语义网与知识图谱

上海大学计算机学院

主讲： 刘 炜

《语义网与知识图谱》

本体论与OWL语言

上海大学计算机学院 刘炜

2020年9月

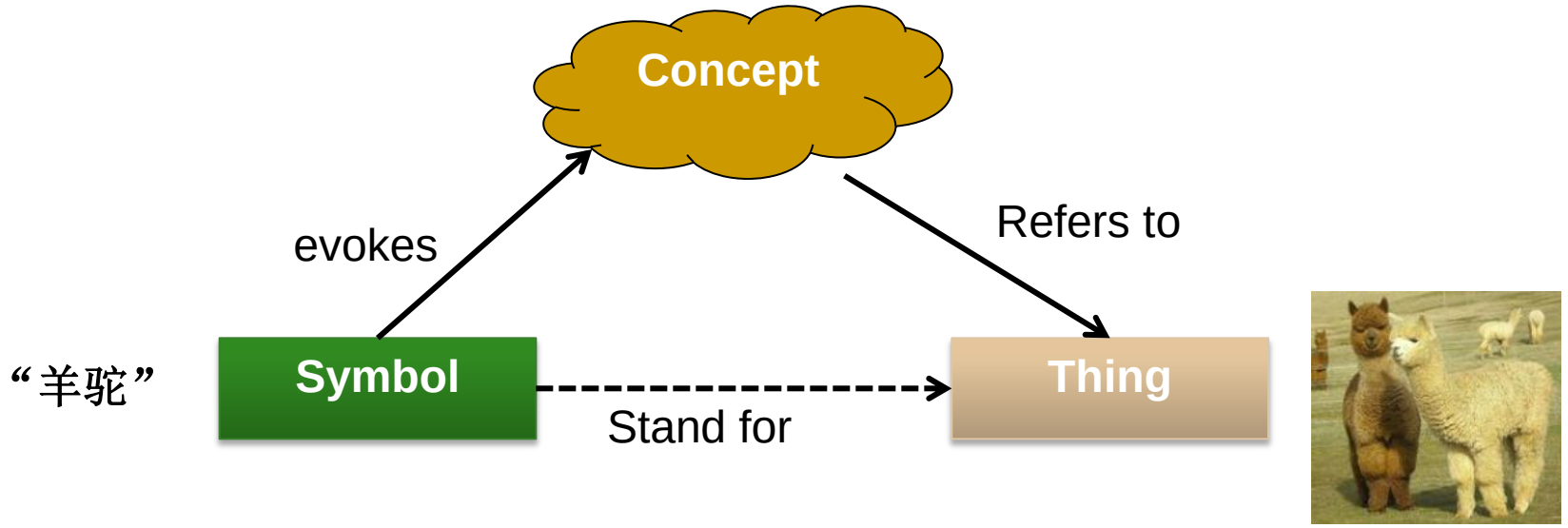
内容概览

一、本体论

二、OWL语言

1. 回顾RDF
2. OWL概述
3. OWL语言大纲
4. OWL文档结构

本体论



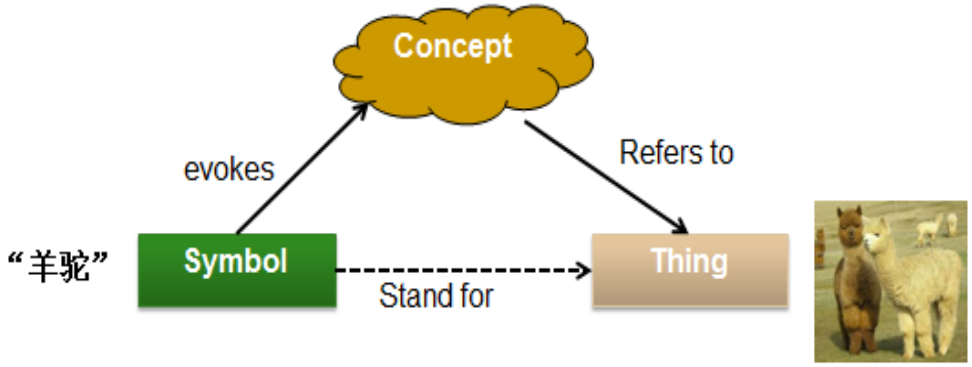
语义表达三角形：

概念（Concept）、符号（Symbol）和实体（Thing/Instance）

本体论

- **概念**是抽象层面的泛指，通常是对一类事物共性的总结；它代表的总是这一类事物的总体，而非特指其中的一个具体事物。
- **符号**，亦即概念的名称，是用以表征概念的标记（可以是文字或图形）。在交流中，人们通常都是通过符号来传达概念及其含义；在此意义上，符号是概念的载体。
- **实体**是概念的具体化、实例化表现，亦即个别化的概念。概念所对应的所有实体构成概念的外延。

建立在文字而非语意的层面上的交流往往带来很多问题。Ontology的提出在根本上是由于交流的需要而产生的。



本体论

- Ontology本是哲学领域的概念，源自哲学之中称为“形而上学”的分支，形而上学所关注的是现实的本质，也就是存在的本质。后被计算机界的研究者借用到信息领域，以描述用于交流和共享的语义空间。
- 信息领域：Ontology 是一种用以描述语义的、概念化的显式说明（an explicit specification of a conceptualization. Gruber, 93）。它是对某一个领域内事物的共同理解（a shared understanding）。

- Ontolgoy 在本质上是对**客观世界的一种分类描述**，因此，Ontology 模型实际上就是一种知识表示模型。

本体论

Ontology是建立在这样一种基于分类的概念层次。其通常由如下部分构成：

- 1) 概念：抽象层次的Class;
- 2) 名义：概念的名字，是用以传达概念的标识和媒介；
- 3) 属性：**概念之间通过属性相互区别并相互联系**；同一类的实例也通过属性值的不同而相互区别。
- 4) 取值：通常需要对**属性的类型、值域**等加以限制，以保证属性的有效性；
- 5) 公理：人们默认已经达成共识且正确的逻辑，是逻辑推理的起点和判断逻辑正确性的依据。

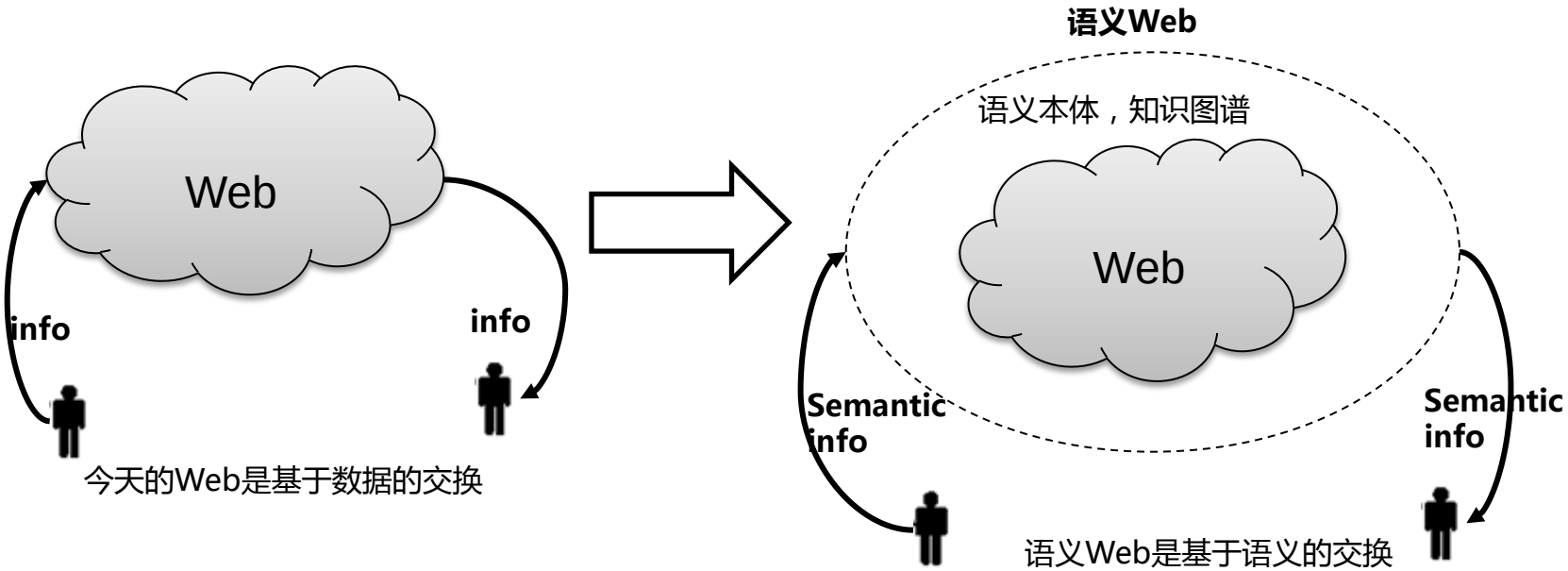
本体(Ontology) = 概念(Concept) + 属性(Property) + 公理(Axiom) + 取值(Value) + 命名(Nominal)

本体论

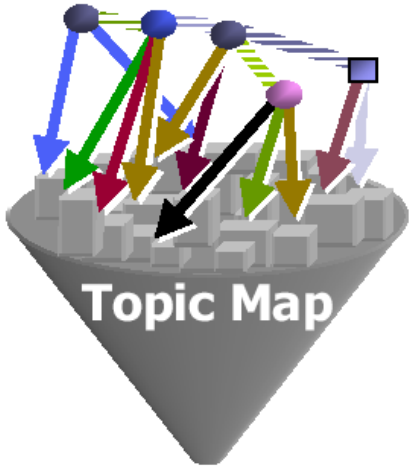
- 当前网络的缺点：
  - 超级链接错综复杂，容易迷失在网状链接中
  - 几乎不能自动处理信息（机器理解、推理能力）
  - 以文字匹配为基础的搜索引擎往往找到大量无关信息
  - HTML不包含机器能阅读的语义信息

- 语义互联网并非独立的另一个Web，而是今天Web的一个延伸。在语义互联网中，**信息被赋予明确而完整的含义**（即语义）；机器可以识别并理解这种语义，从而对Web 中的信息实现自动化采集、分割、组合乃至逻辑推理等等。

本体论


Web
info
info
今天的Web是基于数据的交换
语义Web
语义本体，知识图谱
Web
Semantic info
Semantic info
语义Web是基于语义的交换


Web links


Topic Map

本体论

创建Semantic Web的过程：

1. 对问题领域进行分析，确定语义互联网能够解决问题；
2. 选择或创建Ontology。如果问题领域已经有了领域内的标准化Ontology，则应尽可能选择使用标准化的Ontology；否则，需要根据具体的领域知识创建新的Ontology。
3. 创建新的Ontology 包括以下过程：
    1. 定义Ontology中的classes（这些classes是对领域知识的抽象描述）；
    2. 使用子类-父类（subclass-superclass）继承关系组织这些classes 的层次结构（taxonomy）；
    3. 为class定义属性（slots），并描述属性的约束（包括值的类型、取值范围、可出现次数等）。
4. 依据选择或建立的Ontology，创建对象实例（instances）。
5. 依据需求建立应用程序和服务。

内容概览

一、本体论

二、OWL语言

1. 回顾RDF
2. OWL概述
3. OWL语言大纲
4. OWL文档结构

回顾

- **用于信息交换的语言主要需要包含三个部分：**
  - **数据模型，语义，语法**

- **在目前的框架中，这些部分分别由这些标准构成：**
  - **数据模型：**RDF（Resource Description Framework）
  - **语义层：**RDFS（RDF Schema）,OWL（Web Ontology Language）
  - **语法：Turtle / N-Triples/ RDF-XML**

- **另：RDF有时会作为上述全部的统称**

回顾-RDF

- RDF以图为中心，元素少，使用简单，主要使用下列元素来表示知识：
  资源、属性、图（三元组）
- **资源：**是用以描述需要特定标识的一个实体或者概念的方式，主要通过URI(Uniform Resource Identifier)的形式表现。在三元组里表示主语和宾语。
  例：915房间的冰箱可以标识为：
  http://shu.edu.cn/ces/daselab/room915#fridge
- **属性：**一种特殊的资源，用以描述资源之间的关系，或者一个资源具有的用数据表示的特征。在三元组里表示谓语
  例：成员关系在SKOS中可以描述为：（SKOS是一个词汇表)
  http://www.w3.org/2004/02/skos/core#member

回顾-RDF

图：使用RDF建立知识模型的结构，这里的图是指有向图，资源类比于图中的结点，属性类比于图中的边。

三元组：使用文本表示RDF图的方式，三元组的三元分别为

主体（object），谓词（predicate），客体（subject）

主体客体对应图中的结点，谓词对应边。

< :上海大学计算机学院 > < :成员 > < :上海大学 >.
< :类脑实验室 > < :成员 > < :上海大学计算机学院 >.

回顾-RDFS

- RDFS用以给RDF添加语义，而添加语义的方式主要是通过词汇，所以RDFS中的主要内容是词汇表。
- RDFS词汇表主要可以分成两大类：类、属性
- 类可以理解为元素的集合，集合中的元素称为类的instance（实例、个体、实体）。
- 属性与RDF中的属性一致。

回顾-RDFS

- **RDFS中的类：**

在RDFS中，使用谓词< rdf:type >来表示类与实例之间的关系，比如：

:上海大学 < rdf:type > :学校.

< rdfs:Resource >：所有资源的类，所有RDF定义的事物都属于此

:学校 < rdf:subClassOf > < rdfs:Resource >.

< rdfs:Class >：所有类的类

:学校 < rdf:type > < rdfs:Class >.

< rdfs:Literal >：所有文字（数字、字符串、IRI等）的类

< rdfs:Datatype >：所有数据类型的类

xsd:integer < rdf:type > < rdfs:Datatype >.

< rdf:Property >：所有属性的类

< rdf:type > < rdf:type > < rdf:Property >.

注：这里加<　>是为了更清楚的标识，并不在语言规范之内。

回顾-RDFS

- **RDFS中的属性：**

属性是一种特殊的资源，可以用于对属性进行描述：

< rdfs:domain >：描述属性与其定义域的关系
< :就读 > < rdfs:domain > < :学生 >.

< rdfs:range >：描述属性与其值域的关系
< :就读 > < rdfs:range > < :学校 >.

< rdfs:subClassOf >：描述父子类的关系
< rdfs:Datatype > < rdfs:subClassOf > < rdfs:Literal >.

< rdfs:subPropertyOf >：描述父子属性之间的关系
< :租住 > < rdfs:subPropertyOf > < :居住 >.

回顾-RDFS

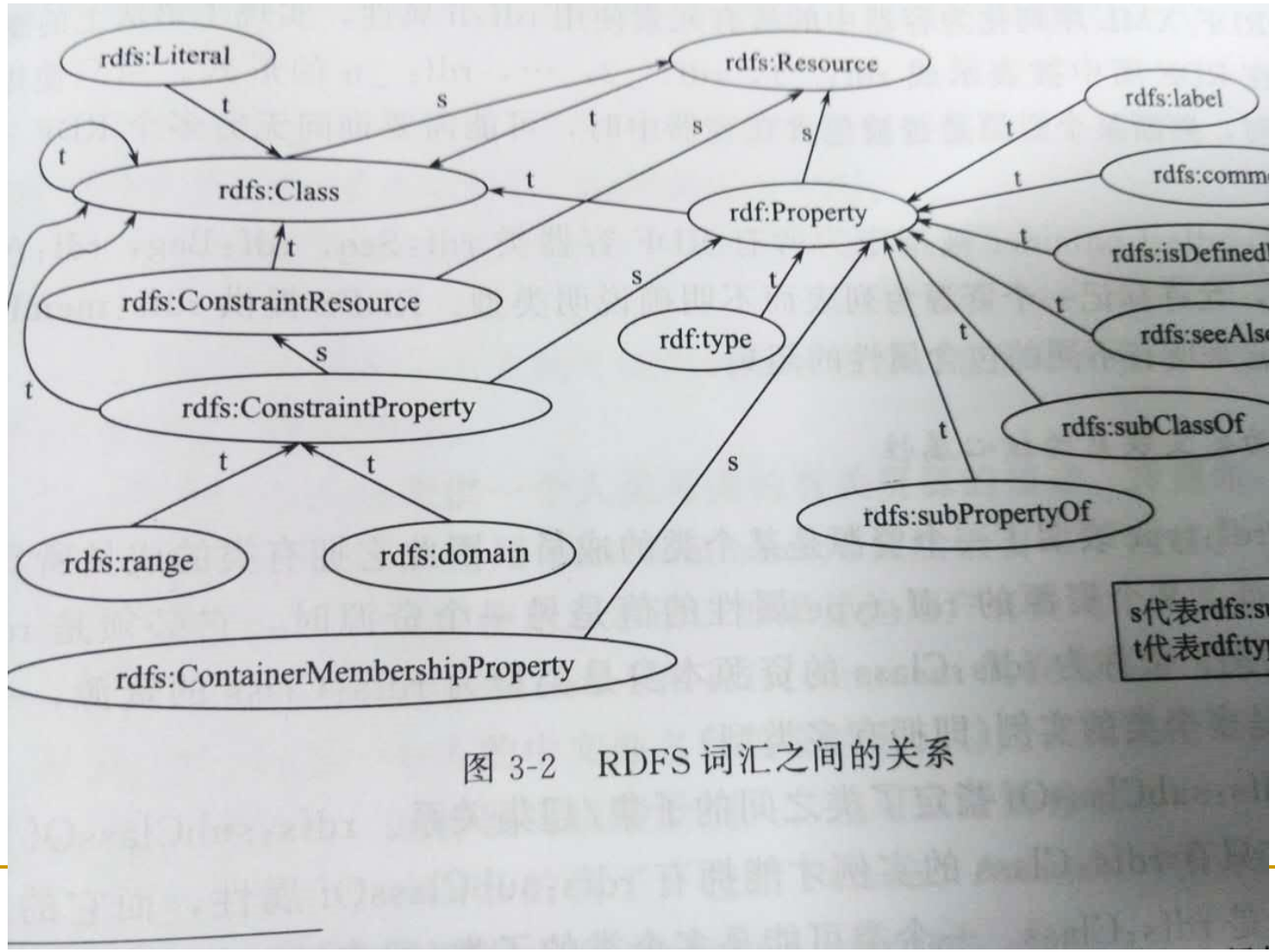


图 3-2　RDFS 词汇之间的关系

回顾-RDFS

RDFS中的描述属性：

< rdfs:label >：描述资源的名称，用以给人阅读

http://rdf.freebase.com/ns/m.07qr_hv < rdfs:label > "Bayes"@en.

< rdfs:comment >：描述资源的信息，用以给人阅读

< :某个资源 > < rdfs:comment > “这里可以随便写点描述什么的".

```
 :
 xmlns:wikipedia="http://en.wikipedia.org/wiki/"
 :
<rdfs:Class rdf:about="&ex;Primates">
  <rdfs:label xml:lang="en">primates</rdfs:label>
  <rdfs:comment>
    Order of mammals. Primates are characterized by an
    advanced brain. They mostly populate the tropical
    earth regions. The term 'Primates' was coined by
    Carl von Linné.
  </rdfs:comment>
  <rdfs:seeAlso rdf:resource="&wikipedia;Primates" />
  <rdfs:subClassOf rdfs:resource="&ex;Mammalia" />
</rdfs:Class>
```

回顾-RDFS

RDFS中的一些工具：

容器：
Class : < rdfs:Container >< rdf:Bag >< rdf:Seq >< rdf:Alt >
< rdfs:ContainerMembershipProperty >
Instance : < rdfs:member >是rdf:property的实例 (rdf:_1, rdf:_2, rdf:_3 ...)

集合：
< rdf:List >< rdf:first >< rdf:rest >< rdf:nil >

具体化词汇：
< rdf:Statement >< rdf:subject >< rdf:predicate >< rdf:object >

功能性词汇：
< rdfs:seeAlso >< rdfs:isDefinedBy >< rdf:value >

回顾-小结

- **RDF和RDFS使用灵活但是表达能力不足，因此，需要一种具有更强表达能力的语言来表示本体。OWL就是这样的一种语言。**

- **对于本体语言而言，主要需求有：**
  - 良定语法：清晰无二义
  - 形式化语义：语言含义能够被精确描述
  - 表达能力：提供精确的定义方式，并能够在其上进行推理

内容概览

一、本体论

二、OWL语言

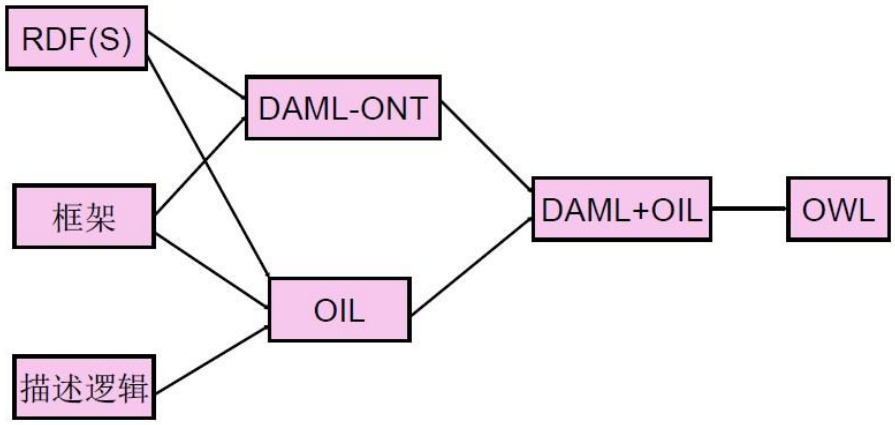
1. 回顾RDF
2. OWL概述
3. OWL语言大纲
4. OWL文档结构

OWL概述

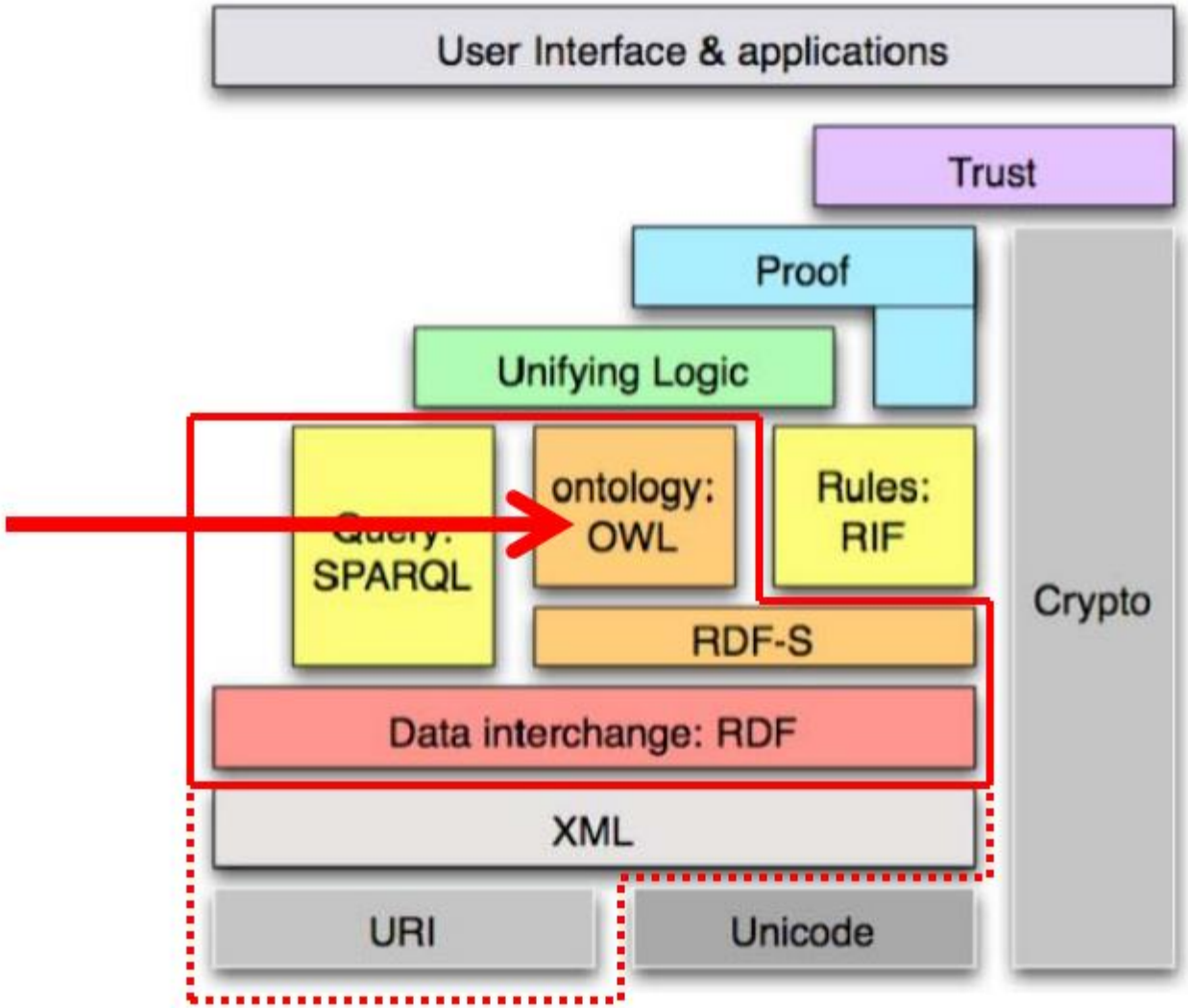
- **OWL**（the ***Web Ontology Language***，网络本体语言）是W3C在2002年7月29日公布的本体建模语言，并以此作为语义网（***Semantic Web***）的标准建模语言。
- 2003年2月21日，W3C公布了OWL1.0语言参考手册的最新草案。

OWL概述

- OWL是一种在互联网上公布和共享实体的语义语言（semantic markup language）。
- OWL起源于DAML+OIL网络实体语言，是作为RDF(the Resource Description Framework)的扩展词典进行开发。
- 其开发目的是提供一种“机器可读”的规范语言。
- OWL通过对XML，RDF以及RDF Schema 提供额外的词汇以及一套完整的语义而大大增加语言的“机器可读性”。

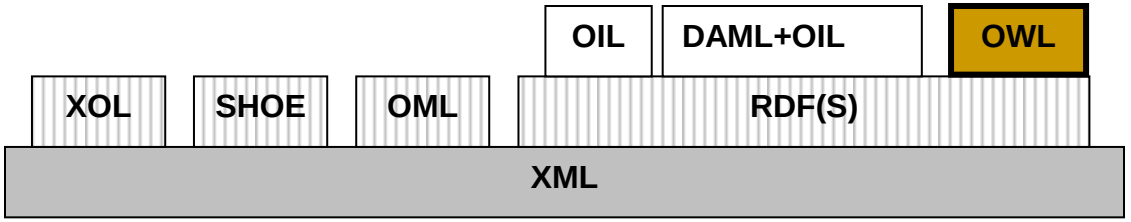


OWL概述


User Interface & applications
Trust
Proof
Unifying Logic
SPARQL
ontology: OWL
Rules: RIF
RDF-S
Data interchange: RDF
XML
URI
Unicode
Crypto

OWL概述

- XML、XML Schema 足以满足拥有一致定义（一个词典）的两者之间进行数据交换
- RDF、RDF Schema 简单的本体语言：概念、子概念、属性、子属性、继承、domain、range等等。但是不够强大，例如缺少cardinality、disjoint等等。



OWL概述

- **OWL语言的设计目标**
  - Ontology evolution 本体进化
  - Ontology interoperability 本体协作
  - Inconsistency detection 不一致检测
  - Balance of expressivity and scalability
    表达性和可伸缩性的权衡
  - Ease of use 易用性
  - Compatibility with other standards 兼容性
  - Internationalization 国际化

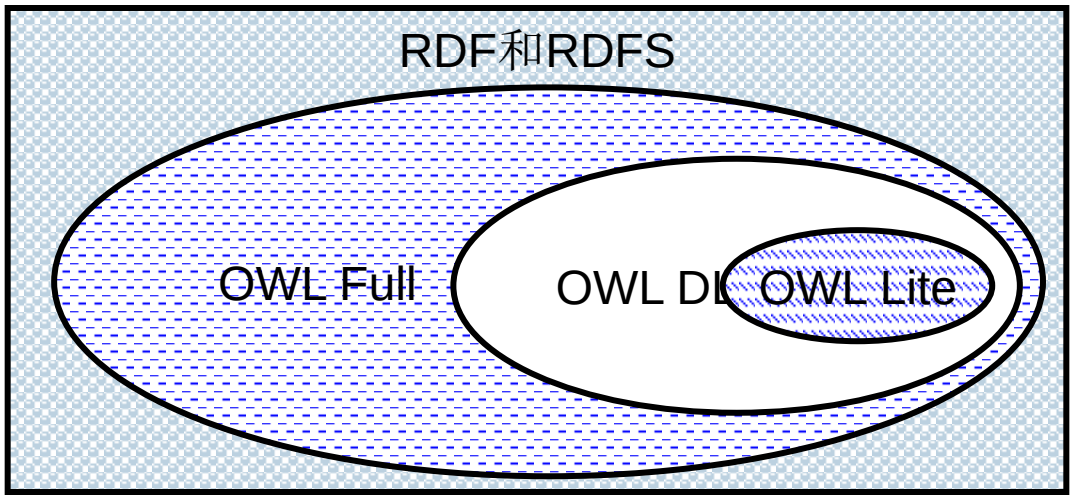
OWL概述

OWL的三个子语言

OWL Lite

OWL DL

OWL Full



从语法上来说，OWL-Lite是三个之中最简单的一个

子语言	描述	例子
OWL Lite	用于提供给那些只需要一个分类层次和简单的属性约束的用户。	支持基数（cardinality），只允许基数为0或1。
OWL DL	支持那些需要在推理系统上进行最大程度表达的用户，这里的推理系统能够保证计算完全性（computational completeness，即所有地结论都能够保证被计算出来）和可决定性（decidability，即所有的计算都在有限的时间内完成）。它包括了OWL语言的所有约束，但是可以被仅仅置于特定的约束下。	当一个类可以是多个类的一个子类时，它被约束不能是另外一个类的实例。
OWL Full	支持那些需要在没有计算保证的语法自由的RDF上进行最大程度表达的用户。它允许在一个Ontology在预定义的（RDF、OWL）词汇表上增加词汇，从而任何推理软件均不能支持OWL FULL的所有feature。	一个类可以被同时表达为许多个体的一个集合以及这个集合中的一个个体。

OWL-Full是OWL的三种子语言中表达能力最强的一个，但不太关心可判定性。

不过也正是由于表达能力太强这个原因，用OWL-Full表示的本体是不能进行自动推理的。

OWL概述

本体建模时选择子语言的考虑

只需要简单约束

- 选择OWL Lite还是OWL DL主要取决于用户需要整个语言在多大程度上给出约束的可表达性；
- 选择OWL DL还是OWL Full主要取决于用户在多大程度上需要RDF的元模型机制（如定义类型的类型以及为类型赋予属性）；

需要更多预定义词汇

- 在使用OWL Full而不是OWL DL时，推理的支持不可预测，因为目前还没有完全的OWL Full的实现。

推理不可预测

OWL概述

OWL与RDF(S)的关系

- OWL成为一种Web语言，不在于它引入了描述逻辑的表达和推理能力，而在于它用URI引用替代了名称，将XML Schema数据类型用于数据值，并且提供了连接到WWW文档上的能力，这些能力都是源于RDF(S)。
- 理想上，OWL是对RDF(S)的一个扩展，OWL可以使用RDF类和属性并增加支持更为丰富的表达元素。

OWL概述

- 但不幸的是，对RDF(S)的扩展和有效推理的需求相冲突
  - RDF(S)拥有一套相当强大和特殊的建模机制，如可以定义类的类、属性的类、属性的属性等。
  - 同时具有描述逻辑和RDF(S)表达能力的知识表示语言，其推理问题是不可判定的。
- OWL DL和OWL Lite都没有完全继承RDF(S)的表达能力，而是采用经典逻辑解释，对RDF(S)的使用做了很多限制：
  - 要求个体、类、特性是不相交的集合
  - 所有的个体都是资源，类是资源的集合，特性是（资源，资源）的集合，类和特性都不解释为资源
  - 禁止出现递归
- OWL Full则完全兼容RDF(S)，同时也包括了OWL DL的全部内容，但这也造成了OWL Full推理问题是不可判定的。

内容概览

一、本体论

二、OWL语言

1. 回顾RDF
2. OWL概述
3. OWL语言大纲
4. OWL文档结构

OWL Lite语言大纲

分类	词条（Terms）
RDF Schema Features	■ *Class* ■ *rdf:Property* ■ *rdfs:subClassOf* ■ *rdfs:subPropertyOf* ■ *rdfs:domain* ■ *rdfs:range* ■ *Individual*
(In)Equality	■ *equivalentClass* ■ *equivalentProperty* ■ *sameIndividualAs* ■ *differentFrom* ■ *allDifferent*

OWL Lite语言大纲

分类	词条（Terms）
Property Characteristics	■ *inverseOf* ■ *TransitiveProperty* ■ *SymmetricProperty* ■ *FunctionalProperty* ■ *InverseFunctionalProperty*
Property Type Restrictions	■ *allValuesFrom* ■ *someValuesFrom*
Class Intersection	■ *intersectionOf*

OWL Lite语言大纲

分类	词条（Terms）
Restricted Cardinality	■ *minCardinality (only 0 or 1)* ■ *maxCardinality (only 0 or 1)* ■ *cardinality (only 0 or 1)*
Header Information	■ *imports* ■ *versionInfo* ■ *priorVersion* ■ *backwardCompatibleWith* ■ *incompatibleWith*
Datatypes	

OWL DL和OWL Full语言大纲

下面给出了在OWL Lite基础上添加的OWL DL和OWL Full语言架构

分类	词条（Terms）
Class Axioms	■ *oneOf* ■ *disjointWith* ■ *equivalentClass* *(applied to class expressions)* ■ *rdfs:subClassOf* *(applied to class expressions)*
Boolean Combinations of Class Expressions	■ *unionOf* ■ *intersectionOf* ■ *complementOf*

OWL DL和OWL Full语言大纲

分类	词条（Terms）
Arbitrary Cardinality	■ *minCardinality* ■ *maxCardinality* ■ *cardinality*
Filler Information	■ *hasValue*

内容概览

一、本体论

二、OWL语言

1. 回顾RDF
2. OWL概述
3. OWL语言大纲
4. OWL文档结构

命名空间

在我们使用一组术语之前，我们需要一个精确地指出哪些具体的词汇表将被用到。

一个标准的本体开头部分里包括一组***XML命名空间（namespace）声明（被包含在rdf:RDF标签里）***。

这些声明用以准确解释文档中的标识符，从而使得Ontology的其他部分具有可读性。

命名空间

```
<rdf:RDF
   xmlns      ="http://www.example.org/wine#"
   xmlns:vin ="http://www.example.org/wine#"
   xmlns:food="http://www.example.org/food#"
   xmlns:owl ="http://www.w3.org/2002/07/owl#"
   xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
   xmlns:dte ="http://www.example.org/wine-dt#"    >
```

命名空间

为帮助书写冗长的URLs，**在本体的定义之前，在文档类型声明（DOCTYPE）中提供一些实体定义（entity definitions）常常是很有用的。**

这些被命名空间声明定义的名称仅当作为XML标签的一部分时才具有意义。

在OWL里，我们经常要用属性值来引用本体标识符。我们可以写出它们的完整URI形式，比如“http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#merlot”。或者，利用实体定义来简略URI的书写，例如

```
<!DOCTYPE owl [
   <!ENTITY vin  "http://www.example.org/wine#" >
   <!ENTITY food "http://www.example.org/food#" > ]>
```

在声明这些实体后，我们可以将“&vin;merlot”作为“http://www.example.org/owl/wine#merlot ”的简写

本体头部

建立了命名空间后，接下来我们通常要在owl:Ontology标签里给出一组关于本体的声明。这些标签支持一些重要的常务工作比如注释、版本控制以及其他本体的引入等。

```
<owl:Ontology rdf:about="http://www.example.org/wine">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:versionInfo>
      $Id: Overview.html,v 1.2 2002/11/08 16:42:25 connolly Exp $
  </owl:versionInfo>
 <owl:imports rdf:resource="http://www.w3.org/TR/2002/WD-owl-guide-20021104/food.owl"/>
</owl: Ontology>
```

本体头部

标签	说明
<rdfs:comment>	给出了本Ontology的主要功能
<owl:versionInfo>	标准标签，给出了供版本控制系统挂钩用的信息，OWL本身并没有什么结构上的约束。
<owl:imports>	提供了引入机制，只给出了一个参数rdf:resource。

其中引入另外一个Ontology将会将它的整个定义的集合加入到知识库中来。需要注意的是，这个标签只是说明了引入一个Ontology的意图，但不总是成功的。在语义网上对网上资源的访问不总是成功的，这需要依照工具实现的情况而定。

OWL基本元素

简单的类

用到的标签：Class, rdfs:subClassOf

一个领域中的最基本概念应分别对应于各个分类层次树的根。OWL中的所有个体都是类owl:Thing的成员。因此，各个用户自定义的类都隐含地是owl:Thing的一个子类。要定义特定领域的根类，只需将它们声明为一个具名类（named class）即可。

OWL也可以定义空类，owl:Nothing。

OWL基本元素

rdfs:subClassOf是用于类的基本分类构造符。它将一个较具体的类与一个较一般的类关联。如果X是Y的一个子类（subclass），那么X的每个实例同时也都是Y的实例。**rdfs:subClassOf关系是可传递的，**即如果X是Y的一个子类，而Y又是Z的一个子类，那么X就是Z的一个子类。

```
<owl:Class rdf:ID="Wine">
   <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
   <rdfs:label xml:lang="en">wine</rdfs:label>
   <rdfs:label xml:lang="fr">vin</rdfs:label>
   ...
 </owl:Class>

 <owl:Class rdf:ID="Pasta">
   <rdfs:subClassOf rdf:resource="#EdibleThing" />
   ...
 </owl:Class>
```

OWL基本元素

个体

一个Individual可以通过声明它是某个类的成员得以表达

```
<Region rdf:ID="CentralCoastRegion" />
```

注意：下面代码的含义与上面的例子相同。

```
<owl:Thing rdf:ID="CentralCoastRegion" />
<owl:Thing rdf:about="#CentralCoastRegion">
    <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

OWL基本元素

类和个体的实例

```
<owl:Class rdf:ID="Grape">

<owl:Class rdf:ID="WineGrape">
      <rdfs:subClassOf rdf:resource="#Grape"/>
</owl:Class>

<WineGrape rdf:ID="CabernetSauvignonGrape" />
```

OWL基本元素

简单属性（Property） --定义属性

所用术语：ObjectProperty, DatatypeProperty, rdfs:subPropertyOf, rdfs:domain, rdfs:range

一个属性是一个二元关系。有两种类型的属性：

数据类型属性（datatype properties）：类实例与RDF文字或XML Schema数据类型间的关系。

对象属性（object properties）：两个类的实例间的关系。

在我们定义一个属性的时候，有一些对该二元关系施加限定的方法。

可以指定定义域（domain）和值域（range）。

可以将一个属性定义为某个已有属性的特殊化（子属性）。

OWL基本元素

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
    <rdfs:domain rdf:resource="#Wine"/>
    <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
    <rdfs:domain rdf:resource="#Meal" />
    <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>
```

在OWL中，不含显式操作符的元素序列代表一个隐式的合取（conjunction）。属性madeFromGrape的定义域（domain）为Wine，*且*值域（range）为WineGrape。也就是说，它把Wine类的实例关联到WineGrape类的实例。为同一属性声明多个定义域表明该属性的定义域是所有这些类的交集（多个值域声明也类似这样）。

OWL基本元素

注意：OWL中值域和定义域信息的使用与程序设计语言中的类型信息有所不同。在程序设计中，类型被用来检查程序设计语言的一致性。**而在OWL中，一个定义域或值域可被用来推断一个类型。**比如，根据下面这段代码：

```
<owl:Thing rdf:ID="LindemansBin65Chardonnay">
    <madeFromGrape rdf:resource="#ChardonnayGrape" />
</owl:Thing>
```

我们可以推断出，LindemansBin65Chardonnay是一种葡萄酒，因为madeFromGrape的定义域为Wine。

OWL基本元素

```
<owl:Class rdf:ID="WineDescriptor" />
<owl:Class rdf:ID="WineColor">
    <rdfs:subClassOf rdf:resource="#WineDescriptor" />
     ...
</owl:Class>
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
    <rdfs:domain rdf:resource="#Wine" />
    <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasColor">
     <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
     <rdfs:range rdf:resource="#WineColor" />
     ...
</owl:ObjectProperty>
```

hasColor是hasWineDescriptor的子属性， **hasColor与hasWineDescriptor的不同在于它的值域被进一步限定为WineColor**。 rdfs:subPropertyOf关系表示：任何事物如果具有一个值为X的hasColor属性，那么它同时具有一个值为X的hasWineDescriptor属性。

OWL基本元素

```
<owl:ObjectProperty rdf:ID="locatedIn">
   <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
   <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#madeFromGrape"/>
            <owl:minCardinality      rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

■扩展Wine的定义来表达“一个葡萄酒是由至少一种WineGrape制成的”了。红色部分定义了一个无名类（unnamed class），该无名类代表至少具有一个madeFromGrape属性的事物集合。我们称这些类为*匿名类*。在Wine类的定义中包含该限定表明属于Wine类的事物，也是该匿名类的成员。也就是说，任何葡萄酒都必须参与至少一个madeFromGrape关系。

OWL基本元素

属性特性

传递属性(TransitiveProperty)：对于任意的x,y和z：P(x,y)与 P(y,z) 蕴含 P(x,z)

对称属性(SymmetricProperty)：对于任意的x和y：P(x,y)当且仅当P(y,x)

函数属性(FunctionalProperty)： 对于所有的x, y, 和z: P(x,y)与P(x,z) 蕴含 y = z

逆属性 （inverseOf）：如果一个属性P1被标记为属性P2的逆，对于所有的x 和 y: P1(x,y) 当且仅当P2(y,x)

反函数属性 （InverseFunctional）：对于所有的x, y 和 z: P(y,x) 与 P(z,x) 蕴含 y = z

InverseFunctional意味着属性的值域中的元素为定义域中的每个元素提供了一个唯一的标识。

OWL基本元素

TransitiveProperty

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

因为圣克鲁斯山地区（SantaCruzMountainsRegion）位于（locatedIn）加利福尼亚地区（CaliforniaRegion），那么它也应该位于（locatedIn）美国地区（USRegion），因为属性locatedIn是传递属性。

OWL基本元素

- **FunctionalProperty**

```
<owl:Class rdf:ID="VintageYear" />

<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range  rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

hasVintageYear属性是函数型属性。一种葡萄酒有着一个特定的制造年份。也即，一个给定的Vintage个体只能使用hasVintageYear属性与单独一个年份相关联。

OWL基本元素

inverseOf

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

各种葡萄酒都有制造商，这些制造商在Wine类的定义中被限制为酿酒厂（Winery）。而每个酿酒厂生产的酒均以该酿酒厂为制造商。

OWL基本元素

属性限制

除了能够指定属性特性，我们还能够使用多种方法进一步在一个明确的上下文中限制属性的值域。这是通过“属性限制”来完成的。

前面所讲述的机制都是***全局的（global）***，因为这些机制都会应用到属性的所有实例。而allValuesFrom与someValuesFrom是***局部的（local）***，**它们仅仅在包含它们的类的定义中起作用**。

owl:allValuesFrom属性限制要求：对于每一个有指定属性实例的类实例，该属性的值必须是由owl:allValuesFrom从句指定的类的成员。

OWL基本元素

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

Wine的制造商必须是Winery。allValuesFrom限制仅仅应用在Wine的hasMaker 属性上。Cheese的制造商并不受这一局部限制的约束。

OWL基本元素

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

allValuesFrom对于所有的葡萄酒，如果它们有制造商，那么所有的制造商都是酿酒厂。
someValuesFrom对于所有的葡萄酒，它们中至少有一个的制造商是酿酒厂。
这两种限制形式间的不同就是全称量词与存在量词间的不同。

OWL基本元素

基数限制：值域限制在0和1的基数表达式(Cardinality expressions)是OWL Lite的一部分。这使得用户能够表示“至少一个”，“不超过一个”，和“恰好一个”这几种意思。OWL DL中还允许使用除0与1以外的正整数值。owl:maxCardinality能够用来指定一个上界。owl:minCardinality能够用来指定一个下界。使用二者的组合就能够将一个属性的基数限制为一个数值区间。

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasVintageYear"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

OWL基本元素

hasValue 使得我们能够根据“特定的”属性值的存在来标识类。因此，一个个体只要至少有“一个”属性值等于hasValue的资源，这一个体就是该类的成员

```
<owl:Class rdf:ID="Burgundy">
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

这里我们声明了所有的Burgundy酒都是干(dry)的酒。也即，它们的hasSugar属性必须至少有一个是值等于Dry（干的）。**与allValuesFrom 和someValuesFrom类似，这是一个局部的限制。它仅仅对Burgundy类的hasSugar属性作出限制。**

本体映射

为了让本体发挥最大的作用，就需要让本体得到充分的共享。为了使得在开发本体时尽可能的节省人力，就需要使得开发出的本体能够被重用。更理想的情况是他们能够被组合使用。

例如，可能同时使用来自某一来源的日期本体(date ontology)和来自另一来源的物理位置本体(physical location ontology),并将位置(location)的概念加以扩展以包括这个位置所处在的时间段。

如果能够找到已经经过广泛使用和精炼的本体，那么采用它才有意义。多个本体的合并工作是非常具有挑战性的。为了维护其一致性,几乎必然需要工具的支持。

本体映射

类和属性之间的等价关系 -- equivalentClass,equivalentProperty

当我们要把一些本体组合在一起作为另一个新的本体的一部分时，能说明在一个本体中的某个类或者属性与另一个本体中的某个类或者属性是等价的

在食物本体中，我们现在想把在餐宴菜肴中对葡萄酒特点的描述与葡萄酒本体相联系起来。达到这一目的一种方法就是在食物本体中定义一个类(&food;Wine)，然后在葡萄酒本体中将一个已有的类声明为与这个类是等价的。

```
<owl:Class rdf:ID="Wine">
  <owl:equivalentClass rdf:resource="&vin;Wine"/>
</owl:Class>
```

类似的，我们可以通过使用owl:equivalentProperty属性声明表达属性的等同。

本体映射

个体间的同一性 --sameAs

描述个体之间相同的机制与描述类之间的相同的机制类似，仅仅只要将两个个体声明成一致的就可以了。

```
<Wine rdf:ID="MikesFavoriteWine">
    <owl:sameAs rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```

注意，在OWL DL中，类仅仅代表着个体的集合而不是个体本身。然而在OWL FULL中，我们能够使用owl:sameAs来表示两个类在各方面均完全一致。修饰（或引用）两个类用sameAs还是用equivalentClass效果是不同的。用sameAs的时候，把一个类解释为一个个体，就像在OWL Full中一样，这有利于对本体进行分类。在OWL Full中，sameAs可以用来引用两个东西，如一个类和一个个体、一个类和一个属性等等，无论什么情况，都将被解释为个体。

本体映射

不同的个体--AllDifferent

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <vin:WineColor rdf:about="#Red" />
    <vin:WineColor rdf:about="#White" />
    <vin:WineColor rdf:about="#Rose" />
  </owl:distinctMembers>
</owl:AllDifferent>
```

注意，owl:distinctMembers属性声明只能和owl:AllDifferent属性声明一起结合使用。

注意，如果我们想要在其他的某个本体中添加一个新的酒的颜色，并表明它是与其他已定义的任何酒的颜色都是不同的，我们可能需要拷贝原来的owl:AllDifferent属性声明，然后将新的颜色添加到列表中。在OWL DL中，没有更加简单的方法以扩展一个声明为owl:AllDifferent的集合。而在OWL Full中，通过使用RDF三元组和rdf:List构造，可以实现一些其他的方法来完成这一扩展。

复杂类

- OWL还提供了一些用于构建类的构造子。这些构造子被用于创建所谓的类表达式。
- OWL支持基本的集合操作，即并，交和补运算。它们分别被命名为owl:unionOf,owl:intersectionOf,和owl:complementOf.
- 类还可以是枚举的。类的外延可以使用oneOf构造子来显示的声明。同时，我们也可以声明类的外延必须是互不相交的。
- 注意类表达式是可以嵌套的，它并不要求要为每一个中间类都提供一个名字。这样就允许我们通过使用集合操作来从匿名类或具有值约束的类来创建复合类。

本体映射

交运算--intersectionOf

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

使用集合操作构造的类与我们目前所看到的所有语法中的定义类似。类的成员完全是通过集合操作来说明的。上面的语句说明WhiteWine *恰好*是类Wine与所有颜色是白色的事物的集合的交集。这就意味着如果某一事物是白色的并且是葡萄酒，那么它就是WhiteWine的实例。

本体映射

并运算--unionOf

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

Fruit类既包含了SweetFruit类的外延也包含了NonSweetFruit的外延。

本体映射

补运算--complementOf

complementOf结构从某个论域(domain of discourse)选出不属于某个类的所有个体。通常它将指向一个非常大的个体集合：

```
<owl:Class rdf:ID="ConsumableThing" />

<owl:Class rdf:ID="NonConsumableThing">
   <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

本体映射

complementOf典型的用法是与其它集合运算符联合使用：

```
<owl:Class rdf:ID="NonFrenchWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine"/>
    <owl:Class>
      <owl:complementOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn" />
          <owl:hasValue rdf:resource="#FrenchRegion" />
        </owl:Restriction>
      </owl:complementOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

该例子定义了一个NonFrenchWine类，它是Wine类与所有不位于法国的事物的集合的交集。

本体映射

枚举类 --oneOf

OWL提供了一种通过直接枚举类的成员的方法来描述类。这是通过使用one of 结构来完成。特别地，这个定义完整地描述了类的外延，因此任何其他个体都不能被声明为属于这个类。

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>

  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White"/>
    <owl:Thing rdf:about="#Rose"/>
    <owl:Thing rdf:about="#Red"/>
  </owl:oneOf>

</owl:Class>
```

本体映射

不相交类 --disjointWith

使用owl:disjointWith构造子可以表达一组类是不相交的。它保证了属于某一个类的个体不能同时又是另一个指定类的实例。

```
<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>

  <owl:disjointWith rdf:resource="#Meat"/>
  <owl:disjointWith rdf:resource="#Fowl"/>
  <owl:disjointWith rdf:resource="#Seafood"/>
  <owl:disjointWith rdf:resource="#Dessert"/>
  <owl:disjointWith rdf:resource="#Fruit"/>

 </owl:Class>
```

Pasta例子声明了多个不相交类。注意它只声明了Pasta与其它所有类是不相交的。例如，它并没有保证Meat和Fruit是不相交的。为了声明一组类是互不相交的，我们必须对每两个类都使用owl:disjointWith来声明。

本体映射

一个常见的需求是定义一个类为一组互不相交的子类的联合（union）。

```
<owl:Class rdf:ID="SweetFruit">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
</owl:Class>

<owl:Class rdf:ID="NonSweetFruit">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
  <owl:disjointWith rdf:resource="#SweetFruit" />
</owl:Class>

<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

总结

- 了解了本体论基本概念。
- 学习了OWL本体语言，OWL是W3C推荐的标准本体语言。
- OWL是对RDF(S)的一个扩展，OWL可以使用RDF类和属性并增加支持更为丰富的表达元素。
- OWL有三种子语言，表达能力和推理能不同。
- 学习了OWL的基本语法。