# 1 Java в десктопных приложениях, как можно реализовывать, разные варианты, преимущества и недостатки

кросс-платформенные возможности Java. Можно создать Java-приложение на Windows, скомпилировать его в байт-код и запустить его на любой другой платформе, поддерживающей виртуальную машину Java (JVM). Таким образом, JVM служит уровнем абстракции между кодом и оборудованием. Если ваша программа не опирается на специфичные для платформы функции и пользовательский интерфейс, ее можно с легкостью перенести: по крайней мере, большую ее часть.

И так основной плюс всех десктопных приложений на java переносимость.

## 1 Консольные приложения

Самый старый, проверенный временем метод написания desktop приложений. В основном используется для написания программ, работающих в фоновом режиме и не требующих взаимодействия с пользователем.

Минусы:

* никакого GUI.

Плюсы:

* высокая производительность.

## 2 Swing

библиотека Swing — это старый, но надежный кросс-платформенный инструмент, интегрированный в различные Java-IDE. Не смотря на то что поддержка Swing компанией Sun Microsystems, прекращена, он по-прежнему является неплохим инструментом разработки java GUI. Аналогом в C# можно назвать Windows Form. Все элементы swing делятся на компоненты (кнопки, поля, чекбоксы, и т. д.) и контейнеры (легковесные(платформо-независимые) и тежеловесные (платформо-зависимые)).

Алгоритм:

1. создаем контейнер;
2. указываем его размеры;
3. задаем завершение работы на закрытие окна;
4. создаем компонент и добавляем его в контейнер;
5. задаем координаты
6. задаем создание экземпляра нашего класса в методе main (все прочее было в конструкторе класса)
7. для контроля действий задаем «ActionListner»

Основные плюсы Swing:

* обязательный MVC. каждые компонент состоит из представления(внешний вид), контроллера(взаимодействие с пользователем) и модели(сведения о состоянии компонента. В то же время, это дает возможность быстро добавить нужную фичу в существущий компонент. Все очень гибко.
* Удобство разработки.
* Более полный набор элементов управления
* Хорошо подходит для создания бизнес-приложений, таких как пользовательские интерфейсы поверх базы данных, где вам не нужны возможности, выходящие за рамки стандартных (и сторонних) элементов управления
* часть библиотеки Java, нет необходимости в дополнительных собственных библиотеках
* работает одинаково на всех платформах
* хорошие онлайн-уроки от Sun / Oracle

Основные минусы Swing:

* Не фиксятся критичные баги.
* Плохая поддержка мультимедиа.
* Создание пользовательских элементов управления или пользовательских скинов очень сложно.
* Синтаксис Java может быть неудобным для создания пользовательских интерфейсов. Средство IDE делает это менее проблемой, но если вам когда-либо нужно что-либо делать вручную, это может стать пыткой.
* Родной внешний вид и поведение могут отличаться от реальной системы.
* Проблемы с многопоточнотся.

В целом Swing удобен в использовании, но не очень красив. Да и настройка расположения объектов не очень

## 3 SWT

SWT(**Standard Widget Toolkit)** библиотека представляет собой кросс-платформенную оболочку для графических библиотек конкретных операционных систем. SWT разработана с использованием стандартного языка Java и получает доступ к специфичным библиотекам различных ОС через JNI (Java Native Interface), который используется для доступа к родным визуальным компонентам операционной системы. SWT использует собственные компоненты, но не подходит для сложного интерфейса.

По сути основное преимущество SWT оноже главный недостаток, SWT не кросплтформенная система а мультикомпилируемая.

Плюсы:

* по возможности использует нативные элементы, поэтому всегда нативное поведение(Элементы конкретной системы)
* может работать быстрее на конкретной системе

Минусы

* требуются собственные библиотеки для каждой поддерживаемой системы
* работает далеко не на всех системах;
* не безопасна

## 4 java FX

JavaFX — лаконичный и современный, но слишком новый, еще не обкатаный.

Как производительность Java Swing, так и производительность Java FX - рекомендуемые варианты в бизнесе. Давайте рассмотрим некоторые ключевые различия между Java Swing и Java FX

1. Swing - это стандартный инструментарий для Java-разработчика по созданию GUI, тогда как JavaFX обеспечивает поддержку платформы для создания настольных приложений.
2. Swing имеет более сложный набор компонентов GUI, тогда как JavaFX имеет приличное количество доступных компонентов UI, но меньше, чем то, что предоставляет Swing.
3. Swing - это унаследованная библиотека, которая полностью поддерживает и предоставляет подключаемые компоненты пользовательского интерфейса, тогда как JavaFX имеет компоненты пользовательского интерфейса, которые все еще развиваются с более продвинутым внешним видом.
4. Swing с момента его появления может создавать компонент пользовательского интерфейса, используя стандартные классы компонентов Java, тогда как Java FX изначально использует декларативный язык, называемый сценарием JavaFX.
5. Swing имеет библиотеку компонентов пользовательского интерфейса и действует как наследие, тогда как JavaFX имеет несколько компонентов, построенных на Swing.
6. В Swing есть поддержка MVC, но она не согласована между компонентами, тогда как поддержка JavaFX очень дружит с MVC. Компоненты и их свойства объявлены в файле FXML, а логика взаимодействия четко выделена в контроллер.

По сути JavaFx является Эволюцией Swing, она сохранила все преимущества и единственным недостатком является его новизна.

Основным нововедением JavaFx яляется введение языка разметки fxml.

Основные плюсы JavaFx:

* Достойная поддержка мультимедиа.
* Легко создавать пользовательские элементы управления, а все существующие элементы управления - с помощью аналога CSS (другие названия параметров и свойств).
* Поддержка анимации хороша - не так хорошо, как флеш, но ее достаточно для базовых анимаций.
* Синтаксис кажется более подходящим для создания пользовательских интерфейсов, особенно для их ручной кодировки - это хорошо, потому что вам нужно будет сдать их код.
* Хорошо подходит для создания мультимедийных приложений или приложений, где вам нужен внешний вид.
* JavaFX очень дружен с шаблоном MVC, и вы можете четко разделить свою работу на: презентацию(FXML, CSS), модели(Java, доменные объекты) и логику (Java).

Основные минусы JavaFx:

* Визуальный дизайнер нуждается в работе, и лично я его не использую. Это может быть большим отрицательным в зависимости от ваших потребностей.
* Все еще не хватает некоторых элементов управления, но это улучшается. Существующие элементы управления, похоже, работают хорошо.

## 5 Electron

Electron это фреймворк для создания нативных приложений с веб-технологиями, такими как: JavaScript, HTML и CSS.

Если вы не знаете Electron, то это по сути веб-браузер (Chromium) в котором работает только ваше веб-приложение… словно настоящая десктопная программа (нет, это не шутка)… это даёт возможность использовать веб-стек и разрабатывать кросс-платформенные десктопные приложения.

Electron включает в себя Chromium(нечто вроде браузера для работы hml5 и react js) и node.js(для сервера соответственно) недавно появилась возможность работать с java как браузер.

Для создания анлогf десктопного приложения достаточно незаметно запустить java сервер одновременно с запуска микробраузера.

# 2 NodeJs и React

JavaScript — однопоточный язык программирования. Это означает, что у него один стек вызовов. Таким образом, в некий момент времени он может выполнять лишь какую-то одну задачу.

Стек вызовов — это структура данных, которая, говоря упрощённо, записывает сведения о месте в программе, где мы находимся. Если мы переходим в функцию, мы помещаем запись о ней в верхнюю часть стека. Когда мы из функции возвращаемся, мы вытаскиваем из стека самый верхний элемент и оказываемся там, откуда вызывали эту функцию. Это — всё, что умеет стек.

Движок JavaScript снова проверит очередь событий на наличие новых событий, только когда стек выполнения будет снова пустым.

## 1 React

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разработан вокруг концепции многоразовых компонентов. Вы определяете небольшие компоненты, и объединяете их, чтобы сформировать более крупные компоненты.

Компонент React — в его простейшей форме — это обычная функция JavaScript. Обьект от React.Component

Во многом компоненты ведут себя как обычные функции JavaScript. Они принимают произвольные входные данные (так называемые «пропсы») и возвращают React-элементы, описывающие, что мы хотим увидеть на экране.

DOM(объектная модель документа)

React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

JavaScript XML (JSX) — это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript. Мы рекомендуем использовать его, когда требуется объяснить React, как должен выглядеть UI. JSX напоминает язык шаблонов, наделённый силой JavaScript. JSX производит «элементы» React.

**Что такое JSX?**

Вместо того, чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, React [разделяет ответственность](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8) с помощью слабо связанных единиц, называемых «компоненты», которые содержат и разметку, и логику

JSX допускает использование любых корректных [JavaScript-выражений](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators) внутри фигурных скобок.

После компиляции каждое JSX-выражение становится обычным вызовом JavaScript-функции, результат которого — объект JavaScript.

Из этого следует, что JSX можно использовать внутри выражений if и циклов for, присваивать переменным, передавать функции в качестве аргумента и возвращать из функции.

для рендеринга React-элемента в корневой узел DOM вызовите [ReactDOM.render()](https://ru.reactjs.org/docs/react-dom.html" \l "render) с React-элементом и корневым DOM узлом в качестве аргументов:

Пока что, мы знаем только один способ обновить интерфейс — это создать новый элемент и передать его в [ReactDOM.render()](https://ru.reactjs.org/docs/react-dom.html" \l "render).

React DOM сравнивает элемент и его дочернее дерево с предыдущей версией и вносит в DOM только минимально необходимые изменения.

**Композиция компонентов**

Компоненты могут ссылаться на другие компоненты в возвращённом ими дереве. Это позволяет нам использовать одну и ту же абстракцию — компоненты — на любом уровне нашего приложения. Неважно, пишем ли мы кнопку, форму или целый экран: все они, как правило, представляют собой компоненты в React-приложениях.

В приложениях, написанных на React с нуля, как правило, есть один компонент App, который находится на самом верху. В случае, если вы переписываете существующее приложение на React, имеет смысл начать работу с маленького компонента типа Button и постепенно двигаться «вверх» по иерархии.

Проще всего объявить React-компонент как функцию:

Ещё компоненты можно определять как [классы ES6](https://developer.mozilla.org/ru-RU/docs/Web/JavaScript/Reference/Classes):

*class Welcome extends React.Component {*

*render() { return <h1>Привет, {this.props.name}</h1>; }}*

С точки зрения React, эти два компонента эквивалентны.

Но элементы могут описывать и наши собственные компоненты:

*const element = <Welcome name="Алиса" />;*

Когда React встречает подобный элемент, он собирает все JSX-атрибуты и дочерние элементы в один объект и передаёт их нашему компоненту. Этот объект и называется «пропсы» (props).

1. Мы передаём React-элемент <Welcome name="Алиса" /> в ReactDOM.render().
2. React вызывает наш компонент Welcome с пропсами {name: 'Алиса'}.
3. Наш компонент Welcome возвращает элемент <h1>Привет, Алиса</h1> в качестве результата.
4. React DOM делает минимальные изменения в DOM, чтобы получилось <h1>Привет, Алиса</h1>.

**Пропсы можно только читать**

Компонент никогда не должен что-то записывать в свои пропсы — вне зависимости от того, функциональный он или классовый.

Важнейшее:

# React компоненты могут иметь внутреннее состояние

Следующее также применимо только к компонентам классам. Упоминал ли я, что некоторые люди называют компоненты только для отображения “глупыми”?

Поле класса state является специальным в любом компоненте классе React. React контролирует состояние каждого компонента для изменений. Но для того, чтобы React сделал это эффективно, мы должны изменить поле состояния с помощью еще одной функции API React, которую нам нужно изучить, это **this.setState**:

Как мы обновляем состояние? Мы передаем объект с новым значением того, что хотим обновить. Обратите внимание, что в обоих вызовах setState мы передаем только одно свойство из поля состояния, а не оба. Это вполне нормально, потому что setState фактически объединяет то, что вы передаете (возвращаемое значение функции), с существующим состоянием. Поэтому, не указывая свойство при вызове setState, мы показываем, что не хотим изменять это свойство (и не удаляем его).

# React всегда наготове

React получил свое название от того, что он реагирует на изменения состояния (хоть и не реактивно, но по графику).

Думайте о том, что входными данными для функции render являются:

* props, которые передаются родителем;
* внутреннее состояние, которое может быть обновлено в любое время;

Когда входные данные функции render меняются, ее результат также может поменяться.

## 2 Node

# 3 Как в веб приложение на Java встроить видеопоток rtsp с ip камеры

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

componentDidMount() {

this.timerID = setInterval(

() => this.tick(),

1000

);

}

componentWillUnmount() {

clearInterval(this.timerID);

}

tick() { this.setState({ date: new Date() }); }

render() {

return (

<div>

<h1>Привет, мир!</h1>

<h2>Сейчас {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

ReactDOM.render(

<Clock />,

document.getElementById('root')

Теперь часы обновляются каждую секунду.

1. Состояние любого смонтированного элемента может измениться. Родитель этого элемента может быть повторно отрисован. Также смонтированный элемент может получить другой набор props. И здесь начинается магия React и наступает именно тот момент, когда React нам так необходим! Честно говоря, до этого он нам особо и не был нужен.
2. История этого компонента не заканчивается, но прежде чем продолжить, нам нужно понять, что же это за состояние, о котором я говорю.

**Пример**

Давайте рассмотрим наше решение и разберём порядок, в котором вызываются методы:

1. Когда мы передаём <Clock /> в ReactDOM.render(), React вызывает конструктор компонента. Clock должен отображать текущее время, поэтому мы задаём начальное состояние this.state объектом с текущим временем.
2. React вызывает метод render() компонента Clock. Таким образом React узнаёт, что отобразить на экране. Далее React обновляет DOM так, чтобы он соответствовал выводу рендера Clock.
3. Как только вывод рендера Clock вставлен в DOM, React вызывает метод жизненного цикла componentDidMount(). Внутри него компонент Clock указывает браузеру установить таймер, который будет вызывать tick() раз в секунду.
4. Таймер вызывает tick() ежесекундно. Внутри tick() мы просим React обновить состояние компонента, вызывая setState() с текущим временем. React реагирует на изменение состояния и снова запускает render(). На этот раз this.state.date в методе render() содержит новое значение, поэтому React заменит DOM. Таким образом компонент Clock каждую секунду обновляет UI.
5. Если компонент Clock когда-либо удалится из DOM, React вызовет метод жизненного цикла componentWillUnmount() и сбросит таймер.

Event как в html @LogoutButton onClick={this.handleLogoutClick}@

React может сгруппировать несколько вызовов setState() в одно обновление для улучшения производительности.

Условный рендеринг в React работает так же, как условные выражения работают в JavaScript. Бывает нужно объяснить React, как состояние влияет на то, какие компоненты требуется скрыть, а какие — отрендерить, и как именно. В таких ситуациях используйте [условный оператор](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80) JavaScript или выражения подобные [if](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/if...else" \t "_blank).Элементы React можно сохранять в переменных. Это может быть удобно, когда какое-то условие определяет, надо ли рендерить одну часть компонента или нет, а другая часть компонента остаётся неизменной.

Метод render возвращает описание того, что вы хотите увидеть на экране. React берёт это описание и отображает результат. Если точнее, render возвращает **React-элемент**, который является легковесным описанием того, что нужно отрендерить. Большинство React-разработчиков используют специальный синтаксис под названием «JSX» для упрощения описания структуры. Во время компиляции синтаксис <div /> преобразовывается в React.createElement('div')

**У каждого компонента React есть история**

1. Сначала мы определяем шаблон для React для создания элементов из компонента.
2. Затем мы указываем React где будем его использовать. Например, внутри вызова функции render другого компонента или с помощью ReactDOM.render.
3. Затем React создает экземпляр элемента и передает ему набор props, к которым мы можем получить доступ с помощью this.props. Эти props — это то, что мы передали на шаге 2.
4. Поскольку это все JavaScript, то будет вызван метод конструктора (если он определен). Это первый метод из тех, что мы называем методами жизненного цикла компонентов.
5. Затем React обрабатывает результат вызова функции render (получает виртуальный узел DOM).
6. Поскольку это первый раз, когда React выполняет рендеринг элемента, React будет взаимодействовать с браузером (от нашего имени, используя DOM API), чтобы отобразить в нем элемент. Этот процесс широко известен как монтирование.
7. Затем React вызывает другой метод жизненного цикла, называемый componentDidMount. Мы можем использовать этот метод, чтобы, например, сделать что-то в DOM, который, как мы знаем, существует в браузере. До этого метода жизненного цикла, DOM, с которым мы работали, был виртуальным.
8. Некоторые истории компонентов заканчиваются здесь. Компоненты демонтируются из DOM браузера по разным причинам. Но перед тем как это произойдет, React вызывает другой метод жизненного цикла, componentWillUnmount.
9. Состояние любого смонтированного элемента может измениться. Родитель этого элемента может быть повторно отрисован. Также смонтированный элемент может получить другой набор props. И здесь начинается магия React и наступает именно тот момент, когда React нам так необходим! Честно говоря, до этого он нам особо и не был нужен.
10. История этого компонента не заканчивается, но прежде чем продолжить, нам нужно понять, что же это за состояние, о котором я говорю.
11. Компоненту может понадобится повторная отрисовка, когда его состояние будет обновлено или когда его родитель решает изменить props, которые он передал компоненту.
12. Если происходит последнее, React вызывает другой метод жизненного цикла, componentWillReceiveProps.
13. Если объект состояния или переданные props изменены, React должен принять важное решение. Должен ли компонент обновляться в DOM? Вот почему он вызывает другой важный метод жизненного цикла, shouldComponentUpdate. Этот метод по сути является вопросом, поэтому, если вам нужно самостоятельно настроить либо оптимизировать процесс отрисовки, вы должны ответить на этот вопрос, вернув либо true, либо false.
14. Если метод shouldComponentUpdate не объявлен, React по умолчанию делает очень умную вещь, которая, на самом деле, достаточно хороша в большинстве ситуаций.
15. Во-первых, React вызывает другой метод жизненного цикла, componentWillUpdate. Затем React рассчитает новое отображение компонента и сравнивает его с последним.
16. Если результат точно такой же, React ничего не делает (нет необходимости разговаривать с браузером).
17. Если есть разница, React переносит эту разницу в браузер, как мы видели раньше.
18. В любом случае, поскольку процесс обновления произошел (даже если результат рендеринга был точно таким же), React вызывает последний метод жизненного цикла, componentDidUpdate.

## 2 Node

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера

В наши дни платформа Node.js является одной из самых популярных платформ для построения эффективных и масштабируемых REST API’s.

Веб-приложения, написанные следуя клиент/серверной архитектуре, работают по следующей схеме — клиент запрашивает нужный ресурс у сервера и сервер отправляет ресурс в ответ. В этой схеме сервер, ответив на запрос, прерывает соединение.

Такая модель эффективна поскольку каждый запрос к серверу потребляет ресурсы (память, процессорное время и т.д.). Для того чтобы обрабатывать каждый последующий запрос от клиента, сервер должен завершить обработку предыдущего.

Значит ли это, что сервер может обрабатывать только один запрос за раз? Не совсем! Когда сервер получает новый запрос он создаёт отдельный **поток** для его обработки.

Для обработки N запросов серверу нужно N потоков. Если сервер получает N+1 запросов, тогда он должен ждать пока один из потоков не станет доступным.

Node.js использует **событийно-ориентированную** модель и **неблокирующую ввод / вывод**архитектуру, что делает его легковесным и эффективным. Это не фреймворк, и не библиотека, это среда выполнения JavaScript.

Node.js использует неблокирующие ввод/вывод операции, что же это значит:

Главный поток не будет блокироваться операциями ввода/вывода.

Сервер будет продолжать обслуживать запросы.

Нам придётся работать с **асинхронным кодом**.

**колбэки**. Эти функции будут выполнены в какой-то из моментов в будущем (Первая, как только сервер получит запрос, а вторая — когда файл будет прочитан с диска и помещён в буфер).

Пока файл считывается с диска, Node.js может обрабатывать другие запросы и даже считывать файл снова и всё это в одном потоке… но как?!

**Цикл событий** (Event Loop)— это магия, которая происходит внутри Node.js. Это буквально бесконечный цикл и на самом деле один поток.

Нет ни одного способа выполнить код параллельно в рамках обработки запроса. Однако, для того, чтобы избежать блокировки сервера, все операции ввода-вывода асинхронны, и используют события для взаимодействия с сервером:

Цикл событий имеет 6 фаз, каждое исполнение всех 6 фаз называют **tick**-ом.

* **timers**: в этой фазе выполняются коллбэки, запланированные методами setTimeout() и setInterval();
* **pending callbacks**: выполняются почти все коллбэки, за исключением событий close, таймеров и setImmediate();
* **idle, prepare**: используется только для внутренних целей;
* **poll**: ответственен за получение новых событий ввода/вывода. Node.js может блокироваться на этом этапе;
* **check**: коллбэки, вызванные методом setImmediate(), выполняються на этом этапе;
* **close callbacks**: например, socket.on('close', ...);

Хорошо, есть только один поток, и этот поток и есть цикл событий, но тогда кто выполняет все операции ввода/вывода?

# 3 Как в веб приложение на Java встроить видеопоток rtsp с ip камеры

Технология RTSP представляет собой специализированный протокол, реализующий удаленное управление оборудованием и мультимедийным потоком, генерирование запросов сервером и клиентом. Команды протокола RTSP позволяют подключать удаленные IP-камеры к системам сигнализации, запускать, останавливать, перенаправлять, переводить в режим паузы поток данных. Команды протокола устанавливают доступ к файлам на удаленном хранилище, параметры трансляции и пересылки видео на локальный плеер, компьютер или смартфон. Технология, разработанная в 1998 г., описана в RFC 2326.

Браузеры не работают с протоколами UDP и RTSP, однако поддерживают WebRTC. IP-камеры, напротив, могут пользоваться UDP и RTSP, но не «понимают» стек протоколов WebRTC. Рассогласование возможностей устраняется промежуточным сервером, выполняющим роль моста между источником потокового видео и браузером.

**Связь камеры с сервером**

Методы протокола RTSP:

* describe — запрос описания содержимого, например, в формате [SDP](https://ru.wikipedia.org/wiki/SDP);
* options — запрос поддерживаемых методов;
* play — запрос начала вещания содержимого;
* pause — запрос временной остановки вещания;
* record — запрос на записывание содержимого сервером;
* redirect — перенаправление на другое содержимое;
* setup — запрос установки транспортного механизма для содержимого;
* announce — обновление данных описания содержимого;
* get\_parameter — запрос указанных параметров у сервера;
* set\_parameter — установка параметров сервера;
* teardown — остановка потока и освобождение ресурсов.

Камера может использовать разные форматы. Rtsp – это просто протокол.

H264

ffmpeg -i rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny\_115k.mov -acodec copy -vcodec copy -t 00:05:00 e:/var/Qwer12.avi

<rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny_115k.mov>

**Связь сервера с браузером**

WebRTC

* Не требуется установка ПО.
* Высокое качество связи благодаря:
  + использованию современных [видео-](https://trueconf.ru/webrtc.html#videocodecs) и [аудиокодеков](https://trueconf.ru/webrtc.html#audiocodecs);
  + автоматическому подстраиванию качества потока под условия соединения;
  + встроенной системе эхо- и шумоподавления;
  + автоматической регулировке уровня чувствительности микрофонов участников (АРУ).
* Высокий уровень безопасности: все соединения защищены и зашифрованы согласно протоколам DTLS и SRTP. При этом WebRTC работает только по протоколу HTTPS, а использующий технологию сайт должен быть подписан сертификатом.
* Поддержка технологии SVC добавлена как часть реализации кодеков VP9 и AV1. Несмотря на то, что на текущий момент всё ещё нет реализации в самих браузерах, программные решения TrueConf позволяют использование SVC в браузерных клиентах.
* Есть встроенный механизм захвата контента, например, рабочего стола.
* Возможность реализации любого интерфейса управления на основе HTML5 и JavaScript.
* Проект с открытым исходным кодом — можно внедрить в свой продукт или сервис.
* Настоящая кросс-платформенность: одно и то же WebRTC приложение будет одинаково хорошо работать на любой операционной системе, десктопной или мобильной, при условии, что браузер поддерживает WebRTC. Это значительно экономит ресурсы на разработку ПО.

Hls