**Список теоретических вопросов на экзамене:**

1. Рынок мобильных устройств. Современное состояние, история, перспективы.

Мобильные устройства включают в себя смартфоны, планшеты, носимые устройства, бортовые компьютеры, …

Рынок мобильных телефонов – это рынок, на котором купли-продажи выступают сотовые телефоны, смартфон и аксессуары к ним. На сегодняшний день на мировом рынке мобильных устройств представлен большой выбор моделей, но лидирует несколько крупных компаний. Основные из них: Samsung, Apple, Xiaomi.

В 1947 году лаборатория Bell Laboratories (США) официально выступила с предложением создать мобильный телефон. Первый прототип мобильного сотового телефона был создан американской компанией Motorola. Это произошло в 1973 году. Создателем устройства стал инженер Мартин Купер. У аппарата отсутствовал дисплей. Батарея телефона позволяла ему работать в режиме ожидания до 8 часов, а в режиме разговора – до одного часа.

В 1987 году Nokia представила Mobira Cityman 900 – первое устройство для сетей NMT (Nordic Mobile Telephony).

Сотовый телефон с момента своего создания постоянно совершенствовался. В 1993 году в мире был выпущен первый сотовый со встроенными часами. Спустя 3 года немецкая компания Siemens начала производить аппараты с диктофоном и цветным дисплеем. В 2000-м году в продаже появились аппараты со встроенной фотокамерой. Примерно в это же время вышли в продажу телефоны со встроенным mp3-плеером. В 1996 году Motorola подтвердила звание инноватора, представив первый телефон–раскладушку – StarTAC. В 2001 году в сотовых телефонах появляется поддержка платформы Java. Это позволило устанавливать на аппараты множество различных приложений. В их числе известнейший сервис обмена мгновенными сообщениями – ICQ. Первый мобильный с поддержкой технологии Bluetooth в 2002 году был выпущен компанией Ericsson. Первая версия Apple iPhone изначально вышла в США в 2007 году. В 2008 году в свет вышел iPhone 3G, а в 2009 году – iPhone 3GS. Как видно, мобильные телефоны постоянно меняются. Технические характеристики, однако, уже не так важны, как раньше и сегодня производители много внимания уделяют сенсорным интерфейсам, интеграции с социальными сервисами, магазинам приложений.

Одной из главных перспектив в развитии рынка мобильных телефонов является конвергенция, когда телефонный аппарат одновременно выполняет функции различных специализированных устройств. При этом уже сегодня конвергентные устройства для ряда пользователей оказываются удобнее в использовании, чем традиционные МР3-плееры или видеокамеры. По прогнозам, указанная тенденция сохранится и в будущем, при этом в телефонах будет появляться все больше принципиально новых возможностей, тесно связанных с другими функциями устройства (например, мобильным интернетом). При этом специализированные решения вряд ли когда-нибудь будут вытеснены с рынка. Помимо того, что качество и удобство работы в специализированных устройствах все-таки выше, чем в конвергентных сотовых телефонах, если вы когда-нибудь отдадите свой телефон в сервисный центр, выполняющий [ремонт nokia](http://www.ymservice.ru/), то вам в любом случае пригодится обычный цифровой фотоаппарат или навигатор. В ближайшем будущем продолжится бурное развитие всевозможных сервисов, которые предоставляют сами компании-производители или их партнеры. В результате аппаратные решения, не привязанные к каким-либо сервисам, рано или поздно перестанут пользоваться спросом и уйдут с рынка. Что же касается сенсорных устройств, то их развитие, наоборот, будет продолжаться все теми же быстрыми темпами.

1. Мобильные операционные системы. История, семейства, особенности.

Подобно тому, как операционная система Linux или Windows управляет настольным компьютером или ноутбуком, мобильная операционная система - это программная платформа, поверх которой другие программы могут работать на мобильных устройствах. Операционная система отвечает за определение функций, доступных на вашем устройстве, таких как клавиатура, WAP, синхронизация с приложениями, электронная почта, обмен текстовыми сообщениями и многое другое. Мобильная ОС также определяет, какие сторонние приложения (мобильные приложения) можно использовать на вашем устройстве.

Первой полноценной мобильной операционной системой можно считать EPOC16, разработанную в 1988 году английским профессором **Дэвидом Поттером**. В 1994 году Psion занялась разработкой следующей, уже 32-разрядной EPOC, которая в 1998 году получила название Symbian OS. Система использовалась в смартфонах и коммуникаторах Nokia, Samsung, Sony Ericsson, Motorola и еще трех десятках производителей мобильных устройств.

2006-2007: Начиная с 2006 года и, существеннее всего увеличили свою долю на рынке две платформы: **BlackBerry OS** и **Windows Mobile**. Лидерство на рынке продолжала удерживать операционная система Symbian OS, которая, однако, постепенно начала сдавать позиции.

2007-2010: В самом начале 2007 года был анонсирован выпуск [первой модели iPhone](https://yablyk.com/iphone/iphone-2g-original/), которая появилась в продаже спустя полгода — в июне. Тогда же состоялся и дебют платформы **iPhone OS**, которая быстрыми темпами начала набирать популярность. Еще одним важным событием стало появление **Android OS** — будущего прямого конкурента **iOS** в борьбе за первую позицию в отрасли. Параллельно с этим увеличивались продажи устройств на **BlackBerry OS**. **Windows Phone** оставался примерно на тех же позициях, а **Symbian OS** продолжала терять свою популярность. BlackBerry OS — операционная система с основным набором приложений для смартфонов и коммуникаторов, выпускаемых компанией Research In Motion Limited (RIM). Windows Mobile — мобильная операционная система, разработанная Microsoft для собственных аппаратных платформ Pocket PC и Smartphone.

2010-2012: Именно на этот период приходится стремительное увеличение рыночной доли **iOS** и особенно **Android OS**. IOS - мобильная операционная система, разработанная компанией Apple на основе Mac OS X первоначально для iPhone, а затем расширена для поддержки таких мобильных устройств, как Apple iPod Touch, iPad и Apple TV. Apple не лицензирует iOS для установки на стороннее оборудование. А система Android разработана фирмой Android, Inc., приобретенной компанией Google. Важной особенностью Android OS является то, что сервисные программы и библиотеки этой системы написаны на Java. **Symbian OS**, что называется, [окончательно «загнулась»](https://yablyk.com/symbian-oficialno-umerla/), платформа **Windows Phone** также потеряла значительное количество своих покупателей, да и популярность **BlackBerry OS** стала снижаться угрожающими темпами.

Объемы рынка продолжили расти и уже к началу 2010 года общее количество проданных смартфонов перевалило за отметку в 50 миллионов штук.

2012-настоящее время: Практически полное исчезновение устройств на **Symbian OS** в значительной мере повлияло именно на Nokia, которая владеет компанией-производителем этой платформы, начиная с 2008 года. Фиаско **Symbian OS** заставило финнов искать для использования в своих новых устройствах другую операционную систему. Ей стала **Windows Phone**, что может означать объединение сил двух крупных брендов — Microsoft и Nokia — в борьбе на мировом рынке. Однако, стоит сказать и о том, что смартфоны Nokia на **Windows Phone** смогут отвоевать ровно столько, сколько им позволят сделать лидеры отрасли — **Android OS** и **iOS**, которые пока не допускают посторонних в свой спор. Причем у каждой из этих платформ есть свои козыри. **iOS** устанавливается только на устройства, производимые компанией Apple, которая очень внимательно следит за качеством аппаратного обеспечения. На стороне **Android OS** — широчайший выбор устройств самых разных ценовых категорий, что стало возможным благодаря открытому исходному коду этой платформы.

1. Особенности разработки приложений для мобильных устройств. Специфика работы мобильных операционных систем.

Любое мобильное приложение – это прикладное программное обеспечение, разработанное для смартфонов или планшетов, которые функционируют с использованием разных операционных систем. Как правило, для работы мобильного приложения, которое связывается с удаленными ресурсами, требуется доступ к глобальной сети.

[Разработка мобильных приложений](https://polygant.net/ru/mobiledev/razrabotka-mobilnykh-prilozhenij/) представляет собой сложную задачу, для решения которой нужны специальные навыки, время и достаточный бюджет. При этом надо учитывать острую конкуренцию в данной области деятельности: каждый день разработчики создают сотни приложений, но только единицы из них становятся популярными и приносят высокую прибыль.

Особенности мобильной разработки:

1. Особая схема дистрибуции приложений
2. Поддержка разнообразного оборудования
3. Высокая вариативность условий использования
4. Разработка под нецелевой операционной системой
5. Малый размер экрана
6. Особенности пользовательского интерфейса
7. Принцип разделения кода и ресурсов
8. Представление приложения как связанного набора компонентов
9. Автоматизация сборки приложения
10. Открытый исходный текст платформы

Многочисленные исследования показали, что мобильное приложение необходимо для компаний, которые работают в сфере услуг, в том числе и интернет-магазинам. Ведь в приложении можно организовать не только информативную часть о товаре или услуге, но также и функционал для их заказа и оплаты.

Разработка приложения — это время и деньги. За каждую дополнительную функцию будет взиматься дополнительная оплата и, соответственно, будут увеличиваться сроки разработки. Поэтому трижды нужно подумать о необходимости той или иной функции в своем мобильном приложении.

Выбор платформы пополняет особенности разработки мобильных приложений. Тут на самом деле нужно провести анализ, чтобы понять, телефонами какой платформы пользуются ваши клиенты? Чтобы под ту платформу и создать свое первое приложение.

Следует помнить, что ОС для мобильных устройств имеют свою специфику, связанную с различиями в требованиях к мобильному устройству в отличие от настольного или портативного компьютера, а также зависящую от их реализации. К числу таких особенностей можно отнести следующие:

1. Ограничения по памяти и скорости процессора.

2. Различные дизайнерские и конструкционные отличия в экранах и экранных навигаторах разных моделей мобильных устройств.

3. Совместимость с основными форматами файлов.

4. Мультимедийные возможности.

5. Поддержка коммуникационных и сетевых технологий.

1. Особенности дистрибуции мобильных приложений.

Дистрибуция мобильных приложений – это предоставление возможности пользователям скачивать и обновлять приложения пользователей.

У дистрибуции мобильных Android-приложений есть особенность – это быстрая публикация. Релиз приложений в Google Play происходит в пределах трёх суток, в редких случаях — длится неделю. Стор не предъявляет особых функциональных требований к продукту, поэтому в нём может быть опубликовано любое приложение, которое не противоречит [правилам площадки](https://play.google.com/intl/ru/about/developer-content-policy/).

Еще особенность это – большая аудитория. Продажи Android-смартфонов [превышают](https://www.kantarworldpanel.com/global/smartphone-os-market-share/) продажи других ОС: в Германии, Франции, Италии доля Android — 80%, в Мексике, Бразилии и Аргентине — 90–95%, в США, Великобритании, Японии около 60%. В России в 2019 году 73% людей [пользовались](https://www.bloomberg.com/graphics/2019-android-global-smartphone-growth/) смартфонами на базе Android.

Сейчас у платформы [2,5 миллиарда](https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote) активных пользователей — это значит, что публикация приложения в Google Play даёт владельцам бизнеса выход на треть населения Земли.

Особенность дистрибуции заключается еще в низкой стоимости размещения в сторе. Выкладывать приложения в сторы можно только с аккаунта разработчика. В Google Play этот аккаунт оплачивается единоразово и стоит всего $25. Для сравнения владельцы аккаунта iOS-разработчика [отдают $99](https://livetyping.com/ru/blog/kak-vylozhit-prilozhenie-v-app-store) каждый год. Google Play не единственный стор, в котором можно разместить Android-приложение — ещё это можно сделать в Samsung Store и Huawei AppGallery. Аккаунты разработчика в них бесплатны, но приложения будут доступны только владельцам девайсов Samsung и Huawei.

Также особенность заключается в скачивании. За 2020 год количество скачиваний приложений в Google Play [увеличилось](https://go.sensortower.com/q2-2020-data-digest.html?utm_source=blog) на 35% и достигло 28 миллиардов — почти в три раза больше, чем у App Store. По [прогнозам](https://www.appannie.com/en/insights/market-data/app-annie-2017-2022-forecast/?roistat_visit=12727990) сервиса App Annie, цифра продолжит расти. Это значит, что владельцы мобильных приложений получат ещё больше пространства для продвижения бизнеса через стор.

Если говорить о дистрибуции приложений для iOS необходимо помнить, что на данный момент все сервисы, кроме TestFlight от Apple, по-прежнему должны учитывать ограничение на количество устройств доступных при использовании AdHoc provision profiles. Следовательно, максимум используемых устройств равен 100, в которые входят в том числе и устройства, используемые непосредственно разработчиками. Самым же большим ограничением является то, что список зарегистрированных устройств можно “обнулять” только раз в год, после продления подписки на iOS Developer Program.

1. Набор инструментов, необходимых для разработки приложения.

Для полноценной разработки приложений понадобится:

* Рабочий компьютер
* Java SDK
* Android SDK
* IDE – Android Studio
* Виртуальное устройство Android
* Мобильное устройство для тестирования

Для работы с Android SDK требуется загрузка Java SDK, независимо от того, планируете разрабатывать на [Java](https://blog.skillfactory.ru/kak-stat-java-programmistom-s-nulya/) или [Kotlin](https://blog.skillfactory.ru/glossary/kotlin/" \t "_blank). Java SDK отвечает за графическое отображение исходного кода. Чтобы увидеть список версий [Java](https://blog.skillfactory.ru/glossary/javascript/), нужно перейти на официальный сайт [Oracle.](http://www.oracle.com/java/technologies/downloads) Для установки необходимо:

1. Выбрать версию. Последние версии Java доступны только для 64-битных систем. Их можно загрузить либо в компрессированном архиве, либо в инсталляционном файле.

2. Зарегистрироваться в системе Oracle. Если у пользователя есть учетная запись — пройти процедуру авторизации.

3. Скачать Java SDK для Android на ПК.

4. Выполнить все требования установочного файла, указав место хранения утилиты.

5. После завершения установки закрыть программу и перезагрузить компьютер.

6. Теперь можно приступать к настройке и работе с Android SDK.

Android SDK — это дополнительный набор инструментов Android Studio, которые помогают написать код, запустить тестирование и отладку, проверить работу приложения на различных версиях операционной системы и оценить результат в реальном времени. Также пакет позволяет пользователям получать информацию о состоянии операционной системы, читать логи и выявлять ошибки. Через SDK для Андроид можно восстанавливать программную оболочку и устанавливать сторонние прошивки.

Android Studio — это IDE (интегрированная среда разработки) для работы с платформой Android. Она позволяет разрабатывать приложения для смартфонов, планшетов, часов, телевизоров и других устройств на этой ОС.

AVD расшифровывается как Android Virtual Device, а именно виртуальное Андроид устройство. На компьютере нужно открыть заранее выбранную и сконфигурированную систему. Это удобно даже когда есть реальный телефон. В эмуляторе можно изменить версию ОС, размер экрана и другие параметры. Это позволяет протестировать то, как работает ваше приложение на разных версиях Android. В Android Studio всеми виртуальными устройствами управляет [AVD manager](https://developer.android.com/studio/run/managing-avds). Его можно открыть через меню Tools в Android studio. В AVD Manager нужно нажать на кнопку создания нового виртуального устройства и выбрать предустановленный шаблон или же создать свой новый. Когда все будет готово, останется лишь запустить созданный эмулятор Android устройства.

1. Виды и типы мобильных приложений. Нативные, веб и гибридные приложения.

**Мобильное приложение — это** программное обеспечение, которое устанавливается на мобильных устройствах, таких как смартфоны или планшеты.

**Существует несколько видов мобильных приложений:**

1. Приложения переднего плана

Выполняют свои функции только когда видимы на экране. В свернутом виде выполнение приложений приостанавливается, например: игры, текстовые редакторы, видеопроигрыватели. Знание жизненного цикла активности для гладкости переключения между фоновым и активным режимами. Разработка удобного и интуитивно понятного интерфейса.

1. Фоновые приложения

После настройки не предполагают взаимодействия с пользователем. Большую часть времени находятся и работают в скрытом состоянии, например: службы экранирования звонков, SMS-автоответчики. Нужно помнить, что совершенно невидимые сервисы будут неуправляемыми. Такие приложения позволяют пользователю только: санкционирование запуска сервиса, настройка сервиса, приостановка работы сервиса, прерывание работы сервиса.

1. Смешанные приложения

Большую часть времени работают в фоновом режиме. Допускают взаимодействие с пользователем даже после настройки, например: мультимедиа-проигрыватели, чаты, почтовые клиенты. Такие приложения, не теряя работоспособности в фоновом режиме, должны реагировать на пользовательский ввод. При взаимодействии с пользователем должны учитывать свое текущее состояние.

1. Виджеты

Небольшие приложения, отображаемые в виде графического объекта на рабочем столе, например: приложения для отображения динамической информации. Планируя разработку приложения, необходимо определить способ его использования, только после этого приступать к проектированию и непосредственно разработке.

**Также существуют несколько типов мобильных приложений:**

1. Нативные приложения

Это самый распространенный тип приложений. Они создаются на основе языков программирования для таких операционных систем, как Android, Windows Phone и iOS. Благодаря таким приложениям можно в полной мере использовать функционал GPS, видеокамер или датчиков ускорения. Главным преимуществом нативных приложений является возможность автономной работы без необходимости подключения к интернету. Чаще всего приложения такого типа распространяются через магазины приложений Play Market, AppStore и другие. Также нативными приложениями могут являться различные электронные библиотеки.​

У нативных приложений есть плюсы: они более быстрые, созданы специально под конкретную платформу, распространяются через магазины приложений, интуитивно понятный интерфейс, используют все возможные устройства. Но есть и минусы: они одноплатформенные, у них специфические языки разработки, они весьма дорогие и они сложнее в поддержке.

1. Веб-приложения

Они создаются с помощью HTML/CSS/JavaScript. Выполняются внутри браузера. Построены точно, как обычные веб-приложения/веб-сайты. Сверстаны таким образом, чтобы хорошо смотреться на экране мобильного устройства.

У веб-приложений есть как плюсы, так и минусы. Плюсы: их просто создавать, просто поддерживать, у них широкий спектр технологий, они дешевле в разработке, у них кроссплатформенность. Минусы: им нужен браузер для запуска, они медленнее нативных приложений, они менее интерактивные, у них нет ярлыка на рабочем столе, они не могут распространяться через магазины, не могут взаимодействовать с аппаратной частью устройства.

1. Гибридные приложения

[Программирование под Android](https://woxapp.com/ru/our-services/android-apps-development/) просто не может работать без гибридных приложений. Они – это соединенные воедино нативные и веб-приложения. Главными преимуществами гибридных приложений являются кроссплатформенность на web-технологиях и возможность доступа к функциям смартфона или другого мобильного устройства. Хотя стоит отметить, что [разработка приложений для iOS](https://woxapp.com/ru/our-services/ios-apps-development/) гибридного типа стоит значительно дороже, чем разработка приложений иного типа. Многие популярные социальные сети имеют свои гибридные приложения, которые можно свободно загрузить из онлайн-магазина.

У гибридных приложений есть свои минусы: они медленнее нативных, но более дорогие, чем веб-приложения. Но как и у других приложений, у гибридных приложений есть немало плюсов: их проще создавать, они гораздо дешевле в разработке нативных приложений, одно такое приложение подходит под все платформы, для них не требуется браузер для работы, а в зависимости от фреймворка они могут иметь доступ к аппаратным функциям приложения, у таких приложений быстрый цикл разработки.

1. Основные языки программирования, применяющиеся для разработки мобильных приложений.

С момента появления Java стал основным языком для разработки мобильных приложений на Android. Он обеспечивает кроссплатформенную поддержку. Java признан языком официальной среды Android Studio, что дает доступ к огромному числу инструментов. Также для Java разработано много библиотек и руководств, документации Google написана со ссылками на этот язык. Кроме того, приложения на Java легко перенести на различные операционные системы. И хотя Java — относительно старый язык программирования, он сохраняет популярность. Несмотря на возрастающую популярность Kotlin, Java остается самым широко используемым языком программирования для продуктов на Android. Kotlin — это новейший язык программирования, полностью совместимый с Java. Эти два языка взаимозаменяемы. В прошлом году Google назвал Kotlin «основным языком для разработки приложений на Android». Kotlin практически лишен недостатков, поэтому быстро набирает популярность в среде Android-разработчиков. Среди преимуществ по сравнению с Java стоит отметить масштабируемость Kotlin. Вообще, когда начинаются споры о том, что лучше, Java или Kotlin, следует помнить, что Kotlin – это обёртка над Java. Чтобы понять документацию Kotlin и получить помощь в процессе разработки мобильных Android приложений, следует знать именно Java. Если на Java мы пишем:

final TextView helloTextView = (TextView) findViewById(R.id.text\_view\_id);

helloTextView.setText("Hello world");

То на Kotlin: helloTextView.text = "Hello world"

HTML5 также находится в этом списке, хотя и не является языком для разработки мобильных приложений — он используется для создания веб-приложений, которые запускаются на любом устройстве через браузер. Приложения, написанные на HTML5, можно объединять с фреймворками, которые предоставляют возможность использовать API-интерфейсы и, сохранять при этом все функции веб-приложения. А вот Android-разработка на JS более жизнеспособна. С фреймворком React Native вы можете создавать мобильные приложения с многофункциональным мобильным UI. При этом такие программы полностью нативны, то есть не являются мобильными веб-приложениями. Всё потому, что React Native использует те же компоненты, что и простые приложения для Android. Несмотря на весь спектр языков, обучение Android-программированию должно начинаться с Java и Kotlin, как двух официальных языков.

1. Необходимое программное обеспечение для разработки нативных мобильных приложений.

Программное обеспечение для разработки приложений — это неотъемлемая часть цифровой трансформации любого рабочего места. Программное обеспечение для разработки приложений обеспечивает совместимость языков программирования в различных экосистемах ОС и стеках приложений, ускоряет работу по разработке программного обеспечения, предоставляя единый графический пользовательский интерфейс для различных задач разработки: редактирования кода, компиляции кода, развертывания приложений и т.д. Также программы отслеживают ошибки, позволяя разработчикам управлять приложениями и устранять проблемы с ними.

Android Studio – это среда разработки, программное обеспечение, в котором есть инструменты и сервисы для создания новых Android нативных-приложений. Многие из существующих операционных систем имеют такую среду разработки.

AndroidStudio – это интегрированная среда разработки для работы с платформой Android. ПО было анонсировано 16 мая 2013 года компанией Google. Софт помогает разрабатывать разнообразные приложения и игры под Android.

Android Studio содержит программное обеспечение, которое может эмулировать смартфон для запуска программ, отладки, просмотра сайтов и много другого.

Это возможно благодаря AVD (Android Virtual Device) Manager. Вы можете настроить даже несколько эмуляторов, подобрав под каждый желаемый размер экрана, версию системы. Это очень полезный функционал, потому как избавляет девелоперов от покупки нескольких смартфонов для тестов.

В программу встроен эмулятор, позволяющий проверить корректную работу приложения на устройствах с разными экранами, с различными соотношениями сторон. Отличительная особенность эмулятора – просмотр приблизительных показателей производительности при запуске приложения на самых популярных устройствах.

ПО для разработки утилит и нативных приложений на Android действительно создает приятное первое впечатление. Android Studio имеет богатый набор инструментов, гибкость в разработке, возможности тестирования, поддержка нескольких языков программирования и встроенный эмулятор делают утилиту одной из лучших в своей нише.

1. Основные характеристики языка программирования Java.

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией. Разработка Java началась в 1990 году, первая официальная версия — Java 1.0, — была выпущена только 21 января 1996 года. Последняя версия - Java SE 11, 25 сентября 2018 года.

Редакции Java:

* Java SE — Java Standard Edition.
* Java EE — Java Enterprise Edition.
* Java ME — Java Micro Edition;
* Java Card.

Язык Java активно используется для создания мобильных приложений под операционную систему Android. При этом программы компилируются в нестандартный байт-код, для использования их виртуальной машиной Dalvik (начиная с Android 5.0 Lollipop  -  ART). Для такой компиляции используется Android SDK (Software Development Kit), разработанный компанией Google.

У языка есть основные характеристики:

* Объектно-ориентированный язык общего назначения
* С-подобный синтаксис
* Разработан компанией Sun (Oracle)
* Существуют бесплатные инструменты разработки
* Компиляция в байт-код, использование нативного кода
* Виртуальные машины доступны для всех распространенных платформ
* Развитая модельная и объектно-ориентированная система
* Автоматическая сборка мусора
* Явная статическая сильная типизация

Java обеспечивает несколько уровней зашиты от вредоносного ПО. Большое внимание в языке Java уделено раннему обнаружению возможных ошибок, динамической проверке (во время выполнения программы), а также исключению ситуаций, которые могут привести к ошибкам. В то время, когда язык Java разрабатывался, в среде программирования существовала проблема - если программа была написана под одну ОС, ее было тяжело или же просто невозможно запустить под другой ОС. Разработчики языка попытались решить эту проблему, и в принципе у них это получилось - программа, написанная на Java выполняется одинаково (ну, почти одинаково) на всех OC и компьютерах с разной архитектурой. Язык Jаvа предназначен для распределенной среды Интернета, поскольку он поддерживает семейство сетевых протоколов ТСР /IP. В Java поддерживается также удаленный вызов методов (RМI - Remote Method Invocation). Такая возможность позволяет вызывать методы из программ через сеть. Во многих отношениях язык Java является более динамичным, чем языки С или С++. Он был разработан так, чтобы легко адаптироваться к постоянно изменяющейся среде. В библиотеки можно свободно добавлять новые методы и объекты, не причиняя никакого вреда. Язык Java позволяет легко получать информацию о ходе выполнения программы. Все языки делятся на интерпретируемые и компилируемые. Java является и тем и другим, но с точки зрения процессора он интерпретируемый.

1. Система типов в Java.

Основа любого языка программирования — данные и опе­рации с ними. Java не исключение. Это строго типизированный язык, поэтому типы данных значат в нём очень многое. Java следит за тем, чтобы все переменные, выражения и значения соответствовали своему типу данных. Поэтому операции, которые допустимы для одного типа, нельзя провести с другим — тип переменной определяет, какие операции с ней можно выполнить.

В Java типы данных делят на две большие группы: примитивные и ссылочные. В состав примитивных типов (или просто примитивов) входят четыре подвида и восемь типов данных:

1) целые числа (byte, short, int, long);

2) числа с плавающей точкой (float, double);

3) логический (boolean);

4) символьный (char).

Ссылочные типы данных ещё называют ссылками. К ним относятся все классы, интерфейсы, массивы, а также тип данных String.

Как выглядят примитивные типы данных:

* byte, 8 бит: -128..127
* short, 16 бит: -32,768..32,767
* int, 32 бита: -2,147,483,648..2,147,483,647
* long, 64 бита: -9,223,372,036,854,775,808..9,223,372,036,854,775,807
* float, 32 бита
* double, 64 бита
* boolean: true и false
* char: 16 бит, Unicode

Особенности примитивных целочисленных типов:

• Целочисленные типы могут задаваться литералами в трех формах: десятичной (например, 2), восьмеричной (например, 02) и шестнадцатеричной (например, 0x2);

• Литералы имеют по умолчанию примитивный тип int;

(например, 1, 10, 100, 10000, 1000000);

• Литералы имеют примитивный тип long, если последним символом является l или L;

(например, 1l, 10L, 100l, 10000L, 1000000l);

Пример;

byte x = 15;

short y = 046;

int z = 0x153;

long k = 1000000000000L;

Типы с плавающей точкой – float и double. Особенности примитивных типов c плавающей точкой:

• Типы с плавающей точкой могут задаваться литералами, которые либо содержат десятичную запятую, либо записаны в т.н. экспоненциальной форме (через символ E или e), либо заканчиваются на символы (F или f – тип float) или символы (D или d) – тип double;

Пример;

double pi = 3.14;

double i = 6.02E23;

float j = 2.718f;

double k = 123.4E+306D;

Логический тип – boolean. Главная особенность примитивного типа boolean:

• существует только два возможных значения – true и false;

Пример: boolean truth = true;

Данная инструкция объявляет переменную c именем truth примитивного типа boolean и присваивает ей значение true.

Символьный тип – char. Особенности примитивного типа char:

• Представляет собой 16-битное значение в кодировке Unicode;

• Литералы данного примитивного типа должны содержаться в одинарных кавычках (’ ’).

Пример:

char sym1 = ’a’;

char sym2 = ’\u03A6’; // ’\u03A6’ is the Greek letter phi [Φ]

char sym3 = ’\t’;

char sym4 = ’\n’;

Строковый тип – String. Является не примитивным типом, а классом. Особенности строкового типа String:

• Литералы строкового типа должны содержаться в двойных кавычках (“”).

Пример:

String str0 = “”;

String str1 = “Good morning!”;

String str2 = “Time to get up!!!”;

String str3 = str1 + str2 + “\n”;

1. Основные операторы Java. Условные операторы, операторы ветвления, ввод-вывод.

[Операторы](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html) на языке программирования Java — это специальные символы, которые выполняют определенные операции над одним, двумя или тремя операндами, а затем возвращают результат. Java предоставляет богатый набор операторов для управления переменными. Мы можем разделить все операторы Java на следующие группы:

* Арифметические Операторы

Арифметические операторы — используются в математических выражениях таким же образом, как они используются в алгебре.

* Операторы отношений

Чтобы проверить равенство, следует использовать символы = =. Например, значение выражения 3 = = 7 равно false.

* Битовые операторы

Java определяет несколько побитовых операторов, которые можно применять к целочисленным типам, long, int, short, char и byte. Побитовый оператор работает с битами и выполняет побитовую операцию.

* Логические Операторы

|| (*условное-ИЛИ (OR)* ) и **&&** (*условное-И (AND)*) совершают операции над булевыми выражениями.

* Операторы присваивания

Используются в **Java**, чтобы присвоить значения переменным. Например,

int age;//возраст

age = 5;

Оператор «**=**» присваивает значение справа от себя переменной, находящейся слева. В примере переменной **age** присвоено значение **5**.

* Разные Операторы (например: условный оператор, операторы ветвления, ввод-вывод)

Условный оператор также известен как **троичный оператор**. Этот оператор состоит из трех операндов и используется для вычисления логических выражений. Цель оператора – решить, какое значение следует присвоить переменной. Оператор записывается как – variable x = (expression) ? value if true : value if false.

Условный оператор if. С помощью оператора if мы можем указать, какой фрагмент кода необходимо выполнить. Если утверждение истинно, выполняется определенная часть кода. Если оно ложно, тогда утверждение внутри оператора if игнорируется.

Условный оператор if-else. Конструкция if-else используется для выбора между двумя способами действий в зависимости от того, является ли заданное условие истинным или ложным.

Условный оператор switch-case. К структуре switch обращаются, когда нужно выбрать определенный блок кода из множества других. Этот оператор эффективно применять при разветвлении кода в нескольких направлениях на основе одной переменной.

**Ветвление** (условная инструкция) - это конструкция языка программирования, обеспечивающая выполнение определённой команды или набора команд только при условии истинности некоторого логического выражения, либо выполнение одной из нескольких команд (наборов команд) в зависимости от значения некоторого выражения.

Для получения данных, введенных пользователем, а также для вывода сообщений нам необходим ряд классов, через которые мы сможем взаимодействовать с консолью.

Для создания потока вывода в класс System определен объект out. В этом объекте определен метод println, который позволяет вывести на консоль некоторое значение с последующим переводом консоли на следующую строку:

|  |  |
| --- | --- |
| 1 | System.out.println("Hello world"); |

В метод println передается любое значение, как правило, строка, которое надо вывести на консоль. При необходимости можно и не переводить курсор на следующую строку. В этом случае можно использовать метод System.out.print(), который аналогичен println за тем исключением, что не осуществляет перевода на следующую строку.

|  |  |
| --- | --- |
| 1 | System.out.print("Hello world"); |

1. Функции в Java. Передача аргументов.

Часто какой-то код в программе нужно повторять много раз в разных местах. Тогда этот код можно поместить в функцию. Функция — это блок кода, у которого есть имя, параметры и возвращаемое значение. Функции можно вызывать по имени в любых местах, в main или в других функциях. Это гораздо удобнее, чем без конца копировать кучу кода, когда вам нужно выполнить его несколько раз.

Функция представляет собой небольшую программу, выполняющую определённые действия, когда её вызывают по имени. В языке программирования Java есть много встроенных и готовых к работе функций, однако никто не мешает создавать пользователю свои функции.

Функцией называют блок кода, у которого есть имя, параметры и возвращаемое значение. По имени можно вызывать этот блок в любом месте программы. Параметры — это входные данные, на которые опирается наша функция.

Многие функции создаются, чтобы посчитать какой-то результат. Этот результат нужно вернуть, то есть передать из кода функции в место, где её вызвали.

Различают два основных вида функций:

1. Функция, которая что-то возвращает;

Данный тип функций является наиболее используемым, так как почти всегда требуется что-то посчитать, произвести некое преобразование и тому подобное, следовательно полученный результат необходимо использовать вне функции. Всё что было создано в функции в конечном счёте, умирает в ней же, в связи с этим необходимо **вернуть результат**в основной код программы. Сделать это помогает оператор return.

Создадим функцию, которая вернёт значение переменной и присвоит переменной в основном коде это значение:

public class Main {

    public static int func(){  //идентификатор доступа, функция статична, тип возвращаемого значения и имя функции без параметров

        int n = 10;  //создание переменной со значением

        return n;  //возврат значения при вызове данной функции

    }

    public static void main(String[] args) {  //блок основного кода программы }

}

Создание функции очень похоже на основной блок кода программы. Создаётся переменная, а потом она возвращается в нужном разработчику месте. В теле функции можно прописать любой код, создать массивы, циклы и любая другая структура.

1. Функция, которая ничего не возвращает;

Данный тип функции используется при работе с глобальными переменными и, если нужно что-то напечатать, вывести на экран. Нет почти никакого отличия между данным типом функции и предыдущим. Необходимо лишь указать другой тип функции и не использовать return.

public class Main {

    public static void func1(){  //идентификатор доступа, функция статична, тип возвращаемого значения и имя функции без параметров

        System.out.println("Привет из функции!");  //вывод на экран сообщения

    }

    public static void main(String[] args) {  //основной блок программы

    }

}

Также функции можно разделить на две категории:

1. Функция без параметров;

Чтобы работать с функциями и получать от них какой-то результат, необходимо вызвать функцию по имени в нужном месте. Используем предыдущие функции и вызовем их в основном блоке:

public class Main {

    public static int func(){  //идентификатор доступа, функция статична, тип возвращаемого значения и имя функции без параметров

        int n = 10;  //создание переменной со значением

        return n;  //возврат значения при вызове данной функции

    }

    public static void func1(){  //идентификатор доступа, функция статична, тип возвращаемого значения и имя функции без параметров

        System.out.println("Привет из функции!");  //вывод на экран сообщения

    }

    public static void main(String[] args) {  //основной блок программы

        int u = func();  //присвоение переменной значения возвращённого функцией

        System.out.println(u);  //вывод на экран значения переменной

        func1();  //вызов функции по имени

    }

}

В результате получим на экран:

10

Привет из функции!

Функция, которая что-то возвращает, обязательно должна вызываться таким образом, чтобы возвращаемое значение было чему-то присвоено.

1. Функция с параметрами;

Бывает необходимость провести над значениями некие действия. Для этого необходимо передать функции эти самые значения. Когда значения передаются в функцию, то они становятся **аргументами функции**.

Создадим функцию с параметрами и вызовем её в основном блоке используя аргументы.  
Возведём переменную в определённую степень и вернём значение в переменную.

**Указываем параметры при создании функции** (переменные в скобках после имени функции). Аргументы должны быть переданы обязательно, иначе функция не заработает, у неё элементарно не будет значения, с которым она должна взаимодействовать.

**Указываем аргументы при вызове функции**(два целочисленных значения):

public class Main {

    public static int func2(int var\_1, int var\_2){  //функция с параметрами

        int n = 1;  //создаём переменную, в которую будем помещать расчётное значение

        for(int i = 0; i < var\_2; i++){  //цикл для возведения в степень

            n = n \* var\_1;  //умножаем новое значение со старым и тем самым возводим в степень

        }

        return n;  //возвращаем посчитанное значение

    }

    public static void main(String[] args) {  //основной блок программы

        int result = func2(10,2);  //вызываем функцию передав два аргумента (возводим десять во вторую степень)

        System.out.println(result);  //вывод значения переменной

    }

В результате получаем: 100}

В функции вы можете помещать всё что угодно. В данном случае аргументы должны быть переданы обязательно, иначе произойдёт ошибка.

Чтобы ошибки не возникало, можно использовать значения по умолчанию при создании функции, но, к сожалению, Java обделена такой возможностью, **значение по умолчанию в функции задать невозможно**.

1. Реализация ООП в Java. Наследование, инкапсуляция, полиморфизм. Методы и атрибуты класса.

Концепции ООП являются основополагающими элементами и составляют основу языка программирования Java. В рамках данного подхода выделяют следующие термины: **абстракция**, **инкапсуляция**, **наследование** и **полиморфизм**. Понимание данных принципов служит ключом к построению целостной картины того, как работают программы, написанные на Java. По большому счету, объектно-ориентированный подход позволяет нам описывать классы, определять методы и переменные таким образом, чтобы затем использовать их вновь, частично либо полностью, без нарушения безопасности.

Как сказано ранее, выделяют четыре основные принципа ООП:

* **Абстракция**. Абстракция означает использование простых вещей для описания чего-то сложного. Например, мы все знаем как пользоваться телевизором, но в тоже время нам не нужно обладать знаниями о том, как он работает чтобы смотреть его. В Java под абстракцией подразумеваются такие вещи, как **объекты**, **классы** и **переменные**, которые в свою очередь лежат в основе более сложного кода. Использование данного принципа позволяет избежать сложности при разработке ПО.
* **Инкапсуляция**. Под инкапсуляцией подразумевается сокрытие полей внутри объекта с целью защиты данных от внешнего, бесконтрольного изменения со стороны других объектов. Доступ к данным (полям) предоставляется посредством публичных методов (геттеров/сеттеров). Это защитный барьер позволяет хранить информацию в безопасности внутри объекта.
* **Наследование**. Это особая функциональность в объектно-ориентированных языках программирования, которая позволяет описывать новые классы на основе уже существующих. При этом поля и методы класса-предка становятся доступны и классам-наследникам. Данная фича делает классы более чистыми и понятным за счет устранения дублирования программного кода.
* **Полиморфизм**. Данный принцип позволяет программистам использовать одни и те же термины для описания различного поведения, зависящего от контекста. Одной из форм полиморфизма в Java является переопределение метода, когда различные формы поведения определяются объектом, из которого данный метод был вызван. Другой формой полиморфизма является перегрузка метода, когда его поведение определяется набором передаваемых в метод аргументов.

Концепции ООП в Java позволяют программистам создавать компоненты, которые можно переиспользовать в различных частях программы, не подвергая данные опасности.

Основная цель использования абстракции — это уменьшение сложности компонентов программы за счет скрытия от программиста, использующего эти компоненты, ненужных ему подробностей. Это позволяет реализовать более сложную логику поверх предоставленной абстракции, не вдаваясь в подробности ее реализации.

Инкапсуляция позволяет нам пользоваться возможностями класса без создания угрозы безопасности данных за счет ограничения прямого доступа к его полям. Также она позволяет изменять код классов, не создавая проблем их пользователям (другим классам). В Java данный принцип достигается за счет использования ключевого слова **private**.

Наследование — еще одна важная концепция ООП, которая позволяет сэкономить время на написании кода. Возможности наследования раскрываются в том, что новому классу передаются свойства и методы уже описанного ранее класса. Класс, который наследуется, называется дочерним (или подклассом). Класс, от которого наследуется новый класс — называется родительским, предком и т. д. В языке программирования Java используется ключевое слово **extends** для того, чтобы указать на класс-предок.

Пример:

abstract class Turtle {

public abstract void draw();

}

class NinjaTurtle extends Turtle {

public NinjaTurtle(int initX, int initY, String name) {

super(initX, initY); // Call superclass’ constructor

// ... do some more initialization stuff…

}

// Declarations for Ninja turtles

}

Полиморфизм предоставляет возможность единообразно обрабатывать объекты с различной реализацией при условии наличия у них общего интерфейса или класса. По-простому: способность вызывать нужные методы у объектов, имеющие разные типы (но находящиеся в одной иерархии). При этом происходит автоматический выбор нужного метода в зависимости от типа объекта.

Объекты с одинаковой структурой данных *(Атрибуты)* и поведением *(Методы или Операции)* группируются в классы. Каждый *класс* описывает возможно бесконечный набор индивидуальных объектов. Каждый объект, как говорят, является образцом или экземпляром *класса*. Каждый образец класса, имеет свои собственные значения для каждого атрибута, но вместе они используют названия атрибутов и операции с другими образцами класса.

Вызов функции к объекту – выполнение *метода,* который является подобным телу процедуры. Метод может и не возвращать никакого значения. Метод - функция, содержащаяся в пределах класса.  
В Java, функция, объявленная внутри класса, называется *методом.* Для объявления членов класса обычно используется ключевое слово public для того, чтобы все функции и переменные, объявленные после этого ключевого слова, были доступны как для других членов класса, так и для любой другой части программы, в которой находится данный класс.

В Java переменная внутри класса называется атрибутом класса, а атрибуты класса также известны как поля. Доступ к атрибутам класса можно получить с помощью объекта класса.

Изображение выглядит как стол

Автоматически созданное описание Изображение выглядит как стол

Автоматически созданное описание

1. Структура проекта приложения для Android. Основные категории файлов.

При создании нового проекта в виде пустого приложения автоматически генерируются две корневые папки: *app* и *Gradle Scripts*. В первой папке включено три подпапки: *manifests*, в котором содержаться файлы конфигурации приложения, *java*, в которой содержится исходный код приложения, и *res*, в котором хранятся файлы ресурсов, таких как картинки, стили и т.д.

Главным и единственным файлом в папке *manifests*, представленным самой программой Android Studio, является AndroidManifest.xml. В данном файле хранится информация о пакетах и компонентах приложения. Он важен также тем, что именно в нем указывается, какой из кодов в папке Java, является основным, и какие вообще коды отображаются в самом приложении. Ведь именно в файле AndroidManifest.xml написаны разрешения приложения в применении компонентов системы Android.

Исходные папки проекта приложения в папке *java* могут содержаться в различных подпапках. В папке *java* есть своя определенная структура файлов.

В папке *res* представлены несколько подпапок, точнее четыре. Она имеется даже при создании тестового базового приложения, и тем более начинает разрастаться при разработке крупных проектов. Оттого не удивительно, что ее файлы могут занимать много места на жестком диске устройства. Сейчас рассмотрим, что хранится в папке *res* при создании проекта приложения. Это 4 подпапки: 1) папка *drawable*, в которой содержаться различные файлы с изображениями; 2) папка *layout*, в которой располагаются xml-файлы, используемые для разработки интерфейса приложения; 3) папка *mipmap*, в которой хранятся иконки приложения; 4) папка *values*, которая хранит xml-файлы со значением цветов, строк, стилей и тем.

Мы рассмотрели, что имеется и хранится в папке *app*. Если рассмотреть папку *Gradle Scripts*, то про неё можно сказать, что в папке содержаться скрипты автоматизации сборки проекта. Значение этой папки не менее важна для самого приложения, так как это существенно упрощает сборку проекта со значительного количества файлов в apk-файл.

1. Файл манифеста. Назначение, основные разделы

Главным и единственным файлом в папке *manifests*, представленным самой программой Android Studio, является AndroidManifest.xml. В данном файле хранится информация о пакетах и компонентах приложения. Он важен также тем, что именно в нем указывается, какой из кодов в папке Java, является основным, и какие вообще коды отображаются в самом приложении. Ведь именно в файле AndroidManifest.xml написаны разрешения приложения в применении компонентов системы Android. Содержит всю необходимую информацию, используемую системой для запуска и выполнения приложения.

Основная информация в манифесте:

* Имя Java пакета приложения
* Описание компонентов приложения
* Определение процессов
* Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями
* Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного
* Список вспомогательных классов
* Определение минимального уровня Android API для приложения
* Список библиотек связанных с приложением

Манифест приложения содержит корневой элемент **<manifest>** с именем пакета, заданным в атрибуте **package**. Он также должен включать атрибут **xmls:android**, который будет предоставлять несколько системных атрибутов, используемых в файле.

В <manifest> можно добавить атрибут **android:versionCode**, который используется для определения текущей версии приложения в виде целого числа, которое увеличивается с каждым обновлением. Также атрибут **android:versionName** используется для указания публичной версии, которая показывается пользователям.

Также можно указать, куда должно устанавливаться приложение: на SD-карту или внутреннюю память, используя атрибут **android:installLocation**.

1. Элементы пользовательского интерфейса Android SDK: Текстовые, графические, элементы управления формами, кнопки. Способы создания элементов.

Элементы пользовательского интерфейса – это интерактивные компоненты в пользовательском интерфейсе вашего приложения. Android предоставляет широкий спектр элементов управления, которые вы можете использовать в своем пользовательском интерфейсе, такие как кнопки, текстовые поля, панели поиска, флажки, кнопки масштабирования, кнопки переключения и многое другое.

Пользовательский интерфейс представлен layout файлом (рус. Макетом), который создаётся, используя язык разметки XML. Макет определяет структуру пользовательского интерфейса экрана. Все элементы макета построены с использованием иерархии объектов View и ViewGroup. View – компонент, который пользователь может видеть и взаимодействовать с ним. В то время как ViewGroup является невидимым контейнером, который определяет структуру макета.

Объекты View обычно называют «виджетами». Они могут быть представлены одним из классов-наследников, таких как Button, TextView, ImageView. Объекты ViewGroup обычно называют «контейнерами». Они определяют, как именно будут располагаться элементы внутри экрана.

Вы можете работать с макетами двумя способами:

* Объявлять элементы пользовательского интерфейса в XML коде. Android предоставляет простой XML файл, в котором можно добавлять разные View и ViewGroup вручную. Вы также можете использовать редактор макетов Android Studio, чтобы создать свой XML-макет, не заглядывая в XML код.
* Создавать элементы макета во время выполнения программы из Java кода. Приложение может создавать объекты View и ViewGroup (и управлять их свойствами) программно.

Android предоставляет ряд элементов управления пользовательским интерфейсом, которые позволяют создавать графический интерфейс пользователя для вашего приложения:

* TextView - прямоугольный блок с текстовой надписью.

<TextView  
android:id="@+id/myTextview"  
android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"  
android:text="Mangnet Brains"  
android:textSize="25dp"  
android:textColor="@android:color/black"  
android:typeface="serif"  
android:gravity="center"  
android:padding="10dp"  
android:layout\_margin="20dp" />

* EditText - поле для ввода текста.

<EditText  
android:id="@+id/myEdittext"  
android:layout\_width="fill\_parent"  
android:layout\_height="wrap\_content"  
android:textSize="20dp"  
android:textStyle="bold"  
android:typeface="serif"  
android:gravity="center"  
android:padding="10dp"  
android:layout\_margin="20dp"  
android:hint="Enter a Number"  
android:singleLine="true"  
android:inputType="textPassword" />

* ImageView - позволяет отобразить изображения на экране устройства.

<ImageView  
android:id="@+id/myimageview"  
android:layout\_width="100dp"  
android:layout\_height="100dp"  
android:layout\_margin="20dp"  
android:gravity="center"  
android:padding="10dp"  
android:src="@drawable/ic\_launcher" />

* CheckBox - флажок, который пользователь может установить или снять тапом.

<CheckBox  
android:id="@+id/checkBox1"  
android:layout\_width="100dp"  
android:layout\_height="wrap\_content"  
android:layout\_margin="20dp"  
android:text="Formget."  
android:checked="true" />  
<CheckBox  
android:id="@+id/checkBox2"  
android:layout\_width="100dp"  
android:layout\_height="wrap\_content"  
android:layout\_margin="20dp"  
android:text="Mailget." />

* RadioGroup и RadioButton - используются для организации группы кнопок-переключателей.

<RadioGroup  
          android:id="@+id/radioGroupExample"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content" >   
        <RadioButton  
           android:id="@+id/radioBtton1"

  android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
   android:checked="true"  
   android:text="RadioButton 1" />   
       <RadioButton  
           android:id="@+id/radioButton2"  
       android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
       android:text="RadioButton 2" />

</RadioGroup>

* Button - простая кнопка с надписью, реагирующая на нажатие (тап).

<Button  
android:id="@+id/button1"  
android:layout\_width="match\_parent"  
android:layout\_height="wrap\_content"  
android:text="Click Here !" />

<ImageButton  
android:id="@+id/imageButton1"  
android:layout\_width="match\_parent"  
android:layout\_height="wrap\_content"  
android:layout\_gravity="center"  
android:src="@drawable/ic\_launcher" />

1. Принцип разделения представления и поведения. Файлы представлений, ресурсы, программный код.

Архитектура Android-приложений: основные принципы:

* Нужно разделять ответственность

Это один из самых важных принципов, который многие не соблюдают. Нужно разделять ответственность между классами. Например, не нужно разрабатывать весь код приложения в «Activity» или «Fragment», потому что эти классы должны отвечать лишь за логику взаимодействия интерфейса и ОС.

* Нужно наладить управление интерфейсом пользователя из модели

Модель — это отдельный компонент, отвечающий за обрабатывание информации для приложения. Модели не имеют зависимости от компонентов приложения, поэтому на них никак не «переносятся» проблемы, связанные с компонентами приложения. Важно соблюдать принцип управления интерфейсом из постоянной модели, потому что это несет в себе следующие свойства: пользователи вашего приложения не потеряют свои данные, если операционная система Андроид удалит вашу программу, освобождая ресурсы системы; ваш продукт будет работать даже в тех случаях, когда устройство не будет подключено к сети или связь с сетью будет слабой и нестабильной.

Ресурсы - один из основных компонентов, с которыми вам придётся работать очень часто. В Android принято держать некоторые объекты - изображения, строковые константы, цвета, анимацию, стили и т.п. за пределами исходного кода. Система поддерживает хранение ресурсов в отдельных файлах. Ресурсы легче поддерживать, обновлять, редактировать.

Каждое приложение на Android содержит каталог для ресурсов **res** и каталог для активов **assets**. Реальное различие между ресурсами и активами заключается в следующем:

* информация в каталоге ресурсов будет доступна в приложении через класс **R**, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге **res**) делает их легкодоступными для использования в коде программы;
* для чтения информации, помещённой в каталог активов **assets** (необработанный формат файла), необходимо использовать **AssetManager** для чтения файла как потока байтов.

Ресурсы в Android являются декларативными. В основном ресурсы хранятся в виде XML-файлов в каталоге **res** с подкаталогами **values**, **drawable-ldpi**, **drawable-mdpi**, **drawable-hdpi**, **layout**, но также бывают и другие типы ресурсов.

В папке *res* представлены несколько подпапок, точнее четыре. Она имеется даже при создании тестового базового приложения, и тем более начинает разрастаться при разработке крупных проектов. Оттого не удивительно, что ее файлы могут занимать много места на жестком диске устройства. Сейчас рассмотрим, что хранится в папке *res* при создании проекта приложения. Это 4 подпапки: 1) папка *drawable*, в которой содержаться различные файлы с изображениями; 2) папка *layout*, в которой располагаются xml-файлы, используемые для разработки интерфейса приложения; 3) папка *mipmap*, в которой хранятся иконки приложения; 4) папка *values*, которая хранит xml-файлы со значением цветов, строк, стилей и тем.

Для удобства система создаёт идентификаторы ресурсов и использует их в файле **R.java** (класс **R**, который содержит ссылки на все ресурсы проекта), что позволяет ссылаться на ресурсы внутри кода программы. Статический класс **R** генерируется на основе ваших заданных ресурсов и создаётся во время компиляции проекта. При создании класс содержит статические подклассы для всех типов ресурсов, для которых был описан хотя бы один экземпляр.

Так как файл **R** генерируется автоматически, то не смысла его редактировать вручную, потому что все изменения будут утеряны при повторной генерации.

В общем виде ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением. Удобств использования ресурсов заключается в том, что их можно изменять без повторной компиляции или новой разработки приложения. Имена файлов для ресурсов должны состоять исключительно из букв в нижнем регистре, чисел и символов подчёркивания.

Самыми распространёнными ресурсами являются, пожалуй, строки (string), цвета (color) и графические рисунки (bitmap). В приложении не рекомендуется применять жёстко написанные строки кода - вместо них следует использовать соответствующие идентификаторы, что позволяет изменять текст строкового ресурса, не изменяя исходного кода.

Использование ресурсов:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="appName">Project Name</string>  
</resources>

public class LoadingActivity extends Activity {

    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        String applicationName = getString(R.string.appName);  
    }  
    //...  
}

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent" >  
    <TextView  
        android:layout\_width="fill\_parent"   
        android:layout\_height="wrap\_content"   
        android:text="@string/appName" />  
</LinearLayout>

1. Компоненты Android приложения. Виды, назначение. Способ описания. Регистрация компонентов в манифесте. Жизненный цикл компонентов.

Архитектура Android приложений основана на идее многократного использования компонентов, которые являются основными строительными блоками. Каждый компонент является отдельной сущностью и помогает определить общее поведение приложения.

Базовые компоненты андроид приложения: **Activity, Service, Intents, ContentProvider, Broadcast Receivers**. Каждый из базовых компонент объявляется в андроид манифесте и может являться точкой входа в приложение.

**Activity:**

Activity – основная единица графического интерфейса (аналог окна или экранной формы)

Видимая часть приложения (экран, окно, форма), отвечает за отображение графического интерфейса пользователя

Например:

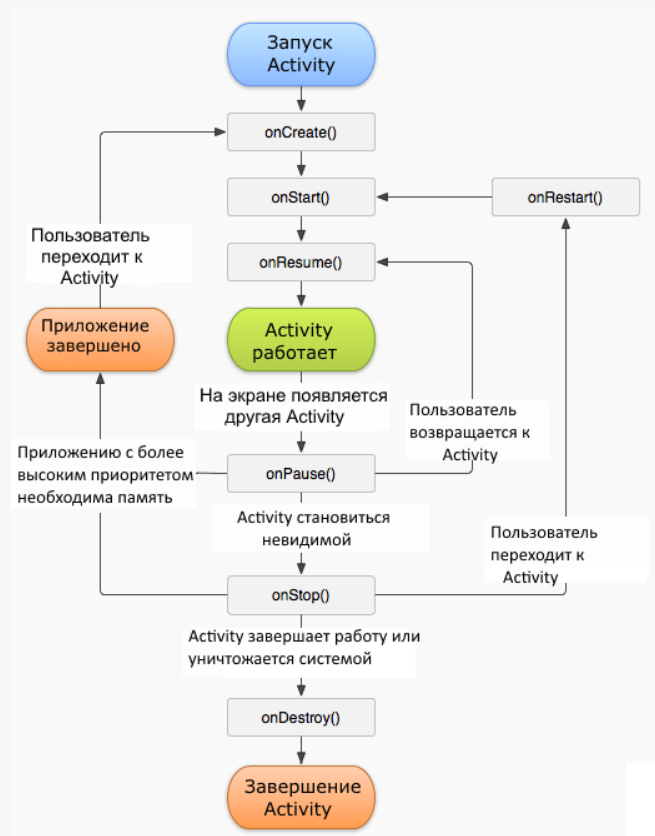
* Приложение для работы с будильником

Приложение может иметь несколько активностей. Активности приложения не зависят друг от друга.

Активность необходимо зарегистрировать в AndroidManifest.xml, метод регистрации, следующий:

<**activity android**:**name**="com.test.TestActivity"></**activity**>

Жизненный цикл:



**Services:**

Компонент, который работает в фоновом режиме, не имеет GUI, выполняет длительные по времени операции или работу для удаленных процессов

Примеры сервисов:

* проверка электронной почты
* получение гео-информации
* Проигрывание музыки в фоновом режиме

Может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

**Service** зарегистрируется в манифесте в секции **<application>**:

<service

android:name=".MyService"

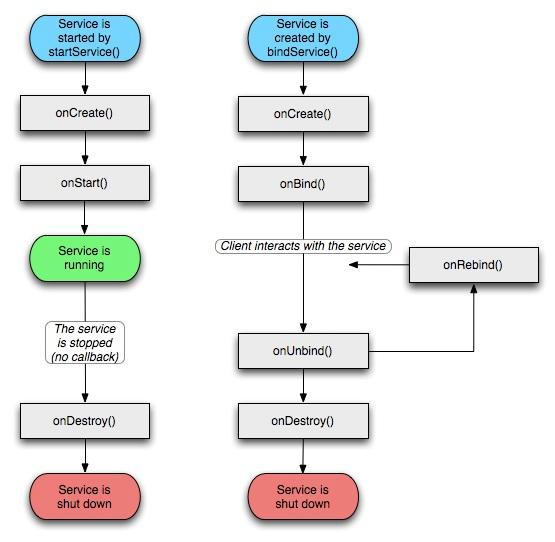
android:enabled="true"

android:exported="true" >

</service>

Жизненный цикл:

В каждом процессе открытия и закрытия Сервиса только onStart может вызываться несколько раз (с помощью нескольких вызовов startService), а другие onCreate, onBind, onUnbind и onDestory могут вызываться только один раз в жизненном цикле.



**Content Provider:**

Content Provider – “прослойка” между приложением и хранилищами данных

Управляет распределенным множеством данных приложения

Например:

* Контент-провайдер в системе Android, управляющий информацией о контактах пользователя

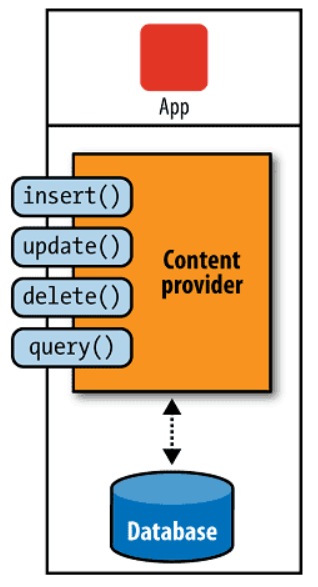
Данные могут храниться в файловой системе, в базе данных SQLite, в сети. Позволяет другим приложениям при наличии у них соответствующих прав делать запросы или даже менять данные.

Чтобы система Android могла узнать о Content Provider, необходимо зарегестрировать его в файле AndroidManifest.xml. В файл манифеста приложения надо добавить элемент <provider> с тремя атрибутами:

* android:name – имя класса, реализующего класс Content Provider
* android:authorities – строка URI, которая должна уникально идентифицировать этот Content Provider среди других, установленных на этом мобильном устройстве
* android:exported – флаг, разрешающий экспорт данных

Компоненты Content Provider, которые не объявлены в манифесте, не видимы в системе Android, и обращение к ним сгенерирует исключение во время выполнения программы.

Жизненный цикл:



**Broadcast Receivers:**

Приемник — компонент, который реагирует на широковещательные извещения

Например:

* Извещение о низком заряде батареи
* Инициирование широковещания

Не отображают пользовательского интерфейса. Могут создавать уведомление на панели состояний.

Существует два способа зарегистрировать BroadcastReceiver (а именно статическая регистрация и динамическая регистрация, статическая регистрация-декларация регистрации в манифесте, динамическая регистрация с использованием других компонентов в коде и регистрация через registerReceiver).

Для статической регистрации в файле манифеста в секции **application** следует создать секцию **receiver** и указать класс приёмника. Атрибут **android:exported="true"** сообщает получателю, что он может принимать широковещательные сообщения вне области приложения. Внутри секции указывается фильтр намерений в виде строки, чтобы определить, какие сообщения приёмник должен прослушивать.

<receiver

android:name=".YourReceiver"

android:enabled="true"

android:exported="true" >

<intent-filter>

<action android:name="ru.alexanderklimov.action.CAT" />

</intent-filter>

</receiver>

Динамическая регистрация происходит с помощью метода **Context.registerReceiver()**.

this.registerReceiver(mTimeBroadCastReceiver, new IntentFilter(

"android.intent.action.TIME\_TICK"));

Перед этим создаётся класс, расширяющий базовый класс **BroadcastReceiver** и реализуется метод обратного вызова **onReceive()** обработчика событий.

Жизненный цикл составляет всего около десяти секунд. Если вы выполните более десяти секунд в onReceive (), вы получите сообщение об ошибке, что программа ANR (Application No Response) не отвечает. Объект BroadcastReceiver действителен только при вызове onReceive (Context, Intent). После возврата из этой функции объект становится недействительным и жизненный цикл заканчивается.

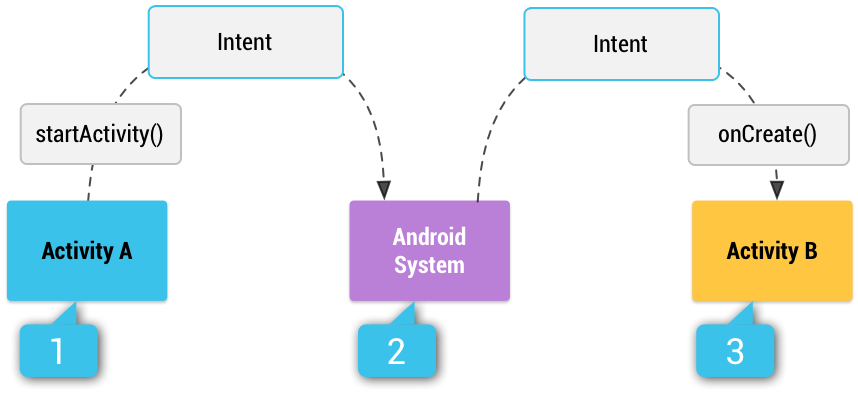
**Intents:**

Intents – системные сообщения, позволяющие приложениям обмениваться информацией между собой и с операционной системой:

* поступление телефонного звонка
* приход sms-сообщения
* вставлена SD-карта
* запущена новая активность

Intents – рекомендованный способ взаимодействия компонентов приложения. Наиболее распространённый сценарий использования намерения - запуск другой активности в своём приложении.

Жизненный цикл:



1. Расположения элементов. Основные виды, особенности, основные атрибуты.

Любая Activity (деятельность, окно приложения) состоит из Views и ViewGroups. View — виджет, который рисуется на экране, например, кнопка, текстовое поле, список. Все такие виджеты наследуются от класса android.view.View.  
ViewGroup — разметка, позволяющая расположить один или несколько View. Разметки наследуются от класса android.view.ViewGroup, который в свою очередь наследуется от android.view.View. Это значит что дочерними элементами разметки могут быть не только View, но и сами разметки.

**Примеры ViewGroups:**  
- **FrameLayout** – разметка для отображения одного элемента. Дочерние View или ViewGroup в FrameLayout выравниваются по верхнему левому углу. Разметка может содержать несколько элементов, но тогда они будут перекрывать друг друга.

Пример:

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/activity\_main"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent">  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Hello World!"  
        android:textSize="26sp"/>  
    <TextView  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Android Nougat 7.1"  
        android:textSize="26sp"  
        android:layout\_marginTop="5dp"/>  
</FrameLayout>

- **LinearLayout** – Разметка для отображения одного или нескольких элементов в одну линию, горизонтально или вертикально. Для выбора ориентации используется атрибут android:orientation с двумя возможными значениями «horizontal» и «vertical».

Пример:

<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:orientation="vertical">  
    <Button  
        android:id="@+id/button4"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Button" />

    <Button ... />  
    <Button ... />  
  
- **TableLayout** – разметка для расположения элементов в виде таблицы. Ряды задаются в xml с помощью тега *TableRow*, а ячейки в каждом ряду создаются автоматически для каждого элемента. Количество колонок в таблице будет равно максимальному количеству элементов в рядах.

Пример:

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical">  
    <TableRow  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent" >  
        <Button .../>  
        <Button .../>  
    </TableRow>  
    <TableRow  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent" >  
        <Button .../>  
    </TableRow>  
</TableLayout>

- RelativeLayout - разметка для расположения элементов относительно родителя или друг друга. Элементы начинают располагаться в указанном порядке, поэтому необходимо чтобы элемент был описан до того, как другой элемент будет на него ссылаться.

Пример:

<RelativeLayout

 android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<ImageView

 android:id="@+id/image"

  android:layout\_width="wrap\_content"

  android:layout\_height="wrap\_content"

  android:src="@drawable/image"

  android:layout\_centerHorizontal="true"

 />

 <Button

  android:layout\_width="wrap\_content"

  android:layout\_height="wrap\_content"

  android:layout\_toLeftOf="@id/image"

  android:text="@string/Left"

 />

 <Button

  android:layout\_width="wrap\_content"

  android:layout\_height="wrap\_content"

  android:layout\_toRightOf="@id/image"

  android:text="@string/Right"

 />

</RelativeLayout>

- WebView – компонент отображает внутри приложения веб-страницу. С его помощью можно загрузить из Интернета любую веб-страницу или HTML-документ, или даже превратить приложение в полноценный браузер.

Пример:

<WebView  
        android:id="@+id/webview"  
        android:layout\_width="300dp"  
        android:layout\_height="500dp"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent" />

WebView wv = findViewById(R.id.webview);  
wv.loadUrl("http://google.com");

1. Ресурсы Android приложения. Виды, назначение. Организация доступа к ресурсам. Альтернативные ресурсы.

Ресурсы - один из основных компонентов, с которыми вам придётся работать очень часто. В Android принято держать некоторые объекты - изображения, строковые константы, цвета, анимацию, стили и т.п. за пределами исходного кода. Система поддерживает хранение ресурсов в отдельных файлах. Ресурсы легче поддерживать, обновлять, редактировать.

Каждое приложение на Android содержит каталог для ресурсов **res** и каталог для активов **assets**. Реальное различие между ресурсами и активами заключается в следующем:

* информация в каталоге ресурсов будет доступна в приложении через класс **R**, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге **res**) делает их легкодоступными для использования в коде программы;
* для чтения информации, помещённой в каталог активов **assets** (необработанный формат файла), необходимо использовать **AssetManager** для чтения файла как потока байтов.

Ресурсы в Android являются декларативными. В основном ресурсы хранятся в виде XML-файлов в каталоге **res** с подкаталогами **values**, **drawable-ldpi**, **drawable-mdpi**, **drawable-hdpi**, **layout**, но также бывают и другие типы ресурсов.

В папке *res* представлены несколько подпапок, точнее четыре. Она имеется даже при создании тестового базового приложения, и тем более начинает разрастаться при разработке крупных проектов. Оттого не удивительно, что ее файлы могут занимать много места на жестком диске устройства. Сейчас рассмотрим, что хранится в папке *res* при создании проекта приложения. Это 4 подпапки: 1) папка *drawable*, в которой содержаться различные файлы с изображениями; 2) папка *layout*, в которой располагаются xml-файлы, используемые для разработки интерфейса приложения; 3) папка *mipmap*, в которой хранятся иконки приложения; 4) папка *values*, которая хранит xml-файлы со значением цветов, строк, стилей и тем.

Для удобства система создаёт идентификаторы ресурсов и использует их в файле **R.java** (класс **R**, который содержит ссылки на все ресурсы проекта), что позволяет ссылаться на ресурсы внутри кода программы. Статический класс **R** генерируется на основе ваших заданных ресурсов и создаётся во время компиляции проекта. При создании класс содержит статические подклассы для всех типов ресурсов, для которых был описан хотя бы один экземпляр.

Так как файл **R** генерируется автоматически, то не смысла его редактировать вручную, потому что все изменения будут утеряны при повторной генерации.

В общем виде ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением. Удобств использования ресурсов заключается в том, что их можно изменять без повторной компиляции или новой разработки приложения. Имена файлов для ресурсов должны состоять исключительно из букв в нижнем регистре, чисел и символов подчёркивания.

Самыми распространёнными ресурсами являются, пожалуй, строки (string), цвета (color) и графические рисунки (bitmap). В приложении не рекомендуется применять жёстко написанные строки кода - вместо них следует использовать соответствующие идентификаторы, что позволяет изменять текст строкового ресурса, не изменяя исходного кода.

Использование ресурсов:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="appName">Project Name</string>  
</resources>

public class LoadingActivity extends Activity {

    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        String applicationName = getString(R.string.appName);  
    }  
    //...  
}

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent" >  
    <TextView  
        android:layout\_width="fill\_parent"   
        android:layout\_height="wrap\_content"   
        android:text="@string/appName" />  
</LinearLayout>

Альтернативные ресурсы — это ресурсы, предназначенные для определенного устройства или конфигурации среды выполнения, например текущий язык, определенный размер экрана или плотность пикселей. Если Android может соответствовать ресурсу, который является более конкретным устройством или конфигурацией, чем ресурс по умолчанию, то этот ресурс будет использоваться вместо этого. Если он не находит альтернативный ресурс, соответствующий текущей конфигурации, будут загружены ресурсы по умолчанию.

Изображение выглядит как стол

Автоматически созданное описание

Например, вот некоторые стандартные и альтернативные ресурсы:

res/

drawable/

icon.png

background.png

drawable-hdpi/

icon.png

background.png

1. Активности (активити). Назначение, создание. Методы жизненного цикла.

Activity – основная единица графического интерфейса (аналог окна или экранной формы)

Видимая часть приложения (экран, окно, форма), отвечает за отображение графического интерфейса пользователя

Например:

* Приложение для работы с будильником

Приложение может иметь несколько активностей. Активности приложения не зависят друг от друга.

Класс **Acitvity** является важнейшим компонентом Android-приложения, а способ запуска и компоновки является фундаментальной частью платформы Android. В отличие от парадигм программирования, где приложение запускается в методе**main()**, система Android инициирует код в экземпляре Activity, вызывая специальные коллбэки, которые соответствуют конкретным этапам жизненного цикла активности.

Класс Activity предназначен для облегчения этой парадигмы. Когда одно приложение вызывает другое, вызывающее приложение запускает активность другого приложения, а не само приложение. Таким образом, активность служит точкой входа для взаимодействия приложения с пользователем.

Есть различные разновидности активностей, хотя все они так или иначе являются наследниками базового класса Activity. Например, если используется библиотека поддержки, то при создании нового проекта **Android Studio**генерирует класс **MainActivity**, который наследует от **AppCompatAcitivty**. Если посмотреть иерархию наследования, то в результате можно увидеть, что в начале этой иерархии будет находиться класс Activity.

Активность представляет собой окно, в котором приложение рисует свой UI. Это окно обычно полностью заполняет экран, но может быть меньше экрана или находиться поверх других окон. Как правило, одна активность представляет собой один экран в приложении. Например, одна активность может реализовывать окно настроек, в то время как другая — окно с выбором фото.

Большинство приложений имеют несколько экранов, из чего можно сделать вывод, что они содержат несколько активностей. Как правило, одна из активностей определяется как главная и является первым экраном, появляющимся при запуске пользователем приложения. Затем каждая активность может запустить другую активность для выполнения каких-либо действий.

Несмотря на то, что активности работают вместе, чтобы организовать хороший пользовательский интерфейс в приложении, каждая активность слабо связано с другими: обычно в приложении есть минимальные зависимости между активностями. Кроме того, активности часто запускают активности других приложений.

Создаётся одноимённый класс **MainActivity.java**. В этом классе будет реализовываться вся работа активности, при создании в ней уже есть переопределённый метод onCreate() и загрузка разметки активности.

public class MainActivity extends AppCompatActivity {

@Override protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}

Как уже говорилось выше, метод **setContentView()** загружает разметку активности. Эта разметка берётся из XML-файла, который размещается в **/res/layout**. При создании проекта там уже находится сгенерированный файл **activity\_main.xml**, идентификатор которого и передаётся в параметры setContentView(). Как правило, идентификатор идентичен названию файла. Если заглянуть в этот файл, то можно увидеть там **TextView** с надписью **Hello World!**, размещённое в центре экрана. Таким образом, то, что будет в коде этого файла, и увидит пользователь при запуске активности, для которой эта разметка установлена.

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity"

>

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello World!"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

/>

</android.support.constraint.ConstraintLayout>

Чтобы приложение могло использовать активность, она должна быть зарегистрирована в манифесте приложения. IDE уже сделала это за разработчика, если открыть файл **AndroidManifest.xml**, то можно обнаружить там внутри **<application>** элемент **<activity>** с именем, аналогичным имени активности.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="ru.androidtools.test">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

**<activity android:name=".MainActivity">**

**<intent-filter>**

**<action android:name="android.intent.action.MAIN"/>**

**<category android:name="android.intent.category.LAUNCHER"/>**

**</intent-filter>**

**</activity>**

</application>

</manifest>

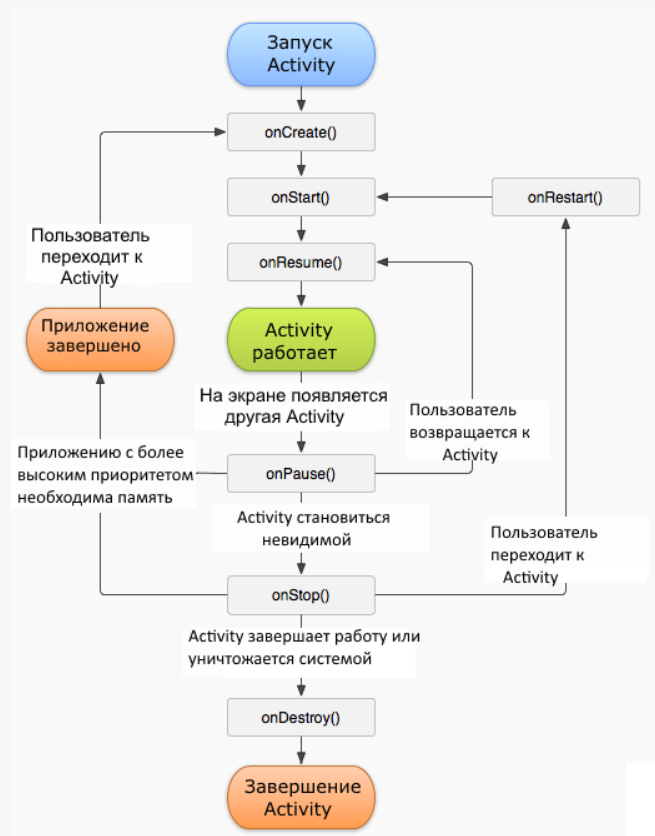
Одна из активностей в приложении должна быть главной. Поэтому внутри **<activity>** был автоматически добавлен интент-фильтр со следующими параметрами:

* **android.intent.action.MAIN** — сообщает системе о том, что данная активность является главной.
* **android.intent.category.LAUNCHER** — сообщает системе о том, что приложение должно показываться в списке приложений.

Вот всё, что нужно для того, чтобы создать активность. Подводя итог, можно сказать, что для создания активности Android Studio выполняет 3 шага:

1. Создание класса, наследующего от класса **Activity**.
2. Создание XML-файла разметки.
3. Регистрация активности в манифесте приложения.

Для навигации между этапами жизненного цикла активности класс Activity предоставляет базовый набор из шести коллбэков: **onCreate()**, **onStart()**, **onResume()**, **onPause()**, **onStop()** и **onDestroy()**. Система вызывает каждый из этих коллбэков как только активность переходит в новое состояние.



1. Сервисы. Назначение, создание. Методы жизненного цикла.

**Services** - компонент, который работает в фоновом режиме, не имеет GUI, выполняет длительные по времени операции или работу для удаленных процессов

Примеры сервисов:

* проверка электронной почты
* получение гео-информации
* Проигрывание музыки в фоновом режиме

Может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

Сервисы предназначены для длительного существования, в отличие от активностей. Они могут работать, постоянно перезапускаясь, выполняя постоянные задачи или выполняя задачи, требующие много времени.

Чтобы определить службу, необходимо создать новый класс, расширяющий базовый класс **Service**. Можно воспользоваться готовым мастером создания класса для сервиса в Android Studio. Щёлкаем правой кнопкой мыши на папке **java** (или на имени пакета) и выбираем **New | Service | Service.** Получим заготовку:

package ru.alexanderklimov.service;

import android.app.Service;

import android.content.Intent;

import android.os.IBinder;

public class MyService extends Service {

public MyService() {

}

@Override

public IBinder onBind(Intent intent) {

// TODO: Return the communication channel to the service.

throw new UnsupportedOperationException("Not yet implemented");

}

}

**Service** автоматически зарегистрируется в манифесте в секции **<application>**:

<service

android:name=".MyService"

android:enabled="true"

android:exported="true" >

</service>

Жизненный цикл:

В каждом процессе открытия и закрытия Сервиса только onStart может вызываться несколько раз (с помощью нескольких вызовов startService), а другие onCreate, onBind, onUnbind и onDestory могут вызываться только один раз в жизненном цикле.

Каждый кастомный класс сервиса должен быть подклассом Service и должен реализовывать и переопределять несколько базовых методов колбэка, связанных с жизненным циклом сервиса.

Вот наиболее важные из них:

* **onStartCommand():** вызывается системой, когда другой компонент хочет запустить сервис, использую startService(). Как только система вызывает onStartCommand(), сервис может работать в фоновом режиме столько времени, сколько потребуется для завершения процесса. Вам нужно не забыть остановить службу вручную, когда работа будет завершена, вызвав stopSelf() или topService(). Эта не нужно, если вы хотите просто выполнить привязку к сервису, не запуская его.
* **onBind():** Вам всегда нужно применять этот метод колбэка. Система использует этот колбэк при вызове bindService(). Используйте его для привязки другого компонента к сервису. Он открывает возможность коммуникации, таким образом вы предоставляете интерфейс для взаимодействия клиентов возвращая IBinder. Если вы не хотите использовать привязку, просто верните null.
* **onCreate():**Используйте этот метод для настройки сервиса перед его запуском. Система вызывает его, только если сервис еще не запущен.
* **onDestroy():** Система вызывает его, чтобы уничтожить сервис, когда он больше не нужен. Это последний вызов, который получает сервис. После этого сервис перестает использовать какие-либо ресурсы.

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

1. Поставщики (контент-провайдеры). Назначение, создание. Методы жизненного цикла.

Content Provider – “прослойка” между приложением и хранилищами данных. Управляет распределенным множеством данных приложения.

Контент-провайдеры (content providers) позволяют обращаться одним приложениям к данным других приложений. И мы таже можем сделать, чтобы другие приложения могули обращаться к данным нашего приложения через некоторый API.

Например:

* Контент-провайдер в системе Android, управляющий информацией о контактах пользователя

Данные могут храниться в файловой системе, в базе данных SQLite, в сети. Позволяет другим приложениям при наличии у них соответствующих прав делать запросы или даже менять данные.

Чтобы система Android могла узнать о Content Provider, необходимо зарегестрировать его в файле AndroidManifest.xml. В файл манифеста приложения надо добавить элемент <provider> с тремя атрибутами:

* android:name – имя класса, реализующего класс Content Provider
* android:authorities – строка URI, которая должна уникально идентифицировать этот Content Provider среди других, установленных на этом мобильном устройстве
* android:exported – флаг, разрешающий экспорт данных

Компоненты Content Provider, которые не объявлены в манифесте, не видимы в системе Android, и обращение к ним сгенерирует исключение во время выполнения программы.

Для создания собственного контент-провайдера нужно унаследовать от абстрактного класса **ContentProvider**:

public class MyContentProvider extends ContentProvider {

}

В классе необходимо реализовать абстрактные методы **query()**, **insert()**, **update()**, **delete()**, **getType()**, **onCreate()**. Прослеживается некоторое сходство с созданием обычной базы данных.

Метод **query()** является обязательным для класса **ContentProvider**. Если мы используем контент-провайдер для обращения к базе данных, то в нём вызывает одноимённый метод **SQLiteDatabase**. Состав метода практически идентичен.

Метод **insert()** содержит два параметра - **URI** и **ContenValues**. Первый параметр работает аналогично, как в методе **query()**. Вторая вставляет данные в нужные колонки таблицы. Для вставки используется вспомогательный метод **insertGuest()**.

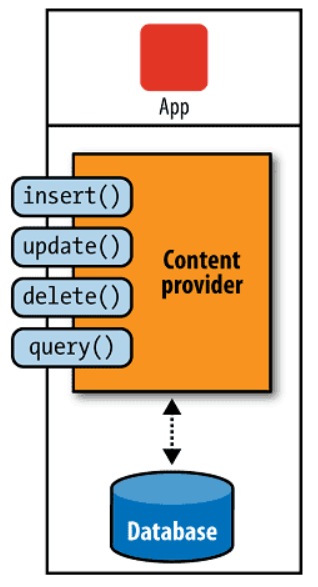
Метод **update ()** обновляет существующую запись от поставщика контента.

Метод **delete ()** удаляет существующую запись из провайдера контента.

Метод **getType ()** возвращает MIME-тип данных по указанному URI.

Метод **onCreate ()** вызывается при запуске провайдера.

Жизненный цикл:



1. Получатели широковещательных сообщений. Назначение, создание. Методы жизненного цикла.

Приемник — компонент, который реагирует на широковещательные извещения

Например:

* Извещение о низком заряде батареи
* Инициирование широковещания

Не отображают пользовательского интерфейса. Могут создавать уведомление на панели состояний.

Широковещательные сообщения делают приложение более открытым; передавая события, использующие сообщения, вы открываете компоненты своего приложения для сторонних приложений, и сторонние разработчики реагируют на события без необходимости изменять ваше оригинальное приложение. В своём приложении вы можете прослушивать широковещательные сообщения других приложений, заменить или улучшить функциональность собственного (или стороннего) приложения или реагировать на системные изменения и события приложений.

Класс **BroadcastReceiver** является базовым для класса, в котором должны происходить получение и обработка сообщений, посылаемых клиентским приложением с помощью вызова метода **sendBroadcast()**.

Существует два способа зарегистрировать BroadcastReceiver (а именно статическая регистрация и динамическая регистрация, статическая регистрация-декларация регистрации в манифесте, динамическая регистрация с использованием других компонентов в коде и регистрация через registerReceiver).

Для статической регистрации в файле манифеста в секции **application** следует создать секцию **receiver** и указать класс приёмника. Атрибут **android:exported="true"** сообщает получателю, что он может принимать широковещательные сообщения вне области приложения. Внутри секции указывается фильтр намерений в виде строки, чтобы определить, какие сообщения приёмник должен прослушивать.

<receiver

android:name=".YourReceiver"

android:enabled="true"

android:exported="true" >

<intent-filter>

<action android:name="ru.alexanderklimov.action.CAT" />

</intent-filter>

</receiver>

Динамическая регистрация происходит с помощью метода **Context.registerReceiver()**.

this.registerReceiver(mTimeBroadCastReceiver, new IntentFilter(

"android.intent.action.TIME\_TICK"));

Перед этим создаётся класс, расширяющий базовый класс **BroadcastReceiver** и реализуется метод обратного вызова **onReceive()** обработчика событий.

Жизненный цикл составляет всего около десяти секунд. Если вы выполните более десяти секунд в onReceive (), вы получите сообщение об ошибке, что программа ANR (Application No Response) не отвечает. Объект BroadcastReceiver действителен только при вызове onReceive (Context, Intent). После возврата из этой функции объект становится недействительным и жизненный цикл заканчивается.

Метод **onReceive()** будет выполнен при получении широковещательного намерения, если полученное намерение соответствует фильтру. Приложения с зарегистрированными приёмниками широковещательных намерений будут запущены автоматически при получении соответствующего намерения. Метод должен быть завершён в течение пяти секунд, иначе появится диалоговое окно о принудительном закрытии.

Когда широковещательное сообщение прибывает для получателя сообщения, Android вызывает его методом **onReceive()** и передаёт в него объект **Intent**, содержащий сообщение. Приёмник широковещательных сообщений является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет **BroadcastReceiver**, т. е. выполняющийся в настоящее время код в методе обратного вызова **onReceive()**, как полагает система, является приоритетным процессом и будет сохранён, кроме случаев критического недостатка памяти в системе.

Для отмены регистрации используется метод **unregisterReceiver()** в контексте приложения, передавая ему в качестве параметра экземпляр широковещательного приёмника (обычно в методе **onPause()**):

unregisterReceiver(receiver);

Для объекта **BroadcastReceiver** нет никаких возможностей видеть или фиксировать намерения, используемые в методе **startActivity()**. Аналогично, когда вы передали намерение для запуска активности через объект **BroadcastReceiver**, вы не сможете найти или запустить требуемую активность. Эти две операции семантически полностью различаются: запуск активности через намерение является приоритетной операцией для системы, изменяющей содержимое экрана устройства, с которым в настоящее время взаимодействует пользователь. Передача широковещательных сообщений для системы является фоновой работой, о которой обычно не знает пользователь и которая, соответственно, имеет более низкий приоритет.

1. Приложение, состоящее из нескольких активностей. Взаимодействие между ними. Фильтры намерений.

Простые приложения состоят из одной активности. Более сложные приложения могут иметь несколько окон, т.е. они состоят из нескольких активностей, которыми надо уметь управлять и которые могут взаимодействовать между собой.

Рассмотрим некоторые особенности взаимодействия между activity в одном приложении. Допустим, у нас есть три activity: MainActivity, SecondActivity и ThirdActivity.

С помощью Intent, например, по нажатию кнопки MainActivity запускает SecondActivity:

|  |  |
| --- | --- |
| 1  2 | Intent intent = new Intent(this, SecondActivity.class);  startActivity(intent); |

На SecondActivity тоже есть кнопка, которая запускает ThirdActivity:

|  |  |
| --- | --- |
| 1  2 | Intent intent = new Intent(this, ThirdActivity.class);  startActivity(intent); |

На ThirdActivity также есть кнопка, которая возвращается к первой activity - MainActivity:

|  |  |
| --- | --- |
| 1  2 | Intent intent = new Intent(this, MainActivity.class);  startActivity(intent); |

Если мы последовательно запустим все activity: из главной MainActivity запустим SecondActivity, из SecondActivity - ThirdActivity, то в итоге у нас сложится следующий стек activity:

|  |
| --- |
| ThirdActivity |
| SecondActivity |
| MainActivity |

Если после этого из ThirdActivity мы захотим обратиться к MainActivity, то метод startActivity() запустит новый объект MainActivity (а не вернется к уже существующему), и стек уже будет выглядеть следующим образом:

|  |
| --- |
| MainActivity |
| ThirdActivity |
| SecondActivity |
| MainActivity |

То есть у нас будут две независимые копии MainActivity. Такое положение нежелательно, если мы просто хотим перейти к существующей. И этот момент надо учитывать.

Если мы нажмем на кнопку **Back (Назад)**, то текущая activity, которая находится на вершине стека, удаляется из стека, и предыдущая activity оказывается на вершине стека и возобновляет свою работу. И таким образом с помощью кнопки Back (Назад) мы сможем перейти к предыдущей activity в стеке. Например, в случае выше если мы нажмем на кнопку Назад, то MainActivity на вершине стека завершает свою работу, и на экране начинает отображаться ThirdActivity

|  |
| --- |
| ThirdActivity |
| SecondActivity |
| MainActivity |

Тем не менее иногда возникает необходимость упавлять переходом между activity. Например, в данном случае нам нежелательно при нажатии на кнопку в ThirdActivity запускать новую копию MainActivity вместо того, чтобы просто перейти к MainActivity, которая была запущена первой и находится в самом низу стека.

Для управления стеком из activity Android предлагает нам использовать флаги - константы, определенные в классе **Intent**. Применение определенного флага позволит нам определенным образом изменить положение в стеке для определенных activity.

Например, возьмем предыдущую задачу, когда после нажатия на кнопку в ThirdActivity запускается новый экземпляр MainActivity. Но мы хотим не запускать новую, а перейти к уже существующей.

|  |
| --- |
| MainActivity |
| ThirdActivity |
| SecondActivity |
| MainActivity |

Чтобы выйти из этой ситуации, мы можем использовать флаг **Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT**:

|  |  |
| --- | --- |
| 1  2  3 | Intent intent = new Intent(this, MainActivity.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT);  startActivity(intent); |

флаг **Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT** перемещает activity, к которой осуществляется переход на вершину стека, если она ужее есть в стеке. И в этом случае после перехода из ThirdActivity к MainActivity стек будет выглядеть следующим образом:

|  |
| --- |
| MainActivity |
| ThirdActivity |
| SecondActivity |

Если же нам просто надо перейти из ThirdActivity к MainActivity, как если бы мы перешли назад с помощью кнопки Back, то мы можем использовать флаги **Intent.FLAG\_ACTIVITY\_CLEAR\_TOP** и **Intent.FLAG\_ACTIVITY\_SINGLE\_TOP**:

|  |  |
| --- | --- |
| 1  2  3 | Intent intent = new Intent(this, MainActivity.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);  startActivity(intent); |

Флаг **Intent.FLAG\_ACTIVITY\_CLEAR\_TOP** очищает все activity кроме той, которая запускается (если она уже есть в стеке). А флаг **Intent.FLAG\_ACTIVITY\_SINGLE\_TOP** указывает, что если в вершине стеке уже есть activity, которую надо запустить, то она НЕ запускается (то она может существовать в стеке только в единичном виде).

В этом случае после перехода из ThirdActivity к MainActivity стек будет полностью очищен, и там останется одна MainActivity.

Еще один флаг - **Intent.FLAG\_ACTIVITY\_NO\_HISTORY** позволит не сохранять в стеке запускаемую activity. Например, при запуске SecondActivity мы не хотим ее сохранять в стеке:

|  |  |
| --- | --- |
| 1  2  3 | Intent intent = new Intent(this, SecondActivity.class);  intent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_HISTORY);  startActivity(intent); |

В этом случае при переходе по цепочке MainActivity -> SecondActivity -> ThirdActivity стек будет выглядеть следующим образом:

|  |
| --- |
| MainActivity |
| ThirdActivity |

Если намерение запрашивает выполнение какого-либо действия с определённым набором данных, то системе нужно уметь выбрать приложение (или компонент) для обслуживания этого запроса. На помощь приходят фильтры намерений, которые используются для регистрации активностей, сервисов и широковещательных приёмников в качестве компонентов, способных выполнять заданные действия с конкретным видом данных. С помощью этих фильтров также регистрируются широковещательные приёмники, настроенные на трансляцию намерением заданного действия или события.

Задействуя фильтры намерений, приложения объявляют, что они могут отвечать на действия, запрашиваемые любой другой программой, установленной на устройстве. Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика намерений, нужно добавить тег <intent-filter> в узел компонента в манифесте.

В фильтре намерений декларируется только три составляющих объекта **Intent**: действие, данные, категория. Дополнения и флаги не играют никакой роли в принятии решения, какой компонент получает намерение.

Например, в любом приложении есть главная активность, которая устанавливается как точка входа для задания:

<activity

android:name=".HelloWorldActivity"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN"

<category android:name="android.intent.category.LAUNCHER"

</intent-filter>

</activity>

Фильтр такого вида в элементе <action> помечает активность, как запускаемую по умолчанию. Элемент <category> заставляет значок и метку для деятельности отображаться на панели Application Launcher, давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

1. Явные и неявные фильтры намерений. Параметры. Действия. Взаимодействие с системными активностями.

Намерение (Intent) — это механизм для описания одной операции. В Android наиболее распространённый сценарий использования намерения - запуск другой активности в приложении. Также можно использовать для объявления о запуске активности или сервиса, направленных на выполнение каких-либо действий или для передачи уведомлений о том, что произошло некое событие. Намерения могут применяться для трансляции сообщений по системе.

Явные намерения

Вспомним код для перехода на вторую активность в своём приложении:

Intent intent = new Intent(HelloWorld.this, AboutActivity.class);

startActivity(intent);

В намерении явно указывается имя класса активности во втором параметре. Данный класс у нас прописан в манифесте, и система, просматривая манифест, запустит нужную активность. Это явный вызов, где вы явно указываете имя класса для запуска. После вызова метода создастся новая активность, которая запустится или возобновит свою работу, переместившись на вершину стека активностей.

// У первой активности

ArrayList<File> fileList = new ArrayList<File>();

Intent intent = new Intent(MainActivity.this, SecondActivity.class);

intent.putExtra("FILES\_LIST", fileList);

startActivity(intent);

// У второй активности

ArrayList<File> filelist =

(ArrayList<File>)getIntent().getSerializableExtra("FILES\_LIST");

Неявные намерения

В этом случае при создании намерения мы используем не имя класса, а указываем параметры action, data, category с определёнными значениями. Система анализирует, что мы тут такого понаписали и начинает подбирать подходящий вариант для запуска.

Комбинация значений в параметрах определяют цель, которую мы хотим достичь. Если параметры намерения совпадают с условиями нашего фильтра, то наше приложение (активность) будет вызвано. Неявные намерения — это механизм, позволяющий запрашивать анонимные компоненты приложений с помощью действий. Также можно передать дополнительные данные в другую активность, используя параметр намерения extras.

Простой пример создания намерения с предопределённым действием ACTION\_VIEW для запуска браузера и перехода на нужный адрес:

Uri address = Uri.parse("http://developer.alexanderklimov.ru");

Intent openLinkIntent = new Intent(Intent.ACTION\_VIEW, address);

if (openLinkIntent.resolveActivity(getPackageManager()) != null) {

startActivity(openlinkIntent);}

else { Log.d("Intent", "Не получается обработать намерение!"); }

В данном случае действие ACTION\_VIEW означает просмотр чего-либо. Мы указали нужные данные (адрес), и происходит запуск новой активности (браузера).

Константы действия

ACTION\_DELETE — Запуск активность, с помощью которой можно удалить данные, указанные в пути URI внутри намерения;

ACTION\_MAIN — Запускается как начальная активность задания

ACTION\_VIEW — Наиболее распространённое действие. Для данных, передаваемых через URI, ищется наиболее подходящий способ вывода. Выбор приложения зависит от схемы (протокола) данных.

Методы

Для работы с категориями в классе Intent определена группа методов:

addCategory() — помещает категорию в объект Intent

removeCategory() — удаляет категорию, которая была добавлена ранее

getCategories() — получает набор всех категорий, находящихся в настоящее время в объекте Intent

Фильтры намерений и запуск заданий

Если намерение запрашивает выполнение какого-либо действия с определённым набором данных, то системе нужно уметь выбрать приложение (или компонент) для обслуживания этого запроса. На помощь приходят фильтры намерений, которые используются для регистрации активностей, сервисов. Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика намерений, нужно добавить тег <intent-filter> в манифесте. В фильтре намерений декларируется только три составляющих объекта Intent: действие, данные, категория. Дополнения и флаги не играют никакой роли в принятии решения, какой компонент получает намерение.

Принцип работы фильтров намерений

При использовании метода startActivity() передаваемое неявное намерение, как правило, доходит лишь до одной активности. Если для выполнения заданного действия с указанными данными годятся сразу несколько активностей, пользователю предоставляется список выбора. Процесс, когда решается, какую активность лучше запустить, называется Утверждением намерений. Его цель — найти наиболее подходящий фильтр намерений.

1. Уведомления.

Toast - всплывающие уведомления. Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности окна приложения, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом текущая деятельность приложения остаётся работоспособной для пользователя. В течение нескольких секунд сообщение плавно закрывается. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Как правило, всплывающее уведомление используется для показа коротких текстовых сообщений.

Для создания всплывающего уведомления необходимо инициализировать объект Toast при помощи метода Toast.makeText(), а затем вызвать метод show() для отображения сообщения на экране:

Toast toast = Toast.makeText(getApplicationContext(),

"Пора покормить кота!", Toast.LENGTH\_SHORT);

toast.show();

По умолчанию стандартное всплывающее уведомление появляется в нижней части экрана. Изменить место появления уведомления можно с помощью метода setGravity(int, int, int). Метод принимает три параметра: стандартная константа для размещения объекта в пределах большего контейнера (например, GRAVITY.CENTER, GRAVITY.TOP и др.); смещение по оси X; смещение по оси Y

Как правило, для Toast используются короткие текстовые сообщения. При необходимости можно добавить к сообщению и картинку. Используя метод setView(), принадлежащий объекту Toast, можно задать любое представление для отображения.

Кроме [Toast-уведомлений](http://developer.alexanderklimov.ru/android/toast.php), существует также другой тип уведомлений, который выводится за пределами приложения, а именно, в верхней части телефона в системной строке состояния в виде значка с небольшим текстом. **Notifications** — это уведомления, которые пользователь видит в верхней части экрана, когда ему приходит новое письмо, сообщение, обновление и т.п.

Когда пользователь открывает расширенное сообщение, Android запускает объект Intent, который определён в соответствии с уведомлением. Можно также конфигурировать уведомление с добавлением звука, вибрации и мигающих индикаторов на мобильном устройстве. Этот вид уведомления удобен в том случае, когда приложение работает в фоновом режиме и должно уведомить пользователя о каком-либо важном событии.

Чтобы создать уведомление в строке состояния, необходимо использовать два класса:

* Notification — определяем свойства уведомления строки состояния: значок, расширенное сообщение и дополнительные параметры настройки (звук и др.)
* NotificationManager — системный сервис Android, который управляет всеми уведомлениями. Экземпляр NotificationManager создаётся при помощи вызова метода from(), а затем, когда надо показать уведомление пользователю, вызывается метод notify()

Для создания уведомления надо создать идентификатор уведомления. Также следует создать идентификатор канала. Далее формируется внешний вид и поведение уведомления через построитель NotificationCompat.Builder. Можно задать текст уведомления, значок, заголовок и прочие атрибуты. Выводится уведомление с помощью метода notify() - своеобразный аналог метода show() у Toast.

// Идентификатор уведомления

private static final int NOTIFY\_ID = 101;

// Идентификатор канала

private static String CHANNEL\_ID = "Cat channel";

@Override

public void onClick(View v) {

NotificationCompat.Builder builder =

new NotificationCompat.Builder(MainActivity.this, CHANNEL\_ID)

.setSmallIcon(R.drawable.ic\_pets\_black\_24dp)

.setContentTitle("Напоминание")

.setContentText("Пора покормить кота")

.setPriority(NotificationCompat.PRIORITY\_DEFAULT);

NotificationManagerCompat notificationManager =

NotificationManagerCompat.from(MainActivity.this);

notificationManager.notify(NOTIFY\_ID, builder.build()); } }); } }

Создадим новые объекты Intent и PendingIntent, которые описывают намерения и целевые действия. PendingIntent — это контейнер для Intent. Этот контейнер может быть использован для последующего запуска, вложенного в него Intent.

|  |  |
| --- | --- |
| 1  2  3 | Intent resultIntent = new Intent(this, MainActivity.class);  PendingIntent resultPendingIntent = PendingIntent.getActivity(this, 0, resultIntent,                 PendingIntent.FLAG\_UPDATE\_CURRENT); |

1. Адаптация приложения к различным размерам экрана. Использование альтернативных приложений. Квалификаторы.

Чтобы создать масштабируемый макет, способный адаптироваться к разным экранам, можно использовать в качестве значений ширины и высоты отдельных компонентов представления параметры "wrap\_content"и "match\_parent". Если используется "wrap\_content", для ширины или высоты представления устанавливается минимальное значение, позволяющее уместить содержание на экран, а параметр "match\_parent" служит для растягивания компонента по размеру родительского представления. Если указать параметры "wrap\_content" и "match\_parent" вместо строго заданных размеров, в представлениях будет использоваться минимально необходимое место или они будут растягиваться на всю доступную длину и ширину соответственно. В результате макет правильно отображается на экранах разных размеров при разных ориентациях.

С помощью вложенных экземпляров объекта [LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) и параметров "wrap\_content" и "match\_parent" можно создавать достаточно сложные макеты. Однако [LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) не дает возможности точно управлять взаимным расположением дочерних представлений: в [LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) они просто помещаются в ряд друг за другом. Если необходимо расположить дочерние представления иным образом, используйте объект [RelativeLayout](http://developer.android.com/reference/android/widget/RelativeLayout.html), позволяющий задать относительные позиции компонентов. Например, одно дочернее представление можно выровнять по левому краю экрана, а другое – по правому.

Например:

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent">

    <TextView

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

    <EditText

        android:layout\_width="match\_parent"

        android:layout\_height="wrap\_content"

    <Button

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"/>

    <Button

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"/>

</RelativeLayout>

Масштабируемые или относительные макеты имеют свои ограничения. Хотя они позволяют создать интерфейс, способный адаптироваться к разным экранам за счет растягивания пространства внутри и вокруг компонентов, пользователю может оказаться не слишком удобно работать с таким интерфейсом. Поэтому в приложении должен использоваться не один масштабируемый макет, а несколько альтернативных вариантов для разных конфигураций экрана. Их можно создать с помощью [квалификаторов конфигураций](http://developer.android.com/guide/practices/screens_support.html#qualifiers), которые позволяют оперативно выбирать ресурсы, отвечающие текущим параметрам экрана (например, разные варианты макетов для экранов разных размеров).

res/layout/main.xml, однопанельный макет (по умолчанию):

<LinearLayout

    android:orientation="vertical"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent">

    <fragment android:id="@+id/headlines"

              android:layout\_height="fill\_parent"

              android:layout\_width="match\_parent" />

</LinearLayout>

Одной из проблем, с которой сталкивались разработчики приложений для устройств Android версий до 3.2, было слишком общее определение «большого» экрана. Квалификатор Smallest-width позволяет определять экраны с заданной минимальной шириной в dp. Квалификатор Smallest-width работает только на устройствах Android 3.2 или более поздних версий. Для совместимости с более ранними устройствами по-прежнему следует использовать абстрактные размеры (small, normal, large и xlarge).

Чтобы не создавать дубликаты файлов и упростить процесс поддержки приложения, используют псевдонимы. Например, можно определить следующие макеты:

res/layout/main.xml (однопанельный макет);

res/layout/main\_twopanes.xml (двухпанельный макет).

Затем добавить следующие два файла:

res/values-large/layout.xml:

<resources>

    <item name="main" type="layout">@layout/main\_twopanes</item>

</resources>

res/values-sw600dp/layout.xml:

<resources>

    <item name="main" type="layout">@layout/main\_twopanes</item>

</resources>

Содержание последних двух файлов одинаково, но сами по себе они не определяют макет. Они служат для того, чтобы назначить файл main в качестве псевдонима main\_twopanes. Так как в них используются селекторы large и sw600dp, они применяются к планшетным ПК и телевизорам на платформе Android независимо от версии.

1. Хранение пользовательских данных в Android приложении. Виды хранилищ.

Помимо прямого доступа к внутренним и внешним областям хранения Android-устройства, платформа Android предлагает базы данных SQLite для хранения реляционных данных и специальные файлы для хранения пар ключ-значение.

## 1. Хранение пар ключ-значение

SharedPreferences — это легкий класс хранения на платформе Android. Он в основном сохраняет некоторые часто используемые конфигурации, такие как состояние окна. Как правило, состояние окна перезагружается в Activity. Обычно OnSaveInstanceState сохраняется с помощью SharedPreferences. Он обеспечивает обычную длинную форму, Int формирование, сохранение типа String. Сам объект SharedPreferences может только получать данные и не поддерживает хранение и модификацию. Модификация хранилища осуществляется через объект Editor.

Активити и службы Android могут использовать метод getDefaultSharedPreferences() класса PreferenceManager, чтобы ссылаться на объект SharedPreferences, который может быть использован и для чтения, и для записи в файл настроек по умолчанию.

* SharedPreferences myPreferences
* PreferenceManager.getDefaultSharedPreferences(MyActivity.this);

Чтобы начать запись в файл настроек, вы должны вызвать метод edit() объекта SharedPreferences, который возвращает объект SharedPreferences.Editor.

* SharedPreferences.Editor myEditor = myPreferences.edit();

Объект SharedPreferences.Editor имеет несколько интуитивных методов, которые можно использовать для хранения новых пар ключ-значение в файле настроек. Например, вы можете использовать метод putString(), чтобы поместить пару ключ-значение со значением типа String. Аналогично для float:метод putFloat(). Следующий пример кода создаёт три пары ключ-значение:

* myEditor.putString("NAME", "Alice");
* myEditor.putInt("AGE", 25);
* myEditor.putBoolean("SINGLE?", true);

После того, как вы добавили все пары, вы должны вызвать метод commit() объекта SharedPreferences.Editor, чтобы сохранить их.

* myEditor.commit();

Чтение из объекта SharedPreferences гораздо проще. Всё, что вам нужно сделать, так это вызвать соответствующий метод get\*(). Например, чтобы получить пару ключ-значение, чьё значение является String, вы должны вызывать метод getString().

* String name = myPreferences.getString("NAME", "unknown");
* int age = myPreferences.getInt("AGE", 0);
* boolean isSingle = myPreferences.getBoolean("SINGLE?", false);

## 2. Использование базы данных SQLite

Каждое приложение для Android может создавать и использовать базы данных SQLite для хранения больших объемов структурированных данных. SQLite — это облегченный встроенный движок базы данных, который поддерживает язык SQL и использует очень мало памяти для хорошей производительности.

Особенности: Для оборудования с ограниченными ресурсами, Нет серверного процесса, Все данные хранятся в одном файле на разных платформах, Можно свободно копировать.

Чтобы создать новую базу данных SQLite или открыть уже существующую, вы можете использовать метод openOrCreateDatabase() внутри своей активити или службы. В качестве аргументов вы должны передать имя своей базы данных и режим, в котором вы хотите её открыть. Наиболее часто используемый режим MODE\_PRIVATE, который гарантирует, что база данных доступна только для вашего приложения. Например, вот как вы можете открыть или создать базу данных с именем my.db:

* SQLiteDatabase myDB = openOrCreateDatabase("my.db", MODE\_PRIVATE, null);

После создания базы данных вы можете использовать метод execSQL() для запуска SQL-инструкций:

* myDB.execSQL( "CREATE TABLE IF NOT EXISTS user (name VARCHAR(200), age INT, is\_single INT)" );

Хотя можно вставить новые строки в таблицу с помощью метода execSQL(), лучше использовать метод insert().

Вот два объекта ContentValues, которые вы можете использовать с таблицей user:

* ContentValues row1 = new ContentValues();
* row1.put("name", "Alice");
* ContentValues row2 = new ContentValues();
* row2.put("name", "Bob");

Ключи, передаваемые методу put(), должны соответствовать именам столбцов в таблице.

Когда объекты ContentValues готовы, вы можете передать их методу insert() вместе с именем таблицы.

* myDB.insert("user", null, row1);
* myDB.insert("user", null, row2);

Чтобы отправить запрос в базу данных, вы можете использовать метод rawQuery(), который возвращает объект Cursor, содержащий результаты запроса.

* Cursor myCursor = myDB.rawQuery("select name, age, is\_single from user", null);

Объект Cursor может содержать ноль или несколько строк. Самый простой способ перебрать все его строки, так это вызвать метод moveToNext() внутри цикла while.

Чтобы получить значение отдельного столбца, вы должны использовать такие методы, как getString() и getInt(), которые ожидают индекс столбца. Например, вот как вы получите все значения, вставленные в таблице user:

* while(myCursor.moveToNext()) {
* String name = myCursor.getString(0);
* int age = myCursor.getInt(1);
* boolean isSingle = (myCursor.getInt(2)) == 1 ? true:false; }

После того, как вы получите все результаты вашего запроса, убедитесь, что вы вызвали метод close() для объекта Cursor, чтобы освободить все ресурсы, которые он хранит.

* myCursor.close();

Аналогичным образом, когда вы закончили все операции с базой данных, не забудьте вызвать метод close() для объекта SQLiteDatabase.

* myDB.close();

## 3. Использование внутреннего хранилища

Каждое приложение Android имеет свой собственный внутренний каталог хранения, взаимодействующий с ним, в котором приложение может хранить текстовые и двоичные файлы. Файлы внутри этого каталога недоступны для пользователя или других приложений, установленных на устройстве пользователя. Они также автоматически удаляются, когда пользователь удаляет приложение.

Прежде чем вы сможете использовать внутренний каталог хранилища, вы должны определить его местоположение. Для этого вы можете вызвать метод getFilesDir(), который доступен как для активити, так и для служб.

* File internalStorageDir = getFilesDir();

Чтобы получить ссылку на файл внутри каталога, вы можете передать имя файла вместе с указанным вами расположением. Например, вот как вы получите ссылку на файл alice.csv:

* File alice = new File(internalStorageDir, "alice.csv");

С этого момента вы можете использовать свои знания о классах и методах I/O (ввода/вывода) Java для чтения или записи в файл. В следующем фрагменте кода показано, как использовать объект FileOutputStream и метод write() для записи в файл:

* // Create file output stream
* fos = new FileOutputStream(alice);
* // Write a line to the file
* fos.write("Alice,25,1".getBytes());
* // Close the file output stream
* fos.close();

1. Общие настройки. Использование их для хранения данных. Методы доступа.

**Общие настройки** — простой и легкий механизм, основанный на парах «ключ — значение» и предназначенный для сохранения примитивных данных приложения, чаще всего пользовательских настроек. Класс **SharedPreferences** позволяет создавать в приложении именованные ассоциативные массивы типа «ключ — значение», которые могут быть использованы различными компонентами приложения (работая при этом в контексте одного и того же приложения). Общие настройки поддерживают базовые типы boolean, string, float, long и integer, что делает их идеальным средством для быстрого сохранения значений по умолчанию, переменных экземпляра класса, текущего состояния UI и пользовательских настроек.

Сохранение значений параметров

Создаём переменную, представляющую экземпляр класса SharedPreferences, который отвечает за работу с настройками:

* private … var prefs: SharedPreferences
* override fun onCreate(savedInstanceState: Bundle?) {
* super.onCreate(savedInstanceState)
* setContentView(R.layout.activity\_main)
* prefs =
* getSharedPreferences("settings", Context.MODE\_PRIVATE) }

Вы передаёте в указанный метод getSharedPreferences() имя вашего файла и стандартное разрешение по умолчанию MODE\_PRIVATE - только приложение имеет доступ к настройкам. Когда мы закрываем приложение, то нам необходимо сохранить информацию в "Общих настройках". Обычно для этих целей используют методы onPause() или onStop(). Чтобы внести изменения в настройки, нужно использовать класс SharedPreferences.Editor. Получить объект Editor можно через вызов метода edit() объекта SharedPreferences. После того, как вы внесли все необходимые изменения, вызовите метод apply(), чтобы изменения вступили в силу.

* override fun onPause() {
* super.onPause()
* // Запоминаем данные
* val editor = prefs.edit()
* editor.putInt(APP\_PREFERENCES\_COUNTER, counter).apply() }

Мы можем прочитать сохранённое значение и использовать его. Сделать это можно в методе onResume().

* override fun onResume() {
* super.onResume()
* if(prefs.contains(APP\_PREFERENCES\_COUNTER)){
* // Получаем число из настроек
* counter = prefs.getInt(APP\_PREFERENCES\_COUNTER, 0)
* // Выводим на экран данные из настроек
* infoTextView.text = "Я насчитал $counter ворон" } }

Мы проверяем сначала наличие ключа APP\_PREFERENCES\_COUNTER, а затем извлекаем из ключа его значение.

1. Использование реляционных хранилищ в андроид приложении. Доступ к данным через поставщиков контента.

Каждое приложение для Android может создавать и использовать базы данных SQLite для хранения больших объемов структурированных данных. Чтобы создать новую базу данных SQLite или открыть уже существующую, вы можете использовать метод openOrCreateDatabase() внутри своей активити или службы. В качестве аргументов вы должны передать имя своей базы данных и режим, в котором вы хотите её открыть. Наиболее часто используемый режим MODE\_PRIVATE, который гарантирует, что база данных доступна только для вашего приложения. Например, вот как вы можете открыть или создать базу данных с именем my.db:

SQLiteDatabase myDB = openOrCreateDatabase("my.db", MODE\_PRIVATE, null);

После создания базы данных вы можете использовать метод execSQL() для запуска SQL-инструкций.

myDB.execSQL( "CREATE TABLE IF NOT EXISTS user (name VARCHAR(200), age INT, is\_single INT)");

Хотя можно вставить новые строки в таблицу с помощью метода execSQL(), лучше использовать метод insert(). Метод insert() ожидает объект ContentValues, содержащий значения для каждого столбца таблицы. Объект ContentValues очень похож на объект Map и содержит пары ключ-значение.

Вот два объекта ContentValues, которые вы можете использовать с таблицей user:

ContentValues row1 = new ContentValues();

row1.put("name", "Alice");

ContentValues row2 = new ContentValues();

row2.put("name", "Bob");

Ключи, которые мы передаем методу put(), должны соответствовать именам столбцов в таблице.

Когда объекты ContentValues готовы, можно передать их методу insert() вместе с именем таблицы.

myDB.insert("user", null, row1); myDB.insert("user", null, row2);

Чтобы отправить запрос в базу данных, можно использовать метод rawQuery(), который возвращает объект Cursor, содержащий результаты запроса. Cursor myCursor = myDB.rawQuery("select name, age, is\_single from user", null);

Чтобы получить значение отдельного столбца, вы должны использовать такие методы, как getString() и getInt().

После того, как получим все результаты вашего запроса, убедитесь, что вы вызвали метод close() для объекта Cursor, чтобы освободить все ресурсы, которые он хранит. myCursor.close();

Аналогичным образом, когда закончим все операции с базой данных, не забудьте вызвать метод close() для объекта SQLiteDatabase.

myDB.close();

Контент-провайдер или "Поставщик содержимого" (Content Provider) — это оболочка, в которую заключены данные. Если ваше приложение использует базу данных SQLite, то только ваше приложение имеет к ней доступ. Но бывают ситуации, когда данные желательно сделать общими. Поставщик содержимого применяется лишь в тех случаях, когда вы хотите использовать данные совместно с другими приложениями, работающих в устройстве.

В Android существует возможность выражения источников данных при помощи передачи состояния представления - REST, в виде абстракций, называемых поставщиками содержимого. В Android используются встроенные поставщики содержимого (пакет android.provider). На верхних уровнях иерархии располагаются базы данных, на нижних - таблицы. Обычно базы данных SQLite имеют расширение DB и доступ к ним открыт только из специальных пакетов реализации (implerentation package). Любой доступ к базе данных из-за пределов этого пакета осуществляется через интерфейс поставщика содержимого.

Для создания собственного контент-провайдера нужно унаследоваться от абстрактного класса ContentProvider:

public class MyContentProvider extends ContentProvider {}

В классе необходимо реализовать абстрактные методы query(), insert(), update(), delete(), getType(), onCreate().Некоторое сходство с созданием обычной базы данных.

А также его следует зарегистрировать в манифесте с помощью тега provider с атрибутами name и authorities. Тег authorities служит для описания базового пути URI, по которому ContentResolver может найти базу данных для взаимодействия.

В поставщиках содержимого также встречаются REST-подобные гиперссылки, предназначенные для поиска данных и работы с ними.

Метод query() является обязательным для класса ContentProvider. Если мы используем контент-провайдер для обращения к базе данных, то в нём вызывает одноимённый метод SQLiteDatabase. Состав метода практически идентичен.

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

cursor = database.query(GuestEntry.TABLE\_NAME, projection, selection, selectionArgs,

null, null, sortOrder);}

Метод insert() содержит два параметра - URI и ContenValues. Первый параметр работает аналогично, как в методе query(). Вторая вставляет данные в нужные колонки таблицы. Для вставки используется вспомогательный метод insertGuest().

ContentResolver. Каждый объект Content, принадлежащий приложению, включает в себя экземпляр класса ContentResolver, который можно получить через метод getContentResolver(). ContentResolver contentResolver = getContentResolver();

ContentResolver используется для выполнения запросов и транзакций от активности к контент-провайдеру. ContentResolver включает в себя методы для запросов и транзакций, аналогичные тем, что содержит ContentProvider.

1. Основные понятия языка JavaScript. История, особенности, сравнение с другими языками. Система типов.

Язык программирования [JavaScript](https://progkids.com/blog/javascript) пользуется довольно высокой популярностью. Связано это с тем, что его можно использовать как для создания динамичности статических веб страниц, так и для написания полноценных приложений SPA и даже компьютерных игр. Первые предпосылки для появления этого языка появились еще в 1992 году, когда была начата разработка скриптового встраиваемого языка Cmm. В 1995 году Брендан Эйх получил задачу внедрить язык программирования в браузер Netscape. Последняя на сегодняшний день версия языка ES6 вышла в 2015 году. Язык придерживается принципа сокращения кода при большей функциональности.

К основным особенностям [JavaScript](https://progkids.com/blog/javascript) языка программирования относятся:

1. Динамическая типизация. Тип данных будет определяться только тогда, когда переменной или const будет присваиваться ее значение.
2. Гибкая работа с функциями. В JS функции можно не только выполнять, но еще и возвращать функции из функций, передавать функции в качестве параметров другим функциями и присваивать функции в качестве значения переменных.
3. JavaScript поддерживается всеми современным браузерами.
4. Объектно-ориентированное программирование. То есть это такая методология программирования, в которой вся программа представляется в виде совокупности объектов

Помимо этого, важной особенностью [JavaScript](https://progkids.com/blog/javascript) является его развитая инфраструктура. На сегодняшний день разработчики могут работать с большим количеством библиотек и фреймворков, несколькими сборщиками, вспомогательными библиотеками и генераторами статических сайтов. Что касается сфер применения язык [JavaScript](https://progkids.com/blog/javascript) широко используется в веб-разработке. Причем работает он в сочетании с HTML и CSS.

Помимо этого, на языке JavaScript можно создавать следующие программы и приложения: Разработка программного обеспечения для мобильных устройств (с помощью React Native). Разработка серверных решений с помощью Node.js.

Преимущества [JavaScript](https://progkids.com/blog/javascript) :

1. Незаменимость при разработке веб сайтов и приложений. Язык легко интегрируется с версткой и сервером.
2. Высокая скорость работы и производительность.
3. Большое количество инструментов и богатая инфраструктура.
4. Относительная простота. Написание программ занимает обычно меньше времени. При этом объем кода также обычно меньше, в сравнении со многими другими языками.
5. Относительная легкость изучения.

Недостатки [JavaScript](https://progkids.com/blog/javascript) :

1. Отсутствует возможность загрузки и чтения файлов.
2. Нестрогая типизация.
3. Уровень безопасности. В такой язык программирования как JS достаточно просто внедрить вредоносный код.

JavaScript является слабо типизированным или динамическим языком. Это значит, что вам не нужно определять тип переменной заранее. Тип определится автоматически во время выполнения программы.

Стандарт ECMAScript определяет 8 типов:

* 6 типов данных являющихся примитивами:
  + [Undefined](https://developer.mozilla.org/ru/docs/Glossary/undefined) (Неопределённый тип)  : typeof instance === "undefined"
  + [Boolean](https://developer.mozilla.org/ru/docs/Glossary/Boolean) (Булев, Логический тип) : typeof instance === "boolean"
  + [Number](https://developer.mozilla.org/ru/docs/Glossary/Number) (Число) : typeof instance === "number"
  + [String](https://developer.mozilla.org/ru/docs/Glossary/String) (Строка) : typeof instance === "string"
  + [BigInt](https://developer.mozilla.org/ru/docs/Glossary/BigInt)  : typeof instance === "bigint"
  + [Symbol](https://developer.mozilla.org/ru/docs/Glossary/Symbol) (в ECMAScript 6)  : typeof instance === "symbol"
* [Null](https://developer.mozilla.org/ru/docs/Glossary/Null) (Null тип ) : typeof instance === "object". Специальный примитив, используемый не только для данных но и в качестве указателя на финальную точку в [Цепочке Прототипов](https://developer.mozilla.org/ru/docs/Web/JavaScript/Inheritance_and_the_prototype_chain);
* [Object](https://developer.mozilla.org/ru/docs/Glossary/Object) (Объект) : typeof instance === "object". Простая структура, используемая не только для хранения данных, но и для создания других структур, где любая структура создаётся с использованием ключевого слова [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new): new [Object](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object), new [Array](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array), new [Map (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map" \o "Currently only available in English (US)), new [Set](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Set), new [WeakMap](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakMap), new [WeakSet](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/WeakSet), new [Date](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Date) и множество других структур;

По стандарту Java считается одним из самых простых языков программирования для изучения на сегодня. Язык JavaScript тоже попадает в эту категорию, поскольку он основан на синтаксисе Java. Считается, что **JavaScript**, как более современный, характеризуется более широким применением.

1. Основные операторы JavaScript. Условные операторы и циклы.

JavaScript поддерживает бинарные и унарные операторы, а также ещё один специальный тернарный оператор - условный оператор. Бинарная операция использует два операнда, один перед оператором и другой за ним: operand1 operator operand2 (3+4 или x\*y).

В свою очередь унарная операция использует один операнд, перед или после оператора: operator operand или operand operator (x++ или ++x).

### [Операторы присваивания](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B_%D0%BF%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

В результате операции присваивания операнду слева от оператора присваивания (знак "=") устанавливается значение, которое берётся из правого операнда. Существуют также составные операторы присваивания, которые используются для сокращённого представления операций (+=, -=, \*=, /=). Для более сложного присваивания в JavaScript есть синтаксис [деструктуризации](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) - это выражение, которое позволяет извлекать данные из массивов или объектов, используя синтаксис, который отражает конструкторы массивов и литералы объектов.

### [Операторы сравнения](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B_%D1%81%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F)

Сравнивает свои операнды и возвращает логическое значение, базируясь на истинности сравнения. Строки сравниваются на основании стандартного лексикографического порядка, используя Unicode-значения. В большинстве случаев, если операнды имеют разный тип, то JavaScript пробует преобразовать их в тип, подходящий для сравнения.

Равно (==) | Возвращает true, если операнды равны.///Не равно (!=)////Больше (>) | Возвращает true, если операнд слева больше операнда справа.///Больше или равно (>=)///Меньше (<)///Меньше или равно (<=).

### [Арифметические операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)

Используют в качестве своих операндов числа (также литералы или переменные) и в качестве результата возвращают одно числовое значение. Стандартными арифметическими операторами являются сложение (+), вычитание (-), умножение (\*), и деление (/). При работе с числами с плавающей точкой эти операторы работают аналогично их работе в большинстве других языках программирования. Остаток от деления (%)///Инкремент (++) | Унарный оператор. Добавляет единицу к своему операнду.///Декремент (--) | Вычитает единицу из значения своего операнда.///Унарный минус Унарный плюс | Возвращает отрицательное значение своего операнда.///Возведение в степень (\*\*)

### [Логические операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)

Обычно используются с булевыми (логическими) значениями; при этом возвращаемое ими значение также является булевым. Однако операторы && и || фактически возвращают значение одного из операндов, поэтому, если эти операторы используются с небулевыми величинами, то возвращаемая ими величина также может быть не булевой.

Логическое И (&&) | expr1 && expr2 (Логическое И) Возвращает операнд expr1, если он может быть преобразован в false; в противном случае возвращает операнд expr2. Таким образом, при использовании булевых величин в качестве операндов, оператор && возвращает true, если оба операнда true; в противном случае возвращает false. Логическое ИЛИ (||) Логическое НЕ (!)

### [Условный (тернарный) оператор](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_(%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9)_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80)

[Условный оператор](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) является единственным оператором JavaScript, который использует три операнда. Оператор принимает одно из двух значений в зависимости от заданного условия. Синтаксис оператора: condition *?* val1 *:* val2 /// (val1 и val2 обязательно должны что-то возвращать, поэтому в этой конструкции нельзя использовать continue или break)

Если condition (условие) - истина, то оператор принимает значение val1. В противном случае оператор принимает значение val2. var status = (age >= 18) ? "adult" : "minor"; Данное выражение присваивает значение "adult" переменной status, если age имеет значение 18 или более. В противном случае переменной status присваивается значение "minor".

## [Цикл for](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Loops_and_iteration#%D1%86%D0%B8%D0%BA%D0%BB_for)

Цикл **[for](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/for)**повторяет действия, пока не произойдёт какое-либо специальное событие завершения цикла. Оператор for в JavaScript аналогичен оператору for в Java и C. Объявление оператора for выглядит следующим образом: for ([начало]; [условие]; [шаг]) выражения

## [Цикл do...while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Loops_and_iteration#%D1%86%D0%B8%D0%BA%D0%BB_do...while)

Цикл [do...while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/do...while) повторяется пока заданное условие истинно. Оператор do...while имеет вид:

Do выражения

while (условие);

выражения выполняются пока условие истинно. Чтобы использовать несколько выражений, используйте блок-выражение { ... }, чтобы сгруппировать их. Если условие истинно, выражения выполнятся снова. В конце каждого прохода условие проверяется. Если условие ложно, выполнение приостанавливается и управление передаётся выражению после do...while.

do {

i += 1;

console.log(i);

} while (i < 5);

## [Цикл while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Loops_and_iteration#%D1%86%D0%B8%D0%BA%D0%BB_while)

Цикл [while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/while) выполняет выражения пока условие истинно. Выглядит он так:

while (условие)

выражения

Если условие становится ложным, выражения в цикле перестают выполняться и управление переходит к выражению после цикла. Условие проверяется на истинность до того, как выполняются выражения в цикле. Если условие истинно, выполняются выражения, а затем условие проверяется снова. Если условие ложно, выполнение приостанавливается и управление переходит к выражению после while.

var n = 0;

var x = 0;

while (n < 3) {

n++;

x += n; }

1. Функции в JavaScript. Передача параметров, возврат значений. Стрелочные функции.

**Функция в JavaScript** специальный тип объектов, позволяющий формализовать средствами языка определённую логику поведения и обработки данных.

Объявление функции состоит из ключевого слова [function](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/function) и следующих частей:

* Имя функции.
* Список параметров (принимаемых функцией), заключённых в круглые скобки () и разделённых запятыми.
* Инструкции, которые будут выполнены после вызова функции, заключают в фигурные скобки { }.

function square(number) { return number \* number; }

Инструкция [return](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/return) указывает на значение, которые будет возвращено функцией. Примитивные параметры (например, число) передаются функции значением; **значение** передаётся в функцию, но если функция меняет значение параметра,**это изменение не отразится глобально или после вызова функции.** Функция вида "function declaration statement" по синтаксису является инструкцией, ещё функция может быть вида "function definition expression". Такая функция может быть **анонимной** (она не имеет имени).

var square = function(number) { return number \* number; };

var x = square(4); // x получает значение 16

**Вызов** функции фактически выполняет указанные действия с указанными параметрами. Например, если вы определите функцию square, вы можете вызвать её следующим образом: square(5); Функция также может вызвать саму себя. Три способа рекурсии: по имени функции; [arguments.callee](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/arguments/callee); по переменной, которая ссылается на функцию. Для примера рассмотрим следующие функцию:

var foo = function bar() {

// здесь будут выражения };

Внутри функции (function body) все следующие вызовы эквивалентны: bar(); arguments.callee(); foo().

Функция, которая вызывает саму себя, называется рекурсивной функцией (recursive function). Получается, что рекурсия аналогична циклу (loop). Оба вызывают некоторый код несколько раз, и оба требуют условия (чтобы избежать бесконечного цикла, вернее бесконечной рекурсии).

В JavaScript параметры функции по умолчанию имеют значение undefined. Однако в некоторых ситуация может быть полезным поменять значение по умолчанию. С параметрами по умолчанию проверка наличия значения параметра в теле функции не нужна. Можно просто указать значение по умолчанию для параметра b в объявлении функции:

function multiply(a, b = 1) {

return a \* b; }

multiply(5); // 5

**Выражения стрелочных функций** имеют более короткий синтаксис по сравнению с [функциональными выражениями](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/function) и лексически привязаны к значению [this](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/this). Выражение стрелочных функций не позволяют задавать имя, поэтому стрелочные функции [анонимны](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function/name), если их ни к чему не присвоить.

### [Базовый синтаксис](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions#%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81)

(param1, param2, …, paramN) => { statements }

(param1, param2, …, paramN) => expression

// эквивалентно: (param1, param2, …, paramN) => { return expression; }

// Круглые скобки не обязательны для единственного параметра:

singleParam => { statements }

// Функция без параметров нуждается в круглых скобках:

() => { statements }///() => expression

// Эквивалентно: () => { return expression; }

До появления стрелочных функций, каждая новая функция имела своё значение [this](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/this).

function Person() {

// В конструкторе Person() `this` указывает на себя.

this.age = 0;

Стрелочные функции не содержат собственный контекст [this](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/this), а используют значение [this](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/this) окружающего контекста. Поскольку значение this определяется лексикой, правила строгого режима ([strict mode](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Strict_mode)) относительно this игнорируются:

var f = () => { 'use strict'; return this; };

f() === window; // или глобальный объект

Тело стрелочной функции может иметь краткую (concise body) или блочную (block body) форму.

Блочная форма не возвращает значение, необходимо явно вернуть значение.

var func = x => x \* x; // краткий синтаксис,

// неявно возвращает результат

var func = (x, y) => { return x + y; }; // блочный синтаксис,

// явно возвращает результат

1. Объектная модель JavaScript. Прототипы. Классы в ES6. Наследование. Объектные литералы.

Для создания механизма управления страницами с помощью JavaScript на клиентской стороне было предложено использовать объектную модель документа. Суть модели в том, что каждый HTML-контейнер — это объект, который характеризуется тройкой: свойства, методы, события. Объектами в JavaScript также является окно браузера и объект Navigator. Объекты с одинаковым набором свойств, методов и событий объединяются в классы. Методы JavaScript определяют функции изменения свойств объекта. С HTML объектами могут происходить события (onLoad, onClick …). Программист JavaScript может изменить обработчик события. В JS реализация наследования в основном реализуется цепочкой прототипов. Основная идея: используйте прототипы, чтобы позволить одному ссылочному типу наследовать свойства и методы другого ссылочного типа.

Предполагая, что объект-прототип равен экземпляру другого типа, объект-прототип в это время будет содержать указатель на другой прототип, и другой прототип также содержит указатель на другой конструктор. Если другой архетип — это другой тип экземпляра, то указанная выше взаимосвязь все еще действует, и этот уровень развития представляет собой цепочку экземпляров и прототипов.

Во-первых, код основного режима цепочки прототипов выглядит следующим образом

function FatherType(){

this.property = true; }

FatherType.prototype.getFatherValue = function(){

return this.property; };

function SonType(){

this.sonproperty = false; }

SonType.prototype = new FatherType (); // Унаследованный FatherType

SonType.prototype.getsonValue = function (){

return this.sonproperty ; };

var son1= new SonType();

alert(instance.getFatherValue()); //true

Приведенный выше код определяет два типа:FatherType И сынТип. У каждого типа есть атрибут и метод. Основное различие между ними в том, что SonType наследует FatherType, а наследование достигается путем создания экземпляра FatherType и присвоения этого экземпляра SonType.prototype. Суть реализации состоит в том, чтобы переписать объект-прототип, чтобы он стал экземпляром нового типа.

Преимущества заимствования конструкторов: относительно цепочки прототипов конструктор может использоваться для передачи параметров родительскому конструктору в конструкторе подкласса. Недостатки заимствования конструкторов: методы определены в конструкторе, поэтому функцию нельзя использовать повторно.

**Преимущества комбинированного наследования:** Сочетая преимущества цепочки прототипов и заимствуя конструктор, он не только реализует повторное использование функции через метод sing в прототипе, но также гарантирует, что каждый экземпляр имеет свои собственные свойства.

## Классы

**class** Person {

constructor(name) {

**this**.name = name;}

sayName() {

console.log(`Person ${**this**.name} said his name`);}}

**const** john = **new** Person('John');

john.sayName(); *// Person John said his name*

Пример выше можно записать в стиле ES5 следующим образом:

**var** Person = **function**(name) {

**this**.name = name;};

Person.prototype.sayName = **function**() {

console.log('Person ' + **this**.name + ' said his name');};

**var** john = **new** Person('John');

john.sayName(); *// Person John said his name*

Что нужно знать про классы:

* Создавая класс, вы пользуетесь блоком кода (всё, что находится между { и }), внутри которого объявляете, всё, что хотите видеть в прототипе.
* Запись class Person означает, что будет создана функция конструктор Person (всё точно так же, как и в ES5)
* Свойство constructor используется для обозначения того, что будет происходить непосредственно в самом конструкторе.
* Все методы для класса используют краткую запись, которую мы [обсуждали ранее](http://jsraccoon.ru/es6-object-literal) в статье про расширение литерала объектов.
* При перечислении методов не надо использовать запятые (на самом деле, они запрещены)

ES6 классы также обладают синтаксическим сахаром для реализации прототипного наследования. Для подобных целей используется extends:

**class** GreatPerson **extends** Person {

constructor(name, phrase) {

**super**(name);

**this**.phrase = phrase;}

sayPhrase() {

console.log(`${**this**.name} says: "${this.phrase}"`) }}

**const** jane = **new** Person('Jane', 'Hello, World!');

jane.sayName(); *// Person Jane said his name*

jane.sayPhrase(); *// Jane says: "Hello, World!"*

# ES6: Расширение литерала объектов

Присваивание свойств выглядит следующим образом:

**let** a = 1, b = 2;

**let** obj = { a, b };

Краткая форма записи методов реализована схожим образом.

**const** obj = {

a() { */\* function body \*/* },

b() { */\* function body \*/* }};

Когда вы используете ES6 вариант методов a() { /\* function body \*/ }, то подразумеваете: “Создай метод a, который будет содержать в себе анонимную функцию”:

**const** obj = {

a() { */\* function body \*/* }};

*// Эквивалентно*

**var** obj = {

a: **function**() { */\* function body \*/* }}

*// Анонимная функция в методе factorial ES6 вариант*

**var** math = {

factorial(num) { **return** num <= 0 ? 1 : num \* factorial(num - 1); }};

console.log(math.factorial(5)); *// factorial is not defined*

Данный способ создания методов короче, но может принести с собой ряд проблем. Использовать его стоит только в тех случаях, когда вы точно уверены, что вы никогда не будете использовать рекурсию или передавать функцию в обработчики событий.

С релизом ES6 установить прототип объекта теперь можно согласно стандарту. Для этого используется функция Object.setPrototypeOf, которая принимает два объекта. Первому переданному объекту будет присвоен второй в качестве прототипа:

**const** proto = { */\* properties and methods \*/* };

**const** obj = { */\* properties and methods \*/* };

**Object**.setPrototypeOf(obj, proto);

1. Текстовый ввод-вывод в JavaScript. Использование консоли как отладчика.

Чтобы обрабатывать информацию в JavaScript, используются переменные. С помощью переменных в JavaScript можно получать информацию от пользователя, изменять её и выводить на веб страницу. Чтобы задать переменную в JavaScript используется конструкция /// let имя переменной = начальное значение; let a = 5; let name = "Иван";

Чтобы вывести информацию на веб страницу с помощью JavaScript, можно использовать модальное окно alert. alert(информация); Для вывода текста в JS используется конструкция alert("Текст");

**Для вывода информации в виде текста в JavaScript используется команда:** document.write(информация);

Для ввода информации в JS используется окно ввода информации prompt 🡪 переменная= prompt('Текст'); Функция prompt принимает два аргумента: result = prompt(title, default); Она выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL. var test = prompt("Тест", ''); // <-- так лучше

Синтаксис: result = confirm(question);

confirm выводит окно с вопросом question с двумя кнопками: OK и CANCEL.

**Результатом будет true при нажатии OK и false – при CANCEL(Esc).**

Например: var isAdmin = confirm("Вы - администратор?"); alert( isAdmin );

prompt выводит сообщение и ждёт, пока пользователь введёт текст, а затем возвращает введённое значение или null, если ввод отменён (CANCEL/Esc). confirm выводит сообщение и ждёт, пока пользователь нажмёт «OK» или «CANCEL» и возвращает true/false.

Поэтапный процесс отладки программы. Есть несколько способов открыть на странице Консоль Разработчика: • Клавиатура. Ctrl + Shift + I, o Internet Explorer. (клавиша - F12) o Mac OS X. (сочетание клавиш) • Панель Меню

Настройка и управление (кнопка справа вверху браузера) > Дополнительные инструменты > Инструменты разработчика

• Контекстное меню

Нажмите правой кнопкой мыши на любом участке веб-страницы, появится контекстное меню, в котором Вам нужно выбрать пункт “Посмотреть код”.

Следующей вкладкой в Инструментах Разработчика является панель Консоль (или Console в браузере Chrome), которая отображает логи работы программы. Например, ошибки, предупреждения, и другую дополнительную информацию, которую генерирует программа JavaScript в ходе своего выполнения. А также эта панель позволяет непосредственно взаимодействовать с кодом JavaScript на странице.

### Пример отладок: Счётчик вызовов: Функция console.count() возвращает количество раз, когда её вызывали. Учтите, что если вы измените строку лога, отдаваемую функции, счётчик обнулится и начнёт отсчёт для новой строки. Также есть функция сброса счётчика console.countReset().

### Таймер в консоли: Команды console.time() и console.timeEnd() позволяют соответственно начать и остановить отсчёт времени. Это нужно для проверки производительности. Также можно передать функциям строку для создания специфического таймера.

### Работа с логическими выражениями: Допустим, нужно проверить, приняло ли какое-то выражение значение false и записать это в лог. В обычной ситуации можно использовать условный оператор if, но в нашем случае куда лучше подойдёт функция console.assert(). Функция принимает в качестве аргумента выражение, а также сообщение либо объект.

function greaterThan(a, b) {

console.assert(a > b, {"message": "a is not greater than b", "a": a, "b": b});}

greaterThan(2, 1);

### Метка времени: Функция console.timeStamp() добавляет событие в таймлайн во время записи. Можно использовать, например, по окончании процесса обработки данных или для того, чтобы поймать момент возвращения вызова API. В качестве аргумента функция принимает название какого-либо события.

1. Массивы и функции высших порядков в Javascript. Деструктуризация массивов и объектов.

В JS функции представлены в виде объектов особого типа — это объекты типа Function. Рассмотрим пример:  
**function** **greeting**() {  
  console.log('Hello World');}  
// Вызов функции  
greeting();  // выводит 'Hello World'

В JavaScript с функциями можно делать то же самое, что можно делать с сущностями других типов, таких, как Object, String, Number. Функции можно передавать как параметры другим функциям. Такие функции, переданные другим, обычно выступают в роли функций обратного вызова (коллбэков). Функции можно назначать переменным, хранить их в массивах, и так далее. Именно поэтому функции в JS — это объекты первого класса.

Функции можно назначать переменным и константам:  
**const** square = **function**(x) {

**return** x \* x;}

Функции, назначенные переменным или константами, можно назначать другим переменным или константам: **const** foo = square;

Функции высшего порядка — это функции, которые работают с другими функциями, либо принимая их в виде параметров, либо возвращая их. Например, встроенные функции JavaScript Array.prototype.map, Array.prototype.filter и Array.prototype.reduce являются функциями высшего порядка.

### Метод Array.prototype.map

Метод map() создаёт новый массив, вызывая, для обработки каждого элемента входного массива, коллбэк, переданный ему в виде аргумента. Этот метод берёт каждое возвращённое коллбэком значение и помещает его в выходной массив. Функция обратного вызова, передаваемая map(), принимает три аргумента: element (элемент), index (индекс) и array (массив).

### Метод Array.prototype.filter

Метод filter() создаёт, на основе массива, новый массив, в которой попадают элементы исходного массива, соответствующие условию, заданному в переданной этому методу функции обратного вызова. Эта функция принимает, как и в случае с методом map(), 3 аргумента: element, index и array. Рассмотрим пример, построенный по той же схеме, что и при рассмотрении метода map().  
Метод Array.prototype.reduce

Метод reduce() обрабатывает каждый элемент массива с помощью коллбэка и помещает результат в единственное выходное значение. Этот метод принимает два параметра: коллбэк и необязательное начальное значение (initialValue). Коллбэк принимает четыре параметра: accumulator (аккумулятор), currentValue (текущее значение), currentIndex (текущий индекс), sourceArray (исходный массив).

Деструктурирующее присваивание в JavaScript — это изящный способ извлечения значений из массивов и объектов.

**Деструктуризация массива** — это особый синтаксис, позволяющий извлекать значения из массива и записывать их в новые переменные с минимумом кода.

const profile = ["Oluwatobi", "Sofela"];

const [firstName, lastName] = profile;

console.log(firstName); // "Oluwatobi"

JavaScript позволяет деструктурировать массив напрямую следующим образом:

const [firstName, lastName] = [

"Oluwatobi",

"Sofela"];

console.log(firstName); // "Oluwatobi"

При использовании деструктуризации массива JavaScript позволяет отдельно объявлять переменные, а уже затем присваивать им значения.

let firstName;

[firstName] = ["Oluwatobi"];

console.log(firstName); // "Oluwatobi"

JavaScript позволяет использовать [оператор остатка](https://www.codesweetly.com/javascript-rest-operator) внутри деструктурирующего массива для присваивания переменной значений остальных элементов обычного массива. Установить значения по умолчанию может быть полезно на тот случай, если значение, которое требуется извлечь из массива, не существует (или имеет значение undefined).

При помощи деструктурирующего присваивания массива можно поменять местами значения двух или нескольких переменных.

**Деструктуризация объекта** — это особый синтаксис, позволяющий извлекать значения из объекта и записывать их в новые переменные с минимумом кода.

JavaScript позволяет использовать деструктуризацию объекта для извлечения значения из свойства в переменную, даже если имена свойства и переменной отличаются.

const profile = {

firstName: "Oluwatobi"};

const { firstName: forename } = profile;

console.log(forename); // "Oluwatobi"

JavaScript допускает прямую деструктуризацию объекта свойств следующим образом:

const { firstName} = { firstName: "Oluwatobi"};

console.log(firstName); // "Oluwatobi"

Примечание:

* Деструктурирующее присваивание объекта нужно обязательно заключать в круглые скобки, чтобы компьютер понимал, что деструктуризация объекта — это литерал объекта, а не блок.
* После круглых скобок деструктурирующего присваивания объекта необходимо ставить точку с запятой (;). Тогда компьютер не будет интерпретировать круглые скобки как вызов функции, которая могла находиться на предыдущей строке.

При помощи деструктурирующего присваивания объекта можно поменять местами значения двух или нескольких переменных.

let firstName = "Oluwatobi";

let website = "CodeSweetly";

({ firstName, website } = {firstName: website, website: firstName});

console.log(firstName); // "CodeSweetly"

console.log(website); // "Oluwatobi"

1. Объектная модель документа. Основные приемы работы.

***Объектная модель документа (DOM)*** — это фундаментальный прикладной программный интерфейс, обеспечивающий возможность работы с содержимым HTML и XML-документов. Document, window — это объекты, чьи интерфейсы вы, как правило, очень часто используете в программировании DOM. Говоря простыми словами, объект window представляет что-то вроде браузера, а объект document - корень самого документа. Element наследуется от общего интерфейса Node, и эти интерфейсы вместе предоставляют много методов и свойств, которые можно применять у отдельных элементов. Прежде всего, следует понимать, что вложенные элементы HTML или XML-документов представлены в виде дерева объектов DOM. Древовидное представление HTML-документа содержит узлы, представляющие элементы или теги, такие как <body> и <p>, и узлы, представляющие строки текста. Узлы, представляющие HTML-элементы, являются узлами типа Element, а узлы, представляющие текст, - узлами типа Text. Document, Element и Text — это подклассы класса Node. Document и Element являются двумя самыми важными классами в модели DOM.

Работа большинства клиентских программ на языке JavaScript так или иначе связана с манипулированием элементами документа. В ходе выполнения эти программы могут использовать глобальную переменную document, ссылающуюся на объект Document. Модель DOM определяет несколько способов выборки элементов. Выбрать элемент или элементы документа можно: по значению атрибута id; по значению атрибута name; по имени тега; по имени класса или классов CSS; по совпадению с определенным селектором CSS.

Все HTML-элементы имеют атрибуты id. Значение этого атрибута должно быть уникальным в пределах документа - никакие два элемента в одном и том же документе не должны иметь одинаковые значения атрибута id. Выбрать элемент по уникальному значению атрибута id можно с помощью метода **getElementById()** объекта Document: var section1 = document.getElementById("section1");

HTML-атрибут name первоначально предназначался для присваивания имен элементам форм, и значение этого атрибута использовалось, когда выполнялась отправка данных формы на сервер. Подобно атрибуту id, атрибут name присваивает имя элементу. Однако, в отличие от id, значение атрибута name не обязано быть уникальным. Выбрать HTML-элементы, опираясь на значения их атрибутов name, можно с помощью метода **getElementsByName()** объекта Document: var radiobuttons = document.getElementsByName("favorite\_color"). Метод getElementsByName() определяется не Document, а классом HTMLDocument, он доступен только в HTML-документах и не доступен в XML-документах.

Метод **getElementsByTagName()** объекта Document позволяет выбрать все HTML или XML-элементы указанного типа (или по имени тега). var spans = document.getElementsByTagName("span"); Подобно методу getElementsByName(), getElementsByTagName() возвращает объект NodeList. Элементы документа включаются в массив NodeList в том же порядке, в каком они следуют в документе, т.е. первый элемент <p> в документе можно выбрать так: var firstParagraph = document.getElementsByTagName("p")[0];

Имена HTML-тегов не чувствительны к регистру символов, и когда getElementsByTagName() применяется к HTML-документу, он выполняет сравнение с именем тега без учета регистра символов.

Объект HTMLDocument также определяет свойства-синонимы **embeds** и **plugins**, являющиеся коллекциями HTMLCollection элементов <embed>. Свойство **document.body** представляет элемент <body> HTML-документа, а свойство **document.head** - элемент <head>. Эти свойства всегда определены в документе: даже если в исходном документе отсутствуют элементы <head> и <body>, браузер создаст их неявно.

Значением HTML-атрибута class является список из нуля или более идентификаторов, разделенных пробелами. Он дает возможность определять множества связанных элементов документа: любые элементы, имеющие в атрибуте class один и тот же идентификатор, являются частью одного множества. Стандарт HTML5 определяет метод **getElementsByClassName()**, позволяющий выбирать множества элементов документа на основе идентификаторов в их атрибутах class. Метод getElementsByClassName() принимает единственный строковый аргумент, но в самой строке может быть указано несколько идентификаторов, разделенных пробелами. Соответствующими будут считаться все элементы, атрибуты class которых содержат все указанные идентификаторы.

1. Событийная модель в JavaScript.

# Событийный цикл: микрозадачи и макрозадачи

Поток выполнения в браузере, равно как и в Node.js, основан на событийном цикле.

Идея событийного цикла очень проста. Есть бесконечный цикл, в котором движок JavaScript ожидает задачи, исполняет их и снова ожидает появления новых.

Общий алгоритм движка:

1. Пока есть задачи:
   * выполнить их, начиная с самой старой
2. Бездействовать до появления новой задачи, а затем перейти к пункту 1

Примеры задач:

* Когда загружается внешний скрипт <script src="...">, то задача – это выполнение этого скрипта.
* Когда пользователь двигает мышь, задача – сгенерировать событие mousemove и выполнить его обработчики.
* Когда истечёт таймер, установленный с помощью setTimeout(func, ...), задача – это выполнение функции func
* И так далее.

Задачи поступают на выполнение – движок выполняет их – затем ожидает новые задачи (во время ожидания практически не нагружая процессор компьютера)

Очередь, которую формируют такие задачи, называют «очередью макрозадач».

Например, когда движок занят выполнением скрипта, пользователь может передвинуть мышь,

Задачи из очереди исполняются по правилу «первым пришёл – первым ушёл». Когда браузер заканчивает выполнение скрипта, он обрабатывает событие mousemove, затем выполняет обработчик, заданный setTimeout, и так далее.

1. Рендеринг (отрисовка страницы) никогда не происходит во время выполнения задачи движком. Не имеет значения, сколь долго выполняется задача. Изменения в DOM отрисовываются только после того, как задача выполнена.
2. Если задача выполняется очень долго, то браузер не может выполнять другие задачи, обрабатывать пользовательские события, поэтому спустя некоторое время браузер предлагает «убить» долго выполняющуюся задачу. Такое возможно, когда в скрипте много сложных вычислений или ошибка, ведущая к бесконечному циклу.
3. Стек технологий веб-разработки. Основные языки разработки. Клиентская и серверная разработка.

Технологический стек — это набор языков программирования, фреймворков и ПО, необходимых для разработки приложения.

Поскольку веб-приложения состоят из клиентской и серверной частей, требования к их функциональности меняют языки программирования, фреймворки и ПО, с помощью которых будет вестись разработка. Другими словами, требования к функциональности клиентской и серверной частей влияют на технологический стек.

Клиентская сторона — это видимая часть веб-приложения, с которой взаимодействуют пользователи. Есть 3 главных элемента разработки клиентской части любого веб-приложения:

* JavaScript — язык программирования, который отвечает за интерактивную часть веб-приложения.
* HTML — язык разметки документов, который необходим для правильного отображения веб-приложения в браузере.
* CSS — формальный язык, который необходим для правильной стилизации веб-приложения.

Если говорить о фреймворках, чаще всего для разработки клиентской части любого веб-приложения используются Bootstrap и React.js.

Серверная часть веб-приложения — это то, что не видит пользователь, потому что она находится под клиентской частью. Для разработки серверной части используются: Язык бэкэнд-программирования; Фреймворки; База данных; Сервер. Также возможна разработка веб-приложения с бессерверной архитектурой.

Разработчики на стороне клиента используют свои навыки программирования для создания визуально привлекательных, функциональных и полезных веб-приложений и динамических веб-сайтов. Эти программисты несут ответственность за каждую часть веб-сайта, которую пользователи видят или с которой взаимодействуют. Домашние страницы, страницы покупок, слайд-шоу — практически любая видимая функция, которая отображается в веб-браузере или требует ввода данных пользователем, попадает под эту дисциплину.

Всё это программное обеспечение работает на клиентском устройстве. Разработчики на стороне клиента также известны как программисты «переднего плана», поскольку «передняя часть» веб-страницы — это то, что получает взаимодействие с пользователем. Эти разработчики получают хорошую оплату за свою работу, зарабатывая в США в среднем 100 208 долларов в год. Конечно, заработная плата может варьироваться в зависимости от местоположения и компании.

Общие языки сценариев, используемые разработчиками интерфейса для создания клиентского кода, включают JavaScript, HTML и CSS.

Чтобы понять, что делает разработчик на стороне сервера, мы должны сначала рассмотреть, как работает веб-сайт и чем он отличается от изображения или интерактивного текстового документа. Клиентская часть веб-сайта — это просто платформа для отображения и сбора информации. Сайт взаимодействует с веб-серверами, используя их для получения и отправки данных из баз данных, чтобы предоставить вам необходимые услуги.

Эти разработчики программного обеспечения проектируют, создают и обслуживают серверный код, который делает возможным такой обмен данными. Эти программисты также известны как разработчики серверной части. Они работают за кадром, следя затем, чтобы на серверах приложений всё работало должным образом. Бэкенд-программистам также очень хорошо платят, их зарплата в США составляет в среднем 117 284 долларов.

Эти программисты обычно владеют серверными языками, такими как Python, SQL, Ruby on Rails, PHP и Java.

1. Стилизация веб-страницы под ОС Android. Адаптивная верстка. Инструменты адаптивной верстки. Подход “mobile first”.

По сравнению со стационарными компьютерами и ноутбуками у мобильных устройств гораздо больше различий в их аппаратной составляющей. Экраны мобильных устройств могут иметь как очень большое, так и очень маленькое разрешение. Помимо этого, они могут автоматически переключаться между вертикальной и горизонтальной ориентацией в момент, когда пользователь поворачивает устройство.

[Адаптивный веб-дизайн](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps) – это термин, означающий набор методов, позволяющих вашему веб-сайту адаптировать разметку под среду, в которой он просматривается. Как правило, это изменяющиеся размеры и ориентация. Основные методы адаптивного веб-дизайна:

* обтекаемая разметка CSS, позволяющая плавно адаптировать страницу под изменяющиеся размеры окна
* использование [медиавыражений](https://developer.mozilla.org/en-US/CSS/Media_queries" \o "en/CSS/Media_queries), подключающих стили по условию, соответственно, [ширине](https://developer.mozilla.org/en-US/CSS/Media_queries#width) и [высоте](https://developer.mozilla.org/en-US/CSS/Media_queries#height) экрана.

[Meta-тег viewport](https://developer.mozilla.org/ru/docs/Mozilla/Mobile/Viewport_meta_tag) указывает браузеру, каким образом отображать ваш сайт в подходящем масштабе на устройстве пользователя.

Для использования сенсорного экрана осуществляется работа с [DOM Touch events](https://developer.mozilla.org/en-US/DOM/Touch_events). Вы не сможете использовать псевдо класс [CSS :hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover), а при проектировании интерактивных элементов нужно будет учитывать тот факт, что пальцы толще, чем указатели мыши. Можно использовать медиавыражение [-moz-touch-enabled](https://developer.mozilla.org/en-US/CSS/Media_queries#-moz-touch-enabled), чтобы использовать нужные правила CSS на устройствах, поддерживающих обработку нажатий на экран.

Чтобы создавать веб-сайты, которые будут работать приемлемо во всех мобильных браузерах:

* Старайтесь избегать использования стилей, специфических для браузеров, таких как свойства CSS с вендорными префиксами.
* Если всё же вам необходимо ими воспользоваться, убедитесь, что другие браузеры применяют свои собственные версии этих свойств, и укажите их.
* Для браузеров, которые не поддерживают эти свойства, обеспечьте приемлемый упрощённый вариант.

# **Адаптивная верстка**

Что представляет из себя **отзывчивый веб-дизайн** (*responsive web design*)? Это гибкая сетка макета вашего сайта, основанная на **относительных единицах измерения**, которая позволяет использовать один макет для разного типа устройств, гибкие изображения и видеоматериалы, и медиазапросы.

**Медиазапросы**, они позволяют отправлять браузеру различные стили в зависимости от определенных условий, например, таких как тип устройства, разрешение, ориентация и тому подобное. И так, **адаптивный веб-дизайн** включает в себя **гибкую сетку макета сайта**, **гибкие изображения** и **видеоматериалы**, **медиазапросы** и **прогрессивное улучшение**.

Для того, чтобы мобильные браузеры автоматически не изменяли размер страниц сайта, необходимо в теге [**<head>**](https://basicweb.ru/html/tag_head.php) разместить метатег, который сообщает браузеру, как обрабатывать размеры страницы и изменять ее масштаб: **<meta name = "viewport" content = "width=device-width, initial-scale = 1">**

* Атрибут **[name](https://basicweb.ru/html/tag_meta_att_name.php" \o "Атрибут name HTML тега meta)** задает имя документа метаданным, значение **"viewport"** дает подсказку браузеру о размере начального размера области просмотра. Функция атрибута **[content](https://basicweb.ru/html/tag_meta_att_content.php" \o "Атрибут content HTML тега meta)** задать значения для этого атрибута.
* Значение **width=device-width** атрибута **[content](https://basicweb.ru/html/tag_meta_att_content.php" \o "Атрибут content HTML тега meta)** сообщает, что ширина страницы устанавливается в соответствии с размером экрана устройства в аппаратно-независимых пикселях
* Значение **initial-scale=1** атрибута **[content](https://basicweb.ru/html/tag_meta_att_content.php" \o "Атрибут content HTML тега meta)** сообщает браузеру, что необходимо установить соответствие **1:1** для пикселей CSS и аппаратно-независимых пикселей вне зависимости от ориентации устройства.

## Приоритет создания стилей:

В настоящее время существует *два* основных подхода:

* **Mobile first** - подход, при котором разработка сайта начинается и ориентирована на пользователей мобильных устройств. В первую очередь описываются стили, предназначенные для смартфонов, а затем дизайн дорабатывается для планшетных устройств и настольных компьютеров. При использовании этого подхода Вы должны учитывать то, что Ваши страницы должны быть оптимизированы, максимально быстро загружаться и отображать важное содержание в первую очередь. Прочая информация должна загружаться и отображаться пользователю по его требованию, страница не должна загружать те ресурсы, которые не потребуются пользователю для получения необходимой информации. Подход **mobile first** может быть намного дороже, чем подход **desktop first** из-за более длительной фазы подготовки и опыта, необходимого для создания проекта.
* **Desktop first** - подход, при котором разработка ведется для многофункциональных проектов, ориентированных на пользователей настольных компьютеров. В первую очередь описываются стили, предназначенные для компьютеров, а затем дизайн дорабатывается для планшетных устройств и смартфонов.

1. Медиа запросы CSS. Их использование для верстки страниц для разных типов экранов.

Медиа-запрос — это функция CSS, которая используется для адаптации макета веб-страницы к различным размерам экрана и типам устройств.

Синтаксис

@media media type and (condition: breakpoint) { // Правила CSS }

Мы можем ориентироваться на различные типы устройств в зависимости от условий. Правила внутри медиа-запроса применяются только при совпадении условий and/or в типах устройств.

# Правило @Media

Оно используется для указания типов целевых устройств: @media () { // Правила CSS }

Внутри скобки мы устанавливаем условие. Например, нам нужно увеличить размер шрифта для мобильных устройств. Для этого устанавливаем максимальную ширину, которая проверяет ширину устройства:

.text {  
 font-size: 14px;  
}@media (max-width: 480px) {  
 .text {  
 font-size: 16px; } }

Обычно размер текста составляет 14 пикселей. Однако, поскольку мы применили медиа-запрос, он изменится до 16 пикселей при максимальной ширине устройства 480 пикселей и меньше. **Важно: медиа-запросы всегда должны находиться в конце CSS-файла.**

Типы устройств размещаются сразу после правила @media. Если тип не указан, правило @media выбирает все типы по умолчанию. Существует множество видов устройств, однако их можно разделить на 4 категории: all — для всех типов устройств; print — для принтеров; screen — для экранов компьютеров, планшетов и смартфонов; speech — для скринридеров, которые «читают» страницу вслух.

Например, чтобы выбрать только экраны, нужно установить ключевое слово screen сразу после правила @media. Правила также необходимо объединить ключевым словом «and»:

@media screen and (max-width: 480px) {  
 .text { font-size: 16px; } }

Точка останова — это ключ, который определяет, когда следует изменить макет и адаптировать новые правила внутри медиа-запросов. Вернемся к нашему примеру:

@media (max-width: 480px) {  
 .text {  
 font-size: 16px; } }

Здесь точка останова составляет 480 пикселей. Теперь медиа-запрос знает, когда устанавливать или перезаписывать новый класс. Если ширина устройства меньше 480 пикселей, то будет применен класс text.

Как уже было сказано выше, точки останова могут различаться и точно определенного стандарта нет, однако приведенные выше примеры используются чаще всего.

1. CSS Flexbox. Особенности, назначение, основные понятия и приемы использования.

**CSS flexbox** — модуль макета гибкого контейнера — представляет собой способ компоновки элементов, в основе лежит идея оси.

Flexbox состоит из**гибкого контейнера (flex container)** и **гибких элементов (flex items)**. Гибкие элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

Модуль flexbox позволяет решать следующие задачи:

* Располагать элементы в одном из четырех направлений: слева направо, справа налево, сверху вниз или снизу вверх.
* Переопределять порядок отображения элементов.
* Автоматически определять размеры элементов таким образом, чтобы они вписывались в доступное пространство.
* Решать проблему с горизонтальным и вертикальным центрированием.
* Переносить элементы внутри контейнера, не допуская его переполнения.
* Создавать колонки одинаковой высоты.

Flexbox решает специфические задачи — создание одномерных макетов, например, навигационной панели, так как flex-элементы можно размещать только по одной из осей. Для описания модуля Flexbox используется определенный набор терминов. Значение flex-flow и режим записи определяют соответствие этих терминов физическим направлениям: верх / право / низ / лево, осям: вертикальная / горизонтальная и размерам: ширина / высота.

**Главная ось (main axis)** — ось, вдоль которой выкладываются flex-элементы. Она простирается в основном измерении.

**Main start и main end** — линии, которые определяют начальную и конечную стороны flex-контейнера, относительно которых выкладываются flex-элементы.

**Основной размер (main size**) — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что из них находится в основном измерении, определяют основной размер flex-контейнера или flex-элемента.

**Поперечная ось** — ось, перпендикулярная главной оси. Она простирается в поперечном измерении.

**Cross start и cross end** — линии, которые определяют начальную и конечную стороны поперечной оси, относительно которых выкладываются flex-элементы.

**Поперечный размер (cross size)** — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что находится в поперечном измерении, являются их поперечным размером.

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для дочерних элементов не работают такие CSS-свойства, как float, clear, vertical-align. Также, на flex-контейнер не оказывают влияние свойства column-\*, создающие колонки в тексте. Модель flexbox-разметки связана с определенным значением CSS-свойства display родительского html-элемента, содержащего внутри себя дочерние блоки. Для управления элементами с помощью этой модели нужно установить свойство display следующим образом:

.flex-container {

/\*генерирует flex-контейнер уровня блока\*/

display: -webkit-flex;

display: flex; }

.flex-container {

/\*генерирует flex-контейнер уровня строки\*/

display: -webkit-inline-flex;

display: inline-flex; }

После установки данных значений свойства каждый дочерний элемент автоматически становится flex-элементом, выстраиваясь в один ряд.

Flex-элементы — блоки, представляющие содержимое flex-контейнера в потоке. Flex-контейнер устанавливает новый контекст форматирования для своего содержимого, который обуславливает следующие особенности:

* Для flex-элементов блокируется их значение свойства display. Значение display: inline-block; и display: table-cell; вычисляется в display: block;.
* Абсолютно позиционированный flex-элемент не участвует в компоновке гибкого макета.
* Поля margin соседних flex-элементов не схлопываются.
* Процентные значения margin и padding вычисляются от внутреннего размера содержащего их блока.
* margin: auto; расширяются, поглощая дополнительное пространство в соответствующем измерении. Их можно использовать для выравнивания или раздвигания смежных flex-элементов.

#### Направление главной оси: flex-direction

Свойство относится к flex-контейнеру. Управляет направлением главной оси, вдоль которой укладываются flex-элементы, в соответствии с текущим режимом записи. Свойство не наследуется.

**Синтаксис**

flex-direction: row; flex-direction: row-reverse;

flex-direction: column; flex-direction: column-reverse;

flex-direction: inherit; flex-direction: initial;

#### Управление многострочностью flex-контейнера: flex-wrap

Свойство определяет, будет ли flex-контейнер однострочным или многострочным, а также задает направление поперечной оси, определяющее направление укладки новых линий flex-контейнера. По умолчанию flex-элементы укладываются в одну строку, вдоль главной оси.

**Синтаксис**

flex-wrap: nowrap; // Значение по умолчанию

flex-wrap: wrap; // Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов

flex-wrap: wrap-reverse; // Flex-элементы переносятся на новые линии, располагаясь в обратном порядке слева-направо

flex-wrap: inherit; // значение свойства в значение по умолчанию

flex-wrap: initial; // Наследует значение свойства от родительского элемента

#### Краткая запись направления и многострочности: flex-flow

Свойство позволяет определить направления главной и поперечной осей, а также возможность переноса flex-элементов при необходимости на несколько строк. Представляет собой сокращённую запись свойств flex-direction и flex-wrap. Значение по умолчанию flex-flow: row nowrap;. Свойство не наследуется.

#### Порядок отображения flex-элементов: order

Свойство определяет порядок, в котором flex-элементы отображаются и располагаются внутри flex-контейнера. Применяется к flex-элементам. Первоначально все flex-элементы имеют order: 0;. При указании значения от -1 для элемента он перемещается в начало сроки, значение 1 — в конец. Если несколько flex-элементов имеют одинаковое значение order, они будут отображаться в соответствии с исходным порядком.

Определяющим аспектом гибкого макета является возможность «сгибать» flex-элементы, изменяя их ширину / высоту (в зависимости от того, какой размер находится на главной оси), чтобы заполнить доступное пространство в основном измерении. Это делается с помощью свойства flex. Flex-контейнер распределяет свободное пространство между своими дочерними элементами (пропорционально их коэффициенту flex-grow) для заполнения контейнера или сжимает их (пропорционально их коэффициенту flex-shrink), чтобы предотвратить переполнение.

Flex-элемент будет полностью «негибок», если его значения flex-grow и flex-shrink равны нулю, и «гибкий» в противном случае. Свойство является сокращённой записью свойств flex-grow, flex-shrink и flex-basis. Значение по умолчанию: flex: 0 1 auto;. Можно указывать как одно, так и все три значения свойств.

#### Коэффициент роста: flex-grow

Свойство определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства. Если сумма значений flex-grow flex-элементов в строке меньше 1, они занимают менее 100% свободного пространства. Свойство не наследуется.

#### Базовый размер: flex-basis

Свойство устанавливает начальный основной размер flex-элемента до распределения свободного пространства в соответствии с коэффициентами гибкости. Для всех значений, кроме auto и content, базовый гибкий размер определяется так же, как width в горизонтальных режимах записи. Процентные значения определяются относительно размера flex-контейнера, а если размер не задан, используемым значением для flex-basis являются размеры его содержимого.

1. CSS Grid. Особенности, назначение, основные понятия и приемы использования.

[Грид-раскладка (CSS Grid Layout)](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout) представляет собой двумерную систему сеток в CSS. Гриды подойдут и для верстки основных областей страницы, и небольших элементов пользовательского интерфейса. Грид представляет собой пересекающийся набор горизонтальных и вертикальных линий, образующих колонки и строки. Элементы могут быть помещены в грид в пределах линий этих колонок и строк. Грид имеет следующие особенности:

Создание грида с фиксированными размерами полоc, например используя пиксели. Это установит грид на определенный пиксель, соответствующим желаемому макету. Сожно создать грид с гибкими размерами, используя проценты или новую единицу измерения — «fr», разработанную для этой цели.

Размещение элементов в заданном месте на гриде используя номера строк, имена или путём привязки к области грида. Грид также содержит алгоритм управления размещением элементов, не имеющих явной позиции на гриде.

Можно определить явную сетку с помощью грид-раскладки. Cпецификация грид-раскладки достаточно гибкая, чтобы добавить при необходимости дополнительные строки и колонки. Также в нее включены такие возможности как, например, добавление «стольких колонок, сколько будет помещено в контейнер». Грид содержит механизм выравнивания, таким образом мы можем контролировать, как элементы выравниваются после размещения в области сетки и как выравнивается вся сетка.

В ячейку или область грида может быть помещено несколько элементов; эти элементы могут частично перекрывать друг друга. Такое наложение можно контролировать с помощью свойства [z-index](https://developer.mozilla.org/ru/docs/Web/CSS/z-index). Грид – это мощная спецификация, и в сочетании с другими частями CSS, такими как [flexbox](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout), поможет вам создать макеты, которые ранее невозможно было построить в CSS. Все начинается с создания сетки в вашем **грид-контейнере**.

Мы создаём grid контейнер, объявляя на элементе display: grid или display: inline-grid. Как только мы это сделаем, все прямые потомки этого элемента станут элементами сетки.

В этом примере у меня есть контейнер div с классом-обёрткой и пятью дочерними элементами внутри.

<div class="wrapper">

<div>One</div>

</div>

Здесь .wrapper является грид-контейнером.

.wrapper { display: grid;}

Все прямые потомки теперь являются грид-элементами.

Мы определяем ряды и колонки в нашей сетке при помощи свойств [grid-template-columns](https://developer.mozilla.org/ru/docs/Web/CSS/grid-template-columns) и [grid-template-rows](https://developer.mozilla.org/ru/docs/Web/CSS/grid-template-rows). Грид-трек – это промежуток между любыми двумя линиями грида. Сейчас я создала грид с тремя колонками шириной по 200px. Каждый дочерний элемент будет располагаться в отдельной ячейке грида.

.wrapper {

display: grid;

grid-template-columns: 200px 200px 200px;}

Размер треков может быть задан с помощью любой единицы длины. Спецификация также вводит дополнительную единицу длины, позволяющую создавать гибкие (flexible) грид-треки. Новая единица длины «fr» представляет собой долю (fraction) доступного пространства в грид-контейнере. Следующее определение грида создаст три одинаковых по ширине трека, расширяющихся и сужающихся в соответствии с доступным пространством. grid-template-columns: 1fr 1fr 1fr;

В следующем примере мы создаём грид с треком в 2fr и двумя треками по 1fr. Доступное пространство разбивается на четыре части. Две части занимает первый трек, и две части – два оставшихся. grid-template-columns: 2fr 1fr 1fr;

При задании размеров явного грида или при определении размеров автоматически создаваемых колонок или строк нам может понадобиться задать трекам минимальный размер, но при этом быть уверенными, что они треки растянутся, чтобы вместить весь добавленный в них контент.

В гриде есть решение этой задачи – функция [minmax()](https://developer.mozilla.org/ru/docs/Web/CSS/minmax). В следующем примере я использую minmax() в качестве [grid-auto-rows](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-rows" \o "Currently only available in English (US)). То есть автоматически создаваемые строки будут как минимум 100 пикселей в высоту, а как максимум – примут значение auto. Использование auto означает следующее: высота строки «растягивается» до размера ячейки с самым высоким элементом контента.

Нужно заметить, что когда мы определяем грид, мы определяем грид-треки, а не грид-линии. После этого грид обеспечивает нас пронумерованными линиями, для использования при размещении элементов. В нашем гриде с тремя колонками и двумя рядами у нас есть четыре линии колонок.

В следующем примере я разместила первые два элемента на нашем трёхколоночном гриде с помощью свойств [grid-column-start (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column-start), [grid-column-end (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column-end), [grid-row-start](https://developer.mozilla.org/ru/docs/Web/CSS/grid-row-start) и [grid-row-end (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row-end). Работая слева направо, первый элемент размещен начиная с колоночной линии 1 и занимает пространство до колоночной линии 4, которая в нашем случае – самая правая линия грида. Наш элемент начинается со строчной линии 1 и заканчивается на строчной линии 3, таким образом занимая два строчных трека.

Второй элемент начинается с колоночной линии 1 и занимает один трек. Это поведение по умолчанию, поэтому нет необходимости задавать конечную линию. Элемент также занимает два строчных трека – со строчной линии 3 до строчной линии 5. Остальные элементы самостоятельно размещаются в свободном месте на гриде.

.wrapper {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-auto-rows: 100px;

}

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

}

Обычные значения, использованные выше, могут быть умещены в одну строку: для колонок – с использованием [grid-column (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column" \o "Currently only available in English (US)), для строк – с использованием [grid-row (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-row" \o "Currently only available in English (US)). Следующий пример сделает такое же расположение, как и предыдущий, но с менее громоздким кодом CSS. значение до слэша («/») – это первая линия, значение после – последняя линия.

## [Грид-ячейки](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout#%D0%B3%D1%80%D0%B8%D0%B4-%D1%8F%D1%87%D0%B5%D0%B9%D0%BA%D0%B8)

Грид-ячейка – наименьшая часть на гриде. Концептуально она похожа на ячейку таблицы. Как мы видели в предыдущих примерах, едва грид определён для родительского элемента, дочерние элементы автоматически размещаются в каждой ячейке заданного грида. На рисунке ниже я выделил первую ячейку грида.

## [Грид-области](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout#%D0%B3%D1%80%D0%B8%D0%B4-%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D0%B8)

Элементы могут занимать одну или несколько ячеек внутри строки или колонки, таким образом, создаётся грид-область. Грид-области должны быть перпендикулярными – невозможно создать область, например, в форме буквы «L». Выделенная грид-область на рисунке ниже занимает два строчных трека и два колоночных.

1. Bootstrap 4. Особенности, назначение, основные понятия и приемы использования.

Bootstrap — это открытый и бесплатный [HTML-](https://blog.skillfactory.ru/glossary/html/), [CSS-](https://blog.skillfactory.ru/glossary/css/) и [JS-фреймворк](https://blog.skillfactory.ru/glossary/javascript/), который используют веб-разработчики для быстрой верстки адаптивных дизайнов сайтов и веб-приложений. Включает в себя CSS- и HTML-шаблоны оформления для веб-форм, меток, типографики, кнопок, блоков навигации и других компонентов веб-интерфейса.

Фреймворк используют [frontend-](https://skillfactory.ru/frontend-razrabotchik?utm_source=blog&utm_medium=referral&utm_campaign=bootstrap&utm_content=coding_fr&utm_term=text" \t "_blank), [fullstack-разработчики](https://skillfactory.ru/python-fullstack-web-developer?utm_source=blog&utm_medium=referral&utm_campaign=bootstrap&utm_content=coding_fpw&utm_term=text" \t "_blank). Для работы с инструментом нужны минимальные знания верстки, поэтому он подходит новичкам.

Bootstrap используется, когда: у сайта много страниц; страницы собраны из простых базовых элементов — кнопок или таблиц; не будет глобального редизайна; шаблонность страниц окупается скоростью внедрения.

**Основные отличия Bootstrap 4 от Bootstrap 3:** сетка и другие компоненты построены на [флексах](https://itchief.ru/html-and-css/flexbox), а не на float; размеры, указываются в rem и em, а не в px; размер шрифта увеличен с 14px до 16px; обновлены почти все компоненты;

Bootstrap требует использования doctype «HTML5». Без него у вас возникнут некоторые проблемы со стилями.

<!DOCTYPE html> ||| <html lang="en"> ||| ... |||</html>

Bootstrap разрабатывался как mobile first, т.е. его настройки прежде всего оптимизированы под мобильные устройства, а уж потом с помощью медиа-запросов мы подгоняем масштаб компонентов как нам необходимо на прочих устройствах. 

Вы можете посмотреть пример этого в действии на странице [starter template](https://bootstrap-4.ru/docs/4.0/getting-started/introduction/" \l "starter-template).

Для большей простоты масштабирования в CSS мы изменяем глобальное значение box-sizing с content-box на border-box.

Система сеток Bootstrap 4 использует контейнеры, ряды и колонки, чтобы удобно располагать содержимое. Бутстрап реализован с помощью [флексбокса](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes) и полностью адаптивен. Ниже приведен пример и глубокий взгляд на объединение сетки.

<div class="container">

<div class="row">

<div class="col-sm">

Одна из трёх колонок

</div>

<div class="col-sm">

Одна из трёх колонок

</div>

<div class="col-sm">

Одна из трёх колонок

</div>

</div>

</div>

Бутстрап использует em и rem для задания большинства размеров, а пиксели px – для «брейкпойнтов» сетки и ширин контейнеров. Это происходит потому, что ширина зоны видимости на каждом устройстве измеряется в пикселях и не изменяется с [размером шрифта](https://drafts.csswg.org/mediaqueries-3/#units).

## Автоматическое расположение с помощью колонок: Используйте классы колонок со специальными контрольными точками (например, .col-sm-6) для легкого расположения колонок без использования явно обозначенных номеров классов.

### Установка ширины одной колонки: Авто-расположение колонок в сетке флексбокса также дает возможность установить ширину одной колонки, при этом остальные «родственные» колонки автоматически изменят свой размер вокруг нее.

<div class="container">

<div class="row">

<div class="col">

1 из 3

</div>

<div class="col-6">

2 из 3 (широкая)

</div>

<div class="col">

3 из 3

</div>

</div>

</div>

### Содержимое адаптивной ширины: Используйте классы col-{breakpoint}-auto для создания колонок, изменяющих свою ширину по ширине содержимого.

### Мультиряд одинаковой ширины: Создавайте колонки одной ширины, которые охватывают множественные ряды, добавлением .w-100 туда, где необходимо сместить вниз на новую строку. Делайте их адаптивными, применяя .w-100 вместе с некоторыми [адаптивными утилитами отображения](https://bootstrap-4.ru/docs/4.0/utilities/display/).

<div class="row">

<div class="col">col</div>

<div class="w-100"></div>

</div>

### Разрывы колонок: Чтобы сместить колонки на новую строку в гибком контейнере, нужно сделать следующее: добавьте элемент с width: 100% туда, где вы хотите обернуть ваши колонки новой строкой. В норме это достигается с помощью множественных .row, но не каждый исполнительный метод может это поддержать.

<div class="row">

<div class="col-6 col-sm-3">.col-6 .col-sm-3</div>

<div class="col-6 col-sm-3">.col-6 .col-sm-3</div>

<!-- Заставит следующие столбцы переходить на новую строку -->

<div class="w-100"></div>

</div>

## Вложенность: Чтобы вложить ваш контент в сетку по умолчанию, добавьте новый класс .row и набор колонок .col-sm-\* внутри существующей колонки .col-sm-\*. Вложенные ряды должны включать набор колонок, которые добавляется к 12-ти или просто нескольким (не обязательно, чтобы вы использовали все 12 возможных колонок).

<div class="row">

<div class="col-sm-9">

<div class="row">

<div class="col-8 col-sm-6">

</div>

<div class="col-4 col-sm-6">

</div>

</div>

</div>

</div>

1. Использование асинхронных запросов в веб-приложении.

**AJAX** – это аббревиатура от «**A**synchronous **J**avaScript **a**nd **X**ML», которая дословно переводится как «асинхронный JavaScript и XML». AJAX – это термин, который описывает как можно, используя существующие технологии получать данные с сервера в фоновом режиме и использовать их для обновления страницы (без перезагрузки). Основная цель AJAX – это сделать сайты и веб-приложения более удобными, быстрыми и отзывчивыми.

**Основные преимущества использования AJAX: снижение трафика** (из-за уменьшения объёма передаваемых данных между клиентом и сервером); **уменьшение нагрузки на сервер** (не нужно генерировать всю страницу, а только ту часть, которую нужно обновить); **увеличение быстродействия и отзывчивости** (нет необходимости в полной перезагрузки страницы, достаточно обновить содержимое только отдельных блоков); **повышение интерактивности** (с помощью AJAX можно сразу отображать результаты и сделать ресурс более удобным для пользования).

Взаимодействие с сервером через асинхронные запросы осуществляется посредством XHR или метода fetch().

## Создание асинхронных запросов с помощью XHR

асинхронный запрос - такой, который выполняется в фоне и не мешает пользователю взаимодействовать со страницей. А это значит, что при отправке такого запроса, страница не «замораживается», с ней можно продолжать взаимодействовать. XHR – это аббревиатура от объекта XMLHttpRequest, который как раз и позволяет взаимодействовать с сервером через AJAX.

Написание запроса можно разбить на следующие этапы:

**Этап 1.** Создание экземпляра объекта XMLHttpRequest: const xhr = new XMLHttpRequest();

**Этап 2.** Инициализация запроса с помощью метода open(): xhr.open(method, url[, async[, user[, password]]]);

Где: method – метод отправки запроса на сервер (GET, POST, PUT, PATCH, DELETE); url – URL для отправки запроса; async – определяет, как следует выполнять запрос: асинхронно (true – по умолчанию) или нет (false); user и password – имя пользователя и пароль, использующиеся для аутентификации (по умолчанию имеют значение null).

**Этап 3.** Назначение обработчика на событие readystatechange объекта xhr: xhr.onreadystatechange = function() { // ...}

Это событие происходит при изменении статуса запроса readyState. readyState – это свойство, содержащее числовой код, по которому можно определить в какой стадии сейчас находится запрос.

xhr.onreadystatechange = function() {

if (xhr.readyState !== 4) {

return; }

// все в порядке, ответ получен}

Следующее, что нужно проверить – это статус HTTP-ответа. Он находится в свойстве status.

Если запрос был успешно выполнен сервером, то его статус будет 200. Другие ответы нам в большинстве случаев не интересны.

Получить данные (ответ от сервера) можно осуществляется с помощью xhr.response: Например, выведем полученный ответ от севера в консоль:

xhr.onreadystatechange = function() {

if (xhr.readyState !== 4 || xhr.status !== 200) {

return; }

const response = xhr.response;

console.log(response); }

**Этап 4.** Отправка запроса. Выполняется это с помощью метода send().

Если запрос асинхронный, то выполнение send() не останавливает дальнейшее выполнение программы. В противном случае (если запрос синхронный) программа приостанавливается и возобновляет своё выполнение только после получения ответа от сервера.

Запросы GET создаются без тела, все параметры, которые нужно передать серверу, здесь указывается в самом URL. Поэтому чтобы тело запроса было пустое, как уже было отмечено выше, нужно вызвать метод send() без аргументов или с аргументом null.

Отправим запрос: xhr.send();

Асинхронные приложения позволяют клиенту (браузеру) мгновенно реагировать на изменения на сервере, не ставя под угрозу пользовательский опыт и предоставляя пользователю спонтанные изменения представления при изменении состояния динамической системы, без необходимости взаимодействия пользователя с интерфейсом. Таким образом, даже при отсутствии взаимодействия с пользователем сервер \* готов \* к ответу, когда происходит что-то новое. Некоторые приложения зависят от взаимодействия с сервером в режиме реального времени и допускают одновременное выполнение нескольких запросов. Им необходимо постоянно поддерживать связь с сервером, поэтому, когда происходит обновление данных, пользователь сразу же замечается, и каждый клиент обновляет свои данные, не ставя под угрозу информацию других.

1. Управление загрузкой содержимого с помощью Ajax. Объект XMLHttpRequest. Fetch API

Модель Ajax предполагает использование веб-API в качестве прокси для более разумного запроса данных, а не просто для того, чтобы браузер перезагружал всю страницу. Изначально AJAX предполагал асинхронное взаимодействие клиента и сервера посредством данных в формате XML. Хотя сейчас XML во многом вытеснил формат JSON. В любом случае AJAX революционизировал веб-среду, позволив создавать динамичные отзывчивые веб-приложения. Это действительно хорошо, потому что: 1) Обновления страницы намного быстрее, и вам не нужно ждать перезагрузки страницы, а это означает, что сайт работает быстрее и воспринимается более отзывчивым. 2) Меньше данных загружается при каждом обновлении, что означает меньшее потребление пропускной способности.

Чтобы ускорить работу, некоторые сайты также сохраняют необходимые файлы и данные на компьютере пользователя при первом обращении к сайту, а это означает, что при последующих посещениях они используют локальные версии вместо загрузки свежих копий, как при первой загрузке страницы. Содержимое загружается с сервера только при его обновлении.

XMLHttpRequest (который часто сокращается до XHR) является довольно старой технологией сейчас - он был изобретён Microsoft в конце 1990-х годов и уже довольно долго стандартизирован в браузерах.

Внутри элемента [<script>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/script) добавьте следующий код. В нем хранится ссылка на элементы [<select>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/select) и [<pre>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/pre) в переменных и определяется [onchange](https://developer.mozilla.org/ru/docs/Web/API/GlobalEventHandlers/onchange" \o "onchange) обработчика событий, так что, когда значение select изменяется, его значение передаётся вызываемой функции updateDisplay() в качестве параметра.

var verseChoose = document.querySelector('select');

var poemDisplay = document.querySelector('pre');

verseChoose.onchange = function() {

var verse = verseChoose.value;

updateDisplay(verse);};

Определим нашу функцию updateDisplay(). Прежде всего, поставим следующее ниже своего предыдущего блока кода - это пустая оболочка функции: function updateDisplay(verse) {};

Мы начнём нашу функцию с создания относительного URL-адреса, указывающего на текстовый файл, который мы хотим загрузить и который понадобится нам позже. Значение элемента [<select>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/select) в любой момент совпадает с текстом внутри выбранного [<option>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/option).

Добавим следующие строки внутри функции updateDisplay():

verse = verse.replace(" ", "");

verse = verse.toLowerCase();

var url = verse + '.txt';

Чтобы начать создание запроса XHR, вам нужно создать новый объект запроса, используя конструктор [XMLHttpRequest()](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest). Назавем его request (запросом). Добавим следующие ниже строки: var request = new XMLHttpRequest();

Затем нужно использовать метод [open()](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/open" \o "open()), чтобы указать, какой [HTTP request method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) использовать для запроса ресурса из сети и какой его URL-адрес. request.open('GET', url);

Затем зададим тип ожидаемого ответа, который определяется как свойство [responseType](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/responseType" \o "responseType) - как text. request.responseType = 'text';

Получение ресурса из сети - это [asynchronous](https://developer.mozilla.org/ru/docs/Glossary/Asynchronous) операция, означающая, что вам нужно дождаться завершения этой операции, прежде чем вы сможете сделать что-либо с этим ответом, иначе будет выброшена ошибка. XHR позволяет вам обрабатывать это, используя обработчик события onload - он запускается при возникновении события [load](https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event" \o "Currently only available in English (US)). Когда это произойдёт, данные ответа будут доступны в свойстве response объекта запроса XHR.

request.onload = function() { poemDisplay.textContent = request.response;};

Вышеприведённая конфигурация запроса XHR  фактически не будет выполняться до тех пор, пока мы не вызовем метод [send()](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/send" \o "send()). Добавим следующее ниже предыдущего дополнения для вызова функции: request.send();

API-интерфейс Fetch - это, в основном, современная замена XHR - недавно он был представлен в браузерах для упрощения асинхронных HTTP-запросов в JavaScript, как для разработчиков, так и для других API, которые строятся поверх Fetch.

Давайте преобразуем последний пример, чтобы использовать Fetch!

fetch(url).then(function(response) { response.text().then(function(text) { poemDisplay.textContent = text; });});

Прежде всего, мы вызываем метод fetch(), передавая ему URL-адрес ресурса, который мы хотим получить. Это современный эквивалент [request.open()](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/open" \o "request.open()) в XHR, плюс вам не нужен эквивалент .send().

После этого вы можете увидеть метод [.then()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise/then), прикреплённый в конец fetch() - этот метод является частью [Promises](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise) - современная функция JavaScript для выполнения асинхронных операций. fetch() возвращает промис, который разрешает ответ, отправленный обратно с сервера, - мы используем .then() для запуска некоторого последующего кода после того, как промис будет разрешено, что является функцией, которую мы определили внутри неё. Это эквивалент обработчика события onload в XHR-версии. Эта функция автоматически передаёт ответ от сервера в качестве параметра, когда обещает fetch(). Внутри функции мы берём ответ и запускаем его метод [text()](https://developer.mozilla.org/en-US/docs/Web/API/Response/text" \o "Currently only available in English (US)), который в основном возвращает ответ как необработанный текст. Это эквивалент request.responseType = 'text' в версии XHR.

Какой механизм лучше использовать зависит от того, над каким проектом вы работаете. XHR существует уже давно и имеет отличную кросс-браузерную поддержку. Fetch and Promises, с другой стороны, являются более поздним дополнением к веб-платформе, хотя они хорошо поддерживаются в браузере, за исключением Internet Explorer и Safari.

Если необходимо поддерживать старые браузеры, тогда может быть предпочтительным решение XHR. Если, однако, работаем над более прогрессивным проектом и не так обеспокоены старыми браузерами, то Fetch может быть хорошим выбором.

1. Базовые принципы работы веб-приложения в оффлайн режиме.

Всё чаще возникает задача научить frontend-приложение работать в автономном режиме. Это значит придать web-приложению свойство mobile- или desktop-программы — функционировать в отсутствии связи с Интернет, а также в случае отказа сервера.  
Решение задачи заключается в следующем: при первом посещении web-страницы, получить с сервера “статические” ресурсы в виде html-, css-, js-файлов, спрайтов и пр.; закэшировать ресурсы на стороне клиента средствами браузера; в дальнейшем при запросе этих же файлов выдавать их из кэша в том случае, если отсутствует соединение с сервером; обновлять изменённые ресурсы в кэше.

[ServiceWorker](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers) — это новая технология, позволяющая запускать javascript-код в браузере в фоновом режиме — аналог сервисов в операционных системах. ServiceWorker запускается с web-ресурса и продолжает работать в браузере независимо от приложения, которое его инициализировало. Часто цель применения ServiceWorker это получение push-уведомлений в браузере и контроль кэшируемых ресурсов, последнее как раз наш случай.

[CacheStorage](https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage) представляет собой контейнер для хранения кэша сетевых ресурсов. Глобальный объект CacheStorage доступен по имени caches. Его составляющие — это объекты типа [Cache](https://developer.mozilla.org/en-US/docs/Web/API/Cache). Cache — это именованное хранилище из пар: объект Request — объект Response. Для каждого закэшированного ресурса экземпляр Cache будет хранить request и response, созданные функцией [fetch](https://learn.javascript.ru/fetch).  
Существенный плюс — гибкая система кэширования. Что и когда кэшировать — всё в наших руках. Не стоит использовать такую методику для “нестатических” ресурсов, т.е. для данных получаемых от сервера. Лучше это делать не в сервисе, а внутри front-приложения на уровне модели и использовать при этом, например, localStorage. Существует такой нюанс: serviceWorker разрешено загрузить только по HTTPS либо с локального сервера. Неприятное ограничение, но в то же время это правильно. HTTPS становится стандартом для популярных сервисов. Самый главный недостаток. ServiceWorker и CacheStorage — обе технологии экспериментальные на текущее время. И поддержка есть у современных Mozilla, Opera и браузеров от Google. Тема web-приложений в режиме offline существует давно. Возможно, ServiceWorker и CacheStorage лишь временное решение.

1. Работа с базами данных.

Существует три способа работы с данными в БД:

1) Вы создаете пустую структуру базы данных. Пользователь работает с приложением и база данных наполняется.

2) Вы уже имеете готовую БД, наполненную данными, которую нужно распространять с приложением, либо парсите данные из файла в assets.

3) Получать данные из сети, по мере необходимости.

Web-приложения в основном являются приложениями, функционирование которых основывается на активной работе с данными. Взаимодействие web-приложений с базами данных (БД) выполняется, так же как и windows-приложений, с помощью технологии ADO.Net и других технологий, как, например, Linq-to-SQL и Entity Framework. Работа с использованием ADO.Net может выполняться как в режиме с поддержкой соединения, так и в отсоединенном режиме.

Основными средствами взаимодействия web-приложений с данными являются следующие: 1) выполняемые выражения вида <%# [выражение] %> и <%$ [выражение] %>; 2) источники данных и компоненты связывания с источниками данных; 3) связывание серверных ЭУ с источниками данных; 4) шаблоны описания серверных ЭУ с использованием выполняемых выражений <%# Eval(...) %>.

В web-приложении в качестве источников данных могут использоваться разные .NET-классы, и нt только те, которые организуют взаимодействие с БД. В ASP.NET любой объект, который поддерживает интерфейс IEnumerable, может быть связан с серверными ЭУ. Интерфейс Enumerable определяет минимальный API, который необходим для перебора содержания источника данных. Однако многие связываемые объекты в действительности реализуют более совершенные версии IE- numerable, такие интерфейсы, как ICollection и IList. В частности, серверные web-ЭУ могут быть связаны с объектами следующих классов:

* объекты коллекции, – включая словари (Dictionary), хэш-таблицы (Hashtable) и массивы (Array);
* объекты класса DataReader провайдеров ADO.NET;
* объекты контейнерных классов ADO.NET, таких как DataSet, DataTable и DataView;
* любые объекты, которые получаются в результате обработки запроса LINQ, поддерживающие интерфейс IQueryable.

Кроме объектов перечисленных классов, источниками данных могут быть специальные *компоненты связывания с источниками данных* (data source components). Компонент связывания с источником данных (КСИД) – это серверный элемент управления, который предназначен для взаимодействия со связанными с данными элементами управления и для скрытия сложности программного связывания ЭУ с источником данных. Такие компоненты не только предоставляют данные для серверных ЭУ, но они также поддерживают выполнение этими ЭУ других общих операций с данными, такие, как добавление новых данных, удаление, сортировка и обновление.

Любое простое или сложное Web приложение должно хранить информацию, для удобства хранения данных и была придумана система баз данных. Самой распространенной СУБД для Web разработок является Mysql. Web приложение делится на 2 этапа, это разработка пользовательского интерфейса и разработку административной части. Аpi интерфейс Mysql в языке программирования PHP состоит из набора функций выполняющих определенную роль.  
Основные используемые функции для разработки приложения с использованием СУБД MySQL: mysql\_connect - используется для соединения с Mysql. mysql\_select\_db - используется для выбора базы данных с которой в дальнейшем мы будем работать. mysql\_query - выполняет запрос. mysql\_fetch\_row - возвращает массив с полученными данными. mysql\_fetch\_assoc - возвращает ассоциативный массив с полученными данными. mysql\_fetch\_array - возвращает смешанный массив состоящий из предыдущих двух. mysql\_real\_escape\_string - служит для экранирования спец символов, используется на предмет SQLinjection атаки. mysql\_insert\_id - возвращает номер авто инкрементного поля (автоматический счетчик записей). mysql\_close - закрывает Mysql соединение.

Существуют определенные стандарты разработки Web приложение, в данном разделе мы коснемся только стандартом связанных с MySQL и передаваемых ей данных.

1. Технологии создания нативных приложений на веб-стеке.

Нативные мобильные приложения — самый распространенный и дорогой в разработке вид, так как создается отдельно для каждой ОС (iOS, Android или другие). Когда мы говорим о мобильных приложениях, чаще всего имеем в виду именно нативные. Преимущества: Высокая производительность (скорость работы и надежность). Это связано с тем, что приложение создается для каждого ОС с учетом его особенностей; Имеют доступ к устройству пользователя и таким функциям как Bluetooth, списки контактов, камера, NFC и другие; Adjust упоминает, что нативные приложения еще и более безопасны с точки зрения защиты данных пользователя. Недостатки: Стоимость разработки; Нативные приложения занимают место в памяти устройства пользователя, причем с каждым новым обновлением это занятое место может расти.

Веб-приложения представляют собой адаптированные веб-сайты, которые открываются через браузеры. Одним из самых распространенных подвидов считают PWA — прогрессивные веб-приложения, которые, по сути, являются нативными приложениями внутри браузера.

С точки зрения архитектуры веб-приложения состоят из двух частей: клиентской (фронтэнд) и серверной (бэкенд — программно-аппаратная часть сервиса). Выделяют три вида веб-приложений, которые определяют подход к разработке: [**SPA или Single Page Application**](https://www.azoft.ru/blog/spa-mpa-pwa/) — это одностраничное веб-приложение, которое загружается на одну HTML-страницу. Динамическое обновление с использованием JavaScript; **MPA или Multi Page Application** — это многостраничные приложения, где при каждом изменении данных или загрузке новой информации страница обновляется; [**PWA или Progressive Web App**](https://www.azoft.ru/blog/progressive-web-apps/) — это своего рода гибрид сайта с мобильным приложением. Они могут устанавливаться на главный экран смартфона, отправлять push-уведомления и работать в офлайн режиме.

Фронтэнд реализует пользовательский интерфейс веб-приложения и загружается в виде динамических веб-страниц. Основные компоненты стека интерфейсов:

HTML — это стандартный язык разметки, который применяют для создания веб-проектов. Его элементы позволяют отображать стандартные блоки страниц, а также представляют форматированный текст, изображения, таблицы, формы ввода данных и т.д.

Каскадные таблицы стилей (CSS) — это язык разметки, который определяет оформление и макет элементов HTML. Таким образом, HTML задаёт структуру, а CSS — стиль.

JavaScript — это язык программирования, который помогает реализовывать сложное поведение веб-страницы. С помощью React разработчики создают веб-приложения, которые изменяют отображение без перезагрузки страницы. Приложения быстро реагируют на действия пользователя, например, когда они заполняют формы, применяют фильтры, добавляют товары в корзину и тд. Одна из ключевых особенностей React — универсальность. Эту библиотеку можно использовать на сервере и на мобильных платформах с помощью React Native. Angular — это фреймворк от компании Google. Прежде всего он нацелен на разработку SPA-решений. Эта среда разработки известна прежде всего потому, что она предоставляет разработчикам лучшие условия для объединения JavaScript с HTML и CSS. Vue — это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue подходит для постепенного внедрения. Легко интегрируется с другими библиотеками и существующими проектами.

Под серверной частью понимают набор аппаратно-программных средств, с помощью которых реализована логика работы приложения. Задача серверной разработки — сделать так, чтобы ответ от сервера доходил до клиента и спроектированные блоки функционировали нужным образом.

PHP — один из самых распространенных языков веб-разработки. PHP в основном применяется, чтобы «оживить» статичные HTML страницы. Для создания веб-приложений на PHP мы используем фреймворк Laravel. Его особенности: поддерживает функциональное, интеграционное и юнит-тестирование, позволяет легко масштабировать приложения, следует лучшим практикам разработки и предлагает большой выбор шаблонов проектирования.

Java — это универсальный язык программирования, который часто используется для веб-разработки, а также для [разработки под Android](https://www.azoft.ru/technologies/android/). Среди преимуществ веб-приложений на Java — кроссплатформенность, мультифункциональность, надёжность и гибкость.

Node.js — кроссплатформенная среда, которая выполняет код JavaScript вне браузера. Node.js позволяет разработчикам использовать JavaScript, чтобы получить инструменты командной строки. На стороне сервера с его помощью можно запускать сценарии для обработки динамического содержимого веб-страницы, перед тем как она будет доступна в веб-браузере пользователя. Преимущества: Не требуют настроек под операционную систему, разработка быстрая и менее дорогая; Не требуют загрузки и не занимают место на устройстве пользователя; Не требуют обновлений. Недостатки: Веб-приложения зависят от браузера. И функции, которые доступны в одном браузере, могут не поддерживаться в другом; Веб-приложения не работают без подключения к интернету.

Веб-приложения, которые выглядят как нативные и имеют их признаки. Плюсы: более бюджетная и быстрая разработка по сравнению с нативными, Более быстрая загрузка, Возможность взаимодействия с ОС устройства. Минусы: Все равно не такая хорошая производительность, как у нативных приложений.

Adjust выделяет их в отдельную группу, хотя React Native — это название фреймворка для разработки кроссплатформенных приложений. Фреймворк открытый, поэтому и другие разработчики могут использовать его для разработки мобильных приложений.