Especificación Versión 1.1

BSON es un formato binario en el que cero o más pares ordenados de clave/valor se almacenan como una sola entidad. Llamamos a esta entidad un *documento* .

La siguiente gramática especifica la versión 1.1 del estándar BSON. Hemos escrito la gramática usando una sintaxis pseudo- BNF . Los datos BSON válidos están representados por el `document`no terminal.

Tipos basicos

Los siguientes tipos básicos se utilizan como terminales en el resto de la gramática. Cada tipo debe ser serializado en formato little-endian.

`byte`	1 byte (8 bits)
`int32`	4 bytes (entero con signo de 32 bits, complemento a dos)
`int64`	8 bytes (entero con signo de 64 bits, complemento a dos)
`uint64`	8 bytes (entero sin signo de 64 bits)
`doble`	8 bytes (coma flotante binaria IEEE 754-2008 de 64 bits)
`decimal128`	16 bytes (coma flotante decimal IEEE 754-2008 de 128 bits)

no terminales

Lo siguiente especifica el resto de la gramática BSON. Tenga en cuenta que las cadenas entrecomilladas representan terminales y deben interpretarse con semántica C (por ejemplo, `"\x01"`representa el byte `0000 0001`). También tenga en cuenta que usamos el `*` operador como abreviatura de repetición (por ejemplo, `("\x01"*2)` es `"\x01\x01"`). Cuando se usa como operador unario, `*`significa que la repetición puede ocurrir 0 o más veces.

`documento`	`::= int32 e_list "\x00"`	Documento BSON. int32 es el número total de bytes que comprende el documento.
`lista_e`	`::= elemento e_list`	
	`| ""`	
`elemento`	`::= "\x01" nombre_e doble`	punto flotante binario de 64 bits
	`| Cadena de nombre electrónico "\x02"`	Cadena UTF-8
	`| Documento de nombre electrónico "\x03"`	Documento incrustado
	`| Documento de nombre electrónico "\x04"`	Formación
	`| "\x05" e_name binario`	Datos binarios
	`| "\x06" e_nombre`	Indefinido (valor) — *Obsoleto*
	`| "\x07" nombre_e (byte*12)`	ID de objeto
	`| "\x08" e_nombre "\x00"`	Booleano "falso"
	`| "\x08" e_nombre "\x01"`	Booleano "verdadero"
	`| "\x09" nombre_e int64`	fecha y hora UTC
	`| "\x0A" nombre_e`	Valor nulo
	`| "\x0B" e_name cstring cstring`	Expresión regular: la primera cstring es el patrón de expresión regular, la segunda es la cadena de opciones de expresión regular. Las opciones se identifican mediante caracteres, que deben

		almacenarse en orden alfabético. Las opciones válidas son 'i' para coincidencias que no distinguen entre mayúsculas y minúsculas, 'm' para coincidencias multilínea, 'x' para modo detallado, 'l' para hacer que \w, \W, etc. dependan de la configuración regional, 's' para modo dotall ('. ' coincide con todo) y 'u' para hacer que \w, \W, etc. coincidan con Unicode.
	`| Cadena de nombre electrónico "\x0C" (byte*12)`	DBPointer: en *desuso*
	`| Cadena de nombre electrónico "\x0D"`	código JavaScript
	`| Cadena de nombre electrónico "\x0E"`	Símbolo. — *Obsoleto*
	`| "\x0F" e_name código_w_s`	Código JavaScript con ámbito: en *desuso*
	`| "\x10" nombre_e int32`	entero de 32 bits
	`| "\x11" nombre_e uint64`	marca de tiempo
	`| "\x12" nombre_e int64`	entero de 64 bits
	`| "\x13" e_name decimal128`	Coma flotante decimal de 128 bits
	`| "\xFF" e_nombre`	Clave mínima
	`| "\x7F" nombre_e`	tecla máx.
`e_name`	`::= cuerda C`	Nombre clave
`cuerda`	`::= int32 (byte*) "\x00"`	Cadena: el int32 es el número de bytes en (byte*) + 1 (para el final '\x00'). El (byte*) es cero o más caracteres codificados en UTF-8.
`cuerda C`	`::= (byte*) "\x00"`	Cero o más caracteres codificados en UTF-8 modificados seguidos de '\x00'. El (byte*) NO DEBE contener '\x00', por lo que no es UTF-8 completo.
`binario`	`::= subtipo int32 (byte*)`	Binario: el int32 es el número de bytes en el (byte*).
`subtipo`	`::= "\x00"`	Subtipo binario genérico
	`| "\x01"`	Función
	`| "\x02"`	Binario (Antiguo)
	`| "\x03"`	UUID (antiguo)
	`| "\x04"`	UUID
	`| "\x05"`	MD5
	`| "\x06"`	Valor BSON cifrado
	`| "\x07"`	Columna BSON comprimida
	`| "\x80"`	Usuario definido
`código_w_s`	`::= documento de cadena int32`	Código con ámbito: *en desuso*

notas

- Matriz: el documento de una matriz es un documento BSON normal con valores enteros para las claves, comenzando con 0 y continuando secuencialmente. Por ejemplo, la matriz ['red', 'blue'] se codificaría como el documento {'0': 'red', '1': 'blue'}. Las claves deben estar en orden numérico ascendente.
- Fecha y hora UTC: el int64 es milisegundos UTC desde la era Unix.

- Marca de tiempo: tipo interno especial utilizado por la replicación y fragmentación de MongoDB. Los primeros 4 bytes son un incremento, los segundos 4 son una marca de tiempo.
- Clave mínima: tipo especial que se compara por debajo de todos los demás valores posibles del elemento BSON.
- Tecla Max: tipo especial que se compara más alto que todos los demás valores posibles del elemento BSON.
- Subtipo binario genérico: este es el subtipo binario más utilizado y debería ser el "predeterminado" para controladores y herramientas.
- El tipo de datos "binario" o "BinData" de BSON se utiliza para representar matrices de bytes. Es algo análogo a la noción de Java de un ByteArray. Los valores binarios de BSON tienen un *subtipo* . Esto se usa para indicar qué tipo de datos hay en la matriz de bytes. Los subtipos de cero a 127 están predefinidos o reservados. Los subtipos del 128 al 255 son definidos por el usuario.
    - \x02 Binario (antiguo): este solía ser el subtipo predeterminado, pero quedó obsoleto en favor de \x00. Los controladores y las herramientas deben asegurarse de manejar \x02 de manera adecuada. La estructura de los datos binarios (la matriz de bytes* en el no terminal binario) debe ser un int32 seguido de un (byte*). El int32 es el número de bytes en la repetición.
    - \x03 UUID (antiguo): este solía ser el subtipo UUID, pero quedó obsoleto en favor de \x04. Los controladores y las herramientas para idiomas con un tipo de UUID nativo deben manejar \x03 de forma adecuada.
    - \x80-\xFF Subtipos "definidos por el usuario". Los datos binarios pueden ser cualquier cosa.
- Código con ámbito: *en desuso.* El int32 es la longitud en bytes del valor completo de code_w_s. La cadena es código JavaScript. El documento es una asignación de identificadores a valores, que representa el ámbito en el que se debe evaluar la cadena.