9.5. Свойство font-display

~ 10 минут

Как работают со шрифтами? Чаще всего их вставляют через Google Fonts или через аббревиатуру `font-face`, не задумываясь, что происходит под капотом. Разберёмся в механизме загрузки и применения шрифтов в браузере и посмотрим, что можно улучшить в «классической» вставке с Google Fonts или `font-face`.

Начнём разбор с вариантов отображения нестандартного шрифта. Есть три основных варианта:

1. **Flash of Invisible Text (FOIT)** или «мелькание невидимого текста». При этом варианте текст на сайте не отображается, пока загружается кастомный шрифт. То есть если открыть сайт, то пока кастомный шрифт не загрузится (тот самый `.woff` или `.woff2` файл), то текста на сайте не будет видно.

2. **Flash of Unstyled Text (FOUT)** или «мелькание неоформленного текста». При этом варианте текст отображается запасным web-безопасным шрифтом или системным шрифтом, пока загружается нестандартный шрифт. Когда загрузится нестандартный, то браузер перерисует все тексты в этот шрифт.

3. **Flash of Faux Text (FOFT)** или «мелькание синтезированного текста». Если уже загрузился кастомный шрифт в жирности 400 и пока не загрузился в жирности 700, то браузер сам попытается сделать жирность 700 из 400. Либо если загрузился шрифт с `font-style: normal` и пока не загрузится `font-style: italic`, браузер будет пытаться сделать из стиля `normal` стиль `italic`.

Мы остановимся подробно на двух вариантах — FOUT и FOFT. Первый вариант (FOIT) плох тем, что пользователь, зайдя на сайт, может попросту не увидеть вообще никакого текста, если вдруг что-то пойдёт не так, и кастомный шрифт не загрузится.

font-display

По умолчанию в браузерах включён режим FOIT, поэтому режим FOUT нужно включать вручную. Этот режим включается довольно просто: достаточно добавить в `@font-face` свойство `font-display`. Посмотрим, какие у этого свойства бывают значения.

`font-display` достаточно хорошо поддерживается всеми актуальными на сегодня браузерами (кроме IE). Это свойство определяет, как будет отображаться шрифт, подключенный через `@font-face`, в зависимости от того, загружается ли он и готов ли к использованию.

Так как мы говорим про процесс загрузки и отображения, то все эти операции связаны со временем. Существует временная шкала отображения шрифтов. Когда браузер встречает элемент с кастомным шрифтом, запускается таймер, который проходит через три периода:

1. Блокировка шрифта. Если браузер за время этого периода не успел загрузить шрифт, то текст будет отрисован невидимым запасным шрифтом в тех элементах, которые используют нестандартный шрифт (то есть текста видно не будет). Если браузер успел загрузить шрифт во время этого периода, то он перерисует элементы с загруженным шрифтом.

2. Подмена шрифта. Если нестандартный шрифт не загружен в тех элементах, которые используют этот шрифт, то будет отображен запасной шрифт. Если браузер успел загрузить шрифт, произойдёт перерисовка загруженным шрифтом.

3. Период отказа. Если до этого момента шрифт так и не загрузился, то он будет помечен браузером как неудачно загруженный, а элементы, которые используют этот нестандартный шрифт, будут отрисованы запасным шрифтом.

Этими периодами можно управлять через значения у свойства `font-display`:

— `auto` — свойство по умолчанию. Браузер сам определяет, как ему загружать кастомный шрифт.

— `block` — для шрифта задаётся короткий период блокировки и ставится бесконечный период подмены. При этом значении текст не будет виден на сайте около трёх секунд (если браузер сможет вообще что-то загрузить). Этот вариант очень похож на работу FOIT, что не совсем уж и хорошо. А если вы используете иконочный шрифт, то совсем плохо, так как если шрифт не загрузится, вместо иконок пользователь увидит квадраты.

— `swap` — для шрифта не задаётся период блокировки совсем и так же, как и у `block`, задаётся бесконечный период подмены. В этом случае сразу будет показан безопасный шрифт, если он определён. Например, если установлено `font-family: Roboto, Arial sans-serif;` браузер сначала покажет всё в Arial, и как только Roboto загрузится сразу же всё перерисует на него.

Можно было бы на этом остановиться, ведь идеальное значение найдено. Наши тексты будут отрисованы в любом случае, если использовать `font-display: swap;`. Но не всё так просто. Существуют и другие значения этого свойства, которые могут помочь в определённых ситуациях.

— `fallback` — для шрифта устанавливаются очень короткий период блокировки (около 100 миллисекунд) и короткий период подмены (около трёх секунд). Такое свойство может понадобиться, если шрифт на сайте имеет второстепенную роль, а главным является контент, который надо читать вдумчиво и без отвлечений. То есть если шрифт не загрузился за этот короткий промежуток (100мс + 3с), то никакой подмены шрифта не будет, и ничего не будет моргать, как при `swap`. Это бывает необходимо, когда пользователь действительно не должен отвлекаться от чтения. При этом, если шрифт так и не загрузился в период подмены и после него, то будет показываться запасной шрифт.

— `optional` — для шрифта устанавливается такой же короткий период блокировки, как и у `fallback` (100мс), но период подмены «обнуляется». То есть если шрифт не успел загрузиться за 100мс, то браузер может либо прервать загрузку шрифта совсем (к примеру, если интернет слабый), либо загрузить его, но показать при следующем входе на страницу или переходе на другую страницу сайта, где также используется шрифт. Если кастомный шрифт не успел загрузиться за период блокировки, далее будет отображаться только запасной шрифт.

Посмотрим работу каждого значения в видео:

Таким образом мы имеем хороший запас различных гибких вариантов работы с кастомным шрифтом.

Как использовать

Если используется `@font-face`, то правило `font-display` прописывается прямо внутри подключения:

```
@font-face {
  font-family: 'Lora';
  font-style: normal;
  font-weight: 400;
  font-display: swap;
  src: url(../fonts/lora400.woff2) format('woff2'),
       url(../fonts/lora400.woff) format('woff');
}
```

Если подключать шрифт через Google Fonts (а это не самый быстрый способ, лучше использовать именно `@font-face` у себя на сайте), Google предоставляет нам `link`. На сегодня в ссылке, которая содержится в атрибуте `href` у тега `<link>`, уже содержится параметр `&display=swap`. При необходимости его можно заменить любым другим нужным значением. Пример получаемой ссылки:

`<https://fonts.googleapis.com/css2?family=Lora:wght@400;700&display=swap>`

FOFT

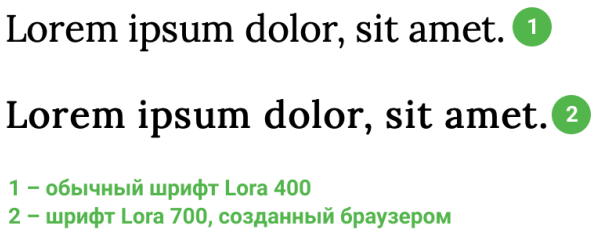
Мы обсудили варианты загрузки и отображения шрифтов, но остался один важный нюанс: шрифт может быть отображён не с той жирностью или не с тем стилем (`font-style`), который мы ожидаем. Как это может произойти?

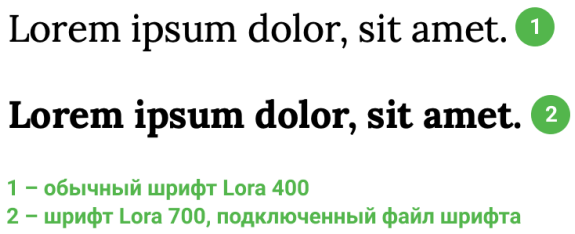
Чаще всего «неаккуратную» жирность или «неаккуратный» наклон шрифта можно заметить, когда подключается шрифт с Google Fonts без указания вариантов жирности и стиля начертания. То есть ссылка для подключения выглядит так:

`<https://fonts.googleapis.com/css2?family=Lora>`

Можно вставить эту ссылку в адресную строку и перейти по ней. Будет видно, что подключается шрифт жирностью 400 и больше ни одна жирность и ни один стиль не подключаются. Если оставить именно такую ссылку для подключения и попробовать установить жирность в 700 у какого-либо элемента с этим шрифтом, то браузер сам попытается сделать правильную жирность, и это будет выглядеть не очень хорошо.

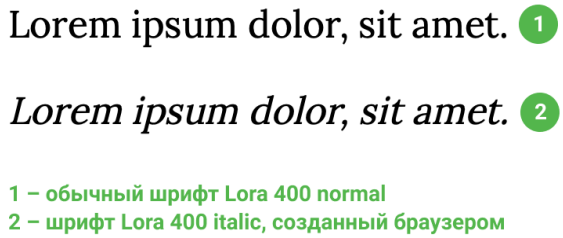
Посмотрим пример:

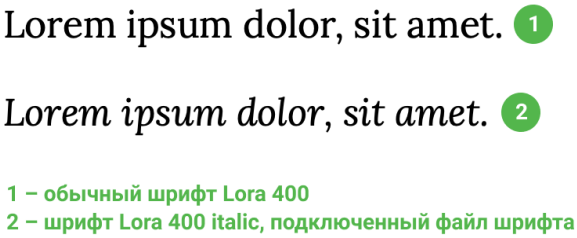




Шрифт Lora 700 при разных подключениях

Ситуация с `italic` не лучше. Браузер также будет пытаться сделать из стиля `normal` стиль `italic`. Пример:





Шрифт Lora 400 italic при разных подключениях

По сути, когда браузер пытается сделать `italic`, он берёт обычный шрифт и наклоняет его. Конечно же, это не то, как должен выглядеть шрифт в действительности.

Если пользоваться подключением через `@font-face` и подключать только шрифт 400 со стилем `normal`, то будут те же проблемы: браузер сам будет пытаться сделать шрифт «жирнее» или `italic`, поэтому не стоит забывать подключать шрифты в нужных начертаниях, если они действительно понадобятся на сайте.

Но важно понимать, что каждое новое подключение — это новый файл. Разберёмся:

1. Если нужен шрифт `font-weight: 400; font-style: normal;`, то это один файл шрифта.
2. Если нужен шрифт `font-weight: 400; font-style: italic;`, то это ещё один файл шрифта.
3. Если нужен шрифт `font-weight: 700; font-style: normal;`, то это ещё один файл шрифта.
4. Если нужен шрифт `font-weight: 700; font-style: italic;`, то это ещё один файл шрифта.

Итого только для подключения жирности 400 и 700, а также их стилей `normal` и `italic`, получается уже четыре файла. Поэтому перед тем, как подключать лю

шрифт, нужно подумать, какие варианты точно понадобятся, и можно ли от каких-то вариантов отказаться, чтобы сайт открывался быстрее.

Не стоит подключать шрифт «на всякий случай», по крайней мере будет достаточно 400 `italic` если основной шрифт сайта жирностью 400 (если основная жирность 300 — значит 300 `italic`). Но это только в том случае, если сайт достаточно крупный, и его будут администрировать, добавляя различные статьи или меняя контент. В случае одностраничника или небольших сайтов подключение шрифта даже основной жирности со стилем `italic` «на всякий случай» — не лучшее решение.

Прочитали главу?

Нажмите кнопку «Готово», чтобы сохранить прогресс.

Готово

Если вы обнаружили ошибку или неработающую ссылку, выделите ее и нажмите Ctrl + Enter

Поиск по материалам

Git

Все материалы

В самом начале

- [ ] Пройдите опрос
- [ ] Укажите персональные данные
- [ ] Изучите регламент
- [ ] Прочитайте FAQ
- [ ] Добавьте свой Гитхаб
- [ ] Выберите наставника
- [ ] Создайте проект

Мой наставник

Выбрать наставника

Работа с наставником

У вас осталось **10** из 10 консультаций.

История




Практикум

Тренажёры

Подписка

Для команд и компаний

Учебник по PHP

Профессии

Фронтенд-разработчик

JavaScript-разработчик

Фулстек-разработчик

Курсы

HTML и CSS. Профессиональная вёрстка сайтов

HTML и CSS. Адаптивная вёрстка и автоматизация

JavaScript. Профессиональная разработка веб-интерфейсов

JavaScript. Архитектура клиентских приложений

React. Разработка сложных клиентских приложений

Node.js. Профессиональная разработка REST API

Node.js и Nest.js. Микросервисная архитектура

TypeScript. Теория типов

Алгоритмы и структуры данных

Паттерны проектирования

Webpack

Vue.js 3. Разработка клиентских приложений

Git и GitHub

Анимация для фронтендеров

Блог

С чего начать

Шпаргалки для разработчиков

Отчеты о курсах

Информация

Об Академии

О центре карьеры

Услуги

Работа наставником

Для учителей

Стать автором

Остальное

Написать нам

Мероприятия

Форум

Соглашение

Конфиденциальность

Сведения об образовательной организации

Лицензия № 4696

Sk
Участник