Немного про доступность и ARIA

The Web Accessibility Initiative's Accessible Rich Internet Applications Suite (*WAI-ARIA*, или просто *ARIA*) — это стандарт доступности активных интернет-приложений и набор инструментов для обеспечения этой доступности.

Сейчас нас интересуют в первую очередь те атрибуты, которые позволят увеличить семантичность HTML-кода, не дублировать контент и улучшить доступность информации через «вспомогательные возможности» (*assistive technologies*).

Этот стандарт доступности в первую очередь нужен для динамических веб-приложений, чтобы сделать эту динамику доступной для пользователей с разными ограничениями. Но и на уровне фиксированного макета есть некоторые вопросы доступности, которые было бы хорошо решить этим методом.

ARIA напоминает атрибут `title`, который добавляет всплывающую подсказку, тем, что также не влияет на работоспособность сайта: это чистой воды оптимизация.

Приведём классический пример использования *ARIA*: вам нужно сделать кнопку, но по каким-то причинам это невозможно выполнить при помощи тега `<button>`. Вы делаете кнопку как `<span>`, но пишете: `<span role="button" aria-checked="false" tabindex="0">`, то есть указываете роль этого элемента, подменяете функционал кнопки (это ведь активный элемент, он может быть отмечен или нет!) и порядок работы подложной кнопки при клике. При этом использовать `role="button"` в случаях, когда роль элемента и без того указана в семантике, не следует. Например, для `<button type="button">` это не нужно.

При этом правила использования технологии *ARIA* гласят, что если можно использовать семантически корректную вёрстку, то нужно верстать теги и не подменять их. Но есть нюансы. Например, такой подход использует система *Bootstrap*.

Но вернёмся к нашим неадаптивным макетам. Какие возможности предоставляет *ARIA* в целом, какие могли бы нам пригодиться, для чего и в каком виде, как правильно их применять и когда лучше этого не делать?

Возможности и задачи *ARIA* по спецификации

- Роли для описания типов веб-элементов, например `menu`, `lng` или `slider`.
- Роли для описания структуры веб-страницы, такой как заголовки, области и таблицы (сетки), например `gridcell` или `td`, но это нужно очень аккуратно применять.
- Свойства, описывающие состояние элементов, например `checked` (выбран) для флажка или `haspopup` (имеет всплывающее окно, поп-ап) для меню.
- Обеспечение навигации с помощью клавиатуры для отдельных элементов.
- Описание динамических элементов, получающих данные с бэкенда.

Атрибуты и значения

Про все атрибуты можно почитать в спецификации. А нас сейчас интересуют следующие: `role`, а также `aria-describedby` и `aria-labelledby`.

Роль, атрибут `role` — это то, какую функцию выполняет элемент. Список ролей с их описаниями лежит в спецификации.

Некоторые роли *ARIA* как бы подменяют семантичные теги, например, `role="article"` означает то же самое, что и тег `<article>`; `role="checkbox"` заменяет `<input type="checkbox">`, `role="button"` позволяет использовать `<div>` или `<span>` вместо `<button>` при соблюдении некоторых условий (в первую очередь — при условии досконального описания всего функционала, соответствующего семантическому элементу) и так далее. Но не всегда значение так очевидно. Так, `role="grid"` означает «сетка, таблица», и подразумевает, что содержимое взаимосвязано.

А ещё бывают роли, значение которых уникально.

Роль `role="img"` можно использовать для того, чтобы маркировать содержимое контейнера как единый визуальный объект. Например, если текст и эмодзи следует воспринимать вместе, как одно изображение, то такая роль это покажет.

```
<div role="img" aria-label="Описание вложенных изображений">
  <img src="pic1.png" alt="">
  <img src="pic2.png" alt="">
</div>
```

Попробуем разобраться с атрибутом `aria-labelledby`, буквально «помечен тем, что». Он связывает элементы, например, виджеты, метки или поля ввода форм с теми элементами, которые представляют собой их лейблы, если не получается удобно подставить лейбл или описание в правильном месте. Элемент, который сам является лейблом, получает атрибут `id`, а помеченный элемент — `aria-labelledby`.

Работе с формами можно научиться в тренажёрах Часть 3: Формы. Знакомство и Часть 4: Формы. Погружение.

В чём разница между `aria-labelledby` и атрибутом `aria-describedby`, который в общем-то тоже связывает объект с его описанием? В многословности этого описания. `aria-labelledby` — это расшифровка, описание сущности элемента, а `aria-describedby` — дополнительная информация. Если грубо, то *лейбл* короткий, описание — длинное.

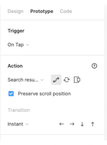
Приведём несколько примеров.

```
<!-- Роль подменяет тег, это должен был быть список определений, id показывает лейбл, помеченные
элементы соответственно содержат атрибут aria-labelledby -->
<dl>
  <dd id="online">Веб-контент</dd>
  <dd role="definition" aria-labelledby="online">Опубликован во всемирной сети интернет</dd>
  <dd role="definition" aria-labelledby="online">Контент, доступный при подключении к сети</dd>

  <dd id="offline">Печатная продукция</dd>
  <dd role="definition" aria-labelledby="offline">Книги, журналы и распечатки</dd>
  <dd role="definition" aria-labelledby="offline">Доступный всегда и при всяких обстоятельствах
контент</dd>
</dl>

<!-- У элемента есть и лейбл и описание, видна разница между этими атрибутами -->
<div role="application" aria-labelledby="calendar" aria-describedby="info">
  <h1 id="calendar">Календарь</h1>
  <p id="info">
    Вот наш план разработки в деталях и с примерными датами.
  </p>
  <div role="grid">
      ...
  </div>
</div>
```

А если для задачи нет правильных тегов, то роль *ARIA* становится ещё более монументальной. Например, табы.



Вкладки в браузерной версии Фигмы

В целом это, конечно, список ссылок. Но и список табов в том числе. Условная вёрстка может выглядеть так:

```
<ul class="page-menu-list">
  <li class="page-menu-iten page-menu-iten-selected">
    <a class="page-menu-link" href="#">Design</a>
  </li>
  <li class="page-menu-iten">
    <a class="page-menu-link" href="#">Prototype</a>
  </li>
  <li class="page-menu-iten">
    <a class="page-menu-link" href="#">Code</a>
  </li>
</ul>
...
<div class="article-content">
  <p>...</p>
</div>
<div class="article-content" hidden>
  <p>...</p>
</div>
<div class="article-content" hidden>
  <p>...</p>
</div>
```

Но как связать эти элементы между собой? Если задать роли, атрибут *ARIA* `aria-selected` и атрибут `aria-labelledby`, это свяжет между собой табы с их контентом, сделает структуру прозрачнее и вообще красиво опишет функционал несемантичных тегов. У технологии *ARIA* есть отдельные роли для группы табов и для каждого отдельного: это `role="tablist"` и `role="tab"` соответственно. А `role="tabpanel"` описывает контент таба, то, что вложено. Атрибут `aria-controls` укажет элемент, содержанием которого управляет элемент с этим атрибутом.

Итак, что у нас получилось с ролями и прочими атрибутами.

```
<ul class="page-menu-list" role="tablist">
  <li class="page-menu-iten page-menu-iten-selected" role="tab" aria-selected="true" aria-
controls="panel-1" id="tab-1">
    <a class="page-menu-link" href="#">Design</a>
  </li>
  <li class="page-menu-iten" role="tab" aria-selected="false" aria-controls="panel-2" id="tab-2">
    <a class="page-menu-link" href="#">Prototype</a>
  </li>
  <li class="page-menu-iten" role="tab" aria-selected="false" aria-controls="panel-3" id="tab-3">
    <a class="page-menu-link" href="#">Code</a>
  </li>
</ul>
...
<div class="article-content" id="panel-1" role="tabpanel" aria-labelledby="tab-1">
  <p>...</p>
</div>
<div class="article-content" id="panel-2" role="tabpanel" aria-labelledby="tab-2" hidden>
  <p>...</p>
</div>
<div class="article-content" id="panel-3" role="tabpanel" aria-labelledby="tab-3" hidden>
  <p>...</p>
</div>
```

Основные правила использования *ARIA*

- *ARIA* используется только если HTML-разметка не позволяет показать всю необходимую семантику, или когда теги используются нетипично. Например, для установки связей между разными элементами, их группировки.
- Помогающая технология нужна для описания свойств, которые могут меняться из-за действий пользователей: например, выбор, клик, размер букв и так далее. Это связано с бэкендом — но технология для этого хорошо подходит.
- Важно использовать свойства и их значения максимально точно: не путать роли, не переопределять значение элементов (не придавать роль кнопки заголовку, например).

Ознакомились со статьёй?

Сохранить прогресс