

به نام خدا

مجموعه سوالات سوال و جواب‌های ریاكت React.js

نویسنده: Sudheer Jonna
مترجم: جعفر رضایی و تیم ماریوتک

مجموعه سوال و جواب‌های ری‌اکت

اگه خوشتون اومد به گیت‌ها بمون مراجعه کنین و بهمون ☆ بدین. اگر هم قصد مشارکت داشتید خیلی خوشحال می‌شیم:

دانلود کتاب به فرمت‌های PDF/Epub

می‌تونین خیلی راحت از نسخه آنلاین استفاده کنین یا اگه به فایل کتاب می‌خوایین دسترسی داشته باشین، از بخش ریلیزهای گیت‌هاب ([این لینک](#)) به فرمت‌های مختلف دانلود کنین.

فهرست

ردیف	سوال
	هسته ری‌اکت
1	ری‌اکت چیه؟
2	اصلی‌ترین ویژگی‌های ری‌اکت کدوما هستن؟
3	JSX چیه؟
4	تفاوت‌های Element و Component چیه؟
5	تو ری‌اکت چطوری کامپوننت می‌سازیم؟
6	چه موقع‌هایی باید از Class Component بجای Function Component استفاده کنیم؟
7	Pure Components چیه؟
8	state تو ری‌اکت چیکار می‌کنه؟

ردیف	سوال
9	props تو ری اکت چیکار می‌کنه؟
10	تفاوت state و props چیه؟
11	چرا نباید state رو مستقیماً آپدیت کنیم؟
12	هدف از متدهای callback موقع استفاده از setState چیه؟
13	تفاوت بین نحوه مدیریت رویداد HTML و React چیه؟
14	چطوری متد یا event رو به تابع callback توی JSX bind کنیم؟
15	چطوری میشه یک مقدار رو به یه تابع callback یا eventHandler پاس بدیم؟
16	Synthetic events (رویدادهای مصنوعی) تو ری اکت کدوما هستن؟
17	عبارات شرطی درون خطی چیه؟
18	پارامترهای key چیکار می‌کنن و مزایای استفاده از اونا توی حلقه‌ها چیه؟
19	کاربرد ref‌ها چیه؟
20	چطوری از ref استفاده کنیم؟
21	forward ref چیه؟
22	بین callback refs و تابع findDOMNode کدوم رو ترجیح میدی؟
23	چرا Ref‌های متنی منقضی محسوب می‌شوند؟
24	Virtual DOM چیه؟
25	Virtual DOM چطوری کار می‌کنه؟
26	تفاوت بین Shadow DOM و Virtual DOM چیه؟
27	React Fiber چیه؟
28	هدف اصلی React Fiber چیه؟
29	کامپوننت‌های کنترل شده چی هستن؟

ردیف	سوال
30	کامپوننت‌های کنترل نشده چی هستن؟
31	تفاوت‌های بین createElement و cloneElement کدوما هستن؟
32	مفهوم lift state up یا مدیریت state در لول بالاتر رو توضیح میدی؟
33	فازهای مختلف از lifecycle کامپوننت کدوما هستن؟
34	متدهای lifecycle کامپوننت کدوما هستن؟
35	کامپوننت‌های Higher-Order چی هستن؟
36	چطوری می‌تونیم props proxy برای کامپوننت‌های HOC ایجاد کنیم؟
37	context چیه؟
38	children prop چیه؟
39	چطوری میشه تو React کامنت نوشت؟
40	چرا توی کامپوننت‌های کلاس باید توی constructor تابع super رو با مقدار props صدا بزنیم؟
41	reconciliation چیه؟
42	چطوری با یه اسم داینامیک set state کنیم؟
43	یه اشتباه رایج برای مدیریت توابع eventها که باعث میشه با هر رندر توابع مجدد ساخته بشن چی هستش؟
44	تابع lazy load که برای lazy استفاده میشه رو می‌تونیم به صورت name export خروجی بگیریم؟
45	چرا ری‌اکت از className بجای class استفاده می‌کنه؟
46	fragmentها چی هستن؟
47	چرا fragmentها از تگ‌های div بهترن؟
48	توی ری‌اکت portalها چیکار می‌کنن؟
49	کامپوننت stateless چیه؟

ردیف	سوال
50	کامپوننت stateful چیه؟
51	چطوری prop های کامپوننت رو اعتبارسنجی کنیم؟
52	مزایای React چیه؟
53	محدودیت های React چیه؟
54	error boundary ها توی ری اکت نسخه 16 چیکار می کنن؟
55	چطوری از error boundary ها توی نسخه 15 ری اکت استفاده کنیم؟
56	روش های پیشنهادی برای type checking چیه؟
57	کاربرد پکیج react-dom چیه؟
58	کاربرد متد render از پکیج react-dom چیه؟
59	ReactDOMServer چیه؟
60	چطوری از InnerHtml توی ری اکت استفاده کنیم؟
61	چطوری توی ری اکت استایل دهی می کنیم؟
62	تفاوت event های ری اکت چیه؟
63	اگه توی constructor بیاییم و setState کنیم چی میشه؟
64	تاثیر استفاده از ایندکس به عنوان key چیه؟
65	نظرت راجع به استفاده از setState توی متد componentWillMount چیه؟
66	اگه از prop توی مقداردهی اولیه state استفاده کنیم چی میشه؟
67	چطوری کامپوننت رو با بررسی یه شرط رندر می کنیم؟
68	چرا وقتی prop ها رو روی یه DOM Element می آیم spread می کنیم باید مراقب باشیم؟
69	چطوری از decorator ها توی ری اکت استفاده کنیم؟
70	چطوری یه کامپوننت رو memoize می کنیم؟

ردیف	سوال
71	چطوری باید Server-Side Rendering یا SSR رو توی ری اکت پیاده کنیم؟
72	چطوری حالت production رو برای ری اکت فعال کنیم؟
73	CRA چیه و چه مزایایی داره؟
74	ترتیب اجرا شدن متدهای life cycle چطوره؟
75	کدوم متدهای life cycle توی نسخه 16 ری اکت منسوخ شدن؟
76	کاربرد متد getDerivedStateFromProps چیه؟
77	کاربرد متد getSnapshotBeforeUpdate چیه؟
78	آیا هوک‌ها جای render props و HOC رو می‌گیرن؟
79	روش توصیه شده برای نام‌گذاری کامپوننت‌ها چیه؟
80	روش توصیه شده برای ترتیب متدها در کلاس کامپوننت‌ها چیه؟
81	کامپوننت تعویض کننده یا switching چیه؟
82	چرا نیاز میشه به تابع setState یه فانکشن callback پاس بدیم؟
83	حالت strict توی ری اکت چیکار می‌کنه؟
84	Mixin‌های ری اکت چی هستن؟
85	چرا isMounted آنتی پترن هست و روش بهتر انجامش چیه؟
86	پشتیبانی ری اکت از pointer event‌ها چطوره؟
87	چرا باید اسم کامپوننت با حرف بزرگ شروع بشه؟
88	آیا prop‌های custom توی ری اکت پشتیبانی میشن؟
89	تفاوت‌های constructor و getInitialState چیه؟
90	می‌تونیم یه کامپوننت رو بدون setState ری‌رندر کنیم؟
91	تفاوت‌های فراخوانی super(-) و super(props) توی کلاس کامپوننت‌های ری اکت چیه؟

ردیف	سوال
92	چطوری توی JSX حلقه یا همون لوپ رو داشته باشیم؟
93	توی attributeها چطوری به prop دسترسی داشته باشیم؟
94	چطوری یه PropTypes برای آرایه‌ای از objectها با shape داشته باشیم؟
95	چطوری classهای یه المنت رو به صورت شرطی رندر کنیم؟
96	تفاوت‌های React و ReactDOM چیه؟
97	چرا ReactDOM رو از React جدا کردن؟
98	چطوری از label تو ری‌اکت استفاده کنیم؟
99	چطوری می‌تونیم چندتا object از استایل‌های درون خطی رو با هم ترکیب کنیم؟
100	چطوری با resize شدن مرورگر یه ویو رو ری‌رندر کنیم؟
101	تفاوت متدهای setState و replaceState چیه؟
102	چطوری به تغییرات state گوش بدیم؟
103	روش توصیه شده برای حذف یک عنصر از آرایه توی state چیه؟
104	امکانش هست که ری‌اکت رو بدون رندر کردن HTML استفاده کنیم؟
105	چطوری میشه با ری‌اکت یه JSON به شکل beautify شده نشون داد؟
106	چرا نمی‌تونیم prop رو آپدیت کنیم؟
107	چطوری می‌تونیم موقع لود صفحه روی یه input فوکوس کنیم؟
108	روش‌های ممکن برای آپدیت کردن object توی state کدوما هستن؟
109	چرا توابع به جای object در setState ترجیح داده می‌شوند؟
110	چطوری می‌تونیم نسخه ری‌اکت جاری رو توی محیط اجرایی بفهمیم؟
111	روش‌های لود کردن polyfill توی CRA کدوما هستن؟
112	توی CRA چطوری از https به جای http استفاده کنیم؟

ردیف	سوال
113	توی CRA چطوری میشه از مسیرهای طولانی برای ایمپورت جلوگیری کرد؟
114	چطوری میشه Google Analytics رو به react-router اضافه کرد؟
115	چطوری یه کامپوننت رو هر ثانیه به روز کنیم؟
116	برای استایل‌دهی‌های درون خطی چطوری باید پیشوندهای مخصوص مرورگرها رو اضافه کرد؟
117	چطوری کامپوننت‌های ری‌اکت رو با es6 می‌تونیم import و export کنیم؟
118	استثنایی که برای نام‌گذاری کامپوننت اجازه استفاده از حرف کوچک رو میده چیه؟
119	چرا تابع سازنده کلاس کامپوننت یکبار صدا زده میشه؟
120	توی ری‌اکت چطوری مقدار ثابت تعریف کنیم؟
121	چطوری توی برنامه event کلیک شدن رو trigger کنیم؟
122	آیا استفاده از async/await توی ری‌اکت ممکنه؟
123	ساختار پوشه‌بندی معروف برا ری‌اکت چطوره؟
124	پکیج‌های مشهور برای انیمیشن کدوما هستن؟
125	مزایای ماژول‌های style چیه؟
126	معروف‌ترین lint‌های ری‌اکت کدوما هستن؟
127	چطوری باید توی کامپوننت درخواست api call بزنیم؟
128	render props چیه؟
	رووتر ری‌اکت
129	React Router چیه؟
130	ارتباط React Router و کتابخونه history چیه؟
131	کامپوننت‌های router توی نسخه ۴ کدوما هستن؟
132	هدف از متدهای push و replace توی history چیه؟

ردیف	سوال
133	چطوری توی برنامه به route خاص جابجا بشیم؟
134	چطوری میشه query پارامترها رو توی ری اکت روتر نسخه ۴ گرفت؟
135	دلیل خطای "Router may have only one child element" چیه؟
136	چطوری میشه به متد history.push پارامتر اضافه کرد؟
137	چطوری میشه صفحه ۴۰۴ ساخت؟
138	توی ری اکت روتر نسخه ۴ چطوری میشه history رو گرفت؟
139	چطوری بعد از لاگین به شکل خودکار ریدایرکت کنیم؟
	چند زبانی در ری اکت
140	React-Intl چیه؟
141	اصلی ترین ویژگی های React Intl کدوما هستن؟
142	دو روش فرمت کردن توی React Intl کدوما هستن؟
143	چطوری از FormattedMessage به عنوان یه placeholder میشه استفاده کرد؟
144	چطوری میشه locale فعلی رو توی React Intl بدست آورد؟
145	چطوری با استفاده از React Intl یه تاریخ رو فرمت بندی کنیم؟
	تست کردن ری اکت
146	توی تست ری اکت Shallow Renderer چیه؟
147	پکیج TestRenderer توی ری اکت چیه؟
148	هدف از پکیج ReactTestUtils چیه؟
149	Jest چیه؟
150	مزایای jest نسبت به jasmine کدوما هستن؟
151	یه مثال ساده از تست با jest بزن؟

ردیف	سوال
	React Redux
152	Flux چیه؟
153	Redux چیه؟
154	مبانی اصلی ریداکس کدوما هستن؟
155	کاستی‌های redux نسبت به flux کدوما هستن؟
156	تفاوت‌های mapDispatchToProps و mapStateToProps چی هست؟
157	توی ریدیوسر می‌تونیم به action ی رو dispatch کنیم؟
158	چطوری میشه خارج از کامپوننت میشه store ریداکس دسترسی داشت؟
159	اشکالات پترن MVW کدوما هستن؟
160	تشابهی بین Redux و RxJS هست؟
161	چطوری میشه به اکشن رو موقع لود dispatch کرد؟
162	چطوری از متد connect از پکیج react-redux استفاده می‌کنیم؟
163	چطوری میشه state ریداکس رو ریست کرد؟
164	هدف از کاراکتر @ توی decorator متد connect چیه؟
165	تفاوت‌های context و React Redux چیه؟
166	چرا به توابع state ریداکس reducer میگن؟
167	توی redux چطوری میشه api request زد؟
168	آیا لازمه همه state همه کامپوننت‌ها مونو توی ریداکس نگهداری کنیم؟
169	روش صحیح برای دسترسی به store ریداکس چیه؟
170	تفاوت‌های component و container توی ریداکس چی هست؟
171	هدف از constant ها تا type ها توی ریداکس چیه؟

ردیف	سوال
172	روش‌های مختلف برای نوشتن mapDispatchToProps چیه؟
173	کاربرد پارامتر ownProps توی mapStateToProps و mapDispatchToProps چیه؟
174	ساختار پوشه‌بندی ریشه ریداکس اکثرا چطوره؟
175	redux-saga چیه؟
176	مدل ذهنی redux-saga چطوره؟
177	تفاوت افکت‌های call و put توی redux-saga چی هست؟
178	Redux Thunk چیه؟
179	تفاوت‌های redux-saga و redux-thunk جیا هستن؟
180	Redux DevTools چیه؟
181	ویژگی‌های Redux DevTools کدوما هستن؟
182	سلکتورهای ریداکس چی هستن و چرا باید ازشون استفاده کنیم؟
183	Redux Form چیه؟
184	اصلی‌ترین ویژگی‌های Redux Form چیه؟
185	چطوری میشه چندتا middleware به ریداکس اضافه کرد؟
186	چطوری میشه توی ریداکس initial state تعریف کرد؟
187	تفاوت‌های Relay با Redux کدوما هستن؟
	React Native
188	تفاوت‌های React Native و React کدوما هستن؟
189	چطوری میشه برنامه React Native رو تست کرد؟
190	چطوری میشه توی React Native لاگ کرد؟
191	چطوری میشه React Native رو دیباگ کرد؟

ردیف	سوال
	کتابخانه‌های مورد استفاده با ری‌اکت
192	کتابخانه reselect چیه و چطوری کار می‌کنه؟
193	Flow چیه؟
194	تفاوت‌های Flow و PropTypes کدوما هستن؟
195	چطوری از آیکون‌های font-awesome توی ری‌اکت استفاده کنیم؟
196	React Dev Tools چیه؟
197	چرا توی کروم devtools برای فایل‌های local لود نمیشه؟
198	چطوری از Polymer توی React استفاده کنیم؟
199	مزایای React نسبت به Vue.js کدوما هستن؟
200	تفاوت‌های React و Angular کدوما هستن؟
201	چرا تب React در DevTools نشان داده نمی‌شود؟
202	Styled components چیه؟
203	یه مثال از Styled Components می‌تونم بگی؟
204	Relay چیه؟
205	چطوری همیشه از تایپ اسکریپت توی create-react-app استفاده کرد؟
	متفرقه
206	اصلی‌ترین ویژگی‌های کتابخانه reselect کدوما هستن؟
207	یه مثال از کارکرد کتابخانه reselect بزن؟
208	توی Redux اکشن چیکار می‌کنه؟
209	استاتیک شی با کلاس‌های ES6 در React کار می‌کنه؟
210	ریداکس رو فقط با ری‌اکت همیشه استفاده کرد؟

ردیف	سوال
211	برای استفاده از Redux به ابزار build خاصی احتیاج داریم؟
212	مقادیر پیش فرض ریداکس فرم چطوری تغییرات رو از state می گیرن؟
213	توی PropTypes های ری اکت چطوری میشه برای یه prop چند نوع داده مجاز مشخص کرد؟
214	می تونیم فایل svg رو به عنوان کامپوننت import کنیم؟
215	چرا استفاده از توابع ref callback درون خطی توصیه نمیشه؟
216	render hijacking توی ری اکت چیه؟
217	پیاده سازی factory یا سازنده HOC چطوره؟
218	چطوری به یه کامپوننت ری اکت عدد پاس بدیم؟
219	لازمه همه state ها رو توی ریداکس مدیریت کنیم؟ لزومی به استفاده از state داخلی داریم؟
220	هدف از متد registerServiceWorker توی ری اکت چیه؟
221	تابع memo ری اکت چیه؟
222	تابع lazy ری اکت چیه؟
223	چطوری با استفاده از تابع setState از رندر غیرضروری جلوگیری کنیم؟
224	توی نسخه ۱۶ ری اکت چطوری میشه آرایه، Strings و یا عدد رو رندر کنیم؟
225	چطوری میشه از تعریف ویژگی در کلاس کامپوننت استفاده کرد؟
226	hook ها چی هستن؟
227	چه قوانینی برای هوک ها باید رعایت بشن؟
228	چطوری میشه از استفاده درست هوک ها اطمینان حاصل کرد؟
229	تفاوت های Flux و Redux کدوما هستن؟
230	مزایای ری اکت روتر نسخه ۴ چیه؟

ردیف	سوال
231	می‌تونن راجع به متد <code>componentDidCatch</code> توضیح بدی؟
232	در چه سناریویی <code>error boundary</code> خطا رو <code>catch</code> نمی‌کنه؟
233	چرا نیازی به <code>error boundaries</code> برای <code>event handler</code> ها نیست؟
234	تفاوت بلوک <code>try catch</code> و <code>error boundary</code> ها چیه؟
235	رفتار خطاهای <code>uncaught</code> در ری‌اکت 16 چیه؟
236	محل مناسب برای قرار دادن <code>error boundary</code> کجاست؟
237	مزیت چاپ شدن <code>stack trace</code> کامپوننت‌ها توی متن ارور <code>boundary</code> ری‌اکت چیه؟
238	متدی که در تعریف کامپوننت‌های <code>class</code> الزامیه؟
239	نوع‌های ممکن برای مقدار بازگشتی متد <code>render</code> کدوما هستن؟
240	هدف اصلی از متد <code>constructor</code> چیه؟
241	آیا تعریف متد سازنده توی ری‌اکت الزامیه؟
242	<code>Default prop</code> ها چی هستن؟
243	چرا نباید تابع <code>setState</code> رو توی متد <code>componentWillUnmount</code> فراخوانی کرد؟
244	کاربرد متد <code>getDerivedStateFromError</code> چیه؟
245	کدوم متدها و به چه ترتیبی در طول ری‌رندر فراخوانی میشن؟
246	کدوم متدها موقع <code>error handling</code> فراخوانی میشن؟
247	کارکرد ویژگی <code>displayName</code> چیه؟
248	سایپورت مرورگرها برای برنامه ری‌اکتی چطوره؟
249	هدف از متد <code>unmountComponentAtNode</code> چیه؟
250	<code>code-splitting</code> چیه؟
251	مزایای حالت <code>strict</code> چیه؟

ردیف	سوال
252	Fragment های دارای key هستند؟
253	آیا ری اکت از همه attribute های HTML پشتیبانی می‌کند؟
254	محدودیت های HOC ها چی هستند؟
255	چطوری میشه forwardRefs رو توی DevTools دیباگ کرد؟
256	مقدار یه props کامپوننت کی true میشه؟
257	NextJS چیه و ویژگی های اصلیش کدوما هستند؟
258	چط،وی کی تونیم یه تابع event handler رو به یه کامپوننت پاس بدیم؟
259	استفاده از توابع arrow برای متدهای render خوبه؟
260	چطوری از اجرای چندباره یه تابع جلوگیری کنیم؟
261	JSX چطوری از حمله های Injection جلوگیری می‌کند؟
262	چطوری element های رندر شده رو آپدیت کنیم؟
263	چرا prop ها read only هستند؟
264	چرا می‌گیم تابع setState از طریق merge کردن state را مدیریت می‌کند؟
265	چطوری می‌تونیم به متد event handler پارامتر پاس بدیم؟
266	چطوری از رندر مجدد کامپوننت ها جلوگیری کنیم؟
267	شرایطی که بدون مشکل پرفورمنس بتونیم از ایندکس به عنوان key استفاده کنیم چی هست؟
268	key های ری اکت باید به صورت عمومی منحصر بفرد باشن؟
269	گزینه های محبوب برای مدیریت فرم ها توی ری اکت کدوما هستند؟
270	مزایای کتابخانه فرمیک نیست به redux form چیه؟
271	چرا اجباری برای استفاده از ارث بری توی ری اکت نیست؟ مزیتی داره؟
272	می‌تونیم از web components توی برنامه ری اکت استفاده کنیم؟

ردیف	سوال
273	dynamic import چیه؟
274	loadable component ها چی هستن؟
275	کامپوننت suspense چیه؟
276	چطوری به ازای route می‌تونیم code splitting داشته باشیم؟
277	یه مثال از نحوه استفاده از context میزنی؟
278	هدف از مقدار پیش‌فرض توی context چیه؟
279	چطوری از contextType استفاده می‌کنین؟
280	consumer چیه؟
281	چطوری مسائل مربوط به پرفورمنس با context رو حل می‌کنین؟
282	هدف از forward ref توی HOC ها چیه؟
283	توی کامپوننت‌ها می‌تونیم پراپ ref داشته باشیم؟
284	چرا در هنگام استفاده از ForwardRef ها نیاز به احتیاط بیشتری در استفاده از کتابخانه‌های جانبی داریم؟
285	چطوری بدون استفاده از ES6 کلاس کامپوننت بسازیم؟
286	استفاده از ری‌اکت بدون JSX ممکن است؟
287	الگوریتم‌های diffing ری‌اکت چی هستن؟
288	قوانینی که توسط الگوریتم‌های diffing پوشش داده می‌شوند کدام هستن؟
289	چه موقعی نیاز هست که از ref ها استفاده کنیم؟
290	برای استفاده از render prop ها لازمه که اسم prop رو render بزاریم؟
291	مشکل استفاده از render props با pure component ها چیه؟
292	چطوری با استفاده از render props می‌تونیم HOC ایجاد کنیم؟
293	تکنیک windowing چیه؟

ردیف	سوال
294	توی JSX به مقدار falsy رو چطوری چاپ کنیم؟
295	یه مورد استفاده معمول از portals مثال میزنی؟
296	توی کامپوننت‌های کنترل نشده چطوری مقداری پیش فرض اضافه کنیم؟
297	stack موردعلاقه شما برای کانفیگ پروژه ری‌اکت چیه؟
298	تفاوت DOM واقعی و Virtual DOM چیه؟
299	چطوری Bootstrap رو به یه برنامه ری‌اکتی اضافه کنیم؟
300	می‌تونی یه لیست‌تی از معروف‌ترین وب‌سایت‌هایی که از ری‌اکت استفاده می‌کنن رو بگی؟
301	استفاده از تکنیک CSS In JS تو ری‌اکت توصیه میشه؟
302	لازمه همه کلاس کامپوننت‌ها رو تبدیل کنیم به هوک؟
303	چطوری همیشه با هوک‌های ری‌اکت دیتا fetch کرد؟
304	هوک‌ها همه موارد کاربرد کلاس‌ها رو پوشش میده؟
305	نسخه پایدار ری‌اکت که از هوک پشتیبانی می‌کنه کدومه؟
306	چرا از حالت destructuring آرایه برای useState استفاده می‌کنیم؟
307	منابعی که باعث معرفی ایده هوک‌ها شدن کدوما بودن؟
308	چطوری به API‌های ضروری اجزای وب دسترسی پیدا کنیم؟
309	formik چیه؟
310	middlewareهای مرسوم برای مدیریت ارتباط‌های asynchronous توی Redux کدوما هستن؟
311	مرورگرها کد JSX رو متوجه میشن؟
312	Data flow یا جریان داده ری‌اکت رو توضیح میدی؟
313	react scripts چیه؟

ردیف	سوال
314	ویژگی‌های create react app چیه؟
315	هدف از متد renderToNodeStream چیه؟
316	MobX چیه؟
317	تفاوت‌های بین Redux و MobX کدوما هستن؟
318	لازمه قبل از شروع ری‌اکت ES6 رو یاد گرفت؟
319	Concurrent Rendering چیه؟
320	تفاوت بین حالت async و concurrent چیه؟
321	می‌تونیم از آدرس‌های دارای url جاوااسکریپت در ری‌اکت 16.9 استفاده کرد؟
322	هدف از پلاگین eslint برای هوک‌ها چیه؟
323	تفاوت‌های Declarative و Imperative توی ری‌اکت چیه؟
324	مزایای استفاده از تایپ اسکریپت با ری‌اکت چیه؟

هسته ری‌اکت

1. ری‌اکت چیه؟

ری‌اکت یه کتابخونه متن‌باز هست که برای ساختن رابط کاربری به خصوص برنامه‌های تک صفحه‌ای استفاده میشه. از این کتابخونه برای مدیریت لایه view توی برنامه‌های وب و موبایل استفاده میشه. توسط [Jordan Walke](#) تولید شده که یه مهندس نرم‌افزار توی شرکت فیس‌بک هستش. اولین بار سال ۲۰۱۱ و روی برنامه اینستاگرام مورد استفاده قرار گرفت.

↑ فهرست مطالب

2. اصلی‌ترین ویژگی‌های ری‌اکت کدوما هستن؟

اصلی‌ترین ویژگی‌های ری‌اکت اینا هستن:

- از **VirtualDOM** به جای RealDOM استفاده می‌کنه چون هزینه تغییرات RealDOM زیاده (یعنی پیدا کردن DOM Element و حذف یا به روز رسانی با سرعت کمتری انجام میشه)
- از **SSR (server side rendering)** پشتیبانی می‌کنه
- از جریان داده‌ها یا data binding به صورت **یک طرفه (unidirectional)** پیروی می‌کنه
- برای توسعه view از UI کامپوننت‌های **reusable/composable** استفاده می‌کنه

[↑ فهرست مطالب](#)

3. JSX چیه؟

JSX یه افزونه با سینتکسی شبیه به XML برای ECMAScript است (مخفف *Javascript* *XML*). اگه بخوایم ساده بگیریم وظیفه اش اینه که سینتکسی ساده تر از `React.createElement` در اختیارتون قرار میده، شما می‌تونین *Javascript* رو در کنار ساختاری شبیه به HTML داشته باشید. تو مثال زیر می‌بینید که نوشته داخل تگ `h1` مثل یک تابع *Javascript* به تابع `render` تحویل داده میشه.

1. function component

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{"Welcome to React world!"}</h1>
      </div>
    );
  }
}
```

2. class component

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{"Welcome to React world!"}</h1>
      </div>
    );
  }
}
```

↑ فهرست مطالب

4. تفاوت‌های Component و Element چیست؟

Element یک شی ساده است که وظیفه داره اون چیزی که روی صفحه نمایش داده میشه رو توصیف کنه، حالا ممکنه به صورت یک DOM node باشه یا به صورت componentهای دیگه. *Element* ها می‌تونن شامل *Elements* های دیگه به عنوان props باشند. ساختن یک *Element* در React کار ساده و کم دردسریه اما وقتی که ساخته شد هیچ وقت نمیشه تغییرش داد.

تو مثال زیر یک شی که توسط React Element ساخته شده رو میبینیم:

```
const element = React.createElement("div", { id: "login-btn" }, "
```

تابع `React.createElement` که توی قطعه کد بالا میبینید یه `object` شبیه به این برمی‌گردونه:

```
{
  type: 'div',
  props: {
    children: 'Login',
    id: 'login-btn'
  }
}
```

و آخرش هم با استفاده از `ReactDOM.render` می‌تونیم توی `DOM` رندر کنیم

```
<div id="login-btn">Login</div>
```

درحالی‌که یه **component** می‌تونه به روشهای مختلفی ساخته بشه. می‌تونه یه `class` باشه یا یه متد `render`. یا حتی به عنوان یه جایگزین ساده‌تر به صورت یک تابع تعریف بشه. در هر دو حالت کامپوننت ساخته شده props رو به عنوان ورودی دریافت می‌کنه و یه

خروجی رو به صورت یه JSX tree برمی‌گردونه. به مثال زیر دقت کنیم که چطور با استفاده از یه تابع و JSX یک کامپوننت ساخته میشه:

```
const Button = ({ onLogin }) => (  
  <div id={"login-btn"} onClick={onLogin}>  
    Login  
  </div>  
);
```

JSX به `React.createElement` ترنسپایل (transpile) میشه:

```
const Button = ({ onLogin }) =>  
  React.createElement(  
    "div",  
    { id: "login-btn", onClick: onLogin },  
    "Login"  
  );
```

↑ فهرست مطالب

5. توری اکت چطوری کامپوننت می‌سازیم؟

تو سوال قبل یه اشاره کوچیک کردیم که دوتا راه برای ساختن کامپوننت وجود داره. ۱. **Function Components**: این ساده‌ترین راه برای ساختن یه کامپوننته. یه *Pure Javascript Function* رو در نظر بگیرید که Props که خودش یه *object* هست رو به عنوان پارامتر ورودی میگیره و یه *React Element* به عنوان خروجی برمی‌گردونه مثل همین مثال پایین:

```
function Greeting({ message }) {  
  return <h1>`Hello, ${message}`</h1>;  
}
```

۲. **Class Components**: شما می‌تونین از *class* که در ES6 به جاوااسکریپت اضافه شده برای این کار استفاده کنیم. کامپوننت مثال قبلی رو اگه بخواییم با *class* پیاده سازی کنیم اینجوری میشه:

```
class Greeting extends React.Component {  
  render() {  
    return <h1>`Hello, ${this.props.message}`</h1>;  
  }  
}
```

فقط یادتون نره تو این روش متد `render` به جورایی `required` میشه.

↑ فهرست مطالب

6. کی باید از Class Component بجای Function Component استفاده کنیم؟

میشه گفت هیچ لزومی به این کار نیست (مگر در مواقع خیلی خاص مثل `error boundary` ها) و از ورژن 16.8 ری اکت به بعد و با اضافه شدن هوک ها به فانکشن کامپوننت ها، شما می تونین از `state` یا `lifecycle method` یا تمامی فیچرهایی که قبلا فقط در کلاس کامپوننت ها قابل استفاده بود، توی فانکشن کامپوننت ها تون استفاده کنین.

ولی قبلا اگه کامپوننت نیاز به `state` یا `lifecycle methods` داشت از کلاس کامپوننت ها باید استفاده می کردیم و در غیر این صورت می رفتیم سراغ فانکشن کامپوننت ها.

↑ فهرست مطالب

7. Pure Components چیه؟

برای اینکه ری رندر شدن یه کامپوننت رو کنترل کنیم، توی کلاس کامپوننت ها بجای `Component` از `PureComponent` کامپوننت مون رو می ساختیم، در حقیقت `PureComponent` دقیقا مثل `Component` می مونه فقط تنها تفاوتی که داره اینه که برخلاف `Component` خودش به صورت خودکار متد `shouldComponentUpdate` رو هندل می کنه. وقتی که `props` یا `state` در کامپوننت تغییر می کنه، `PureComponent` یه مقایسه سطحی روی `props` و `state` انجام میده (`shallow comparison`) در حالیکه `Component` این مقایسه رو به صورت خودکار انجام نمیده و به طور پیش فرض کامپوننت هربار که `shouldComponentUpdate` فراخوانی بشه `re-render` میشه. بنابراین توی `Component` باید این متد `override` بشه. برای انجام این کار روی فانکشن کامپوننت ها، میشه از `React.memo` استفاده کرد.

↑ فهرست مطالب

8. state تو ری اکت چیکار می کنه؟

State در هر کامپوننت بسته به کلاس کامپوننت بودن یا فانکشن بودن نوع متفاوتی دارد، مثلاً در کلاس کامپوننت‌ها *state* به آجکتی که به سری اطلاعات که در طول عمر کامپوننت ما ممکنه تغییر کنه رو در خودش ذخیره می‌کنه. ما باید تمام تلاشمون رو بکنیم که *state*مون در ساده‌ترین حالت ممکن باشه و تاجایی که می‌تونیم تعداد کامپوننت‌هایی که *stateful* هستن رو کاهش بدیم. به عنوان مثال بیایید به کامپوننت *User* رو که به *state* دارد بسازیم:

1. function

```
const User = () => {  
  const [message, setMessage] = useState('Hello react world!');  
  
  return (  
    <div>  
      <h1>{message}</h1>  
    </div>  
  );  
}
```

2. class

```
class User extends React.Component {  
  constructor(props) {  
    super(props);  
  
    this.state = {  
      message: "Welcome to React world",  
    };  
  }  
  
  render() {  
    return (  
      <div>  
        <h1>{this.state.message}</h1>  
      </div>  
    );  
  }  
}
```



state is used for internal communication inside a Component

State و Props بهم شبیه هستند ولی State ها کاملاً در کنترل کامپوننت هستند و فقط مختص به همون کامپوننت هستند (private). یعنی state ها در هیچ کامپوننتی به غیر از اونی که مالک state هست در دسترس نخواهند بود.

[↑ فهرست مطالب](#)

9. props تو ری اکت چیکار می‌کنه؟

`_Prop` ها ورودی کامپوننت‌ها هستند. می‌تونن یه مقدار ساده یا یه `object` شامل یه مجموعه مقدار باشن که در لحظه ایجاد کامپوننت و بر اساس یه قاعده نام گذاری که خیلی شبیه به `attribute` های `HTML` هست، به کامپوننت پاس داده میشن. در واقع این مقادیر، داده‌هایی هستند که از کامپوننت پدر به فرزند تحویل داده میشن. هدف اصلی وجود Props در ری اکت ایجاد ساختارهای زیر در یک کامپوننت، به طور کلی میشه گفت که هدفشون:

- 1 - پاس دادن مقادیر به کامپوننت‌های فرزند.
- 2 - پاس دادن متد تغییر `state` و `trigger` کردن اون متد در زمان تغییر دلخواه.
- 3 - استفاده از اونا برای پاس دادن `jsx` و رندر کردن یه المنت دلخواه داخل یه کامپوننت دیگه.

به عنوان مثال، یه کامپوننت با استفاده از `renderProp` میسازیم:

```
<Element renderProp={<span>Hi there</span>} />
```

این `renderProp` (یا هرچیز دیگه‌ای که شما می‌تونین اسمشو بزارین) در نهایت تبدیل به یک `property` خواهد شد که داخل `props` ورودی کامپوننت به شکل `object` قابل دسترس هست.


```
const Element = (props) => {
  return <div>
    {props.renderProp}
  </div>
}
```

توی کامپوننت بالا به jsx با prop به کامپوننت Element داده میشه و توی اون رندر میشه 😊

[↑ فهرست مطالب](#)

10. تفاوت state و props چیه؟

هر دوتاشون javascript plain object هستن، یعنی یه object که یه سری property داره که به یه سری مقدار ختم میشن و خبری از فانکشن و چیزهای دیگه روی این object وجود نداره (تقریباً). هر دوتاشون وظیفه دارن مقادیری که روی render تاثیر گذار هست رو نگه‌داری کنن اما عملکردشون با توجه به کامپوننت متفاوت خواهد بود. Props شبیه به پارامترهای ورودی یک فانکشن، به کامپوننت پاس داده میشن در حالیکه state شبیه به متغیرهایی که داخل فانکشن ساخته شدن، توسط خود کامپوننت ایجاد و مدیریت میشه.

[↑ فهرست مطالب](#)

11. چرا نباید state رو مستقیماً آپدیت کنیم؟

اگه یه بار تلاش کنید که مستقیماً state رو آپدیت کنید متوجه می‌شین که کامپوننت شما مجدداً render نمیشه.

```
// Wrong
let [message, setMessage] = useState('test');
message = "Hello world";
```

به جای اینکه مستقیماً state رو آپدیت کنیم باید از متد setState در Class Component و از useState در Function Components استفاده کنیم. این متدها یک آپدیت در شی state رو برنامه ریزی و مدیریت می‌کنن و وقتی تغییر انجام شد کامپوننت شما re-render خواهد شد.

```
const [message, setMessage] = React.useState("Hello world");  
  
setMessage("New Hello world");
```

↑ فهرست مطالب

12. هدف از متدهای callback توی استفاده از setState چیه؟

توی کلاس کامپوننت‌ها همیشه بعد از انجام شدن یه `setState`، یه کار خاص دیگه‌ای رو توی `callback` انجام داد.

`callback function` زمانی که `setState` تموم شد و کامپوننت مجدداً `render` شد فراخوانی میشه. از اونجایی که `setState` به شکل **asynchronous** یا همون غیرهمزمان اجرا میشه از `callback` برای کارهایی استفاده میشه که بعد از تابع `setState` قراره اجرا بشن. **نکته مهم:** بهتره که به جای `callback` از `lifecycle` متدها استفاده کنیم.

```
setState({ name: "John" }, () =>  
  console.log("The name has updated and component re-rendered")  
);
```

این سازوکار رو، همیشه روی فانکشن کامپوننت‌ها با یه کاستوم هوک به شکل زیر پیاده‌سازی کرد:

```
function useStateCallback(initialState) {  
  const [state, setState] = useState(initialState);  
  const cbRef = useRef(null); // mutable ref to store current callback  
  
  const setStateCallback = useCallback((state, cb) => {  
    cbRef.current = cb; // store passed callback to ref  
    setState(state);  
  }, []);  
  
  useEffect(() => {  
    // cb.current is `null` on initial render, so we only execute if it's not null  
    if (cbRef.current) {  
      cbRef.current(state);  
      cbRef.current = null; // reset callback after executing  
    }  
  }, [state]);  
  
  return [state, setStateCallback];  
}
```

یه کم پیچیده به نظر میرسه ولی خب می‌تونه یه هوک خیلی کاربردی باشه. به شکل زیر هم ازش استفاده می‌کنیم:

```
const App = () => {
  const [state, setState] = useStateCallback(0); // same API as

  const handleClick = () => {
    setState(
      prev => prev + 1,
      // 2nd argument is callback , `s` is *updated* state
      s => console.log("I am called after setState, state:")
    );
  };

  return <button onClick={handleClick}>Increment</button>;
}
```

↑ فهرست مطالب

13. تفاوت بین نحوه مدیریت رویداد HTML و React چیه؟

1. توی HTML، عنوان رخداد حتما باید با حرف کوچیک شروع بشه یا اصطلاحا *lowercase* باشه:

```
<button onclick="activateLasers()"></button>
```

ولی توی ری‌اکت از *camelCase* پیروی می‌کنه:

```
<button onClick={activateLasers}>Test</button>
```

2. توی HTML می‌تونیم برای جلوگیری از اجرای رفتار پیش‌فرض (preventDefault) یه مقدار `false` برگردونیم:

```
<a href="#" onclick='console.log("The link was clicked."); return
```

ولی توی ری‌اکت برای انجام این مورد حتما باید از `preventDefault` استفاده بشه:

```
function handleClick(event) {
  event.preventDefault();
  console.log("The link was clicked.");
}
```

3. توی HTML برای اجرای تابع حتما باید اونی با گذاشتن پرانتزهایی که بعد اسمش میزاریم invoke کنیم ()
ولی توی ری اکت اجباری به گذاشتن () جلوی اسم تابع نیست. (برای مثال به کد اول و تابع "activateLasers" دقت کنید)

↑ فهرست مطالب

14. چطوری متد یا event رو به تابع callback توی JSX bind کنیم؟

توی کلاس کامپوننت‌ها به سه روش مختلف می‌تونیم this رو به تابع هندلر موردنظرمون bind کنیم:

1. **Bind کردن توی Constructor:** توی کلاس‌های جاوااسکریپتی متدها به صورت پیش‌فرض bound نمیشن. همین موضوع توی کلاس کامپوننت‌های ری اکتی برای متدهای موجود هم رخ میده که اکثرا توی متد سازنده یا همون constructor می‌آییم bind می‌کنیم.

```
class Component extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    //...
  }
}
```

2. **استفاده از فیلد عمومی کلاس (public):** اگه از روش اول خوشتون نمیاد این روش هم می‌تونه context درست رو موقع callback‌ها براتون فراهم کنه.

```
handleClick = () => {
  console.log("this is:", this);
};

<button onClick={this.handleClick}>{"Click me"}</button>
```

3. **توابع arrow توی callback:** می‌تونین از توابع arrow به شکل مستقیم توی callback‌ها استفاده کنین.

```
<button onClick={(event) => this.handleClick(event)}>{"Click me"}</button>
```

نکته: اگر متدهای callback به عنوان prop به کامپوننت‌های فرزندشون پاس داده بشن، ممکنه اون کامپوننت‌ها re-rendering‌های ناخواسته‌ای داشته باشن. توی اینگونه موارد روش توصیه شده استفاده از `bind` یا فیلد عمومی کلاس برای مدیریت پرفورمنس هستش.

↑ فهرست مطالب

15. چطوری میشه یک مقدار رو به یه تابع callback یا eventHandler پاس بدیم؟

می‌تونیم از توابع *arrow* استفاده کنیم که با `wrap` کردن دور *event handler* و پاس دادن مقدار موردنظرمون بهش کارمونو انجام بدیم:

```
const handleClick = (id) => {
  console.log("Hello, your ticket number is", id);
};

return users.map(user => <button onClick={() => handleClick(user.id)}>
```

جدا از این روش‌ها، میشه با ایجاد یه *curry*، یه تابع دیگه دور تابع هندلر خودمون `wrap` کنیم و پارامتر رو به اون پاس بدیم:

```
const handleClick = (id) => () => {
  console.log("Hello, your ticket number is", id);
};

return users.map(user => <button onClick={handleClick(user.id)} />
```

↑ فهرست مطالب

16. Synthetic events (رویدادهای مصنوعی) تو ری‌اکت کدوما هستن؟

`SyntheticEvent` یه رخداد *cross-browser* هست که به‌عنوان یه *wrapper* دور *event* اصلی مرورگر قرار می‌گیره. رابط API برای کارکردن با اون دقیقاً مثل رخداد *native*

مرورگرهاست که شامل `stopPropagation` و `preventDefault` می‌شود، با این تفاوت که این رخدادها بر روی همه مرورگرها کار می‌کنند.

↑ فهرست مطالب

17. عبارات شرطی درون خطی چیست؟

برای بررسی یک شرط می‌تونیم از عبارت شرطی `if` استفاده کنیم، البته عملگرهای درون خطی سه‌گانه (ternary) هم همیشه استفاده کرد که از ویژگی‌های خود JS هستند. جدا از این ویژگی‌ها، می‌تونیم هر عبارتی داخل آکولاد و توی JSX به اصطلاح `embed` یا ترکیب کنیم و با عملگر منطقی `&&` ترکیب کنیم، مثال پایینی رو ببینید:

```
<h1>Hello!</h1>;
{
  messages.length > 0 && !isLogin ? (
    <h2>You have {messages.length} unread messages.</h2>
  ) : (
    <h2>You don't have unread messages.</h2>
  );
}
```

↑ فهرست مطالب

18. پارامترهای `key` چیست؟ چیکار می‌کنند و مزایای استفاده از اونا توی حلقه‌ها چیست؟

پارامتر `key` به `attribute` ویژه است و موقعی که داریم به آرایه از المان‌ها رو ایجاد می‌کنیم این پارامتر رو **باید** بهشون به عنوان `prop` بدیم. `key` به ری‌اکت کمک می‌کنند که بدونند باید کدوم المان رو دقیقاً اضافه، حذف یا به روز کنه. اکثراً از ID یا از یک دیتای یونیک به عنوان `key` استفاده می‌کنند:

```
const todoItems = todos.map((todo) => <li key={todo.id}>{todo.text}
```

موقعی که به آیدی خاص برای المان‌ها نداریم، ممکنه بپایید و از اندیس یا همون `index` به عنوان `key` استفاده کنید:

```
const todoItems = todos.map((todo, index) => (
  <li key={index}>{todo.text}</li>
));
```

نکته:

1. استفاده از `index`ها برای `key` توصیه **نمیشه** چون ممکنه ترتیب عناصر خیلی راحت عوض بشه و این می‌تونه پرفورمنس برنامه رو تحت تاثیر بزاره.
2. اگه بیابین و لیست مورد نظر رو به جای `li` مثلا با یه کامپوننت به اسم `ListItem` جایگزین کنین و `prop` مورد نظر `key` رو به جای `li` به اون پاس بدیم، یه warning توی کنسول خواهیم داشت که میگه `key` پاس داده نشده.

↑ فهرست مطالب

19. کاربرد `ref`ها چیه؟

`ref` به عنوان یه مرجع برای دسترسی مستقیم به اِلِمان موردنظرمون استفاده میشه. تا حد امکان و توی اکثر مواقع بهتره از اونا استفاده نکنیم، البته خیلی می‌تونن کمک کننده باشن چون دسترسی مستقیمی به DOM element یا instance اصلی component بهمون میدن.

↑ فهرست مطالب

20. چطوری از `ref` استفاده کنیم؟

دو تا روش وجود داره:

1. استفاده از `React.createRef()` و پاس دادن اون به `element` مورد نظرمون با `ref` attribute.

```
const Component = () => {  
  const myRef = React.createRef();  
  
  return <div ref={myRef} />;  
};
```

2. اگه از نسخه ۱۶.۸ به بالاتر هم استفاده می‌کنیم که یه هوک به اسم `useRef` هست و می‌تونیم به سادگی توی کامپوننت‌های تابعی ازش استفاده کنیم.
مثل:

```
const RenderCounter = () => {
  const counter = useRef(0);

  // Since the ref value is updated in the render phase,
  // the value can be incremented more than once
  counter.current = counter.current + 1;

  return <h1>{'The component has been re-rendered ${counter} times.'};
};
```

↑ فهرست مطالب

21. forward ref چیست؟

Ref forwarding ویژگی‌ای است که به بعضی از کامپوننت‌ها این اجازه را می‌دهد که *ref* دریافت شده را به کامپوننت فرزند انتقال بدن.

```
const ButtonElement = React.forwardRef((props, ref) => (
  <button ref={ref} className="CustomButton">
    {props.children}
  </button>
));

// Create ref to the DOM button:
const ref = React.createRef();
<ButtonElement ref={ref}>{"Forward Ref"}</ButtonElement>;
```

↑ فهرست مطالب

22. بین callback refs و تابع findDOMNode کدام رو ترجیح میدی؟

ترجیح این است که از callback refs به جای `findDOMNode` استفاده کنیم، چون `findDOMNode` از توسعه کدهای ری‌اکت در آینده جلوگیری می‌کند و ممکنه خطاهای ناخواسته ایجاد کند. رویکرد قدیمی استفاده از `findDOMNode`:


```
class MyComponent extends Component {
  componentDidMount() {
    findDOMNode(this).scrollIntoView();
  }

  render() {
    return <div></div>;
  }
}
```

رویکرد توصیه شده:

```
const MyComponent = () => {
  const nodeRef = useRef();
  useEffect(() => {
    nodeRef.current.scrollIntoView();
  }, []);

  return <div ref={nodeRef} />;
}
```

[↑ فهرست مطالب](#)

23. چرا Ref های متنی منقضی محسوب می شوند؟

اگر قبلاً با ری اکت کار کرده باشید، احتمالاً با این API قدیمی تر آشنا هستید که توی اون ویژگی ref به رشته‌ست، مثل `'ref='textInput` و DOM به صورت `refs.textInput` قابل دسترسیه. البته این ویژگی توی نسخه ۱۶ ری اکت **حذف** شده و توصیه نمیشه استفاده بشه، فقط برای یادگیری هست.

1. ری اکت رو مجبور می‌کنن که عناصر در حال اجرا رو دنبال کنه و این مساله یکم مشکل سازه چون باعث میشه خطاهای عجیب و غریب وقتی که ماژول ری اکت باندل میشه، رخ بده.

2. قابل انعطاف نیستن. اگر یه کتابخونه خارجی یه ref رو روی فرزند قرار بده، کاربر نمی‌تونه یه ref دیگه‌ای رو روی اون اضافه کنه.

3. با یه آنالیزگر استاتیک مثل Flow کار نمی‌کنه و در حقیقت Flow یا تایپ اسکریپت نمی‌تونه حدس بزنه که فریم‌ورک چه کاری روی ref انجام میده، مثل نوع داده اون (که ممکنه متفاوت باشه). ref های callback دار بیشتر با آنالیزگرها سازگارترن.

4. اون طور که اکثر مردم از الگوی "render callback" انتظار دارن کار نمی کنه (برای مثال `</ {DataGrid renderRow={this.renderRow}>`)

```
class MyComponent extends Component {
  renderRow = (index) => {
    // This won't work. Ref will get attached to DataTable rather
    return <input ref={"input-" + index} />;

    // This would work though! Callback refs are awesome.
    return <input ref={(input) => (this["input-" + index] = input)} />;
  };

  render() {
    return <DataTable data={this.props.data} renderRow={this.renderRow} />;
  }
}
```

↑ فهرست مطالب

24. Virtual DOM چیست؟

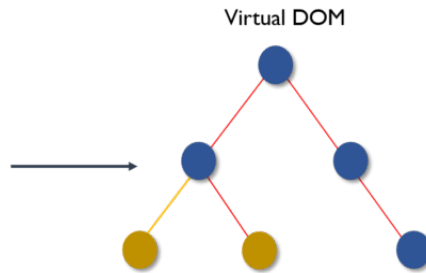
Virtual DOM (VDOM) یا Virtual DOM (VDOM) به کپی داخل memory از DOM واقعی هستش. این کپی از المان‌های رابط کاربری توی حافظه رم نگهداری میشه و همواره با DOM اصلی و واقعی همگام سازی (sync) میشه. این مرحله بین تابع رندر و نمایش element روی صفحه رخ میده و به مجموعه اتفاقاتی که برای مدیریت این موارد انجام میشه *reconciliation* میگن.

↑ فهرست مطالب

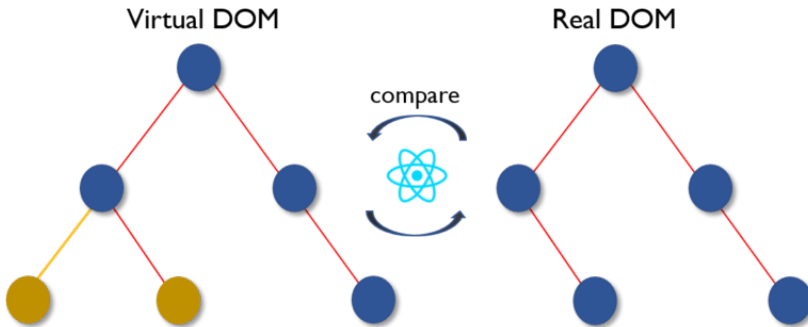
25. Virtual DOM چگونه کار می‌کنه؟

Virtual DOM توی سه مرحله ساده کار می‌کنه.

1. هر زمان که داده‌های اساسی تغییر می‌کنن، کل رابط کاربری توسط DOM مجازی مجددا رندر میشه.

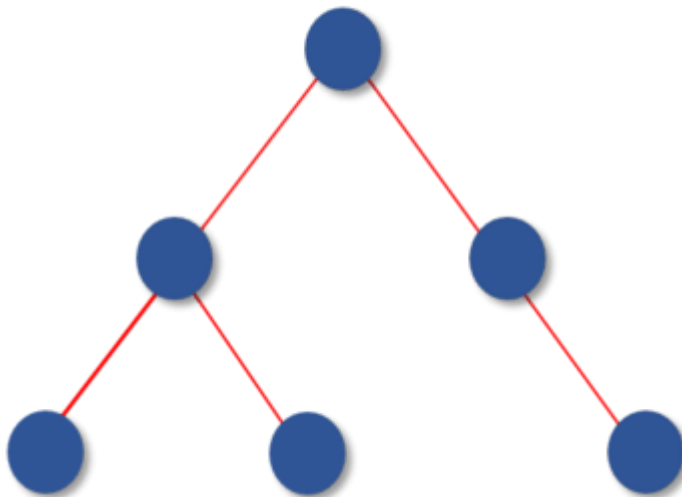


2. تفاوت بین DOM قبلی و جدید محاسبه میشه.



3. بعد از انجام محاسبات، DOM واقعی فقط با مواردی که واقعاً تغییر کردن به روز میشه.

Real DOM (updated)



↑ فهرست مطالب

26. تفاوت بین Virtual DOM و Shadow DOM چیه؟

shadow DOM به تکنولوژی مرورگره که در ابتدا برای تعیین متغیرها و ایزوله سازی CSS در وب کامپوننت (Web component) طراحی شده بود. *virtual DOM* مفهومی که توسط کتابخانه ها برای مدیریت DOM توسط جاوااسکریپت با استفاده از API های مرورگرها اجرا شده.

↑ فهرست مطالب

27. React Fiber چیست؟

Fiber موتور جدید برای عملیات *reconciliation* هست یا میشه گفت که پیاده سازی مجدد الگوریتم هسته ری اکت نسخه ۱۶ هست. هدف پیاده سازی ReactFiber برای بهبود کارکرد توی جاهایی مثل ایجاد انیمیشن، کار روی layout، کار با gesture و قابلیت اینکه عملیات در حال اجرا رو متوقف، قطع یا مجددا فعال کنیم ساخته شده. البته می تونه برای اولویت بندی بروزرسانی های لازم توی DOM رو هم مدیریت کنه.

↑ فهرست مطالب

28. هدف اصلی React Fiber چیست؟

هدف پیاده سازی ReactFiber برای بهبود کارکرد توی ناحیه هایی مثل انیمیشن، layout، کار با gesture بود. میشه گفت مهم ترین ویژگی **incremental-rendering** بوده که قابلیت بخش بندی (chunk کردن) عملیات اجرایی و متوقف و اجرا کردن اون توی فریم های مختلف هست.

↑ فهرست مطالب

29. کامپوننت های کنترل شده چی هستن؟

کامپوننتی که عناصر ورودی رو توی فرم های کاربری کنترل می کنه به عنوان کامپوننت کنترل شده شناخته میشن، توی این کامپوننت ها هر تغییر state به تابع نگهدارنده مرتبط باهاش رو داره. به عنوان مثال، اگر یه input داشته باشیم و بخواییم اسم فرد رو بگیریم و به شکل حروف بزرگ نگهداری کنیم، باید از `handleChange` مثل زیر استفاده کنیم:

```
const [name, setName] = useState('');
const handleChange = (event) => {
  setName(event.target.value.toUpperCase());
}

return <input value={name} onChange={handleChange} />
```

[↑ فهرست مطالب](#)

30. کامپوننت‌های کنترل نشده چی هستن؟

کامپوننت‌های کنترل نشده کامپوننت‌هایی هستن که state‌هاشون رو به صورت داخلی ذخیره می‌کنن و ما می‌تونیم با استفاده از یک ref و از روی DOM مقدار فعلی اون input رو پیدا کنیم. این یکم شبیه HTML سنتیه. مثلاً توی کامپوننت UserProfile زیر، ورودی name با استفاده از ref قابل دسترسیه.

```
const UserProfile = () => {
  const inputRef = useRef();

  const handleSubmit = (event) => {
    alert("A name was submitted: " + inputRef.current.value);
    event.preventDefault();
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        {"Name:"}
        <input type="text" ref={inputRef} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

اکثر مواقع، توصیه میشه که از کامپوننت‌های کنترل‌شده بجای این روش برای پیاده‌سازی فرم‌ها و دریافت داده استفاده کنیم.

[↑ فهرست مطالب](#)

31. تفاوت‌های بین createElement و cloneElement کدوما هستند؟

عناصر JSX به توابع `React.createElement` تبدیل می‌شوند تا عناصر ری‌اکتی بسازند که برای نمایش UI استفاده می‌شوند. درحالی که `cloneElement` برای کلون کردن یک عنصر و فرستادنش به عنوان `prop` جدید استفاده می‌شود.

↑ فهرست مطالب

32. مفهوم lift state up یا مدیریت state در لول بالاتر رو توضیح میدی؟

وقتی که کامپوننت‌های مختلف نیاز به یک داده خاص دارند که بین اونا مشترک به‌تره state‌های مشترک رو تا حد امکان به نزدیک‌ترین کامپوننت بالایی‌شون انتقال بدیم. این مورد به این معنیست که اگر دو کامپوننت فرزند داریم که به یک state مشخص رو دارند مدیریت می‌کنن توی خودشون، اون state رو می‌بریم توی کامپوننت والد و بجای مدیریت به state توی دوتا کامپوننت، از یک جا و توی کامپوننت والد اون رو مدیریت می‌کنیم.

↑ فهرست مطالب

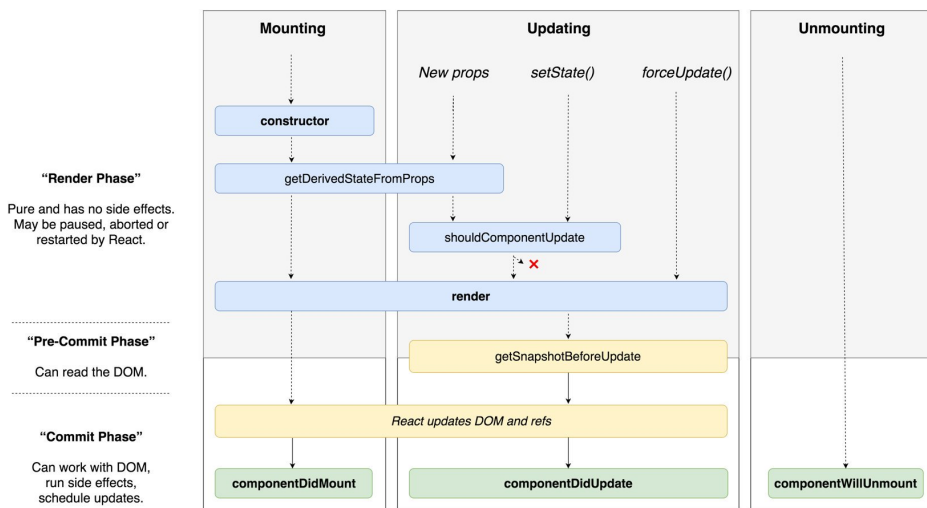
33. فازهای مختلف از lifecycle کامپوننت کدوما هستند؟

چرخه حیات کامپوننت سه مرحله داره:

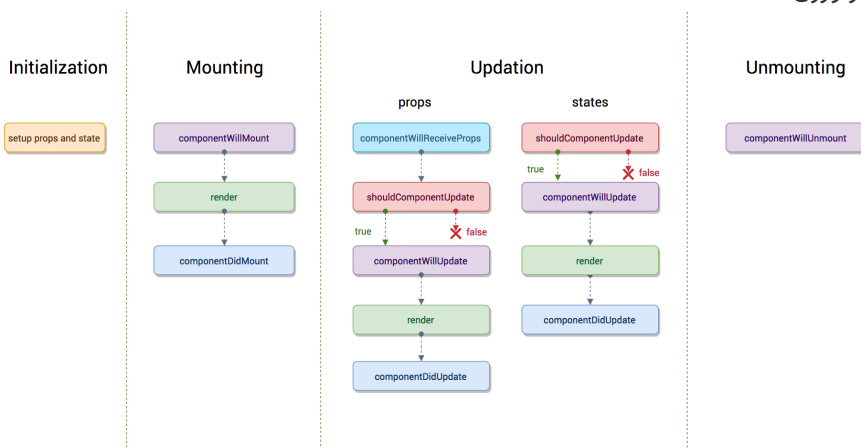
1. **Mounting**: کامپوننت آماده اجرا روی DOM مرورگر هستش. این مرحله مقاردهی اولیه از متدهای `constructor`، `lifecycle` `render`، `getDerivedStateFromProps` و `componentDidMount` رو پوشش میده.
2. **Updating**: در این مرحله، کامپوننت از دو طریق به روزرسانی میشه، ارسال `prop`‌های جدید و به روزرسانی state از طریق `setState`. این مرحله متدهای `getDerivedStateFromProps`، `shouldComponentUpdate`، `render`، `getSnapshotBeforeUpdate` و `componentDidUpdate` رو پوشش میده.
3. **Unmounting**: در مرحله آخر کامپوننت مورد نیاز نیست و از DOM مرورگر حذف میشه. این مرحله فقط شامل متد `componentWillUnmount` میشه.

البته بهتره اینجا این نکته رو بگیم که ری اکت برای به روز کردن DOM به سری فازبندی‌هایی داره که خود اون مرحله رو توی سه تا فاز انجام میده. این پایین به این فازبندی‌ها اشاره می‌کنیم.

1. **Render** کامپوننت بدون هیچ سایدافکتی رندر میشه. این فقط در مورد کامپوننت‌های خالص صدق می‌کنه و در این مرحله، ری اکت می‌تونه رندر رو متوقف، حذف یا restart کنه.
 2. **Pre-commit** قبل از اینکه کامپوننت تغییرات رو روی DOM اعمال کنه، لحظه‌ای وجود داره که به ری اکت اجازه میده از DOM داخل متد `getSnapshotBeforeUpdate` بخونه.
 3. **Commit** ری اکت با DOM کار می‌کنه و lifecycle‌های آخر رو به ترتیب اجرامی کنه، `componentDidMount` برای نصب، `componentDidUpdate` برای به روزرسانی و `componentWillUnmount` برای غیرفعال کردن.
- مراحل ری اکت ۱۶.۳ (یا نسخه تعاملی)



قبل از ورژن ۱۶.۳



34. متدهای lifecycle کامپوننت کدوما هستن؟

ری اکت ۱۶.۳ به بعد

- **getDerivedStateFromProps**: درست قبل از اینکه Render اجرا بشه فراخوانی میشه و در هر بار render فراخوانی میشه.
برای موارد نادری که نیاز داریم از state مشتق بگیریم این متد استفاده میشه. بهتره که اینو بخونید [اگه نیاز داشتن که از state مشتق بگیرین]
<https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html>.
- **componentDidMount**: بعد از اولین رندر اجرا میشه و همه درخواست‌های AJAX، DOM یا بروزرسانی state و تنظیمات event listeners اجرا میشه.
- **shouldComponentUpdate**: تعیین می‌کنه که کامپوننت به روز بشه یا نه.
به طور پیش فرض مقدار **true** رو برمی‌گردونه. اگه مطمئن باشیم که کامپوننت بعد از اینکه state یا props به روزرسانی میشه نیازی به رندر شدن نداره، می‌تونیم مقدار **false** رو برگردونیم. اینجا جای خوبی برای بهبود عملکرده چون این امکان رو بهمون میده که اگه کامپوننت prop جدید میگیره از render مجدد جلوگیری کنیم.
- **getSnapshotBeforeUpdate**: درست قبل از رندر مجدد خروجی به DOM اجرا میشه. هر مقداری که توسط این متد برگشت داده میشه به متد **componentDidUpdate** انتقال داده میشه. برای گرفتن اطلاعات از موقعیت اسکرول DOM مفیده.
- **componentDidUpdate**: بیشتر برای به روزرسانی DOM در پاسخ به تغییرات state یا Prop استفاده میشه. این متد زمانی که **shouldComponentUpdate** مقدار **false** رو برگردونه قابل استفاده ست.
- **componentWillUnmount**: این متد برای کنسل کردن همه درخواست‌های شبکه خروجی یا حذف همه event listenerهای مرتبط با کامپوننت استفاده میشه.

قبل ورژن ۱۶.۳

- **componentWillMount**: قبل از رندر اجرا میشه و برای پیکربندی سطح برنامه توی کامپوننت ریشه استفاده میشه.
- **componentDidMount**: بعد از اولین رندر اجرا میشه و همه درخواست‌های AJAX، DOM یا بروزرسانی state و تنظیمات event listeners اجرا میشه.

- **componentWillReceiveProps:** Executed when particular prop updates to trigger state transitions.
- **shouldComponentUpdate:** تعیین می‌کند که کامپوننت به روز بشه یا نه. به طور پیش فرض مقدار `true` رو برمی‌گردونه. اگه مطمئن باشیم که کامپوننت بعد از اینکه state یا props به روزرسانی میشه نیازی به رندر شدن نداره، می‌تونیم مقدار `false` رو برگردونیم. اینجا جای خوبی برای بهبود عملکرده چون این امکان رو بهمون میده که اگه کامپوننت prop جدید میگیره از render مجدد جلوگیری کنیم.
- **componentWillUpdate:** قبل از رندر مجدد کامپوننت وقتی که تغییرات state و props توسط `shouldComponentUpdate` مقدار `true` رو برگردونده باشه اجرا میشه.
- **componentDidUpdate:** بیشتر برای به روزرسانی DOM در پاسخ به تغییرات state یا Prop استفاده میشه. این متد زمانی که `shouldComponentUpdate` مقدار `false` رو برگردونه قابل استفاده ست.
- **componentWillUnmount:** این متد برای کنسل کردن همه درخواست‌های شبکه خروجی یا حذف همه event listenerهای مرتبط با کامپوننت استفاده میشه.

↑ فهرست مطالب

35. کامپوننت‌های Higher-Order چی هستن؟

کامپوننت با مرتبه بالا (HOC) تابعیه که یه کامپوننت می‌گیره و یه کامپوننت جدید برمی‌گردونه. اصولاً این الگویی که از ماهیت تلفیقی ری‌اکت گرفته شده. ما اینجا رو به عنوان کامپوننت‌های خالص می‌شناسیم چون می‌تونن هر کدوم از کامپوننت‌های فرزندشون رو که به صورت پویا ارائه شدن رو بپذیرن ولی هیچ کدوم از رفتارهای کامپوننت‌های ورودی خودشون رو تغییر نمیدن.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

HOC خیلی جاها می‌تونه استفاده بشه:

1. استفاده مجدد از کد، منطق و مفهوم bootstrap.
2. استفاده برای Render hijacking و کنترل خروجی رندر کامپوننت.
3. مفهوم state و دستکاری اون.
4. دستکاری prop‌ها.

36. چطوری می‌تونیم props proxy برای کامپوننت‌های HOC ایجاد کنیم؟

می‌تونیم prop‌های انتقال داده شده به کامپوننت رو با استفاده از الگوی *props proxy* اضافه یا ویرایش کنیم:

```
function HOC(WrappedComponent) {  
  return class Test extends Component {  
    render() {  
      const newProps = {  
        title: "New Header",  
        footer: false,  
        showFeatureX: false,  
        showFeatureY: true,  
      };  
  
      return <WrappedComponent {...this.props} {...newProps} />;  
    }  
  };  
}
```

37. context چیه؟

Context روشی رو برای انتقال داده بین کامپوننت‌ها فراهم می‌کنه بدون اینکه بخوایم توی هر سطح به صورت دستی داده‌ها رو منتقل کنیم. به عنوان مثال، معتبر بودن کاربر، چند زبانی و فایل‌های زبان، قالب UI مواردی هستن که توی خیلی از کامپوننت‌ها و به شکل عمومی لازم داریم که در دسترس باشن و می‌تونیم از *context* برای مدیریت‌شون استفاده کنیم.

```
const { Provider, Consumer } = React.createContext(defaultValue);
```

38. children prop چیه؟

children به prop هستند که بهمون اجازه میده کامپوننت‌ها رو به عنوان فرزند و به شکل داده به کامپوننت‌های دیگه انتقال بدیم (`prop.children`)، درست مثل prop‌های دیگه‌ای که استفاده می‌کنیم. درخت کامپوننت که بین تگ باز و بسته کامپوننت‌ها قرار داره به اون کامپوننت به عنوان `prop children` پاس داده میشه. به تعداد متد برای کار با این prop‌ها روی react API وجود داره که شامل `React.Children.map` ، `React.Children.forEach` ، `React.Children.count` ، `React.Children.only` ، `React.Children.toArray` میشه. به مثال ساده از استفاده از `children prop` این پایین نوشته شده.

```
const MyDiv = React.createClass({
  render: function () {
    return <div>{this.props.children}</div>;
  },
});

ReactDOM.render(
  <MyDiv>
    <span>{"Hello"}</span>
    <span>{"World"}</span>
  </MyDiv>,
  node
);
```

[↑ فهرست مطالب](#)

39. چطوری میشه تو React کامنت نوشت؟

کامنت‌ها توی React/JSX شبیه به جاوااسکریپت هستن اما کامنت‌های چند خطی توی آکولاد قرار میگیرن. کامنت‌های تک خطی:

```
<div>
  { /* Single-line comments(In vanilla JavaScript, the single-line
    {`Welcome ${user}, let's play React`}
  </div>
```

کامنت‌های چند خطی:

```
<div>
  { /* Multi-line comments for more than
    one line */ }
  { `Welcome ${user}, let's play React` }
</div>
```

[↑ فهرست مطالب](#)

40. چرا توی کامپوننت‌های کلاس باید توی constructor تابع super رو با مقدار props صدا بزنیم؟

کلاس constructor تا زمانی که متد super صدا زده نشده نمی‌تونه از this استفاده کنه. همین مورد در رابطه با کلاس‌های ES6 هم صدق می‌کنه. دلیل اصلی انتقال پارامترهای props به متد فراخوان super دسترسی داشتن به this.props توی constructor هستش.
با پاس دادن props:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    console.log(this.props); // prints { name: 'John', age: 42 }
  }
}
```

بدون پاس داده شدن props:

```
class MyComponent extends React.Component {
  constructor(props) {
    super();

    console.log(this.props); // prints undefined

    // but props parameter is still available
    console.log(props); // prints { name: 'John', age: 42 }
  }

  render() {
    // no difference outside constructor
    console.log(this.props); // prints { name: 'John', age: 42 }
  }
}
```

کد بالا نشون میده که `this.props` فقط توی `constructor` متفاوت عمل می‌کنه و بیرون از `constructor` عملکردش عادیه، دلیلش هم اینه که توی متد سازنده کلاس، هنوز `instance` کامل ساخته نشده و در حال ساخته شدن.

↑ فهرست مطالب

41. `reconciliation` چیه؟

وقتی `state` یا `props` به کامپوننت تغییر می‌کنه، ری‌اکت با مقایسه عنصر تازه `return` شده و نمونه `render` شده قبلی تصمیم می‌گیره که به روزرسانی `DOM` واقعا ضروریه یا نه. وقتی این دو مقدار با هم برابر نباشه، ری‌اکت به روزرسانی `DOM` رو انجام میده. به این فرایند `reconciliation` گفته میشه.

↑ فهرست مطالب

42. چطوری با `set state` داینامیک کنیم؟

اگر برای تبدیل کد `JSX` از `ES6` یا `babel` استفاده می‌کنین می‌تونید این کار رو با `computed property names` انجام بدین.

```
handleInputChange(event) {  
  this.setState({ [event.target.id]: event.target.value })  
}
```

اینجا ما یه فیلد از `Object` رو به شکل متغیر داریم پر می‌کنیم، البته روی فانکشن کامپوننت‌ها و با استفاده از هوک `useState` هم میشه به شکل داینامیک یه `object` رو پر کرد و فقط لازمه یه `state` به شکل `object` داشته باشیم:

```
const [myState, setMyState] = useState();  
const handleInputChange = (event) => {  
  setMyState({ [event.target.id]: event.target.value });  
}
```

↑ فهرست مطالب

43. یه اشتباه رایج برای مدیریت توابع eventها که باعث میشه با هر رندر توابع مجدد ساخته بشن چی هستش؟

باید مطمئن باشیم که موقع استفاده از تابع هندلرمون به عنوان پارامتر، اون تابع صدا زده نشه. مثلا:

```
// Wrong: handleClick is called instead of passed as a reference!  
return <button onClick={this.handleClick()}>{'Click Me'}</button>
```

و به جاش تابع رو بدون پرانتز (فراخوانی) و به شکل رفرنس پاس بدیم:

```
// Correct: handleClick is passed as a reference!  
return <button onClick={this.handleClick}>{'Click Me'}</button>
```

↑ فهرست مطالب

44. تابع lazy که برای lazy load استفاده میشه رو می‌تونیم به صورت name export خروجی بگیریم؟

نه، تابع `React.lazy` در حال حاضر فقط خروجی پیش فرض (default export) رو پشتیبانی می‌کنه. اگه بخوایم ماژول‌هایی رو `import` کنیم که به شکل پیش فرض `exports` نشدن، می‌تونیم یه ماژول واسطه تعریف کنیم که اونا رو به عنوان پیش فرض مجدداً تعریف می‌کنه. همچنین تضمین می‌کنه که `tree shaking` همچنان به کار خودش ادامه میده و کامپوننت موردنظر کدهای استفاده نشده رو نمی‌گیره. بیا یه کامپوننتی بنویسیم که چندین کامپوننت رو به عنوان خروجی ارائه میده.

```
// MoreComponents.js  
export const SomeComponent = /*... */;  
export const UnusedComponent = /*... */;
```

و کامپوننت `MoreComponents.js` رو در فایل واسطه `IntermediateComponent.js` مجدداً به عنوان خروجی تعریف کنیم.

```
// IntermediateComponent.js  
export { SomeComponent as default } from "./MoreComponents.js";
```

حالا می‌تونیم ماژول خودمون رو با استفاده از تابع `lazy` به شکل زیر `import` کنیم.

```
import React, { lazy } from "react";  
const SomeComponent = lazy(() => import("./IntermediateComponent.";
```

45. چرا ری اکت از `className` بجای `class` استفاده می‌کند؟

`class` به کلمه کلیدی توی جاوااسکریپت و فایل با پسوند JSX در حقیقت همون فایل جاوااسکریپت. دلیل اصلی استفاده ری اکت از `className` به جای `class` همینه و به مقدار رشته‌ای رو به عنوان پارامتر `className` پاس میدیم. مثل کد زیر:

```
render() {  
  return <span className='menu navigation-menu'>Menu</span>  
}
```

46. `fragment`ها چی هستن؟

به الگوی مشخص توی ری اکت وجود داره که برای کامپوننت‌هایی استفاده میشه که چندین عنصر یا کامپوننت رو برمیگردونن. `_Fragment`ها این امکان رو فراهم می‌کنن که بتونیم لیستی از فرزندان رو بدون اضافه کردن نودهای اضافی به DOM گروه بندی کنیم.

```
render() {  
  return (  
    <React.Fragment>  
      <ChildA />  
      <ChildB />  
      <ChildC />  
    </React.Fragment>  
  )  
}
```

همچنین به حالت مختصرتر هم وجود داره که به شکل زیر می‌تونیم `fragment` بسازیم:

```
render() {  
  return (  
    <>  
      <ChildA />  
      <ChildB />  
      <ChildC />  
    </>  
  )  
}
```

47. چرا fragment ها از تگ های div بهترن؟

البته می دونیم که نه فقط div بلکه از بقیه تگ های html هم میشه بجای fragment استفاده کرد ولی به دلایل زیر بهتره از fragment استفاده بشه:

1. Fragment ها یه کم سریعترن و با ایجاد نکردن DOM node اضافی حافظه کمتری استفاده می کنن. این فقط روی node های بزرگ و درخت های بزرگ و عمیق مزیت داره.

2. بعضی از مکانیزم های CSS مثل *Flexbox* و *CSS Grid* روابط والد و فرزندی خاصی دارند و اضافه کردن div در وسط، حفظ طرح مورد نظرمون را دشوار می کنه.

3. DOM Inspector بهم ریختگی کمتری داره و میشه راحت تر کدهای برنامه رو دیباگ کرد.

48. توی ری اکت portal ها چیکار می کنن؟

Portal روشی توصیه شده برای رندر کردن کامپوننت فرزند به شکل DOM و خارج از سلسله مراتب DOM کامپوننت والد هستش.

```
ReactDOM.createPortal(child, container);
```

اولین آرگومان یه فرزند قابل رندر شدن هستش، مثل عنصر، رشته، یا *fragment*. آرگومان دوم عنصر DOM هستش.

49. کامپوننت stateless چیه؟

اگه رفتار یه کامپوننت مستقل از state اون کامپوننت باشه بهش کامپوننت *stateless* گفته میشه. می تونیم از یه تابع یا یه کلاس ساخت کامپوننت های *stateless* استفاده کنیم،

50. کامپوننت stateful چیه؟

اگه رفتار یه کامپوننتی به *state* اون کامپوننت وابسته باشه، به عنوان کامپوننت statefull شناخته میشه.

```
const App = () => {  
  const [count, setCount] = useState(0);  
  
  return (  
    //...  
  );  
}
```

قبل از نسخه 16.8 ری اکت:

قبل از اینکه هوک‌ها این امکان رو بهمون بدن که بتونیم از *state* و ویژگی‌های دیگه ری اکت استفاده کنیم بدون نوشتن کلاس کامپوننت‌ها استفاده کنیم، نمی‌تونستیم فانکشن کامپوننت رو statefull کنیم و مجبور بودیم برای کامپوننت ساده فوق یه همچین کلاسی بنویسیم:

```
class App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  
  
  render() {  
    // ...  
  }  
}
```

51. چطوری prop های کامپوننت رو اعتبارسنجی کنیم؟

وقتی برنامه توی حالت development یا در حال توسعه هست، ری اکت به شکل خودکار تمام prop هایی که ما توی کامپوننت استفاده کردیم رو چک می‌کنه تا مطمئن بشه همه‌شون نوع درستی دارن. اگه هر کدوم از prop ها type درستی نداشته باشن توی کنسول بهمون یه warning نشون میده، البته توی حالت production این حالت غیر فعاله.

prop های اجباری با پراپرتی `isRequired` مشخص میشن، همچنین یه سری انواع prop از پیش تعریف شده وجود دارن که می‌تونیم ازشون استفاده کنیم:

PropTypes.number .1

PropTypes.string .2

PropTypes.array .3

PropTypes.object .4

PropTypes.func .5

PropTypes.node .6

PropTypes.element .7

PropTypes.bool .8

PropTypes.symbol .9

PropTypes.any .10

`PropType` ها رو برای یه کامپوننت تستی به اسم `User` اینطوری میشه تعریف کرد:

```
import React from "react";
import PropTypes from "prop-types";

const User = (props) => {
  return (
    <>
      <h1>{`Welcome, ${props.name}`}</h1>
      <h2>{`Age, ${props.age}`}</h2>
    </>
  );
}

User.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired,
};
```

نکته: در ورژن 15.5 ری‌اکت `propType` ها از `React.PropTypes` به کتابخونه جدید `prop-types` انتقال پیدا کردن.

[↑ فهرست مطالب](#)

52. مزایای React چیه؟

1. افزایش عملکرد برنامه با `Virtual DOM`.
2. خوندن و نوشتن راحتتر کدها با `JSX`.

3. امکان رندر شدن در هر دو سمت کاربر و سرور (SSR).
4. ادغام راحت با فریم ورک‌ها (Angular, Backbone).
5. امکان نوشتن تست‌های واحد یا ادغام شده از طریق ابزارهایی مثل Jest.

↑ فهرست مطالب

53. محدودیت‌های React چیه؟

1. ری‌اکت یک کتابخونه برای ساخت لایه view هستش نه یک فریم‌ورک کامل.
2. وجود یک منحنی یادگیری (سختی یادگیری یا همون learning curve) برای کسانی که به تازگی می‌خوان برنامه نویسی وب رو یاد بگیرن.
3. یکپارچه‌سازی ری‌اکت در فریم‌ورک‌های مبتنی بر MVC به یه کانفیگ اضافه‌ای نیاز داره.
4. پیچیدگی کد با inline templating و JSX افزایش پیدامی‌کنه.
5. خیلی کامپوننت‌های کوچیک یا boilerplate‌های کوچیک براش ساخته شدن و ممکنه کمی گیج‌کننده باشه.

↑ فهرست مطالب

54. error boundary ها توی ری‌اکت نسخه 16 چیکار می‌کنن؟

Error boundary ها یا به اصطلاح تحت الفظی مرزهای خطا کامپوننت‌هایی هستن که خطاهای جاوااسکریپت رو هرجایی توی درخت فرزنداش رخ داده باشن catch می‌کنن و خطای موردنظر رو log می‌کنن و علاوه براین می‌تونن یه UI به اصطلاح fallback رو بجای کامپوننت crash شده نشون بدن.

توی یه کلاس کامپوننت با گذاشتن متد `componentDidCatch(error, info)` یا `static getDerivedStateFromError` می‌تونیم یه *boundary* برای زمانی که خطایی رخ میده درست کنیم. مثل:

```

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch(error, info) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>{"Something went wrong."}</h1>;
    }
    return this.props.children;
  }
}

```

بعدشم همیشه ارزش مثل یه کامپوننت عادی استفاده کرد:

```

<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>

```

نکته: از این قابلیت با استفاده از کامپوننت‌های functional همیشه استفاده کرد و در حقیقت احتمالا نیازی هم بهش ندارین، چون اکثر مواقع برای کل برنامه یه `error boundary` تعریف می‌کنیم که می‌تونه `try..catch` باشه.

↑ فهرست مطالب

55. ### چطوری از `error boundary` ها توی نسخه 15 ری‌اکت استفاده کنیم؟
 ری‌اکت توی نسخه 15 با استفاده از متد `unstable_handleError` `error boundary` ها رو مدیریت کرده.
 این متد توی نسخه 16 به `componentDidCatch` تغییر کرده.

56. روش‌های پیشنهادی برای type checking چیست؟

به طور معمول به دو روش همیشه نوع prop ورودی در برنامه‌های ری‌اکتی رو چک کرد. روش اول استفاده از کتابخانه prop-types و اعتبارسنجی ورودی‌های کامپوننت‌ها و روش دوم که برای برنامه‌هایی با کدهای بیشتر توصیه می‌شود، استفاده از static type checker مثل flow یا TypeScript هست که چک کردن نوع داده رو در زمان توسعه و کامپایل انجام می‌دهد و ویژگی‌های مثل auto-completion رو ارائه می‌دهد.

57. کاربرد پکیج react-dom چیست؟

پکیج react-dom متدهای DOM-specific یا مخصوص DOM رو ارائه می‌دهد که می‌تونید توی سطوح بالای برنامه شما استفاده بشه. اکثر کامپوننت‌ها نیازی به استفاده از این ماژول‌ها ندارن. تعدادی از متدهای این پکیج این‌ها هستن:

1. متد render

2. متد hydrate

3. متد unmountComponentAtNode

4. متد findDOMNode

5. متد createPortal

58. کاربرد متد render از پکیج react-dom چیست؟

این متد برای رندرکردن کامپوننت پاس داده شده، توی یه المنت DOM که به عنوان container پاس داده شده، استفاده می‌شود و یه رفرنس به کامپوننت برمی‌گردونه. اگه کامپوننت ری‌اکت قبلاً توی container مورد نظر رندر شده باشه با یه update فقط DOM‌هایی که نیازمند به روز شدن باشن رو ری‌رندر می‌کنه.

```
ReactDOM.render(element, container[, callback])
```

اگر پارامتر سوم که به callback هست پاس داده بشه، هر موقع که رندر یا بهروزرسانی انجام بشه اون تابع هم اجرا میشه.

[↑ فهرست مطالب](#)

59. ReactDOMServer چیه؟

ReactDOMServer این امکان رو بهمون میده که کامپوننت‌ها رو به صورت استاتیک رندر کنیم (معمولا روی node server استفاده میشه). ReactDOMServer عمدتاً برای پیاده سازی سمت سرور استفاده میشه (SSR).

1. متد `renderToString`

2. متد `renderToStaticMarkup`

برای مثال ممکنه یه سرور روی node بسازین که ممکنه Express، Hapi یا Koa باشه و متد `renderToString` رو برای تبدیل کردن کامپوننت root به html اجرا کنید و نتیجه بدست اومده رو به عنوان response به کلاینت پاس بدین.

```
// using Express
import { renderToString } from "react-dom/server";
import MyPage from "./MyPage";

app.get("/", (req, res) => {
  res.write(
    "<!DOCTYPE html><html><head><title>My Page</title></head><body>"
  );
  res.write('<div id="content">');
  res.write(renderToString(<MyPage />));
  res.write("</div></body></html>");
  res.end();
});
```

[↑ فهرست مطالب](#)

60. چطوری از InnerHtml توی ری اکت استفاده کنیم؟

ویژگی `dangerouslySetInnerHTML` جایگزین ری اکت واسه استفاده از `innerHTML` توی DOM مرورگره و کارکردش درست مثل `innerHTML` هستش، استفاده از این ویژگی به خاطر حملات (cross-site-scripting) XSS ریسک بالایی داره.

برای این کار باید به آجکت `innerHTML` به عنوان `key` و به متن `html` به عنوان `value` به این `prop` بفرستیم (یا شاید همون پاس بدیم).
توی مثال پایینی کامپوننت از ویژگی `dangerouslySetInnerHTML` برای قرار دادن `HTML` استفاده کرده.

```
function createMarkup() {  
  return { __html: "First &middot; Second" };  
}  
  
function MyComponent() {  
  return <div dangerouslySetInnerHTML={createMarkup()} />;  
}
```

[↑ فهرست مطالب](#)

61. چطوری توی ری اکت استایل دهی می کنیم؟

`attribute` پیش فرض مورد استفاده برای استایل دهی `style` هستش که به `object` جاوااسکریپت رو به عنوان مقدار ورودی دریافت می کنه. همه `property` های اون بجای `css` عادی `camelCase` هستن. این روش با استایل دهی عادی توی جاوااسکریپت به کم متفاوت، بهینه تر و امن تره، چون جلوی حفره های امنیتی مثل `XSS` رو میگیره.

```
const divStyle = {  
  color: "blue",  
  backgroundImage: "url(" + imgUrl + ")",  
};  
  
function HelloWorldComponent() {  
  return <div style={divStyle}>Hello World!</div>;  
}
```

[↑ فهرست مطالب](#)

62. تفاوت event های ری اکت چیه؟

مدیریت رویدادها روی اِلمان های ری اکت به سری تفاوت های کلی با نحوه مدیریت اونا روی `js` داره:

1. `event handler` های ری اکت به جای حروف کوچک به صورت حروف بزرگ نام گذاری میشن.

2. با JSX ما به تابع رو به جای رشته به عنوان event handler پاس میدیم.

↑ فهرست مطالب

63. اگه توی constructor بیاییم و setState کنیم چی میشه؟

وقتی از `setState` استفاده می‌کنیم، جدا از اینکه به یه آبجکت استیتی اختصاص داده میشه ری‌اکت اون کامپوننت و همه فرزندای اون کامپوننت رو دوباره رندر می‌کنه. ممکنه این ارور رو بگیرین: شما فقط می‌تونید کامپوننت `mount` شده یا در حال `mount` رو به روز رسانی کنید. پس باید بجای `setState` از `this.state` برای مقداردهی state توی constructor استفاده کنیم.

توی فانکشن کامپوننت‌ها هم اگه داخل بدنه تابع یه `setState` کنیم، کامپوننت توی حلقه رندر بی‌نهایت می‌افته و خطا می‌گیریم.

↑ فهرست مطالب

64. تاثیر استفاده از ایندکس به عنوان key چیه؟

`key`ها باید پایدار، قابل پیش بینی و منحصر به فرد باشن تا ری‌اکت بتونه اِلمان‌ها رو رهگیری کنه.

تو کد زیر `key` هر عنصر براساس ترتیبی که توی لیست داره مقدار قرار می‌گیره و به داده‌هایی که میگیرن ربطی نداره. این کار بهینه سازی‌هایی که می‌تونه توسط ری‌اکت انجام بشه رو محدود می‌کنه.

```
todos.map((todo, index) => <Todo {...todo} key={index} />);
```

اگه از داده‌های همون `element` به عنوان کلید بخوایم استفاده کنیم، مثلاً `todo.id`. چونکه همه ویژگی‌هایی که یه کلید باید داشته باشه رو داره، هم استیبله و هم منحصر به فرد، توی این حالت ری‌اکت می‌تونه بدون اینکه لازم باشه دوباره همه اِلمان‌ها رو ارزیابی کنه رندر رو انجام بده.

```
todos.map((todo) => <Todo {...todo} key={todo.id} />);
```

↑ فهرست مطالب

65. نظرت راجع به استفاده از `setState` توی متد `componentWillMount` چیه؟

توصیه همیشه که از مقدار دهی اولیه غیرهمزمان (async) در متد `componentWillMount` استفاده نشه. `componentWillMount` درست قبل از mount شدن اجرا میشه و اون لحظه قبل از فراخوانی متد `render` هست، پس `setState` کردن توی این متد باعث re-render شدن نمیشه. باید از ایجاد هر سایید افکتی توی این متد خودداری کنیم و دقت کنیم که اگه مقدار دهی اولیه غیر همزمان ای داریم این کار رو توی متد `componentDidMount` انجام بدیم نه در متد `componentWillMount`.

```
componentDidMount() {
  axios.get(`api/todos`)
    .then((result) => {
      this.setState({
        messages: [...result.data]
      })
    })
}
```

معادل کد زیر با هوک:

```
useEffect(() => {
  axios.get(`api/todos`)
    .then((result) => {
      setMessages([...result.data])
    })
}, []);
```

↑ فهرست مطالب

66. اگه از `prop` توی مقداردهی اولیه `state` استفاده کنیم چی میشه؟

اگه `prop` های یه کامپوننت بدون اینکه اون کامپوننت رفرش بشه تغییر کنه، مقدار جدید اون `prop` نمایش داده نمیشه چون `state` جاری اون کامپوننت رو به روز رسانی نمی کنه، مقدار دهی اولیه `state` از `prop` ها فقط زمانی که کامپوننت برای بار اول ساخته شده اجرا میشه. کامپوننت زیر مقدار به روزرسانی شده رو نشون نمیده:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      records: [],
      inputValue: this.props.inputValue,
    };
  }

  render() {
    return <div>{this.state.inputValue}</div>;
  }
}
```

و نکته جالبش اینه که استفاده از prop ها توی متد render مقدار رو به روز رسانی می‌کنه:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      record: [],
    };
  }

  render() {
    return <div>{this.props.inputValue}</div>;
  }
}
```

توی فانکشن کامپوننت‌ها هم دقیقاً به همین شکل هست، مقدار اولیه useState اگه از prop ورودی کامپوننت باشه، با تغییر دادن prop مقدار اون state عوض نمیشه، دلیلش هم اینه که هوک useState فقط توی اولین رندر اجرا میشه و بعدش دیگه باید با استفاده از متد setter مقدار اون state رو عوض کنیم.

[↑ فهرست مطالب](#)

67. چطوری کامپوننت رو با بررسی یه شرط رندر می‌کنیم؟

بعضی وقتا ما می‌خوایم کامپوننت‌های مختلفی رو بسته به بعضی state ها رندر کنیم. JSX مقدار `false` یا `undefined` رو رندر نمی‌کنه، بنابراین ما می‌تونیم از short-circuiting

شرطی برای رندر کردن بخش مشخصی از کامپوننت مون استفاده کنیم در صورتی که اون شرط مقدار true رو برگردونده باشه.

```
const MyComponent = ({ name, address }) => (  
  <div>  
    <h2>{name}</h2>  
    {address && <p>{address}</p>}  
  </div>  
);
```

اگه به یه شرط if-else نیاز داریم، میتونیم از عبارت شرطی سه گانه (ternary operator) استفاده کنیم:

```
const MyComponent = ({ name, address }) => (  
  <div>  
    <h2>{name}</h2>  
    {address ? <p>{address}</p> : <p>{"Address is not available"}</p>}  
  </div>  
);
```

↑ فهرست مطالب

68. چرا وقتی prop ها رو روی یه DOM Element می آیم spread می کنیم باید مراقب باشیم؟

وقتی ما prop ها رو spread می کنیم این کارو با ریسک اضافه کردن اتریبیوت های HTML انجام میدیم که این کار خوبی نیست، به جای این کار می تونیم از rest... استفاده کنیم که فقط prop های مورد نیاز رو اضافه می کنه.

```
const ComponentA = () => (  
  <ComponentB isDisplay={true} className={"componentStyle"} />  
);  
  
const ComponentB = ({ isDisplay, ...domProps }) => (  
  <div {...domProps}>{"ComponentB"}</div>  
);
```

↑ فهرست مطالب

69. چطوری از decorator ها توی ری اکت استفاده کنیم؟

می‌تونیم کلاس کامپوننت‌ها رو decorate کنیم، که درست مثل پاس دادن کامپوننت‌ها به تابع هستش. **Decorator**‌ها روش قابل خواندن و انعطاف پذیرتری برای تغییر فانکشنالیتی کامپوننت‌ها هستن.

```
@setTitle("Profile")
class Profile extends React.Component {
  //....
}

/*
  title is a string that will be set as a document title
  WrappedComponent is what our decorator will receive when
  put directly above a component class as seen in the example above
*/
const setTitle = (title) => (WrappedComponent) => {
  return class extends React.Component {
    componentDidMount() {
      document.title = title;
    }

    render() {
      return <WrappedComponent {...this.props} />;
    }
  };
};
```

نکته: Decorator‌ها ویژگی‌هایی هستن که در حال حاضر به ES7 اضافه نشدن، ولی توی پیشنهاد stage 2 هستن.

[↑ فهرست مطالب](#)

70. چطوری یه کامپوننت رو memoize می‌کنیم؟

در حال حاضر کتابخانه‌هایی وجود داره که با هدف memoize کردن ایجاد شدن و می‌تونن توی کامپوننت‌های تابع استفاده بشن، به عنوان مثال کتابخونه `moize` می‌تونه یه کامپوننت رو توی بقیه کامپوننت‌ها memoize کنه.

```
import moize from "moize";
import Component from "../components/Component"; // this module exports Component

const MemoizedFoo = moize.react(Component);

const Consumer = () => {
  <div>
    {"I will memoize the following entry:"}
    <MemoizedFoo />
  </div>;
};
```

به روز رسانی: توی ورژن 16.6.0 ری اکت، `React.memo` رو داریم که کارش اینه که یه کامپوننت با الویت بالاتر فراهم می‌کنه که کامپوننت رو تا زمانی که prop‌ها تغییر کنن، `memoize` می‌کنه. برای استفاده ازش کافیه زمان ساخت کامپوننت از `React.memo` استفاده کنیم.

```
const MemoComponent = React.memo(function MemoComponent(props) {
  /* render using props */
});
// OR
export default React.memo(MyFunctionComponent);
```

[↑ فهرست مطالب](#)

71. چطوری باید Server-Side Rendering یا SSR رو توی ری اکت پیاده کنیم؟

ری اکت در حال حاضر به رندر سمت نود سرور مجهزه، یه ورژن خاصی از DOM رندر در دسترسه که دقیقاً از همون الگوی سمت کاربر پیروی می‌کنه.

```
import ReactDOMServer from "react-dom/server";
import App from "../App";

ReactDOMServer.renderToString(<App />);
```

خروجی این روش یه HTML معمولی به صورت یه رشته‌ست که داخل `body` صفحه به عنوان `response` سرور قرار می‌گیره.

در سمت کاربر، ری اکت محتوای از قبل رندر شده رو تشخیص میده و به صورت فرآیند همگام‌سازی با اونا رو انجام میده (rehydration).

72. چطوری حالت production رو برای ری اکت فعال کنیم؟

میشه از پلاگین `DefinePlugin` که روی وب پیک قابل استفاده هست استفاده کرد و مقدار `NODE_ENV` رو روی `production` ستکرد، با اینکار خطاهای اضافی یا اعتبارسنجی `propTypes` ها روی پروداکشن غیرفعال میشه و جدای این موارد، کدهای نوشته شده بهینه سازی میشن و مثلاً کدهای بلااستفاده حذف میشن، کم حجم سازی انجام میشه و درنتیجه سرعت بهتری رو می تونه به برنامه بده چون سایز `bundle` ایجاد شده کوچیکتر خواهد بود.

73. CRA چیه و چه مزایایی داره؟

ابزار CLI (محیط کدهای دستوری) `create-react-app` این امکان رو بهمون میده که برنامه های ری اکت رو سریع و بدون مراحل پیگیری بسازیم و اجرا کنیم. مثلاً، بیان برنامه `Todo` رو با استفاده از `CRA` بسازیم:

```
# Installation
$ npm install -g create-react-app

# Create new project
$ create-react-app todo-app
$ cd todo-app

# Build, test and run
$ npm run build
$ npm run test
$ npm start
```

این شامل همه اون چیزیه که ما واسه ساختن یه برنامه ری اکت لازم داریم:

1. `ES6`، `JSX`، `React` و روند پشتیبانی `syntax`.
2. موارد اضافی زبان شامل `ES6` و عملگر `object spread` و اینا.
3. `Autoprefixed CSS`، بنابراین نیازی به `webpack`- یا پیشنوندهای دیگه ای نداریم.
4. یه اجرا کننده تست تعاملی با پشتیبانی داخلی برای `coverage reporting`.
5. یه سرور `live development` که اشتباهات معمول رو بهمون هشدار میده.

6. به اسکریپت بیلد برای پک و باندل کردن js، css، و تصاویر برای production همراه با sourcemap و hashها.

↑ فهرست مطالب

74. ترتیب اجرا شدن متدهای life cycle چگونه؟

وقتی به نمونه‌ای از کامپوننت ساخته میشه و داخل DOM اضافه میشه، متدهای lifecycle به ترتیب زیر صدا زده میشن.

1. متد `constructor`

2. متد `static getDerivedStateFromProps`

3. متد `render`

4. متد `componentDidMount`

↑ فهرست مطالب

75. کدام متدهای life cycle توی نسخه 16 ری‌اکت منسوخ شدن؟

متدهای lifecycle روش‌های ناامن کدنویسی هستن و با رندر async مشکل بیشتری پیدا می‌کنن.

1. متد `componentWillMount`

2. متد `componentWillReceiveProps`

3. متد `componentWillUpdate`

تو ورژن 16.3 ری‌اکت این متدها با پیشوند `_UNSAFE` متمایز شدن و تو نسخه 17 ری‌اکت حذف شد.

↑ فهرست مطالب

76. کاربرد متد `getDerivedStateFromProps` چیه؟

بعد از اینکه به کامپوننت بلافاصله بدون خطا و مثل قبل rerender شد، متد استاتیک `getDerivedStateFromProps` صدا زده میشه.

این متد یا state آپدیت شده رو به صورت یه آبجکت برمی گردونه یا null رو برمی گردونه که معنیش اینه prop های جدید به آپدیت شدن state نیازی ندارن.

```
class MyComponent extends React.Component {
  static getDerivedStateFromProps(props, state) {
    //...
  }
}
```

متد `componentDidUpdate` تمام مواردی که توی متد `componentWillReceiveProps` هست رو پوشش میده.

[↑ فهرست مطالب](#)

77. کاربرد متد `getSnapshotBeforeUpdate` چیه؟

متد جدید `getSnapshotBeforeUpdate` بعد از آپدیت های DOM صدا زده میشه. مقدار برگشتی این متد به عنوان پارامتر سوم به متد `componentDidUpdate` پاس داده میشه.

```
class MyComponent extends React.Component {
  getSnapshotBeforeUpdate(prevProps, prevState) {
    //...
  }
}
```

متد `componentDidUpdate` تمام مواردی که توی متد `componentWillUpdate` استفاده میشه رو پوشش میده.

[↑ فهرست مطالب](#)

78. آیا هوک ها جای `render props` و `HOC` رو می گیرن؟

هوک ها می تونن بسیاری از نیازهای ما رو موقع تولید کامپوننت های ری اکتی حل کنن. کامپوننت های با اولویت بالاتر و `render prop` ها هر دوشون فقط یه `child` رو رندر می کنن ولی هوک ها روش راحت تری رو ارائه میدن که از تودرتو بودن درخت کامپوننت ها جلوگیری می کنه.

79. روش توصیه شده برای نام‌گذاری کامپوننت‌ها چیه؟

برای نام‌گذاری کامپوننت‌ها توصیه می‌شه که از اسم هنگام `export` گرفتن به جای `displayName` استفاده کنیم.
استفاده از `displayName` برای نام‌گذاری کامپوننت:

```
export default React.createClass({
  displayName: "TodoApp",
  //...
});
```

روش توصیه شده:

```
const TodoApp = () => ();

export default TodoApp;
```

80. روش توصیه شده برای ترتیب متدها در کلاس کامپوننت‌ها چیه؟

ترتیب توصیه شده متدها از *mounting* تا *render stage*:

1. متدهای `static`
2. متد `constructor`
3. متد `getChildContext`
4. متد `componentWillMount`
5. متد `componentDidMount`
6. متد `componentWillReceiveProps`
7. متد `shouldComponentUpdate`
8. متد `componentWillUpdate`
9. متد `componentDidUpdate`
10. متد `componentWillUnmount`
11. event handler مثل `onClickSubmit` یا `onChangeDescription`

12. متدهای دریافت کننده برای رندر مثل `getSelectReason` یا

`getFooterContent`

13. متدهای رندر اختیاری مثل `renderNavigation` یا

`renderProfilePicture`

14. متد `render`

[↑ فهرست مطالب](#)

81. کامپوننت تعویض کننده یا switching چیه؟

یه کامپوننت *switcher* کامپوننتی‌ه که یکی از چندتا کامپوننت موردنظر رو رندر می‌کنه. لازمه که برای تصمیم گیری بین کامپوننت‌ها از `object` جاوااسکریپتی استفاده کنیم. برای مثال، کدپایین با بررسی `prop` موردنظر `page` بین صفحات مختلف سوییچ می‌کنه:

```
import HomePage from "./HomePage";
import AboutPage from "./AboutPage";
import ServicesPage from "./ServicesPage";
import ContactPage from "./ContactPage";

const PAGES = {
  home: HomePage,
  about: AboutPage,
  services: ServicesPage,
  contact: ContactPage,
};

const Page = (props) => {
  const Handler = PAGES[props.page] || ContactPage;

  return <Handler {...props} />;
};

// The keys of the PAGES object can be used in the prop types to
Page.propTypes = {
  page: PropTypes.oneOf(Object.keys(PAGES)).isRequired,
};
```

[↑ فهرست مطالب](#)

82. چرا نیاز همیشه به تابع `setState` به فانکشن `callback` پاس بدیم؟

دلیلش اینه که `setState` به عملیات `async` یا ناهمزمانه. `state`ها در ری اکت به دلایل عملکردی تغییر می کنن، بنابراین به `state` ممکنه بلافاصله بعد از اینکه `setState` صدا زده شد تغییر نکنه. یعنی اینکه وقتی `setState` رو صدا می زنیم نباید به `state` جاری اعتماد کنیم چون نمی تونیم مطمئن باشیم که اون `state` چی می تونه باشه. راه حلش اینه که به تابع رو با `state` قبلی به عنوان یه آرگومان به `setState` پاس بدیم. بیان فرض کنیم مقدار اولیه `count` صفر هستش. بعد از سه عملیات پشت هم، مقدار `count` فقط یکی افزایش پیدا می کنه.

```
const [count, setCount] = useState(0);

setCount(count + 1);
setCount(count + 1);
setCount(count + 1);
// count === 1, not 3
```

اگه ما به تابع به `setState` پاس بدیم، مقدار `count` به درستی افزایش پیدا می کنه.

```
this.setState((prevState, props) => ({
  count: prevState.count + props.increment,
}));
// this.state.count === 3 as expected
```

↑ فهرست مطالب

83. حالت `strict` توی ری اکت چیکار می کنه؟

`React.StrictMode` به کامپوننت مفید برای هایلایت کردن مشکلات احتمالی توی برنامه ست. `<StrictMode>` درست مثل `<Fragment>` هیچ المان `DOM` اضافه ای رو رندر نمی کنه، بلکه `warning`ها و `additional checks` رو برای فرزندان اون کامپوننت فعال می کنه. این کار فقط در حالت `development` فعال میشه.

```
import React from "react";

function ExampleApplication() {
  return (
    <div>
      <Header />
      <React.StrictMode>
        <div>
          <ComponentOne />
          <ComponentTwo />
        </div>
      </React.StrictMode>
      <Footer />
    </div>
  );
}
```

توی مثال بالا، *strict mode* فقط روی دو کامپوننت `<ComponentOne>` و `<ComponentTwo>` اعمال میشه.

[↑ فهرست مطالب](#)

84. Mixin های ری اکت چی هستن؟

Mixin ها روشی برای جدا کردن کامپوننت هایی با عملکرد مشترک بودن. با توسعه یافتن ری اکت دیگه Mixin ها نباید استفاده بشن و می تونن با کامپوننت های با اولویت بالا (HOC) یا decorator ها جایگزین بشن.

یکی از بیشترین کاربردهای Mixin ها `PureRenderMixin` بود. ممکنه تو بعضی از کامپوننت ها برای جلوگیری از re-render های غیر ضروری وقتی prop ها و state با مقادیر قبلی شون برابر هستن از این Mixin ها استفاده کنیم:

```
const PureRenderMixin = require("react-addons-pure-render-mixin")

const Button = React.createClass({
  mixins: [PureRenderMixin],
  //...
});
```

نکته مهم: Mixin های ری اکت منقضی شدن و دیگه کاربردی ندارن، این سوال فقط برای افزایش آگاهی توی کتاب باقی می مونه.

85. چرا isMounted آنتی پترن هست و روش بهتر انجامش چیه؟

کاربرد اصلی متد `isMounted` برای جلوگیری از فراخوانی `setState` بعد از `unmount` شدن کامپوننت هستش چونکه باعث ایجاد یه خطا میشه. خطاش یه چیزی مثل اینه:

Warning: Can only update a mounted or mounting component. This usu

توی کلاس کامپوننت‌ها هم این شکلی بعضا جلوشو می‌گرفتن:

```
if (this.isMounted()) {  
  this.setState({...})  
}
```

دلیل اینکه این روش توصیه نمیشه اینه که خطایی رو که ری‌اکت بهمون میداد رو داره دور میزنه و حلش نمی‌کنه. بهتره `setState` رو جایی انجام بدیم که توی مواقعی که کامپوننت `mount` نیست اجرا نشه. البته توی نسخه‌های جدید ری‌اکت این کار رو خیلی ساده‌تر میشه انجام داد و فقط کافیه یه هوکی بنویسیم که یه `ref` رو مقداردهی می‌کنه و بعد با بررسی اون `ref` میشه فهمید که کامپوننت `mount` شده یا نه، مثلاً:

```
export const useIsMounted = () => {  
  const componentIsMounted = useRef(true);  
  useEffect(  
    () => () => {  
      componentIsMounted.current = false;  
    },  
    []  
  );  
  return componentIsMounted;  
};
```

یا حتی یه پکیجی ساخته شده به اسم `ismounted` که می‌تونه بهمون کمک کنه که متوجه بشیم کامپوننت `mount` شده یا نه. ولی حواسمون باشه که ازش درست استفاده کنیم.

86. پشتیبانی ری اکت از event pointerها چگونه؟

`EventPointer`ها به روش واحدی رو برای هندل کردن همه ی ایونت‌های ورودی ارائه میدن.

در زمان‌های قدیم ما از موس استفاده میکردیم و برای هندل کردن ایونت‌های مربوط به اون از `event listener`ها استفاده میکردیم ولی امروزه دستگاه‌های زیادی داریم که با داشتن موس ارتباطی ندارن، مثل قلم‌ها یا گوشی‌های صفحه لمسی. باید یادمون باشه که این ایونت‌ها فقط تو مرورگرهایی کار می‌کنن که مشخصه `Pointer Events` رو پشتیبانی می‌کنن.

ایونت‌های زیر در `React DOM` در دسترس هستن:

1. `onPointerDown`
2. `onPointerMove`
3. `onPointerUp`
4. `onPointerCancel`
5. `onGotPointerCapture`
6. `onLostPointerCapture`
7. `onPointerEnter`
8. `onPointerLeave`
9. `onPointerOver`
10. `onPointerOut`

↑ فهرست مطالب

87. چرا باید اسم کامپوننت با حرف بزرگ شروع بشه؟

اگه ما با استفاده از `JSX` کامپوننتمون رو رندر می‌کنیم، اسم کامپوننت باید با حرف بزرگ شروع بشه در غیر این صورت ری اکت خطای تگ غیر قابل تشخیص رو میده. این قرارداد به خاطر اینه که فقط عناصر `HTML` و تگ‌های `svg` می‌تونن با حرف کوچک شروع بشن.

```
} class SomeComponent extends Component
Code goes here //
{
```

می‌تونیم کلاس کامپوننت‌هایی که با حرف کوچک شروع میشن رو هم تعریف کنیم ولی وقتی داریم ایمپورت می‌کنیم باید شامل حروف بزرگ هم باشن:

```
class myComponent extends Component {
  render() {
    return <div />;
  }
}

export default myComponent;
```

وقتی داریم تو یه فایل دیگه‌ای ایمپورت می‌کنیم باید با حرف بزرگ شروع بشه:

```
import MyComponent from "./MyComponent";
```

[↑ فهرست مطالب](#)

88. آیا propهای custom توی ری‌اکت پشتیبانی میشن؟

بله. اون قدیما ری‌اکت attributeهای DOMهای ناشناخته رو نادیده می‌گرفت، اگه JSX رو با یه ویژگی‌ای نوشته بودیم که ری‌اکت تشخیص نمی‌داد، اونو نادیده می‌گرفت. به عنوان مثال:

```
<div mycustomattribute="something" />
```

در ری‌اکت ورژن 15 یه div خالی توی DOM رندر می‌کنیم:

```
<div />
```

در ری‌اکت ورژن 16 هر attribute ناشناخته‌ای توی DOM از بین میره:

```
<div mycustomattribute="something" />
```

این برای attributeهای غیر استاندارد مرورگرهای خاص، DOM APIهای جدید و ادغام با کتابخانه‌های third-party مفیده.

[↑ فهرست مطالب](#)

89. تفاوت‌های constructor و getInitialState چیه؟

وقتی داریم از کلاس‌های ES6 استفاده می‌کنیم باید state رو توی constructor مقداردهی اولیه کنیم و وقتی از `React.createClass` استفاده می‌کنیم باید از متد `getInitialState` استفاده کنیم.

استفاده از کلاس‌های ES6:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      /* initial state */
    };
  }
}
```

استفاده از `React.createClass`:

```
const MyComponent = React.createClass({
  getInitialState() {
    return {
      /* initial state */
    };
  },
});
```

نکته: `React.createClass` در ورژن 16 ری‌اکت حذف شده و به جای اون میشه از کلاس‌های ساده جاوااسکریپت استفاده کرد.

[↑ فهرست مطالب](#)

90. می‌تونیم یه کامپوننت رو بدون `setState` ری‌رندر کنیم؟

در حالت پیش فرض، وقتی `state` یا `prop` کامپوننت تغییر می‌کنه، کامپوننت دوباره رندر میشه. اگه متد `render` به داده‌های دیگه‌ای وابسته باشه، توی فانکشن کامپوننت‌ها می‌تونیم یه `state` تعریف کنیم و با ست کردن یه مقدار جدید توی اون `state` عامدانه باعث رندر مجدد کامپوننت بشیم. مثل:

```
const [tick, setTick] = useState(0);

const reRender = () => setTick(tick => tick++);
```

توی مثال بالا ما یه تابع تولید کردیم که با هر بار فراخوانی اون، می‌تونیم انتظار رندر شدن کامپوننت رو داشته باشیم. توی کلاس کامپوننت‌ها، می‌تونیم با فراخوانی متد `forceUpdate` به ری‌اکت بگیم که این کامپوننت نیازه که دوباره رندر بشه.


```
component.forceUpdate(callback);
```

توصیه همیشه که از متد `forceUpdate` استفاده نکنیم و توی `render` فقط از `this.props` و `this.state` استفاده کنیم.

↑ فهرست مطالب

91. تفاوت‌های فراخوانی `super()` و `super(props)` توی کلاس کامپوننت‌های ری‌اکت چیه؟

اگه بخوایم به `this.props` توی `constructor` دسترسی پیدا کنیم باید `prop`‌ها رو از طریق متد `super` پاس بدیم. استفاده از `super(props)`:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    console.log(this.props); // { name: 'John', ... }
  }
}
```

استفاده از `super`:

```
class MyComponent extends React.Component {
  constructor(props) {
    super();
    console.log(this.props); // undefined
  }
}
```

بیرون از `constructor` هر دو متد مقادیر یکسانی رو برای `this.props` نشون میدن.

↑ فهرست مطالب

92. چطوری توی `JSX` حلقه یا همون لوپ رو داشته باشیم؟

خیلی ساده می‌تونیم از `Array.prototype.map` با سینتکس `arrow` تابع `ES6` استفاده کنیم، برای مثال آرایه‌ای از آیتم‌های یه آبجکت توی آرایه‌ای از کامپوننت‌ها نوشته

```
<tbody>
  {items.map((item) => (
    <SomeComponent key={item.id} name={item.name} />
  ))}
</tbody>
```

نمی‌تونیم با استفاده از حلقه `for` تکرار رو انجام بدیم:

```
<tbody>
  for (let i = 0; i < items.length; i++) {
    <SomeComponent key={items[i].id} name={items[i].name} />
  }
</tbody>
```

به خاطر اینکه تگ‌های JSX داخل *function calls* تبدیل میشن ما نمی‌تونیم از *statement* داخل عبارات استفاده کنیم.

[↑ فهرست مطالب](#)

93. توی attribute‌ها چطوری به prop دسترسی داشته باشیم؟

رایکت و در حقیقت JSX داخل یه *attribute* استفاده از متغیر به شکل عادی رو پشتیبانی نمی‌کنه.

مثلا کد پایین کار نمی‌کنه:

```

```

اما ما می‌تونیم هر عبارت JS رو داخل کرلی براکت (`{ }`) به عنوان مقدار کلی *attribute* قرار بدیم.

مثلا، تکه کد پایین کار می‌کنه:

```
<img className="image" src={`images/` + props.image} />
```

با استفاده از *template strings* هم می‌تونیم بنویسیم:

```
<img className="image" src={`images/${this.props.image}`} />
```

[↑ فهرست مطالب](#)

94. چطوری به PropTypes برای آرایه‌ای از objectها با shape داشته باشیم؟

اگر بخواهیم آرایه‌ای از آبجکت‌ها رو به یه کامپوننت با شکل خاصی پاس بدیم، از `React.PropTypes.arrayOf` به عنوان یه آرگومان برای `React.PropTypes.shape` استفاده می‌کنیم.

```
ReactComponent.propTypes = {  
  arrayWithShape: React.PropTypes.arrayOf(  
    React.PropTypes.shape({  
      color: React.PropTypes.string.isRequired,  
      fontSize: React.PropTypes.number.isRequired,  
    })  
  ).isRequired,  
};
```

↑ فهرست مطالب

95. چطوری classهای یه المنت رو به صورت شرطی رندر کنیم؟

نیاید از کرلی براکت (`{}`) داخل کوتیشن (`' '`) استفاده کنیم چون به عنوان یه رشته در نظر گرفته میشه.

```
<div className="btn-panel {this.props.visible ? 'show': 'hidden'}">
```

به جاش می‌تونیم کرلی بریس رو به بیرون انتقال بدیم. (فراموش نکنیم که از space بین `className`ها استفاده کنیم.)

```
<div className={'btn-panel ' + (this.props.visible ? 'show': 'hid
```

با استفاده از *Template strings* هم می‌تونیم بنویسیم:

```
<div className={`btn-panel ${this.props.visible ? 'show': 'hidden
```

↑ فهرست مطالب

96. تفاوت‌های React و ReactDOM چیه؟

پکیج ری‌اکت شامل `React.createElement` ، `React.Component` ، `React.Children` و `helper` های دیگه که مربوطه به کلاس کامپوننت‌ها و المنت‌ها هستش. می‌تونیم اینا رو به عنوان `isomorphic` یا `universal helpers` که واسه ساختن کامپوننت‌ها نیاز داریم، در نظر بگیریم.

پکیج `react-dom` شامل `ReactDOM.render` میشه و داخل `react-dom/server` می‌تونیم با استفاده از متدهای `ReactDOMServer.renderToString` و `ReactDOMServer.renderToStaticMarkup` `server-side rendering` رو پشتیبانی کنیم.

↑ فهرست مطالب

97. چرا ReactDOM رو از React جدا کردن؟

تیم ری‌اکت سعی کرده تمام ویژگی‌های مرتبط با DOM رو جدا کنه و اونا رو توی یه کتابخونه جدا به اسم `ReactDOM` قرار بده. ری‌اکت ورژن ۱۴ اولین نسخه‌ای بود که توش این کتابخونه‌ها از هم جدا شدن. با یه نگاه به بعضی از پکیج‌های ری‌اکت مثل `react-canvas` ، `react-art` ، `react-native` و `react-three` مشخص میشه که زیبایی و جوهر ری‌اکت هیچ ربطی به مرورگرها یا DOM نداره. برای ساختن محیط‌های بیشتری که ری‌اکت بتونه رندر بشه، تیم ری‌اکت اومد و پکیج اصلی ری‌اکت رو به دو بخش تقسیم کنه: `react` و `react-dom`.

از این طریق تونست کامپوننت‌هایی تولید کنه بین ری‌اکت وب و ری‌اکت نیتیو و... قابل اشتراک باشه.

↑ فهرست مطالب

98. چطوری از label تو ری‌اکت استفاده کنیم؟

اگه سعی کنیم که با استفاده از `for` attribute یه عنصر `<label>` متصل به یه متن رو رندر کنیم، اون وقت ویژگی HTML بودن رو از دست میدیم و یه خطا توی کنسول بهمون نشون میده.

```
<label for={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

از اونجایی که `for` یه کلمه کلیدی رزرو شده توی جاوااسکریپت، به جاش باید از `htmlFor` استفاده کنیم.

```
<label htmlFor={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

↑ فهرست مطالب

99. چطوری می‌تونیم چندتا object از استایل‌های درون خطی رو با هم ترکیب کنیم؟

می‌تونیم از *spread operator* توی ری‌اکت استفاده کنیم:

```
<button style={{...styles.panel.button,...styles.panel.submitButton}}
  {"Submit"}
</button>
```

اگه داریم از ری‌کت نیتیو استفاده می‌کنیم می‌تونیم از شکل آرایه‌ای استایل‌ها استفاده کنیم:

```
<button style={[styles.panel.button, styles.panel.submitButton]}>
  {"Submit"}
</button>
```

↑ فهرست مطالب

100. چطوری با **resize** شدن مرورگر یه ویو رو ری‌رندر کنیم؟

می‌تونیم به رخداد **resize** توی **componentDidMount** گوش کنیم و ابعاد (**width** و **height**) رو تغییر بدیم. البته باید حواسمون باشه که این **listener** رو باید توی متد **componentWillUnmount** حذفش کنیم.

```

class WindowDimensions extends React.Component {
  constructor(props) {
    super(props);
    this.updateDimensions = this.updateDimensions.bind(this);
  }

  componentWillMount() {
    this.updateDimensions();
  }

  componentDidMount() {
    window.addEventListener("resize", this.updateDimensions);
  }

  componentWillUnmount() {
    window.removeEventListener("resize", this.updateDimensions);
  }

  updateDimensions() {
    this.setState({
      width: window.innerWidth,
      height: window.innerHeight,
    });
  }

  render() {
    return (
      <span>
        {this.state.width} x {this.state.height}
      </span>
    );
  }
}

```

همین کار رو با استفاده از هوک‌ها هم میشه انجام داد و برای این کار همین کد رو توی `useEffect` می‌نویسیم.

```
const [dimensions, setDimensions] = useState();
useEffect(() => {
  window.addEventListener("resize", updateDimensions);

  function updateDimensions() {
    setDimensions({
      width: window.innerWidth,
      height: window.innerHeight,
    });
  }

  return () => {
    window.removeEventListener("resize", updateDimensions);
  };
}, []);

return (
  <span>
    {this.state.width} x {this.state.height}
  </span>
);
```

↑ فهرست مطالب

101. تفاوت متدهای `setState` و `replaceState` چیه؟

وقتی که از متد `setState` روی کلاس کامپوننت استفاده می‌کنیم، مقادیر فعلی و قبلی با هم ترکیب می‌شن. `replaceState` حالت فعلی رو با `state`ای که می‌خواهیم جایگزینش می‌کنه. معمولا اگه از `setState` برای جایگزین کردن استفاده کنیم، همه کلیدهای قبلی رو پاک کنیم. البته میشه بجای استفاده از `replaceState` با استفاده از `setState` بیاییم و `state` رو برابر با `false` یا `null` قرار بدیم.

↑ فهرست مطالب

102. چطوری به تغییرات `state` گوش بدیم؟

توی کلاس کامپوننت‌ها هنگام به روز شدن `state` یه سری متدها فراخوانی میشه. با استفاده از این متدها میشه `state` و `prop` فعلی رو با مقادیر جدید مقایسه کرده و یه سری کار که مدنظر داریم رو انجام بدیم.

```
componentWillUpdate(object nextProps, object nextState)  
componentDidUpdate(object prevProps, object prevState)
```

با استفاده از هوک `useEffect` هم این امکان بسادگی قابل انجامه و فقط کافیه به `dependency` های این هوک متغیر مربوط به `state` رو بدیم.

```
const [someState, setSomeState] = useState();  
useEffect(() => {  
  // code  
}, [someState]);
```

[↑ فهرست مطالب](#)

103. روش توصیه شده برای حذف یک عنصر از آرایه `state` چیه؟

استفاده از متد `Array.prototype.filter` آرایه ها روش خوبیه. برای مثال می تونیم یه تابع به اسم `removeItem` برای به روز کردن `state` به شکل زیر در نظر بگیریم.

```
removeItem(index) {  
  this.setState({  
    data: this.state.data.filter((item, i) => i !== index)  
  })  
}
```

[↑ فهرست مطالب](#)

104. امکانش هست که ری اکت رو بدون رندر کردن HTML استفاده کنیم؟

توی نسخه های بالاتر از (`16.2=>`) میشه. برای مثال تکه کد پایین یه سری مثال برای رندر کردن یه مقدار غیر `html` ای هست:

```
render() {  
  return false  
}
```



```
render() {  
  return null  
}
```

```
render() {  
  return []  
}
```

```
render() {  
  return <React.Fragment></React.Fragment>  
}
```

```
render() {  
  return <></>  
}
```

البته حواستون باشه که return کردن undefined کار نخواهد کرد.

[↑ فهرست مطالب](#)

105. چطوری همیشه با ری‌اکت به JSON به شکل beautify شده نشون داد؟

میشه گفت زیاد ربطی به ری‌اکت یا غیر ری‌اکت بودن برنامه نداره ولی در کل میشه با استفاده از تگ `<pre>` و استفاده از option های متد `JSON.stringify` این کار رو انجام داد:

```
const data = { name: "John", age: 42 };  
  
class User extends React.Component {  
  render() {  
    return <pre>{JSON.stringify(data, null, 2)}</pre>;  
  }  
}  
  
React.render(<User />, document.getElementById("container"));
```

[↑ فهرست مطالب](#)

106. چرا نمی‌تونیم prop رو آپدیت کنیم؟

فلسفه ساختاری ری‌اکت به شکلیه که propها باید *immutable* باشن و از بالا به پایین و به صورت سلسه‌مراتبی مقدار بگیرند. به این معنی که پدر هر کامپوننت می‌تونه هر مقداری رو به فرزند پاس بده و فرزند حق دستکاری اونو نداره.

[↑ فهرست مطالب](#)

107. چطوری می‌تونیم موقع لود صفحه روی یه input فوکوس کنیم؟

همیشه با ایجاد یه *ref* برای المنت `input` و استفاده از اون توی `componentDidMount` یا `useEffect` این کار رو کرد:

```
const App = () => {
  const nameInputRef = useRef();
  useEffect(() => {
    nameInputRef.current.focus();
  }, []);

  return (
    <div>
      <input defaultValue={"Won't focus"} />
      <input ref={nameInputRef} defaultValue={"Will focus"} />
    </div>
  );
};
```

همین کد در کلاس کامپوننت:

```

class App extends React.Component {
  componentDidMount() {
    this.nameInput.focus();
  }

  render() {
    return (
      <div>
        <input defaultValue={"Won't focus"} />
        <input
          ref={(input) => (this.nameInput = input)}
          defaultValue={"Will focus"}
        />
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById("app"));

```

↑ فهرست مطالب

108. روش‌های ممکن برای آپدیت کردن object توی state کدوما هستن؟

1. فراخوانی متد `setState` با استفاده از یه `object` برای ترکیب شدن اون:

◦ استفاده از `Object.assign` برای ایجاد یه کپی از `object`:

```

const user = Object.assign({}, this.state.user, { age: 42 });
this.setState({ user });

```

◦ استفاده از عملگر `spread`:

```

const user = {...this.state.user, age: 42 };
this.setState({ user });

```

2. فراخوانی `setState` با یه تابع `callback`: به این شکل میشه پیاده‌سازی کرد:

```

this.setState((prevState) => ({
  user: {
    ...prevState.user,
    age: 42,
  },
}));

```

↑ فهرست مطالب

109. چرا توابع به جای object در setState ترجیح داده می‌شوند؟

ری‌اکت اجازه ترکیب کردن تغییرات state رو با استفاده از متد `setState` فراهم کرده، همین موضوع باعث بهبود پرفورمنس میشه. توی کلاس کامپوننت‌ها `this.props` و `this.state` ممکنه به صورت asynchronous و همزمان به روز بشن، نباید به مقدار اونا برای محاسبه مقدار بعدی اعتماد کرد. برای مثال به این شمارنده که درست کار نمی‌کنه دقت کنیم:

```

// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
});

```

روش توصیه شده فراخوانی متد `setState` با یه تابع بجای object هست. این تابع مقدار state قبلی رو به عنوان پارامتر اول و prop رو به عنوان ورودی دوم می‌گیره و این تابع رو زمانی که مقادیر ورودیش تغییر پیدا کنن فراخوانی می‌کنه.

```

// Correct
this.setState((prevState, props) => ({
  counter: prevState.counter + props.increment,
}));

```

↑ فهرست مطالب

110. چطوری می‌تونیم نسخه ری‌اکت جاری رو توی محیط اجرایی بفهمیم؟

خیلی ساده میشه از مقدار `React.version` برای گرفتن نسخه جاری استفاده کرد.

```
const REACT_VERSION = React.version;

ReactDOM.render(
  <div>{`React version: ${REACT_VERSION}`}</div>,
  document.getElementById("app")
);
```

↑ فهرست مطالب

111. روش‌های لود کردن polyfill توی CRA کدوما هستن؟

1. import دستی از core-js :

یه فایل ایجاد کنیم و اسمشو بزاریم (یه چیزی مثل) `polyfills.js` و توی فایل `index.js` بیایید import کنیمش. کد `npm install core-js` یا `yarn add core-js` رو اجرا کنیم و ویژگی‌هایی که لازم داریم رو از `corejs` بارگذاری کنیم.

```
import "core-js/fn/array/find";
import "core-js/fn/array/includes";
import "core-js/fn/number/is-nan";
```

2. استفاده از سرویس Polyfill:

از سایت `CDN polyfill.io` واسه گرفتن مقدار شخصی سازی شده براساس مرورگر هر فرد استفاده کنیم و خیلی ساده یه خط کد به `index.html` اضافه کنیم:

```
<script src="https://cdn.polyfill.io/v2/polyfill.min.js?features="
```

مثلا توی تکه کد فوق ما برای polyfill کردن `Array.prototype.includes` درخواست دادیم.

↑ فهرست مطالب

112. توی CRA چطوری از https به جای http استفاده کنیم؟

برای این کار لازمه که کانفیگ `HTTPS=true` رو برای `env` جاری ست کنیم، برای این کار حتی لازم هم نیست فایل `env` بسازیم و می‌تونیم توی فایل `package.json` بخش

scripts رو به شکل پایین تغییر بدیم:

```
"scripts": {  
  "start": "set HTTPS=true && react-scripts start"  
}
```

یا حتی به شکل `set HTTPS=true && npm start` هم میشه تغییر داد.

[↑ فهرست مطالب](#)

113. توی CRA چطوری میشه از مسیرهای طولانی برای ایمپورت جلوگیری کرد؟

یه فایل به اسم `env` توی مسیر اصلی پروژه ایجاد می‌کنیم و مسیر مورد نظر خودمون رو اونجا می‌نویسم:

```
NODE_PATH=src/app
```

بعد از این تغییر سرور `develop` رو ریستارت می‌کنیم بعدش دیگه می‌تونیم هر چیزی رو از مسیر `src/app` بارگذاری کنیم و لازم هم نباشه مسیر کاملشو بهش بدیم. این کار رو میشه با بخش `module resolve` توی `webpack` هم انجام داد.

[↑ فهرست مطالب](#)

114. چطوری میشه Google Analytics رو به react-router اضافه کرد؟

یه `listener` به آبجکت `history` اضافه می‌کنیم تا بتونیم لود شدن صفحه رو `track` کنیم:

```
history.listen(function (location) {  
  window.ga("set", "page", location.pathname + location.search);  
  window.ga("send", "pageview", location.pathname + location.search);  
});
```

توی نسخه ۶ از `react-router-dom` دسترسی مستقیم به `history` برداشته شده تا پشتیبانی از `suspense` راحت‌تر باشه، توی این نسخه میشه از `useLocation` به شکل زیر استفاده کرد:

```
const location = useLocation();

useEffect(() => {
  window.ga("set", "page", location);
  window.ga("send", "pageview", location);
}, [location]);
```

[↑ فهرست مطالب](#)

115. چطوری یه کامپوننت رو هر ثانیه به روز کنیم؟

لازمه که از `setInterval` استفاده کنیم تا تغییرات رو اعمال کنیم و البته حواسمون هست که موقع `unmount` این `interval` رو حذف کنیم که باعث `memory leak` نشه.

```
const intervalRef = useRef();

useEffect(() => {
  intervalRef.current = setInterval(() => this.setState({ time: I

  return () => {
    clearInterval(intervalRef.current);
  }
}, [location]);
```

توی کلاس کامپوننت هم به شکل:

```
componentDidMount() {
  this.interval = setInterval(() => this.setState({ time: Date.now()
})

componentWillUnmount() {
  clearInterval(this.interval)
}
```

[↑ فهرست مطالب](#)

116. برای استایل‌دهی‌های درون خطی چطوری باید پیشوندهای مخصوص مرورگرها رو اضافه کرد؟

ری اکت به شکل اتوماتیک پیشوندهای مخصوص مرورگرها روی CSS رو اعمال نمی‌کنه. لازمه که تغییرات رو به شکل دستی اضافه کنیم.

```
<div
  style={{
    transform: "rotate(90deg)",
    WebkitTransform: "rotate(90deg)", // note the capital 'W' here
    msTransform: "rotate(90deg)", // 'ms' is the only lowercase v
  }}
/>
```

↑ فهرست مطالب

117. چطوری کامپوننت‌های ری اکت رو با es6 می‌تونیم import و export کنیم؟

لازمه که از default برای export کردن کامپوننت‌ها استفاده کنیم

```
import React from "react";
import User from "user";

export default class MyProfile extends React.Component {
  render() {
    return <User type="customer">{//...</User>;
  }
}
```

با استفاده از کلمه کلیدی export default می‌تونیم کامپوننت MyProfile (یا هر متغیر و کلاس دیگه‌ای) رو به عنوان یه عضو از مازول فعلی معرفی کرد و بعد از این، برای import کردن اون لزومی به استفاده از عنوان این کامپوننت نیست.

↑ فهرست مطالب

118. استثنایی که برای نام‌گذاری کامپوننت اجازه استفاده از حرف کوچک رو میده چیه؟

همه کامپوننت‌های ری اکت لازمه که با حرف بزرگ شروع بشن، ولی توی این مورد هم یه سری استثناء وجود داره. تگ‌هایی که با property و عملگر dot کار می‌کنن رو میشه به

عنوان کامپوننت‌هایی با حرف کوچک تلقی کرد. برای مثال این تگ می‌تونه syntax معتبری برای ری‌اکت باشه که با حروف کوچک شروع میشه:

```
const Component = () => {  
  return (  
    <obj.component /> // `React.createElement(obj.component)`  
  )  
}
```

[↑ فهرست مطالب](#)

119. چرا تابع سازنده کلاس کامپوننت یکبار صدا زده میشه؟

الگوریتم reconciliation ری‌اکت بعد از رندر کردن کامپوننت با بررسی رندهای مجدد، بررسی می‌کنه که این کامپوننت قبلاً رندر شده یا نه و اگه قبلاً رندر شده باشه، تغییرات جدید رو روی همون instance قبلی رندر می‌کنه و instance جدیدی ساخته نمیشه، پس تابع سازنده هم تنها یکبار صدا زده میشه.

[↑ فهرست مطالب](#)

120. توی ری‌اکت چطوری مقدار ثابت تعریف کنیم؟

می‌تونیم از فیلد **استاتیک** ES7 برای تعریف ثابت استفاده کنیم.

```
class MyComponent extends React.Component {  
  static DEFAULT_PAGINATION = 10;  
}
```

فیلدهای استاتیک بخشی از فیلدهای کلاس (class properties) هستن که توی پروپوزال stage 3 معرفی شدن.

[↑ فهرست مطالب](#)

121. چطوری توی برنامه event کلیک شدن رو trigger کنیم؟

می‌تونیم از ref برای بدست آوردن رفرنس **HTMLInputElement** مورد نظر استفاده کنیم و object بدست اومده رو توی یه متغیر یا property نگهداری کنیم، بعدش از اون رفرنس

می‌تونیم برای اعمال رخداد کلیک استفاده کنیم
که `HTMLElement.click` رو فراخوانی می‌کنه. این فرآیند توی دو گام قابل انجام
هستش:

1. ایجاد ref توی متد `render`:

```
<input ref={inputRef} />
```

2. اعمال رخداد `click` توی `event handler`:

```
inputRef.click();
```

↑ فهرست مطالب

122. آیا استفاده از `async/await` توی ری‌اکت ممکنه؟

اگه بخواییم از `async / await` توی ری‌اکت استفاده کنیم، لازمه که *Babel* و پلاگین `transform-async-to-generator` رو استفاده کنیم. توی React Native اینکار با *Babel* و یه سری `transform`‌ها انجام میشه.

↑ فهرست مطالب

123. ساختار پوشه‌بندی معروف برا ری‌اکت چطوره؟

دو روش معروف برای پوشه‌های ری‌اکت وجود داره:

1. گروه‌بندی براساس ویژگی یا `route`:

یک روش معروف قراردادن فایل‌های JS، CSS، و تست‌ها کنارهم به ازای هر ویژگی یا `route` هست، مثل این ساختار:

```

common/
├─ Avatar.js
├─ Avatar.css
├─ APIUtils.js
└─ APIUtils.test.js
feed/
├─ index.js
├─ Feed.js
├─ Feed.css
├─ FeedStory.js
├─ FeedStory.test.js
└─ FeedAPI.js
profile/
├─ index.js
├─ Profile.js
├─ ProfileHeader.js
├─ ProfileHeader.css
└─ ProfileAPI.js

```

2. گروه‌بندی بر اساس ماهیت فایل:

یک سبک مشهور دیگر گروه‌بندی فایل‌ها براساس ماهیت اونهاست که حالا همین روش هم می‌تونه به شکل‌های مختلف اجرا بشه ولی ساختار پایین می‌تونه یه مثال برای این روش باشه:

```

api/
├─ APIUtils.js
├─ APIUtils.test.js
├─ ProfileAPI.js
└─ UserAPI.js
components/
├─ Avatar.js
├─ Avatar.css
├─ Feed.js
├─ Feed.css
├─ FeedStory.js
├─ FeedStory.test.js
├─ Profile.js
├─ ProfileHeader.js
└─ ProfileHeader.css

```

[↑ فهرست مطالب](#)

124. پکیج‌های مشهور برای انیمیشن کدوما هستن؟

React Motion و React Transition Group، React Spring و React Spring پکیج‌های مشهور برای انیمیشن برای ری‌اکت هستند.

[↑ فهرست مطالب](#)

125. مزایای مازول‌های style چیه؟

خیلی توصیه میشه که از استایل‌دهی‌های سخت و مستقیم برای کامپوننت‌ها پرهیز کنیم. هر مقداری که فقط در یک کامپوننت خاصی مورد استفاده قرار می‌گیره، بهتره که درون همون فایل لود بشه. برای مثال، این استایل‌ها می‌تونن تو یه فایل دیگه انتقال پیدا کنن:

```
export const colors = {
  white,
  black,
  blue,
};

export const space = [0, 8, 16, 32, 64];
```

و تو یه موقعی که نیاز داریم از اون فایل مشخص لود کنیمشون:

```
import { space, colors } from "./styles";
```

[↑ فهرست مطالب](#)

126. معروف‌ترین linterهای ری‌اکت کدوما هستن؟

ESLint یه linter برای JavaScript هستش. یه سری کتابخونه برای کمک به کدنویسی تو سبک‌های مشخص و استاندارد برای eslint وجود داره. یکی از معروف‌ترین پلاگین‌های موجود `eslint-plugin-react` هست. به صورت پیش‌فرض این پلاگین یه سری از `best practice`ها رو برای کدهای نوشته شده بررسی می‌کنه و یه مجموعه از قوانین رو برای کدنویسی الزام می‌کنه. پلاگین مشهور دیگه `eslint-plugin-jsx-a11y` هستش، که برای بررسی نکات و ملزومات معروف در زمینه `accessibility` کمک می‌کنه. چرا که JSX یه سینتکس متفاوت‌تری از HTML ارائه می‌کنه، مشکلاتی که ممکنه مثلاً با `alt` و `tabindex` پیش میاد رو با این پلاگین میشه متوجه شد.

127. چطوری باید توی کامپوننت درخواست api call بزنیم؟

می‌تونیم از کتابخانه‌های AJAX مثل Axios یا حتی از `fetch` که به صورت پیش‌فرض تو مرورگر وجود داره استفاده کنیم. لازمه که توی `Mount` درخواست API رو انجام بدیم و برای به روز کردن کامپوننت می‌تونیم از `setState` استفاده کنیم تا داده بدست اومده رو توی کامپوننت نشون بدیم.

برای مثال، لیست کارمندان از API گرفته میشه و توی state نگهداری میشه:

```
const MyComponent = () => {
  const [employees, setEmployees] = useState([]);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch("https://api.example.com/items")
      .then((res) => res.json())
      .then(
        (result) => {
          setEmployees(result.employees);
        },
        (error) => {
          setError(error);
        }
      );
  }, []);

  return error ? (
    <div>Error: {error.message}</div>
  ) : (
    <ul>
      {employees.map((employee) => (
        <li key={employee.name}>
          {employee.name}--{employee.experience}
        </li>
      ))}
    </ul>
  );
};
```

همین کد روی کلاس کامپوننت به شکل زیر اجرا میشد:

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      employees: [],
      error: null,
    };
  }

  componentDidMount() {
    fetch("https://api.example.com/items")
      .then((res) => res.json())
      .then(
        (result) => {
          this.setState({
            employees: result.employees,
          });
        },
        (error) => {
          this.setState({ error });
        }
      );
  }

  render() {
    const { error, employees } = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else {
      return (
        <ul>
          {employees.map((employee) => (
            <li key={employee.name}>
              {employee.name}–{employee.experience}
            </li>
          ))}
        </ul>
      );
    }
  }
}

```

[↑ فهرست مطالب](#)

128. **render props** چیه؟

Render Props به تکنیک ساده برای به اشتراک گذاری کامپوننت بین کامپوننت‌های دیگر است که با استفاده از `prop` که به تابع یا کامپوننت رو بهش دادیم انجام میشه. کامپوننت زیر از همین روش برای پاس دادن به `React element` استفاده می‌کنه و توی کامپوننت پایین این `prop` رو به شکل یه تابع فراخوانی می‌کنیم و چون یه تابع هست، میتونیم بهش هر مقداری که می‌خواهیم بپاریم این سمت رو پاس بدیم.

```
<DataProvider render={({data}) => <h1>{`Hello ${data.target}`}</h1>}
```

کتابخانه‌هایی مثل `React Router` و `DownShift` از این پترن استفاده می‌کنن.

React Router

129. React Router چیه؟

`React Router` یه کتابخونه قدرتمند برای جابجایی سریع بین صفحات و `flow`های مختلفه که برپایه ری‌اکت نوشته شده و امکان `sync` کردن آدرس وارد شده با صفحات رو توی محیط‌های مختلف فراهم می‌کنه.

↑ فهرست مطالب

130. ارتباط React Router و کتابخونه history چیه؟

`React Router` یه `wrapper` روی کتابخونه `history` هست که اعمال اجرایی بر روی `window.history` رو با استفاده از ابجکت‌های `hash` و `browser` مدیریت می‌کنه. البته این کتابخونه یک نوع دیگه از `history` ها به اسم `memory history` رو هم معرفی می‌کنه که برای محیط‌هایی که به صورت عمومی از `history` پشتیبانی نمی‌کنن کاربرد داره. مثل محیط توسعه برنامه موبایل با `(React Native)` یا محیط‌های `unit test` و `Nodejs`.

↑ فهرست مطالب

131. کامپوننت‌های router توی نسخه ۴ کدوما هستن؟

`React Router v4` سه نوع مختلف از کامپوننت روتر (`<Router>`) رو معرفی می‌کنه:

1. `<BrowserRouter>`

2. `<HashRouter>`

3. <MemoryRouter>

کامپوننت‌های فوق به ترتیب `hash`، `browser`، و `memory history` درست می‌کنن. React Router v4 ساخت `history` رو براساس `context` ارائه شده به آبجکت `router` انجام می‌ده و همین موضوعه که باعث میشه بتونیم از این کتابخونه توی محیط‌های مختلف استفاده کنیم.

↑ فهرست مطالب

132. هدف از متدهای `push` و `replace` توی `history` چیه؟

هر شی از آبجکت `history` دو تا متد برای کار با `state` مرورگر ارائه می‌ده.

1. `push`

2. `replace`

اگه به `history` به شکل یک آرایه از مسیرهای بازدید شده نگاه کنیم، `push` یک جابجایی جدید به مسیر اضافه می‌کنه و `replace` مسیر فعلی رو با یه مسیر جدید جایگزین می‌کنه.

↑ فهرست مطالب

133. چطوری توی برنامه به `route` خاص جابجا بشیم؟

روش‌های مختلفی برای جابجایی در برنامه و توسط کد وجود داره که پایین لیست می‌کنیم، ولی روش آخر (استفاده از هوک‌ها) بهترین و ساده‌ترین روش توی کامپوننت‌های تابعی هست.

1. استفاده از تابع مرتبه بالاتر (`withRouter`) (`higher-order`):

متد `withRouter` آبجکت `history` رو به عنوان یه `prop` به کامپوننت اضافه می‌کنه. روی این `prop` به متدهای `push` و `replace` دسترسی داریم که به سادگی می‌تونه مسیریابی بین `route`ها رو فراهم کنه و نیاز به `context` رو رفع کنه.


```
import { withRouter } from "react-router-dom"; // this also works

const Button = withRouter(({ history }) => (
  <button
    type="button"
    onClick={() => {
      history.push("/new-location");
    }}
  >
    {"Click Me!"}
  </button>
));
```

2. استفاده از کامپوننت `<Route>` و پترن `:render props`

کامپوننت `<Route>` همون prop که متد `withRouter` به کامپوننت میده رو به کامپوننت میده.

```
import { Route } from "react-router-dom";

const Button = () => (
  <Route
    render={({ history }) => (
      <button
        type="button"
        onClick={() => {
          history.push("/new-location");
        }}
      >
        {"Click Me!"}
      </button>
    )}
  />
);
```

3. استفاده از `:context`

استفاده از این مورد توصیه نمیشه و ممکنه به زودی deprecate شود.

```
const Button = (props, context) => (
  <button
    type="button"
    onClick={() => {
      context.history.push("/new-location");
    }}
  >
    {"Click Me!"}
  </button>
);

Button.contextTypes = {
  history: React.PropTypes.shape({
    push: React.PropTypes.func.isRequired,
  }),
};
```

4. استفاده از هوک‌های موجود:

هوک‌هایی برای دسترسی به history و params در این کتابخانه وجود دارد مثل useHistory یا حتی توی نسخه ۶ به بعد هوک useNavigate که راحت‌تر می‌تونه امکان navigate بین صفحات رو فراهم کنه:

```
const Page = (props, context) => {
  const history = useHistory();
  const location = useLocation();
  const { slug } = useParams();

  return (
    <button
      type="button"
      onClick={() => {
        history.push("/new-location");
      }}
    >
      {"Click Me!"}
    </button>
  );
};
```

نسخه ۶:

```
const Page = (props, context) => {
  const navigate = useNavigate();

  return (
    <button
      type="button"
      onClick={() => {
        navigate("/new-location");
      }}
    >
    {"Click Me!"}
    </button>
  );
};
```

[↑ فهرست مطالب](#)

134. چطوری همیشه query پارامترها رو توی ری اکت روتر نسخه ۴ گرفت؟

ساده‌ترین راه برای دسترسی به param های آدرس استفاده از هوک useParams هست.

```
const { slug } = useParams();

console.log(`slug query param`, slug);
```

[↑ فهرست مطالب](#)

135. دلیل خطای "Router may have only one child element" چیه؟

باید کامپوننت Route رو توی بلاک `<Switch>` قرار بدیم چون همین کامپوننت `<Switch>` چون Switch هست که باعث میشه منحصرأ فقط یه route با مسیر فعلی تطابق پیدا کنه و کامپوننت اون route توی صفحه رندر بشه. اولش لازمه که `Switch` رو import کنیم:

```
import { Switch, Router, Route } from "react-router";
```

بعدش route ها رو `<Switch>` تعریف می‌کنیم:

```

<Router>
  <Switch>
    <Route { /*... */ } />
    <Route { /*... */ } />
  </Switch>
</Router>

```

[↑ فهرست مطالب](#)

136. چطوری میشه به متد `history.push` پارامتر اضافه کرد؟

همونطوری که می‌دونیم موقع جابجایی میشه یه `object` به `history` پاس بدیم که یه سری گزینه‌ها رو برامون قابل کانفیگ می‌کنه:

```

this.props.history.push({
  pathname: "/template",
  search: "?name=sudheer",
  state: { detail: response.data },
});

```

این کانفیگ‌ها یکیش `search` هست که می‌تونه پارامتر موردنظر ما رو به مسیر مورد نظر بفرسته.

[↑ فهرست مطالب](#)

137. چطوری میشه صفحه ۴۰۴ ساخت؟

کامپوننت `<Switch>` اولین فرزند `<Route>` ای که با درخواست موجود تطابق داشته باشه رو رندر می‌کنه. از اونجایی که یه `<Route>` بدون `path` یا `path *` همیشه مطابق با درخواست‌هاست، پس هنگام خطای ۴۰۴ این مورد برای رندر استفاده میشه.

```

<Switch>
  <Route exact path="/" component={Home} />
  <Route path="/user" component={User} />
  <Route component={NotFound} />
</Switch>

```

[↑ فهرست مطالب](#)

138. توی ری اکت روتر نسخه ۴ چطوری میشه history رو گرفت؟

1. می‌تونیم یه ماژول درست کنیم که `history` object رو میده و هرجایی خواستیم از این فایل استفاده کنیم.
برای مثال فایل `history.js` رو ایجاد کنید:

```
import { createBrowserHistory } from "history";

export default createBrowserHistory({
  /* pass a configuration object here if needed */
});
```

2. می‌تونیم از کامپوننت `<Router>` بجای روترهای پیش‌فرض استفاده کنیم.
فایل `history.js` بالا رو توی فایل `index.js` لود می‌کنیم:

```
import { Router } from "react-router-dom";
import history from "./history";
import App from "./App";

ReactDOM.render(
  <Router history={history}>
    <App />
  </Router>,
  holder
);
```

3. البته همیشه از متد `push` مثل آبجکت پیش‌فرض `history` استفاده کنیم:

```
// some-other-file.js
import history from "./history";

history.push("/go-here");
```

نکته: روی نسخه ۶ دسترسی مستقیم به `history` حذف شده و برای هر کار یه هوک مختص به اون کار مهیا شده.

[↑ فهرست مطالب](#)

139. چطوری بعد از لاگین به شکل خودکار ری‌دایرکت کنیم؟

پکیج `react-router` امکان استفاده از کامپوننت `<Redirect>` رو توی React Router فراهم می‌کنه. رندر کردن `<Redirect>` باعث جابجایی به مسیر پاس داده شده

میشه. دقیقا مثل ریدایرکت سمت سرور، path مسیر جدید با path فعلی جایگزین می‌شه.

```
import React, { Component } from "react";
import { Redirect } from "react-router";

const Component = () => {
  if (isLoggedIn === true) {
    return <Redirect to="/your/redirect/page" />;
  } else {
    return <div>{"Login Please"}</div>;
  }
}
```

چندزبانگی ری‌اکت

140. React-Intl چیه؟

React Intl یه کتابخونه برای آسان نمودن توسعه برنامه‌های چند زبانه‌ست. این کتابخونه از مجموعه‌ای از کامپوننت‌ها و API‌ها برای فرمت‌بندی رشته‌ها، تاریخ و اعداد رو برای ساده‌سازی فرآیند چندزبانگی فراهم می‌کنه. *React Intl* بخشی از *FormatJS* هست که امکان اتصال به ری‌اکت رو با کامپوننت‌های خودش فراهم می‌کنه.

↑ فهرست مطالب

141. اصلی‌ترین ویژگی‌های *React Intl* کدوما هستن؟

1. نمایش اعداد با جداکننده‌های مشخص
2. نمایش تاریخ و ساعت با فرمت درست
3. نمایش تاریخ بر اساس زمان حال
4. امکان استفاده از لیبل‌ها توی `string`
5. پشتیبانی از بیش از ۱۵۰ زبان
6. اجرا توی محیط مرورگر و `node`
7. دارا بودن استانداردهای داخلی

↑ فهرست مطالب

142. دو روش فرمت کردن توی React Intl کدوما هستن؟

این کتابخونه از دو روش برای فرمت‌بندی رشته‌ها، اعداد و تاریخ استفاده می‌کنه: کامپوننت‌های ری‌اکتی و API.

```
<FormattedMessage
  id={"account"}
  defaultMessage={"The amount is less than minimum balance."}
/>
```

```
const messages = defineMessages({
  accountMessage: {
    id: "account",
    defaultMessage: "The amount is less than minimum balance.",
  },
});

formatMessage(messages.accountMessage);
```

[↑ فهرست مطالب](#)

143. چطوری از FormattedMessage به عنوان یه placeholder همیشه استفاده کرد؟

کامپوننت `<...Formatted>` از `react-intl` بجای بازگرداندن `string` یه المنت برگشت می‌ده و به همین دلیل همیشه ازش به عنوان `placeholder` یا `alt` و... استفاده کرد. اگه جایی لازم شد یه پیامی رو اینجور جاها استفاده کنیم باید از `formatMessage` استفاده کنیم. می‌تونیم شی `intl` رو با استفاده از `injectIntl` HOC به کامپوننت موردنظر `inject` کنیم و بعدشم می‌تونیم از متد `formatMessage` روی این شی استفاده کنیم.

```
import React from "react";
import { injectIntl, intlShape } from "react-intl";

const MyComponent = ({ intl }) => {
  const placeholder = intl.formatMessage({ id: "messageId" });
  return <input placeholder={placeholder} />;
};

MyComponent.propTypes = {
  intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

[↑ فهرست مطالب](#)

144. چطوری همیشه locale فعلی رو توی React Intl بدست آورد؟

می‌تونیم با استفاده از `injectIntl` locale فعلی رو بگیریم:

```
import { injectIntl, intlShape } from "react-intl";

const MyComponent = ({ intl }) => (
  <div>`The current locale is ${intl.locale}`</div>
);

MyComponent.propTypes = {
  intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

[↑ فهرست مطالب](#)

145. چطوری با استفاده از React Intl به تاریخ رو فرمت‌بندی کنیم؟

با استفاده از HOC `injectIntl` می‌تونیم به متد `formatDate` توی کامپوننت خودمون دسترسی داشته باشیم. این متد به صورت داخلی توسط `FormattedDate` استفاده میشه و مقدار `string` تاریخ فرمت بندی شده رو برمی‌گردونه.


```
import { injectIntl, intlShape } from "react-intl";

const stringDate = this.props.intl.formatDate(date, {
  year: "numeric",
  month: "numeric",
  day: "numeric",
});

const MyComponent = ({ intl }) => (
  <div>`The formatted date is ${stringDate}`</div>
);

MyComponent.propTypes = {
  intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

تست ری‌اکت

146. توی تست ری‌اکت Shallow Renderer چیه؟

Shallow rendering برای نوشتن یونیت تست توی ری‌اکت کاربرد داره. این روش بهمون این امکان رو میده که به عمق یک مرتبه کامپوننت موردنظرمون رو رندر کنیم و مقدار بازگردانی شده رو بدون اینکه نگران عملکرد کامپوننت‌های فرزند باشیم، ارزیابی کنیم. برای مثال، اگه کامپوننتی به شکل زیر داشته باشیم:

```
function MyComponent() {
  return (
    <div>
      <span className={"heading"}>{"Title"}</span>
      <span className={"description"}>{"Description"}</span>
    </div>
  );
}
```

می‌تونیم انتظار اجرا به شکل زیر رو داشته باشیم:

```
import ShallowRenderer from "react-test-renderer/shallow";

// in your test
const renderer = new ShallowRenderer();
renderer.render(<MyComponent />);

const result = renderer.getRenderOutput();

expect(result.type).toBe("div");
expect(result.props.children).toEqual([
  <span className={"heading"}>{"Title"}</span>,
  <span className={"description"}>{"Description"}</span>,
]);
```

[↑ فهرست مطالب](#)

147. پکیج TestRenderer توی ری اکت چیه؟

این پکیج یه renderer معرفی می‌کنه که می‌تونیم ازش برای رندر کردن کامپوننت‌ها و تبدیل اونا به یه آبجکت pure JavaScript استفاده کنیم بدون اینکه وابستگی به DOM یا محیط اجرایی موبایلی داشته باشیم. این پکیج گرفتن snapshot از سلسله مرتب view (یه چیزی شبیه به درخت DOM) که توسط ReactDOM یا React Native درست میشه رو بدون نیاز به مرورگر یا jsdom فراهم می‌کنه.

```
import TestRenderer from "react-test-renderer";

const Link = ({ page, children }) => <a href={page}>{children}</a>;

const testRenderer = TestRenderer.create(
  <Link page={"https://www.facebook.com/"}>{"Facebook"}</Link>
);

console.log(testRenderer.toJSON());
// {
//   type: 'a',
//   props: { href: 'https://www.facebook.com/' },
//   children: [ 'Facebook' ]
// }
```

[↑ فهرست مطالب](#)

148. هدف از پکیج ReactTestUtils چیه؟

پکیج `ReactTestUtils` توی پکیج `with-addons` ارائه شده و اجازه اجرای یه سری عملیات روی DOMهای شبیه‌سازی شده رو برای انجام یونیت تست‌ها ارائه می‌ده.

↑ فهرست مطالب

149. Jest چیه؟

`Jest` یه فریم‌ورک برای یونیت تست کردن جاوااسکریپت هستش که توسط فیس بوک و براساس `Jasmine` ساخته شده. `Jest` امکان ایجاد اتوماتیک `mock` (دیتا یا مقدار ثابت برای تست) و محیط `jsdom` رو فراهم می‌کنه و اکثراً برای تست کامپوننت‌ها استفاده میشه.

↑ فهرست مطالب

150. مزایای jest نسبت به jasmine کدوما هستن؟

یه سری برتری‌هایی نسبت به `Jasmine` داره:

- می‌تونه به صورت اتوماتیک تست‌ها رو توی سورس کد پیدا و اجرا کنه
- به صورت اتوماتیک می‌تونه وابستگی‌هایی که داریم رو `mock` کنه
- امکان تست کد `asynchronous` رو به شکل `synchronously` فراهم می‌کنه
- تست‌ها رو با استفاده از یه پیاده‌سازی مصنوعی از `DOM` (`jsdom`) اجرا می‌کنه و بواسطه اون تست‌ها قابلیت اجرا توسط `cli` رو دارن
- تست‌ها به شکل موازی اجرا می‌شن و می‌تونن توی مدت زمان زودتری تموم شن

↑ فهرست مطالب

151. یه مثال ساده از تست با jest بزن؟

خب بیاین یه تست برای تابعی که جمع دو عدد رو توی فایل `sum.js` برامون انجام میده بنویسیم:

```
const sum = (a, b) => a + b;

export default sum;
```

یه فایل به اسم `sum.test.js` ایجاد می‌کنیم که تست‌هامون رو توش بنویسیم:

```
import sum from './sum';

test("adds 1 + 2 to equal 3", () => {
  expect(sum(1, 2)).toBe(3);
});
```

و بعدش به فایل `package.json` بخش پایین رو اضافه می‌کنیم:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

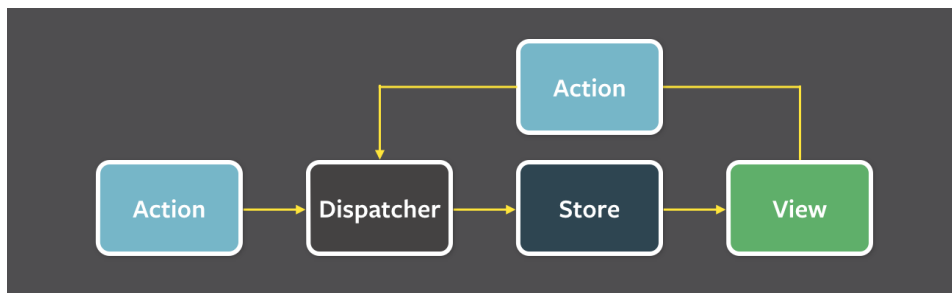
در آخر، دستور `yarn test` یا `npm test` اجرا می‌کنیم و Jest نتیجه تست رو برامون چاپ می‌کنه:

```
$ yarn test
PASS ./sum.test.js
✓ adds 1 + 2 to equal 3 (2ms)
```

React Redux

152. Flux چیست؟

Flux یه الگوی طراحی برنامه است که به عنوان جایگزینی برای اکثر پترن‌های MVC سنتی به کار میره. در حقیقت یه کتابخونه یا فریم‌ورک نیست و یه معماری برای تکمیل کارکرد ری‌اکت با مفهوم جریان داده یک طرفه (Unidirectional Data Flow) به کار میره. فیس‌بوک از این پترن به شکل داخلی برای توسعه ری‌اکت بهره می‌گیره. جریان کار بین `store`، `dispatcher` و `view`های کامپوننت‌ها با ورودی و خروجی مشخص به شکل زیر خواهد بود:



↑ فهرست مطالب

153. Redux چیست؟

Redux یک state manager (مدیریت کننده حالت) قابل پیش‌بینی برای برنامه‌های جاوااسکریپت که برپایه دیزاین پترن *Flux* ایجاد شده. *Redux* می‌تونه با ری‌اکت یا هر کتابخونه دیگه‌ای استفاده بشه. کم حجمه (حدود 2 کیلوبایت) و هیچ وابستگی به کتابخونه دیگه‌ای نداره.

↑ فهرست مطالب

154. مبانی اصلی ریداکس کدوما هستن؟

Redux از سه اصل بنیادی پیروی می‌کنه:

1. یک مرجع کامل و همواره درست: حالت موجود برا کل برنامه در یک درخت object و توی یه store نگهداری میشه. این یکی بودن store باعث میشه دنبال کردن تغییرات در طول زمان و حتی دیباگ کردن برنامه ساده‌تر باشه.
2. State فقط قابل خواندن است: تنها روش ایجاد تغییر در store استفاده از action هستش و نتیجه اجرای این action یک object خواهد بود که رخداد پیش اومده رو توصیف می‌کنه. به این ترتیب مطمئن میشیم که تغییرات فقط با action انجام میشن و هر دیتایی توی store باشه توسط خودمون پر شده.
3. تغییرات با یه سری تابع pure انجام میشن: برای مشخص کردن نحوه انجام تغییرات در store باید reducer بنویسیم. Reducerها فقط یه سری توابع pure هستن که حالت قبلی و action رو به عنوان پارامتر می‌گیرن و حالت بعدی رو برگشت میدن.

↑ فهرست مطالب

155. کاستی‌های redux نسبت به flux کدوما هستند؟

بجای گفتن کاستی‌ها بیابین مواردی که می‌دونیم موقع استفاده از Redux بجای Flux داریم رو بگیم:

1. **باید یاد بگیریم که mutation انجام ندیم:** Flux در مورد mutate کردن داده نظری نمی‌دهد، ولی Redux از mutate کردن داده جلوگیری می‌کنه و پکیج‌های مکمل زیادی برای مطمئن شدن از mutate نشدن state توسط برنامه‌نویس ایجاد شده‌اند. این مورد رو میشه فقط برای محیط توسعه با پکیجی مثل `Immutable.js`، `redux-immutable-state-invariant` یا آموزش تیم برای نوشتن کد بدون mutate دیتا محقق کرد.
2. **باید توی انتخاب پکیج‌ها محتاطانه عمل کنید:** Flux به شکل خاص کاری برای حل مشکلاتی مثل `persist`، `undo/redo` کردن داده یا مدیریت فرم‌ها انجام نداده است. در عوض Redux کلی `middleware` و مکمل `store` برای محقق ساختن همچین نیازهای داره.
3. **شاید هنوز یه جریان داده خوشگل نداشته باشه** در حال حاضر Flux بهمون اجازه به `type check` استاتیک خوب رو میده ولی Redux هنوز پشتیبانی خوبی نداره براش.

↑ فهرست مطالب

156. تفاوت‌های `mapStateToProps` و `mapDispatchToProps` چی هست؟

`mapStateToProps` یه ابزار برای دریافت به روزشدن‌های `state` ها توی کامپوننت هستش (که توسط یه کامپوننت دیگه به روز شده):

```
const mapStateToProps = (state) => {
  return {
    todos: getVisibleTodos(state.todos, state.visibilityFilter),
  };
};
```

`mapDispatchToProps` یه ابزار برای آوردن `action` برای فراخوانی تو کامپوننت ارائه میده (`action`ی که می‌خواهیم `dispatch` کنیم و ممکنه `state` رو عوض کنه):

```
const mapDispatchToProps = (dispatch) => {
  return {
    onClick: (id) => {
      dispatch(toggleTodo(id));
    },
  };
};
```

توصیه همیشه که همیشه از روش "object shorthand" برای دسترسی به

`mapDispatchToProps` استفاده بشه

Redux این action رو توی یه تابع دیگه قرار میده که تقریباً همیشه یه چیزی مثل (...args) `dispatch(onTodoClick(...args))` و تابعی که خودش به عنوان wrapper ساخته رو به کامپوننت مورد نظر ما میده.

```
const mapDispatchToProps = {
  onClick,
};
```

[↑ فهرست مطالب](#)

157. توی ریدیوسر می‌تونیم یه action رو dispatch کنیم؟

Dispatch کردن action توی reducer یه **آنتی پترن** محسوب میشه. reducer نباید هیچ *ساید افکتی* داشته باشه، فقط باید خیلی ساده state قبلی و action فعلی رو بگیره و state جدید رو بده. این کار رو اگه با افزودن یه سری listeners و dispatch کردن با تغییرات reducer هم انجام بدیم باز باعث ایجاد action‌های تودرتو میشه و می‌تونه ساید افکت داشته باشه،

[↑ فهرست مطالب](#)

158. چطوری میشه خارج از کامپوننت میشه store ریداکس دسترسی داشت؟

لازمه که store رو از یه ماژول که با `createStore` ایجاد شده بارگذاری کنیم. البته حواسمون باشه برای انجام این مورد نباید اثری روی window به شکل global ایجاد کنیم.

```
const store = createStore(myReducer);  
  
export default store;
```

↑ فهرست مطالب

159. اشکالات پترن MVW کدوما هستند؟

1. مدیریت DOM خیلی هزینه‌بر هست و می‌تونه باعث کندی و ناکارآمد شدن برنامه بشه.
2. بخاطر circular dependencies (وابستگی چرخشی) یه مدل پیچیده بین model و viewها ایجاد میشه.
3. بخاطر تعامل زیاد برنامه تغییرات خیلی زیادی رخ میده (مثل Google Docs).
4. هیچ روشی ساده و بدون دردسری برای undo کردن (برگشت به عقب) نیست.

↑ فهرست مطالب

160. تشابهی بین Redux و RxJS هست؟

این دو کتابخونه خیلی متفاوتن و برای اهداف متفاوتی استفاده میشن، ولی یه سری تشابه‌های ریزی دارن.

Redux یه ابزار برای مدیریت state توی کل برنامه‌ست. اکثرا هم به عنوان یه معماری برای ایجاد رابط کاربری استفاده میشه. RxJS یه کتابخونه برای برنامه‌نویسی reactive (کنش‌گرا) هستش. اکثرا هم برای انجام تسک‌های asynchronous توی جاوااسکریپت به کار میره.

می‌تونیم بهش به عنوان یه معماری بجای Promise نگاه کنیم. Redux هم از الگوی Reactive استفاده می‌کنه چون Store ریداکس reactive هستش. Store میاد action رو از دور می‌بینه و تغییرات لازم رو توی خودش ایجاد می‌کنه. RxJS هم از الگوی Reactive پیروی می‌کنه، ولی بجای اینکه خودش این architecture رو بسازه میاد به شما یه سری بلاک‌های سازنده به اسم Observable میده که باهاش بتونید الگوی reactive رو اجرا کنید.

↑ فهرست مطالب

161. چطوری میشه یه اکشن رو موقع لود dispatch کرد؟

خیلی ساده میشه اون action رو موقع `mount` اجرا کرد و موقع `render` دیتای مورد نیاز رو داشت.

```
const App = (props) => {
  useEffect(() => {
    props.fetchData();
  }, []);

  return props.isLoading ? (
    <div>{"Loaded"}</div>
  ) : (
    <div>{"Not Loaded"}</div>
  );
};

const mapStateToProps = (state) => ({
  isLoading: state.isLoading,
});

const mapDispatchToProps = { fetchData };

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

↑ فهرست مطالب

162. چطوری از متد `connect` از پکیج `react-redux` استفاده می‌کنیم؟

- برای دسترسی به دیتای نگهداری شده توی ریداکس باید دو گام زیر رو طی کنیم:
1. از متد `mapStateToProps` استفاده می‌کنیم و متغیرهای `state` که از `store` می‌خواهیم لود کنیم رو مشخص می‌کنیم.
 2. `**` با استفاده از متد `connect` دیتا رو به `props` میدیم `**`، چون دیتایی که این HOC میاره به عنوان `props` به کامپوننت داده میشه. متد `connect` رو هم از پکیج `react-redux` باید بارگذاری کنیم.

```
import React from 'react';
import { connect } from 'react-redux';

const App = props => {
  render() {
    return <div>{props.containerData}</div>
  }
};

const mapStateToProps = state => {
  return { containerData: state.data }
};

export default connect(mapStateToProps)(App);
```

[↑ فهرست مطالب](#)

163. چطوری همیشه state ریداکس رو ریست کرد؟

لازمه که توی برنامه یه *root reducer* تعریف کنیم که وظیفه معرفی ریدیوسرهای ایجاد شده با `combineReducers` را دارد.

مثلا بیابین `rootReducer` رو برای ستکردن state اولیه با فراخوانی عمل `USER_LOGOUT` تنظیم کنیم. همونطوری که می‌دونیم، به صورت پیش‌فرض ما بنا رو براین می‌ذاریم که reducerها با اجرای مقدار `undefined` به عنوان پارامتر اول `initialState` رو برمی‌گردونن و حتی actionش هم مهم نیست.

```
const appReducer = combineReducers({
  /* your app's top-level reducers */
});

const rootReducer = (state, action) => {
  if (action.type === "USER_LOGOUT") {
    state = undefined;
  }

  return appReducer(state, action);
};
```

اگه از پکیج `redux-persist` استفاده می‌کنین، احتمالا لازمه که `storage` رو هم خالی کنین. `redux-persist` یه کپی از دیتای موجود در `store` رو توی `localStorage` نگهداری می‌کنه. اولش، لازمه که یه موتور مناسب برای `storage` بارگذاری کنیم که برای تجزیه `state` قبل مقداردهی اون با `undefined` و پاک کردن مقدارشون مورد استفاده قرار می‌گیره.

```
const appReducer = combineReducers({
  /* your app's top-level reducers */
});

const rootReducer = (state, action) => {
  if (action.type === "USER_LOGOUT") {
    Object.keys(state).forEach((key) => {
      storage.removeItem(`persist:${key}`);
    });

    state = undefined;
  }

  return appReducer(state, action);
};
```

[↑ فهرست مطالب](#)

164. هدف از کاراکتر @ توی decorator متد connect چیه؟

کاراکتر (symbol) @ در حقیقت یه نماد از جاوااسکریپت برای مشخص کردن decoratorهاست. *Decorators* این امکان رو بهمون میده که بتونیم برای کلاس و ویژگی‌های (properties) اون یادداشت‌ها و مدیریت‌کننده‌هایی رو توی زمان طراحی اضافه کنیم.

بزاریم یه مثال رو برای Redux بزنیم که یه بار از decorator استفاده کنیم و یه بار بدون اون انجامش بدیم.

:Without decorator ◦

```
import React from "react";
import * as actionCreators from "../actionCreators";
import { bindActionCreators } from "redux";
import { connect } from "react-redux";

function mapStateToProps(state) {
  return { todos: state.todos };
}

function mapDispatchToProps(dispatch) {
  return { actions: bindActionCreators(actionCreators, dispatch) };
}

class MyApp extends React.Component {
  //...define your main app here
}

export default connect(mapStateToProps, mapDispatchToProps)(MyApp);
```

◦ با decorator:

```
import React from "react";
import * as actionCreators from "../actionCreators";
import { bindActionCreators } from "redux";
import { connect } from "react-redux";

function mapStateToProps(state) {
  return { todos: state.todos };
}

function mapDispatchToProps(dispatch) {
  return { actions: bindActionCreators(actionCreators, dispatch) };
}

@connect(mapStateToProps, mapDispatchToProps)
export default class MyApp extends React.Component {
  //...define your main app here
}
```

مثال‌های بالا تقریباً شبیه به هم هستند فقط یکیشون از decoratorها استفاده می‌کنه و اون یکی حالت عادیه. سینتکس decorator هنوز به صورت پیش‌فرض توی هیچ‌کدوم از runtime‌های جاوااسکریپت فعلاً وجود نداره و هنوز به شکل آزمایشی مورد استفاده قرار می‌گیره ولی پروپوزال افزوده شدنش به زبان در دست بررسیه. خوشبختانه فعلاً می‌تونیم از babel برای استفاده از اون استفاده کنیم.

165. تفاوت‌های context و React Redux چیست؟

می‌تونیم از **Context** برای استفاده از state توی مراحل داخلی کامپوننت‌های nested استفاده کنیم و پارامترهای مورد نظرمون رو تا هر عمقی که دلخواه‌مون هست ببریم و استفاده کنیم، که البته context برای همین امر به وجود اومده. این درحالیه که **Redux** خیلی قدرتمندتره و یه سری ویژگی‌هایی رو بهمون میده که نداره هنوز. بعلاوه، خود React Redux به شکل داخلی از context استفاده می‌کنه ولی به شکل عمومی این موضوع رو نشون نمیده.

166. چرا به توابع state ریداکس reducer میگن؟

Reducerها همیشه یه مجموعه از stateها رو جمع‌آوری و تحویل میدن (براساس همه actionهای قبلی). برا همین، اونا به عنوان یه سری کاهنده‌های state عمل می‌کنن. هر وقت یه reducer از Redux فراخوانی میشه، state و action به عنوان پارامتر پاس داده میشن و بعدش این state بر اساس action جاری مقادیرش کاهش یا افزایش داده می‌شوند و بعدش state بعدی برگشت داده میشه. یعنی شما می‌تونین یه مجموعه از داده‌ها رو *reduce* که به state نهایی که دلخواهتون هست برسین.

167. توی redux چطوری میشه api request زد؟

میشه از middleware (میان‌افزار) `redux-thunk` استفاده کرد که اجازه میده بتونیم actionهای async داشته باشیم. بزارین یه مثال از دریافت اطلاعات یه حساب خاص با استفاده از فراخوانی AJAX با استفاده از **fetch API** بزنیم:

```
export function fetchAccount(id) {
  return (dispatch) => {
    dispatch(setLoadingAccountState()); // Show a loading spinner
    fetch(`/account/${id}`, (response) => {
      dispatch(doneFetchingAccount()); // Hide loading spinner
      if (response.status === 200) {
        dispatch(setAccount(response.json)); // Use a normal func
      } else {
        dispatch(someError);
      }
    });
  };
}

function setAccount(data) {
  return { type: "SET_Account", data: data };
}
```

[↑ فهرست مطالب](#)

168. آیا لازمه همه state همه کامپوننت‌ها مونو توی ریداکس نگهداری کنیم؟

نه لزومی نداره، دیتای برنامه رو میشه توی store ریداکس نگهداری کرد و مسائل مربوط به UI به شکل داخلی توی state کامپوننت‌ها نگهداری بشن.

[↑ فهرست مطالب](#)

169. روش صحیح برای دسترسی به store ریداکس چیه؟

بهترین روش برای دسترسی به store و انجام عملیات روی اون استفاده از تابع `connect` هستش که یه کامپوننت جدید ایجاد می‌کنه که کامپوننت جاری توی اون قرار داره و دیتای لازم رو بهش پاس میده. این پترن با عنوان **Higher-Order Components** یا کامپوننت‌های مرتبه بالاتر شناخته میشه و یه روش مورد استفاده برای extend کردن کارکرد کامپوننت‌های ری‌اکتی محسوب میشه. این تابع بهمون این امکان رو میده که state و action‌های مورد نظرمون رو به داخل کامپوننت بیاریم و البته به شکل پیوسته با تغییرات اونا کامپوننت‌مون رو به روز کنیم.

بیابین یه مثال از کامپوننت `<FilterLink>` با استفاده از تابع `connect` برنیم:

```
import { connect } from "react-redux";
import { setVisibilityFilter } from "../actions";
import Link from "../components/Link";

const mapStateToProps = (state, ownProps) => ({
  active: ownProps.filter === state.visibilityFilter,
});

const mapDispatchToProps = (dispatch, ownProps) => ({
  onClick: () => dispatch(setVisibilityFilter(ownProps.filter)),
});

const FilterLink = connect(mapStateToProps, mapDispatchToProps)(L

export default FilterLink;
```

بخاطر مسائل مربوط به پرفورمنس سازنده‌های ریداکس همیشه `connect` رو بجای استفاده از context API برای دسترسی به store ریداکس توصیه می‌کنن.

```
class MyComponent {
  someMethod() {
    doSomethingWith(this.context.store);
  }
}
```

[↑ فهرست مطالب](#)

170. تفاوت‌های component و container توی ریداکس چی هست؟

Component به کامپوننت `class` یا `function` هست که لایه ظاهری و مربوط به UI برنامه شما توی اون قرار می‌گیره.

Container به اصطلاح غیررسمی برای کامپوننت‌هایی هست که به store ریداکس وصل شدن. Container ها به `subscribe` state می‌کنن یا `action` ها رو `dispatch` می‌کنن و هیچ `DOM element` ای رو رندر نمی‌کنن بلکه کامپوننت‌های UI رو به عنوان `child` به روز می‌کنن.

نکته مهم: استفاده از این روش تقریباً توی سال ۲۰۱۹ دیگه منقضی محسوب میشه و چون هوک‌های ری‌اکت خیلی راحت می‌تونن دیتا رو توی هر سطح از کامپوننت برامون لود کنن، پس جدا نشدن این دولایه تاثیر چشم‌گیری توی ساده بودن کدها نخواهد داشت و بعضاً حتی می‌تونه کار رو سخت‌تر کنه، پس به عنوان مترجم توصیه می‌کنم این کار رو انجام ندین:)

171. هدف از constant ها تا type ها توی ریداکس چیه؟

Constant ها یا موارد ثابت بهتون این اجازه رو میدن که کارکرد یه عملکرد مشخص رو به سادگی توی پروژه پیدا کنید. البته از خطاهای ساده‌ای که ممکنه براتون پیش بیاد هم جلوگیری می‌کنه. مثل خطاهای مربوط به type یا `ReferenceError` ها که ممکنه خیلی راحت رخ بدن.

اکثرا مقادیر ثابت `constant` رو توی یه فایل مثل `constants.js` یا `actionTypes.js` قرار می‌دیم.

```
export const ADD_TODO = "ADD_TODO";
export const DELETE_TODO = "DELETE_TODO";
export const EDIT_TODO = "EDIT_TODO";
export const COMPLETE_TODO = "COMPLETE_TODO";
export const COMPLETE_ALL = "COMPLETE_ALL";
export const CLEAR_COMPLETED = "CLEAR_COMPLETED";
```

توی ریداکس از این مقادیر دوتا جا استفاده میشه:

1. موقع ساخت `action`:

مثلا فرض می‌کنیم `actions.js`:

```
import { ADD_TODO } from "./actionTypes";

export function addTodo(text) {
  return { type: ADD_TODO, text };
}
```

2. توی `reducer` ها:

مثلا یه فایل به اسم `reducer.js` رو در نظر بگیرین:


```
import { ADD_TODO } from "./actionTypes";

export default (state = [], action) => {
  switch (action.type) {
    case ADD_TODO:
      return [
        ...state,
        {
          text: action.text,
          completed: false,
        },
      ];
    default:
      return state;
  }
};
```

[↑ فهرست مطالب](#)

172. روش‌های مختلف برای نوشتن `mapDispatchToProps` چیه؟

چندین روش برای `bind` کردن `action` به متد `dispatch` توی `mapDispatchToProps` هستش که پایین بررسی‌شون می‌کنیم:

```
const mapDispatchToProps = (dispatch) => ({
  action: () => dispatch(action()),
});
```

```
const mapDispatchToProps = (dispatch) => ({
  action: bindActionCreators(action, dispatch),
});
```

```
const mapDispatchToProps = { action };
```

روش سوم خلاصه شده روش اوله که معمولا توصیه میشه.

[↑ فهرست مطالب](#)

173. کاربرد پارامتر `ownProps` توی `mapStateToProps` و `mapDispatchToProps` چیه؟

اگه پارامتر `ownProps` ارائه شده باشه، `ReactRedux` پارامترهایی که به کامپوننت پاس داده شدن رو به تابع `connect` پاس میده. پس اگه یه کامپوننت `connect` شده مثل کد زیر داشته باشیم:

```
import ConnectedComponent from "../containers/ConnectedComponent";  
  
<ConnectedComponent user={"john"} />;
```

پارامتر `ownProps` توی `mapStateToProps` و `mapDispatchToProps` یه `object` رو خواهد داشت که مقدار زیر رو داره:

```
{  
  user: "john";  
}
```

می‌تونیم از این مقدار استفاده کنیم تا در مورد مقدار بازگشتی تصمیم بگیریم.

[↑ فهرست مطالب](#)

174. ساختار پوشه‌بندی ریشه ریداکس اکثراً چطوره؟

اکثر برنامه‌های ریداکسی یه ساختاری مثل این دارند:

1. **Components**: که برای کامپوننت‌های *dumb* یا فقط نمایشی که به ریداکس وصل نیستند استفاده می‌شود.

2. **Containers**: که برای کامپوننت‌های *smart* که به ریداکس وصل هستن.

3. **Actions**: که برای همه `action`ها استفاده میشه و هر فایل به بخشی از عملکرد برنامه تعلق داره.

4. **Reducers**: که برای همه `reducer`ها استفاده میشه و هر فایل به یه `state` توی `store` تعلق داره.

5. **Store**: که برای ساختن `store` استفاده میشه.

این ساختار برای یه برنامه کوچک تا بزرگ کاربرد داره. البته اون بخشی ازش که کامپوننت‌های *dumb* و *smart* یا همون `container` و `component` رو بر طبق وصل شدنشون به ریداکس جدا می‌کردیم تقریباً منقصی محسوب میشه.

[↑ فهرست مطالب](#)

175. redux-saga چیست؟

redux-saga به کتابخانه هست که تمرکز اصلیش برای ایجاد side-effectهاست (چیزهای asynchronous مثل fetch کردن داده و غیرشفاف مثل دسترسی به کش مرورگر) که توی برنامه‌های React/Redux با این روش ساده‌تر و بهتر انجام میشه. پکیج ریداکس ساگا روی NPM هست:

```
$ npm install --save redux-saga
```

↑ فهرست مطالب

176. مدل ذهنی redux-saga چگونه است؟

Saga مثل یه thread جداگانه برای برنامه عمل می‌کنه و فقط برای مدیریت ساید افکت کارایی داره. redux-saga به میان‌افزار برای ریداکس هستش، که به معنی اینه که می‌تونه به صورت اتوماتیک توسط action‌های ریداکس شروع بشه، متوقف بشه و یا کار خاصی انجام بده. این میان‌افزار به کل store ریداکس و action‌هایی که کار می‌کنن دسترسی داره و می‌تونه هر action دیگه‌ای رو dispatch کنه.

↑ فهرست مطالب

177. تفاوت افکت‌های call و put توی redux-saga چیست؟

هر دوی افکت‌های call و put سازنده‌های افکت هستن. تابع call برای ایجاد توضیح افکت استفاده میشه که به میان‌افزار دستور میده منتظر call بمونه. تابع put یه افکت ایجاد می‌کنه، که به store می‌گه یه action خاص رو فقط اجرا کنه. بزارین یه مثال در مورد عملکرد این دوتا افکت برای دریافت داده یه کاربر بنیم.

```
function* fetchUserSaga(action) {
  // `call` function accepts rest arguments, which will be passed
  // Instructing middleware to call promise, it resolved value will
  const userData = yield call(api.fetchUser, action.userId);

  // Instructing middleware to dispatch corresponding action.
  yield put({
    type: "FETCH_USER_SUCCESS",
    userData,
  });
}
```

[↑ فهرست مطالب](#)

178. Redux Thunk چیست؟

میان افزار *Redux Thunk* بهمون این اجازه رو میده که actionهایی رو بسازیم که به جای action عادی تابع برگردونن thunk می‌تونه به عنوان یه ایجاد کننده delay برای dispatch کردن یه action استفاده کنیم، یا حتی با بررسی یه شرط خاص یه action رو dispatch کنیم. تابعی که توی action استفاده میشه و `dispatch` و `getState` رو به عنوان پارامتر ورودی می‌گیره.

[↑ فهرست مطالب](#)

179. تفاوت‌های redux-saga و redux-thunk چیست؟

هر دوی *ReduxSaga* و *ReduxThunk* می‌تونن مدیریت سایید افکت‌ها رو به دست بگیرن. توی اکثر سناریوها، Thunk از *Promise* استفاده می‌کنه، درحالی‌که Saga از *Generator* ها استفاده می‌کنه. Thunk تقریباً ساده‌تره و *promise* رو تقریباً همه دولوپرها باهاش آشنا هستن، Sagas/Generators خیلی قوی‌تر هستن و می‌تونن کاربردی‌تر باشن ولی خب لازمه که یاد بگیرینش. هر دوی میان‌افزارها می‌تونن خیلی مفید باشن و شما می‌تونین با Thunks شروع کنین و اگه جایی دیدین نیازمندی‌تون رو برآورده نمی‌کنه سراغ Sagas برید.

[↑ فهرست مطالب](#)

180. Redux DevTools چیست؟

ReduxDevTools به محیط برای مشاهده در لحظه تغییرات ریداکس فراهم می‌کند و قابلیت اجرای مجدد *action* و به رابط کاربری قابل شخصی‌سازی رو فراهم می‌کند. اگر نمی‌خواهین پکیج *ReduxDevTools* رو نصب کنید می‌تونین از افزونه *ReduxDevTools* برای *Chrome* و *Firefox* استفاده کنین.

↑ فهرست مطالب

181. ویژگی‌های *Redux DevTools* کدوما هستن؟

1. بهتون اجازه میده که اطلاعات هر *state* و *payload* پاس داده شده به *action* رو مشاهده کنین.
2. بهتون اجازه میده که *action*‌های اجرا شده رو لغو کنید.
3. اگر به تغییری روی کدهای *reducer* بدین، هر *action*‌ای که *stage* شده رو مجدد ارزیابی می‌کند.
4. اگر به *reducers* به خطایی بده، میشه متوجه شد که در طی انجام شدن دوم *action* این اتفاق افتاده و خطا چی بوده.
5. با `persistState` می‌تونین دیباگ روی موقع *reload*‌های مختلف ذخیره کنید.

↑ فهرست مطالب

182. سلکتورهای ریداکس چی هستن و چرا باید ازشون استفاده کنیم؟

Selector به سری تابع هستن که *state* ریداکس رو به عنوان به پارامتر دریافت می‌کنه و به سری داده که می‌خواهیم رو به کامپوننت پاس میده. برای مثال، دریافت اطلاعات کاربر از ریداکس با *selector* زیر فراهم میشه:

```
const getUserData = (state) => state.user.data;
```

↑ فهرست مطالب

183. *Redux Form* چیه؟

ReduxForm با ری اکت و ریداکس کار می‌کنه تا همه اطلاعات فرم‌ها رو توی *state* ریداکس مدیریت کنیم. *ReduxForm* می‌تونه با *input*های خام *HTML5* هم کار کنه، ولی با فریم‌ورک‌های معروف *UI* مثل *Material*، *ReactWidgets* و *ReactBootstrap* کار کنه.

↑ فهرست مطالب

184. اصلی‌ترین ویژگی‌های *Redux Form* چیه؟

1. ماندگاری مقادیر فیلدهای فرم توی ریداکس.
2. اعتبارسنجی (*sync/async*) و ثبت فرم.
3. فرمت کردن، تجزیه و نرمالسازی مقادیر فیلدها.

↑ فهرست مطالب

185. چطوری میشه چندتا *middleware* به ریداکس اضافه کرد؟

می‌تونیم از `applyMiddleware` استفاده کنیم.
برای مثال میشه از `redux-thunk` و `logger` به عنوان پارامترهای `applyMiddleware` استفاده کنیم:

```
import { createStore, applyMiddleware } from "redux";
const createStoreWithMiddleware = applyMiddleware(
  ReduxThunk,
  logger
)(createStore);
```

↑ فهرست مطالب

186. چطوری میشه توی ریداکس *initial state* تعریف کرد؟

لازم داریم که *state* اولیه رو به عنوان پارامتر دوم به `createStore` پاس بدیم:

```
const rootReducer = combineReducers({
  todos: todos,
  visibilityFilter: visibilityFilter,
});

const initialState = {
  todos: [{ id: 123, name: "example", completed: false }],
};

const store = createStore(rootReducer, initialState);
```

[↑ فهرست مطالب](#)

187. تفاوت‌های Relay با Redux کدوما هستند؟

Relay و Redux توی این مورد که دوتا شونم از یه store استفاده می‌کنن شبیه بهم هستن. تفاوت اصلی این دو اینه که relay فقط state‌هایی رو مدیریت می‌کنه که از سرور تاثیر گرفتن و همه دسترسی‌هایی که به state مربوطه رو با کوئری‌های GraphQL (برای خوندن داده‌ها) و mutationها (برای تغییرات داده) انجام میده. Relay داده‌ها برای شما رو cache می‌کنه و گرفتن داده از سرور رو برای شما بهینه می‌کنه. چون فقط تغییرات رو دریافت میکرد و نه چیز دیگه‌ای.

React Native

188. تفاوت‌های React Native و React کدوما هستند؟

React یه کتابخونه جاوااسکریپتی هست که از اجرای اون روی frontend و اجرای اون روی سرور برای تولید رابط کاربری و برنامه‌های تحت وب پشتیبانی می‌کنه. **React Native** یه فریم‌ورک موبایل هست که کدها رو به کامپوننت‌های native روی موبایل compile می‌کنه و بهمون این اجازه رو میده که برنامه‌های موبایلی (iOS, Android, and Windows) رو با استفاده از جاوااسکریپت بسازیم که از ری‌اکت برای تولید کامپوننت استفاده می‌کنه.

[↑ فهرست مطالب](#)

189. چطوری میشه برنامه React Native رو تست کرد؟

ReactNative می‌تونه توی شبیه‌سازهای سیستم‌عامل‌های موبایلی مثل iOS و Android تست کرد. می‌تونیم برنامه‌های خودمون رو توی برنامه <https://expo.io> (expo) توی گوشی خودمون هم ببینیم که با استفاده از QR-code می‌تونه یه برنامه روی کامپیوتر و گوشی sync کنه، البته باید هر دوی این دستگاه‌ها توی یه شبکه وایرلس باشه.

↑ فهرست مطالب

190. چطوری میشه توی React Native لاگ کرد؟

می‌تونیم از `console.log` ، `console.warn` و غیره استفاده کرد. از نسخه ReactNative 0.29 می‌تونیم خیلی ساده کدهای زیر رو اجرا کنیم که لاگ رو توی خروجی ببینیم:

```
$ react-native log-ios  
$ react-native log-android
```

↑ فهرست مطالب

191. چطوری میشه React Native رو دیباگ کرد؟

- برای دیباگ کردن برنامه ری‌اکت native گام‌های زیر رو طی می‌کنیم:
1. برنامه رو توی شبیه‌ساز iOS اجرا می‌کنیم.
 2. دکمه‌های `Command + D` رو فشار میدیم و یه صفحه وب توی آدرس `http://localhost:8081/debugger-ui` اجرا میشه.
 3. چک‌باکس `On Caught Exceptions` رو برای یه دیباگ بهتر فعال می‌کنیم.
 4. دکمه‌های `Command + Option + I` رو برای اجرای `developer-tools` کروم فشار میدیم یا از طریق منوهای `View` و `Developer` و `DeveloperTools` باز می‌کنیمش.
 5. حالا می‌تونیم برنامه مورد نظر خودمون رو به راحتی تست کنیم.

کتابخونه‌های پشتیبانی شده ری‌اکتی و Integration هاش

192. کتابخونه reselect چیه و چطوری کار می‌کنه؟

memoization به کتابخونه **selector** برای ریداکس هست که از مفهوم *memoization* استفاده می‌کنه. این کتابخونه به صورت اولیه نوشته شده بوده که داده‌های هر برنامه Redux-like یا شبیه ریداکس رو پردازش کنه، ولی نتونسته با هیچ برنامه یا کتابخونه دیگه‌ای گره بخوره.

Reselect به کپی از آخرین inputs/outputs از هر فراخوانی رو نگهداری می‌کنه و فقط زمانی اونو دوباره محاسبه می‌کنه که تغییراتی توی ورودی رخ داده باشه. اگه همون ورودی‌ها دوبار استفاده بشن، Reselect مقدار cache شده رو برمی‌گردونه. *memoization* و *cache*‌ای که استفاده میشه تا حد زیادی قابل شخصی‌سازی.

↑ فهرست مطالب

193. Flow چیه؟

Flow به *static type checker* هستش که طراحی شده تا خطاهای مربوط به نوع‌ها رو توی جاوااسکریپت پیدا کنیم. نوع‌های *flow* می‌تونه خیلی ریزبینانه‌تر از رویکردهای سنتی بررسی نوع عمل کنه. برای مثال، *Flow* بهمون کمک می‌کنه که خطاهای مربوط به دریافت `null` توی برنامه رو کنترل کنیم که توی روش‌های سنتی غیرممکنه تقریباً.

↑ فهرست مطالب

194. تفاوت‌های Flow و PropTypes کدوما هستن؟

Flow به ابزار تجزیه و تحلیل استاتیک (*static-checker*) هستش که از ویژگی‌های بالاتر از زبان استفاده می‌کنه و بهمون کمک می‌کنه که به بخش‌های مختلف برنامه نوع اضافه کنیم و خطاهایی که مرتبط با بررسی نوع‌ها هست رو موقع *compile* ازشون جلوگیری کنیم. *PropTypes* به روش بررسی نوع ساده (موقع *runtime*) هست که روی ری‌اکت اضافه شده. به غیر از نوع‌هایی که به کامپوننت موردنظر به عنوان *prop* داده شده رو نمی‌تونه بررسی کنه. پس اگه دنبال یه روش برای بررسی نوع منعطف هستیم که توی کل پروژه عمل کنه *Flow* یا *TypeScript* روش‌های بهتری هستن.

↑ فهرست مطالب

195. چطوری از آیکون‌های font-awesome توی ری‌اکت استفاده کنیم؟

گام‌های زیر برای استفاده از font-awesome توی ری‌اکت باید طی بشه:

1. پکیج font-awesome رو نصب می‌کنیم:

```
npm install --save font-awesome
```

2. font-awesome رو توی فایل index.js بارگذاری می‌کنیم:

```
import "font-awesome/css/font-awesome.min.css";
```

3. از کلاس این فونت توی className های موردنظر استفاده می‌کنیم:

```
render() {  
  return <div><i className={'fa fa-spinner'} /></div>  
}
```

↑ فهرست مطالب

196. React Dev Tools چیه؟

ReactDeveloperTools بهمون اجازه اینو میده که سلسله مراتب کامپوننت‌های برنامه رو بررسی کنیم و شامل prop و state هم میشه. این مورد به دو روش افزونه (برای Chrome و Firefox) و یه برنامه جانبی مستقل (که با سافاری و مرورگرهای دیگه هم کار می‌کنه) در دسترسه.

پس سه مورد رو می‌تونیم در نظر بگیریم:

1. افزونه Chrome

2. افزونه Firefox

3. برنامه مستقل (Safari، ReactNative و...)

↑ فهرست مطالب

197. چرا توی کروم devtools برای فایل‌های local لود نمیشه؟

اگه یه فایل محلی HTML رو توی مرورگر باز کنیم (...//:file) بعدش لازمه که *ChromeExtensions* یا همون افزونه‌های کروم رو باز کنیم و چک‌باکس Allow access to file URLs رو فعال کنیم.

↑ فهرست مطالب

198. چطوری از Polymer توی React استفاده کنیم؟

1. یه element برای Polymer ایجاد می‌کنیم:

```
<link rel="import" href="../../bower_components/polymer/polymer.html">
Polymer({
  is: "calender-element",
  ready: function () {
    this.textContent = "I am a calender";
  },
});
```

2. کامپوننت Polymer رو با تگ‌های HTML ایجاد می‌کنیم و توی داکيومنت html بارگذاری می‌کنیم، برای مثال اونو توی `index.html` برنامه بارگذاری کنیم:

```
<link
  rel="import"
  href="../../../src/polymer-components/calender-element.html"
/>
```

3. از اون element توی فایل JSX استفاده می‌کنیم:

```
import React from "react";

class MyComponent extends React.Component {
  render() {
    return <calender-element />;
  }
}

export default MyComponent;
```

↑ فهرست مطالب

199. مزایای React نسبت به Vue.js کدوما هستن؟

ری‌اکت مزایای زیر رو نسبت به Vue.js داره:

1. انعطاف پذیری بیشتری رو توی توسعه برنامه‌های بزرگ بهمون میده.
2. تست کردنش راحت‌تره.
3. برای تولید برنامه‌های موبایلی هم مناسبه.
4. اطلاعات و راهکارهای مختلفی براش توی دسترسه.

نکته: لیست موارد فوق صرفاً اظهار نظر شخصی بوده و براساس تجربه حرفه‌ای ممکن است متفاوت باشد. اما به عنوان پارامترهای پایه مفید هستند

****[فهرست] (#فهرست)****

200. تفاوت‌های React و Angular کدوما هستند؟

Angular	React
Angular به فریم ورک و عملکردش کاملاً MVC هستند	ری‌اکت به کتابخانه‌ستو فقط به لایه view دارد
در Angular جریان داده‌ها از دو جهت، یعنی اتصال داده‌های دوطرفه بین والدین و فرزندان رو دارد و به خاطر همین اشکال زدایی سخت تره	در ری‌اکت جریان داده‌ها فقط از یک طریق هستند و به خاطر همین اشکال زدایی راحت تره

نکته: لیست موارد فوق صرفاً اظهار نظر شخصی بوده و براساس تجربه حرفه‌ای ممکن است متفاوت باشد. اما به عنوان پارامترهای پایه مفید هستند

[↑ فهرست مطالب](#)

201. چرا تب React در DevTools نشان داده نمی‌شود؟

موقع بارگیری صفحه، *React DevTools* به گلوبال به اسم `__REACT_DEVTOOLS_GLOBAL_HOOK__` تنظیم می‌کند، بعدش ری‌اکت موقع مقدار دهی اولیه با اون هوک ارتباط برقرار می‌کند. اگه وب سایت از ری‌اکت استفاده نکنه یا ری‌اکت نتونه با DevTools ارتباط برقرار کنه اون تب رو نشون نمیده.

[↑ فهرست مطالب](#)

202. Styled components چیه؟

`styled-components` به کتابخانه جاوااسکریپت برای طراحی ظاهر برنامه‌های ری‌اکت. نقشه برداری بین استایل‌ها و کامپوننت‌ها رو حذف می‌کند و بهمون این امکان رو میده که CSS واقعی رو با جاوااسکریپت بنویسیم.

203. یه مثال از Styled Components می‌تونیم بگی؟

بیاین کامپوننت‌های `<Title>` و `<Wrapper>` رو با استایل‌های خاص برای هر کدوم بسازیم.

```
import React from 'react'
import styled from 'styled-components'

// Create a <Title> component that renders an <h1> which is centered
const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
`

// Create a <Wrapper> component that renders a <section> with some padding
const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`
```

این دو تا متغیر، `Title` و `Wrapper`، کامپوننت‌هایی هستن که می‌تونیم مثل هر کامپوننت دیگه ای رندرشون کنیم.

```
<Wrapper>
  <Title>{'Lets start first styled component!'}</Title>
</Wrapper>
```

204. Relay چیه؟

Relay یه فریم ورک جاوااسکریپت هستش برای ارائه یه لایه داده و ارتباط client-server به برنامه‌های وب با استفاده از لایه view ری‌اکت.

205. چطوری همیشه از تایپ اسکرپت توی create-react-app استفاده کرد؟

با شروع از react-scripts@2.1.0 یا بالاتر، یه پشتیبان داخلی برای typescript وجود داره. میتونیم گزینه `typescript--` رو به صورت زیر منتقل کنیم.

```
npx create-react-app my-app --typescript  
  
# or  
  
yarn create react-app my-app --typescript
```

ولی برای ورژن‌های پایین‌تر وقتی داریم یه پروژه جدید می‌سازیم react scripts، گزینه `scripts---version` رو به عنوان `react-scripts-ts` تنظیم می‌کنیم. `react-scripts-ts` مجموعه‌ای از تنظیمات برای گرفتن پروژه `create-react-app` و آوردن TypeScript داخلش هست. حالا ساختار پروژه باید این شکلی باشه:

```
my-app/  
├─ .gitignore  
├─ images.d.ts  
├─ node_modules/  
├─ public/  
├─ src/  
├─ ...  
├─ package.json  
├─ tsconfig.json  
├─ tsconfig.prod.json  
├─ tsconfig.test.json  
└─ tslint.json
```

متفرقه

206. اصلی‌ترین ویژگی‌های کتابخونه reselect کدوما هستن؟

1. Selector ها داده‌های مشتق شده رو محاسبه میکنه و به ریداکس اجازه میدن حداقل state های ممکن رو ذخیره کنه.
2. Selector ها کارامد هستن. یه selector تا وقتی که یکی از آرگومان هاش تغییر نکرده معتبر نیست.
3. Selector ها قابل ترکیب هستن. اونا می تونن به عنوان ورودی برای بقیه Selector ها استفاده بشن.

207. یه مثال از کاربرد کتابخونه reselect بزن؟

بیاین محاسبات و مقادیر مختلف یه سفارش حمل و نقل رو با استفاده ساده از Reselect انجام بدیم:

```
import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items
const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(
  shopItemsSelector,
  items => items.reduce((acc, item) => acc + item.value, 0)
)

const taxSelector = createSelector(
  subtotalSelector,
  taxPercentSelector,
  (subtotal, taxPercent) => subtotal * (taxPercent / 100)
)

export const totalSelector = createSelector(
  subtotalSelector,
  taxSelector,
  (subtotal, tax) => ({ total: subtotal + tax })
)

let exampleState = {
  shop: {
    taxPercent: 8,
    items: [
      { name: 'apple', value: 1.20 },
      { name: 'orange', value: 0.95 },
    ]
  }
}

console.log(subtotalSelector(exampleState)) // 2.15
console.log(taxSelector(exampleState))      // 0.172
console.log(totalSelector(exampleState))    // { total: 2.322 }
```

[↑ فهرست مطالب](#)

208. توی Redux اکشن چیکار می‌کنه؟

اکشن‌ها آبجکت‌های ساده جاوااسکریپت یا اطلاعاتی هستند که داده‌ها رو از برنامه به store میفرستن. اونا تنها منابع اطلاعاتی برای store هستن. اکشن باید یه ویژگی type داشته باشه که نوع اکشنی که انجام میشه رو نشون بده. برای مثال اکشنی که نشون میده یه آیتم todo جدید اضافه شده:

```
{
  type: 'ADD_TODO',
  text: 'Add todo item'
}
```

[↑ فهرست مطالب](#)

209. استاتیک شی با کلاس‌های ES6 در React کار می‌کنه؟

خیر، استاتیک‌ها فقط با `React.createClass()` کار می‌کنن:

```
someComponent= React.createClass({
  statics: {
    someMethod: function() {
      //..
    }
  }
})
```

اما میتونیم استاتیک‌ها رو داخل کلاس‌های ES6 یا خارج از کلاس مثل زیر بنویسیم،

```
class Component extends React.Component {
  static propTypes = {
    //...
  }

  static someMethod() {
    //...
  }
}
```

```
class Component extends React.Component {
  ....
}

Component.propTypes = {...}
Component.someMethod = function(){...}
```


210. ریداکس رو فقط با ری اکت میشه استفاده کرد؟

ریداکس می‌تونه به عنوان یه ذخیره داده برای لایه UI استفاده بشه. رایج ترین کاربرد ریداکس برای ری اکت و ری اکت نیتیو هستش، ولی اتصالاتی هم برای Angular، Angular 2، Vue، Mithril و موارد دیگه موجوده. ریداکس به راحتی یه مکانیسم اشتراکی ارائه میده که می‌تونه برای کدهای دیگه هم استفاده بشه.

211. برای استفاده از Redux به ابزار build خاصی احتیاج داریم؟

ریداکس در اصل توی ES6 نوشته شده و برای تولید توی ES5 با Webpack و Babel منتشر شده. ما باید بتونیم بدون توجه به مراحل ساخت جاواسکریپت از اون استفاده کنیم. ریداکس همینطور یه ساختار UMD ارائه میده که می‌تونه مستقیم و بدون هیچگونه مراحل ساخت مورد استفاده قرار بگیره.

212. مقادیر پیش فرض ریداکس فرم چطوری تغییرات رو از state می‌گیرن؟

باید تنظیمات `enableReinitialize: true` رو اضافه کنیم.

```
const InitializeFromStateForm = reduxForm({
  form: 'initializeFromState',
  enableReinitialize: true
})(UserEdit)
```

اگه `initialValues` prop به روز بشه، فرممون هم به روز میشه.

213. توی PropTypes های ری اکت چطوری میشه برای یه prop چند نوع داده مجاز مشخص کرد؟

می‌تونیم از یکی از متدهای PropTypes به اسم `oneOfType()` استفاده کنیم. برای مثال، ویژگی `height` رو می‌تونیم با دو نوع `string` یا `number` مثل زیر تعریف کنیم:

```
Component.propTypes = {
  size: PropTypes.oneOfType([
    PropTypes.string,
    PropTypes.number
  ])
}
```

↑ فهرست مطالب

214. می‌تونیم فایل `svg` رو به عنوان کامپوننت `import` کنیم؟

می‌تونیم SVG رو مستقیماً به عنوان یه کامپوننت به جای لود کردنش به عنوان یه فایل ایمپورت کنیم. این ویژگی توی `react-scripts@2.0.0` و ورژن‌های بالاتر در دسترسه.

```
import { ReactComponent as Logo } from './logo.svg'

const App = () => (
  <div>
    { /* Logo is an actual react component */ }
    <Logo />
  </div>
)
```

نکته فراموش نکنیم که موقع ایمپورت کردن از آکولاد استفاده کنیم.

↑ فهرست مطالب

215. چرا استفاده از توابع `ref callback` درون خطی توصیه نمیشه؟

اگه `ref callback` به عنوان یه تابع درون خطی تعریف بشه، در طول به روزرسانی دو بار فراخوانی میشه، یه بار با مقدار `null` و بعد دوباره با عنصر `DOM`. این موضوع به خاطر اینکه

که به نمونه جدیدی از تابع با هر بار رندر ساخته می‌شود، پس ری‌اکت باید ref قبلی رو پاک کنه و به نمونه جدید ایجاد کنه.

```
class UserForm extends Component {
  handleSubmit = () => {
    console.log("Input Value is: ", this.input.value)
  }

  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} /> // Access DOM input
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

اما انتظار ما اینه که وقتی کامپوننت mount شد، ref callback به بار صدا زده بشه. به راه حل سریع استفاده از class property syntax ES6 برای تعریف تابع هستش.

```
class UserForm extends Component {
  handleSubmit = () => {
    console.log("Input Value is: ", this.input.value)
  }

  setSearchInput = (input) => {
    this.input = input
  }

  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={this.setSearchInput} /> // Access DOM input in handler
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

216. render hijacking توی ری اکت چیه؟

مفهوم render hijacking توانایی کنترل اینه که چه کامپوننتی خروجی بقیه کامپوننت هاست. در واقع به این معنیه که ما می‌تونیم با قرار دادن کامپوننت خودمون توی یه کامپوننت با اولویت بالا یه تغییراتی بهش بدیم، مثلاً یه سری prop بهش اضافه کنیم یا تغییرات دیگه ای که باعث تغییر منطق رندر بشه. این در واقع hijacking رو فعال نمیکنه اما با استفاده از HOC این امکان رو فراهم می‌کنیم که کامپوننت رفتار متفاوتی داشته باشه.

↑ فهرست مطالب

217. پیاده‌سازی factory یا سازنده HOC چطوره؟

دو روش اصلی برای اجرای HOC ها توی ری اکت وجود داره. 1. Props Proxy (PP) و 2. Inheritance Inversion (II). اونا روش‌های مختلفی رو برای اداره کردن *WrappedComponent* دنبال می‌کنن.

Props Proxy

تو این روش، متد رندر HOC یه عنصر ری اکت از نوع *WrappedComponent* رو برمی‌گردونه. ما هم prop هایی که HOC دریافت میکنه رو انتقال میدیم، به خاطر همین بهش **Props Proxy** گفته میشه.

```
function ppHOC(WrappedComponent) {
  return class PP extends React.Component {
    render() {
      return <WrappedComponent {...this.props}/>
    }
  }
}
```

****Inheritance Inversion****

In this approach, the returned HOC class (Enhancer) extends the *WrappedComponent*. It is called Inheritance Inversion because instead of the *WrappedComponent* extending some Enhancer class, it is passively extended by the Enhancer. In this way the relationship between them seems **inverse**.

```
function iiHOC(WrappedComponent) {
  return class Enhancer extends WrappedComponent {
    render() {
      return super.render()
    }
  }
}
```

↑ فهرست مطالب

218. چطوری به یه کامپوننت ری‌اکت عدد پاس بدیم؟

اعداد رو باید از طریق آکولاد همونطور که رشته رو داخل کوتیشن قرار میدیم، انتقال بدیم.

```
React.render(<User age={30} department={"IT"} />, document.get
```

↑ فهرست مطالب

219. لازمه همه state‌ها رو توی ریداکس مدیریت کنیم؟ لزومی به استفاده از state داخلی داریم؟

این به تصمیم توسعه دهنده بستگی داره. به عنوان مثال این وظیفه توسعه دهنده‌ستکه بررسی کنه چه نوعی از state‌ها برنامه رو تشکیل بده و هر state کجا باید قرار بگیره. بعضی از کاربرا ترجیح میدن هر قسمت از داده رو توی ریداکس نگه دارن، تا همیشه یه ورژن پشت سر هم و کنترل شده از برنامه شون رو داشته باشن. یه عده دیگه ترجیح میدن UI State یا non-critical رو نگه دارن، مثل "is this dropdown currently open"، توی state داخلی یه کامپوننت.

اینا قوانینی هستن که تعیین می‌کنن چه نوع داده ای باید توی ریداکس قرار بگیره

1. آیا بقیه قسمتای برنامه به این داده‌ها اهمیت میدن؟
2. آیا نیازه که بتونیم یه سری داده‌ها رو از روی این داده‌های اصلی به دست بیاریم؟
3. آیا از این داده‌ها توی چندین کامپوننت استفاده میشه؟
4. آیا نیازه که بتونیم یه state رو به یه بازه زمانی خاصی برگردونیم؟
5. آیا می‌خواهیم داده رو توی حافظه نگه داریم؟ (یعنی به جای درخواست مجدد، از اطلاعات موجود توی state استفاده کنیم)

220. هدف از متد registerServiceWorker توی ری اکت چیه؟

ری اکت به صورت پیش فرض و بدون هیچگونه پیکربندی، یه سرویس دهنده برامون ایجاد میکنه. سرویس دهنده یه API وب هستش که در ذخیره کردن asset ها و فایل های دیگه بهمون کمک میکنه تا وقتی کاربر آفلاینه یا سرعت اینترنتش پایینه، بازم بتونه نتایج رو روی صفحه ببینه. به این ترتیب بهمون کمک میکنه تجربه کاربری بهتری ایجاد کنیم و همون چیزیه که باید در مورد service worker ها بدونیم. این ها همه مواردی بود در رابطه با اضافه کردن قابلیت های آفلاین به سایتمون.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
registerServiceWorker();
```

221. چطوری با استفاده از تابع setState از رندر غیرضروری جلوگیری کنیم؟

میتونیم مقدار فعلی یه state رو با مقدار موجود مقایسه کنیم و تصمیم بگیریم که صفحه مجددا رندر بشه یا نه. اگه مقادیر یکسان بود برای جلوگیری از رندر مجدد باید مقدار null رو برگردونیم و در غیر این صورت آخرین مقدار state رو برمی گردونیم. برای مثال، اطلاعات پروفایل کاربر توی مثال زیر به صورت شرطی رندر شده:

```
getUserProfile = (user) => {
  const latestAddress = user.address;
  this.setState((state) => {
    if (state.address === latestAddress) {
      return null;
    } else {
      return { title: latestAddress };
    }
  });
};
```

222. توی نسخه ۱۶ ری اکت چطوری میشه آرایه، Strings و یا عدد رو رندر کنیم؟

آرایه ها: بر خلاف نسخه های قدیمی، نیازی نیست مطمئن بشیم که متد **render** یه عنصری رو توی ری اکت 16 برمیگردونه. میتونیم عناصر شبیه هم رو بدون نیاز به عنصر بسته بندی به عنوان یه آرایه برگردونیم. به عنوان مثال، بیاین لیست توسعه دهندگان زیر رو بگیریم،

```
const ReactJSDevs = () => {
  return [
    <li key="1">John</li>,
    <li key="2">Jackie</li>,
    <li key="3">Jordan</li>,
  ];
};
```

همینطور میتونیم آیتم های این آرایه رو توی یه کامپوننت دیگه ای ادغام کنیم

```
const JSDevs = () => {
  return (
    <ul>
      <li>Brad</li>
      <li>Brodge</li>
      <ReactJSDevs />
      <li>Brandon</li>
    </ul>
  );
};
```

****رشته‌ها و اعداد:**** همينطور مي‌تونيم انواع رشته‌ها و اعداد رو با متد رندر برگ

```
<"span align="left" dir="ltr">

jsx``
} ()render
;'return 'Welcome to ReactJS questions
{
Number //
} ()render
;return 2018
{
``

<span/>

**[فهرست] (#فهرست)**
```

225. چطوري ميشه از تعريف ويژگي در کلاس کامپوننت استفاده کرد؟

React Class Components can be made much more concise using the class field declarations. You can initialize local state without using the constructor and declare class methods by using arrow functions without the extra need to bind them. Let's take a counter example to demonstrate class field declarations for state without using constructor and methods without ,binding


```

class Counter extends Component {
  state = { value: 0 };

  handleIncrement = () => {
    this.setState((prevState) => ({
      value: prevState.value + 1,
    }));
  };

  handleDecrement = () => {
    this.setState((prevState) => ({
      value: prevState.value - 1,
    }));
  };

  render() {
    return (
      <div>
        {this.state.value}

        <button onClick={this.handleIncrement}>+</button>
        <button onClick={this.handleDecrement}>-</button>
      </div>
    );
  }
}

```

[↑ فهرست مطالب](#)

226. hook ها چی هستن؟

هوک‌ها ویژگی جدیدی هستند که به‌همون این امکان رو میدن که بدون نوشتن کلاس از `useState` ببینیم:

```
<"span align="left" dir="ltr">

jsx``
;import { useState } from "react

} ()function Example
"Declare a new state variable, which we'll call "count //
;(const [count, setCount] = useState(0

) return
<div>
<p>You clicked {count} times</p>
<button onClick={() => setCount(count + 1)}>Click me</button>
</div>
;(
{
...

</span>

**[فهرست] (#فهرست)**
```

227. چه قوانینی برای هوک‌ها باید رعایت بشن؟

برای استفاده از هوک‌ها باید از دو قانون پیروی کنیم

1. هوک‌ها رو فقط در سطح بالای توابع ری اکت صدا کنیم. یعنی نباید هوک‌ها رو توی حلقه‌ها، شرط‌ها یا توابع تودرتو صدا کنیم. با این کار اطمینان حاصل میشه که هوک‌ها با هر بار رندر کامپوننت به همون ترتیب صدا زده میشن و `state` هوک‌ها بین چندین بار استفاده از `useEffect` , `useState` حفظ میشه.
2. هوک‌ها رو فقط توی ری اکت میتونیم استفاده کنیم. توی توابع جاوااسکریپتی نباید هوک‌ها رو صدا بزنیم.

[↑ فهرست مطالب](#)

228. چطوری میشه از استفاده درست هوک‌ها اطمینان حاصل کرد؟

تیم ری اکت به پلاگین ESLint به اسم **eslint-plugin-react-hooks** منتشر کرده که این دو قانون رو اجرا میکنه. با استفاده از دستور زیر میتونیم این پلاگین رو به پروژه مون اضافه کنیم.

```
npm install eslint-plugin-react-hooks@next
```

و تنظیمات زیر رو توی فایل ESLint config اعمال کنیم

```
<span align="left" dir="ltr">
```

```
javascript``
Your ESLint configuration //
}
] : "plugins"
...//
"react-hooks"
, [
} : "rules"
...//
"react-hooks/rules-of-hooks": "error"
{
{
````
```

```

```

**\*\*نکته\*\*** این پلاگین به صورت پیش فرض در نظر گرفته شده تا در ساخت React App

**\*\*[فهرست] (#فهرست)\*\***

## 229. تفاوت‌های Flux و Redux کدوما هستن؟

اینجا تفاوت عمده Flux و Redux گفته شده

Flux	Redux
State قابل تغییره	State غیر قابل تغییره
Store شامل منطق تغییر و State هستش	Store و منطق تغییر از هم جدا هستن
Storeهای مختلفی وجود داره	فقط یه Store وجود داره
تمام Storeها جدا از هم هستن	یه Store با Reducerهای سلسله مراتبی
یه dispatcher تکی داره	مفهومی به اسم dispatcher وجود نداره

Flux	Redux
React components subscribe to the store	کامپوننت‌های Container از تابع connect استفاده می‌کنن.

[↑ فهرست مطالب](#)

## 230. مزایای ری‌اکت روتر نسخه ۴ چیه؟

اینجا مزایای اصلی ماژول React Router V4 گفته شده:

1. توی React Router ورژن ۴، API کلا در مورد کامپوننت هاست. یه Router می‌تونیم به عنوان یه کامپوننت تکی () تجسم کنیم که کامپوننت‌های روتر فرزند () رو دسته بندی میکنه.
2. نیازی به تنظیم دستی history نداریم. روتر از طریق بسته بندی route‌ها با کامپوننت از history مراقبت میکنه.
3. اندازه برنامه فقط با یه ماژول روتر خاص (Web, core یا native) کاهش پیدا میکنه.

[↑ فهرست مطالب](#)

## 231. می‌توننی راجع به متد componentDidCatch توضیح بدی؟

بعد از اینکه خطایی توسط یه کامپوننت با سلسله مراتب پایین تر ارسال شد، متد **componentDidCatch** صدا زده میشه. این متد دو تا پارامتر دریافت میکنه:

1. error: - آبجکت error
  2. info: - یه آبجکت با کلید componentStack که شامل اطلاعاتیه در مورد اینکه کدوم کامپوننت خطا ایجاد کرده.
- ساختار متد به صورت زیر هستش:

```
;(componentDidCatch(error, info
```

[↑ فهرست مطالب](#)

## 232. در چه سناریویی error boundary خطا رو catch نمی‌کنه؟

این‌ها مواردی هستند که error boundary ها اونجا کار نمی‌کنن

1. داخل Event handler ها

2. کد ناهمزمان با استفاده از callback های **setTimeout** یا

**requestAnimationFrame**

3. During Server side rendering

4. موقع ارائه سمت سرور (Server side rendering)

5. وقتی خطاها در خود کد error boundary ها رخ می‌ده.

↑ فهرست مطالب

## 233. چرا نیازی به error boundaries برای event handler نیست؟

Error boundary ها خطاها رو توی event handler نمی‌گیرن. Event handler ها بر خلاف متد رندر یا lifecycle موقع رندر کردن اتفاق نمی‌افته یا فراخوانی نمیشه. بنابراین ری اکت میدونه که این مدل خطاها رو توی event handler چطوری بازایی کنه. اگه هنوز نیاز داریم خطا رو توی event handler بگیریم، می‌تونیم از دستور try / catch جاوااسکریپت مثل زیر استفاده کنیم:

```
class MyComponent extends React.Component {
 constructor(props) {
 super(props);
 this.state = { error: null };
 }

 handleClick = () => {
 try {
 // Do something that could throw
 } catch (error) {
 this.setState({ error });
 }
 };

 render() {
 if (this.state.error) {
 return <h1>Caught an error.</h1>;
 }
 return <div onClick={this.handleClick}>Click Me</div>;
 }
}
```

کد بالا خطا رو با استفاده از try/catch جاوااسکریپت به جای error boundary ها میگیره.

↑ فهرست مطالب

## 234. تفاوت بلوک try catch و error boundary ها چیه؟

بلوک try catch با کد دستوری کار میکنه در حالی که error boundary ها برای ارائه کد اعلانی روی صفحه در نظر گرفته شدن.  
برای مثال، بلوک try catch برای کد دستوری زیر استفاده میشه

```
try {
 showButton();
} catch (error) {
 //...
}
```

در حالی که error boundary ها کدهای اعلانی رو به صورت زیر بسته بندی می کنه،

```
<ErrorBoundary>
 <MyComponent />
</ErrorBoundary>
```

پس اگه خطایی توی متد **componentDidUpdate** توسط **setState** جایی در عمق درخت، رخ بده بازم به درستی به نزدیک ترین error boundary گسترش پیدا میکنه.

↑ فهرست مطالب

## 235. رفتار خطاهای uncaught در ری اکت 16 چیه؟

توی ری اکت ورژن ۱۶، خطاهایی که توسط هیچ error boundary گرفته نشن، منجر به unmount شدن کل درخت کامپوننت ری اکت میشن. دلیل این تصمیم اینه که رابط کاربری خراب بهتره که کامل حذف بشه تا اینکه سر جای خودش باقی بمونه. به عنوان مثال، برای یه برنامه پرداخت بهتره گه هیچی رندر نکنیم تا اینکه بخوایم یه مقدار اشتباه رو نشون بدیم.

↑ فهرست مطالب

## 236. محل مناسب برای قرار دادن error boundary کجاست؟

میزان استفاده از error boundary بر اساس نیاز پروژه به عهده توسعه دهنده ست. می‌تونیم از هر کدام از روش‌های زیر استفاده کنیم

1. می‌تونیم روت کامپوننت‌های سطح بالا رو برای نمایش یه پیغام خطای عمومی واسه کل برنامه بسته بندی کنیم.
2. همین طور می‌تونیم کامپوننت‌های تکی رو توی یه error boundary قرار بدیم تا از خراب شدن کل برنامه محافظت بشه.

↑ فهرست مطالب

## 237. مزیت چاپ شدن stack trace کامپوننت‌ها توی متن ارور boundary ری‌اکت چیه؟

به غیر از پیام‌های خطا و پشته جاوااسکریپت، ری‌اکت ورژن ۱۶ پشته کامپوننت رو با نام فایل و شماره خط با استفاده از مفهوم error boundary نمایش میده. برای مثال، کامپوننت BuggyCounter پشته کامپوننت رو به صورت زیر نشون میده:

```
► React caught an error thrown by BuggyCounter. You should fix this error in your code. react-dom.development.js:7708
React will try to recreate this component tree from scratch using the error boundary you provided, ErrorBoundary.

Error: I crashed!

The error is located at:
 in BuggyCounter (at App.js:26)
 in ErrorBoundary (at App.js:21)
 in div (at App.js:8)
 in App (at index.js:5)
```

↑ فهرست مطالب

## 238. متدی که در تعریف کامپوننت‌های class الزامیه؟

متد render() تنها متد مورد نیاز توی class کامپوننت هستش. به عنوان مثال، همه متدها غیر از متد render توی class کامپوننت اختیاری هستش.

↑ فهرست مطالب

## 239. نوع‌های ممکن برای مقدار بازگشتی متد render کدوما هستن؟

اینجا لیستی از انواع type‌های استفاده شده و برگشت داده شده توسط متد رندر نوشته شده:

1. **عناصر ری اکت** عناصری که به ری اکت دستور میدن تا یه گره DOM رو رندر کنه. این عناصر شامل عناصر html مثل `<div>` و عناصر تعریف شده توسط کاربر هستش.
2. **Arrays and fragments:** Return multiple elements to render as Arrays and Fragments to wrap multiple elements
3. **Portals** فرزندها رو داخل یه زیرشاخه DOM متفاوت رندر میکنه
4. **رشته‌ها و اعداد** رشته‌ها و اعداد رو به عنوان گره متنی توی DOM رندر میکنه.
5. **Boolean یا null** چیزی رندر نمیکنه اما از این type برای رندر کردن محتوای شرطی استفاده میشه.

[↑ فهرست مطالب](#)

## 240. هدف اصلی از متد constructor چیه؟

constructor به طور عمده برای دو منظور استفاده میشه:

1. برای مقدار دهی اولیه local state با تخصیص اِجِکت به `this.state`
2. برای اتصال متدهای event handler به نمونه  
به عنوان مثال کد زیر هر دو مورد بالا رو پوشش میده:

```
} (constructor(props
;super(props
!Don't call this.setState() here //
;{ this.state = { counter: 0
;(this.handleClick = this.handleClick.bind(this
{
```

[↑ فهرست مطالب](#)

## 241. آیا تعریف متد سازنده توی ری اکت الزامیه؟

نه، اجباری نیست. به عنوان مثال، اگه ما state رو مقدار دهی اولیه نکنیم و متدها رو متصل نکنیم، نیازی به پیاده سازی constructor برای کاموننتمون نداریم.

[↑ فهرست مطالب](#)



## 242. Default prop ها چی هستن؟

defaultProp ها به عنوان یه ویژگی روی کلاس کامپوننت تعریف شده تا prop های پیش فرض رو برای کلاس تنظیم کنه. این مورد برای prop های undefined استفاده میشه نه برای prop های null. به عنوان مثال بیاین یه prop پیش فرض رنگ برای کامپوننت button بسازیم.

```
class MyButton extends React.Component {
 //...
}

MyButton.defaultProps = {
 color: "red",
};
```

اگه props.color نشه مقدار پیش فرض روی red تنظیم میشه. به عنوان مثال هر جا بخوایم به prop color دسترسی پیدا کنیم از مقدار پیش فرض استفاده میکنه.

```
render() {
 return <MyButton /> ; // props.color will be set to red
}
```

**\*\*نکته:\*\*** اگه مقدار null رو ارائه بدیم مقدار null باقی می مونه.

[↑ فهرست مطالب](#)

## 243. چرا نباید تابع setState رو توی متد componentWillUnmount فراخوانی کرد؟

setState () رو نباید توی componentWillUnmount () فراخوانی کنیم چون وقتی یه کامپوننت unmount میشه، دیگه هیچوقت دوباره mount نمیشه.

[↑ فهرست مطالب](#)

## 244. کاربرد متد getDerivedStateFromError چیه؟

This lifecycle method is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter

and should return a value to update state. The signature of the lifecycle method is as follows

```
static getDerivedStateFromError(error)
```

Let us take error boundary use case with the above lifecycle method for demonstration purpose

```
class ErrorBoundary extends React.Component {
 constructor(props) {
 super(props);
 this.state = { hasError: false };
 }

 static getDerivedStateFromError(error) {
 // Update state so the next render will show the fallback UI.
 return { hasError: true };
 }

 render() {
 if (this.state.hasError) {
 // You can render any custom fallback UI
 return <h1>Something went wrong.</h1>;
 }

 return this.props.children;
 }
}
```

[↑ فهرست مطالب](#)

## 245. کدام متدها و به چه ترتیبی در طول ری رندر فراخوانی میشن؟

تغییر در prop یا state می تونه باعث به روزسانی بشه. متدهای زیر به ترتیب زیر وقتی به کامپوننت مجددا رندر میشه صدا زده میشه.

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

## 246. کدوم متدها موقع error handling فراخوانی میشن؟

وقتی یه خطایی موقع رندر کردن وجود داشته باشه، توی متد lifecycle، یا توی constructor هر کامپوننت فرزند، متدهای زیر فراخوانی میشه.

1. static getDerivedStateFromError()

2. componentDidCatch()

## 247. کارکرد ویژگی displayName چیه؟

به عنوان مثال، برای سهولت توی اشکال زدایی یه displayName انتخاب می‌کنیم که نشون میده این نتیجه یه HOC withSubscription هستش.

```
function withSubscription(WrappedComponent) {
 class WithSubscription extends React.Component {
 /*... */
 }
 WithSubscription.displayName = `WithSubscription(${getDisplayName(WrappedComponent)})`;
 return WithSubscription;
}
function getDisplayName(WrappedComponent) {
 return (
 WrappedComponent.displayName || WrappedComponent.name || "Component"
);
}
```

## 248. ساینپورت مرورگرها برای برنامه ری‌اکتی چطوره؟

ری‌اکت همه مرورگرهای معروف از جمله اینترنت اکسپلورر ۹ به بالا رو پشتیبانی می‌کنه، اگرچه برای مرورگرهای قدیمی‌تر مثل IE 9 و IE 10 یه سری polyfills نیازه. اگه از polyfill

**es5-shim و es5-sham** استفاده کنیم در اون صورت حتی مرورگرهای قدیمی رو هم پشتیبانی میکنه که متدهای ES5 رو پشتیبانی نمیکنه

↑ فهرست مطالب

## 249. هدف از متد `ReactDOM.unmountComponentAtNode` چیه؟

این متد از بسته `react-dom` در دسترس هستش و کامپوننت `mount` شده رو از `DOM` حذف میکنه و `event handler` و `state` های اون کامپوننت رو فیلتر میکنه. اگه هیچ کامپوننت `mount` شده ای توی `container` وجود نداشته باشه، فراخوانی این تابع هیچ کاری رو انجام نمیده. اگه کامپوننت `unmount` شده ای وجود داشت `true` رو بر می گردونه و اگه هیچ کامپوننتی برای `unmount` شدن وجود نداشت `false` رو برمی گردونه. امضای متد به صورت زیر هستش،

```
ReactDOM.unmountComponentAtNode(container);
```

↑ فهرست مطالب

## 250. `code-splitting` چیه؟

`code-splitting` ویژگی پشتیبانی شده توسط باندلرهایی مثل `webpack` و `browserify` هستش که می تونه بسته های مختلفی ایجاد کنه که می تونه به صورت پویا در زمان اجرا بارگیری بشه. ری اکت `code-splitting` رو از طریق ویژگی `dynamic import` () پشتیبانی میکنه.

برای مثال، در قطعه کد زیر، `moduleA.js` و تمام وابستگی های منحصر به فرد اون رو به عنوان یه قطعه جداگانه ایجاد میکنه که فقط بعد از کلیک کاربر روی دکمه 'Load' بارگیری میشه.

**moduleA.js**

```
const moduleA = "Hello";

export { moduleA };
```

**\*\*App.js\*\***

```
import React, { Component } from "react";

class App extends Component {
 handleClick = () => {
 import("./moduleA")
 .then(({ moduleA }) => {
 // Use moduleA
 })
 .catch((err) => {
 // Handle failure
 });
 };

 render() {
 return (
 <div>
 <button onClick={this.handleClick}>Load</button>
 </div>
);
 }
}

export default App;
```

[↑ فهرست مطالب](#)

## 251. مزایای حالت strict چیه؟

توی موارد زیر به کار میاد

1. شناسایی کامپوننت‌ها با متد **unsafe lifecycle**.
2. هشدار در مورد استفاده از API مربوط به **legacy string ref**.
3. تشخیص **side effect** های غیرمنتظره.
4. شناسایی **API legacy context**.
5. هشدار در مورد استفاده منسوخ **findDOMNode**.

[↑ فهرست مطالب](#)

## 252. Fragment های دارای key هستن؟

Fragment های اعلام شده با سینتکس <React.Fragment> ممکنه key هایی داشته باشن. استفاده عمومی مپ کردن یه مجموعه به آرایه ای از fragment ها به صورت زیر هستش،

```
function Glossary(props) {
 return (
 <dl>
 {props.items.map((item) => (
 // Without the `key`, React will fire a key warning
 <React.Fragment key={item.id}>
 <dt>{item.term}</dt>
 <dd>{item.description}</dd>
 </React.Fragment>
))}
 </dl>
);
}
```

**\*\*یادداشت\*\*** key تنها اتریبیوتی هستش که میشه به Fragment انتقال داد. در آینده، ممکنه از اتریبیوت های اضافه ای هم مثل event handler پشتیبانی بشه.

↑ فهرست مطالب

## 253. آیا ری اکت از همه ی attribute های HTML پشتیبانی می کنه؟

از ری اکت 16، هر دو ویژگی استاندارد یا سفارشی DOM کاملاً پشتیبانی میشن. از اونجایی که کامپوننت های ری اکت اغلب هر دو نوع پراپ های DOM-related و custom رو استفاده می کنن، ری اکت دقیقاً مانند API های DOM از قرارداد camelCase استفاده میکنه. بیاین با استفاده از ویژگی های استاندارد HTML چند مورد رو انتخاب کنیم.

```
<div tabIndex="-1" /> // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} /> // Just like node.readOnly DOM API
```

این prop ها به استثنای موارد خاص، مشابه ویژگی های متناظر HTML کار می کنن. همچنین از تمام ویژگی های svg و پشتیبانی می کنه.

↑ فهرست مطالب

## 254. محدودیت های HOC ها چی هستن؟

کامپوننت‌های با اولویت بالا جدا از مزایایی که دارد، چند تا نکته مهم هم دارد. اینجا چند مورد به ترتیب گفته شده

## 1. از HOC ها توی متد render استفاده نکنیم:

استفاده از HOC توی یه کامپوننت با متد رندر اون کامپوننت توصیه نمیشه.

```
} () render
new version of EnhancedComponent is created on every render //
EnhancedComponent1 !== EnhancedComponent2 //
;(const EnhancedComponent = enhance(MyComponent
!That causes the entire subtree to unmount/remount each time //
;</ return <EnhancedComponent
{
```

کد بالا با remount کردن کامپوننتی که باعث از بین رفتن state اون کامپوننت و همه فرزندانش شده، روی عملکرد تاثیر میذاره. در عوض، HOC ها رو بیرون از تعریف کامپوننت اعمال می‌کنیم تا کامپوننت بدست اومده فقط یه بار ساخته بشه.

## 2. متدهای static باید کپی بشن

وقتی HOC رو روی یه کامپوننت اعمال می‌کنیم، کامپوننت جدید هیچ کدوم از متدهای استاتیک کامپوننت اصلی رو نداره

```
Define a static method //
} () WrappedComponent.staticMethod = function
/*...*/
;{
Now apply a HOC //
;(const EnhancedComponent = enhance(WrappedComponent

The enhanced component has no static method //
typeof EnhancedComponent.staticMethod === "undefined"; // true
```

میتونیم با کپی کردن متدها توی container قبل از return کردنش رو این مشکل غلبه کنیم.

```

javascript``
} (function enhance(WrappedComponent
} class Enhance extends React.Component
/*...*/
{
):Must know exactly which method(s) to copy //
;Enhance.staticMethod = WrappedComponent.staticMethod
;return Enhance
{
``


```

3. **Ref** ها رو همیشه انتقال داد: **\*\***

برای HOC ها نیاز داریم که همه **prop** ها رو به کامپوننت پاس بدیم اما در مورد

**\*\*** [فهرست] (#فهرست) **\*\***

## 255. چطوری همیشه forwardRefs رو توی DevTools دیباگ کرد؟

**React.forwardRef** یه تابع رندر رو به عنوان یه پارامتر میگیره و DevTools از این تابع برای تعیین اینکه چه چیزی باید برای ref forwarding component نمایش داده بشه، استفاده میکنه. برای مثال، اگه ما هیچ اسمی برای تابع رندر نداریم یا از ویژگی displayName استفاده نکنیم، توی DevTools به عنوان "ForwardRef" نمایش داده میشه.

```

const WrappedComponent = React.forwardRef((props, ref) => {
 return <LogProps {...props} forwardedRef={ref} />;
});

```

اما اگه برای تابع رندر اسم گذاشته باشیم اونوقت به صورت **\*(ForwardRef(myFunction))\*** نمایش داده میشه

```

const WrappedComponent = React.forwardRef(function myFunction(prop
 return <LogProps {...props} forwardedRef={ref} />;
});

```

به عنوان یه گزینه دیگه، میتونیم از ویژگی displayName برای تابع forwardRef استفاده کنیم.



```
function logProps(Component) {
 class LogProps extends React.Component {
 //...
 }

 function forwardRef(props, ref) {
 return <LogProps {...props} forwardedRef={ref} />;
 }

 // Give this component a more helpful display name in DevTools.
 // e.g. "ForwardRef(logProps(MyComponent))"
 const name = Component.displayName || Component.name;
 forwardRef.displayName = `logProps(${name})`;

 return React.forwardRef(forwardRef);
}
```

[↑ فهرست مطالب](#)

## 256. مقدار یه props کامپوننت کی true میشه؟

اگه هیچ مقداری رو برای prop انتقال ندیم، به طور پیش فرض true در نظر گرفته میشه. این رفتار در دسترس هستش طوری که با رفتار HTML هم مطابقت داره. به طور مثال، عبارت‌های زیر معادل هم هستن.

```
<MyInput autocomplete />
<MyInput autocomplete={true} />
```

**\*\*یادداشت:\*\*** این مورد توصیه نمیشه چون ممکنه با مختصر نویسی ES6 اشتباه گرفته بشه (مثال، {name: name} مخفف {name: name})

[↑ فهرست مطالب](#)

## 257. NextJS چیه و ویژگی‌های اصلیش کدوما هستن؟

Next.js یه فریمورک محبوب و سبک برای برنامه‌های استاتیک و تحت سرور هستش که توسط ری اکت ساخته شده. همچنین استایل دهی و مسیریابی رو هم ارائه میده. اینجا ویژگی‌های اصلی ارائه شده توسط Next.js آورده شده.

1. server rendering به طور پیش فرض ارائه شده
2. تقسیم خودکار کد برای بارگذاری سریعتر صفحه
3. مسیریابی ساده سمت مشتری (مبتنی بر صفحه)
4. محیط توسعه یافته مبتنی بر بسته وب (HMR)
5. با Express یا هر سرور HTTP دیگه‌ای Node.js قابل پیاده سازی
6. با تنظیمات Babel و Webpack خودمون قابل تنظیمه

↑ فهرست مطالب

## 258. چط،وی کی تونیم یه تابع event handler رو به یه کامپوننت پاس بدیم؟

event handler ها و توابع دیگه رو میتونیم به عنوان prop به کامپوننت‌های فرزند انتقال بدیم. به صورت زیر توی کامپوننت فرزند می‌تونه استفاده بشه،

```

...
<{button onClick={this.handleClick}>
...

[فهرست] (#فهرست)

```

## 259. استفاده از توابع arrow برای متدهای render خوبه؟

بله، می‌تونیم استفاده کنیم. این معمولا ساده ترین راه برای انتقال پارامترها به توابع برگشتی هستش. اما در حین استفاده باید عملکرد را بهینه کنیم.

```

class Foo extends Component {
 handleClick() {
 console.log("Click happened");
 }
 render() {
 return <button onClick={() => this.handleClick()}>Click Me</button>;
 }
}

```

\*\*یادداشت:\*\* استفاده از تابع arrow توی متد رندر یه تابع جدید ایجاد میکنه که هر بار که کامپوننت رندر میشه، ممکنه مفاهیم عملکردی داشته باشه.

## 260. چطوری از اجرای چندباره یه تابع جلوگیری کنیم؟

If you use an event handler such as **onClick** or **onScroll** and want to prevent the callback from being fired too quickly, then you can limit the rate at which the callback is executed. This can be achieved in the below possible ways

**Throttling:** Changes based on a time based frequency. For .1 example, it can be used using `_throttle` lodash function

**Debouncing:** Publish changes after a period of inactivity. For .2 example, it can be used using `_debounce` lodash function

**RequestAnimationFrame throttling:** Changes based on .3 requestAnimationFrame. For example, it can be used using `raf-schd` lodash function

## 261. JSX چطوری از حمله‌های Injection جلوگیری می‌کنه؟

React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered. ,For example, you can embed user input as below

```
const name = response.potentiallyMaliciousInput;
const element = <h1>{name}</h1>;
```

.This way you can prevent XSS(Cross-site-scripting) attacks in the application

## 262. چطوری element های رندر شده رو آپدیت کنیم؟

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method. For example, lets take a

ticking clock example, where it updates the time by calling render method  
,multiple times

```
function tick() {
 const element = (
 <div>
 <h1>Hello, world!</h1>
 <h2>It is {new Date().toLocaleTimeString()}</h2>
 </div>
);
 ReactDOM.render(element, document.getElementById("root"));
}

setInterval(tick, 1000);
```

[↑ فهرست مطالب](#)

## 263. چرا prop ها read only هستند؟

When you declare a component as a function or a class, it must never modify  
,its own props. Let us take a below capital function

```
function capital(amount, interest) {
 return amount + interest;
}
```

The above function is called “pure” because it does not attempt to change their  
inputs, and always return the same result for the same inputs. Hence, React has a  
single rule saying "All React components must act like pure functions with respect  
".to their props

[↑ فهرست مطالب](#)

## 264. چرا می‌گیم تابع setState از طریق merge کردن state را مدیریت می‌کنه؟

When you call setState() in the component, React merges the object you  
provide into the current state. For example, let us take a facebook user with  
,posts and comments details as state variables

```

constructor(props) {
 super(props);
 this.state = {
 posts: [],
 comments: []
 };
}

```

,Now you can update them independently with separate `setState()` calls as below

```

componentDidMount() {
 fetchPosts().then(response => {
 this.setState({
 posts: response.posts
 });
 });

 fetchComments().then(response => {
 this.setState({
 comments: response.comments
 });
 });
}

```

As mentioned in the above code snippets, `this.setState({comments})` updates only `.comments` variable without modifying or replacing `posts` variable

[↑ فهرست مطالب](#)

## 265. چطوری می‌تونیم به متد `event handler` پارامتر پاس بدیم؟

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or bind method.

,Let us take an example of user details updated in a grid

```

:={ (e) => this.updateUser(userId, e)}>Update User details</button>
:={this.updateUser.bind(this, userId)}>Update User details</button>

```

In both the approaches, the synthetic argument `e` is passed as a second argument. You need to pass it explicitly for arrow functions and it forwarded automatically for bind method

## 266. چطوری از رندر مجدد کامپوننت‌ها جلوگیری کنیم؟

You can prevent component from rendering by returning null based on .specific condition. This way it can conditionally render component

```
function Greeting(props) {
 if (!props.loggedIn) {
 return null;
 }

 return <div className="greeting">welcome, {props.name}</div>;
}
```

```
class User extends React.Component {
 constructor(props) {
 super(props);
 this.state = {loggedIn: false, name: 'John'};
 }

 render() {
 return (
 <div>
 //Prevent component render if it is not loggedIn
 <Greeting loggedIn={this.state.loggedIn} />
 <UserDetails name={this.state.name}>
 </div>
);
 }
}
```

In the above example, the greeting component skips its rendering section by .applying condition and returning null value

## 267. شرایطی که بدون مشکل پرفورمنس بتونیم از ایندکس به عنوان key استفاده کنیم چی هست؟

.There are three conditions to make sure, it is safe use the index as a key

- 1. The list and items are static– they are not computed and do not change
- 2. The items in the list have no ids
- 3. The list is never reordered or filtered

[↑ فهرست مطالب](#)

## 268. **key**های ری‌اکت باید به صورت عمومی منحصر بفرد باشن؟

Keys used within arrays should be unique among their siblings but they don't need to be globally unique. i.e, You can use the same keys with two different arrays. For example, the below book component uses two arrays with different arrays

```
function Book(props) {
 const index = (

 {props.pages.map((page) => (
 <li key={page.id}>{page.title}
))}

);
 const content = props.pages.map((page) => (
 <div key={page.id}>
 <h3>{page.title}</h3>
 <p>{page.content}</p>
 <p>{page.pageNumber}</p>
 </div>
));
 return (
 <div>
 {index}
 <hr />
 {content}
 </div>
);
}
```

[↑ فهرست مطالب](#)

## 269. گزینه‌های محبوب برای مدیریت فرم‌ها توی ری‌اکت کدوما هستن؟

Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission. In detail, You can categorize them as follows

1. Getting values in and out of form state

2. Validation and error messages

3. Handling form submission

It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff

↑ فهرست مطالب

## 270. مزایای کتابخانه فرمیک نیست به redux form چیه؟

Below are the main reasons to recommend formik over redux form library

1. The form state is inherently short-term and local, so tracking it in

Redux (or any kind of Flux library) is unnecessary

2. Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases input latency for large apps

3. Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

↑ فهرست مطالب

## 271. چرا اجباری برای استفاده از ارث‌بری توی ری‌اکت نیست؟ مزیتی داره؟

In React, it is recommend using composition instead of inheritance to reuse code between components. Both Props and composition give you all the flexibility you need to customize a component's look and behavior in an explicit and safe way



Whereas, If you want to reuse non-UI functionality between components, it is suggested to extracting it into a separate JavaScript module. Later components import it and use that function, object, or a class, without .extending it

[↑ فهرست مطالب](#)

## 272. می‌تونیم از web components توی برنامه ری‌اکت استفاده کنیم؟

Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components. For example, let us use Vaadin date picker web component as below

```
import React, { Component } from "react";
import "./App.css";
import "@vaadin/vaadin-date-picker";
class App extends Component {
 render() {
 return (
 <div className="App">
 <vaadin-date-picker label="When were you born?"></vaadin-
 </div>
);
 }
}
export default App;
```

[↑ فهرست مطالب](#)

## 273. dynamic import چیه؟

The dynamic import() syntax is a ECMAScript proposal not currently part of the language standard. It is expected to be accepted in the near future. You can achieve code-splitting into your app using dynamic import(). Let's take an example of addition

**Normal Import .1**

```
import { add } from "./math";
console.log(add(10, 20));
```

**\*\*Dynamic Import\*\*** .2

```
import("./math").then((math) => {
 console.log(math.add(10, 20));
});
```

[↑ فهرست مطالب](#)

## 274. **loadable component** چي هستن؟

If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic ,import as a regular component. Lets take an example

```
import loadable from "@loadable/component";

const OtherComponent = loadable(() => import("./OtherComponent"))

function MyComponent() {
 return (
 <div>
 <OtherComponent />
 </div>
);
}
```

Now OtherComponent will be loaded in a separated bundle

[↑ فهرست مطالب](#)

## 275. **suspense** کامپوننت چيه؟

If the module containing the dynamic import is not yet loaded by the time parent component renders, you must show some fallback content while you're waiting for it to load using a loading indicator. This can be done using

**Suspense** component. For example, the below code uses suspense component

```
const OtherComponent = React.lazy(() => import("./OtherComponent"))

function MyComponent() {
 return (
 <div>
 <Suspense fallback={<div>Loading...</div>}>
 <OtherComponent />
 </Suspense>
 </div>
);
}
```

.As mentioned in the above code, Suspense is wrapped above the lazy component

[↑ فهرست مطالب](#)

## 276. چطوری به ازای route می‌تونیم code splitting داشته باشیم؟

One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed. Let us take an example of route based website using libraries like React Router with React.lazy

```
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import React, { Suspense, lazy } from "react";

const Home = lazy(() => import("./routes/Home"));
const About = lazy(() => import("./routes/About"));

const App = () => (
 <Router>
 <Suspense fallback=<div>Loading...</div>>
 <Switch>
 <Route exact path="/" component={Home} />
 <Route path="/about" component={About} />
 </Switch>
 </Suspense>
 </Router>
);
```

.In the above code, the code splitting will happen at each route level

[↑ فهرست مطالب](#)

## 277. یه مثال از نحوه استفاده از context میزنی؟

**Context** is designed to share data that can be considered **global** for a tree of React components. For example, in the code below lets manually thread through a "theme" prop in order to style the Button component

```
// Lets create a context with a default theme value "luna"
const ThemeContext = React.createContext("luna");
// Create App component where it uses provider to pass theme value
class App extends React.Component {
 render() {
 return (
 <ThemeContext.Provider value="nova">
 <Toolbar />
 </ThemeContext.Provider>
);
 }
}
// A middle component where you don't need to pass theme prop any
function Toolbar(props) {
 return (
 <div>
 <ThemedButton />
 </div>
);
}
// Lets read theme value in the button component to use
class ThemedButton extends React.Component {
 static contextType = ThemeContext;
 render() {
 return <Button theme={this.context} />;
 }
}
```

[↑ فهرست مطالب](#)

## 278. هدف از مقدار پیش فرض توی context چیه؟

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them. Below code snippet provides default theme value as Luna.

```
const MyContext = React.createContext(defaultValue);
```

[↑ فهرست مطالب](#)

## 279. چھوری از contextType استفاده می‌کنین؟

ContextType is used to consume the context object. The contextType property can be used in two ways

### 1. contextType as property of class

The contextType property on a class can be assigned a Context object created by React.createContext(). After that, you can consume the nearest current value of that Context type using this.context in any of the lifecycle methods and render function. Lets assign contextType property on MyClass as below

```
class MyClass extends React.Component {
 componentDidMount() {
 let value = this.context;
 /* perform a side-effect at mount using the value of MyContext */
 }
 componentDidUpdate() {
 let value = this.context;
 /*... */
 }
 componentWillUnmount() {
 let value = this.context;
 /*... */
 }
 render() {
 let value = this.context;
 /* render something based on the value of MyContext */
 }
}
MyClass.contextType = MyContext;
```

2. Static field\*\* You can use a static class field to initialize your contextType\*\* using public class field syntax

```
class MyClass extends React.Component {
 static contextType = MyContext;
 render() {
 let value = this.context;
 /* render something based on the value */
 }
}
```

↑ فهرست مطالب

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree. Lets take a simple example

```
<MyContext.Consumer>
 {value => /* render something based on the context value */}
</MyContext.Consumer>
```

[↑ فهرست مطالب](#)

## 281. چطوری مسائل مربوط به پرفورمنس با context رو حل می‌کنین؟

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider's parent re-renders. For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value

```
class App extends React.Component {
 render() {
 return (
 <Provider value={{ something: "something" }}>
 <Toolbar />
 </Provider>
);
 }
}
```

,This can be solved by lifting up the value to parent state

```

class App extends React.Component {
 constructor(props) {
 super(props);
 this.state = {
 value: { something: "something" },
 };
 }

 render() {
 return (
 <Provider value={this.state.value}>
 <Toolbar />
 </Provider>
);
 }
}

```

[↑ فهرست مطالب](#)

## 282. هدف از forward ref توی HOC ها چیه؟

ref داخل کامپوننت‌ها پاس داده نمیشه چون ref یه prop نیست. اون توسط ری‌اکت درست مثل **key** به طور متفاوتی هندل میشه. اگه ما ref رو توی HOC اضافه کنیم، ref به بیرونی‌ترین کامپوننت container اشاره میکنه، نه به کامپوننت wrapped شده. تو این مورد ما می‌تونیم از Forward Ref API استفاده کنیم. برای مثال با استفاده از React.forwardRef API میتونیم ref رو به کامپوننت FancyButton داخلی بفرستیم.



```
function logProps(Component) {
 class LogProps extends React.Component {
 componentDidUpdate(prevProps) {
 console.log("old props:", prevProps);
 console.log("new props:", this.props);
 }

 render() {
 const { forwardedRef, ...rest } = this.props;

 // Assign the custom prop "forwardedRef" as a ref
 return <Component ref={forwardedRef} {...rest} />;
 }
 }

 return React.forwardRef((props, ref) => {
 return <LogProps {...props} forwardedRef={ref} />;
 });
}
```

,Let's use this HOC to log all props that get passed to our “fancy button” component

```
class FancyButton extends React.Component {
 focus() {
 //...
 }

 //...
}
export default logProps(FancyButton);
```

حالا بیاین به ref بسازیم و اونو به کامپوننت FancyButton بفرستیم. توی این مورد می‌تونیم focus رو روی عنصر دکمه تنظیم کنیم.

```
import FancyButton from "./FancyButton";

const ref = React.createRef();
ref.current.focus();
<FancyButton label="Click Me" handleClick={handleClick} ref={ref}>
```

[↑ فهرست مطالب](#)

283. توی کامپوننت‌ها می‌تونیم پراپ ref داشته باشیم؟

توابع منظم یا کلاس کامپوننت‌ها آرگومان `ref` رو دریافت نمی‌کنن و `ref` توی `prop`‌ها هم در دسترس نیست. آرگومان دوم `ref` فقط زمانی وجود داره که ما کامپوننت رو با `React.forwardRef` تعریف کنیم.

↑ فهرست مطالب

## 284. چرا در هنگام استفاده از `ForwardRef`‌ها نیاز به احتیاط بیشتری در استفاده از کتابخانه‌های جانبی داریم؟

وقتی ما شروع به استفاده از `forwardRef` توی یه کامپوننت می‌کنیم، باید با اون به عنوان یه تغییر سریع رفتار کنیم و نسخه اصلی جدیدی از کتابخونه خودمون رو منتشر کنیم. این به این دلیل که کتابخونه ما رفتار متفاوتی داره مثل اینکه چه چیزی به `ref` اختصاص پیدا کرده و چه خروجی‌هایی داریم. این تغییرات می‌تونه برنامه‌ها و بقیه کتابخونه‌های وابسته به رفتار قدیمی رو از بین ببره.

↑ فهرست مطالب

## 285. چطوری بدون استفاده از `ES6` کلاس کامپوننت بسازیم؟

اگه از `ES6` استفاده نمی‌کنیم ممکنه لازم باشه که به جای اون از `create-react-class` استفاده کنیم. برای `prop`‌های پیش فرض، نیاز داریم که `getDefaultProps()` رو به عنوان یه تابع روی آبجکت پاس داده شده تعریف کنیم. در حالی که برای `state` اولیه، باید یه متد `getInitialState` جداگانه ارائه بدیم که یه `state` اولیه برمی‌گردونه.

```
var Greeting = createReactClass({
 getDefaultProps: function () {
 return {
 name: "Jhohn",
 };
 },
 getInitialState: function () {
 return { message: this.props.message };
 },
 handleClick: function () {
 console.log(this.state.message);
 },
 render: function () {
 return <h1>Hello, {this.props.name}</h1>;
 },
});
```

**\*\*یادداشت: \*\*** اگر از `createReactClass` استفاده می‌کنیم اتصال خودکار برای همه روش‌ها در دسترسه. یعنی نیازی به استفاده از `bind(this)` توی `constructor` برای `event handler`‌ها نیست.

[↑ فهرست مطالب](#)

## 286. استفاده از ری‌اکت بدون JSX ممکن است؟

بله، JSX برای استفاده از ری‌اکت اجباری نیست. در واقع مناسب زمانی هست که ما نمی‌خواهیم کامپایلی رو توی محیط `build` تنظیم کنیم. هر عنصر JSX فقط syntactic sugar هستش برای فراخوانی `React.createElement(component, props, ...children)`. برای مثال بیاین یه مثال با `greeting` با JSX بزنیم.

```
class Greeting extends React.Component {
 render() {
 return <div>Hello {this.props.message}</div>;
 }
}

ReactDOM.render(
 <Greeting message="World" />,
 document.getElementById("root")
);
```

میتونیم همین کد رو بدون JSX مثل زیر بنویسیم،

```
class Greeting extends React.Component {
 render() {
 return React.createElement("div", null, `Hello ${this.props.m}`);
 }
}

ReactDOM.render(
 React.createElement(Greeting, { message: "World" }, null),
 document.getElementById("root")
);
```

[↑ فهرست مطالب](#)

## 287. الگوریتم‌های diffing ری‌اکت چی هستن؟

ری‌اکت نیاز به استفاده از الگوریتم‌ها داره تا بفهمه چطور به طور موثر UI رو برای مطابقت با آخرین درخت به‌روز کنه. الگوریتم‌های مختلفی در حال تولید حداقل تعداد عملیات برای تبدیل یه درخت به درخت دیگه هستن. با این حال، الگوریتم‌ها به ترتیب  $O(n^3)$  دارای پیچیدگی هستن، جایی که  $n$  تعداد عناصر موجود در درخت هستش. توی این مورد، برای نمایش ۱۰۰۰ عنصر به ترتیب یک میلیارد مقایسه نیازه و این خیلی هزینه بر هستش. در عوض ری‌اکت یه الگوریتم ابتکاری  $O(n)$  رو بر اساس دو پیش فرض پیاده‌سازی میکنه:

1. دو عنصر از انواع مختلف باعث تولید درخت‌های مختلفی میشه.
2. برنامه نویس می‌تونه اشاره کنه که کدوم یکی از عناصر فرزند ممکنه توی رندرهای مختلف با یه `prop` اصلی پایدار باشن.

[فهرست](#)

## 288. قوانینی که توسط الگوریتم‌های diffing پوشش داده می‌شوند کدام هستن؟

موقع تفاوت بین دو درخت، ری‌اکت اول دو عنصر ریشه رو با هم مقایسه میکنه. رفتار بسته به انواع عناصر ریشه تغییر میکنه. مواردی که اینجا گفته شده قوانینی از الگوریتم reconciliation هستن.

1. عناصر با انواع مختلف:

هر وقت عناصر ریشه انواع مختلفی داشته باشن، ری‌اکت درخت قبلی رو از بین میبره و درخت جدید رو از اول میسازه. برای مثال، عناصر [تا](#)



یا از

تا از انواع مختلف باعث بازسازی کامل میشن.

## 2. عناصر DOM از همان نوع

موقع مقایسه دو عنصر React DOM از همون نوع، React به ویژگی‌های هر دو نگاه می‌کند، همون گره DOM زیرین رو نگه میداره و فقط ویژگی‌های تغییر یافته رو به روز میکنه. بیاین یه مثال با عناصر DOM مشابه به جز ویژگی className بیاریم،

```
<div className="show" title="ReactJS" />
<div className="hide" title="ReactJS" />
```

3. **\*\*عناصر کامپوننت از همان نوع: \*\*** وقتی کامپوننت به‌روز میشه، نمونه ثابت می‌مونه، بنابراین state بین رندر ها حفظ میشه. ری‌اکت برای مطابقت با عنصر جدید prop های نمونه کامپوننت اساسی رو به‌روز میکنه و متدهای componentWillReceiveProps () و componentWillUpdate () رو روی نمونه اصلی صدا می‌زنه. بعد از اون متد render () صدا زده میشه و الگوریتم diff، نتیجه قبلی و نتیجه جدید رو جستجو میکنه. 4. **\*\* when recursing on the children of a\*\*** Recursing On Children:

DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there's a difference. For example, when adding an element at the end of the children, converting between these two trees works well

```

 first
 second

 first
 second
 third

```

## 5. **\*\* هندل کردن کلیدها \*\***

ری‌اکت از ویژگی key پشتیبانی میکنه. وقتی فرزندان key داشته باشن، ری‌اکت از key برای مطابقت دادن فرزندان در درخت اصلی با فرزندان در درخت بعدی استفاده میکنه. برای مثال، اضافه کردن یه key می‌تونه تبدیل درخت رو کارآمد کنه،

```


 <li key="2015">Duke
 <li key="2016">Villanova

 <li key="2014">Connecticut
 <li key="2015">Duke
 <li key="2016">Villanova


```

↑ فهرست مطالب

## 289. چه موقعی نیاز هست که از ref ها استفاده کنیم؟

موارد استفاده کمی برای ref ها وجود دارد

1. مدیریت focus، text selection یا پخش media
2. راه اندازی انیمیشن های ضروری.
3. ادغام با کتابخانه های third-party DOM.

↑ فهرست مطالب

## 290. برای استفاده از render prop لازم که اسم prop رو render بزاریم؟

حتی اگه یه الگویی به اسم render props وجود داشته باشه، برای استفاده از این الگو نیازی به استفاده از یه prop به اسم render نیست. به عنوان مثال، هر prop که تابعی باشه، که کامپوننتی از اون برای دونستن اینکه چه چیزی باید ارائه بده استفاده کنه، از نظر فنی "render prop" هستش. بیاین یه مثال در مورد prop فرزند برای رندر prop بزنیم

```

<Mouse
 children={({mouse}) => (
 <p>
 The mouse position is {mouse.x}, {mouse.y}
 </p>
)}
/>

```

در واقع نیازی نیست که از prop فرزند توی لیست "attribute" ها توی عنصر JSX نام برده بشه. در عوض میتونیم اونی مستقیماً توی المنت نگه داریم.

```
<Mouse>
 {(mouse) => (
 <p>
 The mouse position is {mouse.x}, {mouse.y}
 </p>
)}
</Mouse>
```

وقتی که از روش بالا (تابع بدون نام) برای رندر کردن فرزند استفاده می‌کنیم، به صراحت و اجبار می‌گیم که فرزند پاس داده شده، باید یه تابع توی propType همون باشن.

```
Mouse.propTypes = {
 children: PropTypes.func.isRequired,
};
```

[↑ فهرست مطالب](#)

## 291. مشکل استفاده از render props با pure component ها چیه؟

اگه بیایم داخل متد رندر یه تابعی ایجاد کنیم، در این صورت هدف اصلی pure component ها رو نفی کردیم. چون که مقایسه سطحی prop ها معمولاً همیشه مقدار false رو برای prop های جدید برمی‌گردونه و هر رندر در این حالت یه مقدار جدیدی رو برای رندر ارائه میده. با تعریف یه تابع رندر به عنوان متد instance میتونیم این مشکل رو حل کنیم.

[↑ فهرست مطالب](#)

## 292. چطوری با استفاده از render props می‌تونیم HOC ایجاد کنیم؟

میتونیم کامپوننت‌های با الویت بالا (HOC) رو با استفاده از یه کامپوننت معمولی با یه رندر پیاده سازی کنیم. به عنوان مثال اگه ترجیح میدیم که به جای کامپوننت یه کامپوننت HOC به اسم withMouse داشته باشیم، به راحتی میتونیم با استفاده از کامپوننت با prop رندر، یکی بسازیم.

```
function withMouse(Component) {
 return class extends React.Component {
 render() {
 return (
 <Mouse
 render={mouse => <Component {...this.props} mouse={mouse}
 />
);
 }
 };
}
```

این روش رندر کردن prop ها، انعطاف پذیری استفاده از هر دو الگو رو میده.

[↑ فهرست مطالب](#)

## 293. تکنیک windowing چیه؟

windowing تکنیکیه که فقط زیر مجموعه ای از سطرهامون رو در هر زمان ارائه میده و می‌تونه مدت زمان لازم برای رندر مجدد کامپوننت‌ها و همینطور تعداد گره‌های DOM ایجاد شده روبه طرز چشمگیری کاهش بده. اگه برنامه مون لیست‌های طولانی ای از داده رو ارائه میده، این روش توصیه میشه. react-window و react-virtualized هر دو کتابخونه‌های معروف windowing هستن که چندین کامپوننت قابل استفاده مجدد رو برای نمایش لیست ها، شبکه‌ها و داده‌های جدولی فراهم می‌کنن.

[↑ فهرست مطالب](#)

## 294. توی JSX به مقدار falsy رو چطوری چاپ کنیم؟

مقادیر جعلی مثل null، undefined، false و true معتبر هستند ولی هیچ چیزی رو رندر نمی‌کنن. اگه بخوایم اونا رو نمایش بدیم باید به رشته تبدیلشون کنیم. بیاین یه مثال در مورد تبدیل به رشته بزنیم،

```
<div>My JavaScript variable is {String(myVariable)}.</div>
```

[↑ فهرست مطالب](#)



## 295. یه مورد استفاده معمول از portals مثال میزنی؟

React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context (z-index, position, opacity etc styles) and you need to visually “break out” of its container. For example, dialogs, global message notifications, hovercards, and tooltips.

↑ فهرست مطالب

## 296. توی کامپوننت‌های کنترل نشده چطوری مقداری پیش فرض اضافه کنیم؟

توی ری‌اکت، مشخصه value روی عناصر فرم مقدار رو توی DOM لغو میکنه. با یه کامپوننت کنترل نشده، ممکنه بخوایم ری‌اکت یه مقدار اولیه مشخص کنه ولی به روزرسانی‌های بعدی رو کنترل نشده بذاره. برای هندل کردن این مورد، میتونیم به جای value مشخصه defaultValue رو تعیین کنیم.

```
render() {
 return (
 <form onSubmit={this.handleSubmit}>
 <label>
 User Name:
 <input
 defaultValue="John"
 type="text"
 ref={this.input} />
 </label>
 <input type="submit" value="Submit" />
 </form>
);
}
```

همین کار برای select و textarea هم انجام میشه ولی برای checkbox باید از defaultchecked استفاده کنیم.

↑ فهرست مطالب

## 297. stack مورد علاقه شما برای کانفیگ پروژه ری‌اکت چیه؟

حتی اگر tech stack از توسعه دهنده ای به توسعه دهنده دیگر متفاوت باشد، معروف ترین stack توی کد پروژه boilerplate ری اکت استفاده شده. boilerplate به طور عمده از ریداکس و ریداکس ساکا برای مدیریت استیت و ساید افکت های ناهمزمان، styled-components برای استایل دهی کامپوننت ها، axios برای فراخوانی rest api و پشتیبانی های دیگر از قبیل ESNEXT، reselect، webpack و babel. میتونیم پروژه <https://github.com/react-boilerplate/react-boilerplate> رو کلون کنیم و کار روی هر پروژه ری اکت جدیدی رو شروع کنیم.

↑ فهرست مطالب

## 298. تفاوت DOM واقعی و Virtual DOM چیه؟

اینجا تفاوت های اصلی بین DOM واقعی و DOM مجازی گفته شده:  
واقعی DOM:

1. به روز رسانی ها کند هستن
  2. دستکاری DOM هزینه بر هستش.
  3. می تونیم HTML رو مستقیماً به روز رسانی کنیم.
  4. باعث اتلاف بیش از حد حافظه میشه.
  5. در صورت به روز رسانی یه المنت، یه DOM جدید ایجاد میکنه.
- مجازی DOM:

1. به روز رسانی ها سریع هستن
2. دستکاری DOM خیلی راحت.
3. HTML رو نمیتونیم مستقیماً به روز رسانی کنیم.
4. هیچ اتلاف حافظه ای وجود نداره.
5. در صورت به روز رسانی یه المنت، JSX رو به روز میکنه.

↑ فهرست مطالب

## 299. چطوری Bootstrap رو به یه برنامه ری اکتی اضافه کنیم؟

Bootstrap رو به سه روش میتونیم به برنامه ری اکت اضافه کنیم

1. با استفاده از Bootstrap CDN
- این ساده ترین راه برای اضافه کردن bootstrap هستش. منابع bootstrap css و js رو توی تگ head اضافه می کنیم.

2. Bootstrap as Dependency :

3. bootstrap به عنوان dependency

اگر از یه ابزار build یا بسته نرم افزاری مازولی مثل webpack استفاده می‌کنیم، این بهترین گزینه برای اضافه کردن bootstrap به برنامه ری‌اکت هستش.

```
npm install bootstrap
```

3. بسته React Bootstrap :

در این حالت می‌تونیم bootstrap رو به برنامه ری‌اکت اضافه کنیم تا با استفاده از بسته‌هایی که کامپوننت‌های ری‌اکت رو دوباره ساخته تا منحصر به عنوان کامپوننت‌های ری‌اکت کار کنند. معروف ترین بسته‌ها برای این کار اینا هستند

1. react-bootstrap

2. reactstrap

↑ فهرست مطالب

## 300. می‌تونن یه لیستتی از معروف‌ترین وب‌سایت‌هایی که از ری‌اکت استفاده می‌کنن رو بگی؟

این زیر یه لیست از 10 وب‌سایت مشهور که از ری‌اکت برای فرانت‌اندشون استفاده می‌کنن رو لیست می‌کنیم:

1. Facebook

2. Uber

3. Instagram

4. WhatsApp

5. Khan Academy

6. Airbnb

7. Dropbox

8. Flipboard

9. Netflix

10. PayPal

↑ فهرست مطالب

## 301. استفاده از تکنیک CSS In JS تو ری اکت توصیه میشه؟

ری اکت هیچ ایده‌ای راجع به اینکه استایل‌ها چطوری تعریف شدن نداره اما اگه تازه کار باشین می‌تونین از یه فایل جداگانه `css.*` که مث قبلا توی پروژه‌های ساده استفاده می‌شد کمک بگیرین و با استفاده از `className` از استایل‌ها استفاده کنین. `CSS In Js` یه بخش از خود ری اکت نیست و توسط کتابخونه‌های `third-party` بهش اضافه شده اما اگه می‌خوایین. ازش (`CSS-In-JS`) استفاده کنین کتابخونه `styled-components` می‌تونه گزینه خوبی باشه.

↑ فهرست مطالب

## 302. لازمه همه کلاس کامپوننت‌ها رو تبدیل کنیم به هوک؟

نه. ولی می‌تونین از هوک‌ها توی بعضی از کامپوننت‌های قدیمی یا جدید استفاده کنین و سعی کنین باهاش راحت باشین البته برنامه‌ای برای حذف `classes` از ری اکت هنوز وجود نداره.

↑ فهرست مطالب

## 303. چطوری میشه با هوک‌های ری اکت دیتا `fetch` کرد؟

هوک این افکت اسمش `useEffect` هستش و میشه خیلی ساده ازش برای فراخوانی API با استفاده از `axios` استفاده کرد. نتیجه درخواست رو هم خیلی ساده میشه ریخت تو یه `state` داخلی از `component` که وظیفه این ثبت شدن داده رو هم تابع `setter` از `useState` به عهده می‌گیره. خب بزارین یه مثال بزنینم که لیست مقالات رو از یه API می‌گیره:

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function App() {
 const [data, setData] = useState({ hits: [] });

 useEffect(async () => {
 const result = await axios(
 "http://hn.algolia.com/api/v1/search?query=react"
);

 setData(result.data);
 }, []);

 return (

 {data.hits.map((item) => (
 <li key={item.objectID}>
 {item.title}

))}

);
}

export default App;
```

دقت کنید که به آرایه خالی به عنوان پارامتر دوم به هوک effect دادیم که فقط موقع mount شدن درخواست رو بفرسته و لازم نباشه با هر بار رندر درخواست زده بشه، اگر لازم بود با تغییرات یه مقدار (مثلا شناسه مقاله) درخواست API رو مجدداً بزنیم، می‌تونستیم عنوان متغیر رو توی اون آرایه قرار بدیمش و با هر تغییر اون متغیر افکت مجدداً اجرا بشه.

[↑ فهرست مطالب](#)

## 304. هوک‌ها همه موارد کاربرد کلاس‌ها رو پوشش میدن؟

هوک‌ها همیشه گفت همه موارد کارکردی کلاس‌ها رو پوشش نمیدن ولی با اضافه شدن هوک‌های جدید برنامه‌های خوبی برای آینده هوک‌ها پیش‌بینی میشه. در حال حاضر هیچ هوکی وجود نداره که کارکرد متدهای **getSnapshotBeforeUpdate** و **componentDidCatch** رو محقق کنه.

[↑ فهرست مطالب](#)

## 305. نسخه پایدار ری‌اکت که از هوک پشتیبانی می‌کند کدومه؟

ری‌اکت حالت پایداری از هوک‌ها رو توی نسخه 16.8 برای پکیج‌های زیر منتشر کرد:

1. React DOM

2. React DOM Server

3. React Test Renderer

4. React Shallow Renderer

↑ فهرست مطالب

## 306. چرا از حالت destructuring آرایه برای useState استفاده می‌کنیم؟

وقتی که با استفاده از هوک `useState` به `state` رو معرفی می‌کنیم، یه آرایه دوتایی برمی‌گردونه که اندیس اولش متغیر مورد نظر برای دسترسی به `state` هست و اندیس دوم `setter` یا تغییر دهنده اون `state`. یه روش اینه که با استفاده از اندیس‌های آرایه و `[0]` و `[1]` بهشون دسترسی پیدا کنیم ولی یه کم ممکنه گیج کننده باشه. ولی با استفاده از حالت `destructuring` خیلی ساده‌تر میشه این کار رو انجام داد. برای مثال دسترسی به `state` با اندیس‌های آرایه این شکلی میشد:

```
var userStateVariable = useState("userProfile"); // Returns an array
var user = userStateVariable[0]; // Access first item
var setUser = userStateVariable[1]; // Access second item
```

ولی همون کد با استفاده از `destructuring` آرایه‌ها به شکل پایین درمیا:

```
const [user, setUser] = useState("userProfile");
```

↑ فهرست مطالب

## 307. منابعی که باعث معرفی ایده هوک‌ها شدن کدوما بودن؟

ایده معرفی هوک از منابع مختلفی به وجود اومد. این پایین یه لیستی ازشون رو میاریم:

1. تجربه قبلی که با `functional API` توی پکیج `react-future` داشتن

2. تجربه انجمن ری‌اکت با پراپ `render` مثل کامپوننت‌های `React`

3. متغیرهای `state` و سلول‌های `state` توی `DisplayScript`.

- 4. Subscription های موجود توی Rxjs.
- 5. کامپوننت های reducer توی ReasonReact.

↑ فهرست مطالب

## 308. چطوری به API های ضروری اجزای وب دسترسی پیدا کنیم؟

کامپوننت های web اکثرا به عنوان API های imperative برای اجرای یه وظیفه خاص قلمداد می شن. برای استفاده ازشون باید با استفاده از **ref** که امکان کار با DOM را فراهم می کنه بیاییم یه کامپوننت که به شکل imperative کار می کنه ایجاد کنیم. ولی اگه از وب کامپوننت های کاستوم یا همون third-party استفاده می کنیم، بهترین کار نوشتن یه کامپوننت **wrapper** برای استفاده از اون وب کامپوننت هست.

↑ فهرست مطالب

## 309. formik چیه؟

Formik یه کتابخونه ری اکت هست که امکان حل سه مشکل اساسی رو فراهم می کنه:

1. دریافت و مدیریت مقادیر از state
2. اعتبارسنجی و مدیریت خطاها
3. مدیریت ثبت فرم ها

↑ فهرست مطالب

## 310. middleware های مرسوم برای مدیریت ارتباط های asynchronous توی Redux کدوما هستن؟

یه سری از میان افزارهای (middleware) معروف برای مدیریت فراخوانی action هایی که به شکل asynchronous توی Redux فراخوانی میشن اینا هستن: **Redux Thunk**، **Redux** و **Promise** و **Redux Saga**.

↑ فهرست مطالب

## 311. مرورگرها کد JSX رو متوجه میشن؟

نه، مرورگرها نمی‌تونن کد JSX رو متوجه بشن. مجبوریم که از یه transpiler برای تبدیل کد JSX به کد جاوااسکریپت عادی که مرورگرها متوجه میشن تبدیل کنیم. مشهورترین transpiler در حال حاضر Babel هست که برای اینکار استفاده میشه.

↑ فهرست مطالب

## 312. Data flow یا جریان داده ری‌اکت رو توضیح میدی؟

ری‌اکت از روش جریان داده یک طرفه استفاده می‌کنه. استفاده از prop باعث میشه از تکرار موارد بدیهی جلوگیری بشه و درک کردنش ساده‌تر از روش سنتی data-binding دو طرفه باشه.

↑ فهرست مطالب

## 313. react scripts چیه؟

پکیج `react-scripts` یه مجموعه از اسکریپت‌هاست که توی `create-react-app` برای ایجاد سریع و ساده پروژه ری‌اکتی ازشون استفاده میشه. دستور `react-scripts start` محیط توسعه کد رو ایجاد می‌کنه و یه سرور براتون استارت می‌کنه که از لود در لحظه و داغ مازول‌ها پشتیبانی می‌کنه.

↑ فهرست مطالب

## 314. ویژگی‌های create react app چیه؟

این پایین به یه سری از ویژگی‌های `create-react-app` رو لیست می‌کنیم.

1. ساپورت کامل از Flow و React، JSX، ES6، Typescript
2. Autoprefixed CSS
3. CSS Reset/Normalize
4. سرور live development
5. یه اجرا کننده unit-test که ساپورت built-in برای گزارش coverage داره



6. به اسکریپت build برای bundle کردن فایل‌های CSS، JS و تصاویر که برای استفاده production با قابلیت hash و sourcemap عمل می‌کنه
7. به سرویس ورکر برای استفاده به صورت offline-first که قابلیت استفاده به صورت web-app و pwa رو فراهم می‌کنه

↑ فهرست مطالب

## 315. هدف از متد renderToNodeStream چیه؟

متد `ReactDOMServer#renderToNodeStream` برای تولید HTML روی سرور و ارسال اون به درخواست initial کاربر استفاده می‌شه که باعث میشه صفحات سریع‌تر لود بشن. البته علاوه بر سرعت، به موتورهای جستجو این امکان رو میده که وبسایت شما رو به سادگی crawl کنن و SEO سایت بهتر بشه.

**Note:** البته یادتون باشه که این متد توی مرورگر قابل اجرا نیست و فقط روی سرور کار می‌کنه.

↑ فهرست مطالب

## 316. MobX چیه؟

MobX یه راه‌حل ساده، scalable برای مدیریت state هست که خیلی قوی تست شده. این روش برای برنامه‌نویسی تابعی کنش‌گرا (TFRP) استفاده می‌شه. برای برنامه‌های ری‌اکتی لازمه که پکیج‌های زیر رو نصب کنین:

```
npm install mobx --save
npm install mobx-react --save
```

↑ فهرست مطالب

## 317. تفاوت‌های بین Redux و MobX کدوما هستن؟

این پایین به یه سری از اصلی‌ترین تفاوت‌های Redux و MobX اشاره می‌کنیم:

موضوع	Redux	MobX
-------	-------	------

موضوع	Redux	MobX
تعریف	یه کتابخونه جاواسکریپتی هستش که امکان مدیریت state رو فراهم می‌کنه	یه کتابخونه جاواسکریپتی هستش که امکان مدیریت state به صورت کنش‌گرا رو فراهم می‌کنه
برنامه‌نویسی	به صورت پایهای با ES6 نوشته شده	به صورت پایهای با ES5 نوشته بشه
Store دیتا	فقط یه store برای مدیریت همه داده‌ها وجود داره	بیش از یه store برای ذخیره و مدیریت داده وجود داره
کاربرد	به شکل اساسی برای برنامه‌های پیچیده و بزرگ استفاده میشه	برای برنامه‌های ساده بیشتر کاربرد داره
پرفورمنس	نیاز به یه سری بهبودها داره	پرفورمنس بهتری ارائه میده
چگونگی ذخیره داده	از آبجکت جاواسکریپت به عنوان store استفاده می‌کنه	از observable برای نگهداری داده استفاده می‌کنه

↑ فهرست مطالب

## 318. لازمه قبل از شروع ری‌اکت ES6 رو یاد گرفت؟

نه، اجبار برای یادگرفتن es2015/es6 برای کار با ری‌اکت وجود نداره. ولی توصیه شدیدی میشه که یاد بگیریدش چون منابع خیلی زیادی هستن که به شکل پیش‌فرض با es6 کار شدن. بزارین یه نگاه کلی به مواردی که الزاما با es6 کارشون رو ذکر کنیم:

1. Destructuring: برای گرفتن مقادیر prop و استفاده از اونا توی کامپوننت

```
// in es 5
var someData = this.props.someData;
var dispatch = this.props.dispatch;

// in es6
const { someData, dispatch } = this.props;
```

2. عملگر spread: به پاس دادن prop‌ها به پایین برای کامپوننت‌های فرزند کمک می‌کنه

```
// in es 5
<SomeComponent someData={this.props.someData} dispatch={this.props.dispatch} />

// in es6
<SomeComponent {...this.props} />
```

3. توابع arrow: کدها رو کم حجم تر می کنه

```
// es 5
var users = userList.map(function (user) {
 return {user.name};
});

// es 6
const users = userList.map((user) => {user.name});
```

[↑ فهرست مطالب](#)

## 319 Concurrent Rendering چیست؟

Concurrent rendering باعث میشه برنامه ری اکتی بتونه توی رندر کردن درخت کامپوننت ها به شکل مسئولانه تری عمل کنه و انجام این رندر رو بدون بلاک کردن thread اصلی مرورگر انجام بده. این امر به ری اکت این اجازه رو میده که بتونه اجرا شدن یه رندر طولانی رو به بخش های مرتب شده بر اساس اولویت تقسیم کنه و توی پیک های مختلف رندر رو انجام بده. برای مثال وقتی حالت concurrent فعال باشه، ری اکت یه نیم نگاهی هم به بقیه تسک هایی که هنوز انجام نشدن داره و اگه تسک با اولویت دیگه ای رو ببینه، حالت فعلی که داشت رندر می کرد رو متوقف می کنه و به انجام کار با اولویت تر می رسه. این حالت رو به دو روش میشه فعال کرد:

۱. برای یه بخش از برنامه با wrap کردن کامپوننت توی تگ concurrent:

```
<React.unstable_ConcurrentMode>
 <Something />
</React.unstable_ConcurrentMode>
```

۲. برای کل برنامه با استفاده از createRoot موقع رندر

```
ReactDOM.unstable_createRoot(domNode).render(<App />);
```

[↑ فهرست مطالب](#)

## 320. تفاوت بین حالت async و concurrent چیست؟

هر دوتاشون به یه چیز اشاره می‌کنن. قبلا حالت concurrent با عنوان "Async Mode" توسط تیم ری‌اکت معرفی می‌شد. عنوان این قابلیت به این دلیل تغییر پیدا کرد که قابلیت ری‌اکت برای کار روی مرحله‌های با اولویت متفاوت رو نشون بده. همین موضوع جلوی اشتباهات در مورد طرز تفکر راجع به رندر کردن async رو می‌گیره.

↑ فهرست مطالب

## 321. می‌تونیم از آدرس‌های دارای url جاواسکریپت در ری‌اکت 16.9 استفاده کرد؟

آره، میشه از javascript: استفاده کرد ولی یه warning توی کنسول برامون نشون داده میشه. چون آدرس‌هایی که با javascript: شروع میشن خطرناکن و می‌تونن باعث ایجاد باگ امنیتی توی برنامه بشن.

```
const companyProfile = {
 website: "javascript: alert('Your website is hacked')",
};
// It will log a warning
More details;
```

البته بخاطر داشته باشین که نسخه‌های بعدی ری‌اکت قراره بجای warning یه ارور برای این مورد throw کنن.

↑ فهرست مطالب

## 322. هدف از پلاگین eslint برای هوک‌ها چیست؟

پلاگین ESLint میاد یه سری قوانین برای درست نوشت هوک‌ها رو توی برنامه الزامی می‌کنه. روش تشخیص دادن هوک‌ها هم اینطوریه که می‌گه اگه اسم تابعی با "use" شروع بشه و درست بعد اون یه حرف بزرگ بیاد پس اون تابع هوک هستش. این پلاگین در حالت پایه این دوتا شرط رو الزام می‌کنه:

1. فراخوانی هوک‌ها یا باید داخل یه تابع که عنوانش PascalCase هست (منظور یه کامپوننته) یا یه تابع دیگه که مثلا useSomething هست (custom هوک) انجام بشه.

2. هوک‌ها باید توی همه رندرها با یه ترتیب مشخص اجرا بشن.

↑ فهرست مطالب

## 323. تفاوت‌های Declarative و Imperative توی ری‌اکت چیه؟

یه کامپوننت ساده UI رو تصور کنین، مثلاً یه دکمه "لایک". وقتی که روش کلیک می‌کنین رنگ از خاکستری به آبی تغییر پیدا می‌کنه و اگه دوباره کلیک کنید باز خاکستری میشه. رویش imperative برای انجام این کار اینطوریه:

```
if (user.likes()) {
 if (hasBlue()) {
 removeBlue();
 addGrey();
 } else {
 removeGrey();
 addBlue();
 }
}
```

لازمه اول بررسی کنیم که چه چیزی رو توی اسکرین داریم نمایش می‌دیم و بعدش ببایم state رو عوض کنیم به حالتی که می‌خواهیم برامون نمایش انجام بشه، توی برنامه‌های بزرگ و واقعی مدیریت این حالت‌ها خیلی می‌تونه سخت باشه. در حالت مقابل، روش declarative می‌تونه اینطوری باشه:

```
if (this.state.liked) {
 return <blueLike />;
} else {
 return <greyLike />;
}
```

چون روش declarative حالت‌ها رو جدا در نظر می‌گیره، این بخش از کد براساس state فقط تصمیم می‌گیره که چه ظاهری رو نمایش بده و به همین دلیل درک کردنش ساده‌تره.

↑ فهرست مطالب

## 324. مزایای استفاده از تایپ اسکریپت با ری‌اکت چیه؟

یه سری از مزایای استفاده از typescript با Reactjs اینا هستن:

1. می‌تونیم از آخرین ویژگی‌های جاوااسکریپت استفاده کنیم
2. از interfaceها برای تعریف نوع‌های دلخواه و پیچیده استفاده کنیم
3. IDEهایی مثل VS Code برای TypeScript ساخته شدن
4. با افزایش خوانایی و Validation از خطاهای ناخواسته جلوگیری کنیم

**↑ فهرست مطالب**