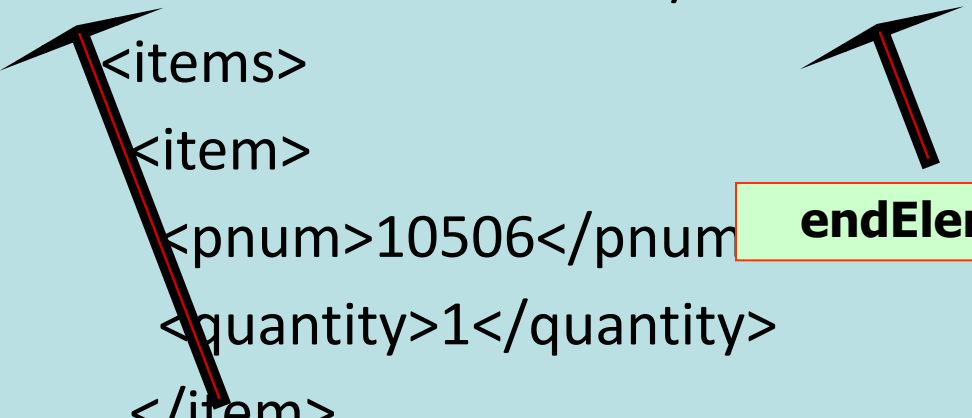
```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC 94</recDate>
 <items>
  <item>
   <pnum>10506</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10507</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10508</pnum>
   <quantity>2</quantity>
  </item>
  <item>
   <pnum>10509</pnum>
   <quantity>3</quantity>
  </item>
 </items>
</order> ... </orders>
```

endDocument

Простой пример данных

Данные заказа XML: несколько заказов, в каждом несколько пунктов, каждый пункт включает номер и кол-во

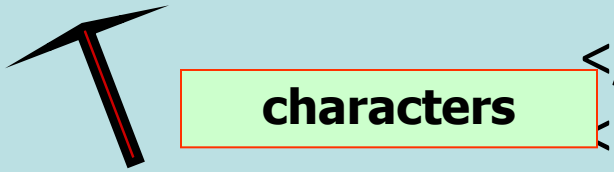
```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC-94</recDate>
  <items>
   <item>
    <pnum>10506</pnum>
    <quantity>1</quantity>
   </item>
   <item>
    <pnum>10507</pnum>
    <quantity>1</quantity>
   </item>
   <item>
    <pnum>10508</pnum>
    <quantity>2</quantity>
   </item>
   <item>
    <pnum>10509</pnum>
    <quantity>3</quantity>
   </item>
  </items>
</order> ... </orders>
```



Простой пример данных

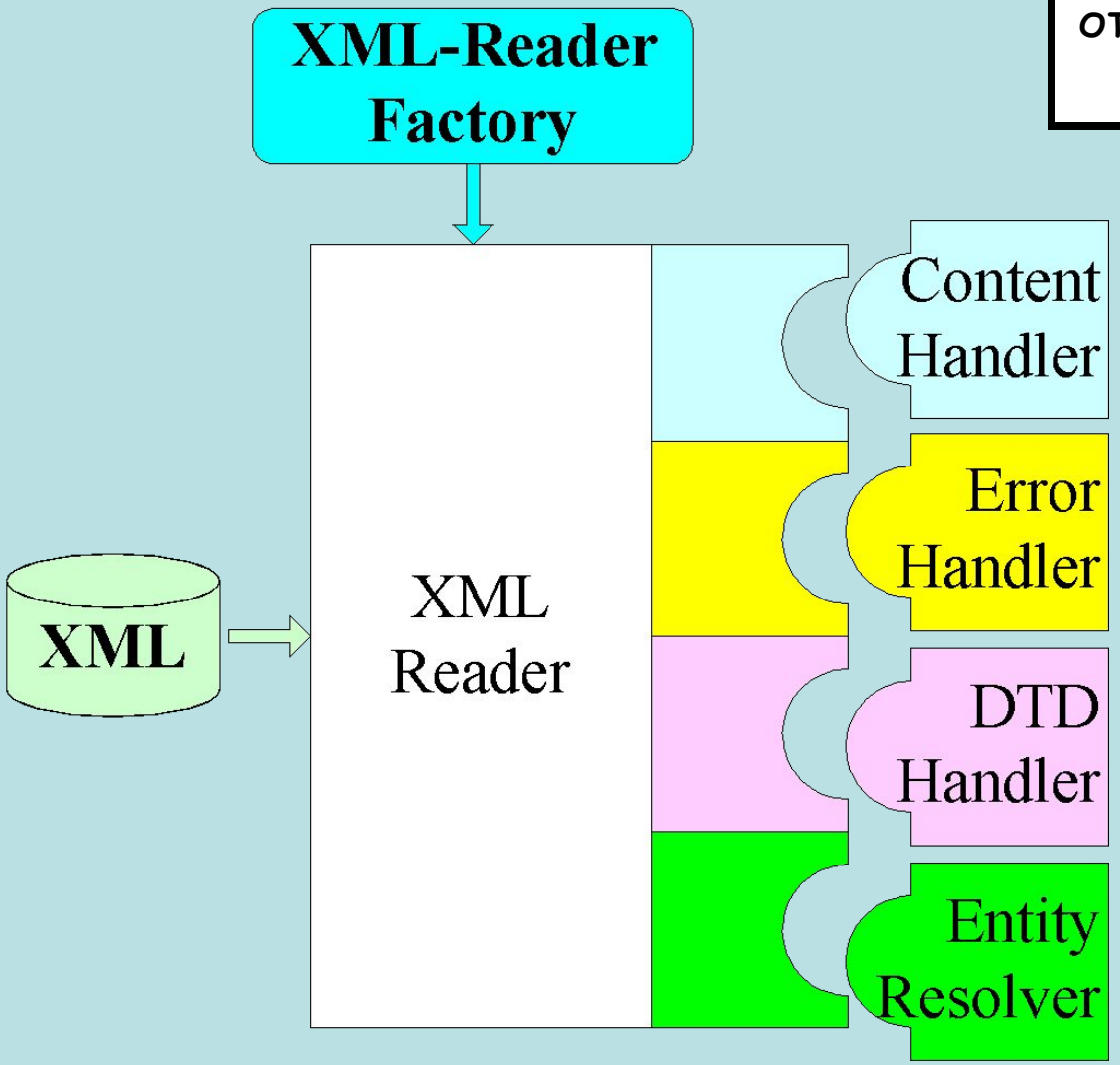
Данные заказа XML: несколько заказов, в каждом несколько пунктов, каждый пункт включает номер и кол-во

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC-94</recDate>
 <items>
  <item>
   <pnum>10506</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10507</pnum>
   <quantity>1</quantity>
  </item>
  <item>
   <pnum>10508</pnum>
   <quantity>2</quantity>
  </item>
  <item>
   <pnum>10509</pnum>
   <quantity>3</quantity>
  </item>
 </items>
</order> ... </orders>
```



SAX-анализатор

Используется для создания SAX-анализатора



Элементы управления событиями документа: открывающий/закрывающий тег ..

Элементы управления ошибок анализа

Элементы управления DTD

Элементы управления

SAX API

Два важных класса в SAX API: SAXParser и HandlerBase.

Создание нового SAXParser-объекта:

```
public SAXParser()
```

Регистрация SAX-элемента управления для объекта анализа для получения извещений о событиях анализа:

```
public void setDocumentHandler(DocumentHandler h)
```

Регистрация элемента управления для обнаружения ошибок:

```
public void setErrorHandler(ErrorHandler h)
```

SAX API

- Класс HandlerBase определяет базовый класс для всех элементов управления.
- Он определяет поведение по-умолчанию для различных элементов управления.
- Программы расширяют этот класс за счет переопределения следующих методов управления событиями:

```
public void startDocument() throws SAXException
public void endDocument() throws SAXException
public void startElement() throws SAXException
public void endElement() throws SAXException
public void characters() throws SAXException
public void warning() throws SAXException
public void error() throws SAXException
```

Создание SAX-анализатора

```
import org.xml.sax.*;
import oracle.xml.parser.v2.SAXParser;
public class SampleApp extends HandlerBase {
  // Global variables declared here
  static public void main(String [] args){
    Parser parser = new SAXParser();
    SampleApp app = new SampleApp();
    parser.setDocumentHandler(app);
    parser.setErrorHandler(app);
    try {
      parser.parse(createURL(args[0]).toString());
    } catch (SAXParseException e) {
        System.out.println(e.getMessage());
      }
  }
}
```

SAX API – простой пример программного кода

Создать SAX-анализатор, который читает файл orders.xml и извлекает различные данные, а в завершение создает запрос на внесение полученных данных в таблицу БД.

```
//Global Variables
Vector itemNum = new Vector();
int numberOfRows, numberOfItems;
String elementEncountered, orderNumber, takenBy,
       customer, receivedDate, partNumber, quantity;
//elementEncountered holds most recent element name

public void startDocument() throws SAXException {
  //Print SQL comment, initialize variable
  System.out.println("--Start of SQL insert Statements");
  itemNum.setSize(1);
  numberOfRows = 0;
}//end startDocument
```

SAX API – простой пример программного кода

```
public void startElement(String name,
                         AttributeList atts) throws SAXException {
  elementEncountered = name;
  if (name.equalsIgnoreCase("items")) {
    numberOfItems = 0;
  }//end if statement
}//end startElement

public void characters(char[] cbuf, int start, int len) {
  if (elementEncountered.equals("orderNumber")
    orderNumber = new String(cbuf,start,len);
  else if(elementEncountered.equals("takenBy")) {
    takenBy = new String(cbuf,start,len);
  ...
}//end characters
```

SAX API – простой пример программного кода

```
public void endElement(String name) throws SAXException{
  if (name.equalsIgnoreCase("item")) {
    numberOfItems++;
    if (numberOfItems == 1) { // first item; create orders row
      System.out.println(
        "insert into orders values('"+
           orderNumber +"','"+ customer +"','"+
           takenBy +"','"+ receivedDate +"','null');");
      }
      System.out.println("insert into odetails values('"+
          orderNumber +"','"+ partNumber +"','"+
          quantity +"');");
  }//end if statement
  if (name.equalsIgnoreCase("items")) {
    System.out.println("--------");
  }
}//end endElement
```

SAX API – простой пример программного кода

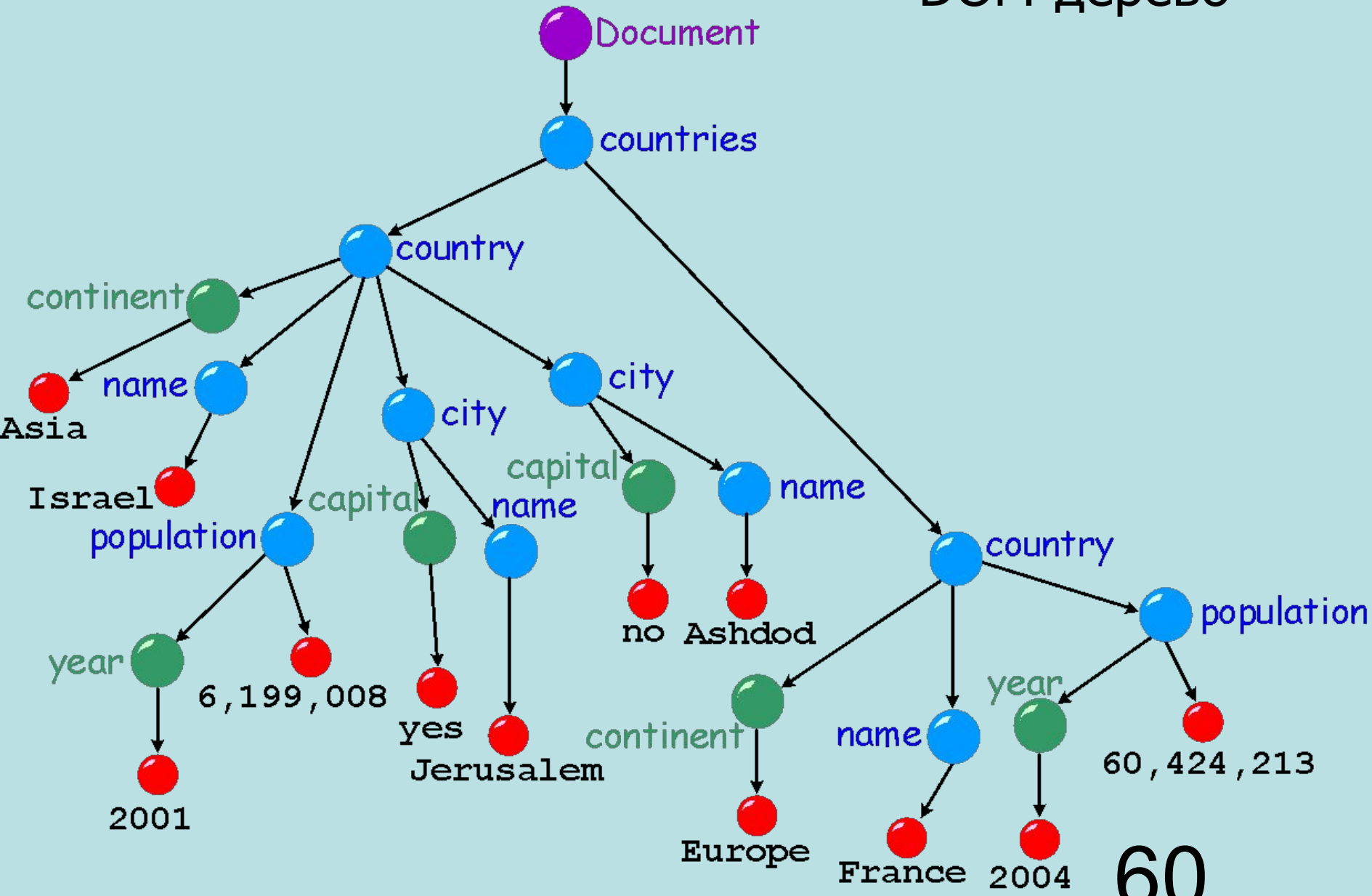
```
public void endDocument(){
  System.out.println("End of SQL insert statements.");
}//end endDocument

public void warning(SAXParseException e)
               throws SAXException {
  System.out.println("Warning:"+ e.getMessage());
}

public void error(SAXParseException e)
               throws SAXException{
  throw new SAXException(e.getMessage());
}
```

DOM-анализатор

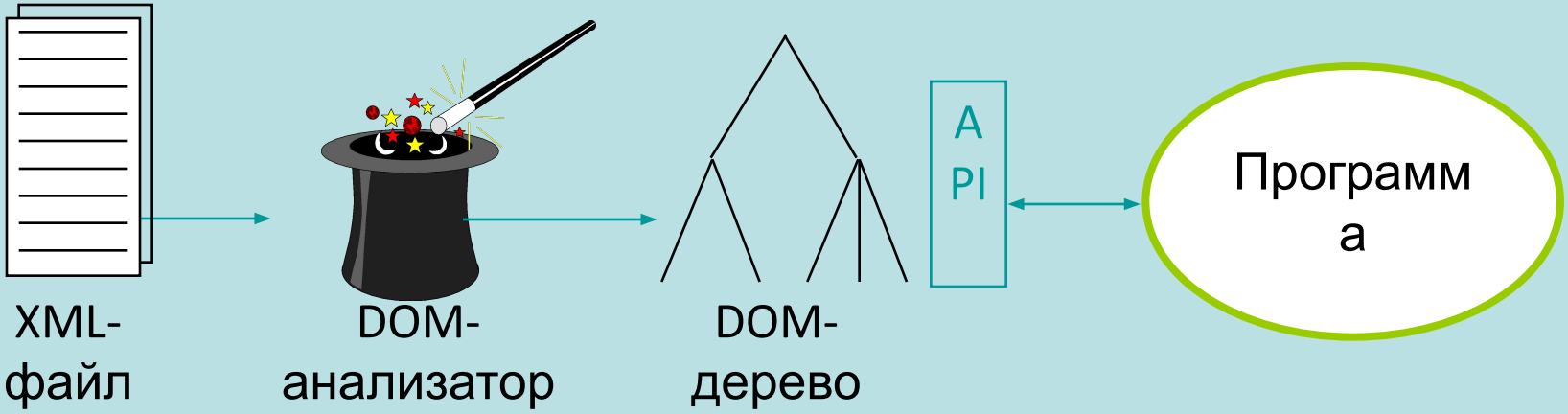
- DOM = Document Object Model (Объектная Модель Документа)
- Анализатор создает дерево объектов документа
- Пользователь получает доступ к данным путем обхода дерева
  - Дерево и его обход определо W3C-стандартами
- API позволяет конструировать, получать доступ и манипулировать структурами и содержимым XML-документов

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60,424,213</population>
  </country>
</countries>
```

DOM-дерево


Document
countries
country
continent
Asia
name
Israel
population
year
2001
6,199,008
capital
city
name
yes
Jerusalem
city
capital
name
no
Ashdod
name
country
continent
Europe
name
France
year
2004
population
60,424,213

Использование DOM-дерева



Интерфейс доступа к узлам

- Узлы DOM-дерева включают
  - Специальный корневой узел (root)
  - Узлы элементы (element)
  - Текстовые узлы и CDATA-секции
  - Атрибуты (attributes)
  - Комментарии (comments)

- Каждый узел в DOM-дереве реализует интерфейс узла

Навигация по узлам

Каждый узел имеет специальное расположение (location) в узле

Узловой (Node) интерфейс определяет методы для навигации по дереву

Node getFirstChild(); – получение первого по порядку наследника

Node getLastChild(); –- получение последнего по порядку наследника

Node getNextSibling(); –- получение следующего родственника текущего уровня (брата)

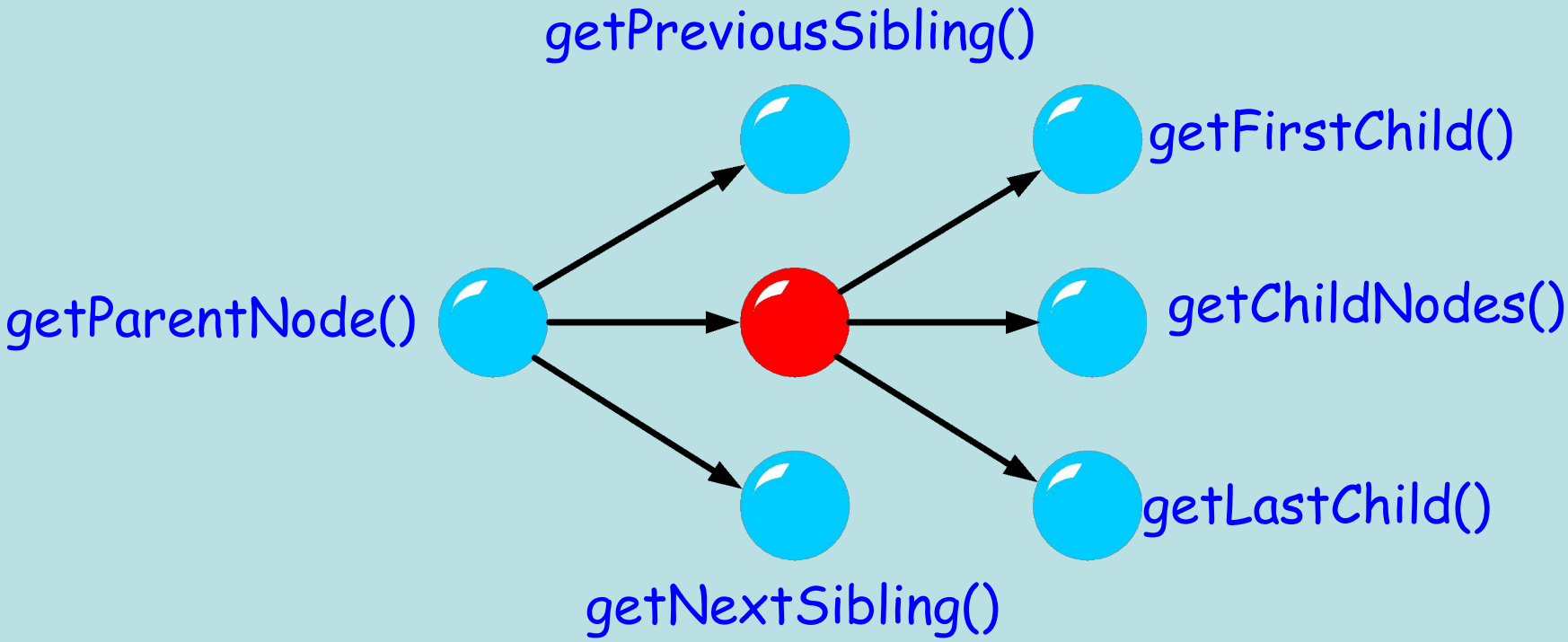
Node getPreviousSibling(); –- получение предыдущего родственника текущего уровня (брата)

Node getParentNode(); –- получение родительского узла

NodeList getChildNodes(); –- получение узлов-наследников

NamedNodeMap getAttributes() –- получение атрибутов

Навигация по узлам



Пример DOM-анализатора

Рассмотрим XML-данные описания географической информации по штатам США

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE geography SYSTEM "geo.dtd">
<geography>
  <state id="georgia">
    <scode>GA</scode>
    <sname>Georgia</sname>
    <capital idref="atlanta"/>
    <citiesin idref="atlanta"/>
    <citiesin idref="columbus"/>
    <citiesin idref="savannah"/>
    <citiesin idref="macon"/>
    <nickname>Peach State</nickname>
    <population>6478216</population>
  </state>
  ...
  <city id="atlanta">
    <ccode>ATL</ccode>
    <cname>Atlanta</cname>
    <stateof idref="georgia"/>
  </city>
  ...
</geography>
```

DTD для географических XML-данных

```
<!ELEMENT geography (state|city)*>
<!ELEMENT state (scode, sname, capital,
                 citiesin*, nickname, population)>
   <!ATTLIST state id ID #REQUIRED>
<!ELEMENT scode (#PCDATA)>
<!ELEMENT sname (#PCDATA)>
<!ELEMENT capital EMPTY>
   <!ATTLIST capital idref IDREF #REQUIRED>
<!ELEMENT citiesin EMPTY>
   <!ATTLIST citiesin idref IDREF #REQUIRED>
<!ELEMENT nickname (#PCDATA)>
<!ELEMENT population (#PCDATA)>
<!ELEMENT city (ccode, cname, stateof)>
   <!ATTLIST city id ID #IMPLIED>
<!ELEMENT ccode (#PCDATA)>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT stateof EMPTY>
   <!ATTLIST stateof idref IDREF #REQUIRED>
```

Географические данные: Структура БД (Oracle)

```
create type city_type as object (
  ccode      varchar2(15),
  cname      varchar2(50)
);

create type cities_in_table as table of city_type;

create table state (
  scode      varchar2(15),
  sname      varchar2(50),
  nickname   varchar2(100),
  population number(30),
  capital    city_type,
  cities_in  cities_in_table,
  primary key (scode))
nested table cities_in store as cities_tab;
```

Создание DOM-анализатора объектов

```
import org.w3c.dom.*;
import org.w3c.dom.Node;
import oracle.xml.parser.v2.*;

public class CreateGeoData {
  static public void main(String[] argv)
      throws SQLException {
  // Get an instance of the parser
  DOMParser parser = new DOMParser();

  // Set various parser options: validation on,
  // warnings shown, error stream set to stderr.
  parser.setErrorStream(System.err);
  parser.setValidationMode(true);
  parser.showWarnings(true);

  // Parse the document.
  parser.parse(url);

  // Obtain the document.
  XMLDocument doc = parser.getDocument();
```

Навигация по DOM-дереву

```
NodeList sl = doc.getElementsByTagName("state");
NodeList cl = doc.getElementsByTagName("city");


XMLNode e = (XMLNode) sl.item(j);
scode = e.valueOf("scode");
sname = e.valueOf("sname");
nickname = e.valueOf("nickname");
population = Long.parseLong(e.valueOf("population"));

XMLNode child = (XMLNode) e.getFirstChild();
while (child != null) {
  if (child.getNodeName().equals("capital"))
     break;
  child = (XMLNode) child.getNextSibling();
}

NamedNodeMap nnm = child.getAttributes();
XMLNode n = (XMLNode) nnm.item(0);
```

Манипулирование узлами

- Наследник узла в DOM-дереве может быть добавлен, изменен, удален, перемещен, скопирован и ...
- Для создания новых узлов используется методы класса Document
  - createElement, createAttribute, createTextNode, createCDATASection , ...
- Для манипулирования узлами используются методы класса Node
  - appendChild, insertBefore, removeChild, replaceChild, setNodeValue, cloneNode(boolean deep) ...

Сравнение SAX и DOM анализаторов: Эффективность

- DOM-объект, создаваемые DOM-анализатором являются сложными и требуют больше памяти для хранения, чем сам XML-файл
  - Затрачивается много времени для предварительного создания
  - Для больших документов это не практично
- SAX-анализаторы сохраняют только локальную информацию, которая учитывается в течение нескольких переходов по дереву
- программирование SAX-анализаторов, в общем, является эффективным способом (но не быстрым)

Трудности программирования в SAX-анализаторе

- Программирование в SAX-анализаторе сложно, например:
  - Как найти элемент *e1,* у которого предком является *e2*?
  - Как найти элемент *e1* , который имеет элемент-наследник (descendant) *e2*?
  - Как найти элемент *e1* ссылающийся на атрибут IDREF attribute элемента *e2*?

Навигация по дереву

- SAX-анализаторы предоставялют доступ к элементам только через последовательное посещение узлов

SAX-анализатор не читает в обратную сторону

- DOM-анализатор может использовать множество методов навигации

- Поэтому использование DOM-анализатора удобнее

Спасибо за Ваше внимание!

Особая благодарность
Rajshekhar Sunderraman
(Институт компьютерных наук,
Государственный университет
штата Джорджии)