# Подробнее о компонентах

**№ урока:** 2 **Курс:** Подробнее о компонентах

**Средства обучения:** Текстовый редактор или IDE, браузер

### Обзор, цель и назначение урока

В данном уроке мы узнаем еще больше о компонентах, изучим их жизненный цикл, разберем все методы жизненного цикла и варианты их использования, научимся использовать refs, работать с дочерними компонентами, взаимодействовать с DOM элементами, узнаем больше о событиях и научимся использовать Inline стили в React. Во второй половине урока мы напишем маленький Google Keep :)

### Изучив материал данного занятия, учащийся сможет:

- Понимать и использовать методы жизненного цикла компонента
- Обращаться к дочерним компонентам
- Строить более сложные приложения
- Работать с localStorage
- Работать с DOM и использовать библиотеки, которые подразумевают обработку DOM элементов

### Содержание урока

1. Жизненный цикл компонента
2. Порядок вызова методов жизненного цикла при инициализации компонента
3. Порядок вызова методов жизненного цикла при изменении параметров
4. Порядок вызова методов жизненного цикла при изменении состояния
5. Порядок вызова методов жизненного цикла при удалении компонента
6. Пояснение методов, когда какие следует использовать
7. Пример. Демонстрация порядка вызовов методов жизненного цикла
8. Пример. Таймер
9. Деление приложение для работы с заметками на компоненты, продумывание структуры
10. Установка nodejs и npm
11. Установка и запуск http сервера
12. Пример. Приложение для управления заметками
13. Inline стили в React
14. Использование this.props.children
15. Refs
16. Использование localStorage для хранения заметок пользователя
17. Как работают события в React – делегирование

### Резюме

## Жизненный цикл компонентов в React

В React есть относительно немного методов жизненного цикла, но все они очень мощные. React дает вам все необходимые методы для контроля свойств и состояния приложения в процессе его жизни.

Есть всего 4 сценария, когда методы жизненного цикла могут быть использованы:

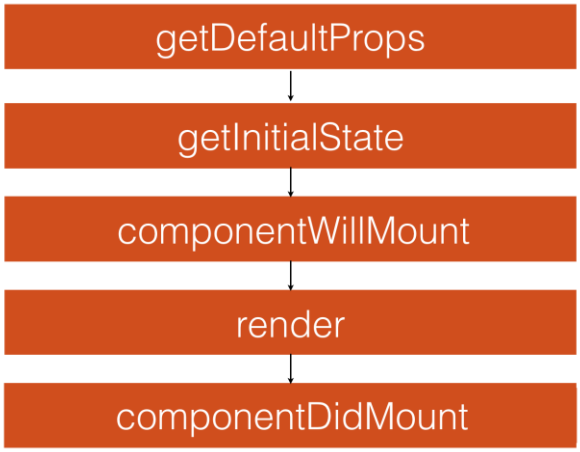
- Инициализация компонента
- Изменение его параметров (props)
- Изменение его состояния (вызов setState)
- Удаление компонента



CyberBionic systematics
CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

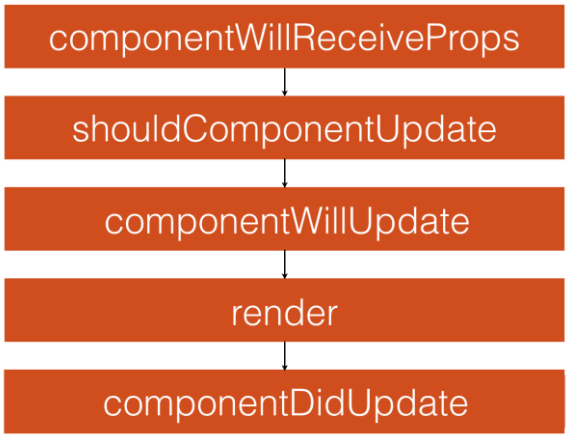
**Инициализация компонента (первый render)**

При первом render компонента методы жизненного цикла будут вызваны в таком порядке:



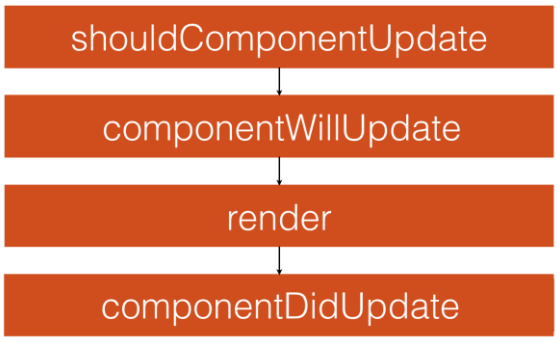

**Изменение параметров (props)**

Когда от родительского компонента приходят измененные параметры, последовательность вызова методов жизненного цикла такая:




**Изменение состояния (вызов setState)**

Когда в компоненте изменяется состояние, то методы жизненного цикла вызываются в таком порядке:




**Удаление компонента**

Перед удаление компонента из DOM будет вызван один единственный метод componentDidMount



CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

componentWillUnmount

## Методы жизненного цикла

### getDefaultProps

Вызывается единожды при инициализации класса. Отвечает за значения параметров по умолчанию.

```
getDefaultProps: function() {
    return {
        name: '',
        age: 0
    };
}
```

### getInitialState

Создан для определения начального состояния компонента.

```
getInitialState: function() {
    return {
        isOpened: true
    };
}
```

### componentWillMount

Вызывается один раз прямо перед тем, как состоится первый render компонента. Вызов setState в рамках данного метода дополнительного рендера не вызовет.

```
componentWillMount: function() {
    // ...
}
```

### componentDidMount

Вызывается один раз прямо сразу после того, как состоялся первый render компонента.

```
componentDidMount: function() {
    // компонент уже находится в DOM
    // здесь можно уже взаимодействовать с DOM напрямую
    // например, использовать jQuery или какие-то сторонние библиотеки
}
```

### componentWillReceiveProps

Вызывается каждый раз, когда компонент получает новые параметры. Не вызывается для первого рендера. Вызов setState в рамках данного метода дополнительного рендера не вызовет.

```
componentWillReceiveProps: function(nextProps) {
    // в nextProps содержится объект с новыми параметрами
    // старые параметры можно получить использование this.props
    this.setState({
        likesIncreasing: nextProps.likeCount > this.props.likeCount
    });
}
```



CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

**shouldComponentUpdate**

Вызывается при изменении параметров или состояния. Возвращает true (если изменение должно вызвать перерисовку компонента) или false (если изменение не влияет на отображение компонента).

```
shouldComponentUpdate: function(nextProps, nextState) {
    return nextProps.id !== this.props.id;
}
```

Если shouldComponentUpdate возвращает false, то метод render() будет пропущен до следующего изменения параметров или состояния. По умолчанию (если не определен), всегда возвращает true. Может быть использован для улучшения быстродействия приложения (чтобы избежать лишних перерисовок), особенно, если используется огромное количество компонентов.

**componentWillUpdate**

Вызывается перед вызовом метода render() при изменении параметров или состояния компонента.

```
componentWillUpdate: function(nextProps, nextState) {
    // в nextProps содержится объект с новыми параметрами
    // в nextState содержится объект с измененным состоянием
}
```

!!! Не используйте setState() в этом методе! Так у вас может произойти зацикливание!

**componentDidUpdate**

Вызывается сразу после вызова метода render() при изменении параметров или состояния компонента.

```
componentDidUpdate: function(prevProps, prevState) {
    // в prevProps содержится объект с предыдущими параметрами
    // в prevState содержится объект с состоянием до изменения
    // измененные параметры и состояние могут быть получены через this.props и
this.state
}
```

Произведенные изменения уже отображены в DOM дереве. Обычно, в данном методе производят какие-то операции с DOM элементами согласно изменениям.

!!! Не используйте setState() в этом методе! Так у вас может произойти зацикливание!

**componentWillUnmount**

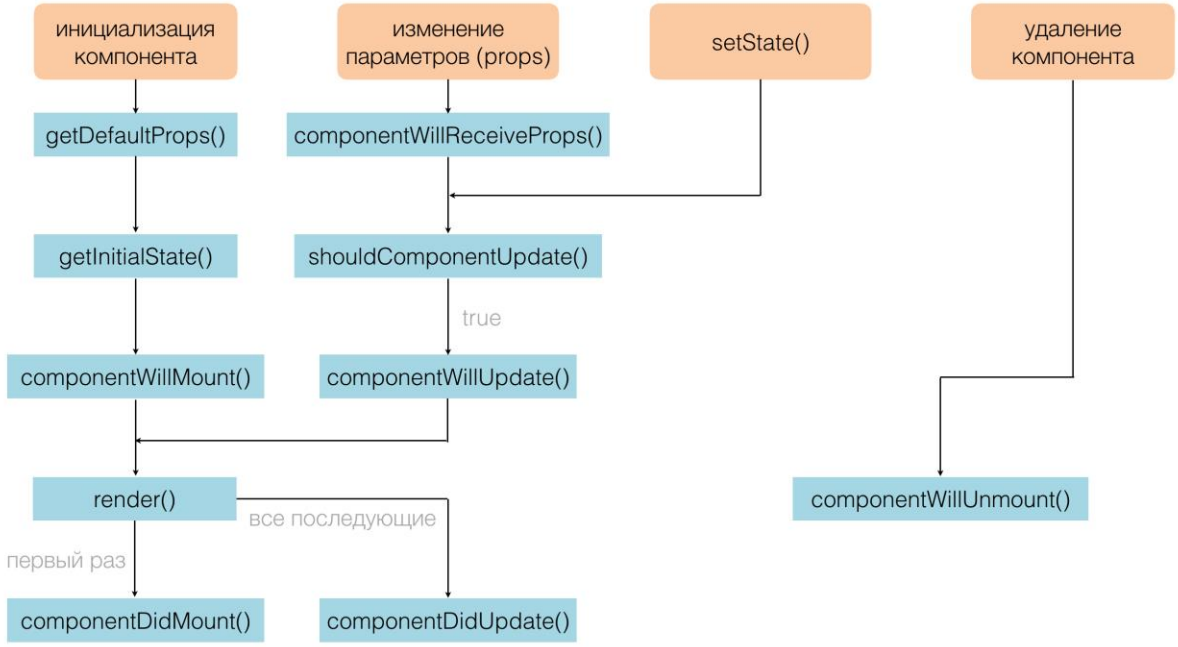
Вызывается перед тем, как компонент будет удален из DOM.

```
componentWillUnmount: function() {
    // обычно, в данном методе происходит некая уборка за компонентом
    // остановка таймеров, удаление ссылок на DOM элементы и т.д.
}
```

Весь жизненный цикл компонента можно представить в виде такой схемы



CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com




## Дочерние компоненты

Иногда вместо того, чтобы писать так:

```
<Article author="Vasya Pupkin" text="Here is article itself" />
```

Очень хочется написать так:

```
<Article author="Vasya Pupkin"> Here is article itself </Article>
```

Для таких случаев существуют this.props.children. В компоненте Article можно просто обратиться к ним для получения всего между открывающимся и закрывающимся тегами.

```
var Acricle = React.createClass({
    render: function() {
        return (
            <div>
                <p>{this.props.children}</p>
                by {this.props.author}
            </div>
        );
    }
});
```

## Inline стили

```
// для удобства работы со стилями они представлены в виде объекта,
// а не строки, объект со стилями передается в атрибут style
var divStyle = {
    color: 'white',
    height: 50, // можно не писать 'px', а использовать числа
    backgroundImage: 'url(' + imgUrl + ')', // вместо background-image
    WebkitTransition: 'all',// все префиксы с заглавной буквы
    msTransition: 'all' // кроме ms-*
};

ReactDOM.render(
```

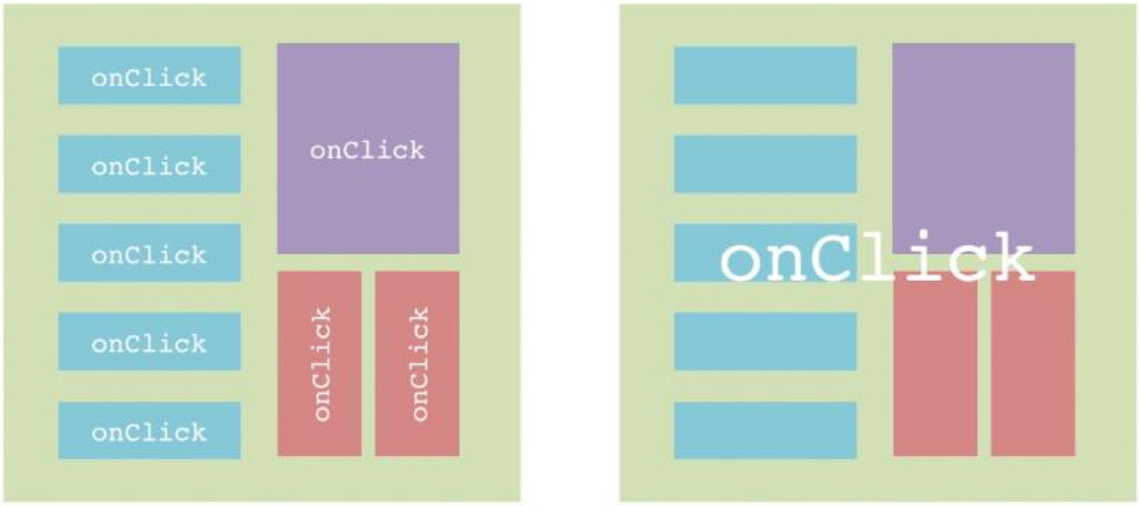


CyberBionic systematics
CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

```
    <div style={divStyle}>Hello World!</div>,
    mountNode
);
```

## Как работают события в React

Если на странице есть 100 (или 1000) различных элементов, на которых вы назначили обработчики события onClick – React вместо того, чтобы назначать обработчик каждому, поставит только один обработчик на их общего предка, а там уже, с помощью всплытия, поймет на каком именно потомке произошло событие и обработает его. Такой подход упрощает инициализацию и экономит память.




### Закрепление материала

1. В каком порядке и какие вызываются методы жизненного цикла при первом render'е компонента, при изменении параметров, при изменении состояния, при удалении компонента?
2. Вызовет ли дополнительный `render` использование `setState()` в методе `componentWillReceiveProps`?
3. Как влияет метод shouldComponentUpdate на render?
4. Какой из методов жизненного цикла следует использовать для снятия события с window?
5. Как обратиться к тому, что лежит между тегами `<Component> Some more JSX here </Component>` из компонента `Component`?
6. Зачем нужны `refs` и как к ним обращаться?
7. Чем отличается определение inline стилей для компонента в React от их использования в html? Как передать inline стили в компонент?
8. Зачем для событий используется `bind`? Когда его следует использовать?
9. Если вы на 1000 компонентов повесите 1000 событий, значит ли это, что будет в действительности объявлено 1000 событий? Почему?

### Самостоятельная деятельность учащегося

**Задание 1: Сделать выбор цвета для заметки**
Уровень сложности: низкий
При создании новой заметки реализовать выбор ее цвета. Можно выбирать из всего спектра или использовать только 5-7 ваших любимых цветов (как в Google Keep).

**Задание 2: Таймер**
Уровень сложности: средний
Нужно усовершенствовать написанный таймер добавлением в него кнопок "Пауза", "Старт" и "Возобновить".



CyberBionic systematics
CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

**Задание 3: Теги для заметок**
Уровень сложности: выше среднего
При создании заметки пользователь может указать список тегов для нее. Затем, при нажатии на тег, должны отфильтровываться заметки с данным тегом.

**Задание 4: To-do list**
Уровень сложности: высокий
Написать список задач. Пользователь может добавить новую задачу, отметить ее как выполненную (просто вычеркнуть ее из списка) и редактировать существующие задачи. Задачи должны сохраняться в localStorage.

## Рекомендуемые ресурсы

**Все примеры из урока и пояснения к домашним заданиям можно найти тут:**
https://github.com/krambertech/react-essential-course

**Ссылки:**

Сайт nodejs - https://nodejs.org

NPM - https://www.npmjs.com

Документация по NPM - https://docs.npmjs.com/

Документация http-server - https://www.npmjs.com/package/http-server

Библиотека Masonry - http://masonry.desandro.com/

**Что почитать:**

Официальная документация React - http://facebook.github.io/react/docs

Советую ознакомиться с JavaScript Style Guide - https://github.com/airbnb/javascript



CyberBionic systematics
CyberBionic Systematics ® 2016
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com