Schema class

new Schema(spec : SchemaSpec) 用一个 SchemaSpec（配置对象）构造一个文档的规则对象

spec : SchemaSpec

当前规则的配置对象，其中的 nodes 和 marks 属性可以保证是 OrderedMap 的实例（不是原始对象）

nodes : Object<NodeType>

一个 schema 中节点名和节点类型对象的键值对映射。

marks : Object<MarkType>

一个 mark 名和 mark 类型对象的键值对映射。

topNodeType : NodeType

当前 schema 的 默认顶级节点 类型。

cached : Object

一个用于计算和缓存每个 schema 中的任何类型值的对象。（想要在其上储存一些东西,考虑属性名重复的问题

node( 在 schema 中新建一个节点

type: string | NodeType,

attrs: ?⁠Object, Attributes 会被以默认值扩展

content: ?⁠Fragment | Node | [Node], content 可能是一个 Fragment、 null、Node 或者一个节点数组

marks: ?⁠[Mark]

)

返回： Node 对象

text( 在 schema 中新建一个文本节点。不允许创建空的文本节点

text: string,

marks: ?⁠[Mark]

)

返回： Node 对象

注: 文本节点和文本块节点不同，注意区分。

mark( 用指定的类型和 attributes 创建一个 mark

type: string | MarkType,

attrs: ?⁠Object

)

返回： Mark 对象

nodeFromJSON( 从一个 JSON 表达式中反序列化出一个节点。该方法 this 已经绑定当前对象

json: Object

) 返回： Node 对象

注: JSON 表达式其实并不是 JavaScript 中通常意义上的 JSON 字符串，而是一个普通对象，它及它的键值都是 plain object。

该对象由相应的 Node.toJSON 生成。

markFromJSON( 从一个 JSON 表达式中反序列化出一个 mark。该方法 this 已经绑定当前对象

json: Object

) 返回： Mark 对象

注: 该对象由相应的 Mark.toJSON 生成。

SchemaSpec 规则配置对象 一个描述 schema 的对象，用来传递给 Schema 构造函数

nodes : Object<NodeSpec> | OrderedMap<NodeSpec>

当前规则中所有节点类型的对象。

对象中，键是节点名，对象的键是对应的 NodeSpec。

节点们在该对象中出现的先后顺序是非常重要的，它决定了默认情况下哪个节点的 parse rules 优先进行， 以及哪个节点是一个 group 优先考虑的节点。

marks: ?⁠Object<MarkSpec> | OrderedMap<MarkSpec>

当前 schema 中的所有 mark 类型的对象。它们出现的顺序决定了在 mark sets 中的存储顺序，以及 解析规则的处理顺序

topNode: ?⁠string

当前 schema 顶级节点的名字，默认是 "doc"

NodeSpec 节点配置对象

参数 :值类型 作用 备注

content :string 当前节点的内容表达式,指定子节点可以使用哪些类型的节点 为 null则该节点不允许任何内容

marks :string 行内节点的标记，值可以是标记名或组名； "\_"：含所有标记， "" 不含任何标记

字段为空，有内容就可以设置所有标记，无内容就不可以设置任何标记

group :string 节点所属的组，可以多个，用空格分隔，

inline :bool 内联节点， 应设置: true 文本节点: 被隐式设置: true

atom :bool true，表示当前节点不是叶节点，但也没直接编辑内容，在 view 中被视为独立单位对待。

注: 「独立单位」是，如在计数上，是 1； 在事件上，内部元素触发的事件被视作是该节点触发的

attrs : AttributeSpec 当前节点的属性设置 AttributeSpec 是属性配置对象, 用来定义节点或者标记属性

selectable : ⁠bool 控制当前类型的节点能否被作为节点选区 所选中。 非文本节点默认： true

draggable : bool 决定在未选中的情况下，当前类型的节点能否被拖拽。 默认是 false

code : bool 指示当前节点包含 code，其会引起一些命令有特别行为。

注: 「特别行为」如：在code节点中。内容是li做code 块处理； 文档中的 li针对 li 进行处理

defining : ⁠bool 当前节点在替换操作中是否被视为重要的父级节点（如粘贴操作）。 默认是 false

false节点内容被全部替换时节点会被移除，为 true 的节点会保留，然后包裹住替换进来的内容，插入内容时，为 true 的父级节点会被尽可能的保留。

建议：自定义paragraph类型及 li 元素，设置 true。因为粘贴到此位置一般会希望为这个样式的延伸

isolating : ⁠bool 设置： true 时（默认false），当前类型节点的两侧会计算作为边界，对于正常的编辑操作：删除、提升，将不会被跨越过去。

举个例子，对于 table 的 cell 节点，该属性应该被设置为 true。

注: 「提升」操作指的是，如在一个二级 li 中，一般用户习惯下，按 shift + tab 会将该二级 li 提升到一级 li。

注: 「跨越」指的是，操作会跨过当前节点到达下一个（或者上一个）节点。

如删除操作，在段落起始位置继续按删除键，光标会跑到上一个节点的尾部；

在 li 起始位置按删除键，光标会跑到上一个 li 结尾处或者直接删除整个 ul/ol；

在 table 的 td 中，在 td 起始位置按删除键跑到上一个 td 结尾， 显然不是预期。

toDOM : ⁠fn(node: Node) 返回: DOMOutputSpec 说明节点序列化成 DOM/HTML 的方式（被DOMSerializer.fromSchema使用）

返回DOM节点或描述DOM节点的结构数组，它有可选的数字0「洞」，表示节点内容出入内容。

对于文本节点，默认创建一个文本 DOM 节点

parseDOM : ⁠ParseRule DOM 解析信息，会被 DOMParser.fromSchema 使用以自动的衍生出一个解析规则。

规则中的节点字段是隐式的（节点的名字会自动填充），这里提供了解析说明，则不需要配置规则中的解析

注: 配置 Editor view 的时候可以配置一个 Parser 和 Serializer，如果提供，则此处就不用写 parseDOM 了。

Parser： 解析器 Serializer : 序列化程序

toDebugString: ⁠fn(node: Node) → string

定义一个该类型节点被序列化成一个字符串形式的默认方法，以做 debugging 用途 debugging: 调试

AttributeSpe 属性配置对象

用来 define node 或者 marks 的 attributes。

default: ?⁠any

该 attribute 的默认值，当没有显式提供值的时候使用。

如果 attributes 没有默认值，则必须在新建一个 node 或者 mark 的时候提供值。

MarkSpec 标记配置对象

attrs : ?⁠Object<AttributeSpec>

当前 mark 类型拿到的 attributes。

inclusive : ?⁠bool

该 marks 是否应该被激活。默认是 true

当光标放到该 mark 的结尾处（或者如果该 mark 开始处同样是父级节点的开始处时，放到 mark 的开始处）时，

注: 「被激活」的意思是，可以通过 API 获取光标所在的 resolvedPos 信息以查到相关的 marks，

对用户来说被激活意味着在该地方输入内容会带上相应的 marks。

excludes : ?⁠string

决定当前 mark 是否能和其他 marks 共存。

0. 应该是由其他 marks 名或者 marks group 组成的以空格分隔的字符串。 当一个 marks 被 added 到一个集合中时，所有的与此 marks 排斥（excludes）的 marks 将会被在添加过程中移除。 如果当前集合包含任何排斥当前的新 mark 的 mark，但是该新 mark 却不排斥它，则当前新的 mark 不会被添加到集合中。你可以使用 "\_" 来表明当前 marks 排斥所有的 schema 中的其他 marks。

1.注: 该段的主要意思是，第一：假设 A 、B 互斥，则 无论 A 添加到包含 B 的集合，还是 B 添加到 包含 A 的集合，已经在集合中的一方会被移除以添加新的 mark； 第二：若假设 A 排斥 B，B 却不排斥 A，则将 B 添加到包含 A 的集合中去的时候，将不会被添加进去。

2. 默认是相同类型的 marks 会互斥。你可以将其设置为一个空的字符串（或者任何不包含 mark 自身名字的字符串） 以允许指定相同类型的多个 marks 共存（之哟啊他们有不同的 attributes）。

group : ?⁠string

当前 mark 所属的 一个 group 或者空格分隔的多个 groups。

spanning : ?⁠bool 默认是 true

决定当序列化为 DOM/HTML 的时候，当前类型的 marks 能否应用到相邻的多个节点上去。

toDOM : ⁠fn( mark: Mark, inline: bool) 返回 DOMOutputSpec

定义当前类型的 marks 序列化为 DOM/HTML 的默认方式。如果结果配置对象包含一个「洞」，则洞的位置就是 mark 内容所在的位置。否则，它会被附加到顶级节点之后。

注: 「否则，它会被附加到顶级节点之后」字面意思吗？有待实验，本人貌似没有印象了。

parseDOM : ?⁠[ParseRule]

当前 mark 的相关的 DOM parser 信息（具体请查看相应的 node spec field）。 在 Rules 中的 mark 字段是隐式的。

MarkType class

name : string mark 类型的名称

schema : Schema 当前 mark 类型所属于的 schema

spec : MarkSpec 当前 mark 类型的配置对象

create( attrs: ?⁠Object) → Mark 创建一个当前类型的 mark

attrs 是 null 或 仅包含部分 marks attributes 的对象。 其他未包含的 attributes，会使用它们的默认值添加上去

removeFromSet(set: [Mark]) → [Mark]

如果当前 mark 类型存在与指定的 mark 集合，则将会返回不含有当前 mark 类型的 marks 集合。

否则，直接返回指定的 marks 集合

注: 看函数名，顾名思义就是在指定 marks 集合中移除当前 mark 类型的 marks。

isInSet(set: [Mark]) → ?⁠Mark 检查当前类型的 marks 是否存在于指定 marks 集合。

excludes(other: MarkType) → bool 查询指定的 mark 类型是否与当前 mark 类型 excluded（互斥）

NodeType class 这种对象包含了节点的类型信息，比如名称以及它表示那种节点

*name : string 该节点类型在 schema 中的名称。*

*schema : Schema 一个指向节点类型所属 Schema 的指针。*

*spec : NodeSpec 当前类型的配置对象。*

*contentMatch : ContentMatch 节点类型内容表达式的起始匹配*。

注: sorry，这个 contentMatch 我用的比较少，所以也不知道是什么意思，貌似源码内部使用的比较多。

inlineContent : bool 如果当前节点类型有内联内容的话，即为 true。

isBlock : bool 当前节点是块级类型的话，即为 true。

注: 判断是否是块级类型是用排除法

如果不是内联类型（即 spec.inline 是 false）且节点类型的名称不是「text」，则该类型是块级类型。

isText : bool 如果是文本类型的节点，即为 true。

注: 也即节点名字是「text」。

isInline : bool 如果是一个内联类型，则为 true。

注: 同样使用排除法，即与 spec.isBlock 互斥。

isTextblock : bool 节点是文本块类型节点则为 true，即一个包含内联内容的块级类型节点

注: 一个块级类型可能包含另一个块级类型，一个文本块类型则只会包含内联内容，哪些节点是内联元素由 schema 决定

注: 文本块类型的判断需要同时满足 spec.isBlock 和 spec.inlineContent 同时为 true。

isLeaf : bool 如果节点不允许内容，则为 true。

注: 是否是叶节点使用的是 spec.contentMatch 是否为空判断的。

isAtom : bool 节点是一个原子节点则为 true，例如，一个没有直接可编辑的内容的节点

hasRequiredAttrs( ) 返回 bool 告诉你该节点类型是否有任何必须的 attributes。

create( attrs: ?⁠Object, content: ? ⁠Fragment | Node | [Node] | null, marks: ?⁠[Mark] ) 返回： Node

创建一个指定类型的节点。 属性可以为 null 则为默认值 marks 默认是 null，表示空的 marks 集合。

createChecked( attrs: ?⁠Object, content: ? ⁠Fragment | Node | [Node] | null, marks: ?⁠[Mark] ) 返回： Node

与 create 类似，但是会检查指定的 content 是否符合节点类型的内容限制，如果不符的话会抛出一个错误

注: 该自定义错误类型为 RangeError。

createAndFill attrs: ?⁠Object, content: ? ⁠Fragment | Node | [Node] | null, marks: ?⁠[Mark] ) 返回： Node

额外判断fragment 开始和结尾的地方是否有必要添加一些节点，让fragment 适应当前 node。

如果没有找到合适的包裹节点，则返回 null。

如果你传递 null 或者 Fragment.empty 作为内容会导致其一定会适合当前 node，因此该方法一定会成功。

注: 因为 null 和 Fragment.empty 不用寻找任何「合适的包裹节点」就能适应当前节点。

validContent( content : Fragment ) → bool

如果指定的 fragment 对当前带有 attributes 的节点是可用的，则返回 true。

allowsMarkType( markType : MarkType) → bool

检查当前节点类型是否允许指定的 mark 类型。

allowsMarks( marks : [Mark]) → bool

检查当前节点类型是否允许指定的 marks 集合。

allowedMarks( marks : [Mark]) → [Mark]

从指定的 marks 集合中移除不允许出现在当前 node 中的 marks

ContentMatch class

该类的实例表示一个节点类型的 content expression（内容表达式） 的匹配状态， 其可以用来寻找是否此处是否有更进一步的内容能够匹配到，以及判断一个位置是否是该节点的可用的结尾。

注: 本小节的方法和类我用的不多，可能在处理一些边缘 case 的情况才会用到，因此很多直译了。

validEnd: bool 当匹配状态表示该节点有一个可用的结尾时为 true。

matchType(type: NodeType) → ?⁠ContentMatch 匹配一个节点类型，如果成功则返回该 ContentMatch 匹配结果。

matchFragment(frag: Fragment, start: ?⁠number = 0, end: ?⁠number = frag.childCount) → ?⁠ContentMatch

尝试去匹配一个 fragment。如果成功则返回 ContentMatch 匹配结果。

defaultType: ?⁠NodeType 获取相应匹配位置的第一个可以被生成的匹配节点类型。

注: 「可以被生成的」指的是该位置不能是文本节点或者不能有必须存在的 attribute 才能被生成。

fillBefore(after: Fragment, toEnd: ?⁠bool = false, startIndex: ?⁠number = 0) → ?⁠Fragment

尝试匹配指定的 fragment，如果失败，则会查看是否可以通过在该 fragment 前面插入一些节点来使之匹配。 如果插入节点后匹配成功，则会返回一个插入的节点组成的 fragment（如果没有需要插入的节点，则可能是空的）。 当 toEnd 为 true 时，只有结果匹配到达了内容表达式的结尾之时，才会返回一个 fragment。

注: 否则返回 undefined。

findWrapping(target: NodeType) → ?⁠[NodeType]

寻找一个包裹指定节点的节点集合，该集合中的节点在包裹住指定类型的节点后才能出现在当前位置。 集合的内容可能是空（如果指定类型节点直接就适合当前位置而无需包裹任何节点时），若不存在相应的包裹节点，则集合也可能是 null

edgeCount: number

在描述内容表达式的有限自动机中该节点拥有的外部边界的数量。

注: 没理解什么意思，需要看源码，我直译的，鼠标悬浮查看原始文档

edge(n: number) → {type: NodeType, next: ContentMatch}

在描述内容表达式的有限自动机中获取该节点第 n 个外部的边界。