# Парсер

При построении VRML-парсера используется подход, известный как «метод рекурсивного спуска» (recursive-descent parser). Пожалуй, самая привлекательная черта этого метода – строгость реализации парсера, которая одновременно сочетается с определенной его простотой.

Основой конструирования парсера является так называемая «грамматика» языка – набор выражений, определяющих более выражения высокого уровня через более простые. Так, например, начинается грамматика VRML (см. [стандарт](http://accad.osu.edu/~pgerstma/class/vnv/resources/info/AnnotatedVrmlRef/appa.htm)):

|  |
| --- |
| *vrmlScene*::=  *statements* ;  *statements*::=  *statement*|  *statement statements*|  *empty*;  *statement*::=  *nodeStatement* |  *protoStatement* |  *routeStatement* ;  *nodeStatement*::=  *node* |  **DEF***nodeNameId node* |  **USE***nodeNameId* ;  …  *node*::=  *nodeTypeId***{** *nodeBody***}** |  **Script {***scriptBody***}** ;  … |

«Рекурсивный спуск» как раз и состоит в разборе исходного кода согласно этой грамматике; мы как бы спускаемся по ней сверху вниз – от общего к частному (таким образом, это *top-down* парсер). При этом каждому выражению в грамматике соответствует специальный метод в парсере. Если открыть исходный код, то можно увидеть там, например, метод *parseStatement*, который, в свою очередь, вызывает *parseNodeStatement*, *parseProtoStatement* и *parseRouteStatement*. Такой подход позволяет создавать хорошо читаемый и строго организованный код (чего не хватает гринкруговской реализации). Пока что в парсере реализован разбор лишь небольшой части грамматики VRML – но, если продолжать в том же духе, код не должен сильно усложниться. Однако к недостаткам подхода следует отнести большое количество методов в реализации (по одному на каждое выражение в грамматике) и, соответственно, довольно «глубокую» рекурсию при разборе VRML-сцены большого объема. Скорее всего, это не вызовет больших проблем.

# Система типов

Каждый VRML-узел представляет собой набор пар *field – value*, и это также строго описывается в грамматике VRML. Там же указывается перечень типов в языке с их определением:

|  |  |
| --- | --- |
| *fieldValue*::=  *sfboolValue* |  *sfcolorValue* |  *sffloatValue* |  *sfimageValue* |  *sfint32Value* |  *sfnodeValue* |  *sfrotationValue* |  *sfstringValue* |  *sftimeValue* |  *sfvec2fValue* |  *sfvec3fValue* |  *mfcolorValue* |  *mffloatValue* |  *mfint32Value* |  *mfnodeValue* |  *mfrotationValue* |  *mfstringValue* |  *mftimeValue* |  *mfvec2fValue* |  *mfvec3fValue* ; | *sfboolValue*::=  **TRUE** |  **FALSE** ;  *sfcolorValue*::=  *float float float* ;  *sffloatValue*::=  ... floating point number in ANSI C floating point format...  *sfimageValue*::=  *int32 int32 int32 int32 ...*  *sfint32Value*::=  [[**+**]|**-**]{[**0-9**]+|**0x**[**0-9a-fA-F**]**+**};  *sfnodeValue*::=  *nodeStatement* |  **NULL** ;  *sfrotationValue*::=  *float float float float* ;  … |

Тут все более-менее просто; единственную сложность представляет тип *sfnodeValue*, который сам по себе является узлом – это означает, что в VRML мы можем иметь несколько уровней вложенности узлов друг в друга (обрабатывать их парсер уже научился).

***Первая задача – реализовать весь набор типов VRML в виде Java-классов (по одному на тип).*** Парочка примеров есть в пакете *ru.hse.se.types* – это *SFBool* и *SFColor* (а также их базовый класс *VRMLType*). Вообще, стоит называть Java-классы так же, как соответствующие типы зовутся в стандарте VRML – так будет проще. Единственное исключение тут – тип *Node*. Он также является потомком *VRMLType*, но сам, в свою очередь, выступает в качестве родителя для всех узлов, которые, собственно, и будут бинами, из которых строится граф сцены. По факту этот *Node* – тип *SFNode*, но, так как он вынесен в пакет *ru.hse.se.nodes*, то имеет смысл назвать его попроще.

Типы из пакета *ru.hse.se.types* используются в пакете *ru.hse.se.nodes* для объявления свойств узлов-бинов (и если у какого-то узла будет свойство булева типа, то надо будет использовать именно *SFBool*, а не стандартный *boolean).* Соответственно, активно используются они и в парсере (опять же, по одному методу *parseXxx* для каждого типа).

# Набор узлов

Собственно узлы, из которых будет строиться наша компонентная JavaBeans-модель, описываются в пакете *ru.hse.se.nodes.* ***Вторая задача – реализовать типы узлов, которые есть в VRML.*** Некоторые зачатки, опять же, уже есть: например, узел *Geometry*, который наследует *Node* и является родителем для узлов *Box* и *Shape*. На самом деле, в VRML несколько больше узлов, относящихся к типу *Geometry* (около 10). Описание их можно найти в стандарте.

Ниже приведен набросок диаграммы классов.

