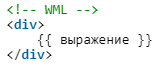
# Wasaby Templates

## Синтаксис шаблонов

### Mustache

Шаблонизатор Wasaby распознаёт конструкции, созданные с помощью {{}}.

Выражение этих конструкций можно составлять из переменных, находящихся в области видимости шаблона, а также из синтаксических конструкций языка JavaScript.

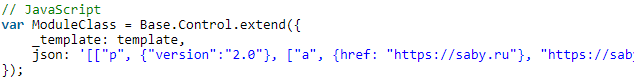


При обработке шаблона вычисляется выражение конструкции, его значение экранируется и, как следствие, выводится на странице как простой текст (не HTML)

### Вывод JsonML

Значение конструкции можно вывести как HTML. Для этого необходимо отрендерить Controls/decorator:Markup с опцией value, которая принимает данные формата JsonML (перед вставкой выполняет санитизацию небезопасных тегов):





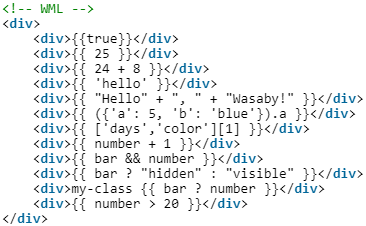
### Спецсимволы HTML

Для отображения спецсимволов html рекомендуется либо вставлять в шаблон сам символ, либо его числовой код. В следующем примере в обоих случаях в вёрстку страницы выводится символ рубля:

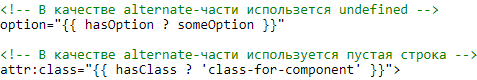
 будет преобразовано в 

### Использование JavaScript в шаблоне

Шаблон поддерживают работу с выражениями языка JavaScript: арифметические операции, операторы сравнения и условные операторы, использование переменных из области видимости шаблона:



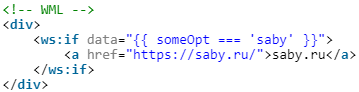
В шаблонах Wasaby Framework тернарный оператор может использовать без alternate-части:



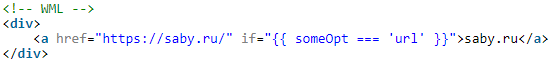
### Условия

**Директива ws:if**

Директива <ws:if> позволяет добавлять html-разметку при выполнении условия, описанного в атрибуте data. Значением атрибута является конструкция шаблонизатора. Когда атрибут принимает значение true, в DOM вставляется содержимое блока <ws:if>:



Короткая нотация:



### Директива ws:else

Директиву <ws:if> можно дополнить блоками <ws:else>. Для всех блоков, кроме последнего, допускается создание условия в атрибуте data:



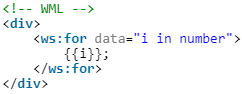
Между блоками <ws:if> и <ws:else> запрещается встраивание разметки!

### Циклы

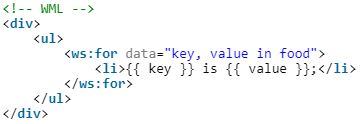
**Директива ws:for**

В атрибуте data допускается организовывать итерации по целочисленному диапазону, объектам, массивам и экземплярам классов Types/entity:Model или Types/entity:Record. Для каждого типа данных используется собственная форма записи.

* Итерация по целочисленному диапазону или элементам массива:



* Итерация по свойствам объекта или элементам массива:



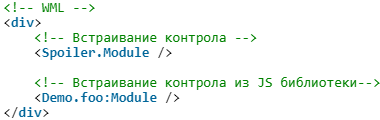
Директива создаёт новую область видимости переменных. Переменные видны только внутри оператора, но не за его пределами.

### Встраивание контрола или внешнего шаблона

**По имени**

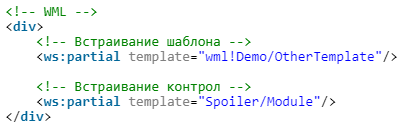
Встроенный таким образом контрол будет автоматически загружен Wasaby, поэтому нет необходимости указывать его в массиве зависимостей модуля.

«/» - заменяют на символ точки



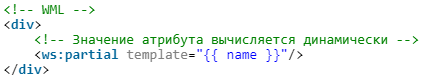
### Директива ws:partial

Используется для добавления в шаблон контрола или другого шаблона. В атрибут template передают имя контрола или шаблона:



Так же нет необходимости указывать добавленный таким образом контрол в массиве зависимостей модуля.

Значение атрибута можно вычислять динамически с помощью синтаксических конструкций. Однако в этом случае не происходит автоматической загрузки зависимости:

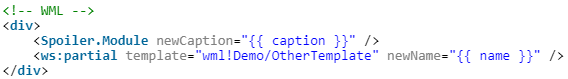


### Передача опций

Добавленный контрол/шаблон создаёт изолированную область видимости, в которой недоступны переменные родительского шаблона. Но эти переменные можно передать в опциях.

В шаблоне опции доступны в переменной \_options, а в JS-модуле — в переменной this.\_options.

Пример: в области видимости родительского шаблона существует переменная caption и name. С помощью опций newCaption и newName в мы передали значения переменных:



**Синтаксис XML для опций**

Общая форма:



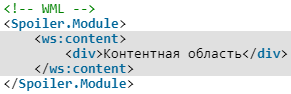
Пример:

<https://codesandbox.io/embed/configuration-options-template-syntax-x5bz4?fontsize=14&hidenavigation=1&theme=dark>

**Контентная опция**

Предназначена для шаблона, который можно задать по месту при встраивании контрола. Если название контентной опции в контроле не задано, то по умолчанию используется название "content".

Пример: Встраиваение шаблона в контентную опцию контрола Spoiler.Module:



Пример 2:



Шаблон контрола ListView:



Контентные опции используются в компонентах для вставки заданного шаблона через конструкцию ws:partial

### Передача атрибутов

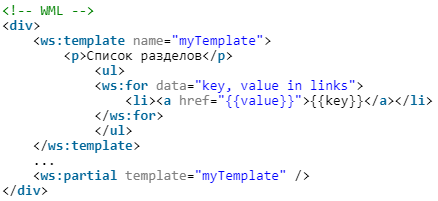
Для встроенного контрола/шаблона используется для обозначения атрибутов префикс attr. Описанные таким образом атрибуты будут добавлены на корневой DOM-элемент:



### Директива <ws:template>. Встроенный шаблон

С помощью этой директивы вы можете объявить встроенный шаблон - фрагмент вёрстки, предназначенный для локального использования.

* Шаблон объявляют до момента его использования
* В атрибуте name задают имя шаблона
* Для добавления встроенного шаблона используют директиву <ws:partial>



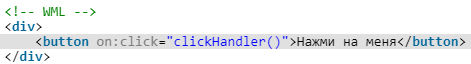
## Работа с событиями

Для подписки на событие используют директиву **on**. Можно подписаться на события элементов шаблона:

* DOM-элемент;
* контрол;
* шаблон, встроенный директивой ws:partial.

Директива on имеет следующую форму записи: on:<событие>="<обработчик>(<параметры>)"

### Подписка на нативное событие DOM-элемента:



JS:



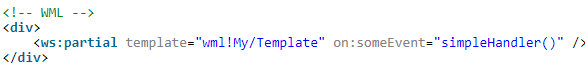
### Подписка на событие контрола:



JS:

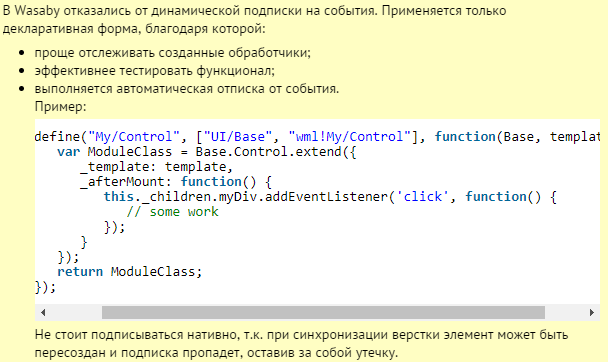


### Подписка на событие шаблона:



JS:





### Передача параметров события из шаблона



TS:



**е** – это дескриптер события (экземпляр класса Vdom/Vdom:SyntheticEvent), он всегда приходит первым аргументом. Он содержит свойства:

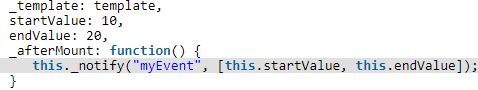
* *nativeEvent* — содержимое этого свойства зависит от типа события. Для нативного события содержит поля интерфейса Event. Принимает null, если событие публикуется контролом.
* *result* — значение, возвращенное из предыдущего обработчика, который выполнен на то же событие.
* *stopped* — признак, что не происходит дальнейшее всплытие события по цепочке.
* *tar*g*et* — DOM-элемент, в котором произошло событие.
* *type* — имя события.

### Публикация пользовательского события

Для публикации события в контроле вызывают метод \_notify(). Первый аргумент метода — обязательный, он принимает имя события.

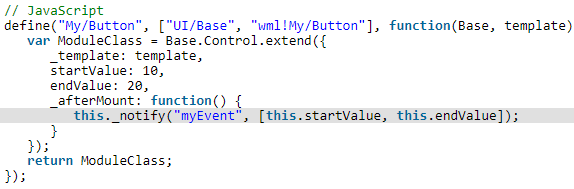


Во втором аргументе метода \_notify() можно передать массив параметров. Они будут доступны в аргументах обработчика события.

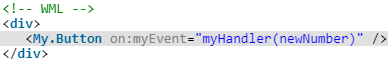


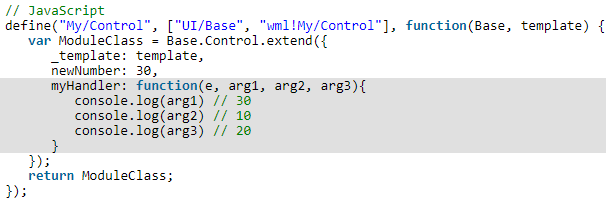
### Приоритет обработки параметров события

В аргументах обработчика события сначала доступны параметры из шаблона, а далее — параметры из модуля, переданные в момент публикации.



Control.wml:

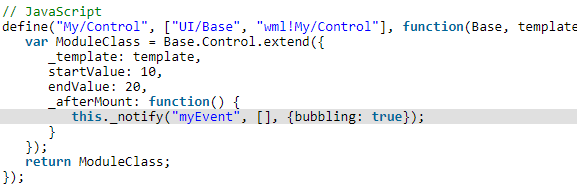




### Всплываемость

При публикации события его можно сделать всплываемым. Тогда оно будет передаваться из контрола (источника события) вверх по всей цепочке родительских **элементов шаблона**. Данное поведение можно использовать для перехвата события и дальнейшей обработки.

Для этого следует в методе \_notify() передать третьим аргументом значение {bubbling: true}.



## Синхронизация опций

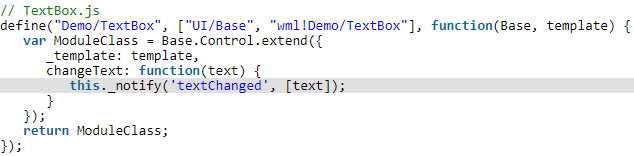
### bind

Wasaby по умолчанию синхронизирует изменения опций в одном направлении — из родительского контрола к дочерними. Для создания двусторонней синхронизации используют директиву bind.



То есть у контрола **TextBox** есть опция **text**, и при ее изменении будет соответственно меняться свойство **myText** у родительского компонента **Module.**

При этом из **TextBox**, должно быть опубликовано событие на изменение значения свойства. Имя публикуемого события должно соответствовать формату **<свойство дочернего контрола>Changed:**



Полная форма записи директивы такова:

bind:<свойство дочернего контрола>="свойство родительского контрола"

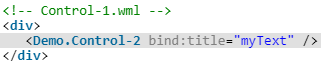
### Зачем нужно событие <свойство дочернего контрола>Changed?

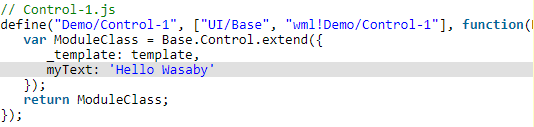
Событие "<свойство дочернего контрола>Changed" может быть предназначено для bind, который объявлен в контроле, который располагается выше родительского контрола. В этом случае нужно выполнить одно из следующих условий:

1. Отправлять событие с параметром bubbling: true
2. Проксировать событие у родительского контрола. Примечание: для проксирования не нужно объявлять bind у родительского контрола. Событие достаточно поймать, а затем — опубликовать такое же. Ниже показан пример как это реализуется для трёх контролов:

* Control 1:

У контрола Control 1 есть свойство myText, которое он передает в опцию title контрола Control 2 и биндит на изменение.





* Control 2:

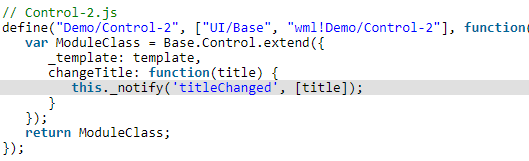
Контрол2 прокидывает в опцию title Control 3 свою опцию title и подписывается на события on:titleChanged Контрола 3, которое будет срабатывать при изменении в нем опции title

<!-- Control-2.wml -->

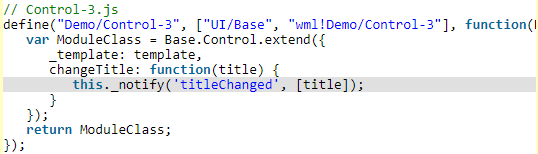
**<div>**

**<Demo.Control-3 title="{{ \_options.title }}" on:titleChanged="changeTitle()" />**

**</div>**



* Control 3:



# TypeScript

## Жизненный цикл

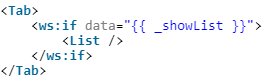
Есть три основных фазы жизненного цикла: создание, обновление и уничтожение

### Фаза «Создание»

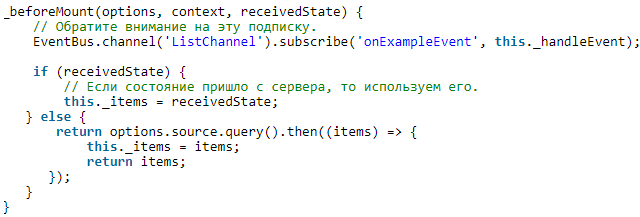
|  |  |  |
| --- | --- | --- |
| **Сигнатура хука** | **Предназначение** | **Момент вызова** |
| \_beforeMount(options, context, receivedState) | Служит для инициализации состояния по полученным опциям.  Избегайте побочных эффектов в этом методе, потому что они могут привести к утечкам памяти. Подробнее читайте в статье: [Опасность побочных эффектов в \_beforeMount](https://wi.sbis.ru/doc/platform/developmentapl/interface-development/ui-library/control/#danger-of-before-mount-side-effects).  Если из этого хука вернуть Promise, то построение этого контрола отложится до его разрешения. Результат Promise, выполненного на сервере, придёт третьим аргументом при построении на клиенте. Более подробно смотрите в [примере](https://wi.sbis.ru/doc/platform/developmentapl/interface-development/ui-library/control/#async-build). | Вызывается до построения вёрстки контрола. Соответственно, здесь ещё нет DOM и детей.  **Единственный хук, который вызывается и на сервере, и на клиенте**. |
| \_componentDidMount(options) | Служит для манипуляций с DOM **перед тем, как пользователь увидит кадр.** Например, для корректирования положения скролла.  **Хук вызывается синхронно**, поэтому может вызвать проблемы с производительность. Поэтому предпочтительнее использовать \_afterMount(), чтобы не откладывать отрисовку кадра | Вызывается синхронно после того, как изменения были **впервые** применены к DOM **до того, как браузер отрисовал кадр.** |
| \_afterMount(options) | **Служит для инициализации состояния, которое зависит от DOM, а также подписок на события.**  Не стоит без необходимости изменять реактивные свойства в данном методе, т.к. это приведёт к перерисовкам. | **Вызывается после построения вёрстки и отрисовки кадра браузером.**  Вызывается один раз после построения на клиенте. |

**Опасность побочных эффектов в beforeMount:**

Пример: Есть шаблон:



\_beforeMount у List выглядит вот так:



За то время, пока List ходит за данными (ждет разрешения промиса), состояние приложения изменяется, и Tab умирает. В то же время, у List \_beforeUnmount не вызовется, потому что не успел вмонтироваться. В следствии, в \_beforeUnmount не произойдет отписка.

*Чтобы избежать этой проблемы, все подписки нужно делать в \_afterMount()*

**Загрузка данных дважды: на сервере и на клиенте**

Если пришел recivedState, то промис возвращать не надо!

### Фаза "Обновление"

|  |  |  |
| --- | --- | --- |
| **Сигнатура хука** | **Предназначение** | **Момент вызова** |
| \_beforeUpdate(newOptions) | **Служит для изменения состояния в ответ на изменение опций**. Старые опции доступны на this.\_options, новые приходят первым аргументом.  **Не забывайте сравнивать старые и новые опции перед изменением состояния.** Более подробно о том, к каким проблемам это приведёт, читайте в статье: [Контрол безусловно меняет состояние в \_beforeUpdate](https://wi.sbis.ru/doc/platform/application-debugging/js/debugging-common-errors/change-state/). | Вызывается перед тем, как контрол обновится с новыми опциями. |
| \_afterRender() | **Служит для манипуляций с DOM перед тем, как пользователь увидит кадр.** Например, для корректирования положения скролла.  Использование данного хука обычно приводит к проблемам с производительностью, например, к [forced reflow](https://wi.sbis.ru/doc/platform/application-debugging/js/debugging-common-errors/forced-reflow/). Поэтому для большинства операций предпочтительнее использовать \_afterUpdate(), чтобы не откладывать отрисовку кадра. | Вызывается синхронно после того, как изменения были применены к DOM до того, как браузер отрисовал кадр. |
| \_afterUpdate(oldOptions) | Служит для изменения состояния после обновления. Здесь можно взаимодействовать с DOM, например, измерять размеры элементов.  **Чтобы избежать зацикливания синхронизаций, меняйте состояние только при наличии изменений.** | Вызывается после обновления контрола и, в отличие от \_afterRender(), уже после того, как кадр был отрисован.  Этот метод вызывается **только** после обновлений. Если вам нужно что-то сделать после построения, то используйте \_afterMount(). |

### Фаза "Уничтожение"

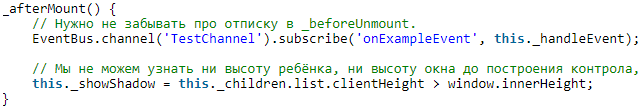
|  |  |  |
| --- | --- | --- |
| **Сигнатура хука** | **Предназначение** | **Момент вызова** |
| \_beforeUnmount() | **Служит для очистки состояния контрола, отписок от событий и т.д.** После вызова этого метода любое взаимодействие с контролом (обращения к свойствам, вызовы методов и т.д.) является ошибкой.  В этом методе не нужно зачищать поля контрола, т.к. Wasaby делает это автоматически. | Вызывается перед уничтожением контрола. |

### Примеры использования хуков

* **\_beforeMount**

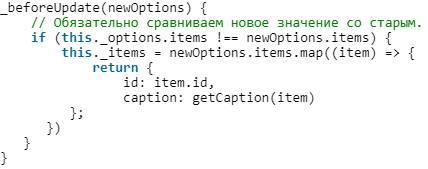
Понятно

* **\_afterMount**

****

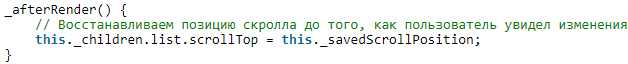
В этом методе DOM уже вмонтирован и показан и мы можем на основании высоты элементов высчитывать значение тени.

* **\_beforeUpdate**

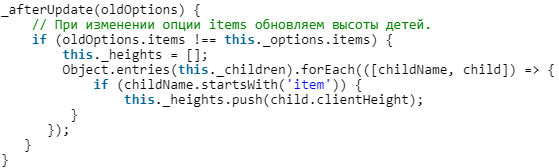
****

Если пришли новые значения опций – то на их основании перерасчитаем реактивные свойства

* **\_afterRender**

****

* **\_afterUpdate**

****

* **\_beforeUnmount**

****