Отзывчивый дизайн

На заре веб-дизайна страницы создавались для экрана определённого размера. Если у пользователя был экран большего или меньшего размера чем ожидал дизайнер, то результат мог быть от нежелательных полос прокрутки, до слишком длинной строки и плохого использования пространства. Поскольку становились доступны много различных размеров экранов, появилась концепция отзывчивого (адаптивного) веб-дизайна (responsive web design (RWD)) — набор методов, которые позволяют веб-страницам менять свой макет и внешний вид в соответствии с разной шириной экрана, разрешением и т.д. Это та самая, идея которая изменила подход к дизайну веба для множества устройств, и в этой статье мы поможем вам понять основные методы, которые вам необходимо знать, чтобы освоить его.

Необходимые знания:	Основы HTML (изучите Введение в HTML), идея о том как работает CSS (изучите Введение в CSS и Устройство CSS.)
Задача:	Понять базовые концепции и историю отзывчивого дизайна.

Исторические макеты сайтов

В какой-то момент истории при разработке веб-сайта у вас было два варианта:

- Вы могли создать жидкий сайт, который будет растягиваться чтобы заполнить окно браузера
- или сайт с фиксированной шириной, который будет иметь фиксированный размер в пикселях.

Эти два подхода, как правило, приводили к тому, что веб-сайт лучше всего выглядел на экране человека, создавшего сайт! Жидкий сайт приводил к раздавленному дизайну на маленьких экранах (как видно ниже) и не читаемо длинным строкам на больших.

Попробуй

Примечание: Посмотрите этот простой жидкий макет: пример, исходный код. При просмотре примера, растягивайте и сжимайте окно браузера чтобы увидеть, как это выглядит при разных размерах.

Сайт с фиксированной шириной рисковал иметь горизонтальную полосу прокрутки на экранах меньших чем ширина сайта (как видно ниже) и много белого пространства на краях дизайна на больших экранах.

Примечание: Посмотрите этот простой макет с фиксированной шириной: пример, исходный код. Снова изучите результат по мере изменения размера окна браузера.

Примечание: Скриншоты выше сделаны используя Responsive Design Mode в Firefox DevTools.

Когда мобильный веб стал становиться реальностью с первыми функциональными телефонами, компании желающие охватить мобильники начали создавать в основном специальные мобильные версии своих сайтов, с различными URL (часто что-то наподобие m.example.com или example.mobi). Это означало, что необходимо было разрабатывать и поддерживать в актуальном состоянии две отдельные версии сайта.

Кроме того, эти мобильные сайты часто предлагали очень урезанный вариант. Поскольку мобильные гаджеты стали мощнее и способными отображать целые веб-сайты, пользователей мобильных устройств раздражало, что они обнаруживали себя запертыми в мобильной версии сайта, неспособные получить доступ к информации, которая, как они знали, есть в полнофункциональной версии сайта.

Гибкий макет до отзывчивого дизайна

Было разработано несколько подходов чтобы попытаться разрешить недостатки построения веб-сайтов жидким методом или методом с фиксированной шириной. В 2004 году Камерон Адамс написал пост Resolution dependent layout, описывающий метод создания дизайна который мог бы адаптироваться к разным разрешениям экрана. Этот подход требовал, чтобы JavaScript узнавал разрешение экрана и загружал корректный CSS.

Зои Миккели Гилленвотер сыграла важную роль в свой работе описав и формализовав различные способы посредствам которых могут быть созданы гибкие сайты, пытаясь найти золотую середину между заполнением экрана или полностью фиксированным размером.

Отзывчивый дизайн

Термин адаптивный дизайн был придуман Итаном Маркоттом в 2010 году и описывал использование трёх методов в сочетании.

1. Первой была идея жидких сеток, нечто что уже исследовала Гилленвотер, что можно прочитать в статье Маркотта - Fluid Grids (опубликовано в 2009 в A List Apart).
2. Вторым методом была идея жидких изображений. Используя очень простой метод настройки свойства max-width на 100%, изображения будут становиться меньше если содержащий столбец становится уже чем изначальный размер изображения, но никогда

не становится больше. Это позволяет изображению уменьшаться чтобы соответствовать столбцу гибких размеров, а не перекрываться с ним, но не расти и становиться пиксельным если столбец становится шире изображения.
3. Третьим ключевым компонентом были медиавыражения. Медиавыражения позволяют переключать тип макета применяя только CSS то, что Камерон Адамс исследовал, используя JavaScript. Вместо того чтобы иметь один макет для всех размеров экранов, макет мог изменяться. Боковые панели можно перемещать для маленьких экранов, либо отображать альтернативную навигацию.

Очень важно понять, что **адаптивный веб-дизайн — это не отдельная технология**, это термин используемый, чтобы описать подход к веб-дизайну или набор лучших практик, используемых для создания макета, который может реагировать на используемое устройство для просмотра контента. В первоначальном исследовании Маркотта это означало гибкие сетки (с использованием floats) и медиавыражения, однако почти за 10 лет, прошедших с момента написания этой статьи, адаптивная работа стала стандартом по умолчанию. Современные методы макета CSS отзывчивы по своей сути, и у нас есть новые штучки, встроенные в веб-платформу для того, чтобы делать дизайн отзывчивых сайтов проще.

Остальная часть этой статьи укажет вам на различные функции веб-платформы, которые вы, возможно, захотите использовать при создании адаптивного сайта.

Медиавыражения

Отзывчивый дизайн появился благодаря медиавыражениям (media queries). Спецификация Media Queries Level 3 стала Рекомендованным Кандидатом в 2009 году, что означает, что она была признана готовой к реализации в браузерах. Медиавыражения позволяют нам проводить серию тестов (например, является ли экран пользователя больше, чем определённая ширина или разрешение) и выборочно применять CSS к стилю страницы соответственно с нуждами пользователя.

Например, следующее медиавыражение проверяет отображается ли текущая страница как экранная медиа (а не как печатный документ) и имеет ли область просмотра ширину как минимум 800 px. CSS будет применяться к селектору '.container' только если эти две вещи истины.

CSS

```
@media screen and (min-width: 800px) {
  .container {
    margin: 1em 2em;
  }
}
```

Вы можете добавлять несколько медиавыражений в пределах одной таблицы стилей, подстраивая весь ваш макет или его части так, чтобы наилучшим образом соответствовать разным размерам экрана. Точки, в которых применяются медиавыражения и меняется макет, известны как контрольные точки.

Общим подходом при использовании медиавыражений является создание простого одноколоночного макета для устройств с узкими экранами (например, мобильные телефоны), затем проверка для больших экранов и применение макета с несколькими столбцам, когда вы знаете, что у вас достаточно ширины экрана, чтобы уместить все. Такой подход часто называют mobile first дизайном.

Узнать больше о медиавыражениях можно в документации MDN.

Гибкие сетки

Отзывчивые сайты не просто меняют свой макет между контрольными точками, они построены на гибких сетках. Гибкая сетка подразумевает что вам не надо заботиться о каждом возможном существующем размере устройства и строить для них идеальный макет в пикселях. Такой подход был бы невозможен имея широкое множество существующих устройств разных размеров, как и факт того, что даже на ПК люди не всегда используют браузер с развёрнутым до максимума окном.

Используя гибкую сетку, вам всего лишь надо добавить контрольную точку и изменить дизайн в точке, когда ваш контент начинает выглядеть плохо. Например, если длина строки становится нечитаемо длинной при увеличении размера экрана, или блок становится сдавленным с двумя словами в каждой строке при сужении экрана.

В первые дни отзывчивого дизайна, нашим единственным вариантом выполнения было использование floats. Гибкий обтекаемый макет достигался путём присвоения каждому элементу процентной ширины удостоверившись, что итоговые значения в макете не превышают 100%. В своей оригинальной статье о плавучих сетках Маркотт подробно описал формулу для преобразования макета, созданного с использованием пикселей, в проценты.

```
target / context = result
```

Например, если размер нашего целевого столбца — 60 пикселей, а контекст (или контейнер) в котором он находится — 960 пикселей, то мы делим 60 на 960 чтобы получить значение которое мы можем использовать в нашем CSS, после переноса десятичной точки вправо на 2 цифры.

CSS

```
.col {
  width: 6.25%; /* 60 / 960 = 0.0625 */
}
```

Этот подход сегодня можно найти во многих местах в Интернете и он задокументирован здесь в разделе макетов в нашей статье устаревших методов макетов. В вашей работе вероятно, что вы столкнётесь с веб-сайтами, использующими этот подход, поэтому стоит понимать его, даже если вы не будете строить современные сайты используя гибкие сетки основанные на float.

Следующий пример демонстрирует простой отзывчивый дизайн, с использованием

медиавыражений и гибких сеток. На узких экранах макет отображает блоки, расположенные друг над другом. На более широких экранах они перемещаются в два столбца:

Примечание: вы можете найти пример и исходный код этого примера на GitHub.

Современные технологии макетов

Современные методы макетов такие как Макет с несколькими столбцами, Flexbox, и Grid являются отзывчивыми по умолчанию. Они все предполагают, что вы пытаетесь создать гибкую сетку и дают вам более лёгкий способ сделать так.

Multicol

Самый старый из этих методов — это `multicol`, когда вы задаёте `column-count`, это отражает то на сколько столбцов вы хотите разбить ваш контент. Далее браузер рассчитывает их размер, размер, который изменится согласно размеру экрана.

CSS

```
.container {
  column-count: 3;
}
```

Если вместо этого вы зададите `column-width`, то вы определите минимальную ширину. Браузер создаст столько столбцов той ширины, сколько будет комфортно умещаться в контейнер, а затем поделит оставшееся пространство между всеми столбцами. Поэтому число столбцов будет меняться согласно тому сколько имеется места.

CSS

```
.container {
  column-width: 10em;
}
```

Flexbox

В Flexbox, в качестве исходного поведения, flex элементы будут сжиматься и распределять пространство между элементами в соответствии с пространством в их контейнере. Изменяя значения `flex-grow` и `flex-shrink` вы можете указать, как вы хотите, чтобы предметы вели

себя когда они сталкиваются с большим или меньшим пространством вокруг себя.

В примере ниже каждый flex элемент будет принимать равное количество пространства во flex контейнере используя запись `flex: 1` как описано в главе Flexbox: Гибкое изменение размеров flex элементов.

CSS

```
.container {
  display: flex;
}
.item {
  flex: 1;
}
```

Примечание: В качестве примера мы перестроили простой отзывчивый макет выше, в этот раз используя flexbox. Вы видите что нас больше не надо использовать странные процентные значения для подсчёта размера столбцов: пример, исходный код.

CSS grid

В макете CSS Grid единицы измерения `fr` позволяют распределять доступное пространство между дорожками сетки. Следующий пример создаёт grid контейнер с тремя дорожками размером `1fr`. Это создаст три вертикальные дорожки, каждая занимающая одну часть свободного пространства в контейнере. Вы можете узнать больше об этом подходе к созданию сетки в теме Изучение Макета Grid в разделе Гибкие grids с единицами fr.

CSS

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

Примечание: версия grid макета ещё проще, поскольку мы можем определить столбцы в .wrapper: пример, исходный код.

Отзывчивые изображения

Самый простой подход к отзывчивым изображениям был описан в ранних статьях Маркотта по отзывчивому дизайну. По сути, вы берёте изображение максимального размера, которое могло понадобиться, и уменьшаете его. Этот подход до сих пор используется и в большинстве таблиц стилей вы найдёте следующий CSS:

CSS

```css
img {
  max-width: 100%;
}
```

Существуют очевидные недостатки к этому подходу. Изображение может быть изображено намного меньше своего исходного размера, что является пустой тратой пропускной способности — пользователь мобильных может загружать изображение, в несколько раз превышающее размер того, что он фактически видит в окне браузера. Кроме того, вам может не понадобиться такое же соотношение сторон изображения на мобильном устройстве, как на компьютере. Либо, учитывая меньший размер изображения на мобильном телефоне, вы можете захотеть показать совсем другое изображение, которое легче понять на маленьком экране. Такие вещи можно достичь, просто уменьшая изображение.

Отзывчивые изображения, используя элемент <picture> и атрибуты `srcset` и `sizes` элемента <img> оба решают эти проблемы. Вы можете указать несколько размеров вместе с «подсказками» (метаданные, описывающие размер экрана и разрешение, для которых изображение лучше всего подходит), и браузер выберет наиболее подходящее изображение для каждого устройства, гарантируя, что пользователь загрузит изображение подходящего размера для устройства, которое они используют.

Вы также можете напрямую использовать изображения разных размеров, обеспечивая разное кадрирование или совершенно другое изображение для разных размеров экрана.

Вы можете найти подробное руководство по отзывчивым изображениям в разделе изучения HTML.

Отзывчивая типографика

Элементом отзывчивого дизайна, не освещённого ранее в работе, была идея отзывчивой типографики. Главным образом, она описывает изменение размеров шрифта в зависимости от ширины экрана при помощи медиавыражений.

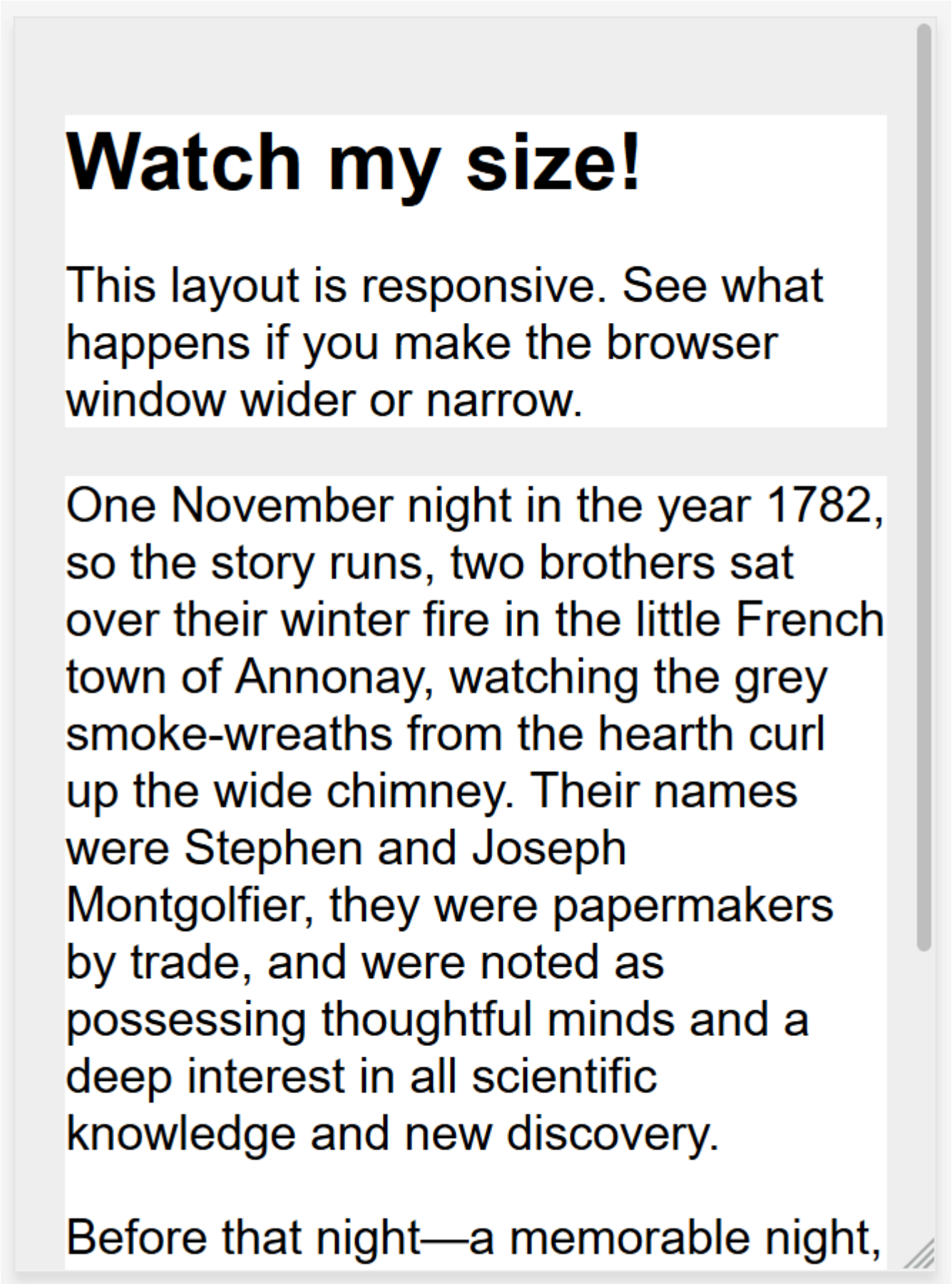
В этом примере, мы хотим задать нашему заголовку первого уровня `4rem`, что значит, что он будет в четыре раза больше нашего базового размера шрифта. Это очень большой заголовок! Мы хотим этот гигантский заголовок только на экранах больших размеров, поэтому мы сначала создаём меньший заголовок, а затем используем медиавыржаение, чтобы переписать его для больших экранов, если мы знаем, что у пользователя есть экран размером как минимум `1200px`.

CSS

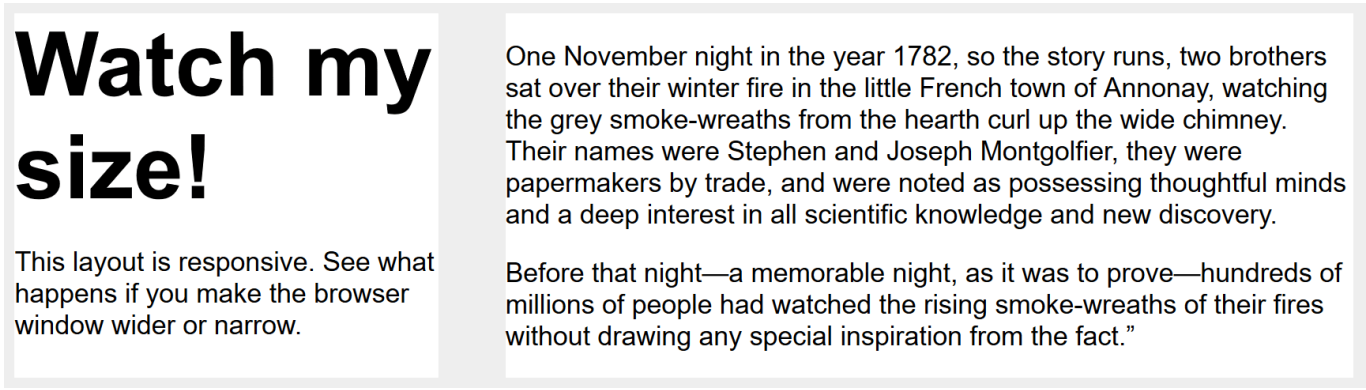
```
html {
  font-size: 1em;
}

h1 {
  font-size: 2rem;
}

@media (min-width: 1200px) {
  h1 {
    font-size: 4rem;
  }
}
```

Мы отредактировали наш приведённый выше пример отзывчивой сетки grid, чтобы он также включал в себя адаптивный тип, используя описанный метод. Вы можете видеть, как заголовок меняет размеры, когда макет переходит в версию с двумя столбцами.

В мобильных версиях заголовок меньше:



На компьютерах, однако, мы видим больший размер заголовка:



Примечание: смотрите этот пример в действии: пример, исходный код.

Такой подход к типографике показывает, что вам не нужно ограничиваться в использовании медиавыражений только изменением макета страницы. Они могут быть использоваться для настройки любого элемента, чтобы сделать его более удобным или привлекательным при других размерах экрана.

Использование единиц просмотра для адаптивной типографики

Интересный подход — использовать единицу области просмотра vw для обеспечения адаптивной типографики. 1vw равен одному проценту ширины области просмотра. Это означает, что если вы установите размер шрифта с помощью vw, он всегда будет соответствовать размеру области просмотра.

CSS

```
h1 {
  font-size: 6vw;
}
```

Проблема с выполнением вышеописанного заключается в том, что пользователь теряет возможность масштабировать любой текстовый набор с помощью единицы измерения vw, поскольку этот текст всегда связан с размером области просмотра. **Поэтому никогда не следует задавать текст, используя только единицы просмотра.**

Решение есть, и оно включает использование calc(). Если вы добавите единицу измерения vw в набор значений, используя фиксированный размер, например ems или rems, тогда текст по-прежнему можно будет масштабировать. По сути, модуль vw добавляет к этому увеличенному значению:

CSS

```
h1 {
  font-size: calc(1.5rem + 3vw);
}
```

Это означает, что нам нужно указать размер шрифта для заголовка только один раз, а не настраивать его для мобильных устройств и переопределять в медиа-запросах. Затем шрифт постепенно увеличивается по мере увеличения размера области просмотра.

Примечание: смотрите этот пример в действии: пример, исходный код.

Метатег области просмотра

Если вы посмотрите на исходный код HTML адаптивной страницы, вы обычно увидите следующий тег <meta> в <head> документа.

HTML

```
<meta name="viewport" content="width=device-width,initial-scale=1" />
```

Этот метатег сообщает мобильным браузерам, что им следует установить ширину области просмотра в соответствии с шириной устройства и масштабировать документ до 100 % от его предполагаемого размера, в результате чего документ будет отображаться в запланированном размере, оптимизированном для мобильных устройств.

Зачем это нужно? Потому что мобильные браузеры склонны лгать о ширине области просмотра.

Этот метатег существует потому, что когда был выпущен оригинальный iPhone и люди начали просматривать веб-сайты на маленьком экране телефона, большинство сайтов не были оптимизированы для мобильных устройств. Поэтому мобильный браузер установит ширину области просмотра на 960 пикселей, отобразит страницу с этой шириной и покажет результат в виде уменьшенной версии макета рабочего стола. Другие мобильные браузеры (например, Google Android) делали то же самое. Пользователи могли увеличивать и перемещать веб-сайт, чтобы просмотреть интересующие их фрагменты, но выглядело это плохо. Вы все равно увидите это сегодня, если вам посчастливится встретить сайт, у которого нет адаптивного дизайна.

Проблема в том, что ваш адаптивный дизайн с точками останова и медиа-запросами не будет работать должным образом в мобильных браузерах. Если у вас узкий макет экрана, ширина области просмотра которого составляет 480 пикселей или меньше, а область просмотра установлена на 960 пикселей, вы никогда не увидите свой узкий макет экрана на мобильных устройствах. Установив `width=device-width` (ширину = ширину устройства), вы заменяете ширину Apple по умолчанию = 960 пикселей (`width=960px`)фактической шириной устройства,

поэтому ваши медиа-запросы будут работать по назначению.

Поэтому вам всегда следует включать приведенную выше строку HTML в заголовок ваших документов.

Есть и другие настройки, которые вы можете использовать с метатегом области просмотра, однако в целом вам следует использовать приведенную выше строку.

- `initial-scale` (начальный масштаб): устанавливает начальный масштаб страницы, который мы установили на 1.
- `height` (высота): устанавливает определенную высоту области просмотра.
- `minimum-scale` (минимальный масштаб): устанавливает минимальный уровень масштабирования.
- `maximum-scale` (максимальный масштаб): устанавливает максимальный уровень масштабирования.
- `user-scalable` (масштабируемый пользователем): предотвращает масштабирование, если установлено значение no (нет).

Вам следует избегать использования `minimum-scale`, `maximum-scale` (минимального и максимального масштаба) и, в частности, установки значения no (нет). для `user-scalable` (пользовательского масштабирования). Пользователям должно быть разрешено увеличивать или уменьшать масштаб настолько, насколько им необходимо; предотвращение этого вызывает проблемы с доступностью.

Резюме

Адаптивный дизайн — это дизайн сайта или приложения, который реагирует на среду, в которой его просматривают. Он включает в себя ряд функций и методов CSS и HTML и теперь, по сути, является тем, как мы создаем веб-сайты по умолчанию. Рассмотрим сайты, которые вы посещаете на своем телефоне — вероятно, довольно необычно встретить сайт, который представляет собой уменьшенную версию для настольного компьютера или на котором вам нужно прокручивать вбок, чтобы найти что-то. Это потому, что сеть перешла на такой подход к адаптивному дизайну.

Также стало намного проще создавать адаптивный дизайн с помощью методов верстки, которые вы изучили на этих уроках. Если вы новичок в веб-разработке сегодня, в вашем распоряжении гораздо больше инструментов, чем на заре адаптивного дизайна. Поэтому стоит проверить возраст любых материалов, на которые вы ссылаетесь. Хотя исторические статьи по-прежнему полезны, современное использование CSS и HTML значительно упрощает создание элегантных и полезных дизайнов, независимо от того, с какого устройства посетитель просматривает сайт.