שאלות ידע:

מערך מול רשימה מקושרת:

מערך: נמצא בזכרון כבלוק כך שהאלמנטים ממוקמים באופן רציף

רשימה: האיברים מפוזרים בזכרון עם רפרנס לאיבר הבא

הכנסה למערך: אם זה בסוך אז מהיר, אם לא אז יכול להיות o(n)

רשימה:

In summary, arrays are better suited for scenarios where random access and memory locality are important, and the size of the collection is known in advance or doesn't change frequently. On the other hand, linked lists are more appropriate when frequent insertions and deletions are required, or when the size of the collection can change dynamically.

מערך: כאשר ידוע מראש הגודל וגודל המערך לא משתנה כל הזמן

רשימה: כאשר משתנה כל הזצן הגודל ויש הכנסות והוצאות כל הזמן

שפות תכנות:

C: תכנות פרוצדורלי: תבנית תכנות או גישה עצובית של כתיבת הקוד. למשל ב C, מחולק לאוסף של פונקציו תהקוראות אחת לשניה. ניתן ב C לתחזק את הזכרון עם מצביעים. משומש בעיקר בתוכנות שעוסקות בעיקר באינטראקציה עם החומרה ומערכת ההפעלה של המחשב

C++, JAVA: כאן הגישה היא תכנות מונחה עצמים שבין העצמים קיימת הירריכה ויחסים מסויימים.

פייתון: כאן יש מפענח, כלומר הקוד לא מקומפל לפני הריצה. כאן יש טיפוסים דינאמיים, מוגדרים בזמן ריצה. משתמשת לscientific computing, data analysis, machine learning web. יש לה ספריות רבות, סינטקס קל, קהילה גדולה ודוקומנטציה.

JavaScript:גם כאן שפה עם מפענח, לרוב משומשת בצד לקוח. even loop:

ה-stack:

כאן נשמרות הקריאות לפונקציות. אם יש קריאה לפונקציה פו, היא מתווספת לסטאק, ואם היא קראה לפונקציה באז, אז באז מתווספת לסטאק מעליה, ועד שבאז לא מסתיימת, ג'אווהסקריפט לא חוזרת לפו.

Event loop:

רכיב שתפקידו לבדוק אם הסטאק ריק, ואם הוא ריק לקחת משימה מה-

callback queue

ולהעביר אותה לסטאק.

Web API:

זה ממשק חיצוני, שקשור בדפדפן, ואחראי על הרצה של משימות אסינכרוניות (כמו קריאות לשרת).

כמובן שגם בנוד (וגם בסביבות אחרות), יש מקבילה לזה, שאחראית על הרצת משימות אסינכרוניות.

כשה-runtime מריץ פונקציה "רגילה" של קוד (כלומר סינכרונית) - היא מתווספת ל-call stack, ואז מתבצעת, ומוצאת משם.

כשהוא מריץ פקודה א-סינכרונית, הפקודה היא למעשה קריאה ל-web api (ולכן הת'רד יכול להמשיך בענייניו ולהריץ את פקודות מהסטאק).

ה-runtime קורא לפקודה מה-web api, וממשיך להריץ פקודות מהסטאק.

כשה-web api מסיים את הפקודה (נגיד הנתונים חזרו מהשרת), הוא מוסיף את הקוד שצריך הרצה ל- callback queue.

ה-event loop מחכה שהסטאק יהיה ריק, ואז לוקח את המשימה הבאה מה-callback queue, ומעביר אותה לסטאק, להרצה.

כך שהטריק הוא שקוד א-סינכורני לא באמת רץ בת'רד של ג'אווהסקריפט, וג'אווהסקריפט רק מתקשרת עם התוצאות של ההרצות האלה (כשאין לה משהו אחר לעשות).

https://www.youtube.com/watch?v=lqLSNG\_79lI&ab\_channel=JamesQQuick

event loop:

callStack: מה שנכנס ישר מבוצע אם לא א-סינכרוני.

webAPI: מגיע אליו דרך call stack, הוא שולח את כל א-סינכרוני

task queueץ כאן נשלח משימות אסינכרוניות בעדיםות גבוהה.

evebt queue: לכאן נכסו המשימות אחרי שסיימו את מה שביצעו ב webAPI.

event loop. יבדוק גאשר callstack ריק יבדוק את event queue (קודם יבדוק את טסק קיו) וזש ישלח משם ל call stack.

ASYNC/AWAIT

כאן נוצר המתנה בתוך הפונקציה המוגדרת א-סינכרונית. אבל מחוץ לפונקציה לא תהיה המתנה

יש לו עדיפות על טאסק קיו ולכן יתבצע קודם, כלומק משימותשהן פרומיס יתבצעו לפני סט טיימאוטMicrotask Queue.

hoist: כאשר משתנים ופונקציות עוברות לראש הסקופ: 1. משתנים: הבדלים בין var, let const: const ו let עולים למעלה אבל לא מאותחלים עם undefined. ואילו var כן. לכן נקבל שגיאה כאשר נפנה לאיבר לפני שאותחלץ functions: גם הן מגיעות לראש הסקופ אבל לא פונקציות חץ.

This: הבדלים בין פונקציה רגילה פונקציית חץ:

פונקציה רגילה: אם הפונקציה היא מתודה בתוך אובייקט, אז ה this הוא האובייקט עצמו.

אם לא בתוך אובייקט אז ה this הוא global

Var: הוא מוגבל לסקופ של הפונקציה או לגלובלי, אבל לא לבלוק סקופ, כלומר for או if

Array.sort() הוא עושה מיון לפי ערך הסטרינג של האלמנטץ כלומר 2 יבוא אחרי 1 או 1000

**Shallow Copy**:: לעשות העתקה רדודה של אובייקט, כלומר אם באובייקט עצמו קיים אובייקט אז העתקה רדודה תגרום לאובייקט החדש להצביע על אותו אובייקט מכונן שנמצא אצלו ובכך לגרום לאובייקט הישן להיות חשוף לשינויים.

new.target: נותן לנו את המידע אם בנאי או פונקציה נקראו עםnew , אם כן שומש אז יחזיר את השם של הclass או את שם הפונקציה:

דוגמא:

class BaseClass {

  constructor() {

    console.log(new.target)//BaseClass

  }

}

let baseObj = new BaseClass();

function new and not new: כאשר משתמשים ב new אז מה שיחזור אלינו הוא אובייקט שמייצג את ה this שבתוך הפונקציה. למשל:

function f1() {

    this.y = 21;

    let x = 10;

    console.log("hi");

    return x;

  }

let noNewFunc2 = new f1();

console.log(noNewFunc2, noNewFunc2.y);//f1 { y: 21 }

בלי new פשוט נקבל את ה x שחוזר אלינו מהפונקציה. הthis בתוך הפונקציה יהיה הגלובלי

Class: כאשר עושים new אז נוצר מופע של המחלקה וה this יהיה קשור למחלקה שנוצרה.

כאשר אין new נקבל שגיאה

לסיכום: הבדלים עיקריים: כאשר משתמשים ב new זה פועל כמו בנאי ויוצר מופע של הפונקציה או המחלקה. בנוסף שימוש בthis הוא שונה כאשר משתמשים ב new

React:

ספריית פרונט-אנד של ג'אווהסקריפט.

אלמנטים מרכזיים:

1. JSX, סינטקס שנותן אפשרות לשלב בין קוד ל HTML.
2. Virtual DOM: נדבר קודם על DOM זה המסמל את הדף שאנחנו רואים בפרונטאנד. סמסמך נראה את כל האלמנטים ואת ההיררכייה בינהם.
3. VIRTUAL DOM: הוא העתק של הDOM, הוא משפר את הביצועיים של שינויים שקורים בדף WEB. למה לעדכן את הDOOM זה איטי? כי ביחד איתו נרנדר גם את כל הCSS וכל שאר הקבצים.

בכל פעם שה-state של האפליקציה שלנו משתנה ריאקט יוצרת את כל ה- DOM הווירטואלי כולו מאפס. יצירת עץ אובייקטים שלם היא מהירה מאוד ולכן זה לא משפיע על הביצועים שלנו. בעצם בכל זמן נתון ריאקט מחזיקה שני Virtual DOM, אחד עם המצב המעודכן והשני עם המצב הקודם.

לאחר שהיא מעדכנת את הקומפוננטות הדרושות ב- Virtual DOM. ריאקט משווה בין שני ה-virtual DOM שהיא מחזיקה באמצעות אלגוריתם (diff algorithm) כדי למצוא את מספר הצעדים המינימלי לעדכון ה-Real DOM ולאחר מכן מעדכנת ב- Real DOM רק את השינויים הנדרשים ולא את כל ה DOM

1. שימוש חוזר בקומפוננטות

קומפוננטה פונקציה ומחלקה:

ממומלץ להשתמש בפונקציות ולא קלאס. יותר מהיר וקצר.

לפני שהכניסו hooks. הקלאס יהיה עדיף כי רק שם אפשר היה להשתמש בstate משלה.

הוקס מאפשרים לנו גם בפונקציות לנההל state

State: מידע שקיים בקומפוננטה שיכול להשתנות בזמן שהקומפוננטה עדיין בחיים. ברגע שהמידע משתנה הקומפוננטה תרנדר את עצמה. כדי לעדכן state, לא נעשה ישירות כי כך הקומפוננטה לא תרנדר את עצמה.

Props: אינפוט שיש לנו להעביר לקומפוננטה